

# Doom source code

Id Software

June 5, 2006

## Contents

<b>1</b>	<b>Common code</b>	<b>4</b>
1.1	doomdata.h . . . . .	4
1.2	doomdef.c . . . . .	7
1.3	doomdef.h . . . . .	8
1.4	doomstat.c . . . . .	13
1.5	doomstat.h . . . . .	14
1.6	doomtype.h . . . . .	18
1.7	dstrings.c . . . . .	19
1.8	dstrings.h . . . . .	21
<b>2</b>	<b>Automap</b>	<b>22</b>
2.1	am_map.c . . . . .	22
2.2	am_map.h . . . . .	43
<b>3</b>	<b>Initialisation/general code</b>	<b>43</b>
3.1	d_english.h . . . . .	43
3.2	d_event.h . . . . .	54
3.3	d_french.h . . . . .	56
3.4	d_items.c . . . . .	63
3.5	d_items.h . . . . .	65
3.6	d_main.c . . . . .	66
3.7	d_main.h . . . . .	84
3.8	d_net.c . . . . .	85
3.9	d_net.h . . . . .	97
3.10	d_player.h . . . . .	99
3.11	d_textur.h . . . . .	103
3.12	d_think.h . . . . .	104
3.13	d_ticcmd.h . . . . .	105
<b>4</b>	<b>Finale code</b>	<b>106</b>
4.1	f_finale.c . . . . .	106
4.2	f_finale.h . . . . .	117
4.3	f_wipe.c . . . . .	118
4.4	f_wipe.h . . . . .	123
<b>5</b>	<b>Main game loop</b>	<b>124</b>
5.1	g_game.c . . . . .	124
5.2	g_game.h . . . . .	150
<b>6</b>	<b>Heads-up display</b>	<b>151</b>
6.1	hu_lib.c . . . . .	151
6.2	hu_lib.h . . . . .	157
6.3	hu_stuff.c . . . . .	160
6.4	hu_stuff.h . . . . .	172

<b>7</b>	<b>System-specific code</b>	<b>173</b>
7.1	i_main.c . . . . .	173
7.2	i_net.c . . . . .	174
7.3	i_net.h . . . . .	179
7.4	i_sound.c . . . . .	180
7.5	i_sound.h . . . . .	195
7.6	i_system.c . . . . .	197
7.7	i_system.h . . . . .	200
7.8	i_video.c . . . . .	201
7.9	i_video.h . . . . .	217
<b>8</b>	<b>Miscellaneous</b>	<b>219</b>
8.1	m_argv.c . . . . .	219
8.2	m_argv.h . . . . .	219
8.3	m_bbox.c . . . . .	220
8.4	m_bbox.h . . . . .	221
8.5	m_cheat.c . . . . .	222
8.6	m_cheat.h . . . . .	224
8.7	m_fixed.c . . . . .	225
8.8	m_fixed.h . . . . .	226
8.9	m_menu.c . . . . .	227
8.10	m_menu.h . . . . .	256
8.11	m_misc.c . . . . .	257
8.12	m_misc.h . . . . .	266
8.13	m_random.c . . . . .	267
8.14	m_random.h . . . . .	268
8.15	m_swap.c . . . . .	269
8.16	m_swap.h . . . . .	270
<b>9</b>	<b>Game logic/behaviour</b>	<b>270</b>
9.1	info.c . . . . .	270
9.2	info.h . . . . .	342
9.3	p_ceilng.c . . . . .	363
9.4	p_doors.c . . . . .	368
9.5	p_enemy.c . . . . .	380
9.6	p_floor.c . . . . .	411
9.7	p_inter.c . . . . .	420
9.8	p_inter.h . . . . .	434
9.9	p_lights.c . . . . .	434
9.10	p_local.h . . . . .	440
9.11	p_map.c . . . . .	445
9.12	p_maputl.c . . . . .	465
9.13	p_mobj.c . . . . .	479
9.14	p_mobj.h . . . . .	494
9.15	p_plats.c . . . . .	499
9.16	p_pspr.c . . . . .	504
9.17	p_pspr.h . . . . .	517
9.18	p_saveg.c . . . . .	518
9.19	p_saveg.h . . . . .	528
9.20	p_setup.c . . . . .	528
9.21	p_setup.h . . . . .	539
9.22	p_sight.c . . . . .	540
9.23	p_spec.c . . . . .	546
9.24	p_spec.h . . . . .	567
9.25	p_switch.c . . . . .	577
9.26	p_telept.c . . . . .	587
9.27	p_tick.c . . . . .	589
9.28	p_tick.h . . . . .	591
9.29	p_user.c . . . . .	592

<b>10 Rendering engine</b>	<b>598</b>
10.1 r_bsp.c . . . . .	598
10.2 r_bsp.h . . . . .	607
10.3 r_data.c . . . . .	608
10.4 r_data.h . . . . .	621
10.5 r_defs.h . . . . .	622
10.6 r_draw.c . . . . .	630
10.7 r_draw.h . . . . .	644
10.8 r_local.h . . . . .	645
10.9 r_main.c . . . . .	646
10.10 r_main.h . . . . .	660
10.11 r_plane.c . . . . .	663
10.12 r_plane.h . . . . .	670
10.13 r_segs.c . . . . .	671
10.14 r_segs.h . . . . .	683
10.15 r_sky.c . . . . .	684
10.16 r_sky.h . . . . .	685
10.17 r_state.h . . . . .	685
10.18 r_things.c . . . . .	688
10.19 r_things.h . . . . .	703
<b>11 Sound code</b>	<b>704</b>
11.1 s_sound.c . . . . .	704
11.2 s_sound.h . . . . .	718
11.3 sounds.c . . . . .	719
11.4 sounds.h . . . . .	723
<b>12 Status bar</b>	<b>728</b>
12.1 st_lib.c . . . . .	728
12.2 st_lib.h . . . . .	732
12.3 st_stuff.c . . . . .	736
12.4 st_stuff.h . . . . .	758
<b>13 General graphic drawing</b>	<b>760</b>
13.1 v_video.c . . . . .	760
13.2 v_video.h . . . . .	767
<b>14 WAD file loading</b>	<b>769</b>
14.1 w_wad.c . . . . .	769
14.2 w_wad.h . . . . .	778
<b>15 Intermission screen</b>	<b>780</b>
15.1 wi_stuff.c . . . . .	780
15.2 wi_stuff.h . . . . .	808
<b>16 Zone memory allocation system</b>	<b>809</b>
16.1 z_zone.c . . . . .	809
16.2 z_zone.h . . . . .	816

# 1 Common code

## 1.1 doomdata.h

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
// all external data is defined here
// most of the data is loaded into different structures at run time
// some internal structures shared by many modules are here
//
//-----

#ifndef __DOOMDATA__
#define __DOOMDATA__

// The most basic types we use, portability.
#include "doomtype.h"

// Some global defines, that configure the game.
#include "doomdef.h"

//
// Map level types.
// The following data structures define the persistent format
// used in the lumps of the WAD files.
//
//
// Lump order in a map WAD: each map needs a couple of lumps
// to provide a complete scene geometry description.
enum
{
    ML_LABEL,                // A separator, name, ExMx or MAPxx
    ML_THINGS,               // Monsters, items..
    ML_LINEDEFS,             // LineDefs, from editing
    ML_SIDEDEFS,             // SideDefs, from editing
    ML_VERTEXES,            // Vertices, edited and BSP splits generated
    ML_SEGS,                 // LineSegs, from LineDefs split by BSP
    ML_SSECTORS,             // SubSectors, list of LineSegs
    ML_NODES,                // BSP nodes
    ML_SECTORS,              // Sectors, from editing
    ML_REJECT,               // LUT, sector-sector visibility
    ML_BLOCKMAP              // LUT, motion clipping, walls/grid element
};

// A single Vertex.
typedef struct
{
```

```

    short            x;
    short            y;
} mapvertex_t;

// A SideDef, defining the visual appearance of a wall,
// by setting textures and offsets.
typedef struct
{
    short            textureoffset;
    short            rowoffset;
    char             toptexture[8];
    char             bottomtexture[8];
    char             midtexture[8];
    // Front sector, towards viewer.
    short            sector;
} mapsidedef_t;

// A LineDef, as used for editing, and as input
// to the BSP builder.
typedef struct
{
    short            v1;
    short            v2;
    short            flags;
    short            special;
    short            tag;
    // sidenum[1] will be -1 if one sided
    short            sidenum[2];
} maplinedef_t;

//
// LineDef attributes.
//

// Solid, is an obstacle.
#define ML_BLOCKING            1

// Blocks monsters only.
#define ML_BLOCKMONSTERS      2

// Backside will not be present at all
// if not two sided.
#define ML_TWOSIDED            4

// If a texture is pegged, the texture will have
// the end exposed to air held constant at the
// top or bottom of the texture (stairs or pulled
// down things) and will move with a height change
// of one of the neighbor sectors.
// Unpegged textures allways have the first row of
// the texture at the top pixel of the line for both
// top and bottom textures (use next to windows).

// upper texture unpegged
#define ML_DONTPEGETOP        8

// lower texture unpegged
#define ML_DONTPEGBOTTOM      16

// In AutoMap: don't map as two sided: IT'S A SECRET!
#define ML_SECRET            32

```

```

// Sound rendering: don't let sound cross two of these.
#define ML_SOUNDBLOCK          64

// Don't draw on the automap at all.
#define ML_DONTDRAW            128

// Set if already seen, thus drawn in automap.
#define ML_MAPPED              256


// Sector definition, from editing.
typedef struct
{
    short          floorheight;
    short          ceilingheight;
    char           floorpic[8];
    char           ceilingpic[8];
    short          lightlevel;
    short          special;
    short          tag;
} mapsector_t;

// SubSector, as generated by BSP.
typedef struct
{
    short          numsegs;
    // Index of first one, segs are stored sequentially.
    short          firstseg;
} mapsubsector_t;

// LineSeg, generated by splitting LineDefs
// using partition lines selected by BSP builder.
typedef struct
{
    short          v1;
    short          v2;
    short          angle;
    short          linedef;
    short          side;
    short          offset;
} mapseg_t;


// BSP node structure.

// Indicate a leaf.
#define NF_SUBSECTOR          0x8000

typedef struct
{
    // Partition line from (x,y) to x+dx,y+dy)
    short          x;
    short          y;
    short          dx;
    short          dy;

    // Bounding box for each child,
    // clip against view frustum.
    short          bbox[2][4];

```

```

// If NF_SUBSECTOR its a subsector,
// else it's a node of another subtree.
unsigned short      children[2];

} mapnode_t;


// Thing definition, position, orientation and type,
// plus skill/visibility flags and attributes.
typedef struct
{
    short            x;
    short            y;
    short            angle;
    short            type;
    short            options;
} mapthing_t;


#endif                          // __DOOMDATA__
//-----
//
// $Log:$
//
//-----

```

## 1.2 doomdef.c

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//   DoomDef - basic defines for DOOM, e.g. Version, game mode
//   and skill level, and display parameters.
//
//-----

static const char
rcsid[] = "$Id: m_bbox.c,v 1.1 1997/02/03 22:45:10 b1 Exp $";


#ifdef __GNUG__
#pragma implementation "doomdef.h"

```

```
#endif
#include "doomdef.h"

// Location for any defines turned variables.

// None.
```

### 1.3 doomdef.h

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
// Internally used data structures for virtually everything,
// key definitions, lots of other stuff.
//
//-----

#ifndef __DOOMDEF__
#define __DOOMDEF__

#include <stdio.h>
#include <string.h>

//
// Global parameters/defines.
//
// DOOM version
enum { VERSION = 110 };

// Game mode handling - identify IWAD version
// to handle IWAD dependend animations etc.
typedef enum
{
    shareware,          // DOOM 1 shareware, E1, M9
    registered,         // DOOM 1 registered, E3, M27
    commercial,         // DOOM 2 retail, E1 M34
    // DOOM 2 german edition not handled
    retail,             // DOOM 1 retail, E4, M36
    indetermined        // Well, no IWAD found.
} GameMode_t;

// Mission packs - might be useful for TC stuff?
typedef enum
{
    doom,               // DOOM 1
```



```

doom2,          // DOOM 2
pack_tnt,       // TNT mission pack
pack_plut,      // Plutonia pack
none

} GameMission_t;

// Identify language to use, software localization.
typedef enum
{
    english,
    french,
    german,
    unknown
} Language_t;

// If rangecheck is undefined,
// most parameter validation debugging code will not be compiled
#define RANGECHECK

// Do or do not use external soundserver.
// The sndserver binary to be run separately
// has been introduced by Dave Taylor.
// The integrated sound support is experimental,
// and unfinished. Default is synchronous.
// Experimental asynchronous timer based is
// handled by SNDINTR.
#define SNDSERV 1
// #define SNDINTR 1

// This one switches between MIT SHM (no proper mouse)
// and XFree86 DGA (mickey sampling). The original
// linuxdoom used SHM, which is default.
// #define X11_DGA 1

//
// For resize of screen, at start of game.
// It will not work dynamically, see visplanes.
//
#define BASE_WIDTH 320

// It is educational but futile to change this
// scaling e.g. to 2. Drawing of status bar,
// menus etc. is tied to the scale implied
// by the graphics.
#define SCREEN_MUL 1
#define INV_ASPECT_RATIO 0.625 // 0.75, ideally

// Defines suck. C sucks.
// C++ might suck for OOP, but it sure is a better C.
// So there.
#define SCREENWIDTH 320
// SCREEN_MUL*BASE_WIDTH //320
#define SCREENHEIGHT 200
// (int)(SCREEN_MUL*BASE_WIDTH*INV_ASPECT_RATIO) //200

// The maximum number of players, multiplayer/networking.

```

```

#define MAXPLAYERS                4

// State updates, number of tics / second.
#define TICRATE                    35

// The current state of the game: whether we are
// playing, gazing at the intermission screen,
// the game final animation, or a demo.
typedef enum
{
    GS_LEVEL,
    GS_INTERMISSION,
    GS_FINALE,
    GS_DEMOSCREEN
} gamestate_t;

//
// Difficulty/skill settings/filters.
//

// Skill flags.
#define MTF_EASY                    1
#define MTF_NORMAL                  2
#define MTF_HARD                    4

// Deaf monsters/do not react to sound.
#define MTF_AMBUSH                  8

typedef enum
{
    sk_baby,
    sk_easy,
    sk_medium,
    sk_hard,
    sk_nightmare
} skill_t;

//
// Key cards.
//
typedef enum
{
    it_bluecard,
    it_yellowcard,
    it_redcard,
    it_blueskull,
    it_yellowskull,
    it_redskull,

    NUMCARDS
} card_t;

// The defined weapons,
// including a marker indicating
// user has not changed weapon.
typedef enum
{
    wp_fist,
    wp_pistol,

```

```

wp_shotgun,
wp_chaingun,
wp_missile,
wp_plasma,
wp_bfg,
wp_chainsaw,
wp_supershotgun,

NUMWEAPONS,

// No pending weapon change.
wp_nochange

} weapontype_t;

// Ammunition types defined.
typedef enum
{
    am_clip,           // Pistol / chaingun ammo.
    am_shell,          // Shotgun / double barreled shotgun.
    am_cell,           // Plasma rifle, BFG.
    am_misl,           // Missile launcher.
    NUMAMMO,
    am_noammo          // Unlimited for chainsaw / fist.
} ammotype_t;

// Power up artifacts.
typedef enum
{
    pw_invulnerability,
    pw_strength,
    pw_invisibility,
    pw_ironfeet,
    pw_allmap,
    pw_infrared,
    NUMPOWERS
} powertype_t;

//
// Power up durations,
// how many seconds till expiration,
// assuming TICRATE is 35 ticks/second.
//
typedef enum
{
    INVULNTICS          = (30*TICRATE),
    INVISTICS           = (60*TICRATE),
    INFRATICS           = (120*TICRATE),
    IRONTICS            = (60*TICRATE)
} powerduration_t;

//
// DOOM keyboard definition.
// This is the stuff configured by Setup.Exe.
// Most key data are simple ascii (uppercased).

```

```

//
#define KEY_RIGHTARROW      0xae
#define KEY_LEFTARROW      0xac
#define KEY_UPARROW        0xad
#define KEY_DOWNARROW      0xaf
#define KEY_ESCAPE          27
#define KEY_ENTER           13
#define KEY_TAB             9
#define KEY_F1              (0x80+0x3b)
#define KEY_F2              (0x80+0x3c)
#define KEY_F3              (0x80+0x3d)
#define KEY_F4              (0x80+0x3e)
#define KEY_F5              (0x80+0x3f)
#define KEY_F6              (0x80+0x40)
#define KEY_F7              (0x80+0x41)
#define KEY_F8              (0x80+0x42)
#define KEY_F9              (0x80+0x43)
#define KEY_F10             (0x80+0x44)
#define KEY_F11             (0x80+0x57)
#define KEY_F12             (0x80+0x58)

#define KEY_BACKSPACE       127
#define KEY_PAUSE           0xff

#define KEY_EQUALS          0x3d
#define KEY_MINUS           0x2d

#define KEY_RSHIFT          (0x80+0x36)
#define KEY_RCTRL           (0x80+0x1d)
#define KEY_RALT            (0x80+0x38)

#define KEY_LALT            KEY_RALT

// DOOM basic types (boolean),
// and max/min values.
// #include "doomtype.h"

// Fixed point.
// #include "m_fixed.h"

// Endianess handling.
// #include "m_swap.h"

// Binary Angles, sine/cosine/atan lookups.
// #include "tables.h"

// Event type.
// #include "d_event.h"

// Game function, skills.
// #include "g_game.h"

// All external data is defined here.
// #include "doomdata.h"

// All important printed strings.
// Language selection (message strings).
// #include "dstrings.h"

// Player is a special actor.
// struct player_s;

```

```

#include "d_items.h"
#include "d_player.h"
#include "p_mobj.h"
#include "d_net.h"

// PLAY
#include "p_tick.h"

// Header, generated by sound utility.
// The utility was written by Dave Taylor.
#include "sounds.h"

#endif          // __DOOMDEF__
//-----
//
// $Log:$
//
//-----

1.4  doomstat.c

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Put all global tate variables here.
//
//-----

static const char
rcsid[] = "$Id: m_bbox.c,v 1.1 1997/02/03 22:45:10 b1 Exp $";

#ifdef __GNUG__
#pragma implementation "doomstat.h"
#endif
#include "doomstat.h"

// Game Mode - identify IWAD as shareware, retail etc.
GameMode_t gamemode = indetermined;
GameMission_t          gamemission = doom;

```

```
// Language.
Language_t    language = english;

// Set if homebrew PWAD stuff has been added.
boolean       modifiedgame;
```

## 1.5 doomstat.h

```
// Emacs style mode select    -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//   All the global variables that store the internal state.
//   Theoretically speaking, the internal state of the engine
//   should be found by looking at the variables collected
//   here, and every relevant module will have to include
//   this header file.
//   In practice, things are a bit messy.
//
//-----

#ifndef __D_STATE__
#define __D_STATE__

// We need globally shared data structures,
// for defining the global state variables.
#include "doomdata.h"
#include "d_net.h"

// We need the playr data structure as well.
#include "d_player.h"

#ifdef __GNUG__
#pragma interface
#endif

// -----
// Command line parameters.
//
extern boolean    nomonsters;        // checkparm of -nomonsters
extern boolean    respawnparm;      // checkparm of -respawn
extern boolean    fastparm;         // checkparm of -fast
```

```

extern boolean      devparm;          // DEBUG: launched with -devparm

// -----
// Game Mode - identify IWAD as shareware, retail etc.
//
extern GameMode_t    gamemode;
extern GameMission_t gamemission;

// Set if homebrew PWAD stuff has been added.
extern boolean      modifiedgame;

// -----
// Language.
extern Language_t    language;

// -----
// Selected skill type, map etc.
//

// Defaults for menu, methinks.
extern skill_t       startskill;
extern int           startepisode;
extern int           startmap;

extern boolean       autostart;

// Selected by user.
extern skill_t       gameskill;
extern int           gameepisode;
extern int           gamemap;

// Nightmare mode flag, single player.
extern boolean       respawnmonsters;

// Netgame? Only true if >1 player.
extern boolean       netgame;

// Flag: true only if started as net deathmatch.
// An enum might handle altdeath/cooperative better.
extern boolean       deathmatch;

// -----
// Internal parameters for sound rendering.
// These have been taken from the DOS version,
// but are not (yet) supported with Linux
// (e.g. no sound volume adjustment with menu.

// These are not used, but should be (menu).
// From m_menu.c:
// Sound FX volume has default, 0 - 15
// Music volume has default, 0 - 15
// These are multiplied by 8.
extern int snd_SfxVolume;    // maximum volume for sound
extern int snd_MusicVolume;  // maximum volume for music

// Current music/sfx card - index useless
// w/o a reference LUT in a sound module.
// Ideally, this would use indices found
// in: /usr/include/linux/soundcard.h
extern int snd_MusicDevice;

```

```

extern int snd_SfxDevice;
// Config file? Same disclaimer as above.
extern int snd_DesiredMusicDevice;
extern int snd_DesiredSfxDevice;

// -----
// Status flags for refresh.
//

// Depending on view size - no status bar?
// Note that there is no way to disable the
// status bar explicitly.
extern boolean statusbaractive;

extern boolean automapactive;          // In AutoMap mode?
extern boolean menuactive;            // Menu overlayed?
extern boolean paused;                // Game Pause?

extern boolean viewactive;

extern boolean nodrawers;
extern boolean noblit;

extern int viewwindowx;
extern int viewwindowy;
extern int viewheight;
extern int viewwidth;
extern int scaledviewwidth;

// This one is related to the 3-screen display mode.
// ANG90 = left side, ANG270 = right
extern int viewangleoffset;

// Player taking events, and displaying.
extern int consoleplayer;
extern int displayplayer;

// -----
// Scores, rating.
// Statistics on a given map, for intermission.
//
extern int totalkills;
extern int totalitems;
extern int totalsecret;

// Timer, for scores.
extern int levelstarttic;             // gametic at level start
extern int leveltime;                // tics in game play for par

// -----
// DEMO playback/recording related stuff.
// No demo, there is a human player in charge?
// Disable save/end game?
extern boolean usergame;

```



```

//?
extern boolean      demoplayback;
extern boolean      demorecording;

// Quit after playing a demo from cmdline.
extern boolean      singledemo;


//?
extern gamestate_t  gamestate;


//-----
// Internal parameters, fixed.
// These are set by the engine, and not changed
// according to user inputs. Partly load from
// WAD, partly set at startup time.


extern      int      gametic;


// Bookkeeping on players - state.
extern      player_t  players[MAXPLAYERS];

// Alive? Disconnected?
extern boolean      playeringame[MAXPLAYERS];


// Player spawn spots for deathmatch.
#define MAX_DM_STARTS 10
extern mapthing_t    deathmatchstarts[MAX_DM_STARTS];
extern mapthing_t*    deathmatch_p;


// Player spawn spots.
extern mapthing_t     playerstarts[MAXPLAYERS];


// Intermission stats.
// Parameters for world map / intermission.
extern wbstartstruct_t wminfo;


// LUT of ammunition limits for each kind.
// This doubles with BackPack powerup item.
extern int             maxammo[NUMAMMO];


//-----
// Internal parameters, used for engine.
//

// File handling stuff.
extern      char      basedefault[1024];
extern FILE* debugfile;

```

```

// if true, load all graphics at level load
extern boolean      precache;

// wipegamestate can be set to -1
// to force a wipe on the next draw
extern gamestate_t  wipegamestate;

extern int          mouseSensitivity;
//?
// debug flag to cancel adaptiveness
extern boolean      singletics;

extern int          bodyqueslot;

// Needed to store the number of the dummy sky flat.
// Used for rendering,
// as well as tracking projectiles etc.
extern int          skyflatnum;

// Netgame stuff (buffers and pointers, i.e. indices).

// This is ???
extern doomcom_t*    doomcom;

// This points inside doomcom.
extern doomdata_t*   netbuffer;

extern ticcmd_t      localcmds[BACKUPTICS];
extern int           rndindex;

extern int           maketic;
extern int           nettics[MAXNETNODES];

extern ticcmd_t      netcmds[MAXPLAYERS][BACKUPTICS];
extern int           ticdup;

#endif
//-----
//
// $Log:$
//
//-----

```

## 1.6 doomtype.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//

```

```

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Simple basic typedefs, isolated here to make it easier
//     separating modules.
//
//-----

#ifndef __DOOMTYPE__
#define __DOOMTYPE__

#ifndef __BYTEBOOL__
#define __BYTEBOOL__
// Fixed to use builtin bool type with C++.
#ifdef __cplusplus
typedef bool boolean;
#else
typedef enum {false, true} boolean;
#endif
typedef unsigned char byte;
#endif

// Predefined with some OS.
#ifdef LINUX
#include <values.h>
#else
#define MAXCHAR          ((char)0x7f)
#define MAXSHORT         ((short)0x7fff)

// Max pos 32-bit int.
#define MAXINT            ((int)0x7fffffff)
#define MAXLONG           ((long)0x7fffffff)
#define MINCHAR           ((char)0x80)
#define MINSHORT          ((short)0x8000)

// Max negative 32-bit integer.
#define MININT            ((int)0x80000000)
#define MINLONG           ((long)0x80000000)
#endif

#endif
//-----
//
// $Log:$
//
//-----

```

## 1.7 dstrings.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//

```

```

// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Globally defined strings.
//
//-----

static const char
rcsid[] = "$Id: m_bbox.c,v 1.1 1997/02/03 22:45:10 b1 Exp $";

#ifdef __GNUG__
#pragma implementation "dstrings.h"
#endif
#include "dstrings.h"

char* endmsg[NUM_QUITMESSAGES+1]=
{
    // DOOM1
    QUITMSG,
    "please don't leave, there's more\ndemons to toast!",
    "let's beat it -- this is turning\ninto a bloodbath!",
    "i wouldn't leave if i were you.\ndos is much worse.",
    "you're trying to say you like dos\nbetter than me, right?",
    "don't leave yet -- there's a\ndemon around that corner!",
    "ya know, next time you come in here\ni'm gonna toast ya.",
    "go ahead and leave. see if i care."

    // QuitDOOM II messages
    "you want to quit?\nthen, thou hast lost an eighth!",
    "don't go now, there's a \ndimensional shambler waiting\nat the dos prompt!",
    "get outta here and go back\nto your boring programs.",
    "if i were your boss, i'd \n deathmatch ya in a minute!",
    "look, bud. you leave now\nand you forfeit your body count!",
    "just leave. when you come\nback, i'll be waiting with a bat.",
    "you're lucky i don't smack\nyou for thinking about leaving."

    // FinalDOOM?
    "fuck you, pussy!\nget the fuck out!",
    "you quit and i'll jizz\nin your cystholes!",
    "if you leave, i'll make\nthe lord drink my jizz.",
    "hey, ron! can we say\n'fuck' in the game?",
    "i'd leave: this is just\nmore monsters and levels.\nwhat a load.",
    "suck it down, asshole!\nyou're a fucking wimp!",
    "don't quit now! we're \nstill spending your money!",

    // Internal debug. Different style, too.
    "THIS IS NO MESSAGE!\nPage intentionally left blank."
};

```

## 1.8 dstrings.h

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
//
// $Log:$
//
// DESCRIPTION:
//      DOOM strings, by language.
//
//-----

#ifndef __DSTRINGS__
#define __DSTRINGS__

// All important printed strings.
// Language selection (message strings).
// Use -DFRENCH etc.

#ifdef FRENCH
#include "d_french.h"
#else
#include "d_english.h"
#endif

// Misc. other strings.
#define SAVEGAMENAME      "doomsav"

//
// File locations,
// relative to current position.
// Path names are OS-sensitive.
//
#define DEVMAPS "devmaps"
#define DEVDATA "devdata"

// Not done in french?

// QuitDOOM messages
#define NUM_QUITMESSAGES  22

extern char* endmsg[];
```

```

#endif
//-----
//
// $Log:$
//
//-----

```

## 2 Automap

### 2.1 am\_map.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
//
//
// $Log:$
//
// DESCRIPTION:  the automap code
//-----

static const char rcsid[] = "$Id: am_map.c,v 1.4 1997/02/03 21:24:33 b1 Exp $";

#include <stdio.h>

#include "z_zone.h"
#include "doomdef.h"
#include "st_stuff.h"
#include "p_local.h"
#include "w_wad.h"

#include "m_cheat.h"
#include "i_system.h"

// Needs access to LFB.
#include "v_video.h"

// State.
#include "doomstat.h"
#include "r_state.h"

// Data.
#include "dstrings.h"

#include "am_map.h"

// For use if I do walls with outsides/insides
#define REDS (256-5*16)

```

```

#define REDRANGE      16
#define BLUES          (256-4*16+8)
#define BLUERANGE     8
#define GREENS         (7*16)
#define GREENRANGE    16
#define GRAYS          (6*16)
#define GRAYSRANGE    16
#define BROWNS         (4*16)
#define BROWNRANGE    16
#define YELLOWS        (256-32+7)
#define YELLOWRANGE   1
#define BLACK          0
#define WHITE          (256-47)

// Automap colors
#define BACKGROUND    BLACK
#define YOURCOLORS    WHITE
#define YOURRANGE     0
#define WALLCOLORS    REDS
#define WALLRANGE     REDRANGE
#define TSWALLCOLORS  GRAYS
#define TSWALLRANGE   GRAYSRANGE
#define FDWALLCOLORS  BROWNS
#define FDWALLRANGE   BROWNRANGE
#define CDWALLCOLORS  YELLOWS
#define CDWALLRANGE   YELLOWRANGE
#define THINGCOLORS   GREENS
#define THINGRANGE    GREENRANGE
#define SECRETWALLCOLORS WALLCOLORS
#define SECRETWALLRANGE WALLRANGE
#define GRIDCOLORS    (GRAYS + GRAYSRANGE/2)
#define GRIDRANGE     0
#define XHAIRCOLORS   GRAYS

// drawing stuff
#define FB            0

#define AM_PANDOWNKEY    KEY_DOWNARROW
#define AM_PANUPKEY     KEY_UPARROW
#define AM_PANRIGHTKEY  KEY_RIGHTARROW
#define AM_PANLEFTKEY   KEY_LEFTARROW
#define AM_ZOOMINKEY    '='
#define AM_ZOOMOUTKEY   '- '
#define AM_STARTKEY     KEY_TAB
#define AM_ENDKEY       KEY_TAB
#define AM_GOBIGKEY     'O'
#define AM_FOLLOWKEY    'f'
#define AM_GRIDKEY      'g'
#define AM_MARKKEY      'm'
#define AM_CLEARMARKKEY 'c'

#define AM_NUMMARKPOINTS 10

// scale on entry
#define INITSCALEMTOF (.2*FRACUNIT)
// how much the automap moves window per tic in frame-buffer coordinates
// moves 140 pixels in 1 second
#define F_PANINC      4
// how much zoom-in per tic
// goes to 2x in 1 second
#define M_ZOOMIN      ((int) (1.02*FRACUNIT))
// how much zoom-out per tic
// pulls out to 0.5x in 1 second
#define M_ZOOMOUT     ((int) (FRACUNIT/1.02))

```

```

// translates between frame-buffer and map distances
#define FTOM(x) FixedMul(((x)<<16),scale_ftom)
#define MTOF(x) (FixedMul((x),scale_mtof)>>16)
// translates between frame-buffer and map coordinates
#define CXMTOF(x) (f_x + MTOF((x)-m_x))
#define CYMTOF(y) (f_y + (f_h - MTOF((y)-m_y)))

// the following is crap
#define LINE_NEVERSEE ML_DONTDRAW

typedef struct
{
    int x, y;
} fpoint_t;

typedef struct
{
    fpoint_t a, b;
} fline_t;

typedef struct
{
    fixed_t          x,y;
} mpoint_t;

typedef struct
{
    mpoint_t a, b;
} mline_t;

typedef struct
{
    fixed_t slp, islp;
} islope_t;

//
// The vector graphics for the automap.
// A line drawing of the player pointing right,
// starting from the middle.
//
#define R ((8*PLAYERRADIUS)/7)
mline_t player_arrow[] = {
    { { -R+R/8, 0 }, { R, 0 } }, // -----
    { { R, 0 }, { R-R/2, R/4 } }, // ----->
    { { R, 0 }, { R-R/2, -R/4 } },
    { { -R+R/8, 0 }, { -R-R/8, R/4 } }, // >----->
    { { -R+R/8, 0 }, { -R-R/8, -R/4 } },
    { { -R+3*R/8, 0 }, { -R+R/8, R/4 } }, // >>----->
    { { -R+3*R/8, 0 }, { -R+R/8, -R/4 } }
};
#undef R
#define NUMPLYRLINES (sizeof(player_arrow)/sizeof(mline_t))

#define R ((8*PLAYERRADIUS)/7)
mline_t cheat_player_arrow[] = {
    { { -R+R/8, 0 }, { R, 0 } }, // -----
    { { R, 0 }, { R-R/2, R/6 } }, // ----->
    { { R, 0 }, { R-R/2, -R/6 } },
    { { -R+R/8, 0 }, { -R-R/8, R/6 } }, // >----->
    { { -R+R/8, 0 }, { -R-R/8, -R/6 } },
    { { -R+3*R/8, 0 }, { -R+R/8, R/6 } }, // >>----->
    { { -R+3*R/8, 0 }, { -R+R/8, -R/6 } },
    { { -R/2, 0 }, { -R/2, -R/6 } }, // >>-d---->

```



```

    { { -R/2, -R/6 }, { -R/2+R/6, -R/6 } },
    { { -R/2+R/6, -R/6 }, { -R/2+R/6, R/4 } },
    { { -R/6, 0 }, { -R/6, -R/6 } }, // >>-dd-->
    { { -R/6, -R/6 }, { 0, -R/6 } },
    { { 0, -R/6 }, { 0, R/4 } },
    { { R/6, R/4 }, { R/6, -R/7 } }, // >>-ddt->
    { { R/6, -R/7 }, { R/6+R/32, -R/7-R/32 } },
    { { R/6+R/32, -R/7-R/32 }, { R/6+R/10, -R/7 } }
};
#undef R
#define NUMCHEATPLYRLINES (sizeof(cheat_player_arrow)/sizeof(mline_t))

#define R (FRACUNIT)
mline_t triangle_guy[] = {
    { { -.867*R, -.5*R }, { .867*R, -.5*R } },
    { { .867*R, -.5*R }, { 0, R } },
    { { 0, R }, { -.867*R, -.5*R } }
};
#undef R
#define NUMTRIANGLEGUYLINES (sizeof(triangle_guy)/sizeof(mline_t))

#define R (FRACUNIT)
mline_t thintriangle_guy[] = {
    { { -.5*R, -.7*R }, { R, 0 } },
    { { R, 0 }, { -.5*R, .7*R } },
    { { -.5*R, .7*R }, { -.5*R, -.7*R } }
};
#undef R
#define NUMTHINTRIANGLEGUYLINES (sizeof(thintriangle_guy)/sizeof(mline_t))

static int      cheating = 0;
static int      grid = 0;

static int      leveljuststarted = 1;          // kluge until AM_LevelInit() is called

boolean         automapactive = false;
static int      finit_width = SCREENWIDTH;
static int      finit_height = SCREENHEIGHT - 32;

// location of window on screen
static int      f_x;
static int      f_y;

// size of window on screen
static int      f_w;
static int      f_h;

static int      lightlev;                      // used for funky strobing effect
static byte*    fb;                          // pseudo-frame buffer
static int      amclock;

static mpoint_t m_paninc; // how far the window pans each tic (map coords)
static fixed_t  mtof_zoommul; // how far the window zooms in each tic (map coords)
static fixed_t  ftom_zoommul; // how far the window zooms in each tic (fb coords)

static fixed_t  m_x, m_y; // LL x,y where the window is on the map (map coords)
static fixed_t  m_x2, m_y2; // UR x,y where the window is on the map (map coords)

//
// width/height of window on map (map coords)
//
static fixed_t  m_w;

```

```

static fixed_t      m_h;

// based on level size
static fixed_t      min_x;
static fixed_t      min_y;
static fixed_t      max_x;
static fixed_t      max_y;

static fixed_t      max_w; // max_x-min_x,
static fixed_t      max_h; // max_y-min_y

// based on player size
static fixed_t      min_w;
static fixed_t      min_h;

static fixed_t      min_scale_mtof; // used to tell when to stop zooming out
static fixed_t      max_scale_mtof; // used to tell when to stop zooming in

// old stuff for recovery later
static fixed_t      old_m_w, old_m_h;
static fixed_t      old_m_x, old_m_y;

// old location used by the Follower routine
static mpoint_t      f_oldloc;

// used by MTOF to scale from map-to-frame-buffer coords
static fixed_t      scale_mtof = INITSCALEMTOF;
// used by FTOM to scale from frame-buffer-to-map coords (=1/scale_mtof)
static fixed_t      scale_ftom;

static player_t      *plr; // the player represented by an arrow

static patch_t      *marknums[10]; // numbers used for marking by the automap
static mpoint_t      markpoints[AM_NUMMARKPOINTS]; // where the points are
static int           markpointnum = 0; // next point to be assigned

static int           followplayer = 1; // specifies whether to follow the player around

static unsigned char cheat_amap_seq[] = { 0xb2, 0x26, 0x26, 0x2e, 0xff };
static cheatseq_t      cheat_amap = { cheat_amap_seq, 0 };

static boolean       stopped = true;

extern boolean       viewactive;
//extern byte         screens[][SCREENWIDTH*SCREENHEIGHT];

void
V_MarkRect
( int      x,
  int      y,
  int      width,
  int      height );

// Calculates the slope and slope according to the x-axis of a line
// segment in map coordinates (with the upright y-axis n' all) so
// that it can be used with the brain-dead drawing stuff.

void
AM_getIslope
( mline_t*      ml,
  islope_t*      is )
{

```

```

int dx, dy;

dy = ml->a.y - ml->b.y;
dx = ml->b.x - ml->a.x;
if (!dy) is->islp = (dx<0?-MAXINT:MAXINT);
else is->islp = FixedDiv(dx, dy);
if (!dx) is->slp = (dy<0?-MAXINT:MAXINT);
else is->slp = FixedDiv(dy, dx);

}

//
//
//
void AM_activateNewScale(void)
{
    m_x += m_w/2;
    m_y += m_h/2;
    m_w = FTOM(f_w);
    m_h = FTOM(f_h);
    m_x -= m_w/2;
    m_y -= m_h/2;
    m_x2 = m_x + m_w;
    m_y2 = m_y + m_h;
}

//
//
//
void AM_saveScaleAndLoc(void)
{
    old_m_x = m_x;
    old_m_y = m_y;
    old_m_w = m_w;
    old_m_h = m_h;
}

//
//
//
void AM_restoreScaleAndLoc(void)
{
    m_w = old_m_w;
    m_h = old_m_h;
    if (!followplayer)
    {
        m_x = old_m_x;
        m_y = old_m_y;
    } else {
        m_x = plr->mo->x - m_w/2;
        m_y = plr->mo->y - m_h/2;
    }
    m_x2 = m_x + m_w;
    m_y2 = m_y + m_h;

    // Change the scaling multipliers
    scale_mtof = FixedDiv(f_w<<FRACBITS, m_w);
    scale_ftom = FixedDiv(FRACUNIT, scale_mtof);
}

//
// adds a marker at the current location
//
void AM_addMark(void)

```

```

{
    markpoints[markpointnum].x = m_x + m_w/2;
    markpoints[markpointnum].y = m_y + m_h/2;
    markpointnum = (markpointnum + 1) % AM_NUMMARKPOINTS;
}

//
// Determines bounding box of all vertices,
// sets global variables controlling zoom range.
//
void AM_findMinMaxBoundaries(void)
{
    int i;
    fixed_t a;
    fixed_t b;

    min_x = min_y = MAXINT;
    max_x = max_y = -MAXINT;

    for (i=0;i<numvertexes;i++)
    {
        if (vertexes[i].x < min_x)
            min_x = vertexes[i].x;
        else if (vertexes[i].x > max_x)
            max_x = vertexes[i].x;

        if (vertexes[i].y < min_y)
            min_y = vertexes[i].y;
        else if (vertexes[i].y > max_y)
            max_y = vertexes[i].y;
    }

    max_w = max_x - min_x;
    max_h = max_y - min_y;

    min_w = 2*PLAYERRADIUS; // const? never changed?
    min_h = 2*PLAYERRADIUS;

    a = FixedDiv(f_w<<FRACBITS, max_w);
    b = FixedDiv(f_h<<FRACBITS, max_h);

    min_scale_mtof = a < b ? a : b;
    max_scale_mtof = FixedDiv(f_h<<FRACBITS, 2*PLAYERRADIUS);
}

//
//
//
void AM_changeWindowLoc(void)
{
    if (m_paninc.x || m_paninc.y)
    {
        followplayer = 0;
        f_oldloc.x = MAXINT;
    }

    m_x += m_paninc.x;
    m_y += m_paninc.y;

    if (m_x + m_w/2 > max_x)
        m_x = max_x - m_w/2;
    else if (m_x + m_w/2 < min_x)

```

```

        m_x = min_x - m_w/2;

    if (m_y + m_h/2 > max_y)
        m_y = max_y - m_h/2;
    else if (m_y + m_h/2 < min_y)
        m_y = min_y - m_h/2;

    m_x2 = m_x + m_w;
    m_y2 = m_y + m_h;
}

//
//
//
void AM_initVariables(void)
{
    int pnum;
    static event_t st_notify = { ev_keyup, AM_MSGENTERED };

    automapactive = true;
    fb = screens[0];

    f_oldloc.x = MAXINT;
    amclock = 0;
    lightlev = 0;

    m_paninc.x = m_paninc.y = 0;
    ftom_zoommul = FRACUNIT;
    mtof_zoommul = FRACUNIT;

    m_w = FTOM(f_w);
    m_h = FTOM(f_h);

    // find player to center on initially
    if (!playeringame[pnum = consoleplayer])
        for (pnum=0;pnum<MAXPLAYERS;pnum++)
            if (playeringame[pnum])
                break;

    plr = &players[pnum];
    m_x = plr->mo->x - m_w/2;
    m_y = plr->mo->y - m_h/2;
    AM_changeWindowLoc();

    // for saving & restoring
    old_m_x = m_x;
    old_m_y = m_y;
    old_m_w = m_w;
    old_m_h = m_h;

    // inform the status bar of the change
    ST_Responder(&st_notify);
}

//
//
//
void AM_loadPics(void)
{
    int i;
    char namebuf[9];

    for (i=0;i<10;i++)

```

```

    {
        sprintf(namebuf, "AMNUM%d", i);
        marknums[i] = W_CacheLumpName(namebuf, PU_STATIC);
    }
}

void AM_unloadPics(void)
{
    int i;

    for (i=0;i<10;i++)
        Z_ChangeTag(marknums[i], PU_CACHE);
}

void AM_clearMarks(void)
{
    int i;

    for (i=0;i<AM_NUMMARKPOINTS;i++)
        markpoints[i].x = -1; // means empty
    markpointnum = 0;
}

//
// should be called at the start of every level
// right now, i figure it out myself
//
void AM_LevelInit(void)
{
    leveljuststarted = 0;

    f_x = f_y = 0;
    f_w = finit_width;
    f_h = finit_height;

    AM_clearMarks();

    AM_findMinMaxBoundaries();
    scale_mtof = FixedDiv(min_scale_mtof, (int) (0.7*FRACUNIT));
    if (scale_mtof > max_scale_mtof)
        scale_mtof = min_scale_mtof;
    scale_ftom = FixedDiv(FRACUNIT, scale_mtof);
}

//
//
//
void AM_Stop (void)
{
    static event_t st_notify = { 0, ev_keyup, AM_MSGEXITED };

    AM_unloadPics();
    automapactive = false;
    ST_Responder(&st_notify);
    stopped = true;
}

//
//
//

```

```

void AM_Start (void)
{
    static int lastlevel = -1, lastepisode = -1;

    if (!stopped) AM_Stop();
    stopped = false;
    if (lastlevel != gamemap || lastepisode != gameepisode)
    {
        AM_LevelInit();
        lastlevel = gamemap;
        lastepisode = gameepisode;
    }
    AM_initVariables();
    AM_loadPics();
}

//
// set the window scale to the maximum size
//
void AM_minOutWindowScale(void)
{
    scale_mtof = min_scale_mtof;
    scale_ftom = FixedDiv(FRACUNIT, scale_mtof);
    AM_activateNewScale();
}

//
// set the window scale to the minimum size
//
void AM_maxOutWindowScale(void)
{
    scale_mtof = max_scale_mtof;
    scale_ftom = FixedDiv(FRACUNIT, scale_mtof);
    AM_activateNewScale();
}

//
// Handle events (user inputs) in automap mode
//
boolean
AM_Responder
( event_t*      ev )
{
    int rc;
    static int cheatstate=0;
    static int bigstate=0;
    static char buffer[20];

    rc = false;

    if (!automapactive)
    {
        if (ev->type == ev_keydown && ev->data1 == AM_STARTKEY)
        {
            AM_Start ();
            viewactive = false;
            rc = true;
        }
    }

    else if (ev->type == ev_keydown)
    {

```

```

rc = true;
switch(ev->data1)
{
case AM_PANRIGHTKEY: // pan right
    if (!followplayer) m_paninc.x = FTOM(F_PANINC);
    else rc = false;
    break;
case AM_PANLEFTKEY: // pan left
    if (!followplayer) m_paninc.x = -FTOM(F_PANINC);
    else rc = false;
    break;
case AM_PANUPKEY: // pan up
    if (!followplayer) m_paninc.y = FTOM(F_PANINC);
    else rc = false;
    break;
case AM_PANDOWNKEY: // pan down
    if (!followplayer) m_paninc.y = -FTOM(F_PANINC);
    else rc = false;
    break;
case AM_ZOOMOUTKEY: // zoom out
    mtof_zoommul = M_ZOOMOUT;
    ftom_zoommul = M_ZOOMIN;
    break;
case AM_ZOOMINKEY: // zoom in
    mtof_zoommul = M_ZOOMIN;
    ftom_zoommul = M_ZOOMOUT;
    break;
case AM_ENDKEY:
    bigstate = 0;
    viewactive = true;
    AM_Stop ();
    break;
case AM_GOBIGKEY:
    bigstate = !bigstate;
    if (bigstate)
    {
        AM_saveScaleAndLoc();
        AM_minOutWindowScale();
    }
    else AM_restoreScaleAndLoc();
    break;
case AM_FOLLOWKEY:
    followplayer = !followplayer;
    f_oldloc.x = MAXINT;
    plr->message = followplayer ? AMSTR_FOLLOWON : AMSTR_FOLLOWOFF;
    break;
case AM_GRIDKEY:
    grid = !grid;
    plr->message = grid ? AMSTR_GRIDON : AMSTR_GRIDOFF;
    break;
case AM_MARKKEY:
    sprintf(buffer, "%s %d", AMSTR_MARKEDSPOT, markpointnum);
    plr->message = buffer;
    AM_addMark();
    break;
case AM_CLEARMARKKEY:
    AM_clearMarks();
    plr->message = AMSTR_MARKSCLEARED;
    break;
default:
    cheatstate=0;
    rc = false;
}
}
if (!deathmatch && cht_CheckCheat(&cheat_amap, ev->data1))
{

```



```

        rc = false;
        cheating = (cheating+1) % 3;
    }
}

else if (ev->type == ev_keyup)
{
    rc = false;
    switch (ev->data1)
    {
        case AM_PANRIGHTKEY:
            if (!followplayer) m_paninc.x = 0;
            break;
        case AM_PANLEFTKEY:
            if (!followplayer) m_paninc.x = 0;
            break;
        case AM_PANUPKEY:
            if (!followplayer) m_paninc.y = 0;
            break;
        case AM_PANDOWNKEY:
            if (!followplayer) m_paninc.y = 0;
            break;
        case AM_ZOOMOUTKEY:
        case AM_ZOOMINKEY:
            mtof_zoommul = FRACUNIT;
            ftom_zoommul = FRACUNIT;
            break;
    }
}

return rc;
}

//
// Zooming
//
void AM_changeWindowScale(void)
{
    // Change the scaling multipliers
    scale_mtof = FixedMul(scale_mtof, mtof_zoommul);
    scale_ftom = FixedDiv(FRACUNIT, scale_mtof);

    if (scale_mtof < min_scale_mtof)
        AM_minOutWindowScale();
    else if (scale_mtof > max_scale_mtof)
        AM_maxOutWindowScale();
    else
        AM_activateNewScale();
}

//
//
//
void AM_doFollowPlayer(void)
{
    if (f_oldloc.x != plr->mo->x || f_oldloc.y != plr->mo->y)
    {
        m_x = FTOM(MTOF(plr->mo->x)) - m_w/2;
        m_y = FTOM(MTOF(plr->mo->y)) - m_h/2;
        m_x2 = m_x + m_w;
    }
}

```

```

    m_y2 = m_y + m_h;
    f_oldloc.x = plr->mo->x;
    f_oldloc.y = plr->mo->y;

    // m_x = FTOM(MTOF(plr->mo->x - m_w/2));
    // m_y = FTOM(MTOF(plr->mo->y - m_h/2));
    // m_x = plr->mo->x - m_w/2;
    // m_y = plr->mo->y - m_h/2;

}

}

//
//
//
void AM_updateLightLev(void)
{
    static nexttic = 0;
    //static int litelevels[] = { 0, 3, 5, 6, 6, 7, 7, 7 };
    static int litelevels[] = { 0, 4, 7, 10, 12, 14, 15, 15 };
    static int litelevelscnt = 0;

    // Change light level
    if (amclock>nexttic)
    {
        lightlev = litelevels[litelevelscnt++];
        if (litelevelscnt == sizeof(litelevels)/sizeof(int)) litelevelscnt = 0;
        nexttic = amclock + 6 - (amclock % 6);
    }

}

//
// Updates on Game Tick
//
void AM_Ticker (void)
{
    if (!automapactive)
        return;

    amclock++;

    if (followplayer)
        AM_doFollowPlayer();

    // Change the zoom if necessary
    if (ftom_zoommul != FRACUNIT)
        AM_changeWindowScale();

    // Change x,y location
    if (m_paninc.x || m_paninc.y)
        AM_changeWindowLoc();

    // Update light level
    // AM_updateLightLev();

}

//
// Clear automap frame buffer.
//

```

```

void AM_clearFB(int color)
{
    memset(fb, color, f_w*f_h);
}

//
// Automap clipping of lines.
//
// Based on Cohen-Sutherland clipping algorithm but with a slightly
// faster reject and precalculated slopes. If the speed is needed,
// use a hash algorithm to handle the common cases.
//
boolean
AM_clipMline
( mline_t*      ml,
  fline_t*      fl )
{
    enum
    {
        LEFT      =1,
        RIGHT     =2,
        BOTTOM     =4,
        TOP       =8
    };

    register      outcode1 = 0;
    register      outcode2 = 0;
    register      outside;

    fpoint_t      tmp;
    int            dx;
    int            dy;

#define DOOUTCODE(oc, mx, my) \
    (oc) = 0; \
    if ((my) < 0) (oc) |= TOP; \
    else if ((my) >= f_h) (oc) |= BOTTOM; \
    if ((mx) < 0) (oc) |= LEFT; \
    else if ((mx) >= f_w) (oc) |= RIGHT;

    // do trivial rejects and outcodes
    if (ml->a.y > m_y2)
        outcode1 = TOP;
    else if (ml->a.y < m_y)
        outcode1 = BOTTOM;

    if (ml->b.y > m_y2)
        outcode2 = TOP;
    else if (ml->b.y < m_y)
        outcode2 = BOTTOM;

    if (outcode1 & outcode2)
        return false; // trivially outside

    if (ml->a.x < m_x)
        outcode1 |= LEFT;
    else if (ml->a.x > m_x2)
        outcode1 |= RIGHT;

    if (ml->b.x < m_x)
        outcode2 |= LEFT;
    else if (ml->b.x > m_x2)

```

```

    outcode2 |= RIGHT;

if (outcode1 & outcode2)
    return false; // trivially outside

// transform to frame-buffer coordinates.
fl->a.x = CXMTOF(ml->a.x);
fl->a.y = CYMTOF(ml->a.y);
fl->b.x = CXMTOF(ml->b.x);
fl->b.y = CYMTOF(ml->b.y);

DOOUTCODE(outcode1, fl->a.x, fl->a.y);
DOOUTCODE(outcode2, fl->b.x, fl->b.y);

if (outcode1 & outcode2)
    return false;

while (outcode1 | outcode2)
{
    // may be partially inside box
    // find an outside point
    if (outcode1)
        outside = outcode1;
    else
        outside = outcode2;

    // clip to each side
    if (outside & TOP)
    {
        dy = fl->a.y - fl->b.y;
        dx = fl->b.x - fl->a.x;
        tmp.x = fl->a.x + (dx*(fl->a.y))/dy;
        tmp.y = 0;
    }
    else if (outside & BOTTOM)
    {
        dy = fl->a.y - fl->b.y;
        dx = fl->b.x - fl->a.x;
        tmp.x = fl->a.x + (dx*(fl->a.y-f_h))/dy;
        tmp.y = f_h-1;
    }
    else if (outside & RIGHT)
    {
        dy = fl->b.y - fl->a.y;
        dx = fl->b.x - fl->a.x;
        tmp.y = fl->a.y + (dy*(f_w-1 - fl->a.x))/dx;
        tmp.x = f_w-1;
    }
    else if (outside & LEFT)
    {
        dy = fl->b.y - fl->a.y;
        dx = fl->b.x - fl->a.x;
        tmp.y = fl->a.y + (dy*(-fl->a.x))/dx;
        tmp.x = 0;
    }

    if (outside == outcode1)
    {
        fl->a = tmp;
        DOOUTCODE(outcode1, fl->a.x, fl->a.y);
    }
    else
    {
        fl->b = tmp;
        DOOUTCODE(outcode2, fl->b.x, fl->b.y);
    }
}

```

```

    }

    if (outcode1 & outcode2)
        return false; // trivially outside
}

return true;
}
#undef DOOUTCODE

//
// Classic Bresenham w/ whatever optimizations needed for speed
//
void
AM_drawFline
( fline_t*      fl,
  int           color )
{
    register int x;
    register int y;
    register int dx;
    register int dy;
    register int sx;
    register int sy;
    register int ax;
    register int ay;
    register int d;

    static fuck = 0;

    // For debugging only
    if (      fl->a.x < 0 || fl->a.x >= f_w
        || fl->a.y < 0 || fl->a.y >= f_h
        || fl->b.x < 0 || fl->b.x >= f_w
        || fl->b.y < 0 || fl->b.y >= f_h)
    {
        fprintf(stderr, "fuck %d \r", fuck++);
        return;
    }

#define PUTDOT(xx,yy,cc) fb[(yy)*f_w+(xx)]=(cc)

    dx = fl->b.x - fl->a.x;
    ax = 2 * (dx<0 ? -dx : dx);
    sx = dx<0 ? -1 : 1;

    dy = fl->b.y - fl->a.y;
    ay = 2 * (dy<0 ? -dy : dy);
    sy = dy<0 ? -1 : 1;

    x = fl->a.x;
    y = fl->a.y;

    if (ax > ay)
    {
        d = ay - ax/2;
        while (1)
        {
            PUTDOT(x,y,color);
            if (x == fl->b.x) return;
            if (d>=0)
            {
                y += sy;
                d -= ax;
            }
        }
    }

```

```

        }
        x += sx;
        d += ay;
    }
}
else
{
    d = ax - ay/2;
    while (1)
    {
        PUTDOT(x, y, color);
        if (y == fl->b.y) return;
        if (d >= 0)
        {
            x += sx;
            d -= ay;
        }
        y += sy;
        d += ax;
    }
}
}

//
// Clip lines, draw visible part of lines.
//
void
AM_drawMline
( mline_t*      ml,
  int           color )
{
    static fline_t fl;

    if (AM_clipMline(ml, &fl))
        AM_drawFline(&fl, color); // draws it on frame buffer using fb coords
}

//
// Draws flat (floor/ceiling tile) aligned grid lines.
//
void AM_drawGrid(int color)
{
    fixed_t x, y;
    fixed_t start, end;
    mline_t ml;

    // Figure out start of vertical gridlines
    start = m_x;
    if ((start-bmaporgx)%(MAPBLOCKUNITS<<FRACBITS))
        start += (MAPBLOCKUNITS<<FRACBITS)
            - ((start-bmaporgx)%(MAPBLOCKUNITS<<FRACBITS));
    end = m_x + m_w;

    // draw vertical gridlines
    ml.a.y = m_y;
    ml.b.y = m_y+m_h;
    for (x=start; x<end; x+=(MAPBLOCKUNITS<<FRACBITS))
    {
        ml.a.x = x;
        ml.b.x = x;
        AM_drawMline(&ml, color);
    }
}

```

```

// Figure out start of horizontal gridlines
start = m_y;
if ((start-bmaporgy)%(MAPBLOCKUNITS<<FRACBITS))
    start += (MAPBLOCKUNITS<<FRACBITS)
        - ((start-bmaporgy)%(MAPBLOCKUNITS<<FRACBITS));
end = m_y + m_h;

// draw horizontal gridlines
ml.a.x = m_x;
ml.b.x = m_x + m_w;
for (y=start; y<end; y+=(MAPBLOCKUNITS<<FRACBITS))
{
    ml.a.y = y;
    ml.b.y = y;
    AM_drawMline(&ml, color);
}

}

//
// Determines visible lines, draws them.
// This is LineDef based, not LineSeg based.
//
void AM_drawWalls(void)
{
    int i;
    static mline_t l;

    for (i=0;i<numlines;i++)
    {
        l.a.x = lines[i].v1->x;
        l.a.y = lines[i].v1->y;
        l.b.x = lines[i].v2->x;
        l.b.y = lines[i].v2->y;
        if (cheating || (lines[i].flags & ML_MAPPED))
        {
            if ((lines[i].flags & LINE_NEVERSEE) && !cheating)
                continue;
            if (!lines[i].backsector)
            {
                AM_drawMline(&l, WALLCOLORS+lightlev);
            }
            else
            {
                if (lines[i].special == 39)
                { // teleporters
                    AM_drawMline(&l, WALLCOLORS+WALLRANGE/2);
                }
                else if (lines[i].flags & ML_SECRET) // secret door
                {
                    if (cheating) AM_drawMline(&l, SECRETWALLCOLORS + lightlev);
                    else AM_drawMline(&l, WALLCOLORS+lightlev);
                }
                else if (lines[i].backsector->floorheight
                        != lines[i].frontsector->floorheight) {
                    AM_drawMline(&l, FDWALLCOLORS + lightlev); // floor level change
                }
                else if (lines[i].backsector->ceilingheight
                        != lines[i].frontsector->ceilingheight) {
                    AM_drawMline(&l, CDWALLCOLORS+lightlev); // ceiling level change
                }
                else if (cheating) {
                    AM_drawMline(&l, TSWALLCOLORS+lightlev);
                }
            }
        }
    }
}

```

```

    }
}
else if (plr->powers[pw_allmap])
{
    if (!(lines[i].flags & LINE_NEVERSEE)) AM_drawMline(&l, GRAYS+3);
}
}
}

```

```

//
// Rotation in 2D.
// Used to rotate player arrow line character.
//
void
AM_rotate
( fixed_t*      x,
  fixed_t*      y,
  angle_t       a )
{
    fixed_t tmpx;

    tmpx =
        FixedMul(*x, finecosine[a>>ANGLETOFINESHIFT])
        - FixedMul(*y, finesine[a>>ANGLETOFINESHIFT]);

    *y =
        FixedMul(*x, finesine[a>>ANGLETOFINESHIFT])
        + FixedMul(*y, finecosine[a>>ANGLETOFINESHIFT]);

    *x = tmpx;
}

```

```

void
AM_drawLineCharacter
( mline_t*      lineguy,
  int            lineguylines,
  fixed_t        scale,
  angle_t        angle,
  int            color,
  fixed_t        x,
  fixed_t        y )
{
    int           i;
    mline_t       l;

    for (i=0; i<lineguylines; i++)
    {
        l.a.x = lineguy[i].a.x;
        l.a.y = lineguy[i].a.y;

        if (scale)
        {
            l.a.x = FixedMul(scale, l.a.x);
            l.a.y = FixedMul(scale, l.a.y);
        }

        if (angle)
            AM_rotate(&l.a.x, &l.a.y, angle);

        l.a.x += x;
        l.a.y += y;

        l.b.x = lineguy[i].b.x;
        l.b.y = lineguy[i].b.y;
    }
}

```



```

    if (scale)
    {
        l.b.x = FixedMul(scale, l.b.x);
        l.b.y = FixedMul(scale, l.b.y);
    }

    if (angle)
        AM_rotate(&l.b.x, &l.b.y, angle);

    l.b.x += x;
    l.b.y += y;

    AM_drawMline(&l, color);
}
}

void AM_drawPlayers(void)
{
    int            i;
    player_t*      p;
    static int      their_colors[] = { GREENS, GRAYS, BROWNS, REDS };
    int            their_color = -1;
    int            color;

    if (!netgame)
    {
        if (cheating)
            AM_drawLineCharacter
                (cheat_player_arrow, NUMCHEATPLYRLINES, 0,
                 plr->mo->angle, WHITE, plr->mo->x, plr->mo->y);
        else
            AM_drawLineCharacter
                (player_arrow, NUMPLYRLINES, 0, plr->mo->angle,
                 WHITE, plr->mo->x, plr->mo->y);
        return;
    }

    for (i=0; i<MAXPLAYERS; i++)
    {
        their_color++;
        p = &players[i];

        if ( (deathmatch && !singledemo) && p != plr)
            continue;

        if (!playeringame[i])
            continue;

        if (p->powers[pw_invisibility])
            color = 246; // *close* to black
        else
            color = their_colors[their_color];

        AM_drawLineCharacter
            (player_arrow, NUMPLYRLINES, 0, p->mo->angle,
             color, p->mo->x, p->mo->y);
    }
}

void
AM_drawThings
( int      colors,
  int      colorrange)

```

```

{
    int            i;
    mobj_t*        t;

    for (i=0;i<numsectors;i++)
    {
        t = sectors[i].thinglist;
        while (t)
        {
            AM_drawLineCharacter
                (thintriangle_guy, NUMTHINTRIANGLEGUYLINES,
                 16<<FRACBITS, t->angle, colors+lightlev, t->x, t->y);
            t = t->snext;
        }
    }
}

void AM_drawMarks(void)
{
    int i, fx, fy, w, h;

    for (i=0;i<AM_NUMMARKPOINTS;i++)
    {
        if (markpoints[i].x != -1)
        {
            //      w = SHORT(marknums[i]->width);
            //      h = SHORT(marknums[i]->height);
            w = 5; // because something's wrong with the wad, i guess
            h = 6; // because something's wrong with the wad, i guess
            fx = CXMTOF(markpoints[i].x);
            fy = CYMTOF(markpoints[i].y);
            if (fx >= f_x && fx <= f_w - w && fy >= f_y && fy <= f_h - h)
                V_DrawPatch(fx, fy, FB, marknums[i]);
        }
    }
}

void AM_drawCrosshair(int color)
{
    fb[(f_w*(f_h+1))/2] = color; // single point for now
}

void AM_Drawer (void)
{
    if (!automapactive) return;

    AM_clearFB(BACKGROUND);
    if (grid)
        AM_drawGrid(GRIDCOLORS);
    AM_drawWalls();
    AM_drawPlayers();
    if (cheating==2)
        AM_drawThings(THINGCOLORS, THINGRANGE);
    AM_drawCrosshair(XHAIRCOLORS);

    AM_drawMarks();

    V_MarkRect(f_x, f_y, f_w, f_h);
}

```

## 2.2 am\_map.h

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//   AutoMap module.
//-----

#ifndef __AMMAP_H__
#define __AMMAP_H__

// Used by ST StatusBar stuff.
#define AM_MSGHEADER (('a'<<24)+('m'<<16))
#define AM_MSGENTERED (AM_MSGHEADER | ('e'<<8))
#define AM_MSGEXITED (AM_MSGHEADER | ('x'<<8))

// Called by main loop.
boolean AM_Responder (event_t* ev);

// Called by main loop.
void AM_Ticker (void);

// Called by main loop,
// called instead of view drawer if automap active.
void AM_Drawer (void);

// Called to force the automap to quit
// if the level is completed while it is up.
void AM_Stop (void);

#endif
//-----
//
// $Log:$
//-----
```

## 3 Initialisation/general code

### 3.1 d\_english.h

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
```

```

// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Printed strings for translation.
//     English language support (default).
//
//-----

#ifdef __D_ENGLISH__
#define __D_ENGLISH__

//
//     Printed strings for translation
//

//
// D_Main.C
//
#define D_DEVSTR      "Development mode ON.\n"
#define D_CDROM       "CD-ROM Version: default.cfg from c:\\doomdata\n"

//
//     M_Menu.C
//
#define PRESSKEY      "press a key."
#define PRESSYN       "press y or n."
#define QUITMSG       "are you sure you want to\nquit this great game?"
#define LOADNET       "you can't do load while in a net game!\n\n"PRESSKEY
#define QLOADNET      "you can't quickload during a netgame!\n\n"PRESSKEY
#define QSAVESPOT      "you haven't picked a quicksave slot yet!\n\n"PRESSKEY
#define SAVEDEAD       "you can't save if you aren't playing!\n\n"PRESSKEY
#define QSPROMPT      "quicksave over your game named\n\n'%s'?\n\n"PRESSYN
#define QLPROMPT      "do you want to quickload the game named\n\n'%s'?\n\n"PRESSYN

#define NEWGAME       \
"you can't start a new game\n"\
"while in a network game.\n\n"PRESSKEY

#define NIGHTMARE      \
"are you sure? this skill level\n"\
"isn't even remotely fair.\n\n"PRESSYN

#define SWSTRING       \
"this is the shareware version of doom.\n\n"\
"you need to order the entire trilogy.\n\n"PRESSKEY

#define MSGOFF         "Messages OFF"
#define MSGON          "Messages ON"
#define NETEND         "you can't end a netgame!\n\n"PRESSKEY
#define ENDGAME        "are you sure you want to end the game?\n\n"PRESSYN

#define DOSY           "(press y to quit)"

#define DETAILHI       "High detail"
#define DETAILLO       "Low detail"

```

```

#define GAMMALVLO      "Gamma correction OFF"
#define GAMMALVL1      "Gamma correction level 1"
#define GAMMALVL2      "Gamma correction level 2"
#define GAMMALVL3      "Gamma correction level 3"
#define GAMMALVL4      "Gamma correction level 4"
#define EMPTYSTRING    "empty slot"

//
//      P_inter.C
//
#define GOTARMOR        "Picked up the armor."
#define GOTMEGA         "Picked up the MegaArmor!"
#define GOTHTHBONUS     "Picked up a health bonus."
#define GOTARMBONUS     "Picked up an armor bonus."
#define GOTSTIM         "Picked up a stimpack."
#define GOTMEDINEED     "Picked up a medikit that you REALLY need!"
#define GOTMEDIKIT      "Picked up a medikit."
#define GOTSUPER        "Supercharge!"

#define GOTBLUECARD     "Picked up a blue keycard."
#define GOTYELWCARD     "Picked up a yellow keycard."
#define GOTREDCARD      "Picked up a red keycard."
#define GOTBLUESKUL     "Picked up a blue skull key."
#define GOTYELWSKUL     "Picked up a yellow skull key."
#define GOTREDSKULL     "Picked up a red skull key."

#define GOTINVUL        "Invulnerability!"
#define GOTBERSERK      "Berserk!"
#define GOTINVIS        "Partial Invisibility"
#define GOTSUIT         "Radiation Shielding Suit"
#define GOTMAP          "Computer Area Map"
#define GOTVISOR        "Light Amplification Visor"
#define GOTMSPHERE      "MegaSphere!"

#define GOTCLIP         "Picked up a clip."
#define GOTCLIPBOX      "Picked up a box of bullets."
#define GOTROCKET       "Picked up a rocket."
#define GOTROCKBOX      "Picked up a box of rockets."
#define GOTCELL         "Picked up an energy cell."
#define GOTCELLBOX      "Picked up an energy cell pack."
#define GOTSHELLS       "Picked up 4 shotgun shells."
#define GOTSHELLBOX     "Picked up a box of shotgun shells."
#define GOTBACKPACK     "Picked up a backpack full of ammo!"

#define GOTBFG9000      "You got the BFG9000! Oh, yes."
#define GOTCHAINGUN     "You got the chaingun!"
#define GOTCHAINSAW     "A chainsaw! Find some meat!"
#define GOTLAUNCHER     "You got the rocket launcher!"
#define GOTPLASMA       "You got the plasma gun!"
#define GOTSHOTGUN      "You got the shotgun!"
#define GOTSHOTGUN2     "You got the super shotgun!"

//
//      P_Doors.C
//
#define PD_BLUEO        "You need a blue key to activate this object"
#define PD_REDO         "You need a red key to activate this object"
#define PD_YELLOWO      "You need a yellow key to activate this object"
#define PD_BLUEK        "You need a blue key to open this door"
#define PD_REDK         "You need a red key to open this door"
#define PD_YELLOWK      "You need a yellow key to open this door"

//
//      G_game.C
//

```

```

#define GGSAVED          "game saved."

//
//      HU_stuff.C
//
#define HUSTR_MSGU       "[Message unsent]"

#define HUSTR_E1M1       "E1M1: Hangar"
#define HUSTR_E1M2       "E1M2: Nuclear Plant"
#define HUSTR_E1M3       "E1M3: Toxin Refinery"
#define HUSTR_E1M4       "E1M4: Command Control"
#define HUSTR_E1M5       "E1M5: Phobos Lab"
#define HUSTR_E1M6       "E1M6: Central Processing"
#define HUSTR_E1M7       "E1M7: Computer Station"
#define HUSTR_E1M8       "E1M8: Phobos Anomaly"
#define HUSTR_E1M9       "E1M9: Military Base"

#define HUSTR_E2M1       "E2M1: Deimos Anomaly"
#define HUSTR_E2M2       "E2M2: Containment Area"
#define HUSTR_E2M3       "E2M3: Refinery"
#define HUSTR_E2M4       "E2M4: Deimos Lab"
#define HUSTR_E2M5       "E2M5: Command Center"
#define HUSTR_E2M6       "E2M6: Halls of the Damned"
#define HUSTR_E2M7       "E2M7: Spawning Vats"
#define HUSTR_E2M8       "E2M8: Tower of Babel"
#define HUSTR_E2M9       "E2M9: Fortress of Mystery"

#define HUSTR_E3M1       "E3M1: Hell Keep"
#define HUSTR_E3M2       "E3M2: Slough of Despair"
#define HUSTR_E3M3       "E3M3: Pandemonium"
#define HUSTR_E3M4       "E3M4: House of Pain"
#define HUSTR_E3M5       "E3M5: Unholy Cathedral"
#define HUSTR_E3M6       "E3M6: Mt. Erebus"
#define HUSTR_E3M7       "E3M7: Limbo"
#define HUSTR_E3M8       "E3M8: Dis"
#define HUSTR_E3M9       "E3M9: Warrens"

#define HUSTR_E4M1       "E4M1: Hell Beneath"
#define HUSTR_E4M2       "E4M2: Perfect Hatred"
#define HUSTR_E4M3       "E4M3: Sever The Wicked"
#define HUSTR_E4M4       "E4M4: Unruly Evil"
#define HUSTR_E4M5       "E4M5: They Will Repent"
#define HUSTR_E4M6       "E4M6: Against Thee Wickedly"
#define HUSTR_E4M7       "E4M7: And Hell Followed"
#define HUSTR_E4M8       "E4M8: Unto The Cruel"
#define HUSTR_E4M9       "E4M9: Fear"

#define HUSTR_1          "level 1: entryway"
#define HUSTR_2          "level 2: underhalls"
#define HUSTR_3          "level 3: the gantlet"
#define HUSTR_4          "level 4: the focus"
#define HUSTR_5          "level 5: the waste tunnels"
#define HUSTR_6          "level 6: the crusher"
#define HUSTR_7          "level 7: dead simple"
#define HUSTR_8          "level 8: tricks and traps"
#define HUSTR_9          "level 9: the pit"
#define HUSTR_10         "level 10: refueling base"
#define HUSTR_11         "level 11: 'o' of destruction!"

#define HUSTR_12         "level 12: the factory"
#define HUSTR_13         "level 13: downtown"
#define HUSTR_14         "level 14: the inmost dens"
#define HUSTR_15         "level 15: industrial zone"
#define HUSTR_16         "level 16: suburbs"
#define HUSTR_17         "level 17: tenements"

```

```

#define HUSTR_18      "level 18: the courtyard"
#define HUSTR_19      "level 19: the citadel"
#define HUSTR_20      "level 20: gotcha!"

#define HUSTR_21      "level 21: nirvana"
#define HUSTR_22      "level 22: the catacombs"
#define HUSTR_23      "level 23: barrels o' fun"
#define HUSTR_24      "level 24: the chasm"
#define HUSTR_25      "level 25: bloodfalls"
#define HUSTR_26      "level 26: the abandoned mines"
#define HUSTR_27      "level 27: monster condo"
#define HUSTR_28      "level 28: the spirit world"
#define HUSTR_29      "level 29: the living end"
#define HUSTR_30      "level 30: icon of sin"

#define HUSTR_31      "level 31: wolfenstein"
#define HUSTR_32      "level 32: grosse"

#define PHUSTR_1      "level 1: congo"
#define PHUSTR_2      "level 2: well of souls"
#define PHUSTR_3      "level 3: aztec"
#define PHUSTR_4      "level 4: caged"
#define PHUSTR_5      "level 5: ghost town"
#define PHUSTR_6      "level 6: baron's lair"
#define PHUSTR_7      "level 7: caughtyard"
#define PHUSTR_8      "level 8: realm"
#define PHUSTR_9      "level 9: abattoire"
#define PHUSTR_10     "level 10: onslaught"
#define PHUSTR_11     "level 11: hunted"

#define PHUSTR_12     "level 12: speed"
#define PHUSTR_13     "level 13: the crypt"
#define PHUSTR_14     "level 14: genesis"
#define PHUSTR_15     "level 15: the twilight"
#define PHUSTR_16     "level 16: the omen"
#define PHUSTR_17     "level 17: compound"
#define PHUSTR_18     "level 18: neurosphere"
#define PHUSTR_19     "level 19: nme"
#define PHUSTR_20     "level 20: the death domain"

#define PHUSTR_21     "level 21: slayer"
#define PHUSTR_22     "level 22: impossible mission"
#define PHUSTR_23     "level 23: tombstone"
#define PHUSTR_24     "level 24: the final frontier"
#define PHUSTR_25     "level 25: the temple of darkness"
#define PHUSTR_26     "level 26: bunker"
#define PHUSTR_27     "level 27: anti-christ"
#define PHUSTR_28     "level 28: the sewers"
#define PHUSTR_29     "level 29: odyssey of noises"
#define PHUSTR_30     "level 30: the gateway of hell"

#define PHUSTR_31     "level 31: cyberden"
#define PHUSTR_32     "level 32: go 2 it"

#define THUSTR_1      "level 1: system control"
#define THUSTR_2      "level 2: human bbq"
#define THUSTR_3      "level 3: power control"
#define THUSTR_4      "level 4: wormhole"
#define THUSTR_5      "level 5: hanger"
#define THUSTR_6      "level 6: open season"
#define THUSTR_7      "level 7: prison"
#define THUSTR_8      "level 8: metal"
#define THUSTR_9      "level 9: stronghold"
#define THUSTR_10     "level 10: redemption"
#define THUSTR_11     "level 11: storage facility"

```

```

#define THUSTR_12      "level 12: crater"
#define THUSTR_13      "level 13: nukage processing"
#define THUSTR_14      "level 14: steel works"
#define THUSTR_15      "level 15: dead zone"
#define THUSTR_16      "level 16: deepest reaches"
#define THUSTR_17      "level 17: processing area"
#define THUSTR_18      "level 18: mill"
#define THUSTR_19      "level 19: shipping/respawning"
#define THUSTR_20      "level 20: central processing"

#define THUSTR_21      "level 21: administration center"
#define THUSTR_22      "level 22: habitat"
#define THUSTR_23      "level 23: lunar mining project"
#define THUSTR_24      "level 24: quarry"
#define THUSTR_25      "level 25: baron's den"
#define THUSTR_26      "level 26: ballistyx"
#define THUSTR_27      "level 27: mount pain"
#define THUSTR_28      "level 28: heck"
#define THUSTR_29      "level 29: river styx"
#define THUSTR_30      "level 30: last call"

#define THUSTR_31      "level 31: pharaoh"
#define THUSTR_32      "level 32: caribbean"

#define HUSTR_CHATMACR01      "I'm ready to kick butt!"
#define HUSTR_CHATMACR02      "I'm OK."
#define HUSTR_CHATMACR03      "I'm not looking too good!"
#define HUSTR_CHATMACR04      "Help!"
#define HUSTR_CHATMACR05      "You suck!"
#define HUSTR_CHATMACR06      "Next time, scumbag..."
#define HUSTR_CHATMACR07      "Come here!"
#define HUSTR_CHATMACR08      "I'll take care of it."
#define HUSTR_CHATMACR09      "Yes"
#define HUSTR_CHATMACR00      "No"

#define HUSTR_TALKTOSELF1      "You mumble to yourself"
#define HUSTR_TALKTOSELF2      "Who's there?"
#define HUSTR_TALKTOSELF3      "You scare yourself"
#define HUSTR_TALKTOSELF4      "You start to rave"
#define HUSTR_TALKTOSELF5      "You've lost it..."

#define HUSTR_MESSAGESENT      "[Message Sent]"

// The following should NOT be changed unless it seems
// just AWFULLY necessary

#define HUSTR_PLRGREEN      "Green: "
#define HUSTR_PLRINDIGO      "Indigo: "
#define HUSTR_PLRBROWN      "Brown: "
#define HUSTR_PLRRED      "Red: "

#define HUSTR_KEYGREEN      'g'
#define HUSTR_KEYINDIGO      'i'
#define HUSTR_KEYBROWN      'b'
#define HUSTR_KEYRED      'r'

//
//      AM_map.C
//

#define AMSTR_FOLLOWON      "Follow Mode ON"
#define AMSTR_FOLLOWOFF      "Follow Mode OFF"

#define AMSTR_GRIDON      "Grid ON"

```



```

#define AMSTR_GRIDOFF          "Grid OFF"

#define AMSTR_MARKEDSPOT      "Marked Spot"
#define AMSTR_MARKSCLEARED    "All Marks Cleared"

//
//      ST_stuff.C
//

#define STSTR_MUS              "Music Change"
#define STSTR_NOMUS           "IMPOSSIBLE SELECTION"
#define STSTR_DQDON           "Degreelessness Mode On"
#define STSTR_DQDOFF          "Degreelessness Mode Off"

#define STSTR_KFAADDED         "Very Happy Ammo Added"
#define STSTR_FAADDED          "Ammo (no keys) Added"

#define STSTR_NCON             "No Clipping Mode ON"
#define STSTR_NCOFF           "No Clipping Mode OFF"

#define STSTR_BEHOLD           "inVuln, Str, Inviso, Rad, Allmap, or Lite-amp"
#define STSTR_BEHOLDX          "Power-up Toggled"

#define STSTR_CHOPPERS         "... doesn't suck - GM"
#define STSTR_CLEV             "Changing Level..."

//
//      F_Finale.C
//
#define E1TEXT \
"Once you beat the big badasses and\n"\
"clean out the moon base you're supposed\n"\
"to win, aren't you? Aren't you? Where's\n"\
"your fat reward and ticket home? What\n"\
"the hell is this? It's not supposed to\n"\
"end this way!\n"\
"\n" \
"It stinks like rotten meat, but looks\n"\
"like the lost Deimos base. Looks like\n"\
"you're stuck on The Shores of Hell.\n"\
"The only way out is through.\n"\
"\n"\
"To continue the DOOM experience, play\n"\
"The Shores of Hell and its amazing\n"\
"sequel, Inferno!\n"

#define E2TEXT \
"You've done it! The hideous cyber-\n"\
"demon lord that ruled the lost Deimos\n"\
"moon base has been slain and you\n"\
"are triumphant! But ... where are\n"\
"you? You clamber to the edge of the\n"\
"moon and look down to see the awful\n"\
"truth.\n" \
"\n"\
"Deimos floats above Hell itself!\n"\
"You've never heard of anyone escaping\n"\
"from Hell, but you'll make the bastards\n"\
"sorry they ever heard of you! Quickly,\n"\
"you rappel down to the surface of\n"\
"Hell.\n"\
"\n" \
"Now, it's on to the final chapter of\n"\
"DOOM! -- Inferno."

```

```
#define E3TEXT \
"The loathsome spiderdemon that\n" \
"masterminded the invasion of the moon\n" \
"bases and caused so much death has had\n" \
"its ass kicked for all time.\n" \
"\n" \
"A hidden doorway opens and you enter.\n" \
"You've proven too tough for Hell to\n" \
"contain, and now Hell at last plays\n" \
"fair -- for you emerge from the door\n" \
"to see the green fields of Earth!\n" \
"Home at last.\n" \
"\n" \
"You wonder what's been happening on\n" \
"Earth while you were battling evil\n" \
"unleashed. It's good that no Hell-\n" \
"spawn could have come through that\n" \
"door with you ..."
```

```
#define E4TEXT \
"the spider mastermind must have sent forth\n" \
"its legions of hellspawn before your\n" \
"final confrontation with that terrible\n" \
"beast from hell. but you stepped forward\n" \
"and brought forth eternal damnation and\n" \
"suffering upon the horde as a true hero\n" \
"would in the face of something so evil.\n" \
"\n" \
"besides, someone was gonna pay for what\n" \
"happened to daisy, your pet rabbit.\n" \
"\n" \
"but now, you see spread before you more\n" \
"potential pain and gibbitude as a nation\n" \
"of demons run amok among our cities.\n" \
"\n" \
"next stop, hell on earth!"
```

// after level 6, put this:

```
#define C1TEXT \
"YOU HAVE ENTERED DEEPLY INTO THE INFESTED\n" \
"STARPORT. BUT SOMETHING IS WRONG. THE\n" \
"MONSTERS HAVE BROUGHT THEIR OWN REALITY\n" \
"WITH THEM, AND THE STARPORT'S TECHNOLOGY\n" \
"IS BEING SUBVERTED BY THEIR PRESENCE.\n" \
"\n" \
"AHEAD, YOU SEE AN OUTPOST OF HELL, A\n" \
"FORTIFIED ZONE. IF YOU CAN GET PAST IT,\n" \
"YOU CAN PENETRATE INTO THE HAUNTED HEART\n" \
"OF THE STARBASE AND FIND THE CONTROLLING\n" \
"SWITCH WHICH HOLDS EARTH'S POPULATION\n" \
"HOSTAGE."
```

// After level 11, put this:

```
#define C2TEXT \
"YOU HAVE WON! YOUR VICTORY HAS ENABLED\n" \
"HUMANKIND TO EVACUATE EARTH AND ESCAPE\n" \
"THE NIGHTMARE. NOW YOU ARE THE ONLY\n" \
"HUMAN LEFT ON THE FACE OF THE PLANET.\n" \
"CANNIBAL MUTATIONS, CARNIVOROUS ALIENS,\n"
```

```

"AND EVIL SPIRITS ARE YOUR ONLY NEIGHBORS.\n"
"YOU SIT BACK AND WAIT FOR DEATH, CONTENT\n"
"THAT YOU HAVE SAVED YOUR SPECIES.\n"
"\n"
"BUT THEN, EARTH CONTROL BEAMS DOWN A\n"
"MESSAGE FROM SPACE: \"SENSORS HAVE LOCATED\n"
"THE SOURCE OF THE ALIEN INVASION. IF YOU\n"
"GO THERE, YOU MAY BE ABLE TO BLOCK THEIR\n"
"ENTRY. THE ALIEN BASE IS IN THE HEART OF\n"
"YOUR OWN HOME CITY, NOT FAR FROM THE\n"
"STARPORT.\" SLOWLY AND PAINFULLY YOU GET\n"
"UP AND RETURN TO THE FRAY."

```

```
// After level 20, put this:
```

```

#define C3TEXT \
"YOU ARE AT THE CORRUPT HEART OF THE CITY,\n"
"SURROUNDED BY THE CORPSES OF YOUR ENEMIES.\n"
"YOU SEE NO WAY TO DESTROY THE CREATURES'\n"
"ENTRYWAY ON THIS SIDE, SO YOU CLENCH YOUR\n"
"TEETH AND PLUNGE THROUGH IT.\n"
"\n"
"THERE MUST BE A WAY TO CLOSE IT ON THE\n"
"OTHER SIDE. WHAT DO YOU CARE IF YOU'VE\n"
"GOT TO GO THROUGH HELL TO GET TO IT?"

```

```
// After level 29, put this:
```

```

#define C4TEXT \
"THE HORRENDOUS VISAGE OF THE BIGGEST\n"
"DEMON YOU'VE EVER SEEN CRUMBLES BEFORE\n"
"YOU, AFTER YOU PUMP YOUR ROCKETS INTO\n"
"HIS EXPOSED BRAIN. THE MONSTER SHRIVELS\n"
"UP AND DIES, ITS THRASHING LIMBS\n"
"DEVASTATING UNTOLD MILES OF HELL'S\n"
"SURFACE.\n"
"\n"
"YOU'VE DONE IT. THE INVASION IS OVER.\n"
"EARTH IS SAVED. HELL IS A WRECK. YOU\n"
"WONDER WHERE BAD FOLKS WILL GO WHEN THEY\n"
"DIE, NOW. WIPING THE SWEAT FROM YOUR\n"
"FOREHEAD YOU BEGIN THE LONG TREK BACK\n"
"HOME. REBUILDING EARTH OUGHT TO BE A\n"
"LOT MORE FUN THAN RUINING IT WAS.\n"

```

```
// Before level 31, put this:
```

```

#define C5TEXT \
"CONGRATULATIONS, YOU'VE FOUND THE SECRET\n"
"LEVEL! LOOKS LIKE IT'S BEEN BUILT BY\n"
"HUMANS, RATHER THAN DEMONS. YOU WONDER\n"
"WHO THE INMATES OF THIS CORNER OF HELL\n"
"WILL BE."

```

```
// Before level 32, put this:
```

```

#define C6TEXT \
"CONGRATULATIONS, YOU'VE FOUND THE\n"
"SUPER SECRET LEVEL! YOU'D BETTER\n"
"BLAZE THROUGH THIS ONE!\n"

```

// after map 06

```
#define P1TEXT \  
"You gloat over the steaming carcass of the\n\  
"Guardian. With its death, you've wrested\n\  
"the Accelerator from the stinking claws\n\  
"of Hell. You relax and glance around the\n\  
"room. Damn! There was supposed to be at\n\  
"least one working prototype, but you can't\n\  
"see it. The demons must have taken it.\n\  
"\n\  
"You must find the prototype, or all your\n\  
"struggles will have been wasted. Keep\n\  
"moving, keep fighting, keep killing.\n\  
"Oh yes, keep living, too."
```

// after map 11

```
#define P2TEXT \  
"Even the deadly Arch-Vile labyrinth could\n\  
"not stop you, and you've gotten to the\n\  
"prototype Accelerator which is soon\n\  
"efficiently and permanently deactivated.\n\  
"\n\  
"You're good at that kind of thing."
```

// after map 20

```
#define P3TEXT \  
"You've bashed and battered your way into\n\  
"the heart of the devil-hive. Time for a\n\  
"Search-and-Destroy mission, aimed at the\n\  
"Gatekeeper, whose foul offspring is\n\  
"cascading to Earth. Yeah, he's bad. But\n\  
"you know who's worse!\n\  
"\n\  
"Grinning evilly, you check your gear, and\n\  
"get ready to give the bastard a little Hell\n\  
"of your own making!"
```

// after map 30

```
#define P4TEXT \  
"The Gatekeeper's evil face is splattered\n\  
"all over the place. As its tattered corpse\n\  
"collapses, an inverted Gate forms and\n\  
"sucks down the shards of the last\n\  
"prototype Accelerator, not to mention the\n\  
"few remaining demons. You're done. Hell\n\  
"has gone back to pounding bad dead folks\n\  
"instead of good live ones. Remember to\n\  
"tell your grandkids to put a rocket\n\  
"launcher in your coffin. If you go to Hell\n\  
"when you die, you'll need it for some\n\  
"final cleaning-up ..."
```

// before map 31

```
#define P5TEXT \  
"You've found the second-hardest level we\n\  
"got. Hope you have a saved game a level or\n\  
"
```

"two previous. If not, be prepared to die\n"\n"aplenty. For master marines only."

// before map 32

```
#define P6TEXT \  
"Betcha wondered just what WAS the hardest\n\  
"level we had ready for ya? Now you know.\n\  
"No one gets out alive."
```

```
#define T1TEXT \  
"You've fought your way out of the infested\n\  
"experimental labs. It seems that UAC has\n\  
"once again gulped it down. With their\n\  
"high turnover, it must be hard for poor\n\  
"old UAC to buy corporate health insurance\n\  
"nowadays..\n\  
"\n\  
"Ahead lies the military complex, now\n\  
"swarming with diseased horrors hot to get\n\  
"their teeth into you. With luck, the\n\  
"complex still has some warlike ordnance\n\  
"laying around."
```

```
#define T2TEXT \  
"You hear the grinding of heavy machinery\n\  
"ahead. You sure hope they're not stamping\n\  
"out new hellspawn, but you're ready to\n\  
"ream out a whole herd if you have to.\n\  
"They might be planning a blood feast, but\n\  
"you feel about as mean as two thousand\n\  
"maniacs packed into one mad killer.\n\  
"\n\  
"You don't plan to go down easy."
```

```
#define T3TEXT \  
"The vista opening ahead looks real damn\n\  
"familiar. Smells familiar, too -- like\n\  
"fried excrement. You didn't like this\n\  
"place before, and you sure as hell ain't\n\  
"planning to like it now. The more you\n\  
"brood on it, the madder you get.\n\  
"Hefting your gun, an evil grin trickles\n\  
"onto your face. Time to take some names."
```

```
#define T4TEXT \  
"Suddenly, all is silent, from one horizon\n\  
"to the other. The agonizing echo of Hell\n\  
"fades away, the nightmare sky turns to\n\  
"blue, the heaps of monster corpses start\n\  
"to evaporate along with the evil stench\n\  
"that filled the air. Jeeze, maybe you've\n\  
"done it. Have you really won?\n\  
"\n\  
"Something rumbles in the distance.\n\  
"A blue light begins to glow inside the\n\  
"ruined skull of the demon-spitter."
```

```
#define T5TEXT \  
"What now? Looks totally different. Kind\n\  
"of like King Tut's condo. Well,\n"
```

```
"whatever's here can't be any worse\n"\
"than usual. Can it? Or maybe it's best\n"\
"to let sleeping gods lie.."
```

```
#define T6TEXT \
"Time for a vacation. You've burst the\n"\
"bowels of hell and by golly you're ready\n"\
"for a break. You mutter to yourself,\n"\
"Maybe someone else can kick Hell's ass\n"\
"next time around. Ahead lies a quiet town,\n"\
"with peaceful flowing water, quaint\n"\
"buildings, and presumably no Hellspawn.\n"\
"\n"\
"As you step off the transport, you hear\n"\
"the stomp of a cyberdemon's iron shoe."
```

```
//
// Character cast strings F_FINALE.C
//
#define CC_ZOMBIE      "ZOMBIEMAN"
#define CC_SHOTGUN     "SHOTGUN GUY"
#define CC_HEAVY       "HEAVY WEAPON DUDE"
#define CC_IMP         "IMP"
#define CC_DEMON       "DEMON"
#define CC_LOST        "LOST SOUL"
#define CC_CACO        "CACODEMON"
#define CC_HELL        "HELL KNIGHT"
#define CC_BARON       "BARON OF HELL"
#define CC_ARACH       "ARACHNOTRON"
#define CC_PAIN        "PAIN ELEMENTAL"
#define CC_REVEN       "REVENANT"
#define CC_MANCU       "MANCUBUS"
#define CC_ARCH        "ARCH-VILE"
#define CC_SPIDER      "THE SPIDER MASTERMIND"
#define CC_CYBER       "THE CYBERDEMON"
#define CC_HERO        "OUR HERO"
```

```
#endif
```

```
//-----
//
// $Log:$
//
//-----
```

## 3.2 d\_event.h

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```

// GNU General Public License for more details.
//
// DESCRIPTION:
//
//-----

#ifndef __D_EVENT__
#define __D_EVENT__

#include "doomtype.h"

//
// Event handling.
//

// Input event types.
typedef enum
{
    ev_keydown,
    ev_keyup,
    ev_mouse,
    ev_joystick
} evtype_t;

// Event structure.
typedef struct
{
    evtype_t      type;
    int           data1;           // keys / mouse/joystick buttons
    int           data2;           // mouse/joystick x move
    int           data3;           // mouse/joystick y move
} event_t;

typedef enum
{
    ga_nothing,
    ga_loadlevel,
    ga_newgame,
    ga_loadgame,
    ga_savegame,
    ga_playdemo,
    ga_completed,
    ga_victory,
    ga_worlddone,
    ga_screenshot
} gameaction_t;

//
// Button/action code definitions.
//
typedef enum
{
    // Press "Fire".
    BT_ATTACK          = 1,
    // Use button, to open doors, activate switches.
    BT_USE              = 2,

    // Flag: game events, not really buttons.

```

```

BT_SPECIAL            = 128,
BT_SPECIALMASK        = 3,

// Flag, weapon change pending.
// If true, the next 3 bits hold weapon num.
BT_CHANGE            = 4,
// The 3bit weapon mask and shift, convenience.
BT_WEAPONMASK        = (8+16+32),
BT_WEAPONSHIFT       = 3,

// Pause the game.
BTS_PAUSE            = 1,
// Save the game at each console.
BTS_SAVEGAME         = 2,

// Savegame slot numbers
// occupy the second byte of buttons.
BTS_SAVEMASK         = (4+8+16),
BTS_SAVESHIFT        = 2,

} buttoncode_t;

```

```

//
// GLOBAL VARIABLES
//
#define MAXEVENTS      64

extern event_t          events[MAXEVENTS];
extern int              eventhead;
extern int              eventtail;

extern gameaction_t     gameaction;

#endif
//-----
//
// $Log:$
//
//-----

```

### 3.3 d\_french.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      Printed strings, french translation.

```



```

//
//-----

#ifndef __D_FRENCH__
#define __D_FRENCH__

//
// D_Main.C
//
#define D_DEVSTR      "MODE DEVELOPPEMENT ON.\n"
#define D_CDROM       "VERSION CD-ROM: DEFAULT.CFG DANS C:\\DOOMDATA\\n"

//
//      M_Menu.C
//
#define PRESSKEY      "APPUYEZ SUR UNE TOUCHE."
#define PRESSYN       "APPUYEZ SUR Y OU N"
#define QUITMSG        "VOUS VOULEZ VRAIMENT\nQUITTER CE SUPER JEU?"
#define LOADNET        "VOUS NE POUVEZ PAS CHARGER\nUN JEU EN RESEAU!\n\n"PRESSKEY
#define QLOADNET       "CHARGEMENT RAPIDE INTERDIT EN RESEAU!\n\n"PRESSKEY
#define QSAVESPOT       "VOUS N'AVEZ PAS CHOISI UN EMPLACEMENT!\n\n"PRESSKEY
#define SAVEDEAD        "VOUS NE POUVEZ PAS SAUVER SI VOUS NE JOUEZ "\
"PAS!\n\n"PRESSKEY
#define QSPROMPT        "SAUVEGARDE RAPIDE DANS LE FICHIER \n\n'%s'?\n\n"PRESSYN
#define QLPROMPT        "VOULEZ-VOUS CHARGER LA SAUVEGARDE"\
"\n\n'%s'?\n\n"PRESSYN
#define NEWGAME         "VOUS NE POUVEZ PAS LANCER\n\n"\
"UN NOUVEAU JEU SUR RESEAU.\n\n"PRESSKEY
#define NIGHTMARE       "VOUS CONFIRMEZ? CE NIVEAU EST\n\n"\
"VRAIMENT IMPITOYABLE!\n"PRESSYN
#define SWSTRING        "CECI EST UNE VERSION SHAREWARE DE DOOM.\n\n"\
"VOUS DEVRIEZ COMMANDER LA TRILOGIE COMPLETE.\n\n"PRESSKEY
#define MSGOFF          "MESSAGES OFF"
#define MSGON           "MESSAGES ON"
#define NETEND          "VOUS NE POUVEZ PAS METTRE FIN A UN JEU SUR "\
"RESEAU!\n\n"PRESSKEY
#define ENDGAME         "VOUS VOULEZ VRAIMENT METTRE FIN AU JEU?\n\n"PRESSYN

#define DOSY            "(APPUYEZ SUR Y POUR REVENIR AU OS.)"

#define DETAILHI        "GRAPHISMES MAXIMUM "
#define DETAILLO        "GRAPHISMES MINIMUM "
#define GAMMALVLO       "CORRECTION GAMMA OFF"
#define GAMMALVL1       "CORRECTION GAMMA NIVEAU 1"
#define GAMMALVL2       "CORRECTION GAMMA NIVEAU 2"
#define GAMMALVL3       "CORRECTION GAMMA NIVEAU 3"
#define GAMMALVL4       "CORRECTION GAMMA NIVEAU 4"
#define EMPTYSTRING     "EMPLACEMENT VIDE"

//
//      P_inter.C
//
#define GOTARMOR         "ARMURE RECUPEREE."
#define GOTMEGA          "MEGA-ARMURE RECUPEREE!"
#define GOTTHBONUS       "BONUS DE SANTE RECUPERE."
#define GOTARMBONUS      "BONUS D'ARMURE RECUPERE."
#define GOTSTIM          "STIMPACK RECUPERE."
#define GOTMEDINEED       "MEDIKIT RECUPERE. VOUS EN AVEZ VRAIMENT BESOIN!"
#define GOTMEDIKIT        "MEDIKIT RECUPERE."
#define GOTSUPER         "SUPERCHARGE!"

#define GOTBLUECARD      "CARTE MAGNETIQUE BLEUE RECUPEREE."
#define GOTYELWCARD      "CARTE MAGNETIQUE JAUNE RECUPEREE."
#define GOTREDCARD       "CARTE MAGNETIQUE ROUGE RECUPEREE."

```

```

#define GOTBLUESKUL      "CLEF CRANE BLEUE RECUPEREE."
#define GOTYELWSKUL      "CLEF CRANE JAUNE RECUPEREE."
#define GOTREDSKULL      "CLEF CRANE ROUGE RECUPEREE."

#define GOTINVUL          "INVULNERABILITE!"
#define GOTBERSERK        "BERSERK!"
#define GOTINVIS          "INVISIBILITE PARTIELLE "
#define GOTSUIT           "COMBINAISON ANTI-RADIATIONS "
#define GOTMAP            "CARTE INFORMATIQUE "
#define GOTVISOR          "VISEUR A AMPLIFICATION DE LUMIERE "
#define GOTMSPHERE        "MEGASPHERE!"

#define GOTCLIP           "CHARGEUR RECUPERE."
#define GOTCLIPBOX        "BOITE DE BALLES RECUPEREE."
#define GOTROCKET         "ROQUETTE RECUPEREE."
#define GOTROCKBOX        "CAISSE DE ROQUETTES RECUPEREE."
#define GOTCELL           "CELLULE D'ENERGIE RECUPEREE."
#define GOTCELLBOX        "PACK DE CELLULES D'ENERGIE RECUPERE."
#define GOTSHELLS         "4 CARTOUCHES RECUPEREES."
#define GOTSHELLBOX       "BOITE DE CARTOUCHES RECUPEREE."
#define GOTBACKPACK       "SAC PLEIN DE MUNITIONS RECUPERE!"

#define GOTBFG9000        "VOUS AVEZ UN BFG9000! OH, OUI!"
#define GOTCHAINGUN       "VOUS AVEZ LA MITRAILLEUSE!"
#define GOTCHAINSAW       "UNE TRONCONNEUSE!"
#define GOTLAUNCHER       "VOUS AVEZ UN LANCE-ROQUETTES!"
#define GOTPLASMA         "VOUS AVEZ UN FUSIL A PLASMA!"
#define GOTSHOTGUN        "VOUS AVEZ UN FUSIL!"
#define GOTSHOTGUN2       "VOUS AVEZ UN SUPER FUSIL!"

//
// P_Doors.C
//
#define PD_BLUEO          "IL VOUS FAUT UNE CLEF BLEUE"
#define PD_REDO           "IL VOUS FAUT UNE CLEF ROUGE"
#define PD_YELLOWO        "IL VOUS FAUT UNE CLEF JAUNE"
#define PD_BLUEK          PD_BLUEO
#define PD_REDK           PD_REDO
#define PD_YELLOWK        PD_YELLOWO

//
// G_game.C
//
#define GGSAVED           "JEU SAUVEGARDE."

//
// HU_stuff.C
//
#define HUSTR_MSGU        "[MESSAGE NON ENVOYE]"

#define HUSTR_E1M1        "E1M1: HANGAR"
#define HUSTR_E1M2        "E1M2: USINE NUCLEAIRE "
#define HUSTR_E1M3        "E1M3: RAFFINERIE DE TOXINES "
#define HUSTR_E1M4        "E1M4: CENTRE DE CONTROLE "
#define HUSTR_E1M5        "E1M5: LABORATOIRE PHOBOS "
#define HUSTR_E1M6        "E1M6: TRAITEMENT CENTRAL "
#define HUSTR_E1M7        "E1M7: CENTRE INFORMATIQUE "
#define HUSTR_E1M8        "E1M8: ANOMALIE PHOBOS "
#define HUSTR_E1M9        "E1M9: BASE MILITAIRE "

#define HUSTR_E2M1        "E2M1: ANOMALIE DEIMOS "
#define HUSTR_E2M2        "E2M2: ZONE DE CONFINEMENT "
#define HUSTR_E2M3        "E2M3: RAFFINERIE"
#define HUSTR_E2M4        "E2M4: LABORATOIRE DEIMOS "
#define HUSTR_E2M5        "E2M5: CENTRE DE CONTROLE "

```

```

#define HISTR_E2M6      "E2M6: HALLS DES DAMNES "
#define HISTR_E2M7      "E2M7: CUVES DE REPRODUCTION "
#define HISTR_E2M8      "E2M8: TOUR DE BABEL "
#define HISTR_E2M9      "E2M9: FORTERESSE DU MYSTERE "

#define HISTR_E3M1      "E3M1: DONJON DE L'ENFER "
#define HISTR_E3M2      "E3M2: BOURBIER DU DESEPOIR "
#define HISTR_E3M3      "E3M3: PANDEMONIUM"
#define HISTR_E3M4      "E3M4: MAISON DE LA DOULEUR "
#define HISTR_E3M5      "E3M5: CATHEDRALE PROFANE "
#define HISTR_E3M6      "E3M6: MONT EREBUS"
#define HISTR_E3M7      "E3M7: LIMBES"
#define HISTR_E3M8      "E3M8: DIS"
#define HISTR_E3M9      "E3M9: CLAPIERS"

#define HISTR_1         "NIVEAU 1: ENTREE "
#define HISTR_2         "NIVEAU 2: HALLS SOUTERRAINS "
#define HISTR_3         "NIVEAU 3: LE FEU NOURRI "
#define HISTR_4         "NIVEAU 4: LE FOYER "
#define HISTR_5         "NIVEAU 5: LES EGOUTS "
#define HISTR_6         "NIVEAU 6: LE BROYEUR "
#define HISTR_7         "NIVEAU 7: L'HERBE DE LA MORT"
#define HISTR_8         "NIVEAU 8: RUSES ET PIEGES "
#define HISTR_9         "NIVEAU 9: LE Puits "
#define HISTR_10        "NIVEAU 10: BASE DE RAVITAILLEMENT "
#define HISTR_11        "NIVEAU 11: LE CERCLE DE LA MORT!"

#define HISTR_12        "NIVEAU 12: L'USINE "
#define HISTR_13        "NIVEAU 13: LE CENTRE VILLE"
#define HISTR_14        "NIVEAU 14: LES ANTRES PROFONDES "
#define HISTR_15        "NIVEAU 15: LA ZONE INDUSTRIELLE "
#define HISTR_16        "NIVEAU 16: LA BANLIEUE"
#define HISTR_17        "NIVEAU 17: LES IMMEUBLES"
#define HISTR_18        "NIVEAU 18: LA COUR "
#define HISTR_19        "NIVEAU 19: LA CITADELLE "
#define HISTR_20        "NIVEAU 20: JE T'AI EU!"

#define HISTR_21        "NIVEAU 21: LE NIRVANA"
#define HISTR_22        "NIVEAU 22: LES CATACOMBES "
#define HISTR_23        "NIVEAU 23: LA GRANDE FETE "
#define HISTR_24        "NIVEAU 24: LE GOUFFRE "
#define HISTR_25        "NIVEAU 25: LES CHUTES DE SANG"
#define HISTR_26        "NIVEAU 26: LES MINES ABANDONNEES "
#define HISTR_27        "NIVEAU 27: CHEZ LES MONSTRES "
#define HISTR_28        "NIVEAU 28: LE MONDE DE L'ESPRIT "
#define HISTR_29        "NIVEAU 29: LA LIMITE "
#define HISTR_30        "NIVEAU 30: L'ICONE DU PECHE "

#define HISTR_31        "NIVEAU 31: WOLFENSTEIN"
#define HISTR_32        "NIVEAU 32: LE MASSACRE"

#define HISTR_CHATMACRO1 "JE SUIS PRET A LEUR EN FAIRE BAVER!"
#define HISTR_CHATMACRO2 "JE VAIS BIEN."
#define HISTR_CHATMACRO3 "JE N'AI PAS L'AIR EN FORME!"
#define HISTR_CHATMACRO4 "AU SECOURS!"
#define HISTR_CHATMACRO5 "TU CRAINS!"
#define HISTR_CHATMACRO6 "LA PROCHAINE FOIS, MINABLE..."
#define HISTR_CHATMACRO7 "VIENS ICI!"
#define HISTR_CHATMACRO8 "JE VAIS M'EN OCCUPER."
#define HISTR_CHATMACRO9 "OUI"
#define HISTR_CHATMACRO0 "NON"

#define HISTR_TALKTOSELF1 "VOUS PARLEZ TOUT SEUL "
#define HISTR_TALKTOSELF2 "QUI EST LA?"

```

```

#define HUSTR_TALKTOSELF3      "VOUS VOUS FAITES PEUR "
#define HUSTR_TALKTOSELF4      "VOUS COMMENCEZ A DELIRER "
#define HUSTR_TALKTOSELF5      "VOUS ETES LARGUE..."

#define HUSTR_MESSAGESENT      "[MESSAGE ENVOYE]"

// The following should NOT be changed unless it seems
// just AWFULLY necessary

#define HUSTR_PLRGREEN          "VERT: "
#define HUSTR_PLRINDIGO         "INDIGO: "
#define HUSTR_PLRBROWN         "BRUN: "
#define HUSTR_PLRRED           "ROUGE: "

#define HUSTR_KEYGREEN          'g'          // french key should be "V"
#define HUSTR_KEYINDIGO         'i'
#define HUSTR_KEYBROWN         'b'
#define HUSTR_KEYRED            'r'

//
//      AM_map.C
//

#define AMSTR_FOLLOWON          "MODE POURSUITE ON"
#define AMSTR_FOLLOWOFF        "MODE POURSUITE OFF"

#define AMSTR_GRIDON           "GRILLE ON"
#define AMSTR_GRIDOFF          "GRILLE OFF"

#define AMSTR_MARKEDSPOT        "REPERE MARQUE "
#define AMSTR_MARKSCLEARED      "REPERES EFFACES "

//
//      ST_stuff.C
//

#define STSTR_MUS                "CHANGEMENT DE MUSIQUE "
#define STSTR_NOMUS             "IMPOSSIBLE SELECTION"
#define STSTR_DQDON             "INVULNERABILITE ON "
#define STSTR_DQDOFF            "INVULNERABILITE OFF"

#define STSTR_KFAADDED           "ARMEMENT MAXIMUM! "
#define STSTR_FAADDED           "ARMES (SAUF CLEFS) AJOUTEES"

#define STSTR_NCON              "BARRIERES ON"
#define STSTR_NCOFF             "BARRIERES OFF"

#define STSTR_BEHOLD            " inVuln, Str, Inviso, Rad, Allmap, or Lite-amp"
#define STSTR_BEHOLDX           "AMELIORATION ACTIVEE"

#define STSTR_CHOPPERS           "... DOESN'T SUCK - GM"
#define STSTR_CLEV              "CHANGEMENT DE NIVEAU..."

//
//      F_Finale.C
//

#define E1TEXT                  "APRES AVOIR VAINCU LES GROS MECHANTS\n"\
"ET NETTOYE LA BASE LUNAIRE, VOUS AVEZ\n"\
"GAGNE, NON? PAS VRAI? OU EST DONC VOTRE\n"\
" RECOMPENSE ET VOTRE BILLET DE\n"\
"RETOUR? QU'EST-QUE CA VEUT DIRE?CE"\
"N'EST PAS LA FIN ESPEREE!\n"\
"\n" \
"CA SENT LA VIANDE PUTREFIEE, MAIS\n"\
"ON DIRAIT LA BASE DEIMOS. VOUS ETES\n"

```

```

"APPAREMMENT BLOQUE AUX PORTES DE L'ENFER.\n" \
"LA SEULE ISSUE EST DE L'AUTRE COTE.\n" \
"\n" \
"POUR VIVRE LA SUITE DE DOOM, JOUEZ\n" \
"A 'AUX PORTES DE L'ENFER' ET A\n" \
"L'EPISODE SUIVANT, 'L'ENFER'!\n"

#define E2TEXT      "VOUS AVEZ REUSSI. L'INFAME DEMON\n" \
"QUI CONTROLAIT LA BASE LUNAIRE DE\n" \
"DEIMOS EST MORT, ET VOUS AVEZ\n" \
"TRIOMPHE! MAIS... OU ETES-VOUS?\n" \
"VOUS GRIMPEZ JUSQU'AU BORD DE LA\n" \
"LUNE ET VOUS DECOUVREZ L'ATROCE\n" \
"VERITE.\n" \
"\n" \
"DEIMOS EST AU-DESSUS DE L'ENFER!\n" \
"VOUS SAVEZ QUE PERSONNE NE S'EN\n" \
"EST JAMAIS ECHAPPE, MAIS CES FUMIERS\n" \
"VONT REGRETTER DE VOUS AVOIR CONNU!\n" \
"VOUS REDESCENDEZ RAPIDEMENT VERS\n" \
"LA SURFACE DE L'ENFER.\n" \
"\n" \
"VOICI MAINTENANT LE CHAPITRE FINAL DE\n" \
"DOOM! -- L'ENFER."

#define E3TEXT      "LE DEMON ARACHNEEN ET REPUGNANT\n" \
"QUI A DIRIGE L'INVASION DES BASES\n" \
"LUNAIRES ET SEME LA MORT VIENT DE SE\n" \
"FAIRE PULVERISER UNE FOIS POUR TOUTES.\n" \
"\n" \
"UNE PORTE SECRETE S'OUVRE. VOUS ENTREZ.\n" \
"VOUS AVEZ PROUVE QUE VOUS POUVIEZ\n" \
"RESISTER AUX HORREURS DE L'ENFER.\n" \
"IL SAIT ETRE BEAU JOUEUR, ET LORSQUE\n" \
"VOUS SORTEZ, VOUS REVOYEZ LES VERTES\n" \
"PRAIRIES DE LA TERRE, VOTRE PLANETE.\n" \
"\n" \
"VOUS VOUS DEMANDEZ CE QUI S'EST PASSE\n" \
"SUR TERRE PENDANT QUE VOUS AVEZ\n" \
"COMBATTU LE DEMON. HEUREUSEMENT,\n" \
"AUCUN GERME DU MAL N'A FRANCHI\n" \
"CETTE PORTE AVEC VOUS..."

// after level 6, put this:

#define C1TEXT      "VOUS ETES AU PLUS PROFOND DE L'ASTROPORT\n" \
"INFESTE DE MONSTRES, MAIS QUELQUE CHOSE\n" \
"NE VA PAS. ILS ONT APPORTE LEUR PROPRE\n" \
"REALITE, ET LA TECHNOLOGIE DE L'ASTROPORT\n" \
"EST AFFECTEE PAR LEUR PRESENCE.\n" \
"\n" \
"DEVANT VOUS, VOUS VOYEZ UN POSTE AVANCE\n" \
"DE L'ENFER, UNE ZONE FORTIFIEE. SI VOUS\n" \
"POUVEZ PASSER, VOUS POURREZ PENETRER AU\n" \
"COEUR DE LA BASE HANTEE ET TROUVER\n" \
"L'INTERRUPTEUR DE CONTROLE QUI GARDE LA\n" \
"POPULATION DE LA TERRE EN OTAGE."

// After level 11, put this:

#define C2TEXT      "VOUS AVEZ GAGNE! VOTRE VICTOIRE A PERMIS\n" \
"A L'HUMANITE D'EVACUER LA TERRE ET\n" \
"D'ECHAPPER AU CAUCHEMAR. VOUS ETES\n"

```

```
"MAINTENANT LE DERNIER HUMAIN A LA SURFACE \n"\  
"DE LA PLANETE. VOUS ETES ENTOURE DE \n"\  
"MUTANTS CANNIBALES, D'EXTRATERRESTRES \n"\  
"CARNIVORES ET D'ESPRITS DU MAL. VOUS \n"\  
"ATTENDEZ CALMEMENT LA MORT, HEUREUX \n"\  
"D'AVOIR PU SAUVER VOTRE RACE.\n"\  
"MAIS UN MESSAGE VOUS PARVIENT SOUDAIN\n"\  
"DE L'ESPACE: \"NOS CAPTEURS ONT LOCALISE\n"\  
"LA SOURCE DE L'INVASION EXTRATERRESTRE.\n"\  
"SI VOUS Y ALLEZ, VOUS POURREZ PEUT-ETRE\n"\  
"LES ARRETER. LEUR BASE EST SITUEE AU COEUR\n"\  
"DE VOTRE VILLE NATALE, PRES DE L'ASTROPORT.\n"\  
"VOUS VOUS RELEVEZ LENTEMENT ET PENIBLEMENT\n"\  
"ET VOUS REPARTEZ POUR LE FRONT."
```

```
// After level 20, put this:
```

```
#define C3TEXT      "VOUS ETES AU COEUR DE LA CITE CORROMPUE,\n"\  
"ENTOURE PAR LES CADAVRES DE VOS ENNEMIS.\n"\  
"VOUS NE VOYEZ PAS COMMENT DETRUIRE LA PORTE\n"\  
"DES CREATURES DE CE COTE. VOUS SERREZ\n"\  
"LES DENTS ET PLONGEZ DANS L'OUVERTURE.\n"\  
"\n"\  
"IL DOIT Y AVOIR UN MOYEN DE LA FERMER\n"\  
"DE L'AUTRE COTE. VOUS ACCEPTEZ DE\n"\  
"TRAVERSER L'ENFER POUR LE FAIRE?"
```

```
// After level 29, put this:
```

```
#define C4TEXT      "LE VISAGE HORRIBLE D'UN DEMON D'UNE\n"\  
"TAILLE INCROYABLE S'EFFONDRE DEVANT\n"\  
"VOUS LORSQUE VOUS TIREZ UNE SALVE DE\n"\  
"ROQUETTES DANS SON CERVEAU. LE MONSTRE\n"\  
"SE RATATINE, SES MEMBRES DECHIQUETES\n"\  
"SE REPANDANT SUR DES CENTAINES DE\n"\  
"KILOMETRES A LA SURFACE DE L'ENFER.\n"\  
"\n"\  
"VOUS AVEZ REUSSI. L'INVASION N'AURA.\n"\  
"PAS LIEU. LA TERRE EST SAUVEE. L'ENFER\n"\  
"EST ANEANTI. EN VOUS DEMANDANT OU Iront\n"\  
"MAINTENANT LES DAMNES, VOUS ESSUYEZ\n"\  
"VOTRE FRONT COUVERT DE SUEUR ET REPARTEZ\n"\  
"VERS LA TERRE. SA RECONSTRUCTION SERA\n"\  
"BEAUCOUP PLUS DROLE QUE SA DESTRUCTION.\n"
```

```
// Before level 31, put this:
```

```
#define C5TEXT      "FELICITATIONS! VOUS AVEZ TROUVE LE\n"\  
"NIVEAU SECRET! IL SEMBLE AVOIR ETE\n"\  
"CONSTRUIT PAR LES HUMAINS. VOUS VOUS\n"\  
"DEMANDEZ QUELS PEUVENT ETRE LES\n"\  
"HABITANTS DE CE COIN PERDU DE L'ENFER."
```

```
// Before level 32, put this:
```

```
#define C6TEXT      "FELICITATIONS! VOUS AVEZ DECOUVERT\n"\  
"LE NIVEAU SUPER SECRET! VOUS FERIEZ\n"\  
"MIEUX DE FONCER DANS CELUI-LA!\n"
```

```
//
```

```
// Character cast strings F_FINALE.C
```

```
//
```

```
#define CC_ZOMBIE      "ZOMBIE"  
#define CC_SHOTGUN     "TYPE AU FUSIL"  
#define CC_HEAVY       "MEC SUPER-ARME"
```

```

#define CC_IMP                "DIABLOTIN"
#define CC_DEMON              "DEMON"
#define CC_LOST               "AME PERDUE"
#define CC_CACO               "CACODEMON"
#define CC_HELL               "CHEVALIER DE L'ENFER"
#define CC_BARON              "BARON DE L'ENFER"
#define CC_ARACH              "ARACHNOTRON"
#define CC_PAIN               "ELEMENTAIRE DE LA DOULEUR"
#define CC_REVEN              "REVENANT"
#define CC_MANCU              "MANCUBUS"
#define CC_ARCH               "ARCHI-INFAME"
#define CC_SPIDER             "L'ARAGNEE CERVEAU"
#define CC_CYBER              "LE CYBERDEMON"
#define CC_HERO               "NOTRE HEROS"

#endif
//-----
//
// $Log:$
//
//-----

```

### 3.4 d\_items.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//-----

static const char
rcsid[] = "$Id:$";

// We are referring to sprite numbers.
#include "info.h"

#ifdef __GNUG__
#pragma implementation "d_items.h"
#endif
#include "d_items.h"

//
// PSPRITE ACTIONS for waepons.

```

```

// This struct controls the weapon animations.
//
// Each entry is:
//   ammo/amunition type
//   upstate
//   downstate
//   readystate
//   atkstate, i.e. attack/fire/hit frame
//   flashstate, muzzle flash
//
weaponinfo_t      weaponinfo[NUMWEAPONS] =
{
    {
        // fist
        am_noammo,
        S_PUNCHUP,
        S_PUNCHDOWN,
        S_PUNCH,
        S_PUNCH1,
        S_NULL
    },
    {
        // pistol
        am_clip,
        S_PISTOLUP,
        S_PISTOLDOWN,
        S_PISTOL,
        S_PISTOL1,
        S_PISTOLFLASH
    },
    {
        // shotgun
        am_shell,
        S_SGUNUP,
        S_SGUNDOWN,
        S_SGUN,
        S_SGUN1,
        S_SGUNFLASH1
    },
    {
        // chaingun
        am_clip,
        S_CHAINUP,
        S_CHAINDOWN,
        S_CHAIN,
        S_CHAIN1,
        S_CHAINFLASH1
    },
    {
        // missile launcher
        am_misl,
        S_MISSILEUP,
        S_MISSILEDOWN,
        S_MISSILE,
        S_MISSILE1,
        S_MISSILEFLASH1
    },
    {
        // plasma rifle
        am_cell,
        S_PLASMAUP,
        S_PLASMADOWN,
        S_PLASMA,
        S_PLASMA1,
        S_PLASMAFLASH1
    }
}

```



```

    },
    {
        // bfg 9000
        am_cell,
        S_BFGUP,
        S_BFGDOWN,
        S_BFG,
        S_BFG1,
        S_BFGFLASH1
    },
    {
        // chainsaw
        am_noammo,
        S_SAWUP,
        S_SAWDOWN,
        S_SAW,
        S_SAW1,
        S_NULL
    },
    {
        // super shotgun
        am_shell,
        S_DSGUNUP,
        S_DSGUNDOWN,
        S_DSGUN,
        S_DSGUN1,
        S_DSGUNFLASH1
    },
};
};

```

### 3.5 d\_items.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Items: key cards, artifacts, weapon, ammunition.
//-----

#ifdef __D_ITEMS__
#define __D_ITEMS__

```

```

#include "doomdef.h"

#ifdef __GNUG__
#pragma interface
#endif

// Weapon info: sprite frames, ammunition use.
typedef struct
{
    ammotype_t    ammo;
    int           upstate;
    int           downstate;
    int           readystate;
    int           atkstate;
    int           flashstate;

} weaponinfo_t;

extern weaponinfo_t    weaponinfo[NUMWEAPONS];

#endif
//-----
//
// $Log:$
//
//-----

```

### 3.6 d\_main.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      DOOM main program (D_DoomMain) and game loop (D_DoomLoop),
//      plus functions to determine game mode (shareware, registered),
//      parse command line parameters, configure game parameters (turbo),
//      and call the startup functions.
//
//-----

static const char rcsid[] = "$Id: d_main.c,v 1.8 1997/02/03 22:45:09 b1 Exp $";

#define      BGCOLOR      7
#define      FGCOLOR      8

```

```

#ifdef NORMALUNIX
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#endif

#include "doomdef.h"
#include "doomstat.h"

#include "dstrings.h"
#include "sounds.h"

#include "z_zone.h"
#include "w_wad.h"
#include "s_sound.h"
#include "v_video.h"

#include "f_finale.h"
#include "f_wipe.h"

#include "m_argv.h"
#include "m_misc.h"
#include "m_menu.h"

#include "i_system.h"
#include "i_sound.h"
#include "i_video.h"

#include "g_game.h"

#include "hu_stuff.h"
#include "wi_stuff.h"
#include "st_stuff.h"
#include "am_map.h"

#include "p_setup.h"
#include "r_local.h"

#include "d_main.h"

//
// D-DoomLoop()
// Not a globally visible function,
// just included for source reference,
// called by D_DoomMain, never exits.
// Manages timing and IO,
// calls all ?_Responder, ?_Ticker, and ?_Drawer,
// calls I_GetTime, I_StartFrame, and I_StartTic
//
void D_DoomLoop (void);

char*                wadfiles[MAXWADFILES];

boolean              devparm;           // started game with -devparm
boolean              nomonsters;        // checkparm of -nomonsters
boolean              respawnparm;      // checkparm of -respawn
boolean              fastparm;         // checkparm of -fast

```

```

boolean            drone;

boolean            singletics = false; // debug flag to cancel adaptiveness


//extern int soundVolume;
//extern int      sfxVolume;
//extern int      musicVolume;

extern boolean     inhelppcreens;

skill_t           startskill;
int               startepisode;
int               startmap;
boolean           autostart;

FILE*             debugfile;

boolean           advancedemo;


char              wadfile[1024];           // primary wad file
char              mapdir[1024];           // directory of development maps
char              basedefault[1024];      // default file


void D_CheckNetGame (void);
void D_ProcessEvents (void);
void G_BuildTiccmd (ticcmd_t* cmd);
void D_DoAdvanceDemo (void);


//
// EVENT HANDLING
//
// Events are asynchronous inputs generally generated by the game user.
// Events can be discarded if no responder claims them
//
event_t           events[MAXEVENTS];
int               eventhead;
int               eventtail;


//
// D_PostEvent
// Called by the I/O functions when input is detected
//
void D_PostEvent (event_t* ev)
{
    events[eventhead] = *ev;
    eventhead = (++eventhead)&(MAXEVENTS-1);
}


//
// D_ProcessEvents
// Send all the events of the given timestamp down the responder chain
//
void D_ProcessEvents (void)
{
    event_t*       ev;

```

```

// IF STORE DEMO, DO NOT ACCEPT INPUT
if ( ( gamemode == commercial )
    && (W_CheckNumForName("map01")<0) )
    return;

for ( ; eventtail != eventhead ; eventtail = (++eventtail)&(MAXEVENTS-1) )
{
    ev = &events[eventtail];
    if (M_Responder (ev))
        continue;                // menu ate the event
    G_Responder (ev);
}
}

//
// D_Display
// draw current display, possibly wiping it from the previous
//

// wipegamestate can be set to -1 to force a wipe on the next draw
gamestate_t      wipegamestate = GS_DEMOSCREEN;
extern boolean setsizeneeded;
extern int        showMessages;
void R_ExecuteSetViewSize (void);

void D_Display (void)
{
    static boolean      viewactivestate = false;
    static boolean      menuactivestate = false;
    static boolean      inhelpscreensstate = false;
    static boolean      fullscreen = false;
    static gamestate_t  oldgamestate = -1;
    static int          borderdrawcount;
    int                 nowtime;
    int                 tics;
    int                 wipestart;
    int                 y;
    boolean              done;
    boolean              wipe;
    boolean              redrawsbar;

    if (nodrawers)
        return;                // for comparative timing / profiling

    redrawsbar = false;

    // change the view size if needed
    if (setsizeneeded)
    {
        R_ExecuteSetViewSize ();
        oldgamestate = -1;                // force background redraw
        borderdrawcount = 3;
    }

    // save the current screen if about to wipe
    if (gamestate != wipegamestate)
    {
        wipe = true;
        wipe_StartScreen(0, 0, SCREENWIDTH, SCREENHEIGHT);
    }
    else

```

```

        wipe = false;

if (gamestate == GS_LEVEL && gametic)
    HU_Erase();

// do buffered drawing
switch (gamestate)
{
    case GS_LEVEL:
        if (!gametic)
            break;
        if (automapactive)
            AM_Drawer ();
        if (wipe || (viewheight != 200 && fullscreen) )
            redrawsbar = true;
        if (inhelptcreensstate && !inhelptcreens)
            redrawsbar = true;          // just put away the help screen
        ST_Drawer (viewheight == 200, redrawsbar );
        fullscreen = viewheight == 200;
        break;

    case GS_INTERMISSION:
        WI_Drawer ();
        break;

    case GS_FINALE:
        F_Drawer ();
        break;

    case GS_DEMOSCREEN:
        D_PageDrawer ();
        break;
}

// draw buffered stuff to screen
I_UpdateNoBlit ();

// draw the view directly
if (gamestate == GS_LEVEL && !automapactive && gametic)
    R_RenderPlayerView (&players[displayplayer]);

if (gamestate == GS_LEVEL && gametic)
    HU_Drawer ();

// clean up border stuff
if (gamestate != oldgamestate && gamestate != GS_LEVEL)
    I_SetPalette (W_CacheLumpName ("PLAYPAL",PU_CACHE));

// see if the border needs to be initially drawn
if (gamestate == GS_LEVEL && oldgamestate != GS_LEVEL)
{
    viewactivestate = false;          // view was not active
    R_FillBackScreen ();              // draw the pattern into the back screen
}

// see if the border needs to be updated to the screen
if (gamestate == GS_LEVEL && !automapactive && scaledviewwidth != 320)
{
    if (menuactive || menuactivestate || !viewactivestate)
        borderdrawcount = 3;
    if (borderdrawcount)
    {
        R_DrawViewBorder ();          // erase old menu stuff
        borderdrawcount--;
    }
}

```

```

}

menuactivestate = menuactive;
viewactivestate = viewactive;
inhelptcreensstate = inhelptcreens;
oldgamestate = wipegamestate = gamestate;

// draw pause pic
if (paused)
{
    if (automapactive)
        y = 4;
    else
        y = viewwindowy+4;
    V_DrawPatchDirect(viewwindowx+(scaledviewwidth-68)/2,
                      y,0,W_CacheLumpName ("M_PAUSE", PU_CACHE));
}

// menus go directly to the screen
M_Drawer ();          // menu is drawn even on top of everything
NetUpdate ();          // send out any new accumulation

// normal update
if (!wipe)
{
    I_FinishUpdate ();          // page flip or blit buffer
    return;
}

// wipe update
wipe_EndScreen(0, 0, SCREENWIDTH, SCREENHEIGHT);

wipestart = I_GetTime () - 1;

do
{
    do
    {
        nowtime = I_GetTime ();
        tics = nowtime - wipestart;
    } while (!tics);
    wipestart = nowtime;
    done = wipe_ScreenWipe(wipe_Melt
                          , 0, 0, SCREENWIDTH, SCREENHEIGHT, tics);

    I_UpdateNoBlit ();
    M_Drawer ();          // menu is drawn even on top of wipes
    I_FinishUpdate ();    // page flip or blit buffer
} while (!done);
}

//
// D_DoomLoop
//
extern boolean          demorecording;

void D_DoomLoop (void)
{
    if (demorecording)
        G_BeginRecording ();
}

```

```

if (M_CheckParm ("-debugfile"))
{
    char    filename[20];
    sprintf (filename,"debug%i.txt",consoleplayer);
    printf ("debug output to: %s\n",filename);
    debugfile = fopen (filename,"w");
}

I_InitGraphics ();

while (1)
{
    // frame synchronous IO operations
    I_StartFrame ();

    // process one or more tics
    if (singletics)
    {
        I_StartTic ();
        D_ProcessEvents ();
        G_BuildTiccmd (&netcmds[consoleplayer][maketic%BACKUPTICS]);
        if (advancedemo)
            D_DoAdvanceDemo ();
        M_Ticker ();
        G_Ticker ();
        gametic++;
        maketic++;
    }
    else
    {
        TryRunTics (); // will run at least one tic
    }

    S_UpdateSounds (players[consoleplayer].mo); // move positional sounds

    // Update display, next frame, with current state.
    D_Display ();

#ifdef SNDSESV
    // Sound mixing for the buffer is synchronous.
    I_UpdateSound();
#endif
    // Synchronous sound output is explicitly called.
#ifdef SNDINTR
    // Update sound output.
    I_SubmitSound();
#endif
    }
}

//
// DEMO LOOP
//
int          demosequence;
int          pagetic;
char         *pagename;

//
// D_PageTicker
// Handles timing for warped projection
//
void D_PageTicker (void)

```



```

{
    if (--pagetic < 0)
        D_AdvanceDemo ();
}

//
// D_PageDrawer
//
void D_PageDrawer (void)
{
    V_DrawPatch (0,0, 0, W_CacheLumpName(pagename, PU_CACHE));
}

//
// D_AdvanceDemo
// Called after each demo or intro demosequence finishes
//
void D_AdvanceDemo (void)
{
    advancedemo = true;
}

//
// This cycles through the demo sequences.
// FIXME - version dependend demo numbers?
//
void D_DoAdvanceDemo (void)
{
    players[consoleplayer].playerstate = PST_LIVE; // not reborn
    advancedemo = false;
    usergame = false; // no save / end game here
    paused = false;
    gameaction = ga_nothing;

    if ( gamemode == retail )
        demosequence = (demosequence+1)%7;
    else
        demosequence = (demosequence+1)%6;

    switch (demosequence)
    {
    case 0:
        if ( gamemode == commercial )
            pagetic = 35 * 11;
        else
            pagetic = 170;
        gamestate = GS_DEMOSCREEN;
        pagename = "TITLEPIC";
        if ( gamemode == commercial )
            S_StartMusic(mus_dm2ttl);
        else
            S_StartMusic (mus_intro);
        break;
    case 1:
        G_DeferedPlayDemo ("demo1");
        break;
    case 2:
        pagetic = 200;
        gamestate = GS_DEMOSCREEN;
        pagename = "CREDIT";
        break;
    }
}

```

```

case 3:
    G_DeferredPlayDemo ("demo2");
    break;
case 4:
    gamestate = GS_DEMOSCREEN;
    if ( gamemode == commercial )
    {
        pagetic = 35 * 11;
        pagename = "TITLEPIC";
        S_StartMusic(mus_dm2ttl);
    }
    else
    {
        pagetic = 200;

        if ( gamemode == retail )
            pagename = "CREDIT";
        else
            pagename = "HELP2";
    }
    break;
case 5:
    G_DeferredPlayDemo ("demo3");
    break;
    // THE DEFINITIVE DOOM Special Edition demo
case 6:
    G_DeferredPlayDemo ("demo4");
    break;
}
}

```

```

//
// D_StartTitle
//
void D_StartTitle (void)
{
    gameaction = ga_nothing;
    demosequence = -1;
    D_AdvanceDemo ();
}

```

```

//      print title for every printed line
char      title[128];

```

```

//
// D_AddFile
//
void D_AddFile (char *file)
{
    int      numwadfiles;
    char      *newfile;

    for (numwadfiles = 0 ; wadfiles[numwadfiles] ; numwadfiles++)
        ;

    newfile = malloc (strlen(file)+1);
    strcpy (newfile, file);
}

```

```

    wadfiles[numwadfiles] = newfile;
}

//
// IdentifyVersion
// Checks availability of IWAD files by name,
// to determine whether registered/commercial features
// should be executed (notably loading PWAD's).
//
void IdentifyVersion (void)
{
    char*      doom1wad;
    char*      doomwad;
    char*      doomuwad;
    char*      doom2wad;

    char*      doom2fwad;
    char*      plutoniawad;
    char*      tntwad;

#ifdef NORMALUNIX
    char *home;
    char *doomwaddir;
    doomwaddir = getenv("DOOMWADDIR");
    if (!doomwaddir)
        doomwaddir = ".";

    // Commercial.
    doom2wad = malloc(strlen(doomwaddir)+1+9+1);
    sprintf(doom2wad, "%s/doom2.wad", doomwaddir);

    // Retail.
    doomuwad = malloc(strlen(doomwaddir)+1+8+1);
    sprintf(doomuwad, "%s/doomu.wad", doomwaddir);

    // Registered.
    doomwad = malloc(strlen(doomwaddir)+1+8+1);
    sprintf(doomwad, "%s/doom.wad", doomwaddir);

    // Shareware.
    doom1wad = malloc(strlen(doomwaddir)+1+9+1);
    sprintf(doom1wad, "%s/doom1.wad", doomwaddir);

    // Bug, dear Shawn.
    // Insufficient malloc, caused spurious realloc errors.
    plutoniawad = malloc(strlen(doomwaddir)+1+*9*/12+1);
    sprintf(plutoniawad, "%s/plutonia.wad", doomwaddir);

    tntwad = malloc(strlen(doomwaddir)+1+9+1);
    sprintf(tntwad, "%s/tnt.wad", doomwaddir);

    // French stuff.
    doom2fwad = malloc(strlen(doomwaddir)+1+10+1);
    sprintf(doom2fwad, "%s/doom2f.wad", doomwaddir);

    home = getenv("HOME");
    if (!home)
        I_Error("Please set $HOME to your home directory");
    sprintf(basedefault, "%s/.doomrc", home);
#endif

    if (M_CheckParm ("-shdev"))
    {

```

```

    gamemode = shareware;
    devparm = true;
    D_AddFile (DEVDATA"doom1.wad");
    D_AddFile (DEVMAPS"data_se/texture1.lmp");
    D_AddFile (DEVMAPS"data_se/pnames.lmp");
    strcpy (basedefault,DEVDATA"default.cfg");
    return;
}

if (M_CheckParm ("-regdev"))
{
    gamemode = registered;
    devparm = true;
    D_AddFile (DEVDATA"doom.wad");
    D_AddFile (DEVMAPS"data_se/texture1.lmp");
    D_AddFile (DEVMAPS"data_se/texture2.lmp");
    D_AddFile (DEVMAPS"data_se/pnames.lmp");
    strcpy (basedefault,DEVDATA"default.cfg");
    return;
}

if (M_CheckParm ("-comdev"))
{
    gamemode = commercial;
    devparm = true;
    /* I don't bother
    if(plutonia)
        D_AddFile (DEVDATA"plutonia.wad");
    else if(tnt)
        D_AddFile (DEVDATA"tnt.wad");
    else*/
        D_AddFile (DEVDATA"doom2.wad");

    D_AddFile (DEVMAPS"cdata/texture1.lmp");
    D_AddFile (DEVMAPS"cdata/pnames.lmp");
    strcpy (basedefault,DEVDATA"default.cfg");
    return;
}

if ( !access (doom2fwad,R_OK) )
{
    gamemode = commercial;
    // C'est ridicule!
    // Let's handle languages in config files, okay?
    language = french;
    printf("French version\n");
    D_AddFile (doom2fwad);
    return;
}

if ( !access (doom2wad,R_OK) )
{
    gamemode = commercial;
    D_AddFile (doom2wad);
    return;
}

if ( !access (plutoniawad, R_OK ) )
{
    gamemode = commercial;
    D_AddFile (plutoniawad);
    return;
}

if ( !access ( tntwad, R_OK ) )

```

```

{
    gamemode = commercial;
    D_AddFile (tntwad);
    return;
}

if ( !access (doomuwad,R_OK) )
{
    gamemode = retail;
    D_AddFile (doomuwad);
    return;
}

if ( !access (doomwad,R_OK) )
{
    gamemode = registered;
    D_AddFile (doomwad);
    return;
}

if ( !access (doom1wad,R_OK) )
{
    gamemode = shareware;
    D_AddFile (doom1wad);
    return;
}

printf("Game mode indeterminate.\n");
gamemode = indetermined;

// We don't abort. Let's see what the PWAD contains.
//exit(1);
//I_Error ("Game mode indeterminate\n");
}

//
// Find a Response File
//
void FindResponseFile (void)
{
    int i;
#define MAXARGVS 100

    for (i = 1; i < myargc; i++)
        if (myargv[i][0] == '@')
        {
            FILE * handle;
            int size;
            int k;
            int index;
            int indexinfile;
            char *infile;
            char *file;
            char *moreargs[20];
            char *firstargv;

            // READ THE RESPONSE FILE INTO MEMORY
            handle = fopen (&myargv[i][1], "rb");
            if (!handle)
            {
                printf ("\nNo such response file!");
                exit(1);
            }
            printf("Found response file %s!\n", &myargv[i][1]);
            fseek (handle, 0, SEEK_END);

```

```

size = ftell(handle);
fseek (handle,0,SEEK_SET);
file = malloc (size);
fread (file,size,1,handle);
fclose (handle);

// KEEP ALL CMDLINE ARGS FOLLOWING @RESPONSEFILE ARG
for (index = 0,k = i+1; k < myargc; k++)
    moreargs[index++] = myargv[k];

firstargv = myargv[0];
myargv = malloc(sizeof(char *)*MAXARGS);
memset(myargv,0,sizeof(char *)*MAXARGS);
myargv[0] = firstargv;

infile = file;
indexinfile = k = 0;
indexinfile++; // SKIP PAST ARGV[0] (KEEP IT)
do
{
    myargv[indexinfile++] = infile+k;
    while(k < size &&
        ((* (infile+k)>= ' ' +1) && ((* (infile+k)<='z'))))
        k++;
    * (infile+k) = 0;
    while(k < size &&
        ((* (infile+k)<= ' ') || ((* (infile+k)>'z'))))
        k++;
} while(k < size);

for (k = 0;k < index;k++)
    myargv[indexinfile++] = moreargs[k];
myargc = indexinfile;

// DISPLAY ARGS
printf("%d command-line args:\n",myargc);
for (k=1;k<myargc;k++)
    printf("%s\n",myargv[k]);

break;
}

}

//
// D_DoomMain
//
void D_DoomMain (void)
{
    int            p;
    char            file[256];

    FindResponseFile ();

    IdentifyVersion ();

    setbuf (stdout, NULL);
    modifiedgame = false;

    nomonsters = M_CheckParm ("-nomonsters");
    respawnparm = M_CheckParm ("-respawn");
    fastparm = M_CheckParm ("-fast");
    devparm = M_CheckParm ("-devparm");
    if (M_CheckParm ("-altdeath"))
        deathmatch = 2;

```

```

else if (M_CheckParm ("-deathmatch"))
    deathmatch = 1;

switch ( gamemode )
{
    case retail:
        sprintf (title,
            "
                "The Ultimate DOOM Startup v%i.%i"
            "
                ",
            VERSION/100,VERSION%100);
        break;
    case shareware:
        sprintf (title,
            "
                "DOOM Shareware Startup v%i.%i"
            "
                ",
            VERSION/100,VERSION%100);
        break;
    case registered:
        sprintf (title,
            "
                "DOOM Registered Startup v%i.%i"
            "
                ",
            VERSION/100,VERSION%100);
        break;
    case commercial:
        sprintf (title,
            "
                "DOOM 2: Hell on Earth v%i.%i"
            "
                ",
            VERSION/100,VERSION%100);
        break;
/*FIXME
    case pack_plut:
        sprintf (title,
            "
                "DOOM 2: Plutonia Experiment v%i.%i"
            "
                ",
            VERSION/100,VERSION%100);
        break;
    case pack_tnt:
        sprintf (title,
            "
                "DOOM 2: TNT - Evilution v%i.%i"
            "
                ",
            VERSION/100,VERSION%100);
        break;
*/
    default:
        sprintf (title,
            "
                "Public DOOM - v%i.%i"
            "
                ",
            VERSION/100,VERSION%100);
        break;
}

printf ("%s\n",title);

if (devparm)
    printf(D_DEVSTR);

if (M_CheckParm("-cdrom"))
{

```

```

    printf(D_CDROM);
    mkdir("c:\\doomdata",0);
    strcpy (basedefault,"c:/doomdata/default.cfg");
}

// turbo option
if ( (p=M_CheckParm ("-turbo")) )
{
    int      scale = 200;
    extern int forwardmove[2];
    extern int sidemove[2];

    if (p<myargc-1)
        scale = atoi (myargv[p+1]);
    if (scale < 10)
        scale = 10;
    if (scale > 400)
        scale = 400;
    printf ("turbo scale: %i%%\n",scale);
    forwardmove[0] = forwardmove[0]*scale/100;
    forwardmove[1] = forwardmove[1]*scale/100;
    sidemove[0] = sidemove[0]*scale/100;
    sidemove[1] = sidemove[1]*scale/100;
}

// add any files specified on the command line with -file wadfile
// to the wad list
//
// convenience hack to allow -wart e m to add a wad file
// prepend a tilde to the filename so wadfile will be reloadable
p = M_CheckParm ("-wart");
if (p)
{
    myargv[p][4] = 'p';      // big hack, change to -warp

    // Map name handling.
    switch (gamemode )
    {
        case shareware:
        case retail:
        case registered:
            sprintf (file,"~"DEVMAPS"E%cM%c.wad",
                    myargv[p+1][0], myargv[p+2][0]);
            printf("Warping to Episode %s, Map %s.\n",
                    myargv[p+1],myargv[p+2]);
            break;

        case commercial:
        default:
            p = atoi (myargv[p+1]);
            if (p<10)
                sprintf (file,"~"DEVMAPS"cdata/map0%i.wad", p);
            else
                sprintf (file,"~"DEVMAPS"cdata/map%i.wad", p);
            break;
    }
    D_AddFile (file);
}

p = M_CheckParm ("-file");
if (p)
{
    // the parms after p are wadfile/lump names,
    // until end of parms or another - preceded parm
    modifiedgame = true;      // homebrew levels

```



```

        while (++p != myargc && myargv[p][0] != '-')
            D_AddFile (myargv[p]);
    }

    p = M_CheckParm ("-playdemo");

    if (!p)
        p = M_CheckParm ("-timedemo");

    if (p && p < myargc-1)
    {
        sprintf (file,"%s.lmp", myargv[p+1]);
        D_AddFile (file);
        printf("Playing demo %s.lmp.\n",myargv[p+1]);
    }

    // get skill / episode / map from parms
    startskill = sk_medium;
    startepisode = 1;
    startmap = 1;
    autostart = false;

    p = M_CheckParm ("-skill");
    if (p && p < myargc-1)
    {
        startskill = myargv[p+1][0]-'1';
        autostart = true;
    }

    p = M_CheckParm ("-episode");
    if (p && p < myargc-1)
    {
        startepisode = myargv[p+1][0]-'0';
        startmap = 1;
        autostart = true;
    }

    p = M_CheckParm ("-timer");
    if (p && p < myargc-1 && deathmatch)
    {
        int    time;
        time = atoi(myargv[p+1]);
        printf("Levels will end after %d minute",time);
        if (time>1)
            printf("s");
        printf(".\n");
    }

    p = M_CheckParm ("-avg");
    if (p && p < myargc-1 && deathmatch)
        printf("Austin Virtual Gaming: Levels will end after 20 minutes\n");

    p = M_CheckParm ("-warp");
    if (p && p < myargc-1)
    {
        if (gamemode == commercial)
            startmap = atoi (myargv[p+1]);
        else
        {
            startepisode = myargv[p+1][0]-'0';
            startmap = myargv[p+2][0]-'0';
        }
        autostart = true;
    }
}

```

```

// init subsystems
printf ("V_Init: allocate screens.\n");
V_Init ();

printf ("M_LoadDefaults: Load system defaults.\n");
M_LoadDefaults ();          // load before initing other systems

printf ("Z_Init: Init zone memory allocation daemon. \n");
Z_Init ();

printf ("W_Init: Init WADfiles.\n");
W_InitMultipleFiles (wadfiles);

// Check for -file in shareware
if (modifiedgame)
{
    // These are the lumps that will be checked in IWAD,
    // if any one is not present, execution will be aborted.
    char name[23][8]=
    {
        "e2m1","e2m2","e2m3","e2m4","e2m5","e2m6","e2m7","e2m8","e2m9",
        "e3m1","e3m3","e3m3","e3m4","e3m5","e3m6","e3m7","e3m8","e3m9",
        "dphoof","bfgga0","heada1","cybra1","spida1d1"
    };
    int i;

    if ( gamemode == shareware)
        I_Error("\nYou cannot -file with the shareware "
            "version. Register!");

    // Check for fake IWAD with right name,
    // but w/o all the lumps of the registered version.
    if (gamemode == registered)
        for (i = 0;i < 23; i++)
            if (W_CheckNumForName(name[i])<0)
                I_Error("\nThis is not the registered version.");
}

// Iff additional PWAD files are used, print modified banner
if (modifiedgame)
{
    /*m*/printf (
        "=====\n"
        "ATTENTION: This version of DOOM has been modified. If you would like to\n"
        "get a copy of the original game, call 1-800-IDGAMES or see the readme file.\n"
        "    You will not receive technical support for modified games.\n"
        "                press enter to continue\n"
        "=====\n"
    );
    getchar ();
}

// Check and print which version is executed.
switch ( gamemode )
{
case shareware:
case indetermined:
    printf (
        "=====\n"
        "                Shareware!\n"
        "=====\n"
    );
};

```

```

        break;
    case registered:
    case retail:
    case commercial:
        printf (
            "=====\\n"
            "          Commercial product - do not distribute!\\n"
            "          Please report software piracy to the SPA: 1-800-388-PIR8\\n"
            "=====\\n"
        );
        break;

    default:
        // Ouch.
        break;
}

printf ("M_Init: Init miscellaneous info.\\n");
M_Init ();

printf ("R_Init: Init DOOM refresh daemon - ");
R_Init ();

printf ("\\nP_Init: Init Playloop state.\\n");
P_Init ();

printf ("I_Init: Setting up machine state.\\n");
I_Init ();

printf ("D_CheckNetGame: Checking network game status.\\n");
D_CheckNetGame ();

printf ("S_Init: Setting up sound.\\n");
S_Init (snd_SfxVolume /* *8 */, snd_MusicVolume /* *8*/* );

printf ("HU_Init: Setting up heads up display.\\n");
HU_Init ();

printf ("ST_Init: Init status bar.\\n");
ST_Init ();

// check for a driver that wants intermission stats
p = M_CheckParm ("-statcopy");
if (p && p<myargc-1)
{
    // for statistics driver
    extern void*      statcopy;

    statcopy = (void*)atoi(myargv[p+1]);
    printf ("External statistics registered.\\n");
}

// start the appropriate game based on parms
p = M_CheckParm ("-record");

if (p && p < myargc-1)
{
    G_RecordDemo (myargv[p+1]);
    autostart = true;
}

p = M_CheckParm ("-playdemo");
if (p && p < myargc-1)
{
    singledemo = true;           // quit after one demo

```

```

        G_DeferedPlayDemo (myargv[p+1]);
        D_DoomLoop (); // never returns
    }

    p = M_CheckParm ("-timedemo");
    if (p && p < myargc-1)
    {
        G_TimeDemo (myargv[p+1]);
        D_DoomLoop (); // never returns
    }

    p = M_CheckParm ("-loadgame");
    if (p && p < myargc-1)
    {
        if (M_CheckParm("-cdrom"))
            sprintf(file, "c:\\doomdata\\"SAVEGAMENAME"%c.dsg",myargv[p+1][0]);
        else
            sprintf(file, SAVEGAMENAME"%c.dsg",myargv[p+1][0]);
        G_LoadGame (file);
    }

    if ( gameaction != ga_loadgame )
    {
        if (autostart || netgame)
            G_InitNew (startskill, startepisode, startmap);
        else
            D_StartTitle (); // start up intro loop
    }

    D_DoomLoop (); // never returns
}

```

### 3.7 d\_main.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     System specific interface stuff.
//
//-----

#ifndef __D_MAIN__
#define __D_MAIN__

#include "d_event.h"

```

```

#ifdef __GNUG__
#pragma interface
#endif

#define MAXWADFILES          20
extern char*                 wadfiles[MAXWADFILES];

void D_AddFile (char *file);

//
// D_DoomMain()
// Not a globally visible function, just included for source reference,
// calls all startup code, parses command line options.
// If not overridden by user input, calls N_AdvanceDemo.
//
void D_DoomMain (void);

// Called by IO functions when input is detected.
void D_PostEvent (event_t* ev);

//
// BASE LEVEL
//
void D_PageTicker (void);
void D_PageDrawer (void);
void D_AdvanceDemo (void);
void D_StartTitle (void);

#endif

```

### 3.8 d\_net.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      DOOM Network game communication and protocol,
//      all OS independend parts.
//
//-----

```

```

static const char rcsid[] = "$Id: d_net.c,v 1.3 1997/02/03 22:01:47 b1 Exp $";

#include "m_menu.h"
#include "i_system.h"
#include "i_video.h"
#include "i_net.h"
#include "g_game.h"
#include "doomdef.h"
#include "doomstat.h"

#define NCMD_EXIT 0x80000000
#define NCMD_RETRANSMIT 0x40000000
#define NCMD_SETUP 0x20000000
#define NCMD_KILL 0x10000000 // kill game
#define NCMD_CHECKSUM 0x0fffffff

doomcom_t* doomcom;
doomdata_t* netbuffer; // points inside doomcom

//
// NETWORKING
//
// gametic is the tic about to (or currently being) run
// maketic is the tick that hasn't had control made for it yet
// nettics[] has the maketics for all players
//
// a gametic cannot be run until nettics[] > gametic for all players
//
#define RESENDCOUNT 10
#define PL_DRONE 0x80 // bit flag in doomdata->player

ticcmd_t localcmds[BACKUPTICS];

ticcmd_t netcmds[MAXPLAYERS][BACKUPTICS];
int nettics[MAXNETNODES];
boolean nodeingame[MAXNETNODES]; // set false as nodes leave game
boolean remoteresend[MAXNETNODES]; // set when local needs tics
int resendto[MAXNETNODES]; // set when remote needs tics
int resendcount[MAXNETNODES];

int nodeforplayer[MAXPLAYERS];

int maketic;
int lastnettictic;
int skiptics;
int ticdup;
int maxsend; // BACKUPTICS/(2*ticdup)-1

void D_ProcessEvents (void);
void G_BuildTiccmd (ticcmd_t *cmd);
void D_DoAdvanceDemo (void);

boolean reboundpacket;
doomdata_t reboundstore;

//
//
//
int NetbufferSize (void)

```

```

{
    return (int)&(((doomdata_t *)0)->cmds[netbuffer->numtics]);
}

//
// Checksum
//
unsigned NetbufferChecksum (void)
{
    unsigned          c;
    int               i,l;

    c = 0x1234567;

    // FIXME -endianess?
#ifdef NORMALUNIX
    return 0;
    // byte order problems
#endif

    l = (NetbufferSize () - (int)&(((doomdata_t *)0)->retransmitfrom))/4;
    for (i=0 ; i<l ; i++)
        c += ((unsigned *)&netbuffer->retransmitfrom)[i] * (i+1);

    return c & NCMD_CHECKSUM;
}

//
//
//
int ExpandTics (int low)
{
    int          delta;

    delta = low - (maketic&0xff);

    if (delta >= -64 && delta <= 64)
        return (maketic&~0xff) + low;
    if (delta > 64)
        return (maketic&~0xff) - 256 + low;
    if (delta < -64)
        return (maketic&~0xff) + 256 + low;

    I_Error ("ExpandTics: strange value %i at maketic %i",low,maketic);
    return 0;
}

//
// HSendPacket
//
void
HSendPacket
(int          node,
 int          flags )
{
    netbuffer->checksum = NetbufferChecksum () | flags;

    if (!node)
    {
        reboundstore = *netbuffer;
        reboundpacket = true;
        return;
    }
}

```

```

if (demoplayback)
    return;

if (!netgame)
    I_Error ("Tried to transmit to another node");

doomcom->command = CMD_SEND;
doomcom->remotenode = node;
doomcom->datalength = NetbufferSize ();

if (debugfile)
{
    int i;
    int realretrans;
    if (netbuffer->checksum & NCMD_RETRANSMIT)
        realretrans = ExpandTics (netbuffer->retransmitfrom);
    else
        realretrans = -1;

    fprintf (debugfile,"send (%i + %i, R %i) [%i] ",
            ExpandTics(netbuffer->starttic),
            netbuffer->numtics, realretrans, doomcom->datalength);

    for (i=0 ; i<doomcom->datalength ; i++)
        fprintf (debugfile,"%i ",((byte *)netbuffer)[i]);

    fprintf (debugfile,"\n");
}

I_NetCmd ();
}

//
// HGetPacket
// Returns false if no packet is waiting
//
boolean HGetPacket (void)
{
    if (reboundpacket)
    {
        *netbuffer = reboundstore;
        doomcom->remotenode = 0;
        reboundpacket = false;
        return true;
    }

    if (!netgame)
        return false;

    if (demoplayback)
        return false;

    doomcom->command = CMD_GET;
    I_NetCmd ();

    if (doomcom->remotenode == -1)
        return false;

    if (doomcom->datalength != NetbufferSize ())
    {
        if (debugfile)
            fprintf (debugfile,"bad packet length %i\n",doomcom->datalength);
        return false;
    }
}

```



```

if (NetbufferChecksum () != (netbuffer->checksum&NCMD_CHECKSUM) )
{
    if (debugfile)
        fprintf (debugfile,"bad packet checksum\n");
    return false;
}

if (debugfile)
{
    int                realretrans;
    int                i;

    if (netbuffer->checksum & NCMD_SETUP)
        fprintf (debugfile,"setup packet\n");
    else
    {
        if (netbuffer->checksum & NCMD_RETRANSMIT)
            realretrans = ExpandTics (netbuffer->retransmitfrom);
        else
            realretrans = -1;

        fprintf (debugfile,"get %i = (%i + %i, R %i)[%i] ",
            doomcom->remotenode,
            ExpandTics(netbuffer->starttic),
            netbuffer->numtics, realretrans, doomcom->datalength);

        for (i=0 ; i<doomcom->datalength ; i++)
            fprintf (debugfile,"%i ",((byte *)netbuffer)[i]);
        fprintf (debugfile,"\n");
    }
}
return true;
}

```

```

//
// GetPackets
//
char    exitmsg[80];

void GetPackets (void)
{
    int                netconsole;
    int                netnode;
    ticcmd_t          *src, *dest;
    int                realend;
    int                realstart;

    while ( HGetPacket() )
    {
        if (netbuffer->checksum & NCMD_SETUP)
            continue;                // extra setup packet

        netconsole = netbuffer->player & ~PL_DRONE;
        netnode = doomcom->remotenode;

        // to save bytes, only the low byte of tic numbers are sent
        // Figure out what the rest of the bytes are
        realstart = ExpandTics (netbuffer->starttic);
        realend = (realstart+netbuffer->numtics);

        // check for exiting the game
        if (netbuffer->checksum & NCMD_EXIT)
        {
            if (!nodeingame[netnode])

```

```

        continue;
nodeingame[netnode] = false;
playeringame[netconsole] = false;
strcpy (exitmsg, "Player 1 left the game");
exitmsg[7] += netconsole;
players[consoleplayer].message = exitmsg;
if (demorecording)
    G_CheckDemoStatus ();
continue;
}

// check for a remote game kill
if (netbuffer->checksum & NCMD_KILL)
    I_Error ("Killed by network driver");

nodeforplayer[netconsole] = netnode;

// check for retransmit request
if ( resendcount[netnode] <= 0
    && (netbuffer->checksum & NCMD_RETRANSMIT) )
{
    resendto[netnode] = ExpandTics(netbuffer->retransmitfrom);
    if (debugfile)
        fprintf (debugfile,"retransmit from %i\n", resendto[netnode]);
    resendcount[netnode] = RESEND_COUNT;
}
else
    resendcount[netnode]--;

// check for out of order / duplicated packet
if (realend == netticks[netnode])
    continue;

if (realend < netticks[netnode])
{
    if (debugfile)
        fprintf (debugfile,
            "out of order packet (%i + %i)\n" ,
            realstart, netbuffer->numtics);
    continue;
}

// check for a missed packet
if (realstart > netticks[netnode])
{
    // stop processing until the other system resends the missed tics
    if (debugfile)
        fprintf (debugfile,
            "missed tics from %i (%i - %i)\n",
            netnode, realstart, netticks[netnode]);
    remoteresend[netnode] = true;
    continue;
}

// update command store from the packet
{
    int                start;

    remoteresend[netnode] = false;

    start = netticks[netnode] - realstart;
    src = &netbuffer->cmds[start];

    while (netticks[netnode] < realend)
    {

```

```

        dest = &netcmds[netconsole][nettics[netnode]%BACKUPTICS];
        nettics[netnode]++;
        *dest = *src;
        src++;
    }
}

//
// NetUpdate
// Builds ticcmds for console player,
// sends out a packet
//
int      gametime;

void NetUpdate (void)
{
    int      nowtime;
    int      newtics;
    int      i,j;
    int      realstart;
    int      gameticdiv;

    // check time
    nowtime = I_GetTime ()/ticdup;
    newtics = nowtime - gametime;
    gametime = nowtime;

    if (newtics <= 0)          // nothing new to update
        goto listen;

    if (skiptics <= newtics)
    {
        newtics -= skiptics;
        skiptics = 0;
    }
    else
    {
        skiptics -= newtics;
        newtics = 0;
    }

    netbuffer->player = consoleplayer;

    // build new ticcmds for console player
    gameticdiv = gametic/ticdup;
    for (i=0 ; i<newtics ; i++)
    {
        I_StartTic ();
        D_ProcessEvents ();
        if (maketic - gameticdiv >= BACKUPTICS/2-1)
            break;          // can't hold any more

        //printf ("mk:%i ",maketic);
        G_BuildTiccmd (&localcmds[maketic%BACKUPTICS]);
        maketic++;
    }

    if (singletics)
        return;          // singletic update is synchronous

```

```

// send the packet to the other nodes
for (i=0 ; i<doomcom->numnodes ; i++)
    if (nodeingame[i])
    {
        netbuffer->starttic = realstart = resendto[i];
        netbuffer->numtics = maketic - realstart;
        if (netbuffer->numtics > BACKUPTICS)
            I_Error ("NetUpdate: netbuffer->numtics > BACKUPTICS");

        resendto[i] = maketic - doomcom->extratics;

        for (j=0 ; j< netbuffer->numtics ; j++)
            netbuffer->cmds[j] =
                localcmds[(realstart+j)%BACKUPTICS];

        if (remotereresend[i])
        {
            netbuffer->retransmitfrom = nettics[i];
            HSendPacket (i, NCMD_RETRANSMIT);
        }
        else
        {
            netbuffer->retransmitfrom = 0;
            HSendPacket (i, 0);
        }
    }

// listen for other packets
listen:
    GetPackets ();
}

//
// CheckAbort
//
void CheckAbort (void)
{
    event_t *ev;
    int          stoptic;

    stoptic = I_GetTime () + 2;
    while (I_GetTime() < stoptic)
        I_StartTic ();

    I_StartTic ();
    for ( ; eventtail != eventhead
        ; eventtail = (++eventtail)&(MAXEVENTS-1) )
    {
        ev = &events[eventtail];
        if (ev->type == ev_keydown && ev->data1 == KEY_ESCAPE)
            I_Error ("Network game synchronization aborted.");
    }
}

//
// D_ArbitrateNetStart
//
void D_ArbitrateNetStart (void)
{
    int          i;
    boolean      gotinfo[MAXNETNODES];

```

```

autostart = true;
memset (gotinfo,0,sizeof(gotinfo));

if (doomcom->consoleplayer)
{
    // listen for setup info from key player
    printf ("listening for network start info...\n");
    while (1)
    {
        CheckAbort ();
        if (!HGetPacket ())
            continue;
        if (netbuffer->checksum & NCMD_SETUP)
        {
            if (netbuffer->player != VERSION)
                I_Error ("Different DOOM versions cannot play a net game!");
            startskill = netbuffer->retransmitfrom & 15;
            deathmatch = (netbuffer->retransmitfrom & 0xc0) >> 6;
            nomonsters = (netbuffer->retransmitfrom & 0x20) > 0;
            respawnparm = (netbuffer->retransmitfrom & 0x10) > 0;
            startmap = netbuffer->starttic & 0x3f;
            startepisode = netbuffer->starttic >> 6;
            return;
        }
    }
}
else
{
    // key player, send the setup info
    printf ("sending network start info...\n");
    do
    {
        CheckAbort ();
        for (i=0 ; i<doomcom->numnodes ; i++)
        {
            netbuffer->retransmitfrom = startskill;
            if (deathmatch)
                netbuffer->retransmitfrom |= (deathmatch<<6);
            if (nomonsters)
                netbuffer->retransmitfrom |= 0x20;
            if (respawnparm)
                netbuffer->retransmitfrom |= 0x10;
            netbuffer->starttic = startepisode * 64 + startmap;
            netbuffer->player = VERSION;
            netbuffer->numtics = 0;
            HSendPacket (i, NCMD_SETUP);
        }
    } while (i < doomcom->numnodes);

    #if 1
        for(i = 10 ; i && HGetPacket(); --i)
        {
            if((netbuffer->player&0x7f) < MAXNETNODES)
                gotinfo[netbuffer->player&0x7f] = true;
        }
    #else
        while (HGetPacket ())
        {
            gotinfo[netbuffer->player&0x7f] = true;
        }
    #endif
}

```

```

    }
}

//
// D_CheckNetGame
// Works out player numbers among the net participants
//
extern      int                      viewangleoffset;

void D_CheckNetGame (void)
{
    int          i;

    for (i=0 ; i<MAXNETNODES ; i++)
    {
        nodeingame[i] = false;
        nettics[i] = 0;
        remoteresend[i] = false;      // set when local needs tics
        resendto[i] = 0;              // which tic to start sending
    }

    // I_InitNetwork sets doomcom and netgame
    I_InitNetwork ();
    if (doomcom->id != DOOMCOM_ID)
        I_Error ("Doomcom buffer invalid!");

    netbuffer = &doomcom->data;
    consoleplayer = displayplayer = doomcom->consoleplayer;
    if (netgame)
        D_ArbitrateNetStart ();

    printf ("startskill %i deathmatch: %i startmap: %i startepisode: %i\n",
            startskill, deathmatch, startmap, startepisode);

    // read values out of doomcom
    ticdup = doomcom->ticdup;
    maxsend = BACKUPTICS/(2*ticdup)-1;
    if (maxsend<1)
        maxsend = 1;

    for (i=0 ; i<doomcom->numplayers ; i++)
        playeringame[i] = true;
    for (i=0 ; i<doomcom->numnodes ; i++)
        nodeingame[i] = true;

    printf ("player %i of %i (%i nodes)\n",
            consoleplayer+1, doomcom->numplayers, doomcom->numnodes);
}

//
// D_QuitNetGame
// Called before quitting to leave a net game
// without hanging the other players
//
void D_QuitNetGame (void)
{
    int          i, j;

    if (debugfile)
        fclose (debugfile);

    if (!netgame || !usergame || consoleplayer == -1 || demoplayback)
        return;

```

```

// send a bunch of packets for security
netbuffer->player = consoleplayer;
netbuffer->numtics = 0;
for (i=0 ; i<4 ; i++)
{
    for (j=1 ; j<doomcom->numnodes ; j++)
        if (nodeingame[j])
            HSendPacket (j, NCMD_EXIT);
    I_WaitVBL (1);
}
}

```

```

//
// TryRunTics
//
int      frameticks[4];
int      frameon;
int      frameskip[4];
int      oldnettics;

extern      boolean      advancedemo;

void TryRunTics (void)
{
    int      i;
    int      lowtic;
    int      entertic;
    static int      oldentertics;
    int      realtics;
    int      availabletics;
    int      counts;
    int      numplaying;

    // get real tics
    entertic = I_GetTime ()/ticdup;
    realtics = entertic - oldentertics;
    oldentertics = entertic;

    // get available tics
    NetUpdate ();

    lowtic = MAXINT;
    numplaying = 0;
    for (i=0 ; i<doomcom->numnodes ; i++)
    {
        if (nodeingame[i])
        {
            numplaying++;
            if (nettics[i] < lowtic)
                lowtic = nettics[i];
        }
    }
    availabletics = lowtic - gametic/ticdup;

    // decide how many tics to run
    if (realtics < availabletics-1)
        counts = realtics+1;
    else if (realtics < availabletics)
        counts = realtics;
    else
        counts = availabletics;
}

```

```

if (counts < 1)
    counts = 1;

frameon++;

if (debugfile)
    fprintf (debugfile,
            "====real: %i avail: %i game: %i\n",
            realtics, availabletics, counts);

if (!demoplayback)
{
    // ideally nettics[0] should be 1 - 3 tics above lowtic
    // if we are consistantly slower, speed up time
    for (i=0 ; i<MAXPLAYERS ; i++)
        if (playeringame[i])
            break;
    if (consoleplayer == i)
    {
        // the key player does not adapt
    }
    else
    {
        if (nettics[0] <= nettics[nodeforplayer[i]])
        {
            gametime--;
            // printf ("-");
        }
        frameskip[frameon&3] = (oldnettics > nettics[nodeforplayer[i]]);
        oldnettics = nettics[0];
        if (frameskip[0] && frameskip[1] && frameskip[2] && frameskip[3])
        {
            skiptics = 1;
            // printf ("+");
        }
    }
}
} // demoplayback

// wait for new tics if needed
while (lowtic < gametic/ticdup + counts)
{
    NetUpdate ();
    lowtic = MAXINT;

    for (i=0 ; i<doomcom->numnodes ; i++)
        if (nodeingame[i] && nettics[i] < lowtic)
            lowtic = nettics[i];

    if (lowtic < gametic/ticdup)
        I_Error ("TryRunTics: lowtic < gametic");

    // don't stay in here forever -- give the menu a chance to work
    if (I_GetTime ()/ticdup - entertic >= 20)
    {
        M_Ticker ();
        return;
    }
}

// run the count * ticdup dics
while (counts--)
{
    for (i=0 ; i<ticdup ; i++)
    {
        if (gametic/ticdup > lowtic)

```



```

        I_Error ("gametic>lowtic");
if (advancedemo)
    D_DoAdvanceDemo ();
M_Ticker ();
G_Ticker ();
gametic++;

// modify command for duplicated tics
if (i != ticdup-1)
{
    ticcmd_t      *cmd;
    int           buf;
    int           j;

    buf = (gametic/ticdup)%BACKUPTICS;
    for (j=0 ; j<MAXPLAYERS ; j++)
    {
        cmd = &netcmds[j][buf];
        cmd->chatchar = 0;
        if (cmd->buttons & BT_SPECIAL)
            cmd->buttons = 0;
    }
}
}
NetUpdate ();          // check for new console commands
}
}

```

### 3.9 d\_net.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Networking stuff.
//-----

#ifdef __D_NET__
#define __D_NET__

#include "d_player.h"

#ifdef __GNUG__
#pragma interface
#endif

//

```

```

// Network play related stuff.
// There is a data struct that stores network
// communication related stuff, and another
// one that defines the actual packets to
// be transmitted.
//

#define DOOMCOM_ID                0x123456781

// Max computers/players in a game.
#define MAXNETNODES                8

// Networking and tick handling related.
#define BACKUPTICS                12

typedef enum
{
    CMD_SEND                = 1,
    CMD_GET                = 2
} command_t;

//
// Network packet data.
//
typedef struct
{
    // High bit is retransmit request.
    unsigned                checksum;
    // Only valid if NCMD_RETRANSMIT.
    byte                    retransmitfrom;

    byte                    starttic;
    byte                    player;
    byte                    numtics;
    ticcmd_t                cmds[BACKUPTICS];
} doomdata_t;

typedef struct
{
    // Supposed to be DOOMCOM_ID?
    long                    id;

    // DOOM executes an int to execute commands.
    short                    intnum;
    // Communication between DOOM and the driver.
    // Is CMD_SEND or CMD_GET.
    short                    command;
    // Is dest for send, set by get (-1 = no packet).
    short                    remotenode;

    // Number of bytes in doomdata to be sent
    short                    datalength;

    // Info common to all nodes.
    // Console is allways node 0.
    short                    numnodes;
    // Flag: 1 = no duplication, 2-5 = dup for slow nets.
    short                    ticdup;
}

```

```

// Flag: 1 = send a backup tic in every packet.
short          extratics;
// Flag: 1 = deathmatch.
short          deathmatch;
// Flag: -1 = new game, 0-5 = load savegame
short          savegame;
short          episode;          // 1-3
short          map;              // 1-9
short          skill;            // 1-5

// Info specific to this node.
short          consoleplayer;
short          numplayers;

// These are related to the 3-display mode,
// in which two drones looking left and right
// were used to render two additional views
// on two additional computers.
// Probably not operational anymore.
// 1 = left, 0 = center, -1 = right
short          angleoffset;
// 1 = drone
short          drone;

// The packet data to be sent.
doomdata_t     data;

} doomcom_t;

// Create any new ticcmds and broadcast to other players.
void NetUpdate (void);

// Broadcasts special packets to other players
// to notify of game exit
void D_QuitNetGame (void);

//? how many ticks to run?
void TryRunTicks (void);

#endif

//-----
//
// $Log:$
//
//-----

```

### 3.10 d\_player.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//

```

```

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//
//-----

#ifdef __D_PLAYER__
#define __D_PLAYER__

// The player data structure depends on a number
// of other structs: items (internal inventory),
// animation states (closely tied to the sprites
// used to represent them, unfortunately).
#include "d_items.h"
#include "p_pspr.h"

// In addition, the player is just a special
// case of the generic moving object/actor.
#include "p_mobj.h"

// Finally, for odd reasons, the player input
// is buffered within the player data struct,
// as commands per game tick.
#include "d_ticcmd.h"

#ifdef __GNUG__
#pragma interface
#endif

//
// Player states.
//
typedef enum
{
    // Playing or camping.
    PST_LIVE,
    // Dead on the ground, view follows killer.
    PST_DEAD,
    // Ready to restart/respawn???
    PST_REBORN
} playerstate_t;

//
// Player internal flags, for cheats and debug.
//
typedef enum
{
    // No clipping, walk through barriers.
    CF_NOCLIP = 1,
    // No damage, no health loss.
    CF_GODMODE = 2,
    // Not really a cheat, just a debug aid.
    CF_NOMOMENTUM = 4

```

```

} cheat_t;

//
// Extended player object info: player_t
//
typedef struct player_s
{
    mobj_t*          mo;
    playerstate_t    playerstate;
    ticcmd_t         cmd;

    // Determine POV,
    // including viewpoint bobbing during movement.
    // Focal origin above r.z
    fixed_t          viewz;
    // Base height above floor for viewz.
    fixed_t          viewheight;
    // Bob/squat speed.
    fixed_t          deltaviewheight;
    // bounded/scaled total momentum.
    fixed_t          bob;

    // This is only used between levels,
    // mo->health is used during levels.
    int              health;
    int              armorpoints;
    // Armor type is 0-2.
    int              armortype;

    // Power ups. invinc and invis are tic counters.
    int              powers[NUMPOWERS];
    boolean          cards[NUMCARDS];
    boolean          backpack;

    // Frags, kills of other players.
    int              frags[MAXPLAYERS];
    weapontype_t     readyweapon;

    // Is wp_nochange if not changing.
    weapontype_t     pendingweapon;

    boolean          weaponowned[NUMWEAPONS];
    int              ammo[NUMAMMO];
    int              maxammo[NUMAMMO];

    // True if button down last tic.
    int              attackdown;
    int              usedown;

    // Bit flags, for cheats and debug.
    // See cheat_t, above.
    int              cheats;

    // Refired shots are less accurate.
    int              refire;

    // For intermission stats.
    int              killcount;
    int              itemcount;
    int              secretcount;

    // Hint messages.
    char*            message;

```

```

// For screen flashing (red or bright).
int             damagecount;
int             bonuscount;

// Who did damage (NULL for floors/ceilings).
mobj_t*         attacker;

// So gun flashes light up areas.
int             extralight;

// Current PLAYPAL, ???
// can be set to REDCOLORMAP for pain, etc.
int             fixedcolormap;

// Player skin colorshift,
// 0-3 for which color to draw player.
int             colormap;

// Overlay view sprites (gun, etc).
pspdef_t        psprites[NUMPSPRITES];

// True if secret level has been done.
boolean         didsecret;

} player_t;

//
// INTERMISSION
// Structure passed e.g. to WI_Start(wb)
//
typedef struct
{
    boolean      in;           // whether the player is in game

    // Player stats, kills, collected items etc.
    int          skills;
    int          sitems;
    int          ssecret;
    int          stime;
    int          frags[4];
    int          score;        // current score on entry, modified on return

} wbplayerstruct_t;

typedef struct
{
    int          epsd;         // episode # (0-2)

    // if true, splash the secret level
    boolean      didsecret;

    // previous and next levels, origin 0
    int          last;
    int          next;

    int          maxkills;
    int          maxitems;
    int          maxsecret;
    int          maxfrags;

    // the par time
    int          partime;

    // index of this player in game

```

```

        int                pnum;

        wbplayerstruct_t    plyr[MAXPLAYERS];

} wbstartstruct_t;

```

```

#endif
//-----
//
// $Log:$
//
//-----

```

### 3.11 d\_textur.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      Typedefs related to textures etc.,
//      isolated here to make it easier separating modules.
//-----

#ifndef __D_TEXTUR__
#define __D_TEXTUR__

#include "doomtype.h"

//
// Flats?
//
// a pic is an unmasked block of pixels
typedef struct
{
    byte        width;
    byte        height;
    byte        data;
} pic_t;

#endif
//-----
//

```

```

// $Log:$
//
//-----

3.12 d.think.h

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
// MapObj data. Map Objects or mobjs are actors, entities,
// thinker, take-your-pick... anything that moves, acts, or
// suffers state changes of more or less violent nature.
//
//-----

#ifdef __D_THINK__
#define __D_THINK__

#ifdef __GNUG__
#pragma interface
#endif

//
// Experimental stuff.
// To compile this as "ANSI C with classes"
// we will need to handle the various
// action functions cleanly.
//
typedef void (*actionf_v)();
typedef void (*actionf_p1)( void* );
typedef void (*actionf_p2)( void*, void* );

typedef union
{
    actionf_p1      acp1;
    actionf_v       acv;
    actionf_p2      acp2;
} actionf_t;

// Historically, "think_t" is yet another
// function pointer to a routine to handle

```



```

// an actor.
typedef actionf_t think_t;

// Doubly linked list of actors.
typedef struct thinker_s
{
    struct thinker_s*    prev;
    struct thinker_s*    next;
    think_t              function;
} thinker_t;

#endif
//-----
//
// $Log:$
//
//-----

```

### 3.13 d.ticcmd.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      System specific interface stuff.
//
//-----

#ifndef __D_TICCMD__
#define __D_TICCMD__

#include "doomtype.h"

#ifdef __GNUG__
#pragma interface
#endif

// The data sampled per tick (single player)
// and transmitted to other peers (multiplayer).
// Mainly movements/button commands per game tick,
// plus a checksum for internal state consistency.
typedef struct
{
    char    forwardmove;    // *2048 for move
    char    sidemove;       // *2048 for move
    short   angleturn;      // <<16 for angle delta

```

```

    short      consistancy;          // checks for net game
    byte       chatchar;
    byte       buttons;
} ticcmd_t;

```

```

#endif

```

```

//-----
//
// $Log:$
//
//-----

```

## 4 Finale code

### 4.1 f\_finale.c

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Game completion, final screen animation.
//
//-----

```

```

static const char
rcsid[] = "$Id: f_finale.c,v 1.5 1997/02/03 21:26:34 b1 Exp $";

```

```

#include <ctype.h>

```

```

// Functions.
#include "i_system.h"
#include "m_swap.h"
#include "z_zone.h"
#include "v_video.h"
#include "w_wad.h"
#include "s_sound.h"

```

```

// Data.
#include "dstrings.h"
#include "sounds.h"

```

```

#include "doomstat.h"
#include "r_state.h"

```

```

// ?
// #include "doomstat.h"

```

```

//#include "r_local.h"
//#include "f_finale.h"

// Stage of animation:
// 0 = text, 1 = art screen, 2 = character cast
int          finalestage;

int          finalecount;

#define      TEXTSPEED      3
#define      TEXTWAIT      250

char*        e1text = E1TEXT;
char*        e2text = E2TEXT;
char*        e3text = E3TEXT;
char*        e4text = E4TEXT;

char*        c1text = C1TEXT;
char*        c2text = C2TEXT;
char*        c3text = C3TEXT;
char*        c4text = C4TEXT;
char*        c5text = C5TEXT;
char*        c6text = C6TEXT;

char*        p1text = P1TEXT;
char*        p2text = P2TEXT;
char*        p3text = P3TEXT;
char*        p4text = P4TEXT;
char*        p5text = P5TEXT;
char*        p6text = P6TEXT;

char*        t1text = T1TEXT;
char*        t2text = T2TEXT;
char*        t3text = T3TEXT;
char*        t4text = T4TEXT;
char*        t5text = T5TEXT;
char*        t6text = T6TEXT;

char*        finaletext;
char*        finaleflat;

void          F_StartCast (void);
void          F_CastTicker (void);
boolean F_CastResponder (event_t *ev);
void          F_CastDrawer (void);

//
// F_StartFinale
//
void F_StartFinale (void)
{
    gameaction = ga_nothing;
    gamestate = GS_FINALE;
    viewactive = false;
    automapactive = false;

    // Okay - IWAD dependend stuff.
    // This has been changed severly, and
    // some stuff might have changed in the process.
    switch ( gamemode )
    {

        // DOOM 1 - E1, E3 or E4, but each nine missions
        case shareware:
        case registered:

```

```

case retail:
{
    S_ChangeMusic(mus_victor, true);

    switch (gameepisode)
    {
        case 1:
            finaleflat = "FLOOR4_8";
            finaletext = e1text;
            break;
        case 2:
            finaleflat = "SFLR6_1";
            finaletext = e2text;
            break;
        case 3:
            finaleflat = "MFLR8_4";
            finaletext = e3text;
            break;
        case 4:
            finaleflat = "MFLR8_3";
            finaletext = e4text;
            break;
        default:
            // Ouch.
            break;
    }
    break;
}

// DOOM II and missions packs with E1, M34
case commercial:
{
    S_ChangeMusic(mus_read_m, true);

    switch (gamemap)
    {
        case 6:
            finaleflat = "SLIME16";
            finaletext = c1text;
            break;
        case 11:
            finaleflat = "RROCK14";
            finaletext = c2text;
            break;
        case 20:
            finaleflat = "RROCK07";
            finaletext = c3text;
            break;
        case 30:
            finaleflat = "RROCK17";
            finaletext = c4text;
            break;
        case 15:
            finaleflat = "RROCK13";
            finaletext = c5text;
            break;
        case 31:
            finaleflat = "RROCK19";
            finaletext = c6text;
            break;
        default:
            // Ouch.
            break;
    }
    break;
}

```

```

    }

    // Indeterminate.
    default:
        S_ChangeMusic(mus_read_m, true);
        finaleflat = "F_SKY1"; // Not used anywhere else.
        finaletext = c1text; // FIXME - other text, music?
        break;
}

finalestage = 0;
finalecount = 0;
}

boolean F_Responder (event_t *event)
{
    if (finalestage == 2)
        return F_CastResponder (event);

    return false;
}

//
// F_Ticker
//
void F_Ticker (void)
{
    int i;

    // check for skipping
    if ( (gamemode == commercial)
        && ( finalecount > 50) )
    {
        // go on to the next level
        for (i=0 ; i<MAXPLAYERS ; i++)
            if (players[i].cmd.buttons)
                break;

        if (i < MAXPLAYERS)
        {
            if (gamemap == 30)
                F_StartCast ();
            else
                gameaction = ga_worlddone;
        }
    }

    // advance animation
    finalecount++;

    if (finalestage == 2)
    {
        F_CastTicker ();
        return;
    }

    if ( gamemode == commercial)
        return;

    if (!finalestage && finalecount>strlen (finaletext)*TEXTSPEED + TEXTWAIT)

```

```

{
    finalecount = 0;
    finalestage = 1;
    wipegamestate = -1;           // force a wipe
    if (gameepisode == 3)
        S_StartMusic (mus_bunny);
}
}

```

```

//
// F_TextWrite
//

```

```

#include "hu_stuff.h"
extern      patch_t *hu_font[HU_FONTSIZE];

```

```

void F_TextWrite (void)
{
    byte*      src;
    byte*      dest;

    int         x,y,w;
    int         count;
    char*       ch;
    int         c;
    int         cx;
    int         cy;

    // erase the entire screen to a tiled background
    src = W_CacheLumpName ( finaleflat , PU_CACHE);
    dest = screens[0];

    for (y=0 ; y<SCREENHEIGHT ; y++)
    {
        for (x=0 ; x<SCREENWIDTH/64 ; x++)
        {
            memcpy (dest, src+((y&63)<<6), 64);
            dest += 64;
        }
        if (SCREENWIDTH&63)
        {
            memcpy (dest, src+((y&63)<<6), SCREENWIDTH&63);
            dest += (SCREENWIDTH&63);
        }
    }

    V_MarkRect (0, 0, SCREENWIDTH, SCREENHEIGHT);

    // draw some of the text onto the screen
    cx = 10;
    cy = 10;
    ch = finaletext;

    count = (finalecount - 10)/TEXTSPEED;
    if (count < 0)
        count = 0;
    for ( ; count ; count-- )
    {
        c = *ch++;
        if (!c)
            break;
        if (c == '\n')

```

```

    {
        cx = 10;
        cy += 11;
        continue;
    }

    c = toupper(c) - HU_FONTSTART;
    if (c < 0 || c > HU_FONTSIZE)
    {
        cx += 4;
        continue;
    }

    w = SHORT (hu_font[c]->width);
    if (cx+w > SCREENWIDTH)
        break;
    V_DrawPatch(cx, cy, 0, hu_font[c]);
    cx+=w;
}

}

//
// Final DOOM 2 animation
// Casting by id Software.
// in order of appearance
//
typedef struct
{
    char                *name;
    mobjtype_t          type;
} castinfo_t;

castinfo_t             castorder[] = {
    {CC_ZOMBIE, MT_POSSESSED},
    {CC_SHOTGUN, MT_SHOTGUY},
    {CC_HEAVY, MT_CHAINGUY},
    {CC_IMP, MT_TROOP},
    {CC_DEMON, MT_SERGEANT},
    {CC_LOST, MT_SKULL},
    {CC_CACO, MT_HEAD},
    {CC_HELL, MT_KNIGHT},
    {CC_BARON, MT_BRUISER},
    {CC_ARACH, MT_BABY},
    {CC_PAIN, MT_PAIN},
    {CC_REVEN, MT_UNDEAD},
    {CC_MANCU, MT_FATSO},
    {CC_ARCH, MT_VILE},
    {CC_SPIDER, MT_SPIDER},
    {CC_CYBER, MT_CYBORG},
    {CC_HERO, MT_PLAYER},

    {NULL, 0}
};

int                castnum;
int                casttics;
state_t*          caststate;
boolean           castdeath;
int                castframes;
int                castonmelee;
boolean           castattacking;

//

```

```

// F_StartCast
//
extern      gamestate_t      wipegamestate;

void F_StartCast (void)
{
    wipegamestate = -1;          // force a screen wipe
    castnum = 0;
    caststate = &states[mobjinfo[castorder[castnum].type].seestate];
    casttics = caststate->tics;
    castdeath = false;
    finalestage = 2;
    castframes = 0;
    castonmelee = 0;
    castattacking = false;
    S_ChangeMusic(mus_evil, true);
}

//
// F_CastTicker
//
void F_CastTicker (void)
{
    int          st;
    int          sfx;

    if (--casttics > 0)
        return;                  // not time to change state yet

    if (caststate->tics == -1 || caststate->nextstate == S_NULL)
    {
        // switch from deathstate to next monster
        castnum++;
        castdeath = false;
        if (castorder[castnum].name == NULL)
            castnum = 0;
        if (mobjinfo[castorder[castnum].type].seesound)
            S_StartSound (NULL, mobjinfo[castorder[castnum].type].seesound);
        caststate = &states[mobjinfo[castorder[castnum].type].seestate];
        castframes = 0;
    }
    else
    {
        // just advance to next state in animation
        if (caststate == &states[S_PLAY_ATK1])
            goto stopattack;      // Oh, gross hack!
        st = caststate->nextstate;
        caststate = &states[st];
        castframes++;

        // sound hacks....
        switch (st)
        {
            case S_PLAY_ATK1:      sfx = sfx_dshtgn; break;
            case S_POSS_ATK2:      sfx = sfx_pistol; break;
            case S_SPOS_ATK2:      sfx = sfx_shotgn; break;
            case S_VILE_ATK2:      sfx = sfx_vilatk; break;
            case S_SKELE_FIST2:    sfx = sfx_skeswg; break;
            case S_SKELE_FIST4:    sfx = sfx_skepck; break;
            case S_SKELE_MISS2:    sfx = sfx_skeatk; break;
            case S_FATT_ATK8:
            case S_FATT_ATK5:
            case S_FATT_ATK2:      sfx = sfx_firsht; break;
        }
    }
}

```



```

    case S_CPOS_ATK2:
    case S_CPOS_ATK3:
    case S_CPOS_ATK4:          sfx = sfx_shotgn; break;
    case S_TROO_ATK3:          sfx = sfx_claw; break;
    case S_SARG_ATK2:          sfx = sfx_sgtatk; break;
    case S_BOSS_ATK2:
    case S_BOS2_ATK2:
    case S_HEAD_ATK2:          sfx = sfx_firsht; break;
    case S_SKULL_ATK2:          sfx = sfx_sklatk; break;
    case S_SPID_ATK2:
    case S_SPID_ATK3:          sfx = sfx_shotgn; break;
    case S_BSPI_ATK2:          sfx = sfx_plasma; break;
    case S_CYBER_ATK2:
    case S_CYBER_ATK4:
    case S_CYBER_ATK6:          sfx = sfx_rlaunc; break;
    case S_PAIN_ATK3:          sfx = sfx_sklatk; break;
    default: sfx = 0; break;
}

if (sfx)
    S_StartSound (NULL, sfx);
}

if (castframes == 12)
{
    // go into attack frame
    castattacking = true;
    if (castonmelee)
        caststate=&states[mobjinfo[castorder[castnum].type].meleestate];
    else
        caststate=&states[mobjinfo[castorder[castnum].type].missilestate];
    castonmelee ^= 1;
    if (caststate == &states[S_NULL])
    {
        if (castonmelee)
            caststate=
                &states[mobjinfo[castorder[castnum].type].meleestate];
        else
            caststate=
                &states[mobjinfo[castorder[castnum].type].missilestate];
    }
}

if (castattacking)
{
    if (castframes == 24
        || caststate == &states[mobjinfo[castorder[castnum].type].seestate] )
    {
        stopattack:
        castattacking = false;
        castframes = 0;
        caststate = &states[mobjinfo[castorder[castnum].type].seestate];
    }
}

casttics = caststate->tics;
if (casttics == -1)
    casttics = 15;
}

//
// F_CastResponder
//

```

```

boolean F_CastResponder (event_t* ev)
{
    if (ev->type != ev_keydown)
        return false;

    if (castdeath)
        return true;                // already in dying frames

    // go into death frame
    castdeath = true;
    caststate = &states[mobjinfo[castorder[castnum].type].deathstate];
    casttics = caststate->tics;
    castframes = 0;
    castattacking = false;
    if (mobjinfo[castorder[castnum].type].deathsound)
        S_StartSound (NULL, mobjinfo[castorder[castnum].type].deathsound);

    return true;
}

void F_CastPrint (char* text)
{
    char*      ch;
    int        c;
    int        cx;
    int        w;
    int        width;

    // find width
    ch = text;
    width = 0;

    while (ch)
    {
        c = *ch++;
        if (!c)
            break;
        c = toupper(c) - HU_FONTSTART;
        if (c < 0 || c > HU_FONTSIZE)
        {
            width += 4;
            continue;
        }

        w = SHORT (hu_font[c]->width);
        width += w;
    }

    // draw it
    cx = 160-width/2;
    ch = text;
    while (ch)
    {
        c = *ch++;
        if (!c)
            break;
        c = toupper(c) - HU_FONTSTART;
        if (c < 0 || c > HU_FONTSIZE)
        {
            cx += 4;
            continue;
        }

        w = SHORT (hu_font[c]->width);

```

```

        V_DrawPatch(cx, 180, 0, hu_font[c]);
        cx+=w;
    }
}

//
// F_CastDrawer
//
void V_DrawPatchFlipped (int x, int y, int scrn, patch_t *patch);

void F_CastDrawer (void)
{
    spritedef_t*      sprdef;
    spriteframe_t*    sprframe;
    int                lump;
    boolean            flip;
    patch_t*           patch;

    // erase the entire screen to a background
    V_DrawPatch (0,0,0, W_CacheLumpName ("BOSSBACK", PU_CACHE));

    F_CastPrint (castorder[castnum].name);

    // draw the current frame in the middle of the screen
    sprdef = &sprites[caststate->sprite];
    sprframe = &sprdef->spriteframes[ caststate->frame & FF_FRAMEMASK];
    lump = sprframe->lump[0];
    flip = (boolean)sprframe->flip[0];

    patch = W_CacheLumpNum (lump+firstspritelump, PU_CACHE);
    if (flip)
        V_DrawPatchFlipped (160,170,0,patch);
    else
        V_DrawPatch (160,170,0,patch);
}

//
// F_DrawPatchCol
//
void
F_DrawPatchCol
( int                x,
  patch_t*           patch,
  int                col )
{
    column_t*         column;
    byte*              source;
    byte*              dest;
    byte*              desttop;
    int                count;

    column = (column_t *)((byte *)patch + LONG(patch->columnofs[col]));
    desttop = screens[0]+x;

    // step through the posts in a column
    while (column->topdelta != 0xff )
    {
        source = (byte *)column + 3;
        dest = desttop + column->topdelta*SCREENWIDTH;
        count = column->length;

        while (count-->0)

```

```

    {
        *dest = *source++;
        dest += SCREENWIDTH;
    }
    column = (column_t *) ( (byte *)column + column->length + 4 );
}

}

//
// F_BunnyScroll
//
void F_BunnyScroll (void)
{
    int            scrolled;
    int            x;
    patch_t*       p1;
    patch_t*       p2;
    char            name[10];
    int            stage;
    static int      laststage;

    p1 = W_CacheLumpName ("PFUB2", PU_LEVEL);
    p2 = W_CacheLumpName ("PFUB1", PU_LEVEL);

    V_MarkRect (0, 0, SCREENWIDTH, SCREENHEIGHT);

    scrolled = 320 - (finalecount-230)/2;
    if (scrolled > 320)
        scrolled = 320;
    if (scrolled < 0)
        scrolled = 0;

    for ( x=0 ; x<SCREENWIDTH ; x++)
    {
        if (x+scrolled < 320)
            F_DrawPatchCol (x, p1, x+scrolled);
        else
            F_DrawPatchCol (x, p2, x+scrolled - 320);
    }

    if (finalecount < 1130)
        return;
    if (finalecount < 1180)
    {
        V_DrawPatch ((SCREENWIDTH-13*8)/2,
                     (SCREENHEIGHT-8*8)/2, 0, W_CacheLumpName ("END0", PU_CACHE));
        laststage = 0;
        return;
    }

    stage = (finalecount-1180) / 5;
    if (stage > 6)
        stage = 6;
    if (stage > laststage)
    {
        S_StartSound (NULL, sfx_pistol);
        laststage = stage;
    }

    sprintf (name, "END%i", stage);
    V_DrawPatch ((SCREENWIDTH-13*8)/2, (SCREENHEIGHT-8*8)/2, 0, W_CacheLumpName (name, PU_CACHE));
}

```

```

//
// F_Drawer
//
void F_Drawer (void)
{
    if (finalestage == 2)
    {
        F_CastDrawer ();
        return;
    }

    if (!finalestage)
        F_TextWrite ();
    else
    {
        switch (gameepisode)
        {
            case 1:
                if ( gamemode == retail )
                    V_DrawPatch (0,0,0,
                                W_CacheLumpName("CREDIT",PU_CACHE));
                else
                    V_DrawPatch (0,0,0,
                                W_CacheLumpName("HELP2",PU_CACHE));
                break;
            case 2:
                V_DrawPatch(0,0,0,
                            W_CacheLumpName("VICTORY2",PU_CACHE));
                break;
            case 3:
                F_BunnyScroll ();
                break;
            case 4:
                V_DrawPatch (0,0,0,
                            W_CacheLumpName("ENDPIC",PU_CACHE));
                break;
        }
    }
}

```

## 4.2 f\_finale.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//
//

```

```
//-----

#ifndef __F_FINALE__
#define __F_FINALE__

#include "doomtype.h"
#include "d_event.h"
//
// FINALE
//

// Called by main loop.
boolean F_Responder (event_t* ev);

// Called by main loop.
void F_Ticker (void);

// Called by main loop.
void F_Drawer (void);

void F_StartFinale (void);

#endif
//-----
//
// $Log:$
//
//-----
```

### 4.3 f\_wipe.c

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Mission begin melt/wipe screen special effect.
//
//-----

static const char rcsid[] = "$Id: f_wipe.c,v 1.2 1997/02/03 22:45:09 b1 Exp $";
```

```

#include "z_zone.h"
#include "i_video.h"
#include "v_video.h"
#include "m_random.h"

#include "doomdef.h"

#include "f_wipe.h"

//
//                      SCREEN WIPE PACKAGE
//

// when zero, stop the wipe
static boolean      go = 0;

static byte*        wipe_scr_start;
static byte*        wipe_scr_end;
static byte*        wipe_scr;

void
wipe_shittyColMajorXform
( short*            array,
  int                width,
  int                height )
{
  int                x;
  int                y;
  short*            dest;

  dest = (short*) Z_Malloc(width*height*2, PU_STATIC, 0);

  for(y=0;y<height;y++)
    for(x=0;x<width;x++)
      dest[x*height+y] = array[y*width+x];

  memcpy(array, dest, width*height*2);

  Z_Free(dest);
}

int
wipe_initColorXForm
( int                width,
  int                height,
  int                ticks )
{
  memcpy(wipe_scr, wipe_scr_start, width*height);
  return 0;
}

int
wipe_doColorXForm
( int                width,
  int                height,
  int                ticks )
{
  boolean            changed;
  byte*              w;
  byte*              e;
  int                newval;

```

```

changed = false;
w = wipe_scr;
e = wipe_scr_end;

while (w!=wipe_scr+width*height)
{
    if (*w != *e)
    {
        if (*w > *e)
        {
            newval = *w - ticks;
            if (newval < *e)
                *w = *e;
            else
                *w = newval;
            changed = true;
        }
        else if (*w < *e)
        {
            newval = *w + ticks;
            if (newval > *e)
                *w = *e;
            else
                *w = newval;
            changed = true;
        }
    }
    w++;
    e++;
}

return !changed;
}

int
wipe_exitColorXForm
( int    width,
  int    height,
  int    ticks )
{
    return 0;
}

static int*    y;

int
wipe_initMelt
( int    width,
  int    height,
  int    ticks )
{
    int i, r;

    // copy start screen to main screen
    memcpy(wipe_scr, wipe_scr_start, width*height);

    // makes this wipe faster (in theory)
    // to have stuff in column-major format
    wipe_shittyColMajorXform((short*)wipe_scr_start, width/2, height);
    wipe_shittyColMajorXform((short*)wipe_scr_end, width/2, height);

    // setup initial column positions
    // (y<0 => not ready to scroll yet)

```



```

y = (int *) Z_Malloc(width*sizeof(int), PU_STATIC, 0);
y[0] = -(M_Random()%16);
for (i=1;i<width;i++)
{
    r = (M_Random()%3) - 1;
    y[i] = y[i-1] + r;
    if (y[i] > 0) y[i] = 0;
    else if (y[i] == -16) y[i] = -15;
}

return 0;
}

int
wipe_doMelt
( int      width,
  int      height,
  int      ticks )
{
    int      i;
    int      j;
    int      dy;
    int      idx;

    short*    s;
    short*    d;
    boolean    done = true;

    width/=2;

    while (ticks--)
    {
        for (i=0;i<width;i++)
        {
            if (y[i]<0)
            {
                y[i]++; done = false;
            }
            else if (y[i] < height)
            {
                dy = (y[i] < 16) ? y[i]+1 : 8;
                if (y[i]+dy >= height) dy = height - y[i];
                s = &((short *)wipe_scr_end)[i*height+y[i]];
                d = &((short *)wipe_scr)[y[i]*width+i];
                idx = 0;
                for (j=dy;j;j--)
                {
                    d[idx] = *(s++);
                    idx += width;
                }
                y[i] += dy;
                s = &((short *)wipe_scr_start)[i*height];
                d = &((short *)wipe_scr)[y[i]*width+i];
                idx = 0;
                for (j=height-y[i];j;j--)
                {
                    d[idx] = *(s++);
                    idx += width;
                }
                done = false;
            }
        }
    }

    return done;
}

```

```

}

int
wipe_exitMelt
( int      width,
  int      height,
  int      ticks )
{
    Z_Free(y);
    return 0;
}

int
wipe_StartScreen
( int      x,
  int      y,
  int      width,
  int      height )
{
    wipe_scr_start = screens[2];
    I_ReadScreen(wipe_scr_start);
    return 0;
}

int
wipe_EndScreen
( int      x,
  int      y,
  int      width,
  int      height )
{
    wipe_scr_end = screens[3];
    I_ReadScreen(wipe_scr_end);
    V_DrawBlock(x, y, 0, width, height, wipe_scr_start); // restore start scr.
    return 0;
}

int
wipe_ScreenWipe
( int      wipeno,
  int      x,
  int      y,
  int      width,
  int      height,
  int      ticks )
{
    int rc;
    static int (*wipes[])(int, int, int) =
    {
        wipe_initColorXForm, wipe_doColorXForm, wipe_exitColorXForm,
        wipe_initMelt, wipe_doMelt, wipe_exitMelt
    };

    void V_MarkRect(int, int, int, int);

    // initial stuff
    if (!go)
    {
        go = 1;
        // wipe_scr = (byte *) Z_Malloc(width*height, PU_STATIC, 0); // DEBUG
        wipe_scr = screens[0];
        (*wipes[wipeno*3])(width, height, ticks);
    }
}

```

```

// do a piece of wipe-in
V_MarkRect(0, 0, width, height);
rc = (*wipes[wipeno*3+1])(width, height, ticks);
// V_DrawBlock(x, y, 0, width, height, wipe_scr); // DEBUG

// final stuff
if (rc)
{
    go = 0;
    (*wipes[wipeno*3+2])(width, height, ticks);
}

return !go;
}

```

## 4.4 f\_wipe.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      Mission start screen wipe/melt, special effects.
//-----

#ifndef __F_WIPE_H__
#define __F_WIPE_H__

//
//      SCREEN WIPE PACKAGE
//

enum
{
    // simple gradual pixel change for 8-bit only
    wipe_ColorXForm,

    // weird screen melt
    wipe_Melt,

    wipe_NUMWIPEs
};

int
wipe_StartScreen
( int      x,
  int      y,
  int      width,
  int      height );

```

```

int
wipe_EndScreen
( int          x,
  int          y,
  int          width,
  int          height );

int
wipe_ScreenWipe
( int          wipeno,
  int          x,
  int          y,
  int          width,
  int          height,
  int          ticks );

#endif
//-----
//
// $Log:$
//
//-----

```

## 5 Main game loop

### 5.1 g\_game.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:  none
//
//-----

static const char
rcsid[] = "$Id: g_game.c,v 1.8 1997/02/03 22:45:09 b1 Exp $";

#include <string.h>
#include <stdlib.h>

#include "doomdef.h"
#include "doomstat.h"

#include "z_zone.h"
#include "f_finale.h"

```

```

#include "m_argv.h"
#include "m_misc.h"
#include "m_menu.h"
#include "m_random.h"
#include "i_system.h"

#include "p_setup.h"
#include "p_saveg.h"
#include "p_tick.h"

#include "d_main.h"

#include "wi_stuff.h"
#include "hu_stuff.h"
#include "st_stuff.h"
#include "am_map.h"

// Needs access to LFB.
#include "v_video.h"

#include "w_wad.h"

#include "p_local.h"

#include "s_sound.h"

// Data.
#include "dstrings.h"
#include "sounds.h"

// SKY handling - still the wrong place.
#include "r_data.h"
#include "r_sky.h"

#include "g_game.h"

#define SAVEGAMESIZE      0x2c000
#define SAVESTRINGSIZE    24

boolean      G_CheckDemoStatus (void);
void         G_ReadDemoTiccmd (ticcmd_t* cmd);
void         G_WriteDemoTiccmd (ticcmd_t* cmd);
void         G_PlayerReborn (int player);
void         G_InitNew (skill_t skill, int episode, int map);

void         G_DoReborn (int playernum);

void         G_DoLoadLevel (void);
void         G_DoNewGame (void);
void         G_DoLoadGame (void);
void         G_DoPlayDemo (void);
void         G_DoCompleted (void);
void         G_DoVictory (void);
void         G_DoWorldDone (void);
void         G_DoSaveGame (void);

gameaction_t  gameaction;
gamestate_t   gamestate;
skill_t       gameskill;

```

```

boolean        respawnmonsters;
int            gameepisode;
int            gamemap;

boolean        paused;
boolean        sendpause;                // send a pause event next tic
boolean        sendsave;                // send a save event next tic
boolean        usergame;                // ok to save / end game

boolean        timingdemo;                // if true, exit with report on completion
boolean        nodrawers;                // for comparative timing purposes
boolean        noblit;                    // for comparative timing purposes
int            starttime;                // for comparative timing purposes

boolean        viewactive;

boolean        deathmatch;                // only if started as net death
boolean        netgame;                // only true if packets are broadcast
boolean        playeringame[MAXPLAYERS];
player_t       players[MAXPLAYERS];

int            consoleplayer;            // player taking events and displaying
int            displayplayer;            // view being displayed
int            gametic;
int            levelstarttic;            // gametic at level start
int            totalkills, totalitems, totalsecret; // for intermission

char           demoname[32];
boolean        demorecording;
boolean        demoplayback;
boolean        netdemo;
byte*          demobuffer;
byte*          demo_p;
byte*          demoend;
boolean        singledemo;                // quit after playing a demo from cmdline

boolean        precache = true;          // if true, load all graphics at start

wbstartstruct_t wminfo;                // parms for world map / intermission

short          consistancy[MAXPLAYERS][BACKUPTICS];

byte*          savebuffer;

//
// controls (have defaults)
//
int            key_right;
int            key_left;

int            key_up;
int            key_down;
int            key_strafeleft;
int            key_straferight;
int            key_fire;
int            key_use;
int            key_strafe;
int            key_speed;

int            mousebfire;
int            mousebstrafe;
int            mousebforward;

int            joybfire;

```

```

int            joybstrafe;
int            joybuse;
int            joybspeed;

#define MAXPLMOVE          (forwardmove[1])

#define TURBOTHRESHOLD      0x32

fixed_t        forwardmove[2] = {0x19, 0x32};
fixed_t        sidemove[2] = {0x18, 0x28};
fixed_t        angleturn[3] = {640, 1280, 320};          // + slow turn

#define SLOWTURNICS        6

#define NUMKEYS            256

boolean        gamekeydown[NUMKEYS];
int            turnheld;                                // for accelerative turning

boolean        mousearray[4];
boolean*        mousebuttons = &mousearray[1];          // allow [-1]

// mouse values are used once
int            mousex;
int            mousey;

int            dclicktime;
int            dclickstate;
int            dclicks;
int            dclicktime2;
int            dclickstate2;
int            dclicks2;

// joystick values are repeated
int            joyxmove;
int            joyymove;
boolean        joyarray[5];
boolean*        joybuttons = &joyarray[1];              // allow [-1]

int            savegameslot;
char            savedescription[32];

#define          BODYQUESIZE      32

mobj_t*        bodyque[BODYQUESIZE];
int            bodyqueslot;

void*          statcopy;                                // for statistics driver

int G_CmdChecksum (ticcmd_t* cmd)
{
    int            i;
    int            sum = 0;

    for (i=0 ; i< sizeof(*cmd)/4 - 1 ; i++)
        sum += ((int *)cmd)[i];

    return sum;
}

```

```

//
// G_BuildTiccmd
// Builds a ticcmd from all of the available inputs
// or reads it from the demo buffer.
// If recording a demo, write it out
//
void G_BuildTiccmd (ticcmd_t* cmd)
{
    int            i;
    boolean        strafe;
    boolean        bstrafe;
    int            speed;
    int            tspeed;
    int            forward;
    int            side;

    ticcmd_t*      base;

    base = I_BaseTiccmd ();          // empty, or external driver
    memcpy (cmd,base,sizeof(*cmd));

    cmd->consistancy =
        consistancy[consoleplayer][maketic%BACKUPTICS];

    strafe = gamekeydown[key_strafe] || mousebuttons[mousebstrafe]
        || joybuttons[joybstrafe];
    speed = gamekeydown[key_speed] || joybuttons[joybspeed];

    forward = side = 0;

    // use two stage accelerative turning
    // on the keyboard and joystick
    if (joyxmove < 0
        || joyxmove > 0
        || gamekeydown[key_right]
        || gamekeydown[key_left])
        turnheld += ticdup;
    else
        turnheld = 0;

    if (turnheld < SLOWTURNICS)
        tspeed = 2;          // slow turn
    else
        tspeed = speed;

    // let movement keys cancel each other out
    if (strafe)
    {
        if (gamekeydown[key_right])
        {
            // fprintf(stderr, "strafe right\n");
            side += sidemove[speed];
        }
        if (gamekeydown[key_left])
        {
            //          fprintf(stderr, "strafe left\n");
            side -= sidemove[speed];
        }
        if (joyxmove > 0)
            side += sidemove[speed];
        if (joyxmove < 0)
            side -= sidemove[speed];
    }
}

```



```

}
else
{
    if (gamekeydown[key_right])
        cmd->angleturn -= angleturn[tspeed];
    if (gamekeydown[key_left])
        cmd->angleturn += angleturn[tspeed];
    if (joyxmove > 0)
        cmd->angleturn -= angleturn[tspeed];
    if (joyxmove < 0)
        cmd->angleturn += angleturn[tspeed];
}

if (gamekeydown[key_up])
{
    // fprintf(stderr, "up\n");
    forward += forwardmove[speed];
}
if (gamekeydown[key_down])
{
    // fprintf(stderr, "down\n");
    forward -= forwardmove[speed];
}
if (joyymove < 0)
    forward += forwardmove[speed];
if (joyymove > 0)
    forward -= forwardmove[speed];
if (gamekeydown[key_straferight])
    side += sidemove[speed];
if (gamekeydown[key_strafeleft])
    side -= sidemove[speed];

// buttons
cmd->chatchar = HU_dequeueChatChar();

if (gamekeydown[key_fire] || mousebuttons[mousebfire]
    || joybuttons[joybfire])
    cmd->buttons |= BT_ATTACK;

if (gamekeydown[key_use] || joybuttons[joybuse] )
{
    cmd->buttons |= BT_USE;
    // clear double clicks if hit use button
    dclicks = 0;
}

// chainsaw overrides
for (i=0 ; i<NUMWEAPONS-1 ; i++)
    if (gamekeydown['1'+i])
    {
        cmd->buttons |= BT_CHANGE;
        cmd->buttons |= i<<BT_WEAPONSHIFT;
        break;
    }

// mouse
if (mousebuttons[mousebforward])
    forward += forwardmove[speed];

// forward double click
if (mousebuttons[mousebforward] != dclickstate && dclicktime > 1 )
{
    dclickstate = mousebuttons[mousebforward];
    if (dclickstate)
        dclicks++;
}

```

```

    if (dclicks == 2)
    {
        cmd->buttons |= BT_USE;
        dclicks = 0;
    }
    else
        dclicktime = 0;
}
else
{
    dclicktime += ticdup;
    if (dclicktime > 20)
    {
        dclicks = 0;
        dclickstate = 0;
    }
}

// strafe double click
bstrafe =
    mousebuttons[mousebstrafe]
    || joybuttons[joybstrafe];
if (bstrafe != dclickstate2 && dclicktime2 > 1 )
{
    dclickstate2 = bstrafe;
    if (dclickstate2)
        dclicks2++;
    if (dclicks2 == 2)
    {
        cmd->buttons |= BT_USE;
        dclicks2 = 0;
    }
    else
        dclicktime2 = 0;
}
else
{
    dclicktime2 += ticdup;
    if (dclicktime2 > 20)
    {
        dclicks2 = 0;
        dclickstate2 = 0;
    }
}

forward += mousey;
if (strafe)
    side += mousex*2;
else
    cmd->angleturn -= mousex*0x8;

mousex = mousey = 0;

if (forward > MAXPLMOVE)
    forward = MAXPLMOVE;
else if (forward < -MAXPLMOVE)
    forward = -MAXPLMOVE;
if (side > MAXPLMOVE)
    side = MAXPLMOVE;
else if (side < -MAXPLMOVE)
    side = -MAXPLMOVE;

cmd->forwardmove += forward;
cmd->sidemove += side;

```

```

// special buttons
if (sendpause)
{
    sendpause = false;
    cmd->buttons = BT_SPECIAL | BTS_PAUSE;
}

if (sendsave)
{
    sendsave = false;
    cmd->buttons = BT_SPECIAL | BTS_SAVEGAME | (savegameslot<<BTS_SAVESHIFT);
}
}

//
// G_DoLoadLevel
//
extern gamestate_t      wipegamestate;

void G_DoLoadLevel (void)
{
    int          i;

    // Set the sky map.
    // First thing, we have a dummy sky texture name,
    // a flat. The data is in the WAD only because
    // we look for an actual index, instead of simply
    // setting one.
    skyflatnum = R_FlatNumForName ( SKYFLATNAME );

    // DOOM determines the sky texture to be used
    // depending on the current episode, and the game version.
    if ( (gamemode == commercial)
        || ( gamemode == pack_tnt )
        || ( gamemode == pack_plut ) )
    {
        skytexture = R_TextureNumForName ("SKY3");
        if (gamemap < 12)
            skytexture = R_TextureNumForName ("SKY1");
        else
            if (gamemap < 21)
                skytexture = R_TextureNumForName ("SKY2");
    }

    levelstarttic = gametic;          // for time calculation

    if (wipegamestate == GS_LEVEL)
        wipegamestate = -1;          // force a wipe

    gamestate = GS_LEVEL;

    for (i=0 ; i<MAXPLAYERS ; i++)
    {
        if (playeringame[i] && players[i].playerstate == PST_DEAD)
            players[i].playerstate = PST_REBORN;
        memset (players[i].frags,0,sizeof(players[i].frags));
    }

    P_SetupLevel (gameepisode, gamemap, 0, gameskill);
    displayplayer = consoleplayer;    // view the guy you are playing
    starttime = I_GetTime ();
    gameaction = ga_nothing;
    Z_CheckHeap ();

```

```

// clear cmd building stuff
memset (gamekeydown, 0, sizeof(gamekeydown));
joyxmove = joyymove = 0;
mousex = mousey = 0;
sendpause = sendsave = paused = false;
memset (mousebuttons, 0, sizeof(mousebuttons));
memset (joybuttons, 0, sizeof(joybuttons));
}

//
// G_Responder
// Get info needed to make ticcmd_ts for the players.
//
boolean G_Responder (event_t* ev)
{
    // allow spy mode changes even during the demo
    if (gamestate == GS_LEVEL && ev->type == ev_keydown
        && ev->data1 == KEY_F12 && (singledemo || !deathmatch) )
    {
        // spy mode
        do
        {
            displayplayer++;
            if (displayplayer == MAXPLAYERS)
                displayplayer = 0;
        } while (!playeringame[displayplayer] && displayplayer != consoleplayer);
        return true;
    }

    // any other key pops up menu if in demos
    if (gameaction == ga_nothing && !singledemo &&
        (demoplayback || gamestate == GS_DEMOSCREEN)
        )
    {
        if (ev->type == ev_keydown ||
            (ev->type == ev_mouse && ev->data1) ||
            (ev->type == ev_joystick && ev->data1) )
        {
            M_StartControlPanel ();
            return true;
        }
        return false;
    }

    if (gamestate == GS_LEVEL)
    {
#ifdef 0
        if (devparm && ev->type == ev_keydown && ev->data1 == ';' )
        {
            G_DeathMatchSpawnPlayer (0);
            return true;
        }
#endif
#ifdef 1
        if (HU_Responder (ev))
            return true;          // chat ate the event
        if (ST_Responder (ev))
            return true;          // status window ate it
        if (AM_Responder (ev))
            return true;          // automap ate it
        }

    if (gamestate == GS_FINALE)
    {
        if (F_Responder (ev))

```

```

        return true;          // finale ate the event
    }

    switch (ev->type)
    {
        case ev_keydown:
            if (ev->data1 == KEY_PAUSE)
            {
                sendpause = true;
                return true;
            }
            if (ev->data1 < NUMKEYS)
                gamekeydown[ev->data1] = true;
            return true;      // eat key down events

        case ev_keyup:
            if (ev->data1 < NUMKEYS)
                gamekeydown[ev->data1] = false;
            return false;     // always let key up events filter down

        case ev_mouse:
            mousebuttons[0] = ev->data1 & 1;
            mousebuttons[1] = ev->data1 & 2;
            mousebuttons[2] = ev->data1 & 4;
            mousex = ev->data2*(mouseSensitivity+5)/10;
            mousey = ev->data3*(mouseSensitivity+5)/10;
            return true;      // eat events

        case ev_joystick:
            joybuttons[0] = ev->data1 & 1;
            joybuttons[1] = ev->data1 & 2;
            joybuttons[2] = ev->data1 & 4;
            joybuttons[3] = ev->data1 & 8;
            joyxmove = ev->data2;
            joymove = ev->data3;
            return true;      // eat events

        default:
            break;
    }

    return false;
}

//
// G_Ticker
// Make ticcmd_ts for the players.
//
void G_Ticker (void)
{
    int            i;
    int            buf;
    ticcmd_t*      cmd;

    // do player reborns if needed
    for (i=0 ; i<MAXPLAYERS ; i++)
        if (playeringame[i] && players[i].playerstate == PST_REBORN)
            G_DoReborn (i);

    // do things to change the game state
    while (gameaction != ga_nothing)
    {
        switch (gameaction)

```

```

{
    case ga_loadlevel:
        G_DoLoadLevel ();
        break;
    case ga_newgame:
        G_DoNewGame ();
        break;
    case ga_loadgame:
        G_DoLoadGame ();
        break;
    case ga_savegame:
        G_DoSaveGame ();
        break;
    case ga_playdemo:
        G_DoPlayDemo ();
        break;
    case ga_completed:
        G_DoCompleted ();
        break;
    case ga_victory:
        F_StartFinale ();
        break;
    case ga_worlddone:
        G_DoWorldDone ();
        break;
    case ga_screenshot:
        M_ScreenShot ();
        gameaction = ga_nothing;
        break;
    case ga_nothing:
        break;
}
}

// get commands, check consistancy,
// and build new consistancy check
buf = (gametic/ticdup)%BACKUPTICS;

for (i=0 ; i<MAXPLAYERS ; i++)
{
    if (playeringame[i])
    {
        cmd = &players[i].cmd;

        memcpy (cmd, &netcmds[i][buf], sizeof(ticcmd_t));

        if (demoplayback)
            G_ReadDemoTiccmd (cmd);
        if (demorecording)
            G_WriteDemoTiccmd (cmd);

        // check for turbo cheats
        if (cmd->forwardmove > TURBOTHRESHOLD
            && !(gametic&31) && ((gametic>>5)&3) == i )
        {
            static char turbomessage[80];
            extern char *player_names[4];
            sprintf (turbomessage, "%s is turbo!",player_names[i]);
            players[consoleplayer].message = turbomessage;
        }

        if (netgame && !netdemo && !(gametic%ticdup) )
        {
            if (gametic > BACKUPTICS
                && consistancy[i][buf] != cmd->consistancy)

```

```

        {
            I_Error ("consistency failure (%i should be %i)",
                    cmd->consistency, consistency[i][buf]);
        }
        if (players[i].mo)
            consistency[i][buf] = players[i].mo->x;
        else
            consistency[i][buf] = rndindex;
    }
}

// check for special buttons
for (i=0 ; i<MAXPLAYERS ; i++)
{
    if (playeringame[i])
    {
        if (players[i].cmd.buttons & BT_SPECIAL)
        {
            switch (players[i].cmd.buttons & BT_SPECIALMASK)
            {
                case BTS_PAUSE:
                    paused ^= 1;
                    if (paused)
                        S_PauseSound ();
                    else
                        S_ResumeSound ();
                    break;

                case BTS_SAVEGAME:
                    if (!savedescription[0])
                        strcpy (savedescription, "NET GAME");
                    savegameslot =
                        (players[i].cmd.buttons & BTS_SAVEMASK)>>BTS_SAVESHIFT;
                    gameaction = ga_savegame;
                    break;
            }
        }
    }
}

// do main actions
switch (gamestate)
{
    case GS_LEVEL:
        P_Ticker ();
        ST_Ticker ();
        AM_Ticker ();
        HU_Ticker ();
        break;

    case GS_INTERMISSION:
        WI_Ticker ();
        break;

    case GS_FINALE:
        F_Ticker ();
        break;

    case GS_DEMOSCREEN:
        D_PageTicker ();
        break;
}
}

```

```

//
// PLAYER STRUCTURE FUNCTIONS
// also see P_SpawnPlayer in P_Things
//

//
// G_InitPlayer
// Called at the start.
// Called by the game initialization functions.
//
void G_InitPlayer (int player)
{
    player_t*      p;

    // set up the saved info
    p = &players[player];

    // clear everything else to defaults
    G_PlayerReborn (player);
}

//
// G_PlayerFinishLevel
// Can when a player completes a level.
//
void G_PlayerFinishLevel (int player)
{
    player_t*      p;

    p = &players[player];

    memset (p->powers, 0, sizeof (p->powers));
    memset (p->cards, 0, sizeof (p->cards));
    p->mo->flags &= ~MF_SHADOW;           // cancel invisibility
    p->extralight = 0;                   // cancel gun flashes
    p->fixedcolormap = 0;                // cancel ir goggles
    p->damagecount = 0;                  // no palette changes
    p->bonuscount = 0;

}

//
// G_PlayerReborn
// Called after a player dies
// almost everything is cleared and initialized
//
void G_PlayerReborn (int player)
{
    player_t*      p;
    int            i;
    int            frags[MAXPLAYERS];
    int            killcount;
    int            itemcount;
    int            secretcount;

    memcpy (frags, players[player].frags, sizeof(frags));
    killcount = players[player].killcount;
    itemcount = players[player].itemcount;
    secretcount = players[player].secretcount;

    p = &players[player];

```



```

memset (p, 0, sizeof(*p));

memcpy (players[player].frags, frags, sizeof(players[player].frags));
players[player].killcount = killcount;
players[player].itemcount = itemcount;
players[player].secretcount = secretcount;

p->usedown = p->attackdown = true;          // don't do anything immediately
p->playerstate = PST_LIVE;
p->health = MAXHEALTH;
p->readyweapon = p->pendingweapon = wp_pistol;
p->weaponowned[wp_fist] = true;
p->weaponowned[wp_pistol] = true;
p->ammo[am_clip] = 50;

for (i=0 ; i<NUMAMMO ; i++)
    p->maxammo[i] = maxammo[i];
}

//
// G_CheckSpot
// Returns false if the player cannot be respawned
// at the given mapthing_t spot
// because something is occupying it
//
void P_SpawnPlayer (mapthing_t* mthing);

boolean
G_CheckSpot
( int                playernum,
  mapthing_t*       mthing )
{
    fixed_t          x;
    fixed_t          y;
    subsector_t*     ss;
    unsigned          an;
    mobj_t*           mo;
    int               i;

    if (!players[playernum].mo)
    {
        // first spawn of level, before corpses
        for (i=0 ; i<playernum ; i++)
            if (players[i].mo->x == mthing->x << FRACBITS
                && players[i].mo->y == mthing->y << FRACBITS)
                return false;
        return true;
    }

    x = mthing->x << FRACBITS;
    y = mthing->y << FRACBITS;

    if (!P_CheckPosition (players[playernum].mo, x, y) )
        return false;

    // flush an old corpse if needed
    if (bodyqueslot >= BODYQUESIZE)
        P_RemoveMobj (bodyque[bodyqueslot%BODYQUESIZE]);
    bodyque[bodyqueslot%BODYQUESIZE] = players[playernum].mo;
    bodyqueslot++;

    // spawn a teleport fog
    ss = R_PointInSubsector (x,y);
    an = ( ANG45 * (mthing->angle/45) ) >> ANGLETOFINESHIFT;

```

```

mo = P_SpawnMobj (x+20*finecosine[an], y+20*finesine[an]
                 , ss->sector->floorheight
                 , MT_TFOG);

if (players[consoleplayer].viewz != 1)
    S_StartSound (mo, sfx_telept);        // don't start sound on first frame

return true;
}

//
// G_DeathMatchSpawnPlayer
// Spawns a player at one of the random death match spots
// called at level load and each death
//
void G_DeathMatchSpawnPlayer (int playernum)
{
    int            i,j;
    int            selections;

    selections = deathmatch_p - deathmatchstarts;
    if (selections < 4)
        I_Error ("Only %i deathmatch spots, 4 required", selections);

    for (j=0 ; j<20 ; j++)
    {
        i = P_Random() % selections;
        if (G_CheckSpot (playernum, &deathmatchstarts[i]) )
        {
            deathmatchstarts[i].type = playernum+1;
            P_SpawnPlayer (&deathmatchstarts[i]);
            return;
        }
    }

    // no good spot, so the player will probably get stuck
    P_SpawnPlayer (&playerstarts[playernum]);
}

//
// G_DoReborn
//
void G_DoReborn (int playernum)
{
    int            i;

    if (!netgame)
    {
        // reload the level from scratch
        gameaction = ga_loadlevel;
    }
    else
    {
        // respawn at the start

        // first dissasociate the corpse
        players[playernum].mo->player = NULL;

        // spawn at random spot if in death match
        if (deathmatch)
        {
            G_DeathMatchSpawnPlayer (playernum);
            return;
        }
    }
}

```

```

    }

    if (G_CheckSpot (playernum, &playerstarts[playernum]) )
    {
        P_SpawnPlayer (&playerstarts[playernum]);
        return;
    }

    // try to spawn at one of the other players spots
    for (i=0 ; i<MAXPLAYERS ; i++)
    {
        if (G_CheckSpot (playernum, &playerstarts[i]) )
        {
            playerstarts[i].type = playernum+1;          // fake as other player
            P_SpawnPlayer (&playerstarts[i]);
            playerstarts[i].type = i+1;                  // restore
            return;
        }
        // he's going to be inside something. Too bad.
    }
    P_SpawnPlayer (&playerstarts[playernum]);
}

}

void G_ScreenShot (void)
{
    gameaction = ga_screenshot;
}

// DOOM Par Times
int pars[4][10] =
{
    {0},
    {0,30,75,120,90,165,180,180,30,165},
    {0,90,90,90,120,90,360,240,30,170},
    {0,90,45,90,150,90,90,165,30,135}
};

// DOOM II Par Times
int cpars[32] =
{
    30,90,120,120,90,150,120,120,270,90,          // 1-10
    210,150,150,150,210,150,420,150,210,150,      // 11-20
    240,150,180,150,150,300,330,420,300,180,      // 21-30
    120,30                                         // 31-32
};

//
// G_DoCompleted
//
boolean          secretextit;
extern char*      pagename;

void G_ExitLevel (void)
{
    secretextit = false;
    gameaction = ga_completed;
}

// Here's for the german edition.
void G_SecretExitLevel (void)

```

```

{
    // IF NO WOLF3D LEVELS, NO SECRET EXIT!
    if ( (gamemode == commercial)
        && (W_CheckNumForName("map31")<0))
        secretexit = false;
    else
        secretexit = true;
    gameaction = ga_completed;
}

void G_DoCompleted (void)
{
    int            i;

    gameaction = ga_nothing;

    for (i=0 ; i<MAXPLAYERS ; i++)
        if (playeringame[i])
            G_PlayerFinishLevel (i);          // take away cards and stuff

    if (automapactive)
        AM_Stop ();

    if ( gamemode != commercial)
        switch(gamemap)
        {
            case 8:
                gameaction = ga_victory;
                return;
            case 9:
                for (i=0 ; i<MAXPLAYERS ; i++)
                    players[i].didsecret = true;
                break;
        }

    // #if 0 Hmmm - why?
    if ( (gamemap == 8)
        && (gamemode != commercial) )
    {
        // victory
        gameaction = ga_victory;
        return;
    }

    if ( (gamemap == 9)
        && (gamemode != commercial) )
    {
        // exit secret level
        for (i=0 ; i<MAXPLAYERS ; i++)
            players[i].didsecret = true;
    }
    // #endif

    wminfo.didsecret = players[consoleplayer].didsecret;
    wminfo.epsd = gameepisode -1;
    wminfo.last = gamemap -1;

    // wminfo.next is 0 biased, unlike gamemap
    if ( gamemode == commercial)
    {
        if (secretexit)
            switch(gamemap)
            {
                case 15: wminfo.next = 30; break;
            }
    }
}

```

```

        case 31: wminfo.next = 31; break;
    }
    else
        switch(gamemap)
        {
            case 31:
            case 32: wminfo.next = 15; break;
            default: wminfo.next = gamemap;
        }
}
else
{
    if (secretexit)
        wminfo.next = 8;          // go to secret level
    else if (gamemap == 9)
    {
        // returning from secret level
        switch (gameepisode)
        {
            case 1:
                wminfo.next = 3;
                break;
            case 2:
                wminfo.next = 5;
                break;
            case 3:
                wminfo.next = 6;
                break;
            case 4:
                wminfo.next = 2;
                break;
        }
    }
    else
        wminfo.next = gamemap;      // go to next level
}

wminfo.maxkills = totalkills;
wminfo.maxitems = totalitems;
wminfo.maxsecret = totalsecret;
wminfo.maxfrags = 0;
if ( gamemode == commercial )
    wminfo.partime = 35*cpars[gamemap-1];
else
    wminfo.partime = 35*pars[gameepisode][gamemap];
wminfo.pnum = consoleplayer;

for (i=0 ; i<MAXPLAYERS ; i++)
{
    wminfo.plyr[i].in = playeringame[i];
    wminfo.plyr[i].skills = players[i].killcount;
    wminfo.plyr[i].sitems = players[i].itemcount;
    wminfo.plyr[i].ssecret = players[i].secretcount;
    wminfo.plyr[i].stime = leveltime;
    memcpy (wminfo.plyr[i].frags, players[i].frags
            , sizeof(wminfo.plyr[i].frags));
}

gamestate = GS_INTERMISSION;
viewactive = false;
automapactive = false;

if (statcopy)
    memcpy (statcopy, &wminfo, sizeof(wminfo));

```

```

    WI_Start (&wminfo);
}

//
// G_WorldDone
//
void G_WorldDone (void)
{
    gameaction = ga_worldddone;

    if (secretexit)
        players[consoleplayer].didsecret = true;

    if ( gamemode == commercial )
    {
        switch (gamemap)
        {
            case 15:
            case 31:
                if (!secretexit)
                    break;
            case 6:
            case 11:
            case 20:
            case 30:
                F_StartFinale ();
                break;
        }
    }
}

void G_DoWorldDone (void)
{
    gamestate = GS_LEVEL;
    gamemap = wminfo.next+1;
    G_DoLoadLevel ();
    gameaction = ga_nothing;
    viewactive = true;
}

//
// G_InitFromSavegame
// Can be called by the startup code or the menu task.
//
extern boolean setsizeneeded;
void R_ExecuteSetViewSize (void);

char          savename[256];

void G_LoadGame (char* name)
{
    strcpy (savename, name);
    gameaction = ga_loadgame;
}

#define VERSIONSIZE          16

void G_DoLoadGame (void)
{
    int          length;
    int          i;

```

```

int          a,b,c;
char         vcheck[VERSIONSIZE];

gameaction = ga_nothing;

length = M_ReadFile (savename, &savebuffer);
save_p = savebuffer + SAVESTRINGSIZE;

// skip the description field
memset (vcheck,0,sizeof(vcheck));
sprintf (vcheck,"version %i",VERSION);
if (strcmp (save_p, vcheck))
    return;                                // bad version
save_p += VERSIONSIZE;

gameskill = *save_p++;
gameepisode = *save_p++;
gamemap = *save_p++;
for (i=0 ; i<MAXPLAYERS ; i++)
    playeringame[i] = *save_p++;

// load a base level
G_InitNew (gameskill, gameepisode, gamemap);

// get the times
a = *save_p++;
b = *save_p++;
c = *save_p++;
leveltime = (a<<16) + (b<<8) + c;

// dearchive all the modifications
P_UnArchivePlayers ();
P_UnArchiveWorld ();
P_UnArchiveThinkers ();
P_UnArchiveSpecials ();

if (*save_p != 0x1d)
    I_Error ("Bad savegame");

// done
Z_Free (savebuffer);

if (setsizeneeded)
    R_ExecuteSetViewSize ();

// draw the pattern into the back screen
R_FillBackScreen ();
}

//
// G_SaveGame
// Called by the menu task.
// Description is a 24 byte text string
//
void
G_SaveGame
( int          slot,
  char*        description )
{
    savegameslot = slot;
    strcpy (savedescription, description);
    sendsave = true;
}

```

```

void G_DoSaveGame (void)
{
    char        name[100];
    char        name2[VERSIONSIZE];
    char*       description;
    int         length;
    int         i;

    if (M_CheckParm("-cdrom"))
        sprintf(name,"c:\\doomdata\\"SAVEGAMENAME"%d.dsg",savegameslot);
    else
        sprintf (name,SAVEGAMENAME"%d.dsg",savegameslot);
    description = savedescription;

    save_p = savebuffer = screens[1]+0x4000;

    memcpy (save_p, description, SAVESTRINGSIZE);
    save_p += SAVESTRINGSIZE;
    memset (name2,0,sizeof(name2));
    sprintf (name2,"version %i",VERSION);
    memcpy (save_p, name2, VERSIONSIZE);
    save_p += VERSIONSIZE;

    *save_p++ = gameskill;
    *save_p++ = gameepisode;
    *save_p++ = gamemap;
    for (i=0 ; i<MAXPLAYERS ; i++)
        *save_p++ = playeringame[i];
    *save_p++ = leveltime>>16;
    *save_p++ = leveltime>>8;
    *save_p++ = leveltime;

    P_ArchivePlayers ();
    P_ArchiveWorld ();
    P_ArchiveThinkers ();
    P_ArchiveSpecials ();

    *save_p++ = 0x1d;                // consistancy marker

    length = save_p - savebuffer;
    if (length > SAVEGAMESIZE)
        I_Error ("Savegame buffer overrun");
    M_WriteFile (name, savebuffer, length);
    gameaction = ga_nothing;
    savedescription[0] = 0;

    players[consoleplayer].message = GGSAVED;

    // draw the pattern into the back screen
    R_FillBackScreen ();
}

//
// G_InitNew
// Can be called by the startup code or the menu task,
// consoleplayer, displayplayer, playeringame[] should be set.
//
skill_t      d_skill;
int          d_episode;
int          d_map;

void
G_DeferredInitNew
( skill_t      skill,

```



```

int          episode,
int          map)
{
    d_skill = skill;
    d_episode = episode;
    d_map = map;
    gameaction = ga_newgame;
}

void G_DoNewGame (void)
{
    demoplayback = false;
    netdemo = false;
    netgame = false;
    deathmatch = false;
    playeringame[1] = playeringame[2] = playeringame[3] = 0;
    respawnparm = false;
    fastparm = false;
    nomonsters = false;
    consoleplayer = 0;
    G_InitNew (d_skill, d_episode, d_map);
    gameaction = ga_nothing;
}

// The sky texture to be used instead of the F_SKY1 dummy.
extern int      skytexture;

void
G_InitNew
( skill_t      skill,
  int          episode,
  int          map )
{
    int          i;

    if (paused)
    {
        paused = false;
        S_ResumeSound ();
    }

    if (skill > sk_nightmare)
        skill = sk_nightmare;

    // This was quite messy with SPECIAL and commented parts.
    // Supposedly hacks to make the latest edition work.
    // It might not work properly.
    if (episode < 1)
        episode = 1;

    if ( gamemode == retail )
    {
        if (episode > 4)
            episode = 4;
    }
    else if ( gamemode == shareware )
    {
        if (episode > 1)
            episode = 1;          // only start episode 1 on shareware
    }
    else

```

```

{
    if (episode > 3)
        episode = 3;
}

if (map < 1)
    map = 1;

if ( (map > 9)
    && ( gamemode != commercial) )
    map = 9;

M_ClearRandom ();

if (skill == sk_nightmare || respawnparm )
    respawnmonsters = true;
else
    respawnmonsters = false;

if (fastparm || (skill == sk_nightmare && gameskill != sk_nightmare) )
{
    for (i=S_SARG_RUN1 ; i<=S_SARG_PAIN2 ; i++)
        states[i].tics >>= 1;
    mobjinfo[MT_BRUISERSHOT].speed = 20*FRACUNIT;
    mobjinfo[MT_HEADSHOT].speed = 20*FRACUNIT;
    mobjinfo[MT_TROOPSHOT].speed = 20*FRACUNIT;
}
else if (skill != sk_nightmare && gameskill == sk_nightmare)
{
    for (i=S_SARG_RUN1 ; i<=S_SARG_PAIN2 ; i++)
        states[i].tics <<= 1;
    mobjinfo[MT_BRUISERSHOT].speed = 15*FRACUNIT;
    mobjinfo[MT_HEADSHOT].speed = 10*FRACUNIT;
    mobjinfo[MT_TROOPSHOT].speed = 10*FRACUNIT;
}

// force players to be initialized upon first level load
for (i=0 ; i<MAXPLAYERS ; i++)
    players[i].playerstate = PST_REBORN;

usergame = true;                // will be set false if a demo
paused = false;
demoplayback = false;
automapactive = false;
viewactive = true;
gameepisode = episode;
gamemap = map;
gameskill = skill;

viewactive = true;

// set the sky map for the episode
if ( gamemode == commercial)
{
    skytexture = R_TextureNumForName ("SKY3");
    if (gamemap < 12)
        skytexture = R_TextureNumForName ("SKY1");
    else
        if (gamemap < 21)
            skytexture = R_TextureNumForName ("SKY2");
}
else

```

```

switch (episode)
{
    case 1:
        skytexture = R_TextureNumForName ("SKY1");
        break;
    case 2:
        skytexture = R_TextureNumForName ("SKY2");
        break;
    case 3:
        skytexture = R_TextureNumForName ("SKY3");
        break;
    case 4:          // Special Edition sky
        skytexture = R_TextureNumForName ("SKY4");
        break;
}

G_DoLoadLevel ();
}

//
// DEMO RECORDING
//
#define DEMOMARKER          0x80

void G_ReadDemoTiccmd (ticcmd_t* cmd)
{
    if (*demo_p == DEMOMARKER)
    {
        // end of demo data stream
        G_CheckDemoStatus ();
        return;
    }
    cmd->forwardmove = ((signed char)*demo_p++);
    cmd->sidemove = ((signed char)*demo_p++);
    cmd->angleturn = ((unsigned char)*demo_p++)<<8;
    cmd->buttons = (unsigned char)*demo_p++;
}

void G_WriteDemoTiccmd (ticcmd_t* cmd)
{
    if (gamekeydown['q'])          // press q to end demo recording
        G_CheckDemoStatus ();
    *demo_p++ = cmd->forwardmove;
    *demo_p++ = cmd->sidemove;
    *demo_p++ = (cmd->angleturn+128)>>8;
    *demo_p++ = cmd->buttons;
    demo_p -= 4;
    if (demo_p > demoend - 16)
    {
        // no more space
        G_CheckDemoStatus ();
        return;
    }

    G_ReadDemoTiccmd (cmd);          // make SURE it is exactly the same
}

//
// G_RecordDemo
//

```

```

void G_RecordDemo (char* name)
{
    int            i;
    int            maxsize;

    usergame = false;
    strcpy (demoname, name);
    strcat (demoname, ".lmp");
    maxsize = 0x20000;
    i = M_CheckParm ("-maxdemo");
    if (i && i<myargc-1)
        maxsize = atoi(myargv[i+1])*1024;
    demobuffer = Z_Malloc (maxsize,PU_STATIC,NULL);
    demoend = demobuffer + maxsize;

    demorecording = true;
}

void G_BeginRecording (void)
{
    int            i;

    demo_p = demobuffer;

    *demo_p++ = VERSION;
    *demo_p++ = gameskill;
    *demo_p++ = gameepisode;
    *demo_p++ = gamemap;
    *demo_p++ = deathmatch;
    *demo_p++ = respawnparm;
    *demo_p++ = fastparm;
    *demo_p++ = nomonsters;
    *demo_p++ = consoleplayer;

    for (i=0 ; i<MAXPLAYERS ; i++)
        *demo_p++ = playeringame[i];
}

//
// G_PlayDemo
//

char*            defdemoname;

void G_DeferedPlayDemo (char* name)
{
    defdemoname = name;
    gameaction = ga_playdemo;
}

void G_DoPlayDemo (void)
{
    skill_t skill;
    int            i, episode, map;

    gameaction = ga_nothing;
    demobuffer = demo_p = W_CacheLumpName (defdemoname, PU_STATIC);
    if ( *demo_p++ != VERSION)
    {
        fprintf( stderr, "Demo is from a different game version!\n");
        gameaction = ga_nothing;
        return;
    }
}

```

```

skill = *demo_p++;
episode = *demo_p++;
map = *demo_p++;
deathmatch = *demo_p++;
respawnparm = *demo_p++;
fastparm = *demo_p++;
nomonsters = *demo_p++;
consoleplayer = *demo_p++;

for (i=0 ; i<MAXPLAYERS ; i++)
    playeringame[i] = *demo_p++;
if (playeringame[1])
{
    netgame = true;
    netdemo = true;
}

// don't spend a lot of time in loadlevel
precache = false;
G_InitNew (skill, episode, map);
precache = true;

usergame = false;
demoplayback = true;
}

//
// G_TimeDemo
//
void G_TimeDemo (char* name)
{
    nodrawers = M_CheckParm ("-nodraw");
    noblit = M_CheckParm ("-noblit");
    timingdemo = true;
    singletics = true;

    defdemoname = name;
    gameaction = ga_playdemo;
}

/*
=====
=
= G_CheckDemoStatus
=
= Called after a death or level completion to allow demos to be cleaned up
= Returns true if a new demo loop action will take place
=====
*/

boolean G_CheckDemoStatus (void)
{
    int          endtime;

    if (timingdemo)
    {
        endtime = I_GetTime ();
        I_Error ("timed %i gametics in %i realtics",gametic
            , endtime-starttime);
    }

    if (demoplayback)
    {

```

```

    if (singledemo)
        I_Quit ();

    Z_ChangeTag (demobuffer, PU_CACHE);
    demoplayback = false;
    netdemo = false;
    netgame = false;
    deathmatch = false;
    playeringame[1] = playeringame[2] = playeringame[3] = 0;
    respawnparm = false;
    fastparm = false;
    nomonsters = false;
    consoleplayer = 0;
    D_AdvanceDemo ();
    return true;
}

if (demorecording)
{
    *demo_p++ = DEMOMARKER;
    M_WriteFile (demoname, demobuffer, demo_p - demobuffer);
    Z_Free (demobuffer);
    demorecording = false;
    I_Error ("Demo %s recorded",demoname);
}

return false;
}

```

## 5.2 g\_game.h

```

// Emacs style mode select    -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//   Duh.
//
//-----

#ifndef __G_GAME__
#define __G_GAME__

#include "doomdef.h"
#include "d_event.h"

```

```

//
// GAME
//
void G_DeathMatchSpawnPlayer (int playernum);

void G_InitNew (skill_t skill, int episode, int map);

// Can be called by the startup code or M_Responder.
// A normal game starts at map 1,
// but a warp test can start elsewhere
void G_DeferedInitNew (skill_t skill, int episode, int map);

void G_DeferedPlayDemo (char* demo);

// Can be called by the startup code or M_Responder,
// calls P_SetupLevel or W_EnterWorld.
void G_LoadGame (char* name);

void G_DoLoadGame (void);

// Called by M_Responder.
void G_SaveGame (int slot, char* description);

// Only called by startup code.
void G_RecordDemo (char* name);

void G_BeginRecording (void);

void G_PlayDemo (char* name);
void G_TimeDemo (char* name);
boolean G_CheckDemoStatus (void);

void G_ExitLevel (void);
void G_SecretExitLevel (void);

void G_WorldDone (void);

void G_Ticker (void);
boolean G_Responder (event_t*      ev);

void G_ScreenShot (void);

#endif
//-----
//
// $Log:$
//
//-----

```

## 6 Heads-up display

### 6.1 hu\_lib.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.

```

```

//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION: heads-up text and input code
//
//-----

static const char
rcsid[] = "$Id: hu_lib.c,v 1.3 1997/01/26 07:44:58 b1 Exp $";

#include <ctype.h>

#include "doomdef.h"

#include "v_video.h"
#include "m_swap.h"

#include "hu_lib.h"
#include "r_local.h"
#include "r_draw.h"

// boolean : whether the screen is always erased
#define noterased viewwindowx

extern boolean      automapactive;      // in AM_map.c

void HUlib_init(void)
{
}

void HUlib_clearTextLine(hu_textline_t* t)
{
    t->len = 0;
    t->l[0] = 0;
    t->needsupdate = true;
}

void
HUlib_initTextLine
( hu_textline_t*      t,
  int                  x,
  int                  y,
  patch_t**            f,
  int                  sc )
{
    t->x = x;
    t->y = y;
    t->f = f;
    t->sc = sc;
    HUlib_clearTextLine(t);
}

boolean
HUlib_addCharToTextLine
( hu_textline_t*      t,
  char                  ch )
{
    if (t->len == HU_MAXLINELENGTH)
        return false;

```



```

else
{
    t->l[t->len++] = ch;
    t->l[t->len] = 0;
    t->needsupdate = 4;
    return true;
}

}

boolean HUlib_delCharFromTextLine(hu_textline_t* t)
{
    if (!t->len) return false;
    else
    {
        t->l[--t->len] = 0;
        t->needsupdate = 4;
        return true;
    }
}

void
HUlib_drawTextLine
( hu_textline_t*      l,
  boolean              drawcursor )
{
    int                i;
    int                w;
    int                x;
    unsigned char      c;

    // draw the new stuff
    x = l->x;
    for (i=0;i<l->len;i++)
    {
        c = toupper(l->l[i]);
        if (c != ' '
            && c >= l->sc
            && c <= '_')
        {
            w = SHORT(l->f[c - l->sc]->width);
            if (x+w > SCREENWIDTH)
                break;
            V_DrawPatchDirect(x, l->y, FG, l->f[c - l->sc]);
            x += w;
        }
        else
        {
            x += 4;
            if (x >= SCREENWIDTH)
                break;
        }
    }

    // draw the cursor if requested
    if (drawcursor
        && x + SHORT(l->f['_'] - l->sc)->width) <= SCREENWIDTH)
    {
        V_DrawPatchDirect(x, l->y, FG, l->f['_'] - l->sc);
    }
}

```

```

// sorta called by HU_Erase and just better darn get things straight
void HUlib_eraseTextLine(hu_textline_t* l)
{
    int                lh;
    int                y;
    int                yoffset;
    static boolean     lastautomapactive = true;

    // Only erases when NOT in automap and the screen is reduced,
    // and the text must either need updating or refreshing
    // (because of a recent change back from the automap)

    if (!automapactive &&
        viewwindowx && l->needsupdate)
    {
        lh = SHORT(l->f[0]->height) + 1;
        for (y=l->y,yoffset=y*SCREENWIDTH ; y<l->y+lh ; y++,yoffset+=SCREENWIDTH)
        {
            if (y < viewwindowy || y >= viewwindowy + viewheight)
                R_VideoErase(yoffset, SCREENWIDTH); // erase entire line
            else
            {
                R_VideoErase(yoffset, viewwindowx); // erase left border
                R_VideoErase(yoffset + viewwindowx + viewwidth, viewwindowx);
                // erase right border
            }
        }
    }

    lastautomapactive = automapactive;
    if (l->needsupdate) l->needsupdate--;
}

void
HUlib_initSText
( hu_stext_t*      s,
  int              x,
  int              y,
  int              h,
  patch_t**        font,
  int              startchar,
  boolean*         on )
{
    int i;

    s->h = h;
    s->on = on;
    s->laston = true;
    s->c1 = 0;
    for (i=0;i<h;i++)
        HUlib_initTextLine(&s->l[i],
                           x, y - i*(SHORT(font[0]->height)+1),
                           font, startchar);
}

void HUlib_addLineToSText(hu_stext_t* s)
{
    int i;

    // add a clear line

```

```

    if (++s->cl == s->h)
        s->cl = 0;
    HUlib_clearTextLine(&s->l[s->cl]);

    // everything needs updating
    for (i=0 ; i<s->h ; i++)
        s->l[i].needsupdate = 4;
}

void
HUlib_addMessageToSText
( hu_stext_t*      s,
  char*            prefix,
  char*            msg )
{
    HUlib_addLineToSText(s);
    if (prefix)
        while (*prefix)
            HUlib_addCharToTextLine(&s->l[s->cl], *(prefix++));

    while (*msg)
        HUlib_addCharToTextLine(&s->l[s->cl], *(msg++));
}

void HUlib_drawSText(hu_stext_t* s)
{
    int i, idx;
    hu_textline_t *l;

    if (!s->on)
        return; // if not on, don't draw

    // draw everything
    for (i=0 ; i<s->h ; i++)
    {
        idx = s->cl - i;
        if (idx < 0)
            idx += s->h; // handle queue of lines

        l = &s->l[idx];

        // need a decision made here on whether to skip the draw
        HUlib_drawTextLine(l, false); // no cursor, please
    }
}

void HUlib_eraseSText(hu_stext_t* s)
{
    int i;

    for (i=0 ; i<s->h ; i++)
    {
        if (s->laston && !s->on)
            s->l[i].needsupdate = 4;
        HUlib_eraseTextLine(&s->l[i]);
    }
    s->laston = *s->on;
}

void
HUlib_initIText

```

```

( hu_itext_t*      it,
  int              x,
  int              y,
  patch_t**        font,
  int              startchar,
  boolean*          on )
{
    it->lm = 0; // default left margin is start of text
    it->on = on;
    it->laston = true;
    HULib_initTextLine(&it->l, x, y, font, startchar);
}

// The following deletion routines adhere to the left margin restriction
void HULib_delCharFromIText(hu_itext_t* it)
{
    if (it->l.len != it->lm)
        HULib_delCharFromTextLine(&it->l);
}

void HULib_eraseLineFromIText(hu_itext_t* it)
{
    while (it->lm != it->l.len)
        HULib_delCharFromTextLine(&it->l);
}

// Resets left margin as well
void HULib_resetIText(hu_itext_t* it)
{
    it->lm = 0;
    HULib_clearTextLine(&it->l);
}

void
HULib_addPrefixToIText
( hu_itext_t*      it,
  char*            str )
{
    while (*str)
        HULib_addCharToTextLine(&it->l, *(str++));
    it->lm = it->l.len;
}

// wrapper function for handling general keyed input.
// returns true if it ate the key
boolean
HULib_keyInIText
( hu_itext_t*      it,
  unsigned char ch )
{
    if (ch >= ' ' && ch <= '_')
        HULib_addCharToTextLine(&it->l, (char) ch);
    else
        if (ch == KEY_BACKSPACE)
            HULib_delCharFromIText(it);
        else
            if (ch != KEY_ENTER)
                return false; // did not eat key

    return true; // ate the key
}

```

```

void HULib_drawIText(hu_itext_t* it)
{
    hu_textline_t *l = &it->l;

    if (!*it->on)
        return;
    HULib_drawTextLine(l, true); // draw the line w/ cursor
}

void HULib_eraseIText(hu_itext_t* it)
{
    if (it->laston && !*it->on)
        it->l.needsupdate = 4;
    HULib_eraseTextLine(&it->l);
    it->laston = *it->on;
}

```

## 6.2 hu.lib.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:  none
//
//-----

#ifdef __HULIB__
#define __HULIB__

// We are referring to patches.
#include "r_defs.h"

// background and foreground screen numbers
// different from other modules.
#define BG 1
#define FG 0

// font stuff
#define HU_CHARERASE KEY_BACKSPACE

#define HU_MAXLINES 4
#define HU_MAXLINELENGTH 80

//
// Typedefs of widgets
//

```

```

// Text Line widget
// (parent of Scrolling Text and Input Text widgets)
typedef struct
{
    // left-justified position of scrolling text window
    int          x;
    int          y;

    patch_t**    f;                // font
    int          sc;                // start character
    char         l[HU_MAXLINELENGTH+1]; // line of text
    int          len;                // current line length

    // whether this line needs to be updated
    int          needsupdate;
} hu_textline_t;

// Scrolling Text window widget
// (child of Text Line widget)
typedef struct
{
    hu_textline_t    l[HU_MAXLINES]; // text lines to draw
    int              h;                // height in lines
    int              cl;                // current line number

    // pointer to boolean stating whether to update window
    boolean*         on;
    boolean          laston;           // last value of *->on.
} hu_stext_t;

// Input Text Line widget
// (child of Text Line widget)
typedef struct
{
    hu_textline_t    l;                // text line to input on

    // left margin past which I am not to delete characters
    int              lm;

    // pointer to boolean stating whether to update window
    boolean*         on;
    boolean          laston;           // last value of *->on;
} hu_itext_t;

//
// Widget creation, access, and update routines
//

// initializes heads-up widget library
void HUlib_init(void);

//
// textline code
//

// clear a line of text
void HUlib_clearTextLine(hu_textline_t *t);

```

```

void      HULib_initTextLine(hu_textline_t *t, int x, int y, patch_t **f, int sc);

// returns success
boolean HULib_addCharToTextLine(hu_textline_t *t, char ch);

// returns success
boolean HULib_delCharFromTextLine(hu_textline_t *t);

// draws tline
void      HULib_drawTextLine(hu_textline_t *l, boolean drawcursor);

// erases text line
void      HULib_eraseTextLine(hu_textline_t *l);


//
// Scrolling Text window widget routines
//

// ?
void
HULib_initSText
( hu_stext_t*      s,
  int              x,
  int              y,
  int              h,
  patch_t**        font,
  int              startchar,
  boolean*         on );

// add a new line
void HULib_addLineToSText(hu_stext_t* s);

// ?
void
HULib_addMessageToSText
( hu_stext_t*      s,
  char*            prefix,
  char*            msg );

// draws stext
void HULib_drawSText(hu_stext_t* s);

// erases all stext lines
void HULib_eraseSText(hu_stext_t* s);

// Input Text Line widget routines
void
HULib_initIText
( hu_itext_t*      it,
  int              x,
  int              y,
  patch_t**        font,
  int              startchar,
  boolean*         on );

// enforces left margin
void HULib_delCharFromIText(hu_itext_t* it);

// enforces left margin
void HULib_eraseLineFromIText(hu_itext_t* it);

// resets line and left margin
void HULib_resetIText(hu_itext_t* it);

```

```

// left of left-margin
void
HUlib_addPrefixToIText
( hu_itext_t*      it,
  char*            str );

// whether eaten
boolean
HUlib_keyInIText
( hu_itext_t*      it,
  unsigned char ch );

void HUlib_drawIText(hu_itext_t* it);

// erases all itext lines
void HUlib_eraseIText(hu_itext_t* it);

#endif
//-----
//
// $Log:$
//
//-----

```

### 6.3 hu\_stuff.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:  Heads-up displays
//
//-----

static const char
rcsid[] = "$Id: hu_stuff.c,v 1.4 1997/02/03 16:47:52 b1 Exp $";

#include <ctype.h>

#include "doomdef.h"

#include "z_zone.h"

#include "m_swap.h"

#include "hu_stuff.h"
#include "hu_lib.h"
#include "w_wad.h"

```



```

#include "s_sound.h"

#include "doomstat.h"

// Data.
#include "dstrings.h"
#include "sounds.h"

//
// Locally used constants, shortcuts.
//
#define HU_TITLE      (mapnames[(gameepisode-1)*9+gamemap-1])
#define HU_TITLE2     (mapnames2[gamemap-1])
#define HU_TITLEP     (mapnamesp[gamemap-1])
#define HU_TITLET     (mapnamest[gamemap-1])
#define HU_TITLEHEIGHT 1
#define HU_TITLX      0
#define HU_TITLEY     (167 - SHORT(hu_font[0]->height))

#define HU_INPUTTOGGLE 't'
#define HU_INPUTX      HU_MSGX
#define HU_INPUTY      (HU_MSGY + HU_MSGHEIGHT*(SHORT(hu_font[0]->height) +1))
#define HU_INPUTWIDTH  64
#define HU_INPUTHEIGHT 1

char*      chat_macros[] =
{
    HISTR_CHATMACRO0,
    HISTR_CHATMACRO1,
    HISTR_CHATMACRO2,
    HISTR_CHATMACRO3,
    HISTR_CHATMACRO4,
    HISTR_CHATMACRO5,
    HISTR_CHATMACRO6,
    HISTR_CHATMACRO7,
    HISTR_CHATMACRO8,
    HISTR_CHATMACRO9
};

char*      player_names[] =
{
    HISTR_PLRGREEN,
    HISTR_PLRINDIGO,
    HISTR_PLRBROWN,
    HISTR_PLRRED
};

char      chat_char; // remove later.
static player_t* plr;
static patch_t* hu_font[HU_FONTSIZE];
static hu_textline_t w_title;
static boolean chat_on;
static hu_itext_t w_chat;
static boolean always_off = false;
static char chat_dest[MAXPLAYERS];
static hu_itext_t w_inputbuffer[MAXPLAYERS];

static boolean message_on;
static boolean message_dontfuckwithme;
static boolean message_nottobefuckedwith;

static hu_stext_t w_message;

```

```

static int                message_counter;

extern int                showMessages;
extern boolean           automapactive;

static boolean           headsupactive = false;

//
// Builtin map names.
// The actual names can be found in DStrings.h.
//

char*        mapnames[] =        // DOOM shareware/registered/retail (Ultimate) names.
{

    HUSTR_E1M1,
    HUSTR_E1M2,
    HUSTR_E1M3,
    HUSTR_E1M4,
    HUSTR_E1M5,
    HUSTR_E1M6,
    HUSTR_E1M7,
    HUSTR_E1M8,
    HUSTR_E1M9,

    HUSTR_E2M1,
    HUSTR_E2M2,
    HUSTR_E2M3,
    HUSTR_E2M4,
    HUSTR_E2M5,
    HUSTR_E2M6,
    HUSTR_E2M7,
    HUSTR_E2M8,
    HUSTR_E2M9,

    HUSTR_E3M1,
    HUSTR_E3M2,
    HUSTR_E3M3,
    HUSTR_E3M4,
    HUSTR_E3M5,
    HUSTR_E3M6,
    HUSTR_E3M7,
    HUSTR_E3M8,
    HUSTR_E3M9,

    HUSTR_E4M1,
    HUSTR_E4M2,
    HUSTR_E4M3,
    HUSTR_E4M4,
    HUSTR_E4M5,
    HUSTR_E4M6,
    HUSTR_E4M7,
    HUSTR_E4M8,
    HUSTR_E4M9,

    "NEWLEVEL",
    "NEWLEVEL",
    "NEWLEVEL",
    "NEWLEVEL",
    "NEWLEVEL",
    "NEWLEVEL",
    "NEWLEVEL",
    "NEWLEVEL",
    "NEWLEVEL"

};

```

```

char*      mapnames2[] =          // DOOM 2 map names.
{
    HUSTR_1,
    HUSTR_2,
    HUSTR_3,
    HUSTR_4,
    HUSTR_5,
    HUSTR_6,
    HUSTR_7,
    HUSTR_8,
    HUSTR_9,
    HUSTR_10,
    HUSTR_11,

    HUSTR_12,
    HUSTR_13,
    HUSTR_14,
    HUSTR_15,
    HUSTR_16,
    HUSTR_17,
    HUSTR_18,
    HUSTR_19,
    HUSTR_20,

    HUSTR_21,
    HUSTR_22,
    HUSTR_23,
    HUSTR_24,
    HUSTR_25,
    HUSTR_26,
    HUSTR_27,
    HUSTR_28,
    HUSTR_29,
    HUSTR_30,
    HUSTR_31,
    HUSTR_32
};

char*      mapnamesp[] =          // Plutonia WAD map names.
{
    PHUSTR_1,
    PHUSTR_2,
    PHUSTR_3,
    PHUSTR_4,
    PHUSTR_5,
    PHUSTR_6,
    PHUSTR_7,
    PHUSTR_8,
    PHUSTR_9,
    PHUSTR_10,
    PHUSTR_11,

    PHUSTR_12,
    PHUSTR_13,
    PHUSTR_14,
    PHUSTR_15,
    PHUSTR_16,
    PHUSTR_17,
    PHUSTR_18,
    PHUSTR_19,
    PHUSTR_20,

    PHUSTR_21,

```

```

    PHUSTR_22,
    PHUSTR_23,
    PHUSTR_24,
    PHUSTR_25,
    PHUSTR_26,
    PHUSTR_27,
    PHUSTR_28,
    PHUSTR_29,
    PHUSTR_30,
    PHUSTR_31,
    PHUSTR_32
};

char *mapnamest[] =          // TNT WAD map names.
{
    THUSTR_1,
    THUSTR_2,
    THUSTR_3,
    THUSTR_4,
    THUSTR_5,
    THUSTR_6,
    THUSTR_7,
    THUSTR_8,
    THUSTR_9,
    THUSTR_10,
    THUSTR_11,

    THUSTR_12,
    THUSTR_13,
    THUSTR_14,
    THUSTR_15,
    THUSTR_16,
    THUSTR_17,
    THUSTR_18,
    THUSTR_19,
    THUSTR_20,

    THUSTR_21,
    THUSTR_22,
    THUSTR_23,
    THUSTR_24,
    THUSTR_25,
    THUSTR_26,
    THUSTR_27,
    THUSTR_28,
    THUSTR_29,
    THUSTR_30,
    THUSTR_31,
    THUSTR_32
};

const char*          shiftxform;

const char french_shiftxform[] =
{
    0,
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
    11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
    21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
    31,
    ' ', '!', '"', '#', '$', '%', '&',
    '"', // shift-'
    '(', ')', '*', '+',

```

```

'?', // shift-,
'_', // shift--
'>', // shift-.
'?', // shift-/
'0', // shift-0
'1', // shift-1
'2', // shift-2
'3', // shift-3
'4', // shift-4
'5', // shift-5
'6', // shift-6
'7', // shift-7
'8', // shift-8
'9', // shift-9
'/',
'.'. // shift-;
'<',
'+', // shift-=
'>', '?', '@',
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
'[', // shift-[
'!', // shift-backslash - OH MY GOD DOES WATCOM SUCK
']', // shift-]
'", '_ ',
'\'', // shift-`
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
'{', '|', '}', '~', 127

```

```
};
```

```
const char english_shiftxform[] =
{
```

```

0,
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
31,
' ', '!', '"', '#', '$', '%', '&',
'", // shift-'
'(', ')', '*', '+',
'<', // shift-,
'_', // shift--
'>', // shift-.
'?', // shift-/
')', // shift-0
'!', // shift-1
'@', // shift-2
'#', // shift-3
'$', // shift-4
'%', // shift-5
'^', // shift-6
'&', // shift-7
'*', // shift-8
'(', // shift-9
':',
':', // shift-;
'<',
'+', // shift-=
'>', '?', '@',
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
'[', // shift-[

```

```

    '!', // shift-backslash - OH MY GOD DOES WATCOM SUCK
    ']', // shift-]
    '"', '_ ',
    '\', // shift-`
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
    'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
    '{', '|', '}', '~', 127
};

```

```

char frenchKeyMap[128]=
{
    0,
    1,2,3,4,5,6,7,8,9,10,
    11,12,13,14,15,16,17,18,19,20,
    21,22,23,24,25,26,27,28,29,30,
    31,
    ' ', '!', '"', '#', '$', '%', '&', '\'', '(', ')', '*', '+', ',', '-', '.', ':', ';',
    '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', 'M', '<', '=', '>', '?',
    '@', 'Q', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'N', 'O',
    'P', 'A', 'R', 'S', 'T', 'U', 'V', 'Z', 'X', 'Y', 'W', '^', '\\', '$', '^', '_ ',
    '@', 'Q', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'N', 'O',
    'P', 'A', 'R', 'S', 'T', 'U', 'V', 'Z', 'X', 'Y', 'W', '^', '\\', '$', '^', 127
};

```

```

char ForeignTranslation(unsigned char ch)
{
    return ch < 128 ? frenchKeyMap[ch] : ch;
}

```

```

void HU_Init(void)
{
    int i;
    int j;
    char buffer[9];

    if (french)
        shiftxform = french_shiftxform;
    else
        shiftxform = english_shiftxform;

    // load the heads-up font
    j = HU_FONTSTART;
    for (i=0;i<HU_FONTSIZE;i++)
    {
        sprintf(buffer, "STCFN%.3d", j++);
        hu_font[i] = (patch_t *) W_CacheLumpName(buffer, PU_STATIC);
    }
}

```

```

void HU_Stop(void)
{
    headsupactive = false;
}

```

```

void HU_Start(void)
{

```

```

    int i;
    char* s;

```

```

    if (headsupactive)
        HU_Stop();

```

```

plr = &players[consoleplayer];
message_on = false;
message_dontfuckwithme = false;
message_nottobefuckedwith = false;
chat_on = false;

// create the message widget
HUlib_initSText(&w_message,
                HU_MSGX, HU_MSGY, HU_MSGHEIGHT,
                hu_font,
                HU_FONTSTART, &message_on);

// create the map title widget
HUlib_initTextLine(&w_title,
                  HU_TITLEX, HU_TITLEY,
                  hu_font,
                  HU_FONTSTART);

switch ( gamemode )
{
    case shareware:
    case registered:
    case retail:
        s = HU_TITLE;
        break;

/* FIXME
    case pack_plut:
        s = HU_TITLEP;
        break;
    case pack_tnt:
        s = HU_TITLET;
        break;
*/

    case commercial:
    default:
        s = HU_TITLE2;
        break;
}

while (*s)
    HUlib_addCharToTextLine(&w_title, *(s++));

// create the chat widget
HUlib_initIText(&w_chat,
               HU_INPUTX, HU_INPUTY,
               hu_font,
               HU_FONTSTART, &chat_on);

// create the inputbuffer widgets
for (i=0 ; i<MAXPLAYERS ; i++)
    HUlib_initIText(&w_inputbuffer[i], 0, 0, 0, 0, &always_off);

headsupactive = true;
}

void HU_Drawer(void)
{
    HUlib_drawSText(&w_message);
    HUlib_drawIText(&w_chat);
    if (automapactive)
        HUlib_drawTextLine(&w_title, false);
}

```

```

}

void HU_Erase(void)
{
    HUlib_eraseSText(&w_message);
    HUlib_eraseIText(&w_chat);
    HUlib_eraseTextLine(&w_title);
}

void HU_Ticker(void)
{
    int i, rc;
    char c;

    // tick down message counter if message is up
    if (message_counter && !--message_counter)
    {
        message_on = false;
        message_nottobefuckedwith = false;
    }

    if (showMessages || message_dontfuckwithme)
    {
        // display message if necessary
        if ((plr->message && !message_nottobefuckedwith)
            || (plr->message && message_dontfuckwithme))
        {
            HUlib_addMessageToSText(&w_message, 0, plr->message);
            plr->message = 0;
            message_on = true;
            message_counter = HU_MSGTIMEOUT;
            message_nottobefuckedwith = message_dontfuckwithme;
            message_dontfuckwithme = 0;
        }
    }

    // else message_on = false;

    // check for incoming chat characters
    if (netgame)
    {
        for (i=0 ; i<MAXPLAYERS; i++)
        {
            if (!playeringame[i])
                continue;
            if (i != consoleplayer
                && (c = players[i].cmd.chatchar))
            {
                if (c <= HU_BROADCAST)
                    chat_dest[i] = c;
                else
                {
                    if (c >= 'a' && c <= 'z')
                        c = (char) shiftxform[(unsigned char) c];
                    rc = HUlib_keyInIText(&w_inputbuffer[i], c);
                    if (rc && c == KEY_ENTER)
                    {
                        if (w_inputbuffer[i].l.len
                            && (chat_dest[i] == consoleplayer+1
                                || chat_dest[i] == HU_BROADCAST))
                        {

```



```

        HUlib_addMessageToSText(&w_message,
                                player_names[i],
                                w_inputbuffer[i].l.l);

        message_nottobefuckedwith = true;
        message_on = true;
        message_counter = HU_MSGTIMEOUT;
        if ( gamemode == commercial )
            S_StartSound(0, sfx_radio);
        else
            S_StartSound(0, sfx_tink);
    }
    HUlib_resetIText(&w_inputbuffer[i]);
}
}
}
players[i].cmd.chatchar = 0;
}
}
}

}

#define QUEUESIZE                128

static char        chatchars[QUEUESIZE];
static int         head = 0;
static int         tail = 0;

void HU_queueChatChar(char c)
{
    if (((head + 1) & (QUEUESIZE-1)) == tail)
    {
        plr->message = HUSTR_MSGU;
    }
    else
    {
        chatchars[head] = c;
        head = (head + 1) & (QUEUESIZE-1);
    }
}

char HU_dequeueChatChar(void)
{
    char c;

    if (head != tail)
    {
        c = chatchars[tail];
        tail = (tail + 1) & (QUEUESIZE-1);
    }
    else
    {
        c = 0;
    }

    return c;
}

boolean HU_Responder(event_t *ev)
{
    static char        lastmessage[HU_MAXLINELENGTH+1];
    char*              macromessage;
    boolean             eatkey = false;

```

```

static boolean      shiftdown = false;
static boolean      altdown = false;
unsigned char       c;
int                 i;
int                 numplayers;

static char          destination_keys[MAXPLAYERS] =
{
    HUSTR_KEYGREEN,
    HUSTR_KEYINDIGO,
    HUSTR_KEYBROWN,
    HUSTR_KEYRED
};

static int           num_nobrainers = 0;

numplayers = 0;
for (i=0 ; i<MAXPLAYERS ; i++)
    numplayers += playeringame[i];

if (ev->data1 == KEY_RSHIFT)
{
    shiftdown = ev->type == ev_keydown;
    return false;
}
else if (ev->data1 == KEY_RALT || ev->data1 == KEY_LALT)
{
    altdown = ev->type == ev_keydown;
    return false;
}

if (ev->type != ev_keydown)
    return false;

if (!chat_on)
{
    if (ev->data1 == HU_MSGREFRESH)
    {
        message_on = true;
        message_counter = HU_MSGTIMEOUT;
        eatkey = true;
    }
    else if (netgame && ev->data1 == HU_INPUTTOGGLE)
    {
        eatkey = chat_on = true;
        HUlib_resetIText(&w_chat);
        HU_queueChatChar(HU_BROADCAST);
    }
    else if (netgame && numplayers > 2)
    {
        for (i=0; i<MAXPLAYERS ; i++)
        {
            if (ev->data1 == destination_keys[i])
            {
                if (playeringame[i] && i!=consoleplayer)
                {
                    eatkey = chat_on = true;
                    HUlib_resetIText(&w_chat);
                    HU_queueChatChar(i+1);
                    break;
                }
            }
            else if (i == consoleplayer)
            {
                num_nobrainers++;
                if (num_nobrainers < 3)

```

```

        plr->message = HUSTR_TALKTOSELF1;
    else if (num_nobrainers < 6)
        plr->message = HUSTR_TALKTOSELF2;
    else if (num_nobrainers < 9)
        plr->message = HUSTR_TALKTOSELF3;
    else if (num_nobrainers < 32)
        plr->message = HUSTR_TALKTOSELF4;
    else
        plr->message = HUSTR_TALKTOSELF5;
    }
}
}
}
else
{
    c = ev->data1;
    // send a macro
    if (altdown)
    {
        c = c - '0';
        if (c > 9)
            return false;
        // fprintf(stderr, "got here\n");
        macromessage = chat_macros[c];

        // kill last message with a '\n'
        HU_queueChatChar(KEY_ENTER); // DEBUG!!!

        // send the macro message
        while (*macromessage)
            HU_queueChatChar(*macromessage++);
        HU_queueChatChar(KEY_ENTER);

        // leave chat mode and notify that it was sent
        chat_on = false;
        strcpy(lastmessage, chat_macros[c]);
        plr->message = lastmessage;
        eatkey = true;
    }
    else
    {
        if (french)
            c = ForeignTranslation(c);
        if (shiftdown || (c >= 'a' && c <= 'z'))
            c = shiftxform[c];
        eatkey = HUlib_keyInIText(&w_chat, c);
        if (eatkey)
        {
            // static unsigned char buf[20]; // DEBUG
            HU_queueChatChar(c);

            // sprintf(buf, "KEY: %d => %d", ev->data1, c);
            // plr->message = buf;
        }
        if (c == KEY_ENTER)
        {
            chat_on = false;
            if (w_chat.l.len)
            {
                strcpy(lastmessage, w_chat.l.l);
                plr->message = lastmessage;
            }
        }
        else if (c == KEY_ESCAPE)

```

```

        chat_on = false;
    }
}

return eatkey;
}

```

## 6.4 hu\_stuff.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:  Head up display
//-----

#ifndef __HU_STUFF_H__
#define __HU_STUFF_H__

#include "d_event.h"

//
// Globally visible constants.
//
#define HU_FONTSTART    '!'          // the first font characters
#define HU_FONTEND      '_'          // the last font characters

// Calculate # of glyphs in font.
#define HU_FONTSIZE      (HU_FONTEND - HU_FONTSTART + 1)

#define HU_BROADCAST      5

#define HU_MSGREFRESH      KEY_ENTER
#define HU_MSGX            0
#define HU_MSGY            0
#define HU_MSGWIDTH        64        // in characters
#define HU_MSGHEIGHT        1        // in lines

#define HU_MSGTIMEOUT      (4*TICRATE)

//
// HEADS UP TEXT
//

void HU_Init(void);
void HU_Start(void);

boolean HU_Responder(event_t* ev);

```

```

void HU_Ticker(void);
void HU_Drawer(void);
char HU_dequeueChatChar(void);
void HU_Erase(void);

```

```

#endif

```

```

//-----
//
// $Log:$
//
//-----

```

## 7 System-specific code

### 7.1 i\_main.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Main program, simply calls D_DoomMain high level loop.
//
//-----

```

```

static const char
rcsid[] = "$Id: i_main.c,v 1.4 1997/02/03 22:45:10 b1 Exp $";

```

```

#include "doomdef.h"

```

```

#include "m_argv.h"

```

```

#include "d_main.h"

```

```

int
main
( int      argc,
  char**   argv )
{
    myargc = argc;
    myargv = argv;

    D_DoomMain ();

    return 0;
}

```

## 7.2 inet.c

```
// Emacs style mode select    -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//-----

static const char
rcsid[] = "$Id: m_bbox.c,v 1.1 1997/02/03 22:45:10 b1 Exp $";

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <unistd.h>
#include <netdb.h>
#include <sys/ioctl.h>

#include "i_system.h"
#include "d_event.h"
#include "d_net.h"
#include "m_argv.h"

#include "doomstat.h"

#ifdef __GNUG__
#pragma implementation "i_net.h"
#endif
#include "i_net.h"

// For some odd reason...
#define ntohl(x) \
    (((unsigned long int)((((unsigned long int)(x) & 0x000000ffU) << 24) | \
        (((unsigned long int)(x) & 0x0000ff00U) << 8) | \
        (((unsigned long int)(x) & 0x00ff0000U) >> 8) | \
        (((unsigned long int)(x) & 0xff000000U) >> 24)))

#define ntohs(x) \
    (((unsigned short int)((((unsigned short int)(x) & 0x00ff) << 8) | \
        (((unsigned short int)(x) & 0xff00) >> 8)))) \
```

```

#define htonl(x) ntohl(x)
#define htons(x) ntohs(x)

void      NetSend (void);
boolean NetListen (void);

//
// NETWORKING
//

int      DOOMPORT =      (IPPORT_USERRESERVED +0x1d );

int      sendsocket;
int      insocket;

struct    sockaddr_in      sendaddress[MAXNETNODES];

void      (*netget) (void);
void      (*netsend) (void);

//
// UDPsocket
//
int UDPsocket (void)
{
    int      s;

    // allocate a socket
    s = socket (PF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (s<0)
        I_Error ("can't create socket: %s",strerror(errno));

    return s;
}

//
// BindToLocalPort
//
void
BindToLocalPort
( int      s,
  int      port )
{
    int      v;
    struct sockaddr_in      address;

    memset (&address, 0, sizeof(address));
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = port;

    v = bind (s, (void *)&address, sizeof(address));
    if (v == -1)
        I_Error ("BindToPort: bind: %s", strerror(errno));
}

//
// PacketSend
//
void PacketSend (void)
{

```

```

int                c;
doomdata_t        sw;

// byte swap
sw.checksum = htonl(netbuffer->checksum);
sw.player = netbuffer->player;
sw.retransmitfrom = netbuffer->retransmitfrom;
sw.starttic = netbuffer->starttic;
sw.numtics = netbuffer->numtics;
for (c=0 ; c< netbuffer->numtics ; c++)
{
    sw.cmds[c].forwardmove = netbuffer->cmds[c].forwardmove;
    sw.cmds[c].sidemove = netbuffer->cmds[c].sidemove;
    sw.cmds[c].angleturn = htons(netbuffer->cmds[c].angleturn);
    sw.cmds[c].consistancy = htons(netbuffer->cmds[c].consistancy);
    sw.cmds[c].chatchar = netbuffer->cmds[c].chatchar;
    sw.cmds[c].buttons = netbuffer->cmds[c].buttons;
}

//printf ("sending %i\n",gametic);
c = sendto (sendsocket , &sw, doomcom->datalength
            ,0,(void *)&sendaddress[doomcom->remotenode]
            ,sizeof(sendaddress[doomcom->remotenode]));

//      if (c == -1)
//          I_Error ("SendPacket error: %s",strerror(errno));
}

//
// PacketGet
//
void PacketGet (void)
{
    int                i;
    int                c;
    struct sockaddr_in fromaddress;
    int                fromlen;
    doomdata_t        sw;

    fromlen = sizeof(fromaddress);
    c = recvfrom (insocket, &sw, sizeof(sw), 0
                  , (struct sockaddr *)&fromaddress, &fromlen );
    if (c == -1 )
    {
        if (errno != EWOULDBLOCK)
            I_Error ("GetPacket: %s",strerror(errno));
        doomcom->remotenode = -1;          // no packet
        return;
    }

    {
        static int first=1;
        if (first)
            printf("len=%d:p=[0x%x 0x%x] \n", c, *(int*)&sw, *((int*)&sw+1));
        first = 0;
    }

    // find remote node number
    for (i=0 ; i<doomcom->numnodes ; i++)
        if ( fromaddress.sin_addr.s_addr == sendaddress[i].sin_addr.s_addr )
            break;

    if (i == doomcom->numnodes)
    {

```



```

    // packet is not from one of the players (new game broadcast)
    doomcom->remotenode = -1;          // no packet
    return;
}

doomcom->remotenode = i;               // good packet from a game player
doomcom->datalength = c;

// byte swap
netbuffer->checksum = ntohl(sw.checksum);
netbuffer->player = sw.player;
netbuffer->retransmitfrom = sw.retransmitfrom;
netbuffer->starttic = sw.starttic;
netbuffer->numtics = sw.numtics;

for (c=0 ; c< netbuffer->numtics ; c++)
{
    netbuffer->cmds[c].forwardmove = sw.cmds[c].forwardmove;
    netbuffer->cmds[c].sidemove = sw.cmds[c].sidemove;
    netbuffer->cmds[c].angleturn = ntohs(sw.cmds[c].angleturn);
    netbuffer->cmds[c].consistency = ntohs(sw.cmds[c].consistency);
    netbuffer->cmds[c].chatchar = sw.cmds[c].chatchar;
    netbuffer->cmds[c].buttons = sw.cmds[c].buttons;
}
}

int GetLocalAddress (void)
{
    char                hostname[1024];
    struct hostent*     hostentry;      // host information entry
    int                 v;

    // get local address
    v = gethostname (hostname, sizeof(hostname));
    if (v == -1)
        I_Error ("GetLocalAddress : gethostname: errno %d",errno);

    hostentry = gethostbyname (hostname);
    if (!hostentry)
        I_Error ("GetLocalAddress : gethostbyname: couldn't get local host");

    return *(int *)hostentry->h_addr_list[0];
}

//
// I_InitNetwork
//
void I_InitNetwork (void)
{
    boolean             trueval = true;
    int                 i;
    int                 p;
    struct hostent*     hostentry;      // host information entry

    doomcom = malloc (sizeof (*doomcom) );
    memset (doomcom, 0, sizeof(*doomcom) );

    // set up for network
    i = M_CheckParm ("-dup");
    if (i && i< myargc-1)
    {
        doomcom->ticdup = myargv[i+1][0]-'0';
    }
}

```

```

        if (doomcom->ticdup < 1)
            doomcom->ticdup = 1;
        if (doomcom->ticdup > 9)
            doomcom->ticdup = 9;
    }
    else
        doomcom-> ticdup = 1;

    if (M_CheckParm ("-extratic"))
        doomcom-> extratics = 1;
    else
        doomcom-> extratics = 0;

    p = M_CheckParm ("-port");
    if (p && p<myargc-1)
    {
        DOOMPORT = atoi (myargv[p+1]);
        printf ("using alternate port %i\n",DOOMPORT);
    }

    // parse network game options,
    // -net <consoleplayer> <host> <host> ...
    i = M_CheckParm ("-net");
    if (!i)
    {
        // single player game
        netgame = false;
        doomcom->id = DOOMCOM_ID;
        doomcom->numplayers = doomcom->numnodes = 1;
        doomcom->deathmatch = false;
        doomcom->consoleplayer = 0;
        return;
    }

    netsend = PacketSend;
    netget = PacketGet;
    netgame = true;

    // parse player number and host list
    doomcom->consoleplayer = myargv[i+1][0]-'1';

    doomcom->numnodes = 1;          // this node for sure

    i++;
    while (++i < myargc && myargv[i][0] != '-')
    {
        sendaddress[doomcom->numnodes].sin_family = AF_INET;
        sendaddress[doomcom->numnodes].sin_port = htons(DOOMPORT);
        if (myargv[i][0] == '.')
        {
            sendaddress[doomcom->numnodes].sin_addr.s_addr
                = inet_addr (myargv[i+1]);
        }
        else
        {
            hostentry = gethostbyname (myargv[i]);
            if (!hostentry)
                I_Error ("gethostbyname: couldn't find %s", myargv[i]);
            sendaddress[doomcom->numnodes].sin_addr.s_addr
                = *(int *)hostentry->h_addr_list[0];
        }
        doomcom->numnodes++;
    }

    doomcom->id = DOOMCOM_ID;

```

```

doomcom->numplayers = doomcom->numnodes;

// build message to receive
insocket = UDPsocket ();
BindToLocalPort (insocket,htons(DOOMPORT));
ioctl (insocket, FIONBIO, &trueval);

sendsocket = UDPsocket ();
}

void I_NetCmd (void)
{
    if (doomcom->command == CMD_SEND)
    {
        netsend ();
    }
    else if (doomcom->command == CMD_GET)
    {
        netget ();
    }
    else
        I_Error ("Bad net cmd: %i\n",doomcom->command);
}

```

### 7.3 i.net.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      System specific network interface stuff.
//-----

#ifndef __I_NET__
#define __I_NET__

#ifdef __GNUG__
#pragma interface
#endif

// Called by D_DoomMain.

void I_InitNetwork (void);

```

```
void I_NetCmd (void);
```

```
#endif
```

```
//-----  
//  
// $Log:$  
//  
//-----
```

## 7.4 i\_sound.c

```
// Emacs style mode select  -*- C++ -*-  
//-----  
//  
// $Id:$  
//  
// Copyright (C) 1993-1996 by id Software, Inc.  
//  
// This program is free software; you can redistribute it and/or  
// modify it under the terms of the GNU General Public License  
// as published by the Free Software Foundation; either version 2  
// of the License, or (at your option) any later version.  
//  
// This program is distributed in the hope that it will be useful,  
// but WITHOUT ANY WARRANTY; without even the implied warranty of  
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
// GNU General Public License for more details.  
//  
// $Log:$  
//  
// DESCRIPTION:  
//      System interface for sound.  
//  
//-----  
  
static const char  
rcsid[] = "$Id: i_unix.c,v 1.5 1997/02/03 22:45:10 b1 Exp $";  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <stdarg.h>  
  
#include <math.h>  
  
#include <sys/time.h>  
#include <sys/types.h>  
  
#ifndef LINUX  
#include <sys/filio.h>  
#endif  
  
#include <fcntl.h>  
#include <unistd.h>  
#include <sys/ioctl.h>  
  
// Linux voxware output.  
#include <linux/soundcard.h>  
  
// Timer stuff. Experimental.  
#include <time.h>  
#include <signal.h>  
  
#include "z_zone.h"
```

```

#include "i_system.h"
#include "i_sound.h"
#include "m_argv.h"
#include "m_misc.h"
#include "w_wad.h"

#include "doomdef.h"

// UNIX hack, to be removed.
#ifdef SNDSESV
// Separate sound server process.
FILE*      sndserver=0;
char*      sndserver_filename = "./sndserver ";
#elif SNDINTR

// Update all 30 millisecs, approx. 30fps synchronized.
// Linux resolution is allegedly 10 millisecs,
// scale is microseconds.
#define SOUND_INTERVAL      500

// Get the interrupt. Set duration in millisecs.
int I_SoundSetTimer( int duration_of_tick );
void I_SoundDelTimer( void );
#else
// None?
#endif

// A quick hack to establish a protocol between
// synchronous mix buffer updates and asynchronous
// audio writes. Probably redundant with gametic.
static int flag = 0;

// The number of internal mixing channels,
// the samples calculated for each mixing step,
// the size of the 16bit, 2 hardware channel (stereo)
// mixing buffer, and the samplerate of the raw data.

// Needed for calling the actual sound output.
#define SAMPLECOUNT      512
#define NUM_CHANNELS      8
// It is 2 for 16bit, and 2 for two channels.
#define BUFMUL            4
#define MIXBUFFERSIZE      (SAMPLECOUNT*BUFMUL)

#define SAMPLERATE        11025      // Hz
#define SAMPLESIZE        2          // 16bit

// The actual lengths of all sound effects.
int      lengths[NUMSFX];

// The actual output device.
int      audio_fd;

// The global mixing buffer.
// Basically, samples from all active internal channels
// are modified and added, and stored in the buffer
// that is submitted to the audio device.
signed short      mixbuffer[MIXBUFFERSIZE];

// The channel step amount...
unsigned int      channelstep[NUM_CHANNELS];
// ... and a 0.16 bit remainder of last step.

```

```

unsigned int          channelstepremainder[NUM_CHANNELS];

// The channel data pointers, start and end.
unsigned char*        channels[NUM_CHANNELS];
unsigned char*        channelsend[NUM_CHANNELS];

// Time/gametic that the channel started playing,
// used to determine oldest, which automatically
// has lowest priority.
// In case number of active sounds exceeds
// available channels.
int                   channelstart[NUM_CHANNELS];

// The sound in channel handles,
// determined on registration,
// might be used to unregister/stop/modify,
// currently unused.
int                   channelhandles[NUM_CHANNELS];

// SFX id of the playing sound effect.
// Used to catch duplicates (like chainsaw).
int                   channelids[NUM_CHANNELS];

// Pitch to stepping lookup, unused.
int                   steptable[256];

// Volume lookups.
int                   vol_lookup[128*256];

// Hardware left and right channel volume lookup.
int*                  channellftvol_lookup[NUM_CHANNELS];
int*                  channelrightvol_lookup[NUM_CHANNELS];

//
// Safe ioctl, convenience.
//
void
myioctl
( int          fd,
  int          command,
  int*         arg )
{
    int          rc;
    extern int   errno;

    rc = ioctl(fd, command, arg);
    if (rc < 0)
    {
        fprintf(stderr, "ioctl(dsp,%d,arg) failed\n", command);
        fprintf(stderr, "errno=%d\n", errno);
        exit(-1);
    }
}

//
// This function loads the sound data from the WAD lump,

```

```

// for single sound.
//
void*
getsfx
( char*      sfxname,
  int*      len )
{
    unsigned char*      sfx;
    unsigned char*      padded_sfx;
    int                 i;
    int                 size;
    int                 paddedsize;
    char                name[20];
    int                 sfxlump;

    // Get the sound data from the WAD, allocate lump
    // in zone memory.
    sprintf(name, "ds%s", sfxname);

    // Now, there is a severe problem with the
    // sound handling, in it is not (yet/anymore)
    // gamemode aware. That means, sounds from
    // DOOM II will be requested even with DOOM
    // shareware.
    // The sound list is wired into sounds.c,
    // which sets the external variable.
    // I do not do runtime patches to that
    // variable. Instead, we will use a
    // default sound for replacement.
    if ( W_CheckNumForName(name) == -1 )
        sfxlump = W_GetNumForName("dspistol");
    else
        sfxlump = W_GetNumForName(name);

    size = W_LumpLength( sfxlump );

    // Debug.
    // fprintf( stderr, "." );
    // fprintf( stderr, "-loading %s (lump %d, %d bytes)\n",
    //          sfxname, sfxlump, size );
    // fflush( stderr );

    sfx = (unsigned char*)W_CacheLumpNum( sfxlump, PU_STATIC );

    // Pads the sound effect out to the mixing buffer size.
    // The original realloc would interfere with zone memory.
    paddedsize = ((size-8 + (SAMPLECOUNT-1)) / SAMPLECOUNT) * SAMPLECOUNT;

    // Allocate from zone memory.
    padded_sfx = (unsigned char*)Z_Malloc( paddedsize+8, PU_STATIC, 0 );
    // ddt: (unsigned char *) realloc(sfx, paddedsize+8);
    // This should interfere with zone memory handling,
    // which does not kick in in the soundserver.

    // Now copy and pad.
    memcpy( padded_sfx, sfx, size );
    for (i=size ; i<paddedsize+8 ; i++)
        padded_sfx[i] = 128;

    // Remove the cached lump.
    Z_Free( sfx );

    // Preserve padded length.
    *len = paddedsize;

```

```

    // Return allocated padded data.
    return (void *) (paddedsfx + 8);
}

//
// This function adds a sound to the
// list of currently active sounds,
// which is maintained as a given number
// (eight, usually) of internal channels.
// Returns a handle.
//
int
addsfx
( int          sfxid,
  int          volume,
  int          step,
  int          seperation )
{
    static unsigned short    handlenums = 0;

    int          i;
    int          rc = -1;

    int          oldest = gametic;
    int          oldestnum = 0;
    int          slot;

    int          rightvol;
    int          leftvol;

    // Chainsaw troubles.
    // Play these sound effects only one at a time.
    if ( sfxid == sfx_sawup
        || sfxid == sfx_sawidl
        || sfxid == sfx_sawful
        || sfxid == sfx_sawhit
        || sfxid == sfx_stnmov
        || sfxid == sfx_pistol
        )
    {
        // Loop all channels, check.
        for (i=0 ; i<NUM_CHANNELS ; i++)
        {
            // Active, and using the same SFX?
            if ( (channels[i])
                && (channelids[i] == sfxid) )
            {
                // Reset.
                channels[i] = 0;
                // We are sure that iff,
                // there will only be one.
                break;
            }
        }
    }

    // Loop all channels to find oldest SFX.
    for (i=0; (i<NUM_CHANNELS) && (channels[i]); i++)
    {
        if (channelstart[i] < oldest)
        {

```



```

        oldestnum = i;
        oldest = channelstart[i];
    }
}

// Tales from the cryptic.
// If we found a channel, fine.
// If not, we simply overwrite the first one, 0.
// Probably only happens at startup.
if (i == NUM_CHANNELS)
    slot = oldestnum;
else
    slot = i;

// Okay, in the less recent channel,
// we will handle the new SFX.
// Set pointer to raw data.
channels[slot] = (unsigned char *) S_sfx[sfxid].data;
// Set pointer to end of raw data.
channelsend[slot] = channels[slot] + lengths[sfxid];

// Reset current handle number, limited to 0..100.
if (!handlenums)
    handlenums = 100;

// Assign current handle number.
// Preserved so sounds could be stopped (unused).
channelhandles[slot] = rc = handlenums++;

// Set stepping???
// Kinda getting the impression this is never used.
channelstep[slot] = step;
// ???
channelstepremainder[slot] = 0;
// Should be gametic, I presume.
channelstart[slot] = gametic;

// Separation, that is, orientation/stereo.
// range is: 1 - 256
seperation += 1;

// Per left/right channel.
// x^2 seperation,
// adjust volume properly.
leftvol =
    volume - ((volume*seperation*seperation) >> 16); //(256*256);
seperation = seperation - 257;
rightvol =
    volume - ((volume*seperation*seperation) >> 16);

// Sanity check, clamp volume.
if (rightvol < 0 || rightvol > 127)
    I_Error("rightvol out of bounds");

if (leftvol < 0 || leftvol > 127)
    I_Error("leftvol out of bounds");

// Get the proper lookup table piece
// for this volume level???
channelleftvol_lookup[slot] = &vol_lookup[leftvol*256];
channelrightvol_lookup[slot] = &vol_lookup[rightvol*256];

// Preserve sound SFX id,
// e.g. for avoiding duplicates of chainsaw.
channelids[slot] = sfxid;

```

```

    // You tell me.
    return rc;
}

//
// SFX API
// Note: this was called by S_Init.
// However, whatever they did in the
// old DPMS based DOS version, this
// were simply dummies in the Linux
// version.
// See soundserver initdata().
//
void I_SetChannels()
{
    // Init internal lookups (raw data, mixing buffer, channels).
    // This function sets up internal lookups used during
    // the mixing process.
    int          i;
    int          j;

    int*          steptablemid = steptable + 128;

    // Okay, reset internal mixing channels to zero.
    /*for (i=0; i<NUM_CHANNELS; i++)
    {
        channels[i] = 0;
    }*/

    // This table provides step widths for pitch parameters.
    // I fail to see that this is currently used.
    for (i=-128 ; i<128 ; i++)
        steptablemid[i] = (int)(pow(2.0, (i/64.0))*65536.0);

    // Generates volume lookup tables
    // which also turn the unsigned samples
    // into signed samples.
    for (i=0 ; i<128 ; i++)
        for (j=0 ; j<256 ; j++)
            vol_lookup[i*256+j] = (i*(j-128)*256)/127;
}

void I_SetSfxVolume(int volume)
{
    // Identical to DOS.
    // Basically, this should propagate
    // the menu/config file setting
    // to the state variable used in
    // the mixing.
    snd_SfxVolume = volume;
}

// MUSIC API - dummy. Some code from DOS version.
void I_SetMusicVolume(int volume)
{
    // Internal state variable.
    snd_MusicVolume = volume;
    // Now set volume on output device.

```

```

    // Whatever( snd_MusciVolume );
}

//
// Retrieve the raw data lump index
// for a given SFX name.
//
int I_GetSfxLumpNum(sfxinfo_t* sfx)
{
    char namebuf[9];
    sprintf(namebuf, "ds%s", sfx->name);
    return W_GetNumForName(namebuf);
}

//
// Starting a sound means adding it
// to the current list of active sounds
// in the internal channels.
// As the SFX info struct contains
// e.g. a pointer to the raw data,
// it is ignored.
// As our sound handling does not handle
// priority, it is ignored.
// Pitching (that is, increased speed of playback)
// is set, but currently not used by mixing.
//
int
I_StartSound
( int          id,
  int          vol,
  int          sep,
  int          pitch,
  int          priority )
{
    // UNUSED
    priority = 0;

#ifdef SNDSESV
    if (sndserver)
    {
        fprintf(sndserver, "p%2.2x%2.2x%2.2x%2.2x\n", id, pitch, vol, sep);
        fflush(sndserver);
    }
    // warning: control reaches end of non-void function.
    return id;
#else
    // Debug.
    //fprintf( stderr, "starting sound %d", id );

    // Returns a handle (not used).
    id = addsfx( id, vol, steptable[pitch], sep );

    // fprintf( stderr, "/handle is %d\n", id );

    return id;
#endif
}

void I_StopSound (int handle)
{
    // You need the handle returned by StartSound.

```

```

// Would be looping all channels,
// tracking down the handle,
// an setting the channel to zero.

// UNUSED.
handle = 0;
}

int I_SoundIsPlaying(int handle)
{
    // Ouch.
    return gametic < handle;
}

//
// This function loops all active (internal) sound
// channels, retrieves a given number of samples
// from the raw sound data, modifies it according
// to the current (internal) channel parameters,
// mixes the per channel samples into the global
// mixbuffer, clamping it to the allowed range,
// and sets up everything for transferring the
// contents of the mixbuffer to the (two)
// hardware channels (left and right, that is).
//
// This function currently supports only 16bit.
//
void I_UpdateSound( void )
{
#ifdef SNDINTR
    // Debug. Count buffer misses with interrupt.
    static int misses = 0;
#endif

    // Mix current sound data.
    // Data, from raw sound, for right and left.
    register unsigned int    sample;
    register int             dl;
    register int             dr;

    // Pointers in global mixbuffer, left, right, end.
    signed short*            leftout;
    signed short*            rightout;
    signed short*            leftend;
    // Step in mixbuffer, left and right, thus two.
    int                      step;

    // Mixing channel index.
    int                      chan;

    // Left and right channel
    // are in global mixbuffer, alternating.
    leftout = mixbuffer;
    rightout = mixbuffer+1;
    step = 2;

    // Determine end, for left channel only
    // (right channel is implicit).
    leftend = mixbuffer + SAMPLECOUNT*step;

```

```

// Mix sounds into the mixing buffer.
// Loop over step*SAMPLECOUNT,
// that is 512 values for two channels.
while (leftout != leftend)
{
    // Reset left/right value.
    dl = 0;
    dr = 0;

    // Love thy L2 chache - made this a loop.
    // Now more channels could be set at compile time
    // as well. Thus loop those channels.
    for ( chan = 0; chan < NUM_CHANNELS; chan++ )
    {
        // Check channel, if active.
        if (channels[ chan ])
        {
            // Get the raw data from the channel.
            sample = *channels[ chan ];
            // Add left and right part
            // for this channel (sound)
            // to the current data.
            // Adjust volume accordingly.
            dl += channellleftvol_lookup[ chan ][sample];
            dr += channelrightvol_lookup[ chan ][sample];
            // Increment index ???
            channelstepremainder[ chan ] += channelstep[ chan ];
            // MSB is next sample???
            channels[ chan ] += channelstepremainder[ chan ] >> 16;
            // Limit to LSB???
            channelstepremainder[ chan ] &= 65536-1;

            // Check whether we are done.
            if (channels[ chan ] >= channelsend[ chan ])
                channels[ chan ] = 0;
        }
    }

    // Clamp to range. Left hardware channel.
    // Has been char instead of short.
    // if (dl > 127) *leftout = 127;
    // else if (dl < -128) *leftout = -128;
    // else *leftout = dl;

    if (dl > 0x7fff)
        *leftout = 0x7fff;
    else if (dl < -0x8000)
        *leftout = -0x8000;
    else
        *leftout = dl;

    // Same for right hardware channel.
    if (dr > 0x7fff)
        *rightout = 0x7fff;
    else if (dr < -0x8000)
        *rightout = -0x8000;
    else
        *rightout = dr;

    // Increment current pointers in mixbuffer.
    leftout += step;
    rightout += step;
}

```

```

#ifdef SNDINTR

```

```

    // Debug check.
    if ( flag )
    {
        misses += flag;
        flag = 0;
    }

    if ( misses > 10 )
    {
        fprintf( stderr, "I_SoundUpdate: missed 10 buffer writes\n");
        misses = 0;
    }

    // Increment flag for update.
    flag++;
#endif
}

//
// This would be used to write out the mixbuffer
// during each game loop update.
// Updates sound buffer and audio device at runtime.
// It is called during Timer interrupt with SNDINTR.
// Mixing now done synchronous, and
// only output be done asynchronous?
//
void
I_SubmitSound(void)
{
    // Write it to DSP device.
    write(audio_fd, mixbuffer, SAMPLECOUNT*BUFMUL);
}

void
I_UpdateSoundParams
( int      handle,
  int      vol,
  int      sep,
  int      pitch)
{
    // I fail too see that this is used.
    // Would be using the handle to identify
    // on which channel the sound might be active,
    // and resetting the channel parameters.

    // UNUSED.
    handle = vol = sep = pitch = 0;
}

void I_ShutdownSound(void)
{
#ifdef SNDSESV
    if (sndserver)
    {
        // Send a "quit" command.
        fprintf(sndserver, "q\n");
        fflush(sndserver);
    }
#else

```

```

// Wait till all pending sounds are finished.
int done = 0;
int i;

// FIXME (below).
fprintf( stderr, "I_ShutdownSound: NOT finishing pending sounds\n");
fflush( stderr );

while ( !done )
{
    for( i=0 ; i<8 && !channels[i] ; i++);

    // FIXME. No proper channel output.
    //if (i==8)
    done=1;
}
#ifdef SNDINTR
    I_SoundDelTimer();
#endif

    // Cleaning up -releasing the DSP device.
    close ( audio_fd );
#endif

    // Done.
    return;
}

```

```

void
I_InitSound()
{
#ifdef SNDSESV
    char buffer[256];

    if (getenv("DOOMWADDIR"))
        sprintf(buffer, "%s/%s",
                getenv("DOOMWADDIR"),
                sndserver_filename);
    else
        sprintf(buffer, "%s", sndserver_filename);

    // start sound process
    if ( !access(buffer, X_OK) )
    {
        strcat(buffer, " -quiet");
        sndserver = popen(buffer, "w");
    }
    else
        fprintf(stderr, "Could not start sound server [%s]\n", buffer);
#else
    int i;

#ifdef SNDINTR
        fprintf( stderr, "I_SoundSetTimer: %d microsecs\n", SOUND_INTERVAL );
        I_SoundSetTimer( SOUND_INTERVAL );
#endif

        // Secure and configure sound device first.

```

```

fprintf( stderr, "I_InitSound: ");

audio_fd = open("/dev/dsp", O_WRONLY);
if (audio_fd<0)
    fprintf(stderr, "Could not open /dev/dsp\n");

i = 11 | (2<<16);
myioctl(audio_fd, SNDCTL_DSP_SETFRAGMENT, &i);
myioctl(audio_fd, SNDCTL_DSP_RESET, 0);

i=SAMPLERATE;

myioctl(audio_fd, SNDCTL_DSP_SPEED, &i);

i=1;
myioctl(audio_fd, SNDCTL_DSP_STEREO, &i);

myioctl(audio_fd, SNDCTL_DSP_GETFMTS, &i);

if (i&=AFMT_S16_LE)
    myioctl(audio_fd, SNDCTL_DSP_SETFMT, &i);
else
    fprintf(stderr, "Could not play signed 16 data\n");

fprintf(stderr, " configured audio device\n" );

// Initialize external data (all sounds) at start, keep static.
fprintf( stderr, "I_InitSound: ");

for (i=1 ; i<NUMSFX ; i++)
{
    // Alias? Example is the chaingun sound linked to pistol.
    if (!S_sfx[i].link)
    {
        // Load data from WAD file.
        S_sfx[i].data = getsfx( S_sfx[i].name, &lengths[i] );
    }
    else
    {
        // Previously loaded already?
        S_sfx[i].data = S_sfx[i].link->data;
        lengths[i] = lengths[(S_sfx[i].link - S_sfx)/sizeof(sfxinfo_t)];
    }
}

fprintf( stderr, " pre-cached all sound data\n");

// Now initialize mixbuffer with zero.
for ( i = 0; i< MIXBUFFERSIZE; i++ )
    mixbuffer[i] = 0;

// Finished initialization.
fprintf(stderr, "I_InitSound: sound module ready\n");

#endif
}

//
// MUSIC API.
// Still no music done.

```



```

// Remains. Dummies.
//
void I_InitMusic(void)          { }
void I_ShutdownMusic(void)     { }

static int      looping=0;
static int      musicdies=-1;

void I_PlaySong(int handle, int looping)
{
    // UNUSED.
    handle = looping = 0;
    musicdies = gametic + TICRATE*30;
}

void I_PauseSong (int handle)
{
    // UNUSED.
    handle = 0;
}

void I_ResumeSong (int handle)
{
    // UNUSED.
    handle = 0;
}

void I_StopSong(int handle)
{
    // UNUSED.
    handle = 0;

    looping = 0;
    musicdies = 0;
}

void I_UnRegisterSong(int handle)
{
    // UNUSED.
    handle = 0;
}

int I_RegisterSong(void* data)
{
    // UNUSED.
    data = NULL;

    return 1;
}

// Is the song playing?
int I_QrySongPlaying(int handle)
{
    // UNUSED.
    handle = 0;
    return looping || musicdies > gametic;
}

//
// Experimental stuff.
// A Linux timer interrupt, for asynchronous
// sound output.
// I ripped this out of the Timer class in

```

```

// our Difference Engine, including a few
// SUN remains...
//
#ifdef sun
    typedef      sigset_t      tSigSet;
#else
    typedef      int           tSigSet;
#endif

// We might use SIGVTALRM and ITIMER_VIRTUAL, if the process
// time independend timer happens to get lost due to heavy load.
// SIGALRM and ITIMER_REAL doesn't really work well.
// There are issues with profiling as well.
static int /*__itimer_which*/ itimer = ITIMER_REAL;

static int sig = SIGALRM;

// Interrupt handler.
void I_HandleSoundTimer( int ignore )
{
    // Debug.
    //fprintf( stderr, "%c", '+' ); fflush( stderr );

    // Feed sound device if necesary.
    if ( flag )
    {
        // See I_SubmitSound().
        // Write it to DSP device.
        write(audio_fd, mixbuffer, SAMPLECOUNT*BUFMUL);

        // Reset flag counter.
        flag = 0;
    }
    else
        return;

    // UNUSED, but required.
    ignore = 0;
    return;
}

// Get the interrupt. Set duration in millisecs.
int I_SoundSetTimer( int duration_of_tick )
{
    // Needed for gametick clockwork.
    struct itimerval    value;
    struct itimerval    ovalue;
    struct sigaction    act;
    struct sigaction    oact;

    int res;

    // This sets to SA_ONESHOT and SA_NOMASK, thus we can not use it.
    //      signal( _sig, handle_SIG_TICK );

    // Now we have to change this attribute for repeated calls.
    act.sa_handler = I_HandleSoundTimer;
#ifdef sun
    //ac      t.sa_mask = _sig;
#endif
    act.sa_flags = SA_RESTART;

    sigaction( sig, &act, &oact );

```

```

value.it_interval.tv_sec      = 0;
value.it_interval.tv_usec    = duration_of_tick;
value.it_value.tv_sec        = 0;
value.it_value.tv_usec       = duration_of_tick;

// Error is -1.
res = setitimer( itimer, &value, &ovalue );

// Debug.
if ( res == -1 )
    fprintf( stderr, "I_SoundSetTimer: interrupt n.a.\n");

return res;
}

// Remove the interrupt. Set duration to zero.
void I_SoundDelTimer()
{
    // Debug.
    if ( I_SoundSetTimer( 0 ) == -1)
        fprintf( stderr, "I_SoundDelTimer: failed to remove interrupt. Doh!\n");
}

```

## 7.5 i\_sound.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
//
// DESCRIPTION:
//      System interface, sound.
//
//-----

#ifndef __I_SOUND__
#define __I_SOUND__

#include "doomdef.h"

// UNIX hack, to be removed.
#ifdef SNDSESV
#include <stdio.h>
extern FILE* sndserver;
extern char* sndserver_filename;
#endif

#include "doomstat.h"
#include "sounds.h"

```

```

// Init at program start...
void I_InitSound();

// ... update sound buffer and audio device at runtime...
void I_UpdateSound(void);
void I_SubmitSound(void);

// ... shut down and relase at program termination.
void I_ShutdownSound(void);


//
//  SFX I/O
//

// Initialize channels?
void I_SetChannels();

// Get raw data lump index for sound descriptor.
int I_GetSfxLumpNum (sfxinfo_t* sfxinfo );


// Starts a sound in a particular sound channel.
int
I_StartSound
( int          id,
  int          vol,
  int          sep,
  int          pitch,
  int          priority );


// Stops a sound channel.
void I_StopSound(int handle);


// Called by S_*( ) functions
// to see if a channel is still playing.
// Returns 0 if no longer playing, 1 if playing.
int I_SoundIsPlaying(int handle);


// Updates the volume, separation,
// and pitch of a sound channel.
void
I_UpdateSoundParams
( int          handle,
  int          vol,
  int          sep,
  int          pitch );


//
//  MUSIC I/O
//

void I_InitMusic(void);
void I_ShutdownMusic(void);
// Volume.
void I_SetMusicVolume(int volume);
// PAUSE game handling.
void I_PauseSong(int handle);
void I_ResumeSong(int handle);
// Registers a song handle to song data.
int I_RegisterSong(void *data);
// Called by anything that wishes to start music.
// plays a song, and when the song is done,

```

```

// starts playing it again in an endless loop.
// Horrible thing to do, considering.
void
I_PlaySong
( int          handle,
  int          looping );
// Stops a song over 3 seconds.
void I_StopSong(int handle);
// See above (register), then think backwards
void I_UnRegisterSong(int handle);

#endif
//-----
//
// $Log:$
//
//-----

```

## 7.6 i\_system.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//
//-----

static const char
rcsid[] = "$Id: m_bbox.c,v 1.1 1997/02/03 22:45:10 b1 Exp $";

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <stdarg.h>
#include <sys/time.h>
#include <unistd.h>

#include "doomdef.h"
#include "m_misc.h"
#include "i_video.h"
#include "i_sound.h"

#include "d_net.h"
#include "g_game.h"

```

```

#ifdef __GNUG__
#pragma implementation "i_system.h"
#endif
#include "i_system.h"

int      mb_used = 6;

void
I_Tactile
( int      on,
  int      off,
  int      total )
{
    // UNUSED.
    on = off = total = 0;
}

ticcmd_t      emptycmd;
ticcmd_t*     I_BaseTiccmd(void)
{
    return &emptycmd;
}

int I_GetHeapSize (void)
{
    return mb_used*1024*1024;
}

byte* I_ZoneBase (int*      size)
{
    *size = mb_used*1024*1024;
    return (byte *) malloc (*size);
}

//
// I_GetTime
// returns time in 1/70th second tics
//
int I_GetTime (void)
{
    struct timeval      tp;
    struct timezone      tzp;
    int                  newtics;
    static int           basetime=0;

    gettimeofday(&tp, &tzp);
    if (!basetime)
        basetime = tp.tv_sec;
    newtics = (tp.tv_sec-basetime)*TICRATE + tp.tv_usec*TICRATE/1000000;
    return newtics;
}

//
// I_Init
//
void I_Init (void)

```

```

{
    I_InitSound();
    // I_InitGraphics();
}

//
// I_Quit
//
void I_Quit (void)
{
    D_QuitNetGame ();
    I_ShutdownSound();
    I_ShutdownMusic();
    M_SaveDefaults ();
    I_ShutdownGraphics();
    exit(0);
}

void I_WaitVBL(int count)
{
#ifdef SGI
    sginap(1);
#else
#ifdef SUN
    sleep(0);
#else
    usleep (count * (1000000/70) );
#endif
#endif
}

void I_BeginRead(void)
{
}

void I_EndRead(void)
{
}

byte*      I_AllocLow(int length)
{
    byte*      mem;

    mem = (byte *)malloc (length);
    memset (mem,0,length);
    return mem;
}

//
// I_Error
//
extern boolean demorecording;

void I_Error (char *error, ...)
{
    va_list      argptr;

    // Message first.
    va_start (argptr,error);
    fprintf (stderr, "Error: ");
    vfprintf (stderr,error,argptr);
    fprintf (stderr, "\n");
    va_end (argptr);
}

```

```

fflush( stderr );

// Shutdown. Here might be other errors.
if (demorecording)
    G_CheckDemoStatus();

D_QuitNetGame ();
I_ShutdownGraphics();

exit(-1);
}

```

## 7.7 i\_system.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      System specific interface stuff.
//-----

#ifndef __I_SYSTEM__
#define __I_SYSTEM__

#include "d_ticcmd.h"
#include "d_event.h"

#ifdef __GNUG__
#pragma interface
#endif

// Called by DoomMain.
void I_Init (void);

// Called by startup code
// to get the ammount of memory to malloc
// for the zone management.
byte*      I_ZoneBase (int *size);

// Called by D_DoomLoop,
// returns current time in tics.
int I_GetTime (void);

//
// Called by D_DoomLoop,
// called before processing any tics in a frame

```



```

// (just after displaying a frame).
// Time consuming synchronous operations
// are performed here (joystick reading).
// Can call D_PostEvent.
//
void I_StartFrame (void);

//
// Called by D_DoomLoop,
// called before processing each tic in a frame.
// Quick synchronous operations are performed here.
// Can call D_PostEvent.
void I_StartTic (void);

// Asynchronous interrupt functions should maintain private queues
// that are read by the synchronous functions
// to be converted into events.

// Either returns a null ticcmd,
// or calls a loadable driver to build it.
// This ticcmd will then be modified by the gameloop
// for normal input.
ticcmd_t* I_BaseTiccmd (void);

// Called by M_Responder when quit is selected.
// Clean exit, displays sell blurb.
void I_Quit (void);

// Allocates from low memory under dos,
// just mallocs under unix
byte* I_AllocLow (int length);

void I_Tactile (int on, int off, int total);

void I_Error (char *error, ...);

#endif
//-----
//
// $Log:$
//
//-----

```

## 7.8 i\_video.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

```

```

// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      DOOM graphics stuff for X11, UNIX.
//
//-----

static const char
rcsid[] = "$Id: i_x.c,v 1.6 1997/02/03 22:45:10 b1 Exp $";

#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>

#include <X11/extensions/XShm.h>
// Had to dig up XShm.c for this one.
// It is in the libXext, but not in the XFree86 headers.
#ifdef LINUX
int XShmGetEventBase( Display* dpy ); // problems with g++?
#endif

#include <stdarg.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>

#include <netinet/in.h>
#include <errno.h>
#include <signal.h>

#include "doomstat.h"
#include "i_system.h"
#include "v_video.h"
#include "m_argv.h"
#include "d_main.h"

#include "doomdef.h"

#define POINTER_WARP_COUNTDOWN      1

Display*      X_display=0;
Window        X_mainWindow;
Colormap      X_cmap;
Visual*       X_visual;
GC            X_gc;
XEvent        X_event;
int           X_screen;
XVisualInfo   X_visualinfo;
XImage*       image;
int           X_width;
int           X_height;

// MIT SHared Memory extension.
boolean       doShm;

XShmSegmentInfo X_shminfo;
int           X_shmeventtype;

```

```

// Fake mouse handling.
// This cannot work properly w/o DGA.
// Needs an invisible mouse cursor at least.
boolean      grabMouse;
int          doPointerWarp = POINTER_WARP_COUNTDOWN;

// Blocky mode,
// replace each 320x200 pixel with multiply*multiply pixels.
// According to Dave Taylor, it still is a bonehead thing
// to use ....
static int    multiply=1;

//
// Translates the key currently in X_event
//

int xlatekey(void)
{
    int rc;

    switch(rc = XKeycodeToKeysym(X_display, X_event.xkey.keycode, 0))
    {
        case XK_Left:      rc = KEY_LEFTARROW;      break;
        case XK_Right:     rc = KEY_RIGHTARROW;     break;
        case XK_Down:      rc = KEY_DOWNARROW;      break;
        case XK_Up:       rc = KEY_UPARROW;         break;
        case XK_Escape:    rc = KEY_ESCAPE;         break;
        case XK_Return:    rc = KEY_ENTER;          break;
        case XK_Tab:       rc = KEY_TAB;            break;
        case XK_F1:        rc = KEY_F1;             break;
        case XK_F2:        rc = KEY_F2;             break;
        case XK_F3:        rc = KEY_F3;             break;
        case XK_F4:        rc = KEY_F4;             break;
        case XK_F5:        rc = KEY_F5;             break;
        case XK_F6:        rc = KEY_F6;             break;
        case XK_F7:        rc = KEY_F7;             break;
        case XK_F8:        rc = KEY_F8;             break;
        case XK_F9:        rc = KEY_F9;             break;
        case XK_F10:       rc = KEY_F10;            break;
        case XK_F11:       rc = KEY_F11;            break;
        case XK_F12:       rc = KEY_F12;            break;

        case XK_BackSpace:
        case XK_Delete:    rc = KEY_BACKSPACE;      break;

        case XK_Pause:     rc = KEY_PAUSE;          break;

        case XK_KP_Equal:
        case XK_equal:     rc = KEY_EQUALS;         break;

        case XK_KP_Subtract:
        case XK_minus:     rc = KEY_MINUS;          break;

        case XK_Shift_L:
        case XK_Shift_R:
            rc = KEY_RSHIFT;
            break;

        case XK_Control_L:
        case XK_Control_R:
            rc = KEY_CTRL;
            break;
    }
}

```

```

        case XK_Alt_L:
        case XK_Meta_L:
        case XK_Alt_R:
        case XK_Meta_R:
            rc = KEY_RALT;
            break;

        default:
            if (rc >= XK_space && rc <= XK_asciitilde)
                rc = rc - XK_space + ' ';
            if (rc >= 'A' && rc <= 'Z')
                rc = rc - 'A' + 'a';
            break;
    }

    return rc;
}

void I_ShutdownGraphics(void)
{
    // Detach from X server
    if (!XShmDetach(X_display, &X_shminfo))
        I_Error("XShmDetach() failed in I_ShutdownGraphics()");

    // Release shared memory.
    shmdt(X_shminfo.shmaddr);
    shmctl(X_shminfo.shmid, IPC_RMID, 0);

    // Paranoia.
    image->data = NULL;
}

//
// I_StartFrame
//
void I_StartFrame (void)
{
    // er?
}

static int      lastmousex = 0;
static int      lastmousey = 0;
boolean         mousemoved = false;
boolean         shmFinished;

void I_GetEvent(void)
{
    event_t event;

    // put event-grabbing stuff in here
    XNextEvent(X_display, &X_event);
    switch (X_event.type)
    {
        case KeyPress:
            event.type = ev_keydown;
            event.data1 = xlatekey();
            D_PostEvent(&event);
            // fprintf(stderr, "k");
            break;
        case KeyRelease:

```

```

    event.type = ev_keyup;
    event.data1 = xlatekey();
    D_PostEvent(&event);
    // fprintf(stderr, "ku");
    break;
case ButtonPress:
    event.type = ev_mouse;
    event.data1 =
        (X_event.xbutton.state & Button1Mask)
        | (X_event.xbutton.state & Button2Mask ? 2 : 0)
        | (X_event.xbutton.state & Button3Mask ? 4 : 0)
        | (X_event.xbutton.button == Button1)
        | (X_event.xbutton.button == Button2 ? 2 : 0)
        | (X_event.xbutton.button == Button3 ? 4 : 0);
    event.data2 = event.data3 = 0;
    D_PostEvent(&event);
    // fprintf(stderr, "b");
    break;
case ButtonRelease:
    event.type = ev_mouse;
    event.data1 =
        (X_event.xbutton.state & Button1Mask)
        | (X_event.xbutton.state & Button2Mask ? 2 : 0)
        | (X_event.xbutton.state & Button3Mask ? 4 : 0);
    // suggest parentheses around arithmetic in operand of |
    event.data1 =
        event.data1
        ^ (X_event.xbutton.button == Button1 ? 1 : 0)
        ^ (X_event.xbutton.button == Button2 ? 2 : 0)
        ^ (X_event.xbutton.button == Button3 ? 4 : 0);
    event.data2 = event.data3 = 0;
    D_PostEvent(&event);
    // fprintf(stderr, "bu");
    break;
case MotionNotify:
    event.type = ev_mouse;
    event.data1 =
        (X_event.xmotion.state & Button1Mask)
        | (X_event.xmotion.state & Button2Mask ? 2 : 0)
        | (X_event.xmotion.state & Button3Mask ? 4 : 0);
    event.data2 = (X_event.xmotion.x - lastmousex) << 2;
    event.data3 = (lastmousey - X_event.xmotion.y) << 2;

    if (event.data2 || event.data3)
    {
        lastmousex = X_event.xmotion.x;
        lastmousey = X_event.xmotion.y;
        if (X_event.xmotion.x != X_width/2 &&
            X_event.xmotion.y != X_height/2)
        {
            D_PostEvent(&event);
            // fprintf(stderr, "m");
            mousemoved = false;
        } else
        {
            mousemoved = true;
        }
    }
    break;

case Expose:
case ConfigureNotify:
    break;

default:

```

```

        if (doShm && X_event.type == X_shmeventtype) shmFinished = true;
        break;
    }
}

Cursor
createnullcursor
( Display*      display,
  Window       root )
{
    Pixmap cursormask;
    XGCValues xgc;
    GC gc;
    XColor dummycolour;
    Cursor cursor;

    cursormask = XCreatePixmap(display, root, 1, 1, 1/*depth*/);
    xgc.function = GXclear;
    gc = XCreateGC(display, cursormask, GCFunction, &xgc);
    XFillRectangle(display, cursormask, gc, 0, 0, 1, 1);
    dummycolour.pixel = 0;
    dummycolour.red = 0;
    dummycolour.flags = 04;
    cursor = XCreatePixmapCursor(display, cursormask, cursormask,
                                &dummycolour,&dummycolour, 0,0);
    XFreePixmap(display,cursormask);
    XFreeGC(display,gc);
    return cursor;
}

//
// I_StartTic
//
void I_StartTic (void)
{
    if (!X_display)
        return;

    while (XPending(X_display))
        I_GetEvent();

    // Warp the pointer back to the middle of the window
    // or it will wander off - that is, the game will
    // loose input focus within X11.
    if (grabMouse)
    {
        if (!--doPointerWarp)
        {
            XWarpPointer( X_display,
                          None,
                          X_mainWindow,
                          0, 0,
                          0, 0,
                          X_width/2, X_height/2);

            doPointerWarp = POINTER_WARP_COUNTDOWN;
        }
    }

    mousemoved = false;
}

```

```

//
// I_UpdateNoBlit
//
void I_UpdateNoBlit (void)
{
    // what is this?
}

//
// I_FinishUpdate
//
void I_FinishUpdate (void)
{

    static int      lasttic;
    int            tics;
    int            i;
    // UNUSED static unsigned char *bigscreen=0;

    // draws little dots on the bottom of the screen
    if (devparm)
    {

        i = I_GetTime();
        tics = i - lasttic;
        lasttic = i;
        if (tics > 20) tics = 20;

        for (i=0 ; i<tics*2 ; i+=2)
            screens[0][ (SCREENHEIGHT-1)*SCREENWIDTH + i] = 0xff;
        for ( ; i<20*2 ; i+=2)
            screens[0][ (SCREENHEIGHT-1)*SCREENWIDTH + i] = 0x0;

    }

    // scales the screen size before blitting it
    if (multiply == 2)
    {
        unsigned int *olineptrs[2];
        unsigned int *ilineptr;
        int x, y, i;
        unsigned int twoopixels;
        unsigned int twomoreopixels;
        unsigned int fouripixels;

        ilineptr = (unsigned int *) (screens[0]);
        for (i=0 ; i<2 ; i++)
            olineptrs[i] = (unsigned int *) &image->data[i*X_width];

        y = SCREENHEIGHT;
        while (y--)
        {
            x = SCREENWIDTH;
            do
            {
                fouripixels = *ilineptr++;
                twoopixels = (fouripixels & 0xff000000)
                    | ((fouripixels>>8) & 0xffff00)
                    | ((fouripixels>>16) & 0xff);
                twomoreopixels = ((fouripixels<<16) & 0xff000000)
                    | ((fouripixels<<8) & 0xffff00)
                    | (fouripixels & 0xff);
#ifdef __BIG_ENDIAN__
                *olineptrs[0]++ = twoopixels;

```

```

        *olineptrs[1]++ = twoopixels;
        *olineptrs[0]++ = twomoreopixels;
        *olineptrs[1]++ = twomoreopixels;
#else
        *olineptrs[0]++ = twomoreopixels;
        *olineptrs[1]++ = twomoreopixels;
        *olineptrs[0]++ = twoopixels;
        *olineptrs[1]++ = twoopixels;
#endif
    } while (x-=4);
    olineptrs[0] += X_width/4;
    olineptrs[1] += X_width/4;
}

}
else if (multiply == 3)
{
    unsigned int *olineptrs[3];
    unsigned int *ilineptr;
    int x, y, i;
    unsigned int fouropixels[3];
    unsigned int fouripixels;

    ilineptr = (unsigned int *) (screens[0]);
    for (i=0 ; i<3 ; i++)
        olineptrs[i] = (unsigned int *) &image->data[i*X_width];

    y = SCREENHEIGHT;
    while (y--)
    {
        x = SCREENWIDTH;
        do
        {
            fouripixels = *ilineptr++;
            fouropixels[0] = (fouripixels & 0xff000000)
                | ((fouripixels>>8) & 0xff0000)
                | ((fouripixels>>16) & 0xffff);
            fouropixels[1] = ((fouripixels<<8) & 0xff000000)
                | (fouripixels & 0xffff00)
                | ((fouripixels>>8) & 0xff);
            fouropixels[2] = ((fouripixels<<16) & 0xffff0000)
                | ((fouripixels<<8) & 0xff00)
                | (fouripixels & 0xff);
#ifdef __BIG_ENDIAN__
            *olineptrs[0]++ = fouropixels[0];
            *olineptrs[1]++ = fouropixels[0];
            *olineptrs[2]++ = fouropixels[0];
            *olineptrs[0]++ = fouropixels[1];
            *olineptrs[1]++ = fouropixels[1];
            *olineptrs[2]++ = fouropixels[1];
            *olineptrs[0]++ = fouropixels[2];
            *olineptrs[1]++ = fouropixels[2];
            *olineptrs[2]++ = fouropixels[2];
#else
            *olineptrs[0]++ = fouropixels[2];
            *olineptrs[1]++ = fouropixels[2];
            *olineptrs[2]++ = fouropixels[2];
            *olineptrs[0]++ = fouropixels[1];
            *olineptrs[1]++ = fouropixels[1];
            *olineptrs[2]++ = fouropixels[1];
            *olineptrs[0]++ = fouropixels[0];
            *olineptrs[1]++ = fouropixels[0];
            *olineptrs[2]++ = fouropixels[0];
#endif
        } while (x-=4);
    }
}

```



```

        olineptrs[0] += 2*X_width/4;
        olineptrs[1] += 2*X_width/4;
        olineptrs[2] += 2*X_width/4;
    }

}

else if (multiply == 4)
{
    // Broken. Gotta fix this some day.
    void Expand4(unsigned *, double *);
    Expand4 ((unsigned *) (screens[0]), (double *) (image->data));
}

if (doShm)
{
    if (!XShmPutImage(
        X_display,
        X_mainWindow,
        X_gc,
        image,
        0, 0,
        0, 0,
        X_width, X_height,
        True ))
        I_Error("XShmPutImage() failed\n");

    // wait for it to finish and processes all input events
    shmFinished = false;
    do
    {
        I_GetEvent();
    } while (!shmFinished);
}

else
{
    // draw the image
    XPutImage(
        X_display,
        X_mainWindow,
        X_gc,
        image,
        0, 0,
        0, 0,
        X_width, X_height );

    // sync up with server
    XSync(X_display, False);
}

}

//
// I_ReadScreen
//
void I_ReadScreen (byte* scr)
{
    memcpy (scr, screens[0], SCREENWIDTH*SCREENHEIGHT);
}

//
// Palette stuff.

```

```

//
static XColor      colors[256];

void UploadNewPalette(Colormap cmap, byte *palette)
{
    register int    i;
    register int    c;
    static boolean   firstcall = true;

#ifdef __cplusplus
    if (X_visualinfo.c_class == PseudoColor && X_visualinfo.depth == 8)
#else
    if (X_visualinfo.class == PseudoColor && X_visualinfo.depth == 8)
#endif
    {
        // initialize the colormap
        if (firstcall)
        {
            firstcall = false;
            for (i=0 ; i<256 ; i++)
            {
                colors[i].pixel = i;
                colors[i].flags = DoRed|DoGreen|DoBlue;
            }

            // set the X colormap entries
            for (i=0 ; i<256 ; i++)
            {
                c = gammatable[usegamma][*palette++];
                colors[i].red = (c<<8) + c;
                c = gammatable[usegamma][*palette++];
                colors[i].green = (c<<8) + c;
                c = gammatable[usegamma][*palette++];
                colors[i].blue = (c<<8) + c;
            }

            // store the colors to the current colormap
            XStoreColors(X_display, cmap, colors, 256);
        }
    }

//
// I_SetPalette
//
void I_SetPalette (byte* palette)
{
    UploadNewPalette(X_cmap, palette);
}

//
// This function is probably redundant,
// if XShmDetach works properly.
// ddt never detached the XShm memory,
// thus there might have been stale
// handles accumulating.
//
void grabsharedmemory(int size)
{
    int          key = ('d'<<24) | ('o'<<16) | ('o'<<8) | 'm';
    struct shmid_ds  shminfo;

```

```

int                minsize = 320*200;
int                id;
int                rc;
// UNUSED int done=0;
int                pollution=5;

// try to use what was here before
do
{
    id = shmget((key_t) key, minsize, 0777); // just get the id
    if (id != -1)
    {
        rc=shmctl(id, IPC_STAT, &shminfo); // get stats on it
        if (!rc)
        {
            if (shminfo.shm_nattch)
            {
                fprintf(stderr, "User %d appears to be running "
                    "DOOM. Is that wise?\n", shminfo.shm_cpid);
                key++;
            }
            else
            {
                if (getuid() == shminfo.shm_perm.cuid)
                {
                    rc = shmctl(id, IPC_RMID, 0);
                    if (!rc)
                        fprintf(stderr,
                            "Was able to kill my old shared memory\n");
                    else
                        I_Error("Was NOT able to kill my old shared memory");

                    id = shmget((key_t)key, size, IPC_CREAT|0777);
                    if (id==-1)
                        I_Error("Could not get shared memory");

                    rc=shmctl(id, IPC_STAT, &shminfo);

                    break;
                }
            }
            if (size >= shminfo.shm_segsz)
            {
                fprintf(stderr,
                    "will use %d's stale shared memory\n",
                    shminfo.shm_cpid);
                break;
            }
            else
            {
                fprintf(stderr,
                    "warning: can't use stale "
                    "shared memory belonging to id %d, "
                    "key=0x%x\n",
                    shminfo.shm_cpid, key);
                key++;
            }
        }
    }
    else
    {
        I_Error("could not get stats on key=%d", key);
    }
}
else

```

```

{
    id = shmget((key_t)key, size, IPC_CREAT|0777);
    if (id==-1)
    {
        extern int errno;
        fprintf(stderr, "errno=%d\n", errno);
        I_Error("Could not get any shared memory");
    }
    break;
}
} while (--pollution);

if (!pollution)
{
    I_Error("Sorry, system too polluted with stale "
           "shared memory segments.\n");
}

X_shminfo.shmid = id;

// attach to the shared memory segment
image->data = X_shminfo.shmaddr = shmat(id, 0, 0);

fprintf(stderr, "shared memory id=%d, addr=0x%x\n", id,
        (int) (image->data));
}

void I_InitGraphics(void)
{
    char*          displayname;
    char*          d;
    int            n;
    int            pnum;
    int            x=0;
    int            y=0;

    // warning: char format, different type arg
    char           xsign=' ';
    char           ysign=' ';

    int            oktodraw;
    unsigned long   attribmask;
    XSetWindowAttributes attribs;
    XGCValues       xgcvalues;
    int            valuemask;
    static int      firsttime=1;

    if (!firsttime)
        return;
    firsttime = 0;

    signal(SIGINT, (void (*)(int)) I_Quit);

    if (M_CheckParm("-2"))
        multiply = 2;

    if (M_CheckParm("-3"))
        multiply = 3;

    if (M_CheckParm("-4"))
        multiply = 4;

    X_width = SCREENWIDTH * multiply;
    X_height = SCREENHEIGHT * multiply;

```

```

// check for command-line display name
if ( (pnum=M_CheckParm("-disp")) ) // suggest parentheses around assignment
    displayname = myargv[pnum+1];
else
    displayname = 0;

// check if the user wants to grab the mouse (quite unnice)
grabMouse = !!M_CheckParm("-grabmouse");

// check for command-line geometry
if ( (pnum=M_CheckParm("-geom")) ) // suggest parentheses around assignment
{
    // warning: char format, different type arg 3,5
    n = sscanf(myargv[pnum+1], "%c%d%c%d", &xsign, &x, &y, &y);

    if (n==2)
        x = y = 0;
    else if (n==6)
    {
        if (xsign == '-')
            x = -x;
        if (ysign == '-')
            y = -y;
    }
    else
        I_Error("bad -geom parameter");
}

// open the display
X_display = XOpenDisplay(displayname);
if (!X_display)
{
    if (displayname)
        I_Error("Could not open display [%s]", displayname);
    else
        I_Error("Could not open display (DISPLAY=[%s])", getenv("DISPLAY"));
}

// use the default visual
X_screen = DefaultScreen(X_display);
if (!XMatchVisualInfo(X_display, X_screen, 8, PseudoColor, &X_visualinfo))
    I_Error("xdoom currently only supports 256-color PseudoColor screens");
X_visual = X_visualinfo.visual;

// check for the MITSHM extension
doShm = XShmQueryExtension(X_display);

// even if it's available, make sure it's a local connection
if (doShm)
{
    if (!displayname) displayname = (char *) getenv("DISPLAY");
    if (displayname)
    {
        d = displayname;
        while (*d && (*d != ':')) d++;
        if (*d) *d = 0;
        if (!(strcmp(displayname, "unix") || !*displayname)) doShm = false;
    }
}

fprintf(stderr, "Using MITSHM extension\n");

// create the colormap
X_cmap = XCreateColormap(X_display, RootWindow(X_display,
```

```

X_screen), X_visual, AllocAll);

// setup attributes for main window
attribmask = CWEventMask | CWColormap | CWBorderPixel;
attribs.event_mask =
    KeyPressMask
    | KeyReleaseMask
    // | PointerMotionMask | ButtonPressMask | ButtonReleaseMask
    | ExposureMask;

attribs.colormap = X_cmap;
attribs.border_pixel = 0;

// create the main window
X_mainWindow = XCreateWindow(
    X_display,
    RootWindow(X_display, X_screen),
    x, y,
    X_width, X_height,
    0, // borderwidth
    8, // depth
    InputOutput,
    X_visual,
    attribmask,
    &attribs );

XDefineCursor(X_display, X_mainWindow,
    createnullcursor( X_display, X_mainWindow ) );

// create the GC
valuemask = GCGraphicsExposures;
xgcvalues.graphics_exposures = False;
X_gc = XCreateGC(
    X_display,
    X_mainWindow,
    valuemask,
    &xgcvalues );

// map the window
XMapWindow(X_display, X_mainWindow);

// wait until it is OK to draw
oktodraw = 0;
while (!oktodraw)
{
    XNextEvent(X_display, &X_event);
    if (X_event.type == Expose
        && !X_event.xexpose.count)
    {
        oktodraw = 1;
    }
}

// grabs the pointer so it is restricted to this window
if (grabMouse)
    XGrabPointer(X_display, X_mainWindow, True,
        ButtonPressMask|ButtonReleaseMask|PointerMotionMask,
        GrabModeAsync, GrabModeAsync,
        X_mainWindow, None, CurrentTime);

if (doShm)
{
    X_shmeventtype = XShmGetEventBase(X_display) + ShmCompletion;

    // create the image
    image = XShmCreateImage(
        X_display,

```

```

        X_visual,
        8,
        ZPixmap,
        0,
        &X_shminfo,
        X_width,
        X_height );

grabsharedmemory(image->bytes_per_line * image->height);

// UNUSED
// create the shared memory segment
// X_shminfo.shmid = shmget (IPC_PRIVATE,
// image->bytes_per_line * image->height, IPC_CREAT | 0777);
// if (X_shminfo.shmid < 0)
// {
//     perror("");
//     I_Error("shmget() failed in InitGraphics()");
// }
// fprintf(stderr, "shared memory id=%d\n", X_shminfo.shmid);
// attach to the shared memory segment
// image->data = X_shminfo.shmaddr = shmat(X_shminfo.shmid, 0, 0);

if (!image->data)
{
    perror("");
    I_Error("shmat() failed in InitGraphics()");
}

// get the X server to attach to it
if (!XShmAttach(X_display, &X_shminfo))
    I_Error("XShmAttach() failed in InitGraphics()");

}
else
{
    image = XCreateImage(
        X_display,
        X_visual,
        8,
        ZPixmap,
        0,
        (char*)malloc(X_width * X_height),
        X_width, X_height,
        8,
        X_width );
}

if (multiply == 1)
    screens[0] = (unsigned char *) (image->data);
else
    screens[0] = (unsigned char *) malloc (SCREENWIDTH * SCREENHEIGHT);
}

unsigned      exptable[256];

void InitExpand (void)
{
    int          i;

    for (i=0 ; i<256 ; i++)

```

```

        exptable[i] = i | (i<<8) | (i<<16) | (i<<24);
    }

double
        exptable2[256*256];

void InitExpand2 (void)
{
    int            i;
    int            j;
    // UNUSED unsigned    iexp, jexp;
    double*        exp;
    union
    {
        double            d;
        unsigned            u[2];
    } pixel;

    printf ("building exptable2...\n");
    exp = exptable2;
    for (i=0 ; i<256 ; i++)
    {
        pixel.u[0] = i | (i<<8) | (i<<16) | (i<<24);
        for (j=0 ; j<256 ; j++)
        {
            pixel.u[1] = j | (j<<8) | (j<<16) | (j<<24);
            *exp++ = pixel.d;
        }
    }
    printf ("done.\n");
}

int            initied;

void
Expand4
( unsigned*        lineptr,
  double*        xline )
{
    double            dpixel;
    unsigned            x;
    unsigned            y;
    unsigned            fourpixels;
    unsigned            step;
    double*            exp;

    exp = exptable2;
    if (!initied)
    {
        initied = 1;
        InitExpand2 ();
    }

    step = 3*SCREENWIDTH/2;

    y = SCREENHEIGHT-1;
    do
    {
        x = SCREENWIDTH;

        do
        {
            fourpixels = lineptr[0];

            dpixel = *(double *) ( (int)exp + ( (fourpixels&0xffff0000)>>13) );

```



```

xline[0] = dpixel;
xline[160] = dpixel;
xline[320] = dpixel;
xline[480] = dpixel;

dpixel = *(double *) ( (int)exp + ( (fourpixels&0xffff)<<3 ) );
xline[1] = dpixel;
xline[161] = dpixel;
xline[321] = dpixel;
xline[481] = dpixel;

fourpixels = lineptr[1];

dpixel = *(double *) ( (int)exp + ( (fourpixels&0xffff0000)>>13 ) );
xline[2] = dpixel;
xline[162] = dpixel;
xline[322] = dpixel;
xline[482] = dpixel;

dpixel = *(double *) ( (int)exp + ( (fourpixels&0xffff)<<3 ) );
xline[3] = dpixel;
xline[163] = dpixel;
xline[323] = dpixel;
xline[483] = dpixel;

fourpixels = lineptr[2];

dpixel = *(double *) ( (int)exp + ( (fourpixels&0xffff0000)>>13 ) );
xline[4] = dpixel;
xline[164] = dpixel;
xline[324] = dpixel;
xline[484] = dpixel;

dpixel = *(double *) ( (int)exp + ( (fourpixels&0xffff)<<3 ) );
xline[5] = dpixel;
xline[165] = dpixel;
xline[325] = dpixel;
xline[485] = dpixel;

fourpixels = lineptr[3];

dpixel = *(double *) ( (int)exp + ( (fourpixels&0xffff0000)>>13 ) );
xline[6] = dpixel;
xline[166] = dpixel;
xline[326] = dpixel;
xline[486] = dpixel;

dpixel = *(double *) ( (int)exp + ( (fourpixels&0xffff)<<3 ) );
xline[7] = dpixel;
xline[167] = dpixel;
xline[327] = dpixel;
xline[487] = dpixel;

lineptr+=4;
xline+=8;
} while (x-=16);
xline += step;
} while (y--);
}

```

## 7.9 i\_video.h

```
// Emacs style mode select  -*- C++ -*-
```

```

//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      System specific interface stuff.
//
//-----

#ifndef __I_VIDEO__
#define __I_VIDEO__

#include "doomtype.h"

#ifdef __GNUG__
#pragma interface
#endif

// Called by D_DoomMain,
// determines the hardware configuration
// and sets up the video mode
void I_InitGraphics (void);

void I_ShutdownGraphics(void);

// Takes full 8 bit values.
void I_SetPalette (byte* palette);

void I_UpdateNoBlit (void);
void I_FinishUpdate (void);

// Wait for vertical retrace or pause a bit.
void I_WaitVBL(int count);

void I_ReadScreen (byte* scr);

void I_BeginRead (void);
void I_EndRead (void);

#endif
//-----
//
// $Log:$
//
//-----

```

## 8 Miscellaneous

### 8.1 m\_argv.c

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//-----

static const char
rcsid[] = "$Id: m_argv.c,v 1.1 1997/02/03 22:45:10 b1 Exp $";

#include <string.h>

int          myargc;
char**       myargv;

//
// M_CheckParm
// Checks for the given parameter
// in the program's command line arguments.
// Returns the argument number (1 to argc-1)
// or 0 if not present
int M_CheckParm (char *check)
{
    int          i;

    for (i = 1; i < myargc; i++)
    {
        if ( !strcasecmp(check, myargv[i]) )
            return i;
    }

    return 0;
}
```

### 8.2 m\_argv.h

```
// Emacs style mode select  -*- C++ -*-
```

```
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
// Nil.
//
//-----

#ifdef __M_ARGV__
#define __M_ARGV__

//
// MISC
//
extern int myargc;
extern char** myargv;

// Returns the position of the given parameter
// in the arg list (0 if not found).
int M_CheckParm (char* check);

#endif
//-----
```

### 8.3 m\_bbox.c

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
```

```

//      Main loop menu stuff.
//      Random number LUT.
//      Default Config File.
//      PCX Screenshots.
//
//-----

static const char
rcsid[] = "$Id: m_bbox.c,v 1.1 1997/02/03 22:45:10 b1 Exp $";

#ifdef __GNUG__
#pragma implementation "m_bbox.h"
#endif
#include "m_bbox.h"

void M_ClearBox (fixed_t *box)
{
    box[BOXTOP] = box[BOXRIGHT] = MININT;
    box[BOXBOTTOM] = box[BOXLEFT] = MAXINT;
}

void
M_AddToBox
( fixed_t*      box,
  fixed_t      x,
  fixed_t      y )
{
    if (x<box[BOXLEFT])
        box[BOXLEFT] = x;
    else if (x>box[BOXRIGHT])
        box[BOXRIGHT] = x;
    if (y<box[BOXBOTTOM])
        box[BOXBOTTOM] = y;
    else if (y>box[BOXTOP])
        box[BOXTOP] = y;
}

```

## 8.4 m\_bbox.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//

```

```

// DESCRIPTION:
//   Nil.
//
//-----

#ifndef __M_BBOX__
#define __M_BBOX__

#include <values.h>

#include "m_fixed.h"

// Bounding box coordinate storage.
enum
{
    BOXTOP,
    BOXBOTTOM,
    BOXLEFT,
    BOXRIGHT
};          // bbox coordinates

// Bounding box functions.
void M_ClearBox (fixed_t*          box);

void
M_AddToBox
( fixed_t*          box,
  fixed_t          x,
  fixed_t          y );

#endif
//-----
//
// $Log:$
//
//-----

```

## 8.5 m\_cheat.c

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//   Cheat sequence checking.
//
//-----

```

```

static const char
rcsid[] = "$Id: m_cheat.c,v 1.1 1997/02/03 21:24:34 b1 Exp $";

#include "m_cheat.h"

//
// CHEAT SEQUENCE PACKAGE
//

static int                firsttime = 1;
static unsigned char      cheat_xlate_table[256];

//
// Called in st_stuff module, which handles the input.
// Returns a 1 if the cheat was successful, 0 if failed.
//
int
cht_CheckCheat
( cheatseq_t*            cht,
  char                  key )
{
    int i;
    int rc = 0;

    if (firsttime)
    {
        firsttime = 0;
        for (i=0;i<256;i++) cheat_xlate_table[i] = SCRAMBLE(i);
    }

    if (!cht->p)
        cht->p = cht->sequence; // initialize if first time

    if (*cht->p == 0)
        *(cht->p++) = key;
    else if
        (cheat_xlate_table[(unsigned char)key] == *cht->p) cht->p++;
    else
        cht->p = cht->sequence;

    if (*cht->p == 1)
        cht->p++;
    else if (*cht->p == 0xff) // end of sequence character
    {
        cht->p = cht->sequence;
        rc = 1;
    }

    return rc;
}

void
cht_GetParam
( cheatseq_t*            cht,
  char*                  buffer )
{
    unsigned char *p, c;

    p = cht->sequence;
    while (*(p++) != 1);
}

```

```

do
{
    c = *p;
    *(buffer++) = c;
    *(p++) = 0;
}
while (c && *p!=0xff );

if (*p==0xff)
    *buffer = 0;

}

```

## 8.6 m\_cheat.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      Cheat code checking.
//-----

#ifndef __M_CHEAT__
#define __M_CHEAT__

//
// CHEAT SEQUENCE PACKAGE
//

#define SCRAMBLE(a) \
(((a)&1)<<7) + (((a)&2)<<5) + ((a)&4) + (((a)&8)<<1) \
+ (((a)&16)>>1) + ((a)&32) + (((a)&64)>>5) + (((a)&128)>>7))

typedef struct
{
    unsigned char*    sequence;
    unsigned char*    p;
} cheatseq_t;

int
cht_CheckCheat
( cheatseq_t*        cht,
  char                key );

void

```



```

cht_GetParam
( cheatseq_t*          cht,
  char*                buffer );

```

```

#endif
//-----
//
// $Log:$
//
//-----

```

## 8.7 m\_fixed.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Fixed point implementation.
//
//-----

```

```

static const char
rcsid[] = "$Id: m_bbox.c,v 1.1 1997/02/03 22:45:10 b1 Exp $";

```

```

#include "stdlib.h"

```

```

#include "doomtype.h"
#include "i_system.h"

```

```

#ifdef __GNUG__
#pragma implementation "m_fixed.h"
#endif
#include "m_fixed.h"

```

```

// Fixme. __USE_C_FIXED__ or something.

```

```

fixed_t
FixedMul
( fixed_t      a,
  fixed_t      b )
{
    return ((long long) a * (long long) b) >> FRACBITS;
}

```

```

//
// FixedDiv, C version.
//

fixed_t
FixedDiv
( fixed_t      a,
  fixed_t      b )
{
    if ( (abs(a)>>14) >= abs(b))
        return (a^b)<0 ? MININT : MAXINT;
    return FixedDiv2 (a,b);
}

fixed_t
FixedDiv2
( fixed_t      a,
  fixed_t      b )
{
#ifdef 0
    long long c;
    c = ((long long)a<<16) / ((long long)b);
    return (fixed_t) c;
#endif

    double c;

    c = ((double)a) / ((double)b) * FRACUNIT;

    if (c >= 2147483648.0 || c < -2147483648.0)
        I_Error("FixedDiv: divide by zero");
    return (fixed_t) c;
}

```

## 8.8 m\_fixed.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Fixed point arithemtics, implementation.
//-----

#ifdef __M_FIXED__
#define __M_FIXED__

```

```

#ifdef __GNUG__
#pragma interface
#endif

//
// Fixed point, 32bit as 16.16.
//
#define FRACBITS          16
#define FRACUNIT          (1<<FRACBITS)

typedef int fixed_t;

fixed_t FixedMul          (fixed_t a, fixed_t b);
fixed_t FixedDiv          (fixed_t a, fixed_t b);
fixed_t FixedDiv2         (fixed_t a, fixed_t b);

#endif
//-----
//
// $Log:$
//
//-----

```

## 8.9 m\_menu.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      DOOM selection menu, options, episode etc.
//      Sliders and icons. Kinda widget stuff.
//
//-----

static const char
rcsid[] = "$Id: m_menu.c,v 1.7 1997/02/03 22:45:10 b1 Exp $";

#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <ctype.h>

```

```

#include "doomdef.h"
#include "dstrings.h"

#include "d_main.h"

#include "i_system.h"
#include "i_video.h"
#include "z_zone.h"
#include "v_video.h"
#include "w_wad.h"

#include "r_local.h"

#include "hu_stuff.h"

#include "g_game.h"

#include "m_argv.h"
#include "m_swap.h"

#include "s_sound.h"

#include "doomstat.h"

// Data.
#include "sounds.h"

#include "m_menu.h"

extern patch_t*          hu_font[HU_FONTSIZE];
extern boolean           message_dontfuckwithme;

extern boolean           chat_on;                // in heads-up code

//
// defaulted values
//
int                      mouseSensitivity;       // has default

// Show messages has default, 0 = off, 1 = on
int                      showMessages;

// Blocky mode, has default, 0 = high, 1 = normal
int                      detailLevel;
int                      screenblocks;           // has default

// temp for screenblocks (0-9)
int                      screenSize;

// -1 = no quicksave slot picked!
int                      quickSaveSlot;

// 1 = message to be printed
int                      messageToPrint;
// ...and here is the message string!
char*                    messageString;

// message x & y
int                      messx;
int                      messy;

```

```

int                messageLastMenuActive;

// timed message = no input from user
boolean            messageNeedsInput;

void      (*messageRoutine)(int response);

#define SAVESTRINGSIZE      24

char gammamsg[5][26] =
{
    GAMMALVLO,
    GAMMALVL1,
    GAMMALVL2,
    GAMMALVL3,
    GAMMALVL4
};

// we are going to be entering a savegame string
int                saveStringEnter;
int                saveSlot;           // which slot to save in
int                saveCharIndex;      // which char we're editing
// old save description before edit
char                saveOldString[SAVESTRINGSIZE];

boolean            inhelpscreens;
boolean            menuactive;

#define SKULLXOFF      -32
#define LINEHEIGHT     16

extern boolean      sendpause;
char                savegamestrings[10][SAVESTRINGSIZE];

char                endstring[160];

//
// MENU TYPEDEFS
//
typedef struct
{
    // 0 = no cursor here, 1 = ok, 2 = arrows ok
    short            status;

    char             name[10];

    // choice = menu item #.
    // if status = 2,
    // choice=0:leftarrow,1:rightarrow
    void             (*routine)(int choice);

    // hotkey in menu
    char             alphaKey;
} menuitem_t;

typedef struct menu_s
{
    short            numitems;           // # of menu items
    struct menu_s*   prevMenu;          // previous menu
    menuitem_t*      menuitems;         // menu items
    void             (*routine)();      // draw routine
    short            x;

```

```

    short          y;                // x,y of menu
    short          lastOn;           // last item user was on in menu
} menu_t;

short             itemOn;            // menu item skull is on
short             skullAnimCounter;  // skull animation counter
short             whichSkull;        // which skull to draw

// graphic name of skulls
// warning: initializer-string for array of chars is too long
char skullName[2][/*8*/9] = {"M_SKULL1","M_SKULL2"};

// current menudef
menu_t*           currentMenu;

//
// PROTOTYPES
//
void M_NewGame(int choice);
void M_Episode(int choice);
void M_ChooseSkill(int choice);
void M_LoadGame(int choice);
void M_SaveGame(int choice);
void M_Options(int choice);
void M_EndGame(int choice);
void M_ReadThis(int choice);
void M_ReadThis2(int choice);
void M_QuitDOOM(int choice);

void M_ChangeMessages(int choice);
void M_ChangeSensitivity(int choice);
void M_SfxVol(int choice);
void M_MusicVol(int choice);
void M_ChangeDetail(int choice);
void M_SizeDisplay(int choice);
void M_StartGame(int choice);
void M_Sound(int choice);

void M_FinishReadThis(int choice);
void M_LoadSelect(int choice);
void M_SaveSelect(int choice);
void M_ReadSaveStrings(void);
void M_QuickSave(void);
void M_QuickLoad(void);

void M_DrawMainMenu(void);
void M_DrawReadThis1(void);
void M_DrawReadThis2(void);
void M_DrawNewGame(void);
void M_DrawEpisode(void);
void M_DrawOptions(void);
void M_DrawSound(void);
void M_DrawLoad(void);
void M_DrawSave(void);

void M_DrawSaveLoadBorder(int x,int y);
void M_SetupNextMenu(menu_t *menudef);
void M_DrawThermo(int x,int y,int thermWidth,int thermDot);
void M_DrawEmptyCell(menu_t *menu,int item);
void M_DrawSelCell(menu_t *menu,int item);
void M_WriteText(int x, int y, char *string);
int M_StringWidth(char *string);
int M_StringHeight(char *string);
void M_StartControlPanel(void);
void M_StartMessage(char *string,void *routine,boolean input);

```

```

void M_StopMessage(void);
void M_ClearMenus (void);


//
// DOOM MENU
//
enum
{
    newgame = 0,
    options,
    loadgame,
    savegame,
    readthis,
    quitdoom,
    main_end
} main_e;

menuitem_t MainMenu[]=
{
    {1,"M_NGAME",M_NewGame,'n'},
    {1,"M_OPTION",M_Options,'o'},
    {1,"M_LOADG",M_LoadGame,'l'},
    {1,"M_SAVEG",M_SaveGame,'s'},
    // Another hickup with Special edition.
    {1,"M_RDTHIS",M_ReadThis,'r'},
    {1,"M_QUITG",M_QuitDOOM,'q'}
};

menu_t MainDef =
{
    main_end,
    NULL,
    MainMenu,
    M_DrawMainMenu,
    97,64,
    0
};


//
// EPISODE SELECT
//
enum
{
    ep1,
    ep2,
    ep3,
    ep4,
    ep_end
} episodes_e;

menuitem_t EpisodeMenu[]=
{
    {1,"M_EPI1", M_Episode,'k'},
    {1,"M_EPI2", M_Episode,'t'},
    {1,"M_EPI3", M_Episode,'i'},
    {1,"M_EPI4", M_Episode,'t'}
};

menu_t EpiDef =
{
    ep_end,                // # of menu items

```

```

    &MainDef,          // previous menu
    EpisodeMenu,      // menuitem_t ->
    M_DrawEpisode,    // drawing routine ->
    48,63,            // x,y
    ep1               // lastOn
};

//
// NEW GAME
//
enum
{
    killthings,
    toorough,
    hurtme,
    violence,
    nightmare,
    newg_end
} newgame_e;

menuitem_t NewGameMenu[]=
{
    {1,"M_JKILL",      M_ChooseSkill, 'i'},
    {1,"M_ROUGH",      M_ChooseSkill, 'h'},
    {1,"M_HURT",       M_ChooseSkill, 'h'},
    {1,"M_ULTRA",      M_ChooseSkill, 'u'},
    {1,"M_NMARE",      M_ChooseSkill, 'n'}
};

menu_t NewDef =
{
    newg_end,          // # of menu items
    &EpiDef,           // previous menu
    NewGameMenu,       // menuitem_t ->
    M_DrawNewGame,     // drawing routine ->
    48,63,             // x,y
    hurtme             // lastOn
};

//
// OPTIONS MENU
//
enum
{
    endgame,
    messages,
    detail,
    scrnsize,
    option_empty1,
    mousesens,
    option_empty2,
    soundvol,
    opt_end
} options_e;

menuitem_t OptionsMenu[]=
{
    {1,"M_ENDGAM",     M_EndGame, 'e'},
    {1,"M_MESSG",      M_ChangeMessages, 'm'},
    {1,"M_DETAIL",     M_ChangeDetail, 'g'},
    {2,"M_SCRNSZ",     M_SizeDisplay, 's'},
    {-1,"",0},
    {2,"M_MSENS",      M_ChangeSensitivity, 'm'},

```



```

        {-1,"",0},
        {1,"M_SVOL",      M_Sound,'s'}
};

menu_t OptionsDef =
{
    opt_end,
    &MainDef,
    OptionsMenu,
    M_DrawOptions,
    60,37,
    0
};

//
// Read This! MENU 1 & 2
//
enum
{
    rdthsemt1,
    read1_end
} read_e;

menuitem_t ReadMenu1[] =
{
    {1,"",M_ReadThis2,0}
};

menu_t ReadDef1 =
{
    read1_end,
    &MainDef,
    ReadMenu1,
    M_DrawReadThis1,
    280,185,
    0
};

enum
{
    rdthsemt2,
    read2_end
} read_e2;

menuitem_t ReadMenu2[] =
{
    {1,"",M_FinishReadThis,0}
};

menu_t ReadDef2 =
{
    read2_end,
    &ReadDef1,
    ReadMenu2,
    M_DrawReadThis2,
    330,175,
    0
};

//
// SOUND VOLUME MENU
//
enum
{
    sfx_vol,

```

```

    sfx_empty1,
    music_vol,
    sfx_empty2,
    sound_end
} sound_e;

menuitem_t SoundMenu[]=
{
    {2,"M_SFXVOL",M_SfxVol,'s'},
    {-1,"",0},
    {2,"M_MUSVOL",M_MusicVol,'m'},
    {-1,"",0}
};

menu_t SoundDef =
{
    sound_end,
    &OptionsDef,
    SoundMenu,
    M_DrawSound,
    80,64,
    0
};

//
// LOAD GAME MENU
//
enum
{
    load1,
    load2,
    load3,
    load4,
    load5,
    load6,
    load_end
} load_e;

menuitem_t LoadMenu[]=
{
    {1,"", M_LoadSelect,'1'},
    {1,"", M_LoadSelect,'2'},
    {1,"", M_LoadSelect,'3'},
    {1,"", M_LoadSelect,'4'},
    {1,"", M_LoadSelect,'5'},
    {1,"", M_LoadSelect,'6'}
};

menu_t LoadDef =
{
    load_end,
    &MainDef,
    LoadMenu,
    M_DrawLoad,
    80,54,
    0
};

//
// SAVE GAME MENU
//
menuitem_t SaveMenu[]=
{
    {1,"", M_SaveSelect,'1'},
    {1,"", M_SaveSelect,'2'},

```

```

    {1,"", M_SaveSelect,'3'},
    {1,"", M_SaveSelect,'4'},
    {1,"", M_SaveSelect,'5'},
    {1,"", M_SaveSelect,'6'}
};

menu_t SaveDef =
{
    load_end,
    &MainDef,
    SaveMenu,
    M_DrawSave,
    80,54,
    0
};

//
// M_ReadSaveStrings
// read the strings from the savegame files
//
void M_ReadSaveStrings(void)
{
    int         handle;
    int         count;
    int         i;
    char        name[256];

    for (i = 0;i < load_end;i++)
    {
        if (M_CheckParm("-cdrom"))
            sprintf(name,"c:\\doomdata\\"SAVEGAMENAME"%d.dsg",i);
        else
            sprintf(name,SAVEGAMENAME"%d.dsg",i);

        handle = open (name, O_RDONLY | 0, 0666);
        if (handle == -1)
        {
            strcpy(&savegamestrings[i][0],EMPTYSTRING);
            LoadMenu[i].status = 0;
            continue;
        }
        count = read (handle, &savegamestrings[i], SAVESTRINGSIZE);
        close (handle);
        LoadMenu[i].status = 1;
    }
}

//
// M_LoadGame & Cie.
//
void M_DrawLoad(void)
{
    int         i;

    V_DrawPatchDirect (72,28,0,W_CacheLumpName("M_LOADG",PU_CACHE));
    for (i = 0;i < load_end; i++)
    {
        M_DrawSaveLoadBorder(LoadDef.x,LoadDef.y+LINEHEIGHT*i);
        M_WriteText(LoadDef.x,LoadDef.y+LINEHEIGHT*i,savegamestrings[i]);
    }
}

```

```

//
// Draw border for the savegame description
//
void M_DrawSaveLoadBorder(int x,int y)
{
    int            i;

    V_DrawPatchDirect (x-8,y+7,0,W_CacheLumpName("M_LSLEFT",PU_CACHE));

    for (i = 0;i < 24;i++)
    {
        V_DrawPatchDirect (x,y+7,0,W_CacheLumpName("M_LSCNTR",PU_CACHE));
        x += 8;
    }

    V_DrawPatchDirect (x,y+7,0,W_CacheLumpName("M_LSRGHT",PU_CACHE));
}

//
// User wants to load this game
//
void M_LoadSelect(int choice)
{
    char    name[256];

    if (M_CheckParm("-cdrom"))
        sprintf(name,"c:\\doomdata\\"SAVEGAMENAME"%d.dsg",choice);
    else
        sprintf(name,SAVEGAMENAME"%d.dsg",choice);
    G_LoadGame (name);
    M_ClearMenus ();
}

//
// Selected from DOOM menu
//
void M_LoadGame (int choice)
{
    if (netgame)
    {
        M_StartMessage(LOADNET,NULL,false);
        return;
    }

    M_SetupNextMenu(&LoadDef);
    M_ReadSaveStrings();
}

//
// M_SaveGame & Cie.
//
void M_DrawSave(void)
{
    int            i;

    V_DrawPatchDirect (72,28,0,W_CacheLumpName("M_SAVEG",PU_CACHE));
    for (i = 0;i < load_end; i++)
    {
        M_DrawSaveLoadBorder(LoadDef.x,LoadDef.y+LINEHEIGHT*i);
        M_WriteText(LoadDef.x,LoadDef.y+LINEHEIGHT*i,savegamestrings[i]);
    }
}

```

```

    if (saveStringEnter)
    {
        i = M_StringWidth(savegamestrings[saveSlot]);
        M_WriteText(LoadDef.x + i, LoadDef.y + LINEHEIGHT * saveSlot, "_");
    }
}

//
// M_Responder calls this when user is finished
//
void M_DoSave(int slot)
{
    G_SaveGame (slot, savegamestrings[slot]);
    M_ClearMenus ();

    // PICK QUICKSAVE SLOT YET?
    if (quickSaveSlot == -2)
        quickSaveSlot = slot;
}

//
// User wants to save. Start string input for M_Responder
//
void M_SaveSelect(int choice)
{
    // we are going to be intercepting all chars
    saveStringEnter = 1;

    saveSlot = choice;
    strcpy(saveOldString, savegamestrings[choice]);
    if (!strcmp(savegamestrings[choice], EMPTYSTRING))
        savegamestrings[choice][0] = 0;
    saveCharIndex = strlen(savegamestrings[choice]);
}

//
// Selected from DOOM menu
//
void M_SaveGame (int choice)
{
    if (!usergame)
    {
        M_StartMessage(SAVEDEAD, NULL, false);
        return;
    }

    if (gamestate != GS_LEVEL)
        return;

    M_SetupNextMenu(&SaveDef);
    M_ReadSaveStrings();
}

//
//      M_QuickSave
//
char    tempstring[80];

void M_QuickSaveResponse(int ch)
{
    if (ch == 'y')
    {

```

```

        M_DoSave(quickSaveSlot);
        S_StartSound(NULL,sfx_swthx);
    }
}

void M_QuickSave(void)
{
    if (!usergame)
    {
        S_StartSound(NULL,sfx_oof);
        return;
    }

    if (gamestate != GS_LEVEL)
        return;

    if (quickSaveSlot < 0)
    {
        M_StartControlPanel();
        M_ReadSaveStrings();
        M_SetupNextMenu(&SaveDef);
        quickSaveSlot = -2;          // means to pick a slot now
        return;
    }
    sprintf(tempstring,QSPROMPT,savegamestrings[quickSaveSlot]);
    M_StartMessage(tempstring,M_QuickSaveResponse,true);
}

//
// M_QuickLoad
//
void M_QuickLoadResponse(int ch)
{
    if (ch == 'y')
    {
        M_LoadSelect(quickSaveSlot);
        S_StartSound(NULL,sfx_swthx);
    }
}

void M_QuickLoad(void)
{
    if (netgame)
    {
        M_StartMessage(QLOADNET,NULL,false);
        return;
    }

    if (quickSaveSlot < 0)
    {
        M_StartMessage(QSAVESPOT,NULL,false);
        return;
    }
    sprintf(tempstring,QLPROMPT,savegamestrings[quickSaveSlot]);
    M_StartMessage(tempstring,M_QuickLoadResponse,true);
}

//
// Read This Menus

```

```

// Had a "quick hack to fix romero bug"
//
void M_DrawReadThis1(void)
{
    inhelppcreens = true;
    switch ( gamemode )
    {
        case commercial:
            V_DrawPatchDirect (0,0,0,W_CacheLumpName("HELP",PU_CACHE));
            break;
        case shareware:
        case registered:
        case retail:
            V_DrawPatchDirect (0,0,0,W_CacheLumpName("HELP1",PU_CACHE));
            break;
        default:
            break;
    }
    return;
}

//
// Read This Menus - optional second page.
//
void M_DrawReadThis2(void)
{
    inhelppcreens = true;
    switch ( gamemode )
    {
        case retail:
        case commercial:
            // This hack keeps us from having to change menus.
            V_DrawPatchDirect (0,0,0,W_CacheLumpName("CREDIT",PU_CACHE));
            break;
        case shareware:
        case registered:
            V_DrawPatchDirect (0,0,0,W_CacheLumpName("HELP2",PU_CACHE));
            break;
        default:
            break;
    }
    return;
}

//
// Change Sfx & Music volumes
//
void M_DrawSound(void)
{
    V_DrawPatchDirect (60,38,0,W_CacheLumpName("M_SVOL",PU_CACHE));

    M_DrawThermo(SoundDef.x,SoundDef.y+LINEHEIGHT*(sfx_vol+1),
        16,snd_SfxVolume);

    M_DrawThermo(SoundDef.x,SoundDef.y+LINEHEIGHT*(music_vol+1),
        16,snd_MusicVolume);
}

void M_Sound(int choice)
{
    M_SetupNextMenu(&SoundDef);
}

```

```

void M_SfxVol(int choice)
{
    switch(choice)
    {
        case 0:
            if (snd_SfxVolume)
                snd_SfxVolume--;
            break;
        case 1:
            if (snd_SfxVolume < 15)
                snd_SfxVolume++;
            break;
    }

    S_SetSfxVolume(snd_SfxVolume /* *8 */);
}

void M_MusicVol(int choice)
{
    switch(choice)
    {
        case 0:
            if (snd_MusicVolume)
                snd_MusicVolume--;
            break;
        case 1:
            if (snd_MusicVolume < 15)
                snd_MusicVolume++;
            break;
    }

    S_SetMusicVolume(snd_MusicVolume /* *8 */);
}

//
// M_DrawMainMenu
//
void M_DrawMainMenu(void)
{
    V_DrawPatchDirect (94,2,0,W_CacheLumpName("M_DOOM",PU_CACHE));
}

//
// M_NewGame
//
void M_DrawNewGame(void)
{
    V_DrawPatchDirect (96,14,0,W_CacheLumpName("M_NEWG",PU_CACHE));
    V_DrawPatchDirect (54,38,0,W_CacheLumpName("M_SKILL",PU_CACHE));
}

void M_NewGame(int choice)
{
    if (netgame && !demoplayback)
    {
        M_StartMessage(NEWGAME,NULL,false);
        return;
    }
}

```



```

    if ( gamemode == commercial )
        M_SetupNextMenu(&NewDef);
    else
        M_SetupNextMenu(&EpiDef);
}

//
//      M_Episode
//
int      epi;

void M_DrawEpisode(void)
{
    V_DrawPatchDirect (54,38,0,W_CacheLumpName("M_EPISOD",PU_CACHE));
}

void M_VerifyNightmare(int ch)
{
    if (ch != 'y')
        return;

    G_DeferedInitNew(nightmare,epi+1,1);
    M_ClearMenus ();
}

void M_ChooseSkill(int choice)
{
    if (choice == nightmare)
    {
        M_StartMessage(NIGHTMARE,M_VerifyNightmare,true);
        return;
    }

    G_DeferedInitNew(choice,epi+1,1);
    M_ClearMenus ();
}

void M_Episode(int choice)
{
    if ( (gamemode == shareware)
        && choice)
    {
        M_StartMessage(SWSTRING,NULL,false);
        M_SetupNextMenu(&ReadDef1);
        return;
    }

    // Yet another hack...
    if ( (gamemode == registered)
        && (choice > 2))
    {
        fprintf( stderr,
            "M_Episode: 4th episode requires UltimateDOOM\n");
        choice = 0;
    }

    epi = choice;
    M_SetupNextMenu(&NewDef);
}

//

```

```

// M_Options
//
char    detailNames[2][9]      = {"M_GDHIGH","M_GDLOW"};
char    msgNames[2][9]        = {"M_MSGOFF","M_MSGON"};

void M_DrawOptions(void)
{
    V_DrawPatchDirect (108,15,0,W_CacheLumpName("M_OPTTTL",PU_CACHE));

    V_DrawPatchDirect (OptionsDef.x + 175,OptionsDef.y+LINEHEIGHT*detail,0,
                        W_CacheLumpName(detailNames[detailLevel],PU_CACHE));

    V_DrawPatchDirect (OptionsDef.x + 120,OptionsDef.y+LINEHEIGHT*messages,0,
                        W_CacheLumpName(msgNames[showMessages],PU_CACHE));

    M_DrawThermo(OptionsDef.x,OptionsDef.y+LINEHEIGHT*(mousesens+1),
                  10,mouseSensitivity);

    M_DrawThermo(OptionsDef.x,OptionsDef.y+LINEHEIGHT*(scrnsize+1),
                  9,screenSize);
}

void M_Options(int choice)
{
    M_SetupNextMenu(&OptionsDef);
}

//
//      Toggle messages on/off
//
void M_ChangeMessages(int choice)
{
    // warning: unused parameter 'int choice'
    choice = 0;
    showMessages = 1 - showMessages;

    if (!showMessages)
        players[consoleplayer].message = MSGOFF;
    else
        players[consoleplayer].message = MSGON ;

    message_dontfuckwithme = true;
}

//
// M_EndGame
//
void M_EndGameResponse(int ch)
{
    if (ch != 'y')
        return;

    currentMenu->lastOn = itemOn;
    M_ClearMenus ();
    D_StartTitle ();
}

void M_EndGame(int choice)
{
    choice = 0;
    if (!usergame)

```

```

{
    S_StartSound(NULL,sfx_oof);
    return;
}

if (netgame)
{
    M_StartMessage(NETEND,NULL,false);
    return;
}

M_StartMessage(ENDGAME,M_EndGameResponse,true);
}

```

```

//
// M_ReadThis
//
void M_ReadThis(int choice)
{
    choice = 0;
    M_SetupNextMenu(&ReadDef1);
}

void M_ReadThis2(int choice)
{
    choice = 0;
    M_SetupNextMenu(&ReadDef2);
}

void M_FinishReadThis(int choice)
{
    choice = 0;
    M_SetupNextMenu(&MainDef);
}

```

```

//
// M_QuitDOOM
//
int      quitsounds[8] =
{
    sfx_pldeth,
    sfx_dmpain,
    sfx_popain,
    sfx_slop,
    sfx_telept,
    sfx_posit1,
    sfx_posit3,
    sfx_sgtatk
};

int      quitsounds2[8] =
{
    sfx_vilact,
    sfx_getpow,
    sfx_boscub,
    sfx_slop,
    sfx_skeswg,
    sfx_kntdth,
    sfx_bspact,

```

```

    sfx_sgtatk
};

void M_QuitResponse(int ch)
{
    if (ch != 'y')
        return;
    if (!netgame)
    {
        if (gamemode == commercial)
            S_StartSound(NULL,quitsounds2[(gametic>>2)&7]);
        else
            S_StartSound(NULL,quitsounds[(gametic>>2)&7]);
        I_WaitVBL(105);
    }
    I_Quit ();
}

void M_QuitDOOM(int choice)
{
    // We pick index 0 which is language sensitive,
    // or one at random, between 1 and maximum number.
    if (language != english )
        sprintf(endstring,"%s\n\n"DOSY, endmsg[0] );
    else
        sprintf(endstring,"%s\n\n"DOSY, endmsg[ (gametic%(NUM_QUITMESSAGES-2))+1 ] );

    M_StartMessage(endstring,M_QuitResponse,true);
}

void M_ChangeSensitivity(int choice)
{
    switch(choice)
    {
        case 0:
            if (mouseSensitivity)
                mouseSensitivity--;
            break;
        case 1:
            if (mouseSensitivity < 9)
                mouseSensitivity++;
            break;
    }
}

void M_ChangeDetail(int choice)
{
    choice = 0;
    detailLevel = 1 - detailLevel;

    // FIXME - does not work. Remove anyway?
    fprintf( stderr, "M_ChangeDetail: low detail mode n.a.\n");

    return;
}

```

```

/*R_SetViewSize (screenblocks, detailLevel);

if (!detailLevel)
    players[consoleplayer].message = DETAILHI;
else
    players[consoleplayer].message = DETAILLO;*/
}

void M_SizeDisplay(int choice)
{
    switch(choice)
    {
        case 0:
            if (screenSize > 0)
            {
                screenblocks--;
                screenSize--;
            }
            break;
        case 1:
            if (screenSize < 8)
            {
                screenblocks++;
                screenSize++;
            }
            break;
    }

    R_SetViewSize (screenblocks, detailLevel);
}

//
//      Menu Functions
//
void
M_DrawThermo
( int      x,
  int      y,
  int      thermWidth,
  int      thermDot )
{
    int      xx;
    int      i;

    xx = x;
    V_DrawPatchDirect (xx,y,0,W_CacheLumpName("M_THERML",PU_CACHE));
    xx += 8;
    for (i=0;i<thermWidth;i++)
    {
        V_DrawPatchDirect (xx,y,0,W_CacheLumpName("M_THERMM",PU_CACHE));
        xx += 8;
    }
    V_DrawPatchDirect (xx,y,0,W_CacheLumpName("M_THERMR",PU_CACHE));

    V_DrawPatchDirect ((x+8) + thermDot*8,y,
                        0,W_CacheLumpName("M_THERMO",PU_CACHE));
}

```

```

void
M_DrawEmptyCell
( menu_t*      menu,
  int          item )
{
    V_DrawPatchDirect (menu->x - 10,      menu->y+item*LINEHEIGHT - 1, 0,
                        W_CacheLumpName("M_CELL1",PU_CACHE));
}

void
M_DrawSelCell
( menu_t*      menu,
  int          item )
{
    V_DrawPatchDirect (menu->x - 10,      menu->y+item*LINEHEIGHT - 1, 0,
                        W_CacheLumpName("M_CELL2",PU_CACHE));
}

void
M_StartMessage
( char*        string,
  void*        routine,
  boolean      input )
{
    messageLastMenuActive = menuactive;
    messageToPrint = 1;
    messageString = string;
    messageRoutine = routine;
    messageNeedsInput = input;
    menuactive = true;
    return;
}

void M_StopMessage(void)
{
    menuactive = messageLastMenuActive;
    messageToPrint = 0;
}

//
// Find string width from hu_font chars
//
int M_StringWidth(char* string)
{
    int      i;
    int      w = 0;
    int      c;

    for (i = 0; i < strlen(string); i++)
    {
        c = toupper(string[i]) - HU_FONTSTART;
        if (c < 0 || c >= HU_FONTSIZE)
            w += 4;
        else
            w += SHORT (hu_font[c]->width);
    }
}

```

```

    return w;
}

//
//      Find string height from hu_font chars
//
int M_StringHeight(char* string)
{
    int          i;
    int          h;
    int          height = SHORT(hu_font[0]->height);

    h = height;
    for (i = 0; i < strlen(string); i++)
        if (string[i] == '\n')
            h += height;

    return h;
}

//
//      Write a string using the hu_font
//
void
M_WriteText
( int          x,
  int          y,
  char*        string)
{
    int          w;
    char*        ch;
    int          c;
    int          cx;
    int          cy;

    ch = string;
    cx = x;
    cy = y;

    while(1)
    {
        c = *ch++;
        if (!c)
            break;
        if (c == '\n')
        {
            cx = x;
            cy += 12;
            continue;
        }

        c = toupper(c) - HU_FONTSTART;
        if (c < 0 || c >= HU_FONTSIZE)
        {
            cx += 4;
            continue;
        }

        w = SHORT (hu_font[c]->width);
        if (cx+w > SCREENWIDTH)
            break;
    }
}

```

```

        V_DrawPatchDirect(cx, cy, 0, hu_font[c]);
        cx+=w;
    }
}

//
// CONTROL PANEL
//

//
// M_Responder
//
boolean M_Responder (event_t* ev)
{
    int            ch;
    int            i;
    static int     joywait = 0;
    static int     mousewait = 0;
    static int     mousey = 0;
    static int     lasty = 0;
    static int     mousex = 0;
    static int     lastx = 0;

    ch = -1;

    if (ev->type == ev_joystick && joywait < I_GetTime())
    {
        if (ev->data3 == -1)
        {
            ch = KEY_UPARROW;
            joywait = I_GetTime() + 5;
        }
        else if (ev->data3 == 1)
        {
            ch = KEY_DOWNARROW;
            joywait = I_GetTime() + 5;
        }

        if (ev->data2 == -1)
        {
            ch = KEY_LEFTARROW;
            joywait = I_GetTime() + 2;
        }
        else if (ev->data2 == 1)
        {
            ch = KEY_RIGHTARROW;
            joywait = I_GetTime() + 2;
        }

        if (ev->data1&1)
        {
            ch = KEY_ENTER;
            joywait = I_GetTime() + 5;
        }
        if (ev->data1&2)
        {
            ch = KEY_BACKSPACE;
            joywait = I_GetTime() + 5;
        }
    }
    else
    {
        if (ev->type == ev_mouse && mousewait < I_GetTime())

```



```

{
    mousey += ev->data3;
    if (mousey < lasty-30)
    {
        ch = KEY_DOWNARROW;
        mousewait = I_GetTime() + 5;
        mousey = lasty -= 30;
    }
    else if (mousey > lasty+30)
    {
        ch = KEY_UPARROW;
        mousewait = I_GetTime() + 5;
        mousey = lasty += 30;
    }

    mousex += ev->data2;
    if (mousex < lastx-30)
    {
        ch = KEY_LEFTARROW;
        mousewait = I_GetTime() + 5;
        mousex = lastx -= 30;
    }
    else if (mousex > lastx+30)
    {
        ch = KEY_RIGHTARROW;
        mousewait = I_GetTime() + 5;
        mousex = lastx += 30;
    }

    if (ev->data1&1)
    {
        ch = KEY_ENTER;
        mousewait = I_GetTime() + 15;
    }

    if (ev->data1&2)
    {
        ch = KEY_BACKSPACE;
        mousewait = I_GetTime() + 15;
    }
}
else
    if (ev->type == ev_keydown)
    {
        ch = ev->data1;
    }
}

if (ch == -1)
    return false;

// Save Game string input
if (saveStringEnter)
{
    switch(ch)
    {
        case KEY_BACKSPACE:
            if (saveCharIndex > 0)
            {
                saveCharIndex--;
                savegamestrings[saveSlot][saveCharIndex] = 0;
            }
            break;

```

```

    case KEY_ESCAPE:
        saveStringEnter = 0;
        strcpy(&savegamestrings[saveSlot][0], saveOldString);
        break;

    case KEY_ENTER:
        saveStringEnter = 0;
        if (savegamestrings[saveSlot][0])
            M_DoSave(saveSlot);
        break;

    default:
        ch = toupper(ch);
        if (ch != 32)
            if (ch-HU_FONTSTART < 0 || ch-HU_FONTSTART >= HU_FONTSIZE)
                break;
        if (ch >= 32 && ch <= 127 &&
            saveCharIndex < SAVESTRINGSIZE-1 &&
            M_StringWidth(savegamestrings[saveSlot]) <
            (SAVESTRINGSIZE-2)*8)
        {
            savegamestrings[saveSlot][saveCharIndex++] = ch;
            savegamestrings[saveSlot][saveCharIndex] = 0;
        }
        break;
    }
    return true;
}

// Take care of any messages that need input
if (messageToPrint)
{
    if (messageNeedsInput == true &&
        !(ch == ' ' || ch == '\n' || ch == '\t' || ch == KEY_ESCAPE))
        return false;

    menuactive = messageLastMenuActive;
    messageToPrint = 0;
    if (messageRoutine)
        messageRoutine(ch);

    menuactive = false;
    S_StartSound(NULL, sfx_swtx);
    return true;
}

if (devparm && ch == KEY_F1)
{
    G_ScreenShot ();
    return true;
}

// F-Keys
if (!menuactive)
    switch(ch)
    {
        case KEY_MINUS: // Screen size down
            if (automapactive || chat_on)
                return false;
            M_SizeDisplay(0);
            S_StartSound(NULL, sfx_stnmov);
            return true;

        case KEY_EQUALS: // Screen size up

```

```

    if (automapactive || chat_on)
        return false;
    M_SizeDisplay(1);
    S_StartSound(NULL,sfx_stnmov);
    return true;

case KEY_F1:          // Help key
    M_StartControlPanel ();

    if ( gamemode == retail )
        currentMenu = &ReadDef2;
    else
        currentMenu = &ReadDef1;

    itemOn = 0;
    S_StartSound(NULL,sfx_swtn);
    return true;

case KEY_F2:          // Save
    M_StartControlPanel();
    S_StartSound(NULL,sfx_swtn);
    M_SaveGame(0);
    return true;

case KEY_F3:          // Load
    M_StartControlPanel();
    S_StartSound(NULL,sfx_swtn);
    M_LoadGame(0);
    return true;

case KEY_F4:          // Sound Volume
    M_StartControlPanel ();
    currentMenu = &SoundDef;
    itemOn = sfx_vol;
    S_StartSound(NULL,sfx_swtn);
    return true;

case KEY_F5:          // Detail toggle
    M_ChangeDetail(0);
    S_StartSound(NULL,sfx_swtn);
    return true;

case KEY_F6:          // Quicksave
    S_StartSound(NULL,sfx_swtn);
    M_QuickSave();
    return true;

case KEY_F7:          // End game
    S_StartSound(NULL,sfx_swtn);
    M_EndGame(0);
    return true;

case KEY_F8:          // Toggle messages
    M_ChangeMessages(0);
    S_StartSound(NULL,sfx_swtn);
    return true;

case KEY_F9:          // Quickload
    S_StartSound(NULL,sfx_swtn);
    M_QuickLoad();
    return true;

case KEY_F10:         // Quit DOOM
    S_StartSound(NULL,sfx_swtn);
    M_QuitDOOM(0);

```

```

        return true;

    case KEY_F11:          // gamma toggle
        usegamma++;
        if (usegamma > 4)
            usegamma = 0;
        players[consoleplayer].message = gammamsg[usegamma];
        I_SetPalette (W_CacheLumpName ("PLAYPAL",PU_CACHE));
        return true;

}

// Pop-up menu?
if (!menuactive)
{
    if (ch == KEY_ESCAPE)
    {
        M_StartControlPanel ();
        S_StartSound(NULL,sfx_swtn);
        return true;
    }
    return false;
}

// Keys usable within menu
switch (ch)
{
    case KEY_DOWNARROW:
        do
        {
            if (itemOn+1 > currentMenu->numitems-1)
                itemOn = 0;
            else itemOn++;
            S_StartSound(NULL,sfx_pstop);
        } while(currentMenu->menuitems[itemOn].status==1);
        return true;

    case KEY_UPARROW:
        do
        {
            if (!itemOn)
                itemOn = currentMenu->numitems-1;
            else itemOn--;
            S_StartSound(NULL,sfx_pstop);
        } while(currentMenu->menuitems[itemOn].status==1);
        return true;

    case KEY_LEFTARROW:
        if (currentMenu->menuitems[itemOn].routine &&
            currentMenu->menuitems[itemOn].status == 2)
        {
            S_StartSound(NULL,sfx_stnmov);
            currentMenu->menuitems[itemOn].routine(0);
        }
        return true;

    case KEY_RIGHTARROW:
        if (currentMenu->menuitems[itemOn].routine &&
            currentMenu->menuitems[itemOn].status == 2)
        {
            S_StartSound(NULL,sfx_stnmov);
            currentMenu->menuitems[itemOn].routine(1);
        }
}

```

```

    return true;

case KEY_ENTER:
    if (currentMenu->menuitems[itemOn].routine &&
        currentMenu->menuitems[itemOn].status)
    {
        currentMenu->lastOn = itemOn;
        if (currentMenu->menuitems[itemOn].status == 2)
        {
            currentMenu->menuitems[itemOn].routine(1);    // right arrow
            S_StartSound(NULL,sfx_stnmov);
        }
        else
        {
            currentMenu->menuitems[itemOn].routine(itemOn);
            S_StartSound(NULL,sfx_pistol);
        }
    }
    return true;

case KEY_ESCAPE:
    currentMenu->lastOn = itemOn;
    M_ClearMenus ();
    S_StartSound(NULL,sfx_swtxch);
    return true;

case KEY_BACKSPACE:
    currentMenu->lastOn = itemOn;
    if (currentMenu->prevMenu)
    {
        currentMenu = currentMenu->prevMenu;
        itemOn = currentMenu->lastOn;
        S_StartSound(NULL,sfx_swtxchn);
    }
    return true;

default:
    for (i = itemOn+1; i < currentMenu->numitems; i++)
        if (currentMenu->menuitems[i].alphaKey == ch)
        {
            itemOn = i;
            S_StartSound(NULL,sfx_pstop);
            return true;
        }
    for (i = 0; i <= itemOn; i++)
        if (currentMenu->menuitems[i].alphaKey == ch)
        {
            itemOn = i;
            S_StartSound(NULL,sfx_pstop);
            return true;
        }
    break;
}

return false;
}

//
// M_StartControlPanel
//
void M_StartControlPanel (void)
{

```

```

// intro might call this repeatedly
if (menuactive)
    return;

menuactive = 1;
currentMenu = &MainDef;      // JDC
itemOn = currentMenu->lastOn; // JDC
}

//
// M_Drawer
// Called after the view has been rendered,
// but before it has been blitted.
//
void M_Drawer (void)
{
    static short      x;
    static short      y;
    short             i;
    short             max;
    char              string[40];
    int               start;

    inhelpscreens = false;

    // Horiz. & Vertically center string and print it.
    if (messageToPrint)
    {
        start = 0;
        y = 100 - M_StringHeight(messageString)/2;
        while(*(messageString+start))
        {
            for (i = 0; i < strlen(messageString+start); i++)
                if (*(messageString+start+i) == '\n')
                {
                    memset(string,0,40);
                    strncpy(string,messageString+start,i);
                    start += i+1;
                    break;
                }

            if (i == strlen(messageString+start))
            {
                strcpy(string,messageString+start);
                start += i;
            }

            x = 160 - M_StringWidth(string)/2;
            M_WriteText(x,y,string);
            y += SHORT(hu_font[0]->height);
        }
        return;
    }

    if (!menuactive)
        return;

    if (currentMenu->routine)
        currentMenu->routine();      // call Draw routine

    // DRAW MENU
    x = currentMenu->x;
    y = currentMenu->y;

```

```

    max = currentMenu->numitems;

    for (i=0;i<max;i++)
    {
        if (currentMenu->menuitems[i].name[0])
            V_DrawPatchDirect (x,y,0,
                               W_CacheLumpName(currentMenu->menuitems[i].name ,PU_CACHE));
        y += LINEHEIGHT;
    }

    // DRAW SKULL
    V_DrawPatchDirect(x + SKULLXOFF,currentMenu->y - 5 + itemOn*LINEHEIGHT, 0,
                      W_CacheLumpName(skullName[whichSkull],PU_CACHE));
}

//
// M_ClearMenus
//
void M_ClearMenus (void)
{
    menuactive = 0;
    // if (!netgame && usergame && paused)
    //     sendpause = true;
}

//
// M_SetupNextMenu
//
void M_SetupNextMenu(menu_t *menundef)
{
    currentMenu = menundef;
    itemOn = currentMenu->lastOn;
}

//
// M_Ticker
//
void M_Ticker (void)
{
    if (--skullAnimCounter <= 0)
    {
        whichSkull ^= 1;
        skullAnimCounter = 8;
    }
}

//
// M_Init
//
void M_Init (void)
{
    currentMenu = &MainDef;
    menuactive = 0;
    itemOn = currentMenu->lastOn;
    whichSkull = 0;
    skullAnimCounter = 10;
    screenSize = screenblocks - 3;
}

```

```

messageToPrint = 0;
messageString = NULL;
messageLastMenuActive = menuactive;
quickSaveSlot = -1;

// Here we could catch other version dependencies,
// like HELP1/2, and four episodes.

switch ( gamemode )
{
    case commercial:
        // This is used because DOOM 2 had only one HELP
        // page. I use CREDIT as second page now, but
        // kept this hack for educational purposes.
        MainMenu[readthis] = MainMenu[quitdoom];
        MainDef.numitems--;
        MainDef.y += 8;
        NewDef.prevMenu = &MainDef;
        ReadDef1.routine = M_DrawReadThis1;
        ReadDef1.x = 330;
        ReadDef1.y = 165;
        ReadMenu1[0].routine = M_FinishReadThis;
        break;
    case shareware:
        // Episode 2 and 3 are handled,
        // branching to an ad screen.
    case registered:
        // We need to remove the fourth episode.
        EpiDef.numitems--;
        break;
    case retail:
        // We are fine.
    default:
        break;
}
}

```

## 8.10 m\_menu.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//   Menu widget stuff, episode selection and such.
//
//-----

```



```

#ifndef __M_MENU__
#define __M_MENU__

#include "d_event.h"

//
// MENUS
//
// Called by main loop,
// saves config file and calls I_Quit when user exits.
// Even when the menu is not displayed,
// this can resize the view and change game parameters.
// Does all the real work of the menu interaction.
boolean M_Responder (event_t *ev);

// Called by main loop,
// only used for menu (skull cursor) animation.
void M_Ticker (void);

// Called by main loop,
// draws the menus directly into the screen buffer.
void M_Drawer (void);

// Called by D_DoomMain,
// loads the config file.
void M_Init (void);

// Called by intro code to force menu up upon a keypress,
// does nothing if menu is already up.
void M_StartControlPanel (void);

#endif
//-----
//
// $Log:$
//
//-----

```

## 8.11 m\_misc.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//

```

```

//
// $Log:$
//
// DESCRIPTION:
//      Main loop menu stuff.
//      Default Config File.
//      PCX Screenshots.
//
//-----

static const char
rcsid[] = "$Id: m_misc.c,v 1.6 1997/02/03 22:45:10 b1 Exp $";

#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

#include <ctype.h>

#include "doomdef.h"

#include "z_zone.h"

#include "m_swap.h"
#include "m_argv.h"

#include "w_wad.h"

#include "i_system.h"
#include "i_video.h"
#include "v_video.h"

#include "hu_stuff.h"

// State.
#include "doomstat.h"

// Data.
#include "dstrings.h"

#include "m_misc.h"

//
// M_DrawText
// Returns the final X coordinate
// HU_Init must have been called to init the font
//
extern patch_t*          hu_font[HU_FONTSIZE];

int
M_DrawText
( int          x,
  int          y,
  boolean      direct,
  char*        string )
{
    int        c;
    int        w;

    while (*string)
    {
        c = toupper(*string) - HU_FONTSTART;

```

```

        string++;
        if (c < 0 || c> HU_FONTSIZE)
        {
            x += 4;
            continue;
        }

        w = SHORT (hu_font[c]->width);
        if (x+w > SCREENWIDTH)
            break;
        if (direct)
            V_DrawPatchDirect(x, y, 0, hu_font[c]);
        else
            V_DrawPatch(x, y, 0, hu_font[c]);
        x+=w;
    }

    return x;
}

//
// M_WriteFile
//
#ifdef O_BINARY
#define O_BINARY 0
#endif

boolean
M_WriteFile
( char const*      name,
  void*           source,
  int             length )
{
    int            handle;
    int            count;

    handle = open ( name, O_WRONLY | O_CREAT | O_TRUNC | O_BINARY, 0666);

    if (handle == -1)
        return false;

    count = write (handle, source, length);
    close (handle);

    if (count < length)
        return false;

    return true;
}

//
// M_ReadFile
//
int
M_ReadFile
( char const*      name,
  byte**          buffer )
{
    int            handle, count, length;
    struct stat     fileinfo;
    byte           *buf;

```

```

    handle = open (name, O_RDONLY | O_BINARY, 0666);
    if (handle == -1)
        I_Error ("Couldn't read file %s", name);
    if (fstat (handle,&fileinfo) == -1)
        I_Error ("Couldn't read file %s", name);
    length = fileinfo.st_size;
    buf = Z_Malloc (length, PU_STATIC, NULL);
    count = read (handle, buf, length);
    close (handle);

    if (count < length)
        I_Error ("Couldn't read file %s", name);

    *buffer = buf;
    return length;
}

```

```

//
// DEFAULTS
//
int          usemouse;
int          usejoystick;

extern int    key_right;
extern int    key_left;
extern int    key_up;
extern int    key_down;

extern int    key_strafeleft;
extern int    key_straferight;

extern int    key_fire;
extern int    key_use;
extern int    key_strafe;
extern int    key_speed;

extern int    mousebfire;
extern int    mousebstrafe;
extern int    mousebforward;

extern int    joybfire;
extern int    joybstrafe;
extern int    joybuse;
extern int    joybspeed;

extern int    viewwidth;
extern int    viewheight;

extern int    mouseSensitivity;
extern int    showMessages;

extern int    detailLevel;

extern int    screenblocks;

extern int    showMessages;

// machine-independent sound params
extern      int      numChannels;

// UNIX hack, to be removed.
#ifdef SNDSERV

```

```

extern char*      sndserver_filename;
extern int        mb_used;
#endif

#ifdef LINUX
char*             mousetype;
char*             mousedev;
#endif

extern char*      chat_macros[];

typedef struct
{
    char*          name;
    int*           location;
    int            defaultvalue;
    int            scantranslate;          // PC scan code hack
    int            untranslated;           // lousy hack
} default_t;

default_t          defaults[] =
{
    {"mouse_sensitivity",&mouseSensitivity, 5},
    {"sfx_volume",&snd_SfxVolume, 8},
    {"music_volume",&snd_MusicVolume, 8},
    {"show_messages",&showMessages, 1},

#ifdef NORMALUNIX
    {"key_right",&key_right, KEY_RIGHTARROW},
    {"key_left",&key_left, KEY_LEFTARROW},
    {"key_up",&key_up, KEY_UPARROW},
    {"key_down",&key_down, KEY_DOWNARROW},
    {"key_strafeleft",&key_strafeleft, ','},
    {"key_straferight",&key_straferight, '.'},

    {"key_fire",&key_fire, KEY_RCTRL},
    {"key_use",&key_use, ' '},
    {"key_strafe",&key_strafe, KEY_RALT},
    {"key_speed",&key_speed, KEY_RSHIFT},

// UNIX hack, to be removed.
#endif
#ifdef SNDSESV
    {"sndserver", (int *) &sndserver_filename, (int) "sndserver"},
    {"mb_used", &mb_used, 2},
#endif

#ifdef LINUX
    {"mousedev", (int*)&mousedev, (int)"/dev/ttyS0"},
    {"mousetype", (int*)&mousetype, (int)"microsoft"},
#endif

    {"use_mouse",&usemouse, 1},
    {"mouseb_fire",&mousebfire,0},
    {"mouseb_strafe",&mousebstrafe,1},
    {"mouseb_forward",&mousebforward,2},

    {"use_joystick",&usejoystick, 0},
    {"joyb_fire",&joybfire,0},
    {"joyb_strafe",&joybstrafe,1},
    {"joyb_use",&joybuse,3},

```

```

{"joyb_speed",&joybspeed,2},

{"screenblocks",&screenblocks, 9},
{"detaillevel",&detailLevel, 0},

{"snd_channels",&numChannels, 3},


{"usegamma",&usegamma, 0},

{"chatmacro0", (int *) &chat_macros[0], (int) HISTR_CHATMACRO0 },
{"chatmacro1", (int *) &chat_macros[1], (int) HISTR_CHATMACRO1 },
{"chatmacro2", (int *) &chat_macros[2], (int) HISTR_CHATMACRO2 },
{"chatmacro3", (int *) &chat_macros[3], (int) HISTR_CHATMACRO3 },
{"chatmacro4", (int *) &chat_macros[4], (int) HISTR_CHATMACRO4 },
{"chatmacro5", (int *) &chat_macros[5], (int) HISTR_CHATMACRO5 },
{"chatmacro6", (int *) &chat_macros[6], (int) HISTR_CHATMACRO6 },
{"chatmacro7", (int *) &chat_macros[7], (int) HISTR_CHATMACRO7 },
{"chatmacro8", (int *) &chat_macros[8], (int) HISTR_CHATMACRO8 },
{"chatmacro9", (int *) &chat_macros[9], (int) HISTR_CHATMACRO9 }

};

int          numdefaults;
char*        defaultfile;


//
// M_SaveDefaults
//
void M_SaveDefaults (void)
{
    int          i;
    int          v;
    FILE*        f;

    f = fopen (defaultfile, "w");
    if (!f)
        return; // can't write the file, but don't complain

    for (i=0 ; i<numdefaults ; i++)
    {
        if (defaults[i].defaultvalue > -0xfff
            && defaults[i].defaultvalue < 0xfff)
        {
            v = *defaults[i].location;
            fprintf (f,"%s\t\t%i\n",defaults[i].name,v);
        } else {
            fprintf (f,"%s\t\t\t%s\n",defaults[i].name,
                    * (char **) (defaults[i].location));
        }
    }

    fclose (f);
}


//
// M_LoadDefaults
//
extern byte     scantoke[128];

void M_LoadDefaults (void)
{

```

```

int            i;
int            len;
FILE*          f;
char           def[80];
char           strparm[100];
char*          newstring;
int            parm;
boolean        isstring;

// set everything to base values
numdefaults = sizeof(defaults)/sizeof(defaults[0]);
for (i=0 ; i<numdefaults ; i++)
    *defaults[i].location = defaults[i].defaultvalue;

// check for a custom default file
i = M_CheckParm ("-config");
if (i && i<myargc-1)
{
    defaultfile = myargv[i+1];
    printf ("        default file: %s\n",defaultfile);
}
else
    defaultfile = basedefault;

// read the file in, overriding any set defaults
f = fopen (defaultfile, "r");
if (f)
{
    while (!feof(f))
    {
        isstring = false;
        if (fscanf (f, "%79s %[^\n]\n", def, strparm) == 2)
        {
            if (strparm[0] == '"')
            {
                // get a string default
                isstring = true;
                len = strlen(strparm);
                newstring = (char *) malloc(len);
                strparm[len-1] = 0;
                strcpy(newstring, strparm+1);
            }
            else if (strparm[0] == '0' && strparm[1] == 'x')
                sscanf(strparm+2, "%x", &parm);
            else
                sscanf(strparm, "%i", &parm);
            for (i=0 ; i<numdefaults ; i++)
                if (!strcmp(def, defaults[i].name))
                {
                    if (!isstring)
                        *defaults[i].location = parm;
                    else
                        *defaults[i].location =
                            (int) newstring;
                    break;
                }
        }
    }

    fclose (f);
}

}

//

```

```

// SCREEN SHOTS
//

typedef struct
{
    char            manufacturer;
    char            version;
    char            encoding;
    char            bits_per_pixel;

    unsigned short   xmin;
    unsigned short   ymin;
    unsigned short   xmax;
    unsigned short   ymax;

    unsigned short   hres;
    unsigned short   vres;

    unsigned char     palette[48];

    char            reserved;
    char            color_planes;
    unsigned short    bytes_per_line;
    unsigned short    palette_type;

    char            filler[58];
    unsigned char     data;           // unbounded
} pcx_t;

//
// WritePCXfile
//
void
WritePCXfile
( char*            filename,
  byte*            data,
  int              width,
  int              height,
  byte*            palette )
{
    int            i;
    int            length;
    pcx_t*         pcx;
    byte*          pack;

    pcx = Z_Malloc (width*height*2+1000, PU_STATIC, NULL);

    pcx->manufacturer = 0x0a;           // PCX id
    pcx->version = 5;                   // 256 color
    pcx->encoding = 1;                  // uncompressed
    pcx->bits_per_pixel = 8;            // 256 color
    pcx->xmin = 0;
    pcx->ymin = 0;
    pcx->xmax = SHORT(width-1);
    pcx->ymax = SHORT(height-1);
    pcx->hres = SHORT(width);
    pcx->vres = SHORT(height);
    memset (pcx->palette,0,sizeof(pcx->palette));
    pcx->color_planes = 1;              // chunky image
    pcx->bytes_per_line = SHORT(width);
    pcx->palette_type = SHORT(2);       // not a grey scale
    memset (pcx->filler,0,sizeof(pcx->filler));

```



```

// pack the image
pack = &pcx->data;

for (i=0 ; i<width*height ; i++)
{
    if ( (*data & 0xc0) != 0xc0)
        *pack++ = *data++;
    else
    {
        *pack++ = 0xc1;
        *pack++ = *data++;
    }
}

// write the palette
*pack++ = 0x0c;          // palette ID byte
for (i=0 ; i<768 ; i++)
    *pack++ = *palette++;

// write output file
length = pack - (byte *)pcx;
M_WriteFile (filename, pcx, length);

Z_Free (pcx);
}

//
// M_ScreenShot
//
void M_ScreenShot (void)
{
    int          i;
    byte*        linear;
    char         lbmname[12];

    // munge planar buffer to linear
    linear = screens[2];
    I_ReadScreen (linear);

    // find a file name to save it to
    strcpy(lbmname,"DOOM00.pcx");

    for (i=0 ; i<=99 ; i++)
    {
        lbmname[4] = i/10 + '0';
        lbmname[5] = i%10 + '0';
        if (access(lbmname,0) == -1)
            break;          // file doesn't exist
    }
    if (i==100)
        I_Error ("M_ScreenShot: Couldn't create a PCX");

    // save the pcx file
    WritePCXfile (lbmname, linear,
        SCREENWIDTH, SCREENHEIGHT,
        W_CacheLumpName ("PLAYPAL",PU_CACHE));

    players[consoleplayer].message = "screen shot";
}

```

## 8.12 m\_misc.h

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//
//-----

#ifndef __M_MISC__
#define __M_MISC__

#include "doomtype.h"
//
// MISC
//

boolean
M_WriteFile
( char const*      name,
  void*           source,
  int             length );

int
M_ReadFile
( char const*      name,
  byte**          buffer );

void M_ScreenShot (void);

void M_LoadDefaults (void);

void M_SaveDefaults (void);

int
M_DrawText
( int             x,
  int             y,
  boolean         direct,
  char*           string );

#endif
//-----
// $Log:$
```

```

//
//-----

8.13  m_random.c

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Random number LUT.
//
//-----

static const char rcsid[] = "$Id: m_random.c,v 1.1 1997/02/03 22:45:11 b1 Exp $";

//
// M_Random
// Returns a 0-255 number
//
unsigned char rndtable[256] = {
    0,  8, 109, 220, 222, 241, 149, 107,  75, 248, 254, 140,  16,  66 ,
    74,  21, 211,  47,  80, 242, 154,  27, 205, 128, 161,  89,  77,  36 ,
    95, 110,  85,  48, 212, 140, 211, 249,  22,  79, 200,  50,  28, 188 ,
    52, 140, 202, 120,  68, 145,  62,  70, 184, 190,  91, 197, 152, 224 ,
    149, 104,  25, 178, 252, 182, 202, 182, 141, 197,   4,  81, 181, 242 ,
    145,  42,  39, 227, 156, 198, 225, 193, 219,  93, 122, 175, 249,   0 ,
    175, 143,  70, 239,  46, 246, 163,  53, 163, 109, 168, 135,   2, 235 ,
    25,  92,  20, 145, 138,  77,  69, 166,  78, 176, 173, 212, 166, 113 ,
    94, 161,  41,  50, 239,  49, 111, 164,  70,  60,   2,  37, 171,  75 ,
    136, 156,  11,  56,  42, 146, 138, 229,  73, 146,  77,  61,  98, 196 ,
    135, 106,  63, 197, 195,  86,  96, 203, 113, 101, 170, 247, 181, 113 ,
    80, 250, 108,   7, 255, 237, 129, 226,  79, 107, 112, 166, 103, 241 ,
    24, 223, 239, 120, 198,  58,  60,  82, 128,   3, 184,  66, 143, 224 ,
    145, 224,  81, 206, 163,  45,  63,  90, 168, 114,  59,  33, 159,  95 ,
    28, 139, 123,  98, 125, 196,  15,  70, 194, 253,  54,  14, 109, 226 ,
    71,  17, 161,  93, 186,  87, 244, 138,  20,  52, 123, 251,  26,  36 ,
    17,  46,  52, 231, 232,  76,  31, 221,  84,  37, 216, 165, 212, 106 ,
    197, 242,  98,  43,  39, 175, 254, 145, 190,  84, 118, 222, 187, 136 ,
    120, 163, 236, 249
};

int      rndindex = 0;
int      prndindex = 0;

// Which one is deterministic?
int P_Random (void)
{
    prndindex = (prndindex+1)&0xff;

```

```

    return rndtable[prndindex];
}

int M_Random (void)
{
    rndindex = (rndindex+1)&0xff;
    return rndtable[rndindex];
}

void M_ClearRandom (void)
{
    rndindex = prndindex = 0;
}

```

## 8.14 m\_random.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//
//-----

#ifndef __M_RANDOM__
#define __M_RANDOM__

#include "doomtype.h"

// Returns a number from 0 to 255,
// from a lookup table.
int M_Random (void);

// As M_Random, but used only by the play simulation.
int P_Random (void);

// Fix randoms for demos.
void M_ClearRandom (void);

#endif
//-----
//

```

```

// $Log:$
//
//-----

8.15  m_swap.c

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Endianess handling, swapping 16bit and 32bit.
//
//-----

static const char
rcsid[] = "$Id: m_bbox.c,v 1.1 1997/02/03 22:45:10 b1 Exp $";

#ifdef __GNUG__
#pragma implementation "m_swap.h"
#endif
#include "m_swap.h"

// Not needed with big endian.
#ifdef __BIG_ENDIAN__

// Swap 16bit, that is, MSB and LSB byte.
unsigned short SwapSHORT(unsigned short x)
{
    // No masking with 0xFF should be necessary.
    return (x>>8) | (x<<8);
}

// Swapping 32bit.
unsigned long SwapLONG( unsigned long x)
{
    return
        (x>>24)
        | ((x>>8) & 0xff00)
        | ((x<<8) & 0xff0000)
        | (x<<24);
}

#endif

```

## 8.16 m\_swap.h

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      Endianess handling, swapping 16bit and 32bit.
//-----

#ifndef __M_SWAP__
#define __M_SWAP__

#ifdef __GNUG__
#pragma interface
#endif

// Endianess handling.
// WAD files are stored little endian.
#ifdef __BIG_ENDIAN__
short      SwapSHORT(short);
long       SwapLONG(long);
#define SHORT(x)      ((short)SwapSHORT((unsigned short) (x)))
#define LONG(x)       ((long)SwapLONG((unsigned long) (x)))
#else
#define SHORT(x)      (x)
#define LONG(x)       (x)
#endif

#endif

//-----
//
// $Log:$
//-----
```

## 9 Game logic/behaviour

### 9.1 info.c

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
```

```

// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Thing frame/state LUT,
//     generated by multigen utility.
//     This one is the original DOOM version, preserved.
//
//-----

static const char
rcsid[] = "$Id: info.c,v 1.3 1997/01/26 07:45:00 b1 Exp $";

// Data.
#include "sounds.h"
#include "m_fixed.h"

#ifdef __GNUG__
#pragma implementation "info.h"
#endif
#include "info.h"

#include "p_mobj.h"

char *sprnames[NUMSPRITES] = {
    "TROO", "SHTG", "PUNG", "PISG", "PISF", "SHTF", "SHT2", "CHGG", "CHGF", "MISG",
    "MISF", "SAWG", "PLSG", "PLSF", "BFGG", "BFGF", "BLUD", "PUFF", "BAL1", "BAL2",
    "PLSS", "PLSE", "MISL", "BFS1", "BFE1", "BFE2", "TFOG", "IFOG", "PLAY", "POSS",
    "SPOS", "VILE", "FIRE", "FATB", "FBXP", "SKEL", "MANF", "FATT", "CPOS", "SARG",
    "HEAD", "BAL7", "BOSS", "BOS2", "SKUL", "SPID", "BSPI", "APLS", "APBX", "CYBR",
    "PAIN", "SSWV", "KEEN", "BBRN", "BOSF", "ARM1", "ARM2", "BAR1", "BEXP", "FCAN",
    "BON1", "BON2", "BKEY", "RKEY", "YKEY", "BSKU", "RSKU", "YSKU", "STIM", "MEDI",
    "SOUL", "PINV", "PSTR", "PINS", "MEGA", "SUIT", "PMAP", "PVIS", "CLIP", "AMMO",
    "ROCK", "BROK", "CELL", "CELP", "SHEL", "SBOX", "BPAK", "BFUG", "MGUN", "CSAW",
    "LAUN", "PLAS", "SHOT", "SGN2", "COLU", "SMT2", "GOR1", "POL2", "POL5", "POL4",
    "POL3", "POL1", "POL6", "GOR2", "GOR3", "GOR4", "GOR5", "SMIT", "COL1", "COL2",
    "COL3", "COL4", "CAND", "CBRA", "COL6", "TRE1", "TRE2", "ELEC", "CEYE", "FSKU",
    "COL5", "TBLU", "TGRN", "TRED", "SMBT", "SMGT", "SMRT", "HDB1", "HDB2", "HDB3",
    "HDB4", "HDB5", "HDB6", "POB1", "POB2", "BRS1", "TLMP", "TLP2"
};

// Doesn't work with g++, needs actionf_p1
void A_Light0();
void A_WeaponReady();
void A_Lower();
void A_Raise();
void A_Punch();
void A_ReFire();
void A_FirePistol();
void A_Light1();
void A_FireShotgun();
void A_Light2();
void A_FireShotgun2();

```

```

void A_CheckReload();
void A_OpenShotgun2();
void A_LoadShotgun2();
void A_CloseShotgun2();
void A_FireCGun();
void A_GunFlash();
void A_FireMissile();
void A_Saw();
void A_FirePlasma();
void A_BFGsound();
void A_FireBFG();
void A_BFGSpray();
void A_Explode();
void A_Pain();
void A_PlayerScream();
void A_Fall();
void A_XScream();
void A_Look();
void A_Chase();
void A_FaceTarget();
void A_PosAttack();
void A_Scream();
void A_SPosAttack();
void A_VileChase();
void A_VileStart();
void A_VileTarget();
void A_VileAttack();
void A_StartFire();
void A_Fire();
void A_FireCrackle();
void A_Tracer();
void A_SkelWhoosh();
void A_SkelFist();
void A_SkelMissile();
void A_FatRaise();
void A_FatAttack1();
void A_FatAttack2();
void A_FatAttack3();
void A_BossDeath();
void A_CPosAttack();
void A_CPosRefire();
void A_TroopAttack();
void A_SargAttack();
void A_HeadAttack();
void A_BruisAttack();
void A_SkullAttack();
void A_Metal();
void A_SpidRefire();
void A_BabyMetal();
void A_BspiAttack();
void A_Hoof();
void A_CyberAttack();
void A_PainAttack();
void A_PainDie();
void A_KeenDie();
void A_BrainPain();
void A_BrainScream();
void A_BrainDie();
void A_BrainAwake();
void A_BrainSpit();
void A_SpawnSound();
void A_SpawnFly();
void A_BrainExplode();

```



```

state_t      states[NUMSTATES] = {
    {SPR_TROO,0,-1,{NULL},S_NULL,0,0},          // S_NULL
    {SPR_SHTG,4,0,{A_Light0},S_NULL,0,0},       // S_LIGHTDONE
    {SPR_PUNG,0,1,{A_WeaponReady},S_PUNCH,0,0}, // S_PUNCH
    {SPR_PUNG,0,1,{A_Lower},S_PUNCHDOWN,0,0},   // S_PUNCHDOWN
    {SPR_PUNG,0,1,{A_Raise},S_PUNCHUP,0,0},      // S_PUNCHUP
    {SPR_PUNG,1,4,{NULL},S_PUNCH2,0,0},          // S_PUNCH1
    {SPR_PUNG,2,4,{A_Punch},S_PUNCH3,0,0},       // S_PUNCH2
    {SPR_PUNG,3,5,{NULL},S_PUNCH4,0,0},          // S_PUNCH3
    {SPR_PUNG,2,4,{NULL},S_PUNCH5,0,0},          // S_PUNCH4
    {SPR_PUNG,1,5,{A_ReFire},S_PUNCH,0,0},       // S_PUNCH5
    {SPR_PISG,0,1,{A_WeaponReady},S_PISTOL,0,0}, // S_PISTOL
    {SPR_PISG,0,1,{A_Lower},S_PISTOLDOWN,0,0},   // S_PISTOLDOWN
    {SPR_PISG,0,1,{A_Raise},S_PISTOLUP,0,0},     // S_PISTOLUP
    {SPR_PISG,0,4,{NULL},S_PISTOL2,0,0},         // S_PISTOL1
    {SPR_PISG,1,6,{A_FirePistol},S_PISTOL3,0,0}, // S_PISTOL2
    {SPR_PISG,2,4,{NULL},S_PISTOL4,0,0},         // S_PISTOL3
    {SPR_PISG,1,5,{A_ReFire},S_PISTOL,0,0},     // S_PISTOL4
    {SPR_PISF,32768,7,{A_Light1},S_LIGHTDONE,0,0}, // S_PISTOLFLASH
    {SPR_SHTG,0,1,{A_WeaponReady},S_SGUN,0,0},   // S_SGUN
    {SPR_SHTG,0,1,{A_Lower},S_SGUNDOWN,0,0},     // S_SGUNDOWN
    {SPR_SHTG,0,1,{A_Raise},S_SGUNUP,0,0},       // S_SGUNUP
    {SPR_SHTG,0,3,{NULL},S_SGUN2,0,0},           // S_SGUN1
    {SPR_SHTG,0,7,{A_FireShotgun},S_SGUN3,0,0},   // S_SGUN2
    {SPR_SHTG,1,5,{NULL},S_SGUN4,0,0},           // S_SGUN3
    {SPR_SHTG,2,5,{NULL},S_SGUN5,0,0},           // S_SGUN4
    {SPR_SHTG,3,4,{NULL},S_SGUN6,0,0},           // S_SGUN5
    {SPR_SHTG,2,5,{NULL},S_SGUN7,0,0},           // S_SGUN6
    {SPR_SHTG,1,5,{NULL},S_SGUN8,0,0},           // S_SGUN7
    {SPR_SHTG,0,3,{NULL},S_SGUN9,0,0},           // S_SGUN8
    {SPR_SHTG,0,7,{A_ReFire},S_SGUN,0,0},        // S_SGUN9
    {SPR_SHTF,32768,4,{A_Light1},S_SGUNFLASH2,0,0}, // S_SGUNFLASH1
    {SPR_SHTF,32769,3,{A_Light2},S_LIGHTDONE,0,0}, // S_SGUNFLASH2
    {SPR_SHT2,0,1,{A_WeaponReady},S_DSGUN,0,0},   // S_DSGUN
    {SPR_SHT2,0,1,{A_Lower},S_DSGUNDOWN,0,0},     // S_DSGUNDOWN
    {SPR_SHT2,0,1,{A_Raise},S_DSGUNUP,0,0},       // S_DSGUNUP
    {SPR_SHT2,0,3,{NULL},S_DSGUN2,0,0},           // S_DSGUN1
    {SPR_SHT2,0,7,{A_FireShotgun2},S_DSGUN3,0,0}, // S_DSGUN2
    {SPR_SHT2,1,7,{NULL},S_DSGUN4,0,0},           // S_DSGUN3
    {SPR_SHT2,2,7,{A_CheckReload},S_DSGUN5,0,0}, // S_DSGUN4
    {SPR_SHT2,3,7,{A_OpenShotgun2},S_DSGUN6,0,0}, // S_DSGUN5
    {SPR_SHT2,4,7,{NULL},S_DSGUN7,0,0},           // S_DSGUN6
    {SPR_SHT2,5,7,{A_LoadShotgun2},S_DSGUN8,0,0}, // S_DSGUN7
    {SPR_SHT2,6,6,{NULL},S_DSGUN9,0,0},           // S_DSGUN8
    {SPR_SHT2,7,6,{A_CloseShotgun2},S_DSGUN10,0,0}, // S_DSGUN9
    {SPR_SHT2,0,5,{A_ReFire},S_DSGUN,0,0},        // S_DSGUN10
    {SPR_SHT2,1,7,{NULL},S_DSNR2,0,0},            // S_DSNR1
    {SPR_SHT2,0,3,{NULL},S_DSGUNDOWN,0,0},        // S_DSNR2
    {SPR_SHT2,32776,5,{A_Light1},S_DSGUNFLASH2,0,0}, // S_DSGUNFLASH1
    {SPR_SHT2,32777,4,{A_Light2},S_LIGHTDONE,0,0}, // S_DSGUNFLASH2
    {SPR_CHGG,0,1,{A_WeaponReady},S_CHAIN,0,0},   // S_CHAIN
    {SPR_CHGG,0,1,{A_Lower},S_CHAINDOWN,0,0},     // S_CHAINDOWN
    {SPR_CHGG,0,1,{A_Raise},S_CHAINUP,0,0},       // S_CHAINUP
    {SPR_CHGG,0,4,{A_FireCGun},S_CHAIN2,0,0},     // S_CHAIN1
    {SPR_CHGG,1,4,{A_FireCGun},S_CHAIN3,0,0},     // S_CHAIN2
    {SPR_CHGG,1,0,{A_ReFire},S_CHAIN,0,0},        // S_CHAIN3
    {SPR_CHGF,32768,5,{A_Light1},S_LIGHTDONE,0,0}, // S_CHAINFLASH1
    {SPR_CHGF,32769,5,{A_Light2},S_LIGHTDONE,0,0}, // S_CHAINFLASH2
    {SPR_MISG,0,1,{A_WeaponReady},S_MISSILE,0,0}, // S_MISSILE
    {SPR_MISG,0,1,{A_Lower},S_MISSILEDOWN,0,0},   // S_MISSILEDOWN
    {SPR_MISG,0,1,{A_Raise},S_MISSILEUP,0,0},     // S_MISSILEUP
    {SPR_MISG,1,8,{A_GunFlash},S_MISSILE2,0,0},   // S_MISSILE1
    {SPR_MISG,1,12,{A_FireMissile},S_MISSILE3,0,0}, // S_MISSILE2
    {SPR_MISG,1,0,{A_ReFire},S_MISSILE,0,0},      // S_MISSILE3
    {SPR_MISF,32768,3,{A_Light1},S_MISSILEFLASH2,0,0}, // S_MISSILEFLASH1

```

```

{SPR_MISF,32769,4,{NULL},S_MISSILEFLASH3,0,0},          // S_MISSILEFLASH2
{SPR_MISF,32770,4,{A_Light2},S_MISSILEFLASH4,0,0},      // S_MISSILEFLASH3
{SPR_MISF,32771,4,{A_Light2},S_LIGHTDONE,0,0},          // S_MISSILEFLASH4
{SPR_SAWG,2,4,{A_WeaponReady},S_SAWB,0,0},              // S_SAW
{SPR_SAWG,3,4,{A_WeaponReady},S_SAW,0,0},               // S_SAWB
{SPR_SAWG,2,1,{A_Lower},S_SAWDOWN,0,0},                 // S_SAWDOWN
{SPR_SAWG,2,1,{A_Raise},S_SAWUP,0,0},                   // S_SAWUP
{SPR_SAWG,0,4,{A_Saw},S_SAW2,0,0},                      // S_SAW1
{SPR_SAWG,1,4,{A_Saw},S_SAW3,0,0},                      // S_SAW2
{SPR_SAWG,1,0,{A_ReFire},S_SAW,0,0},                    // S_SAW3
{SPR_PLSG,0,1,{A_WeaponReady},S_PLASMA,0,0},             // S_PLASMA
{SPR_PLSG,0,1,{A_Lower},S_PLASMADOWN,0,0},              // S_PLASMADOWN
{SPR_PLSG,0,1,{A_Raise},S_PLASMAUP,0,0},                // S_PLASMAUP
{SPR_PLSG,0,3,{A_FirePlasma},S_PLASMA2,0,0},            // S_PLASMA1
{SPR_PLSG,1,20,{A_ReFire},S_PLASMA,0,0},                // S_PLASMA2
{SPR_PLSF,32768,4,{A_Light1},S_LIGHTDONE,0,0},          // S_PLASMAFLASH1
{SPR_PLSF,32769,4,{A_Light1},S_LIGHTDONE,0,0},          // S_PLASMAFLASH2
{SPR_BFGG,0,1,{A_WeaponReady},S_BFG,0,0},               // S_BFG
{SPR_BFGG,0,1,{A_Lower},S_BFGDOWN,0,0},                 // S_BFGDOWN
{SPR_BFGG,0,1,{A_Raise},S_BFGUP,0,0},                   // S_BFGUP
{SPR_BFGG,0,20,{A_BFGsound},S_BFG2,0,0},                // S_BFG1
{SPR_BFGG,1,10,{A_GunFlash},S_BFG3,0,0},                // S_BFG2
{SPR_BFGG,1,10,{A_FireBFG},S_BFG4,0,0},                 // S_BFG3
{SPR_BFGG,1,20,{A_ReFire},S_BFG,0,0},                   // S_BFG4
{SPR_BFGF,32768,11,{A_Light1},S_BFGFLASH2,0,0},         // S_BFGFLASH1
{SPR_BFGF,32769,6,{A_Light2},S_LIGHTDONE,0,0},          // S_BFGFLASH2
{SPR_BLUD,2,8,{NULL},S_BLOOD2,0,0},                      // S_BLOOD1
{SPR_BLUD,1,8,{NULL},S_BLOOD3,0,0},                      // S_BLOOD2
{SPR_BLUD,0,8,{NULL},S_NULL,0,0},                        // S_BLOOD3
{SPR_PUFF,32768,4,{NULL},S_PUFF2,0,0},                   // S_PUFF1
{SPR_PUFF,1,4,{NULL},S_PUFF3,0,0},                       // S_PUFF2
{SPR_PUFF,2,4,{NULL},S_PUFF4,0,0},                       // S_PUFF3
{SPR_PUFF,3,4,{NULL},S_NULL,0,0},                        // S_PUFF4
{SPR_BAL1,32768,4,{NULL},S_TBALL2,0,0},                  // S_TBALL1
{SPR_BAL1,32769,4,{NULL},S_TBALL1,0,0},                  // S_TBALL2
{SPR_BAL1,32770,6,{NULL},S_TBALLX2,0,0},                 // S_TBALLX1
{SPR_BAL1,32771,6,{NULL},S_TBALLX3,0,0},                 // S_TBALLX2
{SPR_BAL1,32772,6,{NULL},S_NULL,0,0},                     // S_TBALLX3
{SPR_BAL2,32768,4,{NULL},S_RBALL2,0,0},                  // S_RBALL1
{SPR_BAL2,32769,4,{NULL},S_RBALL1,0,0},                  // S_RBALL2
{SPR_BAL2,32770,6,{NULL},S_RBALLX2,0,0},                 // S_RBALLX1
{SPR_BAL2,32771,6,{NULL},S_RBALLX3,0,0},                 // S_RBALLX2
{SPR_BAL2,32772,6,{NULL},S_NULL,0,0},                     // S_RBALLX3
{SPR_PLSS,32768,6,{NULL},S_PLASBALL2,0,0},               // S_PLASBALL
{SPR_PLSS,32769,6,{NULL},S_PLASBALL,0,0},                // S_PLASBALL2
{SPR_PLSE,32768,4,{NULL},S_PLASEXP2,0,0},                // S_PLASEXP
{SPR_PLSE,32769,4,{NULL},S_PLASEXP3,0,0},                // S_PLASEXP2
{SPR_PLSE,32770,4,{NULL},S_PLASEXP4,0,0},                // S_PLASEXP3
{SPR_PLSE,32771,4,{NULL},S_PLASEXP5,0,0},                // S_PLASEXP4
{SPR_PLSE,32772,4,{NULL},S_NULL,0,0},                     // S_PLASEXP5
{SPR_MISL,32768,1,{NULL},S_ROCKET,0,0},                  // S_ROCKET
{SPR_BFS1,32768,4,{NULL},S_BFGSHOT2,0,0},                // S_BFGSHOT
{SPR_BFS1,32769,4,{NULL},S_BFGSHOT,0,0},                 // S_BFGSHOT2
{SPR_BFE1,32768,8,{NULL},S_BFGGLAND2,0,0},               // S_BFGGLAND
{SPR_BFE1,32769,8,{NULL},S_BFGGLAND3,0,0},               // S_BFGGLAND2
{SPR_BFE1,32770,8,{A_BFGSpray},S_BFGGLAND4,0,0},        // S_BFGGLAND3
{SPR_BFE1,32771,8,{NULL},S_BFGGLAND5,0,0},               // S_BFGGLAND4
{SPR_BFE1,32772,8,{NULL},S_BFGGLAND6,0,0},               // S_BFGGLAND5
{SPR_BFE1,32773,8,{NULL},S_NULL,0,0},                     // S_BFGGLAND6
{SPR_BFE2,32768,8,{NULL},S_BFGEXP2,0,0},                 // S_BFGEXP
{SPR_BFE2,32769,8,{NULL},S_BFGEXP3,0,0},                 // S_BFGEXP2
{SPR_BFE2,32770,8,{NULL},S_BFGEXP4,0,0},                 // S_BFGEXP3
{SPR_BFE2,32771,8,{NULL},S_NULL,0,0},                     // S_BFGEXP4
{SPR_MISL,32769,8,{A_Explode},S_EXPLODE2,0,0},           // S_EXPLODE1
{SPR_MISL,32770,6,{NULL},S_EXPLODE3,0,0},                // S_EXPLODE2

```

```

{SPR_MISL,32771,4,{NULL},S_NULL,0,0}, // S_EXPLODE3
{SPR_TFOG,32768,6,{NULL},S_TFOG01,0,0}, // S_TFOG
{SPR_TFOG,32769,6,{NULL},S_TFOG02,0,0}, // S_TFOG01
{SPR_TFOG,32768,6,{NULL},S_TFOG2,0,0}, // S_TFOG02
{SPR_TFOG,32769,6,{NULL},S_TFOG3,0,0}, // S_TFOG2
{SPR_TFOG,32770,6,{NULL},S_TFOG4,0,0}, // S_TFOG3
{SPR_TFOG,32771,6,{NULL},S_TFOG5,0,0}, // S_TFOG4
{SPR_TFOG,32772,6,{NULL},S_TFOG6,0,0}, // S_TFOG5
{SPR_TFOG,32773,6,{NULL},S_TFOG7,0,0}, // S_TFOG6
{SPR_TFOG,32774,6,{NULL},S_TFOG8,0,0}, // S_TFOG7
{SPR_TFOG,32775,6,{NULL},S_TFOG9,0,0}, // S_TFOG8
{SPR_TFOG,32776,6,{NULL},S_TFOG10,0,0}, // S_TFOG9
{SPR_TFOG,32777,6,{NULL},S_NULL,0,0}, // S_TFOG10
{SPR_IFOG,32768,6,{NULL},S_IFOG01,0,0}, // S_IFOG
{SPR_IFOG,32769,6,{NULL},S_IFOG02,0,0}, // S_IFOG01
{SPR_IFOG,32768,6,{NULL},S_IFOG2,0,0}, // S_IFOG02
{SPR_IFOG,32769,6,{NULL},S_IFOG3,0,0}, // S_IFOG2
{SPR_IFOG,32770,6,{NULL},S_IFOG4,0,0}, // S_IFOG3
{SPR_IFOG,32771,6,{NULL},S_IFOG5,0,0}, // S_IFOG4
{SPR_IFOG,32772,6,{NULL},S_NULL,0,0}, // S_IFOG5
{SPR_PLAY,0,-1,{NULL},S_NULL,0,0}, // S_PLAY
{SPR_PLAY,0,4,{NULL},S_PLAY_RUN2,0,0}, // S_PLAY_RUN1
{SPR_PLAY,1,4,{NULL},S_PLAY_RUN3,0,0}, // S_PLAY_RUN2
{SPR_PLAY,2,4,{NULL},S_PLAY_RUN4,0,0}, // S_PLAY_RUN3
{SPR_PLAY,3,4,{NULL},S_PLAY_RUN1,0,0}, // S_PLAY_RUN4
{SPR_PLAY,4,12,{NULL},S_PLAY,0,0}, // S_PLAY_ATK1
{SPR_PLAY,32773,6,{NULL},S_PLAY_ATK1,0,0}, // S_PLAY_ATK2
{SPR_PLAY,6,4,{NULL},S_PLAY_PAIN2,0,0}, // S_PLAY_PAIN
{SPR_PLAY,6,4,{A_Pain},S_PLAY,0,0}, // S_PLAY_PAIN2
{SPR_PLAY,7,10,{NULL},S_PLAY_DIE2,0,0}, // S_PLAY_DIE1
{SPR_PLAY,8,10,{A_PlayerScream},S_PLAY_DIE3,0,0}, // S_PLAY_DIE2
{SPR_PLAY,9,10,{A_Fall},S_PLAY_DIE4,0,0}, // S_PLAY_DIE3
{SPR_PLAY,10,10,{NULL},S_PLAY_DIE5,0,0}, // S_PLAY_DIE4
{SPR_PLAY,11,10,{NULL},S_PLAY_DIE6,0,0}, // S_PLAY_DIE5
{SPR_PLAY,12,10,{NULL},S_PLAY_DIE7,0,0}, // S_PLAY_DIE6
{SPR_PLAY,13,-1,{NULL},S_NULL,0,0}, // S_PLAY_DIE7
{SPR_PLAY,14,5,{NULL},S_PLAY_XDIE2,0,0}, // S_PLAY_XDIE1
{SPR_PLAY,15,5,{A_XScream},S_PLAY_XDIE3,0,0}, // S_PLAY_XDIE2
{SPR_PLAY,16,5,{A_Fall},S_PLAY_XDIE4,0,0}, // S_PLAY_XDIE3
{SPR_PLAY,17,5,{NULL},S_PLAY_XDIE5,0,0}, // S_PLAY_XDIE4
{SPR_PLAY,18,5,{NULL},S_PLAY_XDIE6,0,0}, // S_PLAY_XDIE5
{SPR_PLAY,19,5,{NULL},S_PLAY_XDIE7,0,0}, // S_PLAY_XDIE6
{SPR_PLAY,20,5,{NULL},S_PLAY_XDIE8,0,0}, // S_PLAY_XDIE7
{SPR_PLAY,21,5,{NULL},S_PLAY_XDIE9,0,0}, // S_PLAY_XDIE8
{SPR_PLAY,22,-1,{NULL},S_NULL,0,0}, // S_PLAY_XDIE9
{SPR_POSS,0,10,{A_Look},S_POSS_STND2,0,0}, // S_POSS_STND
{SPR_POSS,1,10,{A_Look},S_POSS_STND,0,0}, // S_POSS_STND2
{SPR_POSS,0,4,{A_Chase},S_POSS_RUN2,0,0}, // S_POSS_RUN1
{SPR_POSS,0,4,{A_Chase},S_POSS_RUN3,0,0}, // S_POSS_RUN2
{SPR_POSS,1,4,{A_Chase},S_POSS_RUN4,0,0}, // S_POSS_RUN3
{SPR_POSS,1,4,{A_Chase},S_POSS_RUN5,0,0}, // S_POSS_RUN4
{SPR_POSS,2,4,{A_Chase},S_POSS_RUN6,0,0}, // S_POSS_RUN5
{SPR_POSS,2,4,{A_Chase},S_POSS_RUN7,0,0}, // S_POSS_RUN6
{SPR_POSS,3,4,{A_Chase},S_POSS_RUN8,0,0}, // S_POSS_RUN7
{SPR_POSS,3,4,{A_Chase},S_POSS_RUN1,0,0}, // S_POSS_RUN8
{SPR_POSS,4,10,{A_FaceTarget},S_POSS_ATK2,0,0}, // S_POSS_ATK1
{SPR_POSS,5,8,{A_PosAttack},S_POSS_ATK3,0,0}, // S_POSS_ATK2
{SPR_POSS,4,8,{NULL},S_POSS_RUN1,0,0}, // S_POSS_ATK3
{SPR_POSS,6,3,{NULL},S_POSS_PAIN2,0,0}, // S_POSS_PAIN
{SPR_POSS,6,3,{A_Pain},S_POSS_RUN1,0,0}, // S_POSS_PAIN2
{SPR_POSS,7,5,{NULL},S_POSS_DIE2,0,0}, // S_POSS_DIE1
{SPR_POSS,8,5,{A_Scream},S_POSS_DIE3,0,0}, // S_POSS_DIE2
{SPR_POSS,9,5,{A_Fall},S_POSS_DIE4,0,0}, // S_POSS_DIE3
{SPR_POSS,10,5,{NULL},S_POSS_DIE5,0,0}, // S_POSS_DIE4
{SPR_POSS,11,-1,{NULL},S_NULL,0,0}, // S_POSS_DIE5

```

```

{SPR_POSS,12,5,{NULL},S_POSS_XDIE2,0,0},          // S_POSS_XDIE1
{SPR_POSS,13,5,{A_XScream},S_POSS_XDIE3,0,0},      // S_POSS_XDIE2
{SPR_POSS,14,5,{A_Fall},S_POSS_XDIE4,0,0},          // S_POSS_XDIE3
{SPR_POSS,15,5,{NULL},S_POSS_XDIE5,0,0},           // S_POSS_XDIE4
{SPR_POSS,16,5,{NULL},S_POSS_XDIE6,0,0},           // S_POSS_XDIE5
{SPR_POSS,17,5,{NULL},S_POSS_XDIE7,0,0},           // S_POSS_XDIE6
{SPR_POSS,18,5,{NULL},S_POSS_XDIE8,0,0},           // S_POSS_XDIE7
{SPR_POSS,19,5,{NULL},S_POSS_XDIE9,0,0},           // S_POSS_XDIE8
{SPR_POSS,20,-1,{NULL},S_NULL,0,0},                // S_POSS_XDIE9
{SPR_POSS,10,5,{NULL},S_POSS_RAISE2,0,0},           // S_POSS_RAISE1
{SPR_POSS,9,5,{NULL},S_POSS_RAISE3,0,0},            // S_POSS_RAISE2
{SPR_POSS,8,5,{NULL},S_POSS_RAISE4,0,0},            // S_POSS_RAISE3
{SPR_POSS,7,5,{NULL},S_POSS_RUN1,0,0},              // S_POSS_RAISE4
{SPR_SPOS,0,10,{A_Look},S_SPOS_STND2,0,0},          // S_SPOS_STND
{SPR_SPOS,1,10,{A_Look},S_SPOS_STND,0,0},           // S_SPOS_STND2
{SPR_SPOS,0,3,{A_Chase},S_SPOS_RUN2,0,0},           // S_SPOS_RUN1
{SPR_SPOS,0,3,{A_Chase},S_SPOS_RUN3,0,0},           // S_SPOS_RUN2
{SPR_SPOS,1,3,{A_Chase},S_SPOS_RUN4,0,0},           // S_SPOS_RUN3
{SPR_SPOS,1,3,{A_Chase},S_SPOS_RUN5,0,0},           // S_SPOS_RUN4
{SPR_SPOS,2,3,{A_Chase},S_SPOS_RUN6,0,0},           // S_SPOS_RUN5
{SPR_SPOS,2,3,{A_Chase},S_SPOS_RUN7,0,0},           // S_SPOS_RUN6
{SPR_SPOS,3,3,{A_Chase},S_SPOS_RUN8,0,0},           // S_SPOS_RUN7
{SPR_SPOS,3,3,{A_Chase},S_SPOS_RUN1,0,0},           // S_SPOS_RUN8
{SPR_SPOS,4,10,{A_FaceTarget},S_SPOS_ATK2,0,0},      // S_SPOS_ATK1
{SPR_SPOS,32773,10,{A_SPosAttack},S_SPOS_ATK3,0,0}, // S_SPOS_ATK2
{SPR_SPOS,4,10,{NULL},S_SPOS_RUN1,0,0},             // S_SPOS_ATK3
{SPR_SPOS,6,3,{NULL},S_SPOS_PAIN2,0,0},             // S_SPOS_PAIN
{SPR_SPOS,6,3,{A_Pain},S_SPOS_RUN1,0,0},            // S_SPOS_PAIN2
{SPR_SPOS,7,5,{NULL},S_SPOS_DIE2,0,0},             // S_SPOS_DIE1
{SPR_SPOS,8,5,{A_Scream},S_SPOS_DIE3,0,0},          // S_SPOS_DIE2
{SPR_SPOS,9,5,{A_Fall},S_SPOS_DIE4,0,0},            // S_SPOS_DIE3
{SPR_SPOS,10,5,{NULL},S_SPOS_DIE5,0,0},            // S_SPOS_DIE4
{SPR_SPOS,11,-1,{NULL},S_NULL,0,0},                // S_SPOS_DIE5
{SPR_SPOS,12,5,{NULL},S_SPOS_XDIE2,0,0},           // S_SPOS_XDIE1
{SPR_SPOS,13,5,{A_XScream},S_SPOS_XDIE3,0,0},      // S_SPOS_XDIE2
{SPR_SPOS,14,5,{A_Fall},S_SPOS_XDIE4,0,0},          // S_SPOS_XDIE3
{SPR_SPOS,15,5,{NULL},S_SPOS_XDIE5,0,0},           // S_SPOS_XDIE4
{SPR_SPOS,16,5,{NULL},S_SPOS_XDIE6,0,0},           // S_SPOS_XDIE5
{SPR_SPOS,17,5,{NULL},S_SPOS_XDIE7,0,0},           // S_SPOS_XDIE6
{SPR_SPOS,18,5,{NULL},S_SPOS_XDIE8,0,0},           // S_SPOS_XDIE7
{SPR_SPOS,19,5,{NULL},S_SPOS_XDIE9,0,0},           // S_SPOS_XDIE8
{SPR_SPOS,20,-1,{NULL},S_NULL,0,0},                // S_SPOS_XDIE9
{SPR_SPOS,11,5,{NULL},S_SPOS_RAISE2,0,0},           // S_SPOS_RAISE1
{SPR_SPOS,10,5,{NULL},S_SPOS_RAISE3,0,0},           // S_SPOS_RAISE2
{SPR_SPOS,9,5,{NULL},S_SPOS_RAISE4,0,0},            // S_SPOS_RAISE3
{SPR_SPOS,8,5,{NULL},S_SPOS_RAISE5,0,0},            // S_SPOS_RAISE4
{SPR_SPOS,7,5,{NULL},S_SPOS_RUN1,0,0},              // S_SPOS_RAISE5
{SPR_VILE,0,10,{A_Look},S_VILE_STND2,0,0},          // S_VILE_STND
{SPR_VILE,1,10,{A_Look},S_VILE_STND,0,0},           // S_VILE_STND2
{SPR_VILE,0,2,{A_VileChase},S_VILE_RUN2,0,0},       // S_VILE_RUN1
{SPR_VILE,0,2,{A_VileChase},S_VILE_RUN3,0,0},       // S_VILE_RUN2
{SPR_VILE,1,2,{A_VileChase},S_VILE_RUN4,0,0},       // S_VILE_RUN3
{SPR_VILE,1,2,{A_VileChase},S_VILE_RUN5,0,0},       // S_VILE_RUN4
{SPR_VILE,2,2,{A_VileChase},S_VILE_RUN6,0,0},       // S_VILE_RUN5
{SPR_VILE,2,2,{A_VileChase},S_VILE_RUN7,0,0},       // S_VILE_RUN6
{SPR_VILE,3,2,{A_VileChase},S_VILE_RUN8,0,0},       // S_VILE_RUN7
{SPR_VILE,3,2,{A_VileChase},S_VILE_RUN9,0,0},       // S_VILE_RUN8
{SPR_VILE,4,2,{A_VileChase},S_VILE_RUN10,0,0},      // S_VILE_RUN9
{SPR_VILE,4,2,{A_VileChase},S_VILE_RUN11,0,0},      // S_VILE_RUN10
{SPR_VILE,5,2,{A_VileChase},S_VILE_RUN12,0,0},      // S_VILE_RUN11
{SPR_VILE,5,2,{A_VileChase},S_VILE_RUN1,0,0},       // S_VILE_RUN12
{SPR_VILE,32774,0,{A_VileStart},S_VILE_ATK2,0,0},   // S_VILE_ATK1
{SPR_VILE,32774,10,{A_FaceTarget},S_VILE_ATK3,0,0}, // S_VILE_ATK2
{SPR_VILE,32775,8,{A_VileTarget},S_VILE_ATK4,0,0},  // S_VILE_ATK3
{SPR_VILE,32776,8,{A_FaceTarget},S_VILE_ATK5,0,0},  // S_VILE_ATK4

```

```

{SPR_VILE,32777,8,{A_FaceTarget},S_VILE_ATK6,0,0},          // S_VILE_ATK5
{SPR_VILE,32778,8,{A_FaceTarget},S_VILE_ATK7,0,0},          // S_VILE_ATK6
{SPR_VILE,32779,8,{A_FaceTarget},S_VILE_ATK8,0,0},          // S_VILE_ATK7
{SPR_VILE,32780,8,{A_FaceTarget},S_VILE_ATK9,0,0},          // S_VILE_ATK8
{SPR_VILE,32781,8,{A_FaceTarget},S_VILE_ATK10,0,0},          // S_VILE_ATK9
{SPR_VILE,32782,8,{A_VileAttack},S_VILE_ATK11,0,0},          // S_VILE_ATK10
{SPR_VILE,32783,20,{NULL},S_VILE_RUN1,0,0},                  // S_VILE_ATK11
{SPR_VILE,32794,10,{NULL},S_VILE_HEAL2,0,0},                  // S_VILE_HEAL1
{SPR_VILE,32795,10,{NULL},S_VILE_HEAL3,0,0},                  // S_VILE_HEAL2
{SPR_VILE,32796,10,{NULL},S_VILE_RUN1,0,0},                  // S_VILE_HEAL3
{SPR_VILE,16,5,{NULL},S_VILE_PAIN2,0,0},                      // S_VILE_PAIN
{SPR_VILE,16,5,{A_Pain},S_VILE_RUN1,0,0},                      // S_VILE_PAIN2
{SPR_VILE,16,7,{NULL},S_VILE_DIE2,0,0},                        // S_VILE_DIE1
{SPR_VILE,17,7,{A_Scream},S_VILE_DIE3,0,0},                  // S_VILE_DIE2
{SPR_VILE,18,7,{A_Fall},S_VILE_DIE4,0,0},                      // S_VILE_DIE3
{SPR_VILE,19,7,{NULL},S_VILE_DIE5,0,0},                        // S_VILE_DIE4
{SPR_VILE,20,7,{NULL},S_VILE_DIE6,0,0},                        // S_VILE_DIE5
{SPR_VILE,21,7,{NULL},S_VILE_DIE7,0,0},                        // S_VILE_DIE6
{SPR_VILE,22,7,{NULL},S_VILE_DIE8,0,0},                        // S_VILE_DIE7
{SPR_VILE,23,5,{NULL},S_VILE_DIE9,0,0},                        // S_VILE_DIE8
{SPR_VILE,24,5,{NULL},S_VILE_DIE10,0,0},                      // S_VILE_DIE9
{SPR_VILE,25,-1,{NULL},S_NULL,0,0},                            // S_VILE_DIE10
{SPR_FIRE,32768,2,{A_StartFire},S_FIRE2,0,0},                  // S_FIRE1
{SPR_FIRE,32769,2,{A_Fire},S_FIRE3,0,0},                      // S_FIRE2
{SPR_FIRE,32768,2,{A_Fire},S_FIRE4,0,0},                      // S_FIRE3
{SPR_FIRE,32769,2,{A_Fire},S_FIRE5,0,0},                      // S_FIRE4
{SPR_FIRE,32770,2,{A_FireCrackle},S_FIRE6,0,0},              // S_FIRE5
{SPR_FIRE,32769,2,{A_Fire},S_FIRE7,0,0},                      // S_FIRE6
{SPR_FIRE,32770,2,{A_Fire},S_FIRE8,0,0},                      // S_FIRE7
{SPR_FIRE,32769,2,{A_Fire},S_FIRE9,0,0},                      // S_FIRE8
{SPR_FIRE,32770,2,{A_Fire},S_FIRE10,0,0},                     // S_FIRE9
{SPR_FIRE,32771,2,{A_Fire},S_FIRE11,0,0},                     // S_FIRE10
{SPR_FIRE,32770,2,{A_Fire},S_FIRE12,0,0},                     // S_FIRE11
{SPR_FIRE,32771,2,{A_Fire},S_FIRE13,0,0},                     // S_FIRE12
{SPR_FIRE,32770,2,{A_Fire},S_FIRE14,0,0},                     // S_FIRE13
{SPR_FIRE,32771,2,{A_Fire},S_FIRE15,0,0},                     // S_FIRE14
{SPR_FIRE,32772,2,{A_Fire},S_FIRE16,0,0},                     // S_FIRE15
{SPR_FIRE,32771,2,{A_Fire},S_FIRE17,0,0},                     // S_FIRE16
{SPR_FIRE,32772,2,{A_Fire},S_FIRE18,0,0},                     // S_FIRE17
{SPR_FIRE,32771,2,{A_Fire},S_FIRE19,0,0},                     // S_FIRE18
{SPR_FIRE,32772,2,{A_FireCrackle},S_FIRE20,0,0},              // S_FIRE19
{SPR_FIRE,32773,2,{A_Fire},S_FIRE21,0,0},                      // S_FIRE20
{SPR_FIRE,32772,2,{A_Fire},S_FIRE22,0,0},                      // S_FIRE21
{SPR_FIRE,32773,2,{A_Fire},S_FIRE23,0,0},                      // S_FIRE22
{SPR_FIRE,32772,2,{A_Fire},S_FIRE24,0,0},                      // S_FIRE23
{SPR_FIRE,32773,2,{A_Fire},S_FIRE25,0,0},                      // S_FIRE24
{SPR_FIRE,32774,2,{A_Fire},S_FIRE26,0,0},                      // S_FIRE25
{SPR_FIRE,32775,2,{A_Fire},S_FIRE27,0,0},                      // S_FIRE26
{SPR_FIRE,32774,2,{A_Fire},S_FIRE28,0,0},                      // S_FIRE27
{SPR_FIRE,32775,2,{A_Fire},S_FIRE29,0,0},                      // S_FIRE28
{SPR_FIRE,32774,2,{A_Fire},S_FIRE30,0,0},                      // S_FIRE29
{SPR_FIRE,32775,2,{A_Fire},S_NULL,0,0},                        // S_FIRE30
{SPR_PUFF,1,4,{NULL},S_SMOKE2,0,0},                            // S_SMOKE1
{SPR_PUFF,2,4,{NULL},S_SMOKE3,0,0},                            // S_SMOKE2
{SPR_PUFF,1,4,{NULL},S_SMOKE4,0,0},                            // S_SMOKE3
{SPR_PUFF,2,4,{NULL},S_SMOKE5,0,0},                            // S_SMOKE4
{SPR_PUFF,3,4,{NULL},S_NULL,0,0},                              // S_SMOKE5
{SPR_FATB,32768,2,{A_Tracer},S_TRACER2,0,0},                  // S_TRACER
{SPR_FATB,32769,2,{A_Tracer},S_TRACER,0,0},                  // S_TRACER2
{SPR_FBP,32768,8,{NULL},S_TRACEEXP2,0,0},                     // S_TRACEEXP1
{SPR_FBP,32769,6,{NULL},S_TRACEEXP3,0,0},                     // S_TRACEEXP2
{SPR_FBP,32770,4,{NULL},S_NULL,0,0},                          // S_TRACEEXP3
{SPR_SKEL,0,10,{A_Look},S_SKEL_STND2,0,0},                    // S_SKEL_STND
{SPR_SKEL,1,10,{A_Look},S_SKEL_STND,0,0},                     // S_SKEL_STND2
{SPR_SKEL,0,2,{A_Chase},S_SKEL_RUN2,0,0},                     // S_SKEL_RUN1

```

```

{SPR_SKEL,0,2,{A_Chase},S_SKEL_RUN3,0,0}, // S_SKEL_RUN2
{SPR_SKEL,1,2,{A_Chase},S_SKEL_RUN4,0,0}, // S_SKEL_RUN3
{SPR_SKEL,1,2,{A_Chase},S_SKEL_RUN5,0,0}, // S_SKEL_RUN4
{SPR_SKEL,2,2,{A_Chase},S_SKEL_RUN6,0,0}, // S_SKEL_RUN5
{SPR_SKEL,2,2,{A_Chase},S_SKEL_RUN7,0,0}, // S_SKEL_RUN6
{SPR_SKEL,3,2,{A_Chase},S_SKEL_RUN8,0,0}, // S_SKEL_RUN7
{SPR_SKEL,3,2,{A_Chase},S_SKEL_RUN9,0,0}, // S_SKEL_RUN8
{SPR_SKEL,4,2,{A_Chase},S_SKEL_RUN10,0,0}, // S_SKEL_RUN9
{SPR_SKEL,4,2,{A_Chase},S_SKEL_RUN11,0,0}, // S_SKEL_RUN10
{SPR_SKEL,5,2,{A_Chase},S_SKEL_RUN12,0,0}, // S_SKEL_RUN11
{SPR_SKEL,5,2,{A_Chase},S_SKEL_RUN1,0,0}, // S_SKEL_RUN12
{SPR_SKEL,6,0,{A_FaceTarget},S_SKEL_FIST2,0,0}, // S_SKEL_FIST1
{SPR_SKEL,6,6,{A_SkelWhoosh},S_SKEL_FIST3,0,0}, // S_SKEL_FIST2
{SPR_SKEL,7,6,{A_FaceTarget},S_SKEL_FIST4,0,0}, // S_SKEL_FIST3
{SPR_SKEL,8,6,{A_SkelFist},S_SKEL_RUN1,0,0}, // S_SKEL_FIST4
{SPR_SKEL,32777,0,{A_FaceTarget},S_SKEL_MISS2,0,0}, // S_SKEL_MISS1
{SPR_SKEL,32777,10,{A_FaceTarget},S_SKEL_MISS3,0,0}, // S_SKEL_MISS2
{SPR_SKEL,10,10,{A_SkelMissile},S_SKEL_MISS4,0,0}, // S_SKEL_MISS3
{SPR_SKEL,10,10,{A_FaceTarget},S_SKEL_RUN1,0,0}, // S_SKEL_MISS4
{SPR_SKEL,11,5,{NULL},S_SKEL_PAIN2,0,0}, // S_SKEL_PAIN
{SPR_SKEL,11,5,{A_Pain},S_SKEL_RUN1,0,0}, // S_SKEL_PAIN2
{SPR_SKEL,11,7,{NULL},S_SKEL_DIE2,0,0}, // S_SKEL_DIE1
{SPR_SKEL,12,7,{NULL},S_SKEL_DIE3,0,0}, // S_SKEL_DIE2
{SPR_SKEL,13,7,{A_Scream},S_SKEL_DIE4,0,0}, // S_SKEL_DIE3
{SPR_SKEL,14,7,{A_Fall},S_SKEL_DIE5,0,0}, // S_SKEL_DIE4
{SPR_SKEL,15,7,{NULL},S_SKEL_DIE6,0,0}, // S_SKEL_DIE5
{SPR_SKEL,16,-1,{NULL},S_NULL,0,0}, // S_SKEL_DIE6
{SPR_SKEL,16,5,{NULL},S_SKEL_RAISE2,0,0}, // S_SKEL_RAISE1
{SPR_SKEL,15,5,{NULL},S_SKEL_RAISE3,0,0}, // S_SKEL_RAISE2
{SPR_SKEL,14,5,{NULL},S_SKEL_RAISE4,0,0}, // S_SKEL_RAISE3
{SPR_SKEL,13,5,{NULL},S_SKEL_RAISE5,0,0}, // S_SKEL_RAISE4
{SPR_SKEL,12,5,{NULL},S_SKEL_RAISE6,0,0}, // S_SKEL_RAISE5
{SPR_SKEL,11,5,{NULL},S_SKEL_RUN1,0,0}, // S_SKEL_RAISE6
{SPR_MANF,32768,4,{NULL},S_FATSHOT2,0,0}, // S_FATSHOT1
{SPR_MANF,32769,4,{NULL},S_FATSHOT1,0,0}, // S_FATSHOT2
{SPR_MISL,32769,8,{NULL},S_FATSHOTX2,0,0}, // S_FATSHOTX1
{SPR_MISL,32770,6,{NULL},S_FATSHOTX3,0,0}, // S_FATSHOTX2
{SPR_MISL,32771,4,{NULL},S_NULL,0,0}, // S_FATSHOTX3
{SPR_FATT,0,15,{A_Look},S_FATT_STND2,0,0}, // S_FATT_STND
{SPR_FATT,1,15,{A_Look},S_FATT_STND,0,0}, // S_FATT_STND2
{SPR_FATT,0,4,{A_Chase},S_FATT_RUN2,0,0}, // S_FATT_RUN1
{SPR_FATT,0,4,{A_Chase},S_FATT_RUN3,0,0}, // S_FATT_RUN2
{SPR_FATT,1,4,{A_Chase},S_FATT_RUN4,0,0}, // S_FATT_RUN3
{SPR_FATT,1,4,{A_Chase},S_FATT_RUN5,0,0}, // S_FATT_RUN4
{SPR_FATT,2,4,{A_Chase},S_FATT_RUN6,0,0}, // S_FATT_RUN5
{SPR_FATT,2,4,{A_Chase},S_FATT_RUN7,0,0}, // S_FATT_RUN6
{SPR_FATT,3,4,{A_Chase},S_FATT_RUN8,0,0}, // S_FATT_RUN7
{SPR_FATT,3,4,{A_Chase},S_FATT_RUN9,0,0}, // S_FATT_RUN8
{SPR_FATT,4,4,{A_Chase},S_FATT_RUN10,0,0}, // S_FATT_RUN9
{SPR_FATT,4,4,{A_Chase},S_FATT_RUN11,0,0}, // S_FATT_RUN10
{SPR_FATT,5,4,{A_Chase},S_FATT_RUN12,0,0}, // S_FATT_RUN11
{SPR_FATT,5,4,{A_Chase},S_FATT_RUN1,0,0}, // S_FATT_RUN12
{SPR_FATT,6,20,{A_FatRaise},S_FATT_ATK2,0,0}, // S_FATT_ATK1
{SPR_FATT,32775,10,{A_FatAttack1},S_FATT_ATK3,0,0}, // S_FATT_ATK2
{SPR_FATT,8,5,{A_FaceTarget},S_FATT_ATK4,0,0}, // S_FATT_ATK3
{SPR_FATT,6,5,{A_FaceTarget},S_FATT_ATK5,0,0}, // S_FATT_ATK4
{SPR_FATT,32775,10,{A_FatAttack2},S_FATT_ATK6,0,0}, // S_FATT_ATK5
{SPR_FATT,8,5,{A_FaceTarget},S_FATT_ATK7,0,0}, // S_FATT_ATK6
{SPR_FATT,6,5,{A_FaceTarget},S_FATT_ATK8,0,0}, // S_FATT_ATK7
{SPR_FATT,32775,10,{A_FatAttack3},S_FATT_ATK9,0,0}, // S_FATT_ATK8
{SPR_FATT,8,5,{A_FaceTarget},S_FATT_ATK10,0,0}, // S_FATT_ATK9
{SPR_FATT,6,5,{A_FaceTarget},S_FATT_RUN1,0,0}, // S_FATT_ATK10
{SPR_FATT,9,3,{NULL},S_FATT_PAIN2,0,0}, // S_FATT_PAIN
{SPR_FATT,9,3,{A_Pain},S_FATT_RUN1,0,0}, // S_FATT_PAIN2
{SPR_FATT,10,6,{NULL},S_FATT_DIE2,0,0}, // S_FATT_DIE1

```

```

{SPR_FATT,11,6,{A_Scream},S_FATT_DIE3,0,0}, // S_FATT_DIE2
{SPR_FATT,12,6,{A_Fall},S_FATT_DIE4,0,0}, // S_FATT_DIE3
{SPR_FATT,13,6,{NULL},S_FATT_DIE5,0,0}, // S_FATT_DIE4
{SPR_FATT,14,6,{NULL},S_FATT_DIE6,0,0}, // S_FATT_DIE5
{SPR_FATT,15,6,{NULL},S_FATT_DIE7,0,0}, // S_FATT_DIE6
{SPR_FATT,16,6,{NULL},S_FATT_DIE8,0,0}, // S_FATT_DIE7
{SPR_FATT,17,6,{NULL},S_FATT_DIE9,0,0}, // S_FATT_DIE8
{SPR_FATT,18,6,{NULL},S_FATT_DIE10,0,0}, // S_FATT_DIE9
{SPR_FATT,19,-1,{A_BossDeath},S_NULL,0,0}, // S_FATT_DIE10
{SPR_FATT,17,5,{NULL},S_FATT_RAISE2,0,0}, // S_FATT_RAISE1
{SPR_FATT,16,5,{NULL},S_FATT_RAISE3,0,0}, // S_FATT_RAISE2
{SPR_FATT,15,5,{NULL},S_FATT_RAISE4,0,0}, // S_FATT_RAISE3
{SPR_FATT,14,5,{NULL},S_FATT_RAISE5,0,0}, // S_FATT_RAISE4
{SPR_FATT,13,5,{NULL},S_FATT_RAISE6,0,0}, // S_FATT_RAISE5
{SPR_FATT,12,5,{NULL},S_FATT_RAISE7,0,0}, // S_FATT_RAISE6
{SPR_FATT,11,5,{NULL},S_FATT_RAISE8,0,0}, // S_FATT_RAISE7
{SPR_FATT,10,5,{NULL},S_FATT_RUN1,0,0}, // S_FATT_RAISE8
{SPR_CPOS,0,10,{A_Look},S_CPOS_STND2,0,0}, // S_CPOS_STND
{SPR_CPOS,1,10,{A_Look},S_CPOS_STND,0,0}, // S_CPOS_STND2
{SPR_CPOS,0,3,{A_Chase},S_CPOS_RUN2,0,0}, // S_CPOS_RUN1
{SPR_CPOS,0,3,{A_Chase},S_CPOS_RUN3,0,0}, // S_CPOS_RUN2
{SPR_CPOS,1,3,{A_Chase},S_CPOS_RUN4,0,0}, // S_CPOS_RUN3
{SPR_CPOS,1,3,{A_Chase},S_CPOS_RUN5,0,0}, // S_CPOS_RUN4
{SPR_CPOS,2,3,{A_Chase},S_CPOS_RUN6,0,0}, // S_CPOS_RUN5
{SPR_CPOS,2,3,{A_Chase},S_CPOS_RUN7,0,0}, // S_CPOS_RUN6
{SPR_CPOS,3,3,{A_Chase},S_CPOS_RUN8,0,0}, // S_CPOS_RUN7
{SPR_CPOS,3,3,{A_Chase},S_CPOS_RUN1,0,0}, // S_CPOS_RUN8
{SPR_CPOS,4,10,{A_FaceTarget},S_CPOS_ATK2,0,0}, // S_CPOS_ATK1
{SPR_CPOS,32773,4,{A_CPosAttack},S_CPOS_ATK3,0,0}, // S_CPOS_ATK2
{SPR_CPOS,32772,4,{A_CPosAttack},S_CPOS_ATK4,0,0}, // S_CPOS_ATK3
{SPR_CPOS,5,1,{A_CPosRefire},S_CPOS_ATK2,0,0}, // S_CPOS_ATK4
{SPR_CPOS,6,3,{NULL},S_CPOS_PAIN2,0,0}, // S_CPOS_PAIN
{SPR_CPOS,6,3,{A_Pain},S_CPOS_RUN1,0,0}, // S_CPOS_PAIN2
{SPR_CPOS,7,5,{NULL},S_CPOS_DIE2,0,0}, // S_CPOS_DIE1
{SPR_CPOS,8,5,{A_Scream},S_CPOS_DIE3,0,0}, // S_CPOS_DIE2
{SPR_CPOS,9,5,{A_Fall},S_CPOS_DIE4,0,0}, // S_CPOS_DIE3
{SPR_CPOS,10,5,{NULL},S_CPOS_DIE5,0,0}, // S_CPOS_DIE4
{SPR_CPOS,11,5,{NULL},S_CPOS_DIE6,0,0}, // S_CPOS_DIE5
{SPR_CPOS,12,5,{NULL},S_CPOS_DIE7,0,0}, // S_CPOS_DIE6
{SPR_CPOS,13,-1,{NULL},S_NULL,0,0}, // S_CPOS_DIE7
{SPR_CPOS,14,5,{NULL},S_CPOS_XDIE2,0,0}, // S_CPOS_XDIE1
{SPR_CPOS,15,5,{A_XScream},S_CPOS_XDIE3,0,0}, // S_CPOS_XDIE2
{SPR_CPOS,16,5,{A_Fall},S_CPOS_XDIE4,0,0}, // S_CPOS_XDIE3
{SPR_CPOS,17,5,{NULL},S_CPOS_XDIE5,0,0}, // S_CPOS_XDIE4
{SPR_CPOS,18,5,{NULL},S_CPOS_XDIE6,0,0}, // S_CPOS_XDIE5
{SPR_CPOS,19,-1,{NULL},S_NULL,0,0}, // S_CPOS_XDIE6
{SPR_CPOS,13,5,{NULL},S_CPOS_RAISE2,0,0}, // S_CPOS_RAISE1
{SPR_CPOS,12,5,{NULL},S_CPOS_RAISE3,0,0}, // S_CPOS_RAISE2
{SPR_CPOS,11,5,{NULL},S_CPOS_RAISE4,0,0}, // S_CPOS_RAISE3
{SPR_CPOS,10,5,{NULL},S_CPOS_RAISE5,0,0}, // S_CPOS_RAISE4
{SPR_CPOS,9,5,{NULL},S_CPOS_RAISE6,0,0}, // S_CPOS_RAISE5
{SPR_CPOS,8,5,{NULL},S_CPOS_RAISE7,0,0}, // S_CPOS_RAISE6
{SPR_CPOS,7,5,{NULL},S_CPOS_RUN1,0,0}, // S_CPOS_RAISE7
{SPR_TROO,0,10,{A_Look},S_TROO_STND2,0,0}, // S_TROO_STND
{SPR_TROO,1,10,{A_Look},S_TROO_STND,0,0}, // S_TROO_STND2
{SPR_TROO,0,3,{A_Chase},S_TROO_RUN2,0,0}, // S_TROO_RUN1
{SPR_TROO,0,3,{A_Chase},S_TROO_RUN3,0,0}, // S_TROO_RUN2
{SPR_TROO,1,3,{A_Chase},S_TROO_RUN4,0,0}, // S_TROO_RUN3
{SPR_TROO,1,3,{A_Chase},S_TROO_RUN5,0,0}, // S_TROO_RUN4
{SPR_TROO,2,3,{A_Chase},S_TROO_RUN6,0,0}, // S_TROO_RUN5
{SPR_TROO,2,3,{A_Chase},S_TROO_RUN7,0,0}, // S_TROO_RUN6
{SPR_TROO,3,3,{A_Chase},S_TROO_RUN8,0,0}, // S_TROO_RUN7
{SPR_TROO,3,3,{A_Chase},S_TROO_RUN1,0,0}, // S_TROO_RUN8
{SPR_TROO,4,8,{A_FaceTarget},S_TROO_ATK2,0,0}, // S_TROO_ATK1
{SPR_TROO,5,8,{A_FaceTarget},S_TROO_ATK3,0,0}, // S_TROO_ATK2

```

```

{SPR_TROO,6,6,{A_TroopAttack},S_TROO_RUN1,0,0},          // S_TROO_ATK3
{SPR_TROO,7,2,{NULL},S_TROO_PAIN2,0,0},                  // S_TROO_PAIN
{SPR_TROO,7,2,{A_Pain},S_TROO_RUN1,0,0},                  // S_TROO_PAIN2
{SPR_TROO,8,8,{NULL},S_TROO_DIE2,0,0},                    // S_TROO_DIE1
{SPR_TROO,9,8,{A_Scream},S_TROO_DIE3,0,0},                // S_TROO_DIE2
{SPR_TROO,10,6,{NULL},S_TROO_DIE4,0,0},                   // S_TROO_DIE3
{SPR_TROO,11,6,{A_Fall},S_TROO_DIE5,0,0},                 // S_TROO_DIE4
{SPR_TROO,12,-1,{NULL},S_NULL,0,0},                        // S_TROO_DIE5
{SPR_TROO,13,5,{NULL},S_TROO_XDIE2,0,0},                  // S_TROO_XDIE1
{SPR_TROO,14,5,{A_XScream},S_TROO_XDIE3,0,0},             // S_TROO_XDIE2
{SPR_TROO,15,5,{NULL},S_TROO_XDIE4,0,0},                  // S_TROO_XDIE3
{SPR_TROO,16,5,{A_Fall},S_TROO_XDIE5,0,0},                // S_TROO_XDIE4
{SPR_TROO,17,5,{NULL},S_TROO_XDIE6,0,0},                  // S_TROO_XDIE5
{SPR_TROO,18,5,{NULL},S_TROO_XDIE7,0,0},                  // S_TROO_XDIE6
{SPR_TROO,19,5,{NULL},S_TROO_XDIE8,0,0},                  // S_TROO_XDIE7
{SPR_TROO,20,-1,{NULL},S_NULL,0,0},                        // S_TROO_XDIE8
{SPR_TROO,12,8,{NULL},S_TROO_RAISE2,0,0},                  // S_TROO_RAISE1
{SPR_TROO,11,8,{NULL},S_TROO_RAISE3,0,0},                  // S_TROO_RAISE2
{SPR_TROO,10,6,{NULL},S_TROO_RAISE4,0,0},                  // S_TROO_RAISE3
{SPR_TROO,9,6,{NULL},S_TROO_RAISE5,0,0},                  // S_TROO_RAISE4
{SPR_TROO,8,6,{NULL},S_TROO_RUN1,0,0},                     // S_TROO_RAISE5
{SPR_SARG,0,10,{A_Look},S_SARG_STND2,0,0},                 // S_SARG_STND
{SPR_SARG,1,10,{A_Look},S_SARG_STND,0,0},                  // S_SARG_STND2
{SPR_SARG,0,2,{A_Chase},S_SARG_RUN2,0,0},                  // S_SARG_RUN1
{SPR_SARG,0,2,{A_Chase},S_SARG_RUN3,0,0},                  // S_SARG_RUN2
{SPR_SARG,1,2,{A_Chase},S_SARG_RUN4,0,0},                  // S_SARG_RUN3
{SPR_SARG,1,2,{A_Chase},S_SARG_RUN5,0,0},                  // S_SARG_RUN4
{SPR_SARG,2,2,{A_Chase},S_SARG_RUN6,0,0},                  // S_SARG_RUN5
{SPR_SARG,2,2,{A_Chase},S_SARG_RUN7,0,0},                  // S_SARG_RUN6
{SPR_SARG,3,2,{A_Chase},S_SARG_RUN8,0,0},                  // S_SARG_RUN7
{SPR_SARG,3,2,{A_Chase},S_SARG_RUN1,0,0},                  // S_SARG_RUN8
{SPR_SARG,4,8,{A_FaceTarget},S_SARG_ATK2,0,0},             // S_SARG_ATK1
{SPR_SARG,5,8,{A_FaceTarget},S_SARG_ATK3,0,0},             // S_SARG_ATK2
{SPR_SARG,6,8,{A_SargAttack},S_SARG_RUN1,0,0},             // S_SARG_ATK3
{SPR_SARG,7,2,{NULL},S_SARG_PAIN2,0,0},                     // S_SARG_PAIN
{SPR_SARG,7,2,{A_Pain},S_SARG_RUN1,0,0},                     // S_SARG_PAIN2
{SPR_SARG,8,8,{NULL},S_SARG_DIE2,0,0},                     // S_SARG_DIE1
{SPR_SARG,9,8,{A_Scream},S_SARG_DIE3,0,0},                 // S_SARG_DIE2
{SPR_SARG,10,4,{NULL},S_SARG_DIE4,0,0},                     // S_SARG_DIE3
{SPR_SARG,11,4,{A_Fall},S_SARG_DIE5,0,0},                  // S_SARG_DIE4
{SPR_SARG,12,4,{NULL},S_SARG_DIE6,0,0},                     // S_SARG_DIE5
{SPR_SARG,13,-1,{NULL},S_NULL,0,0},                         // S_SARG_DIE6
{SPR_SARG,13,5,{NULL},S_SARG_RAISE2,0,0},                  // S_SARG_RAISE1
{SPR_SARG,12,5,{NULL},S_SARG_RAISE3,0,0},                  // S_SARG_RAISE2
{SPR_SARG,11,5,{NULL},S_SARG_RAISE4,0,0},                  // S_SARG_RAISE3
{SPR_SARG,10,5,{NULL},S_SARG_RAISE5,0,0},                  // S_SARG_RAISE4
{SPR_SARG,9,5,{NULL},S_SARG_RAISE6,0,0},                   // S_SARG_RAISE5
{SPR_SARG,8,5,{NULL},S_SARG_RUN1,0,0},                      // S_SARG_RAISE6
{SPR_HEAD,0,10,{A_Look},S_HEAD_STND,0,0},                   // S_HEAD_STND
{SPR_HEAD,0,3,{A_Chase},S_HEAD_RUN1,0,0},                   // S_HEAD_RUN1
{SPR_HEAD,1,5,{A_FaceTarget},S_HEAD_ATK2,0,0},              // S_HEAD_ATK1
{SPR_HEAD,2,5,{A_FaceTarget},S_HEAD_ATK3,0,0},              // S_HEAD_ATK2
{SPR_HEAD,32771,5,{A_HeadAttack},S_HEAD_RUN1,0,0},          // S_HEAD_ATK3
{SPR_HEAD,4,3,{NULL},S_HEAD_PAIN2,0,0},                     // S_HEAD_PAIN
{SPR_HEAD,4,3,{A_Pain},S_HEAD_PAIN3,0,0},                   // S_HEAD_PAIN2
{SPR_HEAD,5,6,{NULL},S_HEAD_RUN1,0,0},                       // S_HEAD_PAIN3
{SPR_HEAD,6,8,{NULL},S_HEAD_DIE2,0,0},                       // S_HEAD_DIE1
{SPR_HEAD,7,8,{A_Scream},S_HEAD_DIE3,0,0},                  // S_HEAD_DIE2
{SPR_HEAD,8,8,{NULL},S_HEAD_DIE4,0,0},                       // S_HEAD_DIE3
{SPR_HEAD,9,8,{NULL},S_HEAD_DIE5,0,0},                       // S_HEAD_DIE4
{SPR_HEAD,10,8,{A_Fall},S_HEAD_DIE6,0,0},                   // S_HEAD_DIE5
{SPR_HEAD,11,-1,{NULL},S_NULL,0,0},                         // S_HEAD_DIE6
{SPR_HEAD,11,8,{NULL},S_HEAD_RAISE2,0,0},                   // S_HEAD_RAISE1
{SPR_HEAD,10,8,{NULL},S_HEAD_RAISE3,0,0},                   // S_HEAD_RAISE2
{SPR_HEAD,9,8,{NULL},S_HEAD_RAISE4,0,0},                     // S_HEAD_RAISE3

```



```

{SPR_HEAD,8,8,{NULL},S_HEAD_RAISE5,0,0}, // S_HEAD_RAISE4
{SPR_HEAD,7,8,{NULL},S_HEAD_RAISE6,0,0}, // S_HEAD_RAISE5
{SPR_HEAD,6,8,{NULL},S_HEAD_RUN1,0,0}, // S_HEAD_RAISE6
{SPR_BAL7,32768,4,{NULL},S_BRBALL2,0,0}, // S_BRBALL1
{SPR_BAL7,32769,4,{NULL},S_BRBALL1,0,0}, // S_BRBALL2
{SPR_BAL7,32770,6,{NULL},S_BRBALLX2,0,0}, // S_BRBALLX1
{SPR_BAL7,32771,6,{NULL},S_BRBALLX3,0,0}, // S_BRBALLX2
{SPR_BAL7,32772,6,{NULL},S_NULL,0,0}, // S_BRBALLX3
{SPR_BOSS,0,10,{A_Look},S_BOSS_STND2,0,0}, // S_BOSS_STND
{SPR_BOSS,1,10,{A_Look},S_BOSS_STND,0,0}, // S_BOSS_STND2
{SPR_BOSS,0,3,{A_Chase},S_BOSS_RUN2,0,0}, // S_BOSS_RUN1
{SPR_BOSS,0,3,{A_Chase},S_BOSS_RUN3,0,0}, // S_BOSS_RUN2
{SPR_BOSS,1,3,{A_Chase},S_BOSS_RUN4,0,0}, // S_BOSS_RUN3
{SPR_BOSS,1,3,{A_Chase},S_BOSS_RUN5,0,0}, // S_BOSS_RUN4
{SPR_BOSS,2,3,{A_Chase},S_BOSS_RUN6,0,0}, // S_BOSS_RUN5
{SPR_BOSS,2,3,{A_Chase},S_BOSS_RUN7,0,0}, // S_BOSS_RUN6
{SPR_BOSS,3,3,{A_Chase},S_BOSS_RUN8,0,0}, // S_BOSS_RUN7
{SPR_BOSS,3,3,{A_Chase},S_BOSS_RUN1,0,0}, // S_BOSS_RUN8
{SPR_BOSS,4,8,{A_FaceTarget},S_BOSS_ATK2,0,0}, // S_BOSS_ATK1
{SPR_BOSS,5,8,{A_FaceTarget},S_BOSS_ATK3,0,0}, // S_BOSS_ATK2
{SPR_BOSS,6,8,{A_BruisAttack},S_BOSS_RUN1,0,0}, // S_BOSS_ATK3
{SPR_BOSS,7,2,{NULL},S_BOSS_PAIN2,0,0}, // S_BOSS_PAIN
{SPR_BOSS,7,2,{A_Pain},S_BOSS_RUN1,0,0}, // S_BOSS_PAIN2
{SPR_BOSS,8,8,{NULL},S_BOSS_DIE2,0,0}, // S_BOSS_DIE1
{SPR_BOSS,9,8,{A_Scream},S_BOSS_DIE3,0,0}, // S_BOSS_DIE2
{SPR_BOSS,10,8,{NULL},S_BOSS_DIE4,0,0}, // S_BOSS_DIE3
{SPR_BOSS,11,8,{A_Fall},S_BOSS_DIE5,0,0}, // S_BOSS_DIE4
{SPR_BOSS,12,8,{NULL},S_BOSS_DIE6,0,0}, // S_BOSS_DIE5
{SPR_BOSS,13,8,{NULL},S_BOSS_DIE7,0,0}, // S_BOSS_DIE6
{SPR_BOSS,14,-1,{A_BossDeath},S_NULL,0,0}, // S_BOSS_DIE7
{SPR_BOSS,14,8,{NULL},S_BOSS_RAISE2,0,0}, // S_BOSS_RAISE1
{SPR_BOSS,13,8,{NULL},S_BOSS_RAISE3,0,0}, // S_BOSS_RAISE2
{SPR_BOSS,12,8,{NULL},S_BOSS_RAISE4,0,0}, // S_BOSS_RAISE3
{SPR_BOSS,11,8,{NULL},S_BOSS_RAISE5,0,0}, // S_BOSS_RAISE4
{SPR_BOSS,10,8,{NULL},S_BOSS_RAISE6,0,0}, // S_BOSS_RAISE5
{SPR_BOSS,9,8,{NULL},S_BOSS_RAISE7,0,0}, // S_BOSS_RAISE6
{SPR_BOSS,8,8,{NULL},S_BOSS_RUN1,0,0}, // S_BOSS_RAISE7
{SPR_BOS2,0,10,{A_Look},S_BOS2_STND2,0,0}, // S_BOS2_STND
{SPR_BOS2,1,10,{A_Look},S_BOS2_STND,0,0}, // S_BOS2_STND2
{SPR_BOS2,0,3,{A_Chase},S_BOS2_RUN2,0,0}, // S_BOS2_RUN1
{SPR_BOS2,0,3,{A_Chase},S_BOS2_RUN3,0,0}, // S_BOS2_RUN2
{SPR_BOS2,1,3,{A_Chase},S_BOS2_RUN4,0,0}, // S_BOS2_RUN3
{SPR_BOS2,1,3,{A_Chase},S_BOS2_RUN5,0,0}, // S_BOS2_RUN4
{SPR_BOS2,2,3,{A_Chase},S_BOS2_RUN6,0,0}, // S_BOS2_RUN5
{SPR_BOS2,2,3,{A_Chase},S_BOS2_RUN7,0,0}, // S_BOS2_RUN6
{SPR_BOS2,3,3,{A_Chase},S_BOS2_RUN8,0,0}, // S_BOS2_RUN7
{SPR_BOS2,3,3,{A_Chase},S_BOS2_RUN1,0,0}, // S_BOS2_RUN8
{SPR_BOS2,4,8,{A_FaceTarget},S_BOS2_ATK2,0,0}, // S_BOS2_ATK1
{SPR_BOS2,5,8,{A_FaceTarget},S_BOS2_ATK3,0,0}, // S_BOS2_ATK2
{SPR_BOS2,6,8,{A_BruisAttack},S_BOS2_RUN1,0,0}, // S_BOS2_ATK3
{SPR_BOS2,7,2,{NULL},S_BOS2_PAIN2,0,0}, // S_BOS2_PAIN
{SPR_BOS2,7,2,{A_Pain},S_BOS2_RUN1,0,0}, // S_BOS2_PAIN2
{SPR_BOS2,8,8,{NULL},S_BOS2_DIE2,0,0}, // S_BOS2_DIE1
{SPR_BOS2,9,8,{A_Scream},S_BOS2_DIE3,0,0}, // S_BOS2_DIE2
{SPR_BOS2,10,8,{NULL},S_BOS2_DIE4,0,0}, // S_BOS2_DIE3
{SPR_BOS2,11,8,{A_Fall},S_BOS2_DIE5,0,0}, // S_BOS2_DIE4
{SPR_BOS2,12,8,{NULL},S_BOS2_DIE6,0,0}, // S_BOS2_DIE5
{SPR_BOS2,13,8,{NULL},S_BOS2_DIE7,0,0}, // S_BOS2_DIE6
{SPR_BOS2,14,-1,{NULL},S_NULL,0,0}, // S_BOS2_DIE7
{SPR_BOS2,14,8,{NULL},S_BOS2_RAISE2,0,0}, // S_BOS2_RAISE1
{SPR_BOS2,13,8,{NULL},S_BOS2_RAISE3,0,0}, // S_BOS2_RAISE2
{SPR_BOS2,12,8,{NULL},S_BOS2_RAISE4,0,0}, // S_BOS2_RAISE3
{SPR_BOS2,11,8,{NULL},S_BOS2_RAISE5,0,0}, // S_BOS2_RAISE4
{SPR_BOS2,10,8,{NULL},S_BOS2_RAISE6,0,0}, // S_BOS2_RAISE5
{SPR_BOS2,9,8,{NULL},S_BOS2_RAISE7,0,0}, // S_BOS2_RAISE6

```

```

{SPR_BOS2,8,8,{NULL},S_BOS2_RUN1,0,0},          // S_BOS2_RAISE7
{SPR_SKUL,32768,10,{A_Look},S_SKULL_STND2,0,0},    // S_SKULL_STND
{SPR_SKUL,32769,10,{A_Look},S_SKULL_STND,0,0},    // S_SKULL_STND2
{SPR_SKUL,32768,6,{A_Chase},S_SKULL_RUN2,0,0},    // S_SKULL_RUN1
{SPR_SKUL,32769,6,{A_Chase},S_SKULL_RUN1,0,0},    // S_SKULL_RUN2
{SPR_SKUL,32770,10,{A_FaceTarget},S_SKULL_ATK2,0,0},    // S_SKULL_ATK1
{SPR_SKUL,32771,4,{A_SkullAttack},S_SKULL_ATK3,0,0},    // S_SKULL_ATK2
{SPR_SKUL,32770,4,{NULL},S_SKULL_ATK4,0,0},    // S_SKULL_ATK3
{SPR_SKUL,32771,4,{NULL},S_SKULL_ATK3,0,0},    // S_SKULL_ATK4
{SPR_SKUL,32772,3,{NULL},S_SKULL_PAIN2,0,0},    // S_SKULL_PAIN
{SPR_SKUL,32772,3,{A_Pain},S_SKULL_RUN1,0,0},    // S_SKULL_PAIN2
{SPR_SKUL,32773,6,{NULL},S_SKULL_DIE2,0,0},    // S_SKULL_DIE1
{SPR_SKUL,32774,6,{A_Scream},S_SKULL_DIE3,0,0},    // S_SKULL_DIE2
{SPR_SKUL,32775,6,{NULL},S_SKULL_DIE4,0,0},    // S_SKULL_DIE3
{SPR_SKUL,32776,6,{A_Fall},S_SKULL_DIE5,0,0},    // S_SKULL_DIE4
{SPR_SKUL,9,6,{NULL},S_SKULL_DIE6,0,0},    // S_SKULL_DIE5
{SPR_SKUL,10,6,{NULL},S_NULL,0,0},    // S_SKULL_DIE6
{SPR_SPID,0,10,{A_Look},S_SPID_STND2,0,0},    // S_SPID_STND
{SPR_SPID,1,10,{A_Look},S_SPID_STND,0,0},    // S_SPID_STND2
{SPR_SPID,0,3,{A_Metal},S_SPID_RUN2,0,0},    // S_SPID_RUN1
{SPR_SPID,0,3,{A_Chase},S_SPID_RUN3,0,0},    // S_SPID_RUN2
{SPR_SPID,1,3,{A_Chase},S_SPID_RUN4,0,0},    // S_SPID_RUN3
{SPR_SPID,1,3,{A_Chase},S_SPID_RUN5,0,0},    // S_SPID_RUN4
{SPR_SPID,2,3,{A_Metal},S_SPID_RUN6,0,0},    // S_SPID_RUN5
{SPR_SPID,2,3,{A_Chase},S_SPID_RUN7,0,0},    // S_SPID_RUN6
{SPR_SPID,3,3,{A_Chase},S_SPID_RUN8,0,0},    // S_SPID_RUN7
{SPR_SPID,3,3,{A_Chase},S_SPID_RUN9,0,0},    // S_SPID_RUN8
{SPR_SPID,4,3,{A_Metal},S_SPID_RUN10,0,0},    // S_SPID_RUN9
{SPR_SPID,4,3,{A_Chase},S_SPID_RUN11,0,0},    // S_SPID_RUN10
{SPR_SPID,5,3,{A_Chase},S_SPID_RUN12,0,0},    // S_SPID_RUN11
{SPR_SPID,5,3,{A_Chase},S_SPID_RUN1,0,0},    // S_SPID_RUN12
{SPR_SPID,32768,20,{A_FaceTarget},S_SPID_ATK2,0,0},    // S_SPID_ATK1
{SPR_SPID,32774,4,{A_SPosAttack},S_SPID_ATK3,0,0},    // S_SPID_ATK2
{SPR_SPID,32775,4,{A_SPosAttack},S_SPID_ATK4,0,0},    // S_SPID_ATK3
{SPR_SPID,32775,1,{A_SpidRefire},S_SPID_ATK2,0,0},    // S_SPID_ATK4
{SPR_SPID,8,3,{NULL},S_SPID_PAIN2,0,0},    // S_SPID_PAIN
{SPR_SPID,8,3,{A_Pain},S_SPID_RUN1,0,0},    // S_SPID_PAIN2
{SPR_SPID,9,20,{A_Scream},S_SPID_DIE2,0,0},    // S_SPID_DIE1
{SPR_SPID,10,10,{A_Fall},S_SPID_DIE3,0,0},    // S_SPID_DIE2
{SPR_SPID,11,10,{NULL},S_SPID_DIE4,0,0},    // S_SPID_DIE3
{SPR_SPID,12,10,{NULL},S_SPID_DIE5,0,0},    // S_SPID_DIE4
{SPR_SPID,13,10,{NULL},S_SPID_DIE6,0,0},    // S_SPID_DIE5
{SPR_SPID,14,10,{NULL},S_SPID_DIE7,0,0},    // S_SPID_DIE6
{SPR_SPID,15,10,{NULL},S_SPID_DIE8,0,0},    // S_SPID_DIE7
{SPR_SPID,16,10,{NULL},S_SPID_DIE9,0,0},    // S_SPID_DIE8
{SPR_SPID,17,10,{NULL},S_SPID_DIE10,0,0},    // S_SPID_DIE9
{SPR_SPID,18,30,{NULL},S_SPID_DIE11,0,0},    // S_SPID_DIE10
{SPR_SPID,18,-1,{A_BossDeath},S_NULL,0,0},    // S_SPID_DIE11
{SPR_BSPI,0,10,{A_Look},S_BSPI_STND2,0,0},    // S_BSPI_STND
{SPR_BSPI,1,10,{A_Look},S_BSPI_STND,0,0},    // S_BSPI_STND2
{SPR_BSPI,0,20,{NULL},S_BSPI_RUN1,0,0},    // S_BSPI_SIGHT
{SPR_BSPI,0,3,{A_BabyMetal},S_BSPI_RUN2,0,0},    // S_BSPI_RUN1
{SPR_BSPI,0,3,{A_Chase},S_BSPI_RUN3,0,0},    // S_BSPI_RUN2
{SPR_BSPI,1,3,{A_Chase},S_BSPI_RUN4,0,0},    // S_BSPI_RUN3
{SPR_BSPI,1,3,{A_Chase},S_BSPI_RUN5,0,0},    // S_BSPI_RUN4
{SPR_BSPI,2,3,{A_Chase},S_BSPI_RUN6,0,0},    // S_BSPI_RUN5
{SPR_BSPI,2,3,{A_Chase},S_BSPI_RUN7,0,0},    // S_BSPI_RUN6
{SPR_BSPI,3,3,{A_BabyMetal},S_BSPI_RUN8,0,0},    // S_BSPI_RUN7
{SPR_BSPI,3,3,{A_Chase},S_BSPI_RUN9,0,0},    // S_BSPI_RUN8
{SPR_BSPI,4,3,{A_Chase},S_BSPI_RUN10,0,0},    // S_BSPI_RUN9
{SPR_BSPI,4,3,{A_Chase},S_BSPI_RUN11,0,0},    // S_BSPI_RUN10
{SPR_BSPI,5,3,{A_Chase},S_BSPI_RUN12,0,0},    // S_BSPI_RUN11
{SPR_BSPI,5,3,{A_Chase},S_BSPI_RUN1,0,0},    // S_BSPI_RUN12
{SPR_BSPI,32768,20,{A_FaceTarget},S_BSPI_ATK2,0,0},    // S_BSPI_ATK1
{SPR_BSPI,32774,4,{A_BspiAttack},S_BSPI_ATK3,0,0},    // S_BSPI_ATK2

```

```

{SPR_BSPI,32775,4,{NULL},S_BSPI_ATK4,0,0}, // S_BSPI_ATK3
{SPR_BSPI,32775,1,{A_SpidRefire},S_BSPI_ATK2,0,0}, // S_BSPI_ATK4
{SPR_BSPI,8,3,{NULL},S_BSPI_PAIN2,0,0}, // S_BSPI_PAIN
{SPR_BSPI,8,3,{A_Pain},S_BSPI_RUN1,0,0}, // S_BSPI_PAIN2
{SPR_BSPI,9,20,{A_Scream},S_BSPI_DIE2,0,0}, // S_BSPI_DIE1
{SPR_BSPI,10,7,{A_Fall},S_BSPI_DIE3,0,0}, // S_BSPI_DIE2
{SPR_BSPI,11,7,{NULL},S_BSPI_DIE4,0,0}, // S_BSPI_DIE3
{SPR_BSPI,12,7,{NULL},S_BSPI_DIE5,0,0}, // S_BSPI_DIE4
{SPR_BSPI,13,7,{NULL},S_BSPI_DIE6,0,0}, // S_BSPI_DIE5
{SPR_BSPI,14,7,{NULL},S_BSPI_DIE7,0,0}, // S_BSPI_DIE6
{SPR_BSPI,15,-1,{A_BossDeath},S_NULL,0,0}, // S_BSPI_DIE7
{SPR_BSPI,15,5,{NULL},S_BSPI_RAISE2,0,0}, // S_BSPI_RAISE1
{SPR_BSPI,14,5,{NULL},S_BSPI_RAISE3,0,0}, // S_BSPI_RAISE2
{SPR_BSPI,13,5,{NULL},S_BSPI_RAISE4,0,0}, // S_BSPI_RAISE3
{SPR_BSPI,12,5,{NULL},S_BSPI_RAISE5,0,0}, // S_BSPI_RAISE4
{SPR_BSPI,11,5,{NULL},S_BSPI_RAISE6,0,0}, // S_BSPI_RAISE5
{SPR_BSPI,10,5,{NULL},S_BSPI_RAISE7,0,0}, // S_BSPI_RAISE6
{SPR_BSPI,9,5,{NULL},S_BSPI_RUN1,0,0}, // S_BSPI_RAISE7
{SPR_APLS,32768,5,{NULL},S_ARACH_PLAZ2,0,0}, // S_ARACH_PLAZ
{SPR_APLS,32769,5,{NULL},S_ARACH_PLAZ,0,0}, // S_ARACH_PLAZ2
{SPR_APBX,32768,5,{NULL},S_ARACH_PLEX2,0,0}, // S_ARACH_PLEX
{SPR_APBX,32769,5,{NULL},S_ARACH_PLEX3,0,0}, // S_ARACH_PLEX2
{SPR_APBX,32770,5,{NULL},S_ARACH_PLEX4,0,0}, // S_ARACH_PLEX3
{SPR_APBX,32771,5,{NULL},S_ARACH_PLEX5,0,0}, // S_ARACH_PLEX4
{SPR_APBX,32772,5,{NULL},S_NULL,0,0}, // S_ARACH_PLEX5
{SPR_CYBR,0,10,{A_Look},S_CYBER_STND2,0,0}, // S_CYBER_STND
{SPR_CYBR,1,10,{A_Look},S_CYBER_STND,0,0}, // S_CYBER_STND2
{SPR_CYBR,0,3,{A_Hoof},S_CYBER_RUN2,0,0}, // S_CYBER_RUN1
{SPR_CYBR,0,3,{A_Chase},S_CYBER_RUN3,0,0}, // S_CYBER_RUN2
{SPR_CYBR,1,3,{A_Chase},S_CYBER_RUN4,0,0}, // S_CYBER_RUN3
{SPR_CYBR,1,3,{A_Chase},S_CYBER_RUN5,0,0}, // S_CYBER_RUN4
{SPR_CYBR,2,3,{A_Chase},S_CYBER_RUN6,0,0}, // S_CYBER_RUN5
{SPR_CYBR,2,3,{A_Chase},S_CYBER_RUN7,0,0}, // S_CYBER_RUN6
{SPR_CYBR,3,3,{A_Metal},S_CYBER_RUN8,0,0}, // S_CYBER_RUN7
{SPR_CYBR,3,3,{A_Chase},S_CYBER_RUN1,0,0}, // S_CYBER_RUN8
{SPR_CYBR,4,6,{A_FaceTarget},S_CYBER_ATK2,0,0}, // S_CYBER_ATK1
{SPR_CYBR,5,12,{A_CyberAttack},S_CYBER_ATK3,0,0}, // S_CYBER_ATK2
{SPR_CYBR,4,12,{A_FaceTarget},S_CYBER_ATK4,0,0}, // S_CYBER_ATK3
{SPR_CYBR,5,12,{A_CyberAttack},S_CYBER_ATK5,0,0}, // S_CYBER_ATK4
{SPR_CYBR,4,12,{A_FaceTarget},S_CYBER_ATK6,0,0}, // S_CYBER_ATK5
{SPR_CYBR,5,12,{A_CyberAttack},S_CYBER_RUN1,0,0}, // S_CYBER_ATK6
{SPR_CYBR,6,10,{A_Pain},S_CYBER_RUN1,0,0}, // S_CYBER_PAIN
{SPR_CYBR,7,10,{NULL},S_CYBER_DIE2,0,0}, // S_CYBER_DIE1
{SPR_CYBR,8,10,{A_Scream},S_CYBER_DIE3,0,0}, // S_CYBER_DIE2
{SPR_CYBR,9,10,{NULL},S_CYBER_DIE4,0,0}, // S_CYBER_DIE3
{SPR_CYBR,10,10,{NULL},S_CYBER_DIE5,0,0}, // S_CYBER_DIE4
{SPR_CYBR,11,10,{NULL},S_CYBER_DIE6,0,0}, // S_CYBER_DIE5
{SPR_CYBR,12,10,{A_Fall},S_CYBER_DIE7,0,0}, // S_CYBER_DIE6
{SPR_CYBR,13,10,{NULL},S_CYBER_DIE8,0,0}, // S_CYBER_DIE7
{SPR_CYBR,14,10,{NULL},S_CYBER_DIE9,0,0}, // S_CYBER_DIE8
{SPR_CYBR,15,30,{NULL},S_CYBER_DIE10,0,0}, // S_CYBER_DIE9
{SPR_CYBR,15,-1,{A_BossDeath},S_NULL,0,0}, // S_CYBER_DIE10
{SPR_PAIN,0,10,{A_Look},S_PAIN_STND,0,0}, // S_PAIN_STND
{SPR_PAIN,0,3,{A_Chase},S_PAIN_RUN2,0,0}, // S_PAIN_RUN1
{SPR_PAIN,0,3,{A_Chase},S_PAIN_RUN3,0,0}, // S_PAIN_RUN2
{SPR_PAIN,1,3,{A_Chase},S_PAIN_RUN4,0,0}, // S_PAIN_RUN3
{SPR_PAIN,1,3,{A_Chase},S_PAIN_RUN5,0,0}, // S_PAIN_RUN4
{SPR_PAIN,2,3,{A_Chase},S_PAIN_RUN6,0,0}, // S_PAIN_RUN5
{SPR_PAIN,2,3,{A_Chase},S_PAIN_RUN1,0,0}, // S_PAIN_RUN6
{SPR_PAIN,3,5,{A_FaceTarget},S_PAIN_ATK2,0,0}, // S_PAIN_ATK1
{SPR_PAIN,4,5,{A_FaceTarget},S_PAIN_ATK3,0,0}, // S_PAIN_ATK2
{SPR_PAIN,32773,5,{A_FaceTarget},S_PAIN_ATK4,0,0}, // S_PAIN_ATK3
{SPR_PAIN,32773,0,{A_PainAttack},S_PAIN_RUN1,0,0}, // S_PAIN_ATK4
{SPR_PAIN,6,6,{NULL},S_PAIN_PAIN2,0,0}, // S_PAIN_PAIN
{SPR_PAIN,6,6,{A_Pain},S_PAIN_RUN1,0,0}, // S_PAIN_PAIN2

```

```

{SPR_PAIN,32775,8,{NULL},S_PAIN_DIE2,0,0}, // S_PAIN_DIE1
{SPR_PAIN,32776,8,{A_Scream},S_PAIN_DIE3,0,0}, // S_PAIN_DIE2
{SPR_PAIN,32777,8,{NULL},S_PAIN_DIE4,0,0}, // S_PAIN_DIE3
{SPR_PAIN,32778,8,{NULL},S_PAIN_DIE5,0,0}, // S_PAIN_DIE4
{SPR_PAIN,32779,8,{A_PainDie},S_PAIN_DIE6,0,0}, // S_PAIN_DIE5
{SPR_PAIN,32780,8,{NULL},S_NULL,0,0}, // S_PAIN_DIE6
{SPR_PAIN,12,8,{NULL},S_PAIN_RAISE2,0,0}, // S_PAIN_RAISE1
{SPR_PAIN,11,8,{NULL},S_PAIN_RAISE3,0,0}, // S_PAIN_RAISE2
{SPR_PAIN,10,8,{NULL},S_PAIN_RAISE4,0,0}, // S_PAIN_RAISE3
{SPR_PAIN,9,8,{NULL},S_PAIN_RAISE5,0,0}, // S_PAIN_RAISE4
{SPR_PAIN,8,8,{NULL},S_PAIN_RAISE6,0,0}, // S_PAIN_RAISE5
{SPR_PAIN,7,8,{NULL},S_PAIN_RUN1,0,0}, // S_PAIN_RAISE6
{SPR_SSWV,0,10,{A_Look},S_SSWV_STND2,0,0}, // S_SSWV_STND
{SPR_SSWV,1,10,{A_Look},S_SSWV_STND,0,0}, // S_SSWV_STND2
{SPR_SSWV,0,3,{A_Chase},S_SSWV_RUN2,0,0}, // S_SSWV_RUN1
{SPR_SSWV,0,3,{A_Chase},S_SSWV_RUN3,0,0}, // S_SSWV_RUN2
{SPR_SSWV,1,3,{A_Chase},S_SSWV_RUN4,0,0}, // S_SSWV_RUN3
{SPR_SSWV,1,3,{A_Chase},S_SSWV_RUN5,0,0}, // S_SSWV_RUN4
{SPR_SSWV,2,3,{A_Chase},S_SSWV_RUN6,0,0}, // S_SSWV_RUN5
{SPR_SSWV,2,3,{A_Chase},S_SSWV_RUN7,0,0}, // S_SSWV_RUN6
{SPR_SSWV,3,3,{A_Chase},S_SSWV_RUN8,0,0}, // S_SSWV_RUN7
{SPR_SSWV,3,3,{A_Chase},S_SSWV_RUN1,0,0}, // S_SSWV_RUN8
{SPR_SSWV,4,10,{A_FaceTarget},S_SSWV_ATK2,0,0}, // S_SSWV_ATK1
{SPR_SSWV,5,10,{A_FaceTarget},S_SSWV_ATK3,0,0}, // S_SSWV_ATK2
{SPR_SSWV,32774,4,{A_CPosAttack},S_SSWV_ATK4,0,0}, // S_SSWV_ATK3
{SPR_SSWV,5,6,{A_FaceTarget},S_SSWV_ATK5,0,0}, // S_SSWV_ATK4
{SPR_SSWV,32774,4,{A_CPosAttack},S_SSWV_ATK6,0,0}, // S_SSWV_ATK5
{SPR_SSWV,5,1,{A_CPosRefire},S_SSWV_ATK2,0,0}, // S_SSWV_ATK6
{SPR_SSWV,7,3,{NULL},S_SSWV_PAIN2,0,0}, // S_SSWV_PAIN
{SPR_SSWV,7,3,{A_Pain},S_SSWV_RUN1,0,0}, // S_SSWV_PAIN2
{SPR_SSWV,8,5,{NULL},S_SSWV_DIE2,0,0}, // S_SSWV_DIE1
{SPR_SSWV,9,5,{A_Scream},S_SSWV_DIE3,0,0}, // S_SSWV_DIE2
{SPR_SSWV,10,5,{A_Fall},S_SSWV_DIE4,0,0}, // S_SSWV_DIE3
{SPR_SSWV,11,5,{NULL},S_SSWV_DIE5,0,0}, // S_SSWV_DIE4
{SPR_SSWV,12,-1,{NULL},S_NULL,0,0}, // S_SSWV_DIE5
{SPR_SSWV,13,5,{NULL},S_SSWV_XDIE2,0,0}, // S_SSWV_XDIE1
{SPR_SSWV,14,5,{A_XScream},S_SSWV_XDIE3,0,0}, // S_SSWV_XDIE2
{SPR_SSWV,15,5,{A_Fall},S_SSWV_XDIE4,0,0}, // S_SSWV_XDIE3
{SPR_SSWV,16,5,{NULL},S_SSWV_XDIE5,0,0}, // S_SSWV_XDIE4
{SPR_SSWV,17,5,{NULL},S_SSWV_XDIE6,0,0}, // S_SSWV_XDIE5
{SPR_SSWV,18,5,{NULL},S_SSWV_XDIE7,0,0}, // S_SSWV_XDIE6
{SPR_SSWV,19,5,{NULL},S_SSWV_XDIE8,0,0}, // S_SSWV_XDIE7
{SPR_SSWV,20,5,{NULL},S_SSWV_XDIE9,0,0}, // S_SSWV_XDIE8
{SPR_SSWV,21,-1,{NULL},S_NULL,0,0}, // S_SSWV_XDIE9
{SPR_SSWV,12,5,{NULL},S_SSWV_RAISE2,0,0}, // S_SSWV_RAISE1
{SPR_SSWV,11,5,{NULL},S_SSWV_RAISE3,0,0}, // S_SSWV_RAISE2
{SPR_SSWV,10,5,{NULL},S_SSWV_RAISE4,0,0}, // S_SSWV_RAISE3
{SPR_SSWV,9,5,{NULL},S_SSWV_RAISE5,0,0}, // S_SSWV_RAISE4
{SPR_SSWV,8,5,{NULL},S_SSWV_RUN1,0,0}, // S_SSWV_RAISE5
{SPR_KEEN,0,-1,{NULL},S_KEENSTND,0,0}, // S_KEENSTND
{SPR_KEEN,0,6,{NULL},S_COMMKEEN2,0,0}, // S_COMMKEEN
{SPR_KEEN,1,6,{NULL},S_COMMKEEN3,0,0}, // S_COMMKEEN2
{SPR_KEEN,2,6,{A_Scream},S_COMMKEEN4,0,0}, // S_COMMKEEN3
{SPR_KEEN,3,6,{NULL},S_COMMKEEN5,0,0}, // S_COMMKEEN4
{SPR_KEEN,4,6,{NULL},S_COMMKEEN6,0,0}, // S_COMMKEEN5
{SPR_KEEN,5,6,{NULL},S_COMMKEEN7,0,0}, // S_COMMKEEN6
{SPR_KEEN,6,6,{NULL},S_COMMKEEN8,0,0}, // S_COMMKEEN7
{SPR_KEEN,7,6,{NULL},S_COMMKEEN9,0,0}, // S_COMMKEEN8
{SPR_KEEN,8,6,{NULL},S_COMMKEEN10,0,0}, // S_COMMKEEN9
{SPR_KEEN,9,6,{NULL},S_COMMKEEN11,0,0}, // S_COMMKEEN10
{SPR_KEEN,10,6,{A_KeenDie},S_COMMKEEN12,0,0}, // S_COMMKEEN11
{SPR_KEEN,11,-1,{NULL},S_NULL,0,0}, // S_COMMKEEN12
{SPR_KEEN,12,4,{NULL},S_KEENPAIN2,0,0}, // S_KEENPAIN
{SPR_KEEN,12,8,{A_Pain},S_KEENSTND,0,0}, // S_KEENPAIN2
{SPR_BBRN,0,-1,{NULL},S_NULL,0,0}, // S_BRAIN

```

```

{SPR_BBRN,1,36,{A_BrainPain},S_BRAIN,0,0},          // S_BRAIN_PAIN
{SPR_BBRN,0,100,{A_BrainScream},S_BRAIN_DIE2,0,0},    // S_BRAIN_DIE1
{SPR_BBRN,0,10,{NULL},S_BRAIN_DIE3,0,0},            // S_BRAIN_DIE2
{SPR_BBRN,0,10,{NULL},S_BRAIN_DIE4,0,0},            // S_BRAIN_DIE3
{SPR_BBRN,0,-1,{A_BrainDie},S_NULL,0,0},             // S_BRAIN_DIE4
{SPR_SSWV,0,10,{A_Look},S_BRAINEYE,0,0},            // S_BRAINEYE
{SPR_SSWV,0,181,{A_BrainAwake},S_BRAINEYE1,0,0},     // S_BRAINEYESEE
{SPR_SSWV,0,150,{A_BrainSpit},S_BRAINEYE1,0,0},     // S_BRAINEYE1
{SPR_BOSF,32768,3,{A_SpawnSound},S_SPAWN2,0,0},      // S_SPAWN1
{SPR_BOSF,32769,3,{A_SpawnFly},S_SPAWN3,0,0},       // S_SPAWN2
{SPR_BOSF,32770,3,{A_SpawnFly},S_SPAWN4,0,0},       // S_SPAWN3
{SPR_BOSF,32771,3,{A_SpawnFly},S_SPAWN1,0,0},       // S_SPAWN4
{SPR_FIRE,32768,4,{A_Fire},S_SPAWNFIRE2,0,0},       // S_SPAWNFIRE1
{SPR_FIRE,32769,4,{A_Fire},S_SPAWNFIRE3,0,0},       // S_SPAWNFIRE2
{SPR_FIRE,32770,4,{A_Fire},S_SPAWNFIRE4,0,0},       // S_SPAWNFIRE3
{SPR_FIRE,32771,4,{A_Fire},S_SPAWNFIRE5,0,0},       // S_SPAWNFIRE4
{SPR_FIRE,32772,4,{A_Fire},S_SPAWNFIRE6,0,0},       // S_SPAWNFIRE5
{SPR_FIRE,32773,4,{A_Fire},S_SPAWNFIRE7,0,0},       // S_SPAWNFIRE6
{SPR_FIRE,32774,4,{A_Fire},S_SPAWNFIRE8,0,0},       // S_SPAWNFIRE7
{SPR_FIRE,32775,4,{A_Fire},S_NULL,0,0},             // S_SPAWNFIRE8
{SPR_MISL,32769,10,{NULL},S_BRAINEXPLODE2,0,0},     // S_BRAINEXPLODE1
{SPR_MISL,32770,10,{NULL},S_BRAINEXPLODE3,0,0},     // S_BRAINEXPLODE2
{SPR_MISL,32771,10,{A_BrainExplode},S_NULL,0,0},    // S_BRAINEXPLODE3
{SPR_ARM1,0,6,{NULL},S_ARM1A,0,0},                 // S_ARM1
{SPR_ARM1,32769,7,{NULL},S_ARM1,0,0},              // S_ARM1A
{SPR_ARM2,0,6,{NULL},S_ARM2A,0,0},                 // S_ARM2
{SPR_ARM2,32769,6,{NULL},S_ARM2,0,0},              // S_ARM2A
{SPR_BAR1,0,6,{NULL},S_BAR2,0,0},                  // S_BAR1
{SPR_BAR1,1,6,{NULL},S_BAR1,0,0},                  // S_BAR2
{SPR_BEXP,32768,5,{NULL},S_BEXP2,0,0},             // S_BEXP
{SPR_BEXP,32769,5,{A_Scream},S_BEXP3,0,0},         // S_BEXP2
{SPR_BEXP,32770,5,{NULL},S_BEXP4,0,0},             // S_BEXP3
{SPR_BEXP,32771,10,{A_Explode},S_BEXP5,0,0},       // S_BEXP4
{SPR_BEXP,32772,10,{NULL},S_NULL,0,0},             // S_BEXP5
{SPR_FCAN,32768,4,{NULL},S_BBAR2,0,0},             // S_BBAR1
{SPR_FCAN,32769,4,{NULL},S_BBAR3,0,0},             // S_BBAR2
{SPR_FCAN,32770,4,{NULL},S_BBAR1,0,0},             // S_BBAR3
{SPR_BON1,0,6,{NULL},S_BON1A,0,0},                 // S_BON1
{SPR_BON1,1,6,{NULL},S_BON1B,0,0},                 // S_BON1A
{SPR_BON1,2,6,{NULL},S_BON1C,0,0},                 // S_BON1B
{SPR_BON1,3,6,{NULL},S_BON1D,0,0},                 // S_BON1C
{SPR_BON1,2,6,{NULL},S_BON1E,0,0},                 // S_BON1D
{SPR_BON1,1,6,{NULL},S_BON1,0,0},                  // S_BON1E
{SPR_BON2,0,6,{NULL},S_BON2A,0,0},                 // S_BON2
{SPR_BON2,1,6,{NULL},S_BON2B,0,0},                 // S_BON2A
{SPR_BON2,2,6,{NULL},S_BON2C,0,0},                 // S_BON2B
{SPR_BON2,3,6,{NULL},S_BON2D,0,0},                 // S_BON2C
{SPR_BON2,2,6,{NULL},S_BON2E,0,0},                 // S_BON2D
{SPR_BON2,1,6,{NULL},S_BON2,0,0},                  // S_BON2E
{SPR_BKEY,0,10,{NULL},S_BKEY2,0,0},                // S_BKEY
{SPR_BKEY,32769,10,{NULL},S_BKEY,0,0},             // S_BKEY2
{SPR_RKEY,0,10,{NULL},S_RKEY2,0,0},                // S_RKEY
{SPR_RKEY,32769,10,{NULL},S_RKEY,0,0},             // S_RKEY2
{SPR_YKEY,0,10,{NULL},S_YKEY2,0,0},                // S_YKEY
{SPR_YKEY,32769,10,{NULL},S_YKEY,0,0},             // S_YKEY2
{SPR_BSKU,0,10,{NULL},S_BSKULL2,0,0},              // S_BSKULL
{SPR_BSKU,32769,10,{NULL},S_BSKULL,0,0},           // S_BSKULL2
{SPR_RSKU,0,10,{NULL},S_RSKULL2,0,0},              // S_RSKULL
{SPR_RSKU,32769,10,{NULL},S_RSKULL,0,0},           // S_RSKULL2
{SPR_YSKU,0,10,{NULL},S_YSKULL2,0,0},              // S_YSKULL
{SPR_YSKU,32769,10,{NULL},S_YSKULL,0,0},           // S_YSKULL2
{SPR_STIM,0,-1,{NULL},S_NULL,0,0},                 // S_STIM
{SPR_MEDI,0,-1,{NULL},S_NULL,0,0},                 // S_MEDI
{SPR_SOUL,32768,6,{NULL},S_SOUL2,0,0},             // S_SOUL
{SPR_SOUL,32769,6,{NULL},S_SOUL3,0,0},             // S_SOUL2

```

```

{SPR_SOUL,32770,6,{NULL},S_SOUL4,0,0}, // S_SOUL3
{SPR_SOUL,32771,6,{NULL},S_SOUL5,0,0}, // S_SOUL4
{SPR_SOUL,32770,6,{NULL},S_SOUL6,0,0}, // S_SOUL5
{SPR_SOUL,32769,6,{NULL},S_SOUL,0,0}, // S_SOUL6
{SPR_PINV,32768,6,{NULL},S_PINV2,0,0}, // S_PINV
{SPR_PINV,32769,6,{NULL},S_PINV3,0,0}, // S_PINV2
{SPR_PINV,32770,6,{NULL},S_PINV4,0,0}, // S_PINV3
{SPR_PINV,32771,6,{NULL},S_PINV,0,0}, // S_PINV4
{SPR_PSTR,32768,-1,{NULL},S_NULL,0,0}, // S_PSTR
{SPR_PINS,32768,6,{NULL},S_PINS2,0,0}, // S_PINS
{SPR_PINS,32769,6,{NULL},S_PINS3,0,0}, // S_PINS2
{SPR_PINS,32770,6,{NULL},S_PINS4,0,0}, // S_PINS3
{SPR_PINS,32771,6,{NULL},S_PINS,0,0}, // S_PINS4
{SPR_MEGA,32768,6,{NULL},S_MEGA2,0,0}, // S_MEGA
{SPR_MEGA,32769,6,{NULL},S_MEGA3,0,0}, // S_MEGA2
{SPR_MEGA,32770,6,{NULL},S_MEGA4,0,0}, // S_MEGA3
{SPR_MEGA,32771,6,{NULL},S_MEGA,0,0}, // S_MEGA4
{SPR_SUIT,32768,-1,{NULL},S_NULL,0,0}, // S_SUIT
{SPR_PMAP,32768,6,{NULL},S_PMAP2,0,0}, // S_PMAP
{SPR_PMAP,32769,6,{NULL},S_PMAP3,0,0}, // S_PMAP2
{SPR_PMAP,32770,6,{NULL},S_PMAP4,0,0}, // S_PMAP3
{SPR_PMAP,32771,6,{NULL},S_PMAP5,0,0}, // S_PMAP4
{SPR_PMAP,32770,6,{NULL},S_PMAP6,0,0}, // S_PMAP5
{SPR_PMAP,32769,6,{NULL},S_PMAP,0,0}, // S_PMAP6
{SPR_PVIS,32768,6,{NULL},S_PVIS2,0,0}, // S_PVIS
{SPR_PVIS,1,6,{NULL},S_PVIS,0,0}, // S_PVIS2
{SPR_CLIP,0,-1,{NULL},S_NULL,0,0}, // S_CLIP
{SPR_AMMO,0,-1,{NULL},S_NULL,0,0}, // S_AMMO
{SPR_ROCK,0,-1,{NULL},S_NULL,0,0}, // S_ROCK
{SPR_BROK,0,-1,{NULL},S_NULL,0,0}, // S_BROK
{SPR_CELL,0,-1,{NULL},S_NULL,0,0}, // S_CELL
{SPR_CELP,0,-1,{NULL},S_NULL,0,0}, // S_CELP
{SPR_SHEL,0,-1,{NULL},S_NULL,0,0}, // S_SHEL
{SPR_SBOX,0,-1,{NULL},S_NULL,0,0}, // S_SBOX
{SPR_BPAK,0,-1,{NULL},S_NULL,0,0}, // S_BPAK
{SPR_BFUG,0,-1,{NULL},S_NULL,0,0}, // S_BFUG
{SPR_MGUN,0,-1,{NULL},S_NULL,0,0}, // S_MGUN
{SPR_CSAW,0,-1,{NULL},S_NULL,0,0}, // S_CSAW
{SPR_LAUN,0,-1,{NULL},S_NULL,0,0}, // S_LAUN
{SPR_PLAS,0,-1,{NULL},S_NULL,0,0}, // S_PLAS
{SPR_SHOT,0,-1,{NULL},S_NULL,0,0}, // S_SHOT
{SPR_SGN2,0,-1,{NULL},S_NULL,0,0}, // S_SHOT2
{SPR_COLU,32768,-1,{NULL},S_NULL,0,0}, // S_COLU
{SPR_SMT2,0,-1,{NULL},S_NULL,0,0}, // S_STALAG
{SPR_GOR1,0,10,{NULL},S_BLOODYTWITCH2,0,0}, // S_BLOODYTWITCH
{SPR_GOR1,1,15,{NULL},S_BLOODYTWITCH3,0,0}, // S_BLOODYTWITCH2
{SPR_GOR1,2,8,{NULL},S_BLOODYTWITCH4,0,0}, // S_BLOODYTWITCH3
{SPR_GOR1,1,6,{NULL},S_BLOODYTWITCH,0,0}, // S_BLOODYTWITCH4
{SPR_PLAY,13,-1,{NULL},S_NULL,0,0}, // S_DEADTORSO
{SPR_PLAY,18,-1,{NULL},S_NULL,0,0}, // S_DEADBOTTOM
{SPR_POL2,0,-1,{NULL},S_NULL,0,0}, // S_HEADSONSTICK
{SPR_POL5,0,-1,{NULL},S_NULL,0,0}, // S_GIBS
{SPR_POL4,0,-1,{NULL},S_NULL,0,0}, // S_HEADONASTICK
{SPR_POL3,32768,6,{NULL},S_HEADCANDLES2,0,0}, // S_HEADCANDLES
{SPR_POL3,32769,6,{NULL},S_HEADCANDLES,0,0}, // S_HEADCANDLES2
{SPR_POL1,0,-1,{NULL},S_NULL,0,0}, // S_DEADSTICK
{SPR_POL6,0,6,{NULL},S_LIVESTICK2,0,0}, // S_LIVESTICK
{SPR_POL6,1,8,{NULL},S_LIVESTICK,0,0}, // S_LIVESTICK2
{SPR_GOR2,0,-1,{NULL},S_NULL,0,0}, // S_MEAT2
{SPR_GOR3,0,-1,{NULL},S_NULL,0,0}, // S_MEAT3
{SPR_GOR4,0,-1,{NULL},S_NULL,0,0}, // S_MEAT4
{SPR_GOR5,0,-1,{NULL},S_NULL,0,0}, // S_MEAT5
{SPR_SMIT,0,-1,{NULL},S_NULL,0,0}, // S_STALAGTITE
{SPR_COL1,0,-1,{NULL},S_NULL,0,0}, // S_TALLGRNCOL
{SPR_COL2,0,-1,{NULL},S_NULL,0,0}, // S_SHRTGRNCOL

```

```

{SPR_COL3,0,-1,{NULL},S_NULL,0,0}, // S_TALLREDCOL
{SPR_COL4,0,-1,{NULL},S_NULL,0,0}, // S_SHRTREDCOL
{SPR_CAND,32768,-1,{NULL},S_NULL,0,0}, // S_CANDLESTIK
{SPR_CBRA,32768,-1,{NULL},S_NULL,0,0}, // S_CANDELABRA
{SPR_COL6,0,-1,{NULL},S_NULL,0,0}, // S_SKULLCOL
{SPR_TRE1,0,-1,{NULL},S_NULL,0,0}, // S_TORCHTREE
{SPR_TRE2,0,-1,{NULL},S_NULL,0,0}, // S_BIGTREE
{SPR_ELEC,0,-1,{NULL},S_NULL,0,0}, // S_TECHPILLAR
{SPR_CEYE,32768,6,{NULL},S_EVILEYE2,0,0}, // S_EVILEYE
{SPR_CEYE,32769,6,{NULL},S_EVILEYE3,0,0}, // S_EVILEYE2
{SPR_CEYE,32770,6,{NULL},S_EVILEYE4,0,0}, // S_EVILEYE3
{SPR_CEYE,32769,6,{NULL},S_EVILEYE,0,0}, // S_EVILEYE4
{SPR_FSKU,32768,6,{NULL},S_FLOATSKULL2,0,0}, // S_FLOATSKULL
{SPR_FSKU,32769,6,{NULL},S_FLOATSKULL3,0,0}, // S_FLOATSKULL2
{SPR_FSKU,32770,6,{NULL},S_FLOATSKULL,0,0}, // S_FLOATSKULL3
{SPR_COL5,0,14,{NULL},S_HEARTCOL2,0,0}, // S_HEARTCOL
{SPR_COL5,1,14,{NULL},S_HEARTCOL,0,0}, // S_HEARTCOL2
{SPR_TBLU,32768,4,{NULL},S_BLUETORCH2,0,0}, // S_BLUETORCH
{SPR_TBLU,32769,4,{NULL},S_BLUETORCH3,0,0}, // S_BLUETORCH2
{SPR_TBLU,32770,4,{NULL},S_BLUETORCH4,0,0}, // S_BLUETORCH3
{SPR_TBLU,32771,4,{NULL},S_BLUETORCH,0,0}, // S_BLUETORCH4
{SPR_TGRN,32768,4,{NULL},S_GREENTORCH2,0,0}, // S_GREENTORCH
{SPR_TGRN,32769,4,{NULL},S_GREENTORCH3,0,0}, // S_GREENTORCH2
{SPR_TGRN,32770,4,{NULL},S_GREENTORCH4,0,0}, // S_GREENTORCH3
{SPR_TGRN,32771,4,{NULL},S_GREENTORCH,0,0}, // S_GREENTORCH4
{SPR_TRED,32768,4,{NULL},S_REDTORCH2,0,0}, // S_REDTORCH
{SPR_TRED,32769,4,{NULL},S_REDTORCH3,0,0}, // S_REDTORCH2
{SPR_TRED,32770,4,{NULL},S_REDTORCH4,0,0}, // S_REDTORCH3
{SPR_TRED,32771,4,{NULL},S_REDTORCH,0,0}, // S_REDTORCH4
{SPR_SMBT,32768,4,{NULL},S_BTORCHSHRT2,0,0}, // S_BTORCHSHRT
{SPR_SMBT,32769,4,{NULL},S_BTORCHSHRT3,0,0}, // S_BTORCHSHRT2
{SPR_SMBT,32770,4,{NULL},S_BTORCHSHRT4,0,0}, // S_BTORCHSHRT3
{SPR_SMBT,32771,4,{NULL},S_BTORCHSHRT,0,0}, // S_BTORCHSHRT4
{SPR_SMG,32768,4,{NULL},S_GTORCHSHRT2,0,0}, // S_GTORCHSHRT
{SPR_SMG,32769,4,{NULL},S_GTORCHSHRT3,0,0}, // S_GTORCHSHRT2
{SPR_SMG,32770,4,{NULL},S_GTORCHSHRT4,0,0}, // S_GTORCHSHRT3
{SPR_SMG,32771,4,{NULL},S_GTORCHSHRT,0,0}, // S_GTORCHSHRT4
{SPR_SMRT,32768,4,{NULL},S_RTORCHSHRT2,0,0}, // S_RTORCHSHRT
{SPR_SMRT,32769,4,{NULL},S_RTORCHSHRT3,0,0}, // S_RTORCHSHRT2
{SPR_SMRT,32770,4,{NULL},S_RTORCHSHRT4,0,0}, // S_RTORCHSHRT3
{SPR_SMRT,32771,4,{NULL},S_RTORCHSHRT,0,0}, // S_RTORCHSHRT4
{SPR_HDB1,0,-1,{NULL},S_NULL,0,0}, // S_HANGNOGUTS
{SPR_HDB2,0,-1,{NULL},S_NULL,0,0}, // S_HANGNOBRAIN
{SPR_HDB3,0,-1,{NULL},S_NULL,0,0}, // S_HANGTLOOKDN
{SPR_HDB4,0,-1,{NULL},S_NULL,0,0}, // S_HANGTSKULL
{SPR_HDB5,0,-1,{NULL},S_NULL,0,0}, // S_HANGTLOOKUP
{SPR_HDB6,0,-1,{NULL},S_NULL,0,0}, // S_HANGTNOBRAIN
{SPR_POB1,0,-1,{NULL},S_NULL,0,0}, // S_COLONGIBS
{SPR_POB2,0,-1,{NULL},S_NULL,0,0}, // S_SMALLPOOL
{SPR_BRS1,0,-1,{NULL},S_NULL,0,0}, // S_BRAINSTEM
{SPR_TLMP,32768,4,{NULL},S_TECHLAMP2,0,0}, // S_TECHLAMP
{SPR_TLMP,32769,4,{NULL},S_TECHLAMP3,0,0}, // S_TECHLAMP2
{SPR_TLMP,32770,4,{NULL},S_TECHLAMP4,0,0}, // S_TECHLAMP3
{SPR_TLMP,32771,4,{NULL},S_TECHLAMP,0,0}, // S_TECHLAMP4
{SPR_TLP2,32768,4,{NULL},S_TECH2LAMP2,0,0}, // S_TECH2LAMP
{SPR_TLP2,32769,4,{NULL},S_TECH2LAMP3,0,0}, // S_TECH2LAMP2
{SPR_TLP2,32770,4,{NULL},S_TECH2LAMP4,0,0}, // S_TECH2LAMP3
{SPR_TLP2,32771,4,{NULL},S_TECH2LAMP,0,0}, // S_TECH2LAMP4
};

```

```

mobjinfo_t mobjinfo[NUMMOBJTYPES] = {

    { // MT_PLAYER
      -1, // doomednum

```

```

S_PLAY,                // spawnstate
100,                   // spawnhealth
S_PLAY_RUN1,           // seestate
sfx_None,              // seesound
0,                     // reactiontime
sfx_None,              // attacksound
S_PLAY_PAIN,           // painstate
255,                   // painchance
sfx_plpain,            // painsound
S_NULL,                // meleestate
S_PLAY_ATK1,           // missilestate
S_PLAY_DIE1,           // deathstate
S_PLAY_XDIE1,          // xdeathstate
sfx_pldeth,            // deathsound
0,                     // speed
16*FRACUNIT,          // radius
56*FRACUNIT,          // height
100,                   // mass
0,                     // damage
sfx_None,              // activesound
MF_SOLID|MF_SHOOTABLE|MF_DROPOFF|MF_PICKUP|MF_NOTDMATCH, // flags
S_NULL                 // raisestate
},

{                       // MT_POSSESSED
3004,                  // doomednum
S_POSS_STND,           // spawnstate
20,                    // spawnhealth
S_POSS_RUN1,           // seestate
sfx_posit1,            // seesound
8,                     // reactiontime
sfx_pistol,            // attacksound
S_POSS_PAIN,           // painstate
200,                   // painchance
sfx_popain,            // painsound
0,                     // meleestate
S_POSS_ATK1,           // missilestate
S_POSS_DIE1,           // deathstate
S_POSS_XDIE1,          // xdeathstate
sfx_podth1,            // deathsound
8,                     // speed
20*FRACUNIT,          // radius
56*FRACUNIT,          // height
100,                   // mass
0,                     // damage
sfx_posact,            // activesound
MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
S_POSS_RAISE1         // raisestate
},

{                       // MT_SHOTGUY
9,                     // doomednum
S_SPOS_STND,           // spawnstate
30,                    // spawnhealth
S_SPOS_RUN1,           // seestate
sfx_posit2,            // seesound
8,                     // reactiontime
0,                     // attacksound
S_SPOS_PAIN,           // painstate
170,                   // painchance
sfx_popain,            // painsound
0,                     // meleestate
S_SPOS_ATK1,           // missilestate
S_SPOS_DIE1,           // deathstate
S_SPOS_XDIE1,          // xdeathstate

```



```

sfx_podth2,          // deathsound
8,                  // speed
20*FRACUNIT,         // radius
56*FRACUNIT,         // height
100,                // mass
0,                  // damage
sfx_posact,          // activesound
MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
S_SPOS_RAISE1        // raisestate
},

{
    // MT_VILE
    64,              // doomednum
    S_VILE_STND,     // spawnstate
    700,             // spawnhealth
    S_VILE_RUN1,     // seestate
    sfx_vilsit,      // seesound
    8,               // reactiontime
    0,               // attacksound
    S_VILE_PAIN,     // painstate
    10,              // painchance
    sfx_vipain,      // painsound
    0,               // meleestate
    S_VILE_ATK1,     // missilestate
    S_VILE_DIE1,     // deathstate
    S_NULL,          // xdeathstate
    sfx_vildth,      // deathsound
    15,              // speed
    20*FRACUNIT,     // radius
    56*FRACUNIT,     // height
    500,             // mass
    0,               // damage
    sfx_vilact,      // activesound
    MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
    S_NULL           // raisestate
},

{
    // MT_FIRE
    -1,              // doomednum
    S_FIRE1,         // spawnstate
    1000,            // spawnhealth
    S_NULL,          // seestate
    sfx_None,        // seesound
    8,               // reactiontime
    sfx_None,        // attacksound
    S_NULL,          // painstate
    0,               // painchance
    sfx_None,        // painsound
    S_NULL,          // meleestate
    S_NULL,          // missilestate
    S_NULL,          // deathstate
    S_NULL,          // xdeathstate
    sfx_None,        // deathsound
    0,               // speed
    20*FRACUNIT,     // radius
    16*FRACUNIT,     // height
    100,             // mass
    0,               // damage
    sfx_None,        // activesound
    MF_NOBLOCKMAP|MF_NOGRAVITY, // flags
    S_NULL           // raisestate
},

{
    // MT_UNDEAD
    66,              // doomednum

```

```

S_SKEL_STND,          // spawnstate
300,                  // spawnhealth
S_SKEL_RUN1,          // seestate
sfx_skedit,          // seesound
8,                    // reactiontime
0,                    // attacksound
S_SKEL_PAIN,          // painstate
100,                  // painchance
sfx_popain,           // painsound
S_SKEL_FIST1,         // meleestate
S_SKEL_MISS1,         // missilestate
S_SKEL_DIE1,          // deathstate
S_NULL,              // xdeathstate
sfx_skedit,           // deathsound
10,                    // speed
20*FRACUNIT,          // radius
56*FRACUNIT,          // height
500,                  // mass
0,                    // damage
sfx_skeact,           // activesound
MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
S_SKEL_RAISE1        // raisestate
},

{ // MT_TRACER
-1,                    // doomednum
S_TRACER,              // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_skeatk,            // seesound
8,                    // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                    // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_TRACEEXP1,           // deathstate
S_NULL,                // xdeathstate
sfx_barexp,            // deathsound
10*FRACUNIT,           // speed
11*FRACUNIT,           // radius
8*FRACUNIT,            // height
100,                   // mass
10,                    // damage
sfx_None,              // activesound
MF_NOBLOCKMAP|MF_MISSILE|MF_DROPOFF|MF_NOGRAVITY, // flags
S_NULL                // raisestate
},

{ // MT_SMOKE
-1,                    // doomednum
S_SMOKE1,              // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                    // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                    // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate

```

```

    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                  // damage
    sfx_None,           // activesound
    MF_NOBLOCKMAP|MF_NOGRAVITY, // flags
    S_NULL              // raisestate
},

{
    // MT_FATSO
    67,                 // doomednum
    S_FATT_STND,        // spawnstate
    600,                // spawnhealth
    S_FATT_RUN1,        // seestate
    sfx_mansit,         // seesound
    8,                  // reactiontime
    0,                  // attacksound
    S_FATT_PAIN,        // painstate
    80,                 // painchance
    sfx_mnpain,         // painsound
    0,                  // meleestate
    S_FATT_ATK1,        // missilestate
    S_FATT_DIE1,        // deathstate
    S_NULL,             // xdeathstate
    sfx_mandth,         // deathsound
    8,                  // speed
    48*FRACUNIT,        // radius
    64*FRACUNIT,        // height
    1000,               // mass
    0,                  // damage
    sfx_posact,         // activesound
    MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
    S_FATT_RAISE1       // raisestate
},

{
    // MT_FATSHOT
    -1,                 // doomednum
    S_FATSHOT1,         // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_firsht,         // seesound
    8,                  // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                  // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_FATSHOTX1,        // deathstate
    S_NULL,             // xdeathstate
    sfx_firxpl,         // deathsound
    20*FRACUNIT,        // speed
    6*FRACUNIT,         // radius
    8*FRACUNIT,         // height
    100,                // mass
    8,                  // damage
    sfx_None,           // activesound
    MF_NOBLOCKMAP|MF_MISSILE|MF_DROPOFF|MF_NOGRAVITY, // flags
    S_NULL              // raisestate
},

{
    // MT_CHAINGUY
    65,                 // doomednum

```

```

S_CPOS_STND,          // spawnstate
70,                  // spawnhealth
S_CPOS_RUN1,          // seestate
sfx_posit2,          // seesound
8,                   // reactiontime
0,                   // attacksound
S_CPOS_PAIN,          // painstate
170,                 // painchance
sfx_popain,          // painsound
0,                   // meleestate
S_CPOS_ATK1,          // missilestate
S_CPOS_DIE1,          // deathstate
S_CPOS_XDIE1,         // xdeathstate
sfx_podth2,          // deathsound
8,                   // speed
20*FRACUNIT,         // radius
56*FRACUNIT,         // height
100,                 // mass
0,                   // damage
sfx_posact,          // activesound
MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
S_CPOS_RAISE1        // raisestate
},

{                    // MT_TROOP
3001,                // doomednum
S_TROO_STND,         // spawnstate
60,                  // spawnhealth
S_TROO_RUN1,         // seestate
sfx_bgsit1,          // seesound
8,                   // reactiontime
0,                   // attacksound
S_TROO_PAIN,         // painstate
200,                 // painchance
sfx_popain,          // painsound
S_TROO_ATK1,         // meleestate
S_TROO_ATK1,         // missilestate
S_TROO_DIE1,         // deathstate
S_TROO_XDIE1,        // xdeathstate
sfx_bgdth1,          // deathsound
8,                   // speed
20*FRACUNIT,         // radius
56*FRACUNIT,         // height
100,                 // mass
0,                   // damage
sfx_bgact,           // activesound
MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
S_TROO_RAISE1        // raisestate
},

{                    // MT_SERGEANT
3002,                // doomednum
S_SARG_STND,         // spawnstate
150,                 // spawnhealth
S_SARG_RUN1,         // seestate
sfx_sgtsit,          // seesound
8,                   // reactiontime
sfx_sgtatk,          // attacksound
S_SARG_PAIN,         // painstate
180,                 // painchance
sfx_dmpain,          // painsound
S_SARG_ATK1,         // meleestate
0,                   // missilestate
S_SARG_DIE1,         // deathstate
S_NULL,              // xdeathstate

```

```

sfx_sgtdth,          // deathsound
10,                  // speed
30*FRACUNIT,         // radius
56*FRACUNIT,         // height
400,                 // mass
0,                   // damage
sfx_dmact,           // activesound
MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
S_SARG_RAISE1        // raisestate
},

{
    // MT_SHADOWS
    58,                // doomednum
    S_SARG_STND,        // spawnstate
    150,               // spawnhealth
    S_SARG_RUN1,        // seestate
    sfx_sgtsit,        // seesound
    8,                 // reactiontime
    sfx_sgtatk,        // attacksound
    S_SARG_PAIN,        // painstate
    180,               // painchance
    sfx_dmpain,        // painsound
    S_SARG_ATK1,        // meleestate
    0,                 // missilestate
    S_SARG_DIE1,        // deathstate
    S_NULL,            // xdeathstate
    sfx_sgtdth,        // deathsound
    10,                // speed
    30*FRACUNIT,        // radius
    56*FRACUNIT,        // height
    400,               // mass
    0,                 // damage
    sfx_dmact,         // activesound
    MF_SOLID|MF_SHOOTABLE|MF_SHADOW|MF_COUNTKILL, // flags
    S_SARG_RAISE1      // raisestate
},

{
    // MT_HEAD
    3005,              // doomednum
    S_HEAD_STND,        // spawnstate
    400,               // spawnhealth
    S_HEAD_RUN1,        // seestate
    sfx_cacsit,        // seesound
    8,                 // reactiontime
    0,                 // attacksound
    S_HEAD_PAIN,        // painstate
    128,               // painchance
    sfx_dmpain,        // painsound
    0,                 // meleestate
    S_HEAD_ATK1,        // missilestate
    S_HEAD_DIE1,        // deathstate
    S_NULL,            // xdeathstate
    sfx_cacdth,        // deathsound
    8,                 // speed
    31*FRACUNIT,        // radius
    56*FRACUNIT,        // height
    400,               // mass
    0,                 // damage
    sfx_dmact,         // activesound
    MF_SOLID|MF_SHOOTABLE|MF_FLOAT|MF_NOGRAVITY|MF_COUNTKILL, // flags
    S_HEAD_RAISE1      // raisestate
},

{
    // MT_BRUISER
    3003,              // doomednum

```

```

S_BOSS_STND,          // spawnstate
1000,                 // spawnhealth
S_BOSS_RUN1,          // seestate
sfx_brssit,           // seesound
8,                    // reactiontime
0,                    // attacksound
S_BOSS_PAIN,          // painstate
50,                   // painchance
sfx_dmpain,           // painsound
S_BOSS_ATK1,          // meleestate
S_BOSS_ATK1,          // missilestate
S_BOSS_DIE1,          // deathstate
S_NULL,               // xdeathstate
sfx_brsdth,           // deathsound
8,                    // speed
24*FRACUNIT,          // radius
64*FRACUNIT,          // height
1000,                 // mass
0,                    // damage
sfx_dmact,            // activesound
MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
S_BOSS_RAISE1         // raisestate
},

{                      // MT_BRUISERSHOT
-1,                   // doomednum
S_BREBALL1,           // spawnstate
1000,                 // spawnhealth
S_NULL,               // seestate
sfx_firsht,           // seesound
8,                    // reactiontime
sfx_None,             // attacksound
S_NULL,               // painstate
0,                    // painchance
sfx_None,             // painsound
S_NULL,               // meleestate
S_NULL,               // missilestate
S_BREBALLX1,          // deathstate
S_NULL,               // xdeathstate
sfx_firxpl,           // deathsound
15*FRACUNIT,          // speed
6*FRACUNIT,           // radius
8*FRACUNIT,           // height
100,                  // mass
8,                    // damage
sfx_None,             // activesound
MF_NOBLOCKMAP|MF_MISSILE|MF_DROPOFF|MF_NOGRAVITY, // flags
S_NULL                // raisestate
},

{                      // MT_KNIGHT
69,                   // doomednum
S_BOS2_STND,          // spawnstate
500,                  // spawnhealth
S_BOS2_RUN1,          // seestate
sfx_kntsit,           // seesound
8,                    // reactiontime
0,                    // attacksound
S_BOS2_PAIN,          // painstate
50,                   // painchance
sfx_dmpain,           // painsound
S_BOS2_ATK1,          // meleestate
S_BOS2_ATK1,          // missilestate
S_BOS2_DIE1,          // deathstate
S_NULL,               // xdeathstate

```

```

sfx_kntdth,          // deathsound
8,                  // speed
24*FRACUNIT,         // radius
64*FRACUNIT,         // height
1000,               // mass
0,                  // damage
sfx_dmact,           // activesound
MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
S_BOS2_RAISE1        // raisestate
},

{                  // MT_SKULL
3006,              // doomednum
S_SKULL_STND,      // spawnstate
100,               // spawnhealth
S_SKULL_RUN1,      // seestate
0,                // seesound
8,                // reactiontime
sfx_sklatk,        // attacksound
S_SKULL_PAIN,      // painstate
256,              // painchance
sfx_dmpain,        // painsound
0,                // meleestate
S_SKULL_ATK1,      // missilestate
S_SKULL_DIE1,      // deathstate
S_NULL,           // xdeathstate
sfx_firxpl,        // deathsound
8,                // speed
16*FRACUNIT,       // radius
56*FRACUNIT,       // height
50,               // mass
3,                // damage
sfx_dmact,         // activesound
MF_SOLID|MF_SHOOTABLE|MF_FLOAT|MF_NOGRAVITY, // flags
S_NULL            // raisestate
},

{                  // MT_SPIDER
7,                // doomednum
S_SPID_STND,       // spawnstate
3000,              // spawnhealth
S_SPID_RUN1,       // seestate
sfx_spisit,        // seesound
8,                // reactiontime
sfx_shotgn,        // attacksound
S_SPID_PAIN,       // painstate
40,               // painchance
sfx_dmpain,        // painsound
0,                // meleestate
S_SPID_ATK1,       // missilestate
S_SPID_DIE1,       // deathstate
S_NULL,           // xdeathstate
sfx_spidth,        // deathsound
12,               // speed
128*FRACUNIT,      // radius
100*FRACUNIT,      // height
1000,             // mass
0,                // damage
sfx_dmact,         // activesound
MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
S_NULL            // raisestate
},

{                  // MT_BABY
68,               // doomednum

```

```

S_BSPI_STND,          // spawnstate
500,                  // spawnhealth
S_BSPI_SIGHT,         // seestate
sfx_bspst,           // seesound
8,                    // reactiontime
0,                    // attacksound
S_BSPI_PAIN,          // painstate
128,                  // painchance
sfx_dmpain,          // painsound
0,                    // meleestate
S_BSPI_ATK1,          // missilestate
S_BSPI_DIE1,          // deathstate
S_NULL,              // xdeathstate
sfx_bspdth,          // deathsound
12,                    // speed
64*FRACUNIT,          // radius
64*FRACUNIT,          // height
600,                  // mass
0,                    // damage
sfx_bspact,          // activesound
MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
S_BSPI_RAISE1        // raisestate
},

{                      // MT_CYBORG
16,                    // doomednum
S_CYBER_STND,          // spawnstate
4000,                  // spawnhealth
S_CYBER_RUN1,          // seestate
sfx_cybsit,           // seesound
8,                    // reactiontime
0,                    // attacksound
S_CYBER_PAIN,          // painstate
20,                    // painchance
sfx_dmpain,          // painsound
0,                    // meleestate
S_CYBER_ATK1,          // missilestate
S_CYBER_DIE1,          // deathstate
S_NULL,              // xdeathstate
sfx_cybdth,          // deathsound
16,                    // speed
40*FRACUNIT,          // radius
110*FRACUNIT,          // height
1000,                  // mass
0,                    // damage
sfx_dmact,            // activesound
MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
S_NULL                // raisestate
},

{                      // MT_PAIN
71,                    // doomednum
S_PAIN_STND,          // spawnstate
400,                  // spawnhealth
S_PAIN_RUN1,          // seestate
sfx_pesit,           // seesound
8,                    // reactiontime
0,                    // attacksound
S_PAIN_PAIN,          // painstate
128,                  // painchance
sfx_pepain,          // painsound
0,                    // meleestate
S_PAIN_ATK1,          // missilestate
S_PAIN_DIE1,          // deathstate
S_NULL,              // xdeathstate

```



```

sfx_pedth,          // deathsound
8,                  // speed
31*FRACUNIT,        // radius
56*FRACUNIT,        // height
400,                // mass
0,                  // damage
sfx_dmact,          // activesound
MF_SOLID|MF_SHOOTABLE|MF_FLOAT|MF_NOGRAVITY|MF_COUNTKILL, // flags
S_PAIN_RAISE1       // raisestate
},

{
    // MT_WOLFSS
    84,              // doomednum
    S_SSWV_STND,     // spawnstate
    50,              // spawnhealth
    S_SSWV_RUN1,     // seestate
    sfx_sssit,       // seesound
    8,               // reactiontime
    0,               // attacksound
    S_SSWV_PAIN,     // painstate
    170,             // painchance
    sfx_popain,      // painsound
    0,               // meleestate
    S_SSWV_ATK1,     // missilestate
    S_SSWV_DIE1,     // deathstate
    S_SSWV_XDIE1,    // xdeathstate
    sfx_ssdth,       // deathsound
    8,               // speed
    20*FRACUNIT,     // radius
    56*FRACUNIT,     // height
    100,             // mass
    0,               // damage
    sfx_posact,      // activesound
    MF_SOLID|MF_SHOOTABLE|MF_COUNTKILL, // flags
    S_SSWV_RAISE1    // raisestate
},

{
    // MT_KEEN
    72,              // doomednum
    S_KEENSTND,     // spawnstate
    100,             // spawnhealth
    S_NULL,          // seestate
    sfx_None,        // seesound
    8,               // reactiontime
    sfx_None,        // attacksound
    S_KEENPAIN,     // painstate
    256,             // painchance
    sfx_keenpn,     // painsound
    S_NULL,          // meleestate
    S_NULL,          // missilestate
    S_COMMKEEN,     // deathstate
    S_NULL,          // xdeathstate
    sfx_keendt,     // deathsound
    0,               // speed
    16*FRACUNIT,     // radius
    72*FRACUNIT,     // height
    10000000,        // mass
    0,               // damage
    sfx_None,        // activesound
    MF_SOLID|MF_SPAWNCEILING|MF_NOGRAVITY|MF_SHOOTABLE|MF_COUNTKILL, // flags
    S_NULL           // raisestate
},

{
    // MT_BOSSBRAIN
    88,              // doomednum

```

```

S_BRAIN,          // spawnstate
250,              // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_BRAIN_PAIN,    // painstate
255,            // painchance
sfx_bospn,       // painsound
S_NULL,         // meleestate
S_NULL,         // missilestate
S_BRAIN_DIE1,    // deathstate
S_NULL,         // xdeathstate
sfx_bosdth,      // deathsound
0,              // speed
16*FRACUNIT,     // radius
16*FRACUNIT,     // height
10000000,        // mass
0,              // damage
sfx_None,        // activesound
MF_SOLID|MF_SHOOTABLE, // flags
S_NULL          // raisestate
},

{                // MT_BOSSSPIT
89,             // doomednum
S_BRAINEYE,     // spawnstate
1000,           // spawnhealth
S_BRAINEYESEE,  // seestate
sfx_None,       // seesound
8,             // reactiontime
sfx_None,       // attacksound
S_NULL,        // painstate
0,            // painchance
sfx_None,      // painsound
S_NULL,       // meleestate
S_NULL,       // missilestate
S_NULL,       // deathstate
S_NULL,       // xdeathstate
sfx_None,     // deathsound
0,           // speed
20*FRACUNIT, // radius
32*FRACUNIT, // height
100,         // mass
0,          // damage
sfx_None,   // activesound
MF_NOBLOCKMAP|MF_NOSECTOR, // flags
S_NULL     // raisestate
},

{                // MT_BOSSTARGET
87,             // doomednum
S_NULL,         // spawnstate
1000,           // spawnhealth
S_NULL,         // seestate
sfx_None,       // seesound
8,             // reactiontime
sfx_None,       // attacksound
S_NULL,        // painstate
0,            // painchance
sfx_None,      // painsound
S_NULL,       // meleestate
S_NULL,       // missilestate
S_NULL,       // deathstate
S_NULL,       // xdeathstate

```

```

    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    32*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_NOBLOCKMAP|MF_NOSECTOR, // flags
    S_NULL             // raisestate
},

{
    // MT_SPAWNSHOT
    -1,                // doomednum
    S_SPAWN1,          // spawnstate
    1000,              // spawnhealth
    S_NULL,            // seestate
    sfx_bospit,        // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,            // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,            // meleestate
    S_NULL,            // missilestate
    S_NULL,            // deathstate
    S_NULL,            // xdeathstate
    sfx_firxpl,        // deathsound
    10*FRACUNIT,       // speed
    6*FRACUNIT,        // radius
    32*FRACUNIT,       // height
    100,               // mass
    3,                 // damage
    sfx_None,          // activesound
    MF_NOBLOCKMAP|MF_MISSILE|MF_DROPOFF|MF_NOGRAVITY|MF_NOCLIP, // flags
    S_NULL             // raisestate
},

{
    // MT_SPAWNFIRE
    -1,                // doomednum
    S_SPAWNFIRE1,      // spawnstate
    1000,              // spawnhealth
    S_NULL,            // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,            // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,            // meleestate
    S_NULL,            // missilestate
    S_NULL,            // deathstate
    S_NULL,            // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_NOBLOCKMAP|MF_NOGRAVITY, // flags
    S_NULL             // raisestate
},

{
    // MT_BARREL
    2035,              // doomednum

```

```

S_BAR1,          // spawnstate
20,              // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_NULL,          // painstate
0,              // painchance
sfx_None,        // painsound
S_NULL,          // meleestate
S_NULL,          // missilestate
S_BEXP,          // deathstate
S_NULL,          // xdeathstate
sfx_barexp,      // deathsound
0,              // speed
10*FRACUNIT,     // radius
42*FRACUNIT,     // height
100,             // mass
0,              // damage
sfx_None,        // activesound
MF_SOLID|MF_SHOOTABLE|MF_NOBLOOD, // flags
S_NULL          // raisestate
},

{                // MT_TROOPSHOT
-1,             // doomednum
S_TBALL1,       // spawnstate
1000,           // spawnhealth
S_NULL,         // seestate
sfx_firsht,     // seesound
8,             // reactiontime
sfx_None,       // attacksound
S_NULL,         // painstate
0,             // painchance
sfx_None,       // painsound
S_NULL,         // meleestate
S_NULL,         // missilestate
S_TBALLX1,      // deathstate
S_NULL,         // xdeathstate
sfx_firxpl,     // deathsound
10*FRACUNIT,    // speed
6*FRACUNIT,     // radius
8*FRACUNIT,     // height
100,            // mass
3,             // damage
sfx_None,       // activesound
MF_NOBLOCKMAP|MF_MISSILE|MF_DROPOFF|MF_NOGRAVITY, // flags
S_NULL          // raisestate
},

{                // MT_HEADSHOT
-1,             // doomednum
S_RBALL1,       // spawnstate
1000,           // spawnhealth
S_NULL,         // seestate
sfx_firsht,     // seesound
8,             // reactiontime
sfx_None,       // attacksound
S_NULL,         // painstate
0,             // painchance
sfx_None,       // painsound
S_NULL,         // meleestate
S_NULL,         // missilestate
S_RBALLX1,      // deathstate
S_NULL,         // xdeathstate

```

```

sfx_firxpl,          // deathsound
10*FRACUNIT,         // speed
6*FRACUNIT,          // radius
8*FRACUNIT,          // height
100,                 // mass
5,                   // damage
sfx_None,            // activesound
MF_NOBLOCKMAP|MF_MISSILE|MF_DROPOFF|MF_NOGRAVITY,           // flags
S_NULL               // raisestate
},

{
    // MT_ROCKET
    -1,              // doomednum
    S_ROCKET,        // spawnstate
    1000,            // spawnhealth
    S_NULL,          // seestate
    sfx_rlaunc,      // seesound
    8,               // reactiontime
    sfx_None,        // attacksound
    S_NULL,          // painstate
    0,               // painchance
    sfx_None,        // painsound
    S_NULL,          // meleestate
    S_NULL,          // missilestate
    S_EXPLODE1,      // deathstate
    S_NULL,          // xdeathstate
    sfx_barexp,      // deathsound
    20*FRACUNIT,     // speed
    11*FRACUNIT,     // radius
    8*FRACUNIT,      // height
    100,             // mass
    20,              // damage
    sfx_None,        // activesound
    MF_NOBLOCKMAP|MF_MISSILE|MF_DROPOFF|MF_NOGRAVITY,           // flags
    S_NULL           // raisestate
},

{
    // MT_PLASMA
    -1,              // doomednum
    S_PLASBALL,      // spawnstate
    1000,            // spawnhealth
    S_NULL,          // seestate
    sfx_plasma,      // seesound
    8,               // reactiontime
    sfx_None,        // attacksound
    S_NULL,          // painstate
    0,               // painchance
    sfx_None,        // painsound
    S_NULL,          // meleestate
    S_NULL,          // missilestate
    S_PLASEXP,       // deathstate
    S_NULL,          // xdeathstate
    sfx_firxpl,      // deathsound
    25*FRACUNIT,     // speed
    13*FRACUNIT,     // radius
    8*FRACUNIT,      // height
    100,             // mass
    5,               // damage
    sfx_None,        // activesound
    MF_NOBLOCKMAP|MF_MISSILE|MF_DROPOFF|MF_NOGRAVITY,           // flags
    S_NULL           // raisestate
},

{
    // MT_BFG
    -1,              // doomednum

```

```

S_BFGSHOT,          // spawnstate
1000,                // spawnhealth
S_NULL,              // seestate
0,                   // seesound
8,                   // reactiontime
sfx_None,             // attacksound
S_NULL,              // painstate
0,                   // painchance
sfx_None,             // painsound
S_NULL,              // meleestate
S_NULL,              // missilestate
S_BFGGLAND,          // deathstate
S_NULL,              // xdeathstate
sfx_rxplod,          // deathsound
25*FRACUNIT,         // speed
13*FRACUNIT,         // radius
8*FRACUNIT,          // height
100,                 // mass
100,                 // damage
sfx_None,             // activesound
MF_NOBLOCKMAP|MF_MISSILE|MF_DROPOFF|MF_NOGRAVITY, // flags
S_NULL               // raisestate
},

{                    // MT_ARACHPLAZ
-1,                  // doomednum
S_ARACH_PLAZ,        // spawnstate
1000,                // spawnhealth
S_NULL,              // seestate
sfx_plasma,          // seesound
8,                   // reactiontime
sfx_None,             // attacksound
S_NULL,              // painstate
0,                   // painchance
sfx_None,             // painsound
S_NULL,              // meleestate
S_NULL,              // missilestate
S_ARACH_PLEX,        // deathstate
S_NULL,              // xdeathstate
sfx_firxpl,          // deathsound
25*FRACUNIT,         // speed
13*FRACUNIT,         // radius
8*FRACUNIT,          // height
100,                 // mass
5,                   // damage
sfx_None,             // activesound
MF_NOBLOCKMAP|MF_MISSILE|MF_DROPOFF|MF_NOGRAVITY, // flags
S_NULL               // raisestate
},

{                    // MT_PUFF
-1,                  // doomednum
S_PUFF1,              // spawnstate
1000,                // spawnhealth
S_NULL,              // seestate
sfx_None,             // seesound
8,                   // reactiontime
sfx_None,             // attacksound
S_NULL,              // painstate
0,                   // painchance
sfx_None,             // painsound
S_NULL,              // meleestate
S_NULL,              // missilestate
S_NULL,              // deathstate
S_NULL,              // xdeathstate

```

```

    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_NOBLOCKMAP|MF_NOGRAVITY, // flags
    S_NULL             // raisestate
},

{
    // MT_BLOOD
    -1,                // doomednum
    S_BLOOD1,          // spawnstate
    1000,              // spawnhealth
    S_NULL,            // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,            // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,            // meleestate
    S_NULL,            // missilestate
    S_NULL,            // deathstate
    S_NULL,            // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_NOBLOCKMAP,     // flags
    S_NULL             // raisestate
},

{
    // MT_TFOG
    -1,                // doomednum
    S_TFOG,            // spawnstate
    1000,              // spawnhealth
    S_NULL,            // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,            // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,            // meleestate
    S_NULL,            // missilestate
    S_NULL,            // deathstate
    S_NULL,            // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_NOBLOCKMAP|MF_NOGRAVITY, // flags
    S_NULL             // raisestate
},

{
    // MT_IFOG
    -1,                // doomednum

```

```

S_IFOG,           // spawnstate
1000,             // spawnhealth
S_NULL,           // seestate
sfx_None,         // seesound
8,               // reactiontime
sfx_None,         // attacksound
S_NULL,           // painstate
0,               // painchance
sfx_None,         // painsound
S_NULL,           // meleestate
S_NULL,           // missilestate
S_NULL,           // deathstate
S_NULL,           // xdeathstate
sfx_None,         // deathsound
0,               // speed
20*FRACUNIT,      // radius
16*FRACUNIT,      // height
100,              // mass
0,               // damage
sfx_None,         // activesound
MF_NOBLOCKMAP|MF_NOGRAVITY, // flags
S_NULL           // raisestate
},

{                // MT_TELEPORTMAN
14,              // doomednum
S_NULL,          // spawnstate
1000,            // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_NULL,          // painstate
0,              // painchance
sfx_None,        // painsound
S_NULL,          // meleestate
S_NULL,          // missilestate
S_NULL,          // deathstate
S_NULL,          // xdeathstate
sfx_None,        // deathsound
0,              // speed
20*FRACUNIT,     // radius
16*FRACUNIT,     // height
100,             // mass
0,              // damage
sfx_None,        // activesound
MF_NOBLOCKMAP|MF_NOSECTOR, // flags
S_NULL           // raisestate
},

{                // MT_EXTRABFG
-1,              // doomednum
S_BFGEXP,        // spawnstate
1000,            // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_NULL,          // painstate
0,              // painchance
sfx_None,        // painsound
S_NULL,          // meleestate
S_NULL,          // missilestate
S_NULL,          // deathstate
S_NULL,          // xdeathstate

```



```

    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_NOBLOCKMAP|MF_NOGRAVITY, // flags
    S_NULL             // raisestate
},

{
    // MT_MISCO
    2018,              // doomednum
    S_ARM1,             // spawnstate
    1000,              // spawnhealth
    S_NULL,             // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,             // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SPECIAL,        // flags
    S_NULL             // raisestate
},

{
    // MT_MISC1
    2019,              // doomednum
    S_ARM2,             // spawnstate
    1000,              // spawnhealth
    S_NULL,             // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,             // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SPECIAL,        // flags
    S_NULL             // raisestate
},

{
    // MT_MISC2
    2014,              // doomednum

```

```

S_BON1,          // spawnstate
1000,            // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_NULL,          // painstate
0,              // painchance
sfx_None,        // painsound
S_NULL,          // meleestate
S_NULL,          // missilestate
S_NULL,          // deathstate
S_NULL,          // xdeathstate
sfx_None,        // deathsound
0,              // speed
20*FRACUNIT,     // radius
16*FRACUNIT,     // height
100,             // mass
0,              // damage
sfx_None,        // activesound
MF_SPECIAL|MF_COUNTITEM, // flags
S_NULL          // raisestate
},

{                // MT_MISC3
2015,           // doomednum
S_BON2,         // spawnstate
1000,           // spawnhealth
S_NULL,         // seestate
sfx_None,       // seesound
8,             // reactiontime
sfx_None,       // attacksound
S_NULL,         // painstate
0,             // painchance
sfx_None,       // painsound
S_NULL,         // meleestate
S_NULL,         // missilestate
S_NULL,         // deathstate
S_NULL,         // xdeathstate
sfx_None,       // deathsound
0,             // speed
20*FRACUNIT,    // radius
16*FRACUNIT,    // height
100,            // mass
0,             // damage
sfx_None,       // activesound
MF_SPECIAL|MF_COUNTITEM, // flags
S_NULL         // raisestate
},

{                // MT_MISC4
5,              // doomednum
S_BKEY,         // spawnstate
1000,           // spawnhealth
S_NULL,         // seestate
sfx_None,       // seesound
8,             // reactiontime
sfx_None,       // attacksound
S_NULL,         // painstate
0,             // painchance
sfx_None,       // painsound
S_NULL,         // meleestate
S_NULL,         // missilestate
S_NULL,         // deathstate
S_NULL,         // xdeathstate

```

```

    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SPECIAL|MF_NOTDMATCH, // flags
    S_NULL             // raisestate
},

{
    // MT_MISC5
    13,                // doomednum
    S_RKEY,             // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_None,           // seesound
    8,                  // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                  // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                  // damage
    sfx_None,           // activesound
    MF_SPECIAL|MF_NOTDMATCH, // flags
    S_NULL             // raisestate
},

{
    // MT_MISC6
    6,                  // doomednum
    S_YKEY,             // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_None,           // seesound
    8,                  // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                  // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                  // damage
    sfx_None,           // activesound
    MF_SPECIAL|MF_NOTDMATCH, // flags
    S_NULL             // raisestate
},

{
    // MT_MISC7
    39,                 // doomednum

```

```

S_YSKULL,          // spawnstate
1000,              // spawnhealth
S_NULL,            // seestate
sfx_None,          // seesound
8,                // reactiontime
sfx_None,          // attacksound
S_NULL,            // painstate
0,                // painchance
sfx_None,          // painsound
S_NULL,            // meleestate
S_NULL,            // missilestate
S_NULL,            // deathstate
S_NULL,            // xdeathstate
sfx_None,          // deathsound
0,                // speed
20*FRACUNIT,       // radius
16*FRACUNIT,       // height
100,               // mass
0,                // damage
sfx_None,          // activesound
MF_SPECIAL|MF_NOTDMATCH, // flags
S_NULL            // raisestate
},

{                  // MT_MISC8
38,               // doomednum
S_RSKULL,         // spawnstate
1000,             // spawnhealth
S_NULL,           // seestate
sfx_None,         // seesound
8,               // reactiontime
sfx_None,         // attacksound
S_NULL,           // painstate
0,               // painchance
sfx_None,         // painsound
S_NULL,           // meleestate
S_NULL,           // missilestate
S_NULL,           // deathstate
S_NULL,           // xdeathstate
sfx_None,         // deathsound
0,               // speed
20*FRACUNIT,     // radius
16*FRACUNIT,     // height
100,             // mass
0,               // damage
sfx_None,         // activesound
MF_SPECIAL|MF_NOTDMATCH, // flags
S_NULL           // raisestate
},

{                  // MT_MISC9
40,               // doomednum
S_BSKULL,         // spawnstate
1000,             // spawnhealth
S_NULL,           // seestate
sfx_None,         // seesound
8,               // reactiontime
sfx_None,         // attacksound
S_NULL,           // painstate
0,               // painchance
sfx_None,         // painsound
S_NULL,           // meleestate
S_NULL,           // missilestate
S_NULL,           // deathstate
S_NULL,           // xdeathstate

```

```

sfx_None,          // deathsound
0,                // speed
20*FRACUNIT,      // radius
16*FRACUNIT,      // height
100,              // mass
0,                // damage
sfx_None,          // activesound
MF_SPECIAL|MF_NOTDMATCH, // flags
S_NULL            // raisestate
},

{
    // MT_MISC10
    2011,          // doomednum
    S_STIM,        // spawnstate
    1000,          // spawnhealth
    S_NULL,        // seestate
    sfx_None,      // seesound
    8,             // reactiontime
    sfx_None,      // attacksound
    S_NULL,        // painstate
    0,             // painchance
    sfx_None,      // painsound
    S_NULL,        // meleestate
    S_NULL,        // missilestate
    S_NULL,        // deathstate
    S_NULL,        // xdeathstate
    sfx_None,      // deathsound
    0,             // speed
    20*FRACUNIT,   // radius
    16*FRACUNIT,   // height
    100,           // mass
    0,             // damage
    sfx_None,      // activesound
    MF_SPECIAL,    // flags
    S_NULL         // raisestate
},

{
    // MT_MISC11
    2012,          // doomednum
    S_MEDI,        // spawnstate
    1000,          // spawnhealth
    S_NULL,        // seestate
    sfx_None,      // seesound
    8,             // reactiontime
    sfx_None,      // attacksound
    S_NULL,        // painstate
    0,             // painchance
    sfx_None,      // painsound
    S_NULL,        // meleestate
    S_NULL,        // missilestate
    S_NULL,        // deathstate
    S_NULL,        // xdeathstate
    sfx_None,      // deathsound
    0,             // speed
    20*FRACUNIT,   // radius
    16*FRACUNIT,   // height
    100,           // mass
    0,             // damage
    sfx_None,      // activesound
    MF_SPECIAL,    // flags
    S_NULL         // raisestate
},

{
    // MT_MISC12
    2013,          // doomednum

```

```

S_SOUL,           // spawnstate
1000,             // spawnhealth
S_NULL,           // seestate
sfx_None,         // seesound
8,               // reactiontime
sfx_None,         // attacksound
S_NULL,           // painstate
0,               // painchance
sfx_None,         // painsound
S_NULL,           // meleestate
S_NULL,           // missilestate
S_NULL,           // deathstate
S_NULL,           // xdeathstate
sfx_None,         // deathsound
0,               // speed
20*FRACUNIT,      // radius
16*FRACUNIT,      // height
100,              // mass
0,               // damage
sfx_None,         // activesound
MF_SPECIAL|MF_COUNTITEM, // flags
S_NULL           // raisestate
},

{                // MT_INV
2022,            // doomednum
S_PINV,          // spawnstate
1000,            // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_NULL,          // painstate
0,              // painchance
sfx_None,        // painsound
S_NULL,          // meleestate
S_NULL,          // missilestate
S_NULL,          // deathstate
S_NULL,          // xdeathstate
sfx_None,        // deathsound
0,              // speed
20*FRACUNIT,     // radius
16*FRACUNIT,     // height
100,             // mass
0,              // damage
sfx_None,        // activesound
MF_SPECIAL|MF_COUNTITEM, // flags
S_NULL           // raisestate
},

{                // MT_MISC13
2023,            // doomednum
S_PSTR,          // spawnstate
1000,            // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_NULL,          // painstate
0,              // painchance
sfx_None,        // painsound
S_NULL,          // meleestate
S_NULL,          // missilestate
S_NULL,          // deathstate
S_NULL,          // xdeathstate

```

```

    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SPECIAL|MF_COUNTITEM, // flags
    S_NULL             // raisestate
},

{
    // MT_INS
    2024,              // doomednum
    S_PINS,            // spawnstate
    1000,              // spawnhealth
    S_NULL,            // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,            // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,            // meleestate
    S_NULL,            // missilestate
    S_NULL,            // deathstate
    S_NULL,            // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SPECIAL|MF_COUNTITEM, // flags
    S_NULL             // raisestate
},

{
    // MT_MISC14
    2025,              // doomednum
    S_SUIT,            // spawnstate
    1000,              // spawnhealth
    S_NULL,            // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,            // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,            // meleestate
    S_NULL,            // missilestate
    S_NULL,            // deathstate
    S_NULL,            // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SPECIAL,        // flags
    S_NULL             // raisestate
},

{
    // MT_MISC15
    2026,              // doomednum

```

```

S_PMAP,          // spawnstate
1000,            // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_NULL,          // painstate
0,              // painchance
sfx_None,        // painsound
S_NULL,          // meleestate
S_NULL,          // missilestate
S_NULL,          // deathstate
S_NULL,          // xdeathstate
sfx_None,        // deathsound
0,              // speed
20*FRACUNIT,     // radius
16*FRACUNIT,     // height
100,            // mass
0,              // damage
sfx_None,        // activesound
MF_SPECIAL|MF_COUNTITEM, // flags
S_NULL          // raisestate
},

{                // MT_MISC16
2045,           // doomednum
S_PVIS,         // spawnstate
1000,          // spawnhealth
S_NULL,         // seestate
sfx_None,       // seesound
8,             // reactiontime
sfx_None,       // attacksound
S_NULL,         // painstate
0,             // painchance
sfx_None,       // painsound
S_NULL,         // meleestate
S_NULL,         // missilestate
S_NULL,         // deathstate
S_NULL,         // xdeathstate
sfx_None,       // deathsound
0,             // speed
20*FRACUNIT,    // radius
16*FRACUNIT,    // height
100,           // mass
0,             // damage
sfx_None,       // activesound
MF_SPECIAL|MF_COUNTITEM, // flags
S_NULL         // raisestate
},

{                // MT_MEGA
83,            // doomednum
S_MEGA,        // spawnstate
1000,          // spawnhealth
S_NULL,        // seestate
sfx_None,      // seesound
8,            // reactiontime
sfx_None,      // attacksound
S_NULL,        // painstate
0,            // painchance
sfx_None,      // painsound
S_NULL,        // meleestate
S_NULL,        // missilestate
S_NULL,        // deathstate
S_NULL,        // xdeathstate

```



```

    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SPECIAL|MF_COUNTITEM, // flags
    S_NULL             // raisestate
},

{
    // MT_CLIP
    2007,              // doomednum
    S_CLIP,            // spawnstate
    1000,              // spawnhealth
    S_NULL,            // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,            // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,            // meleestate
    S_NULL,            // missilestate
    S_NULL,            // deathstate
    S_NULL,            // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SPECIAL,        // flags
    S_NULL             // raisestate
},

{
    // MT_MISC17
    2048,              // doomednum
    S_AMMO,            // spawnstate
    1000,              // spawnhealth
    S_NULL,            // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,            // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,            // meleestate
    S_NULL,            // missilestate
    S_NULL,            // deathstate
    S_NULL,            // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SPECIAL,        // flags
    S_NULL             // raisestate
},

{
    // MT_MISC18
    2010,              // doomednum

```

```

S_ROCK,           // spawnstate
1000,             // spawnhealth
S_NULL,           // seestate
sfx_None,         // seesound
8,               // reactiontime
sfx_None,         // attacksound
S_NULL,           // painstate
0,               // painchance
sfx_None,         // painsound
S_NULL,           // meleestate
S_NULL,           // missilestate
S_NULL,           // deathstate
S_NULL,           // xdeathstate
sfx_None,         // deathsound
0,               // speed
20*FRACUNIT,      // radius
16*FRACUNIT,      // height
100,              // mass
0,               // damage
sfx_None,         // activesound
MF_SPECIAL,       // flags
S_NULL           // raisestate
},

{                // MT_MISC19
2046,            // doomednum
S_BROK,          // spawnstate
1000,            // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_NULL,          // painstate
0,              // painchance
sfx_None,        // painsound
S_NULL,          // meleestate
S_NULL,          // missilestate
S_NULL,          // deathstate
S_NULL,          // xdeathstate
sfx_None,        // deathsound
0,              // speed
20*FRACUNIT,     // radius
16*FRACUNIT,     // height
100,             // mass
0,              // damage
sfx_None,        // activesound
MF_SPECIAL,      // flags
S_NULL           // raisestate
},

{                // MT_MISC20
2047,            // doomednum
S_CELL,          // spawnstate
1000,            // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_NULL,          // painstate
0,              // painchance
sfx_None,        // painsound
S_NULL,          // meleestate
S_NULL,          // missilestate
S_NULL,          // deathstate
S_NULL,          // xdeathstate

```

```

    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                  // damage
    sfx_None,           // activesound
    MF_SPECIAL,         // flags
    S_NULL              // raisestate
},

{                      // MT_MISC21
    17,                // doomednum
    S_CELP,            // spawnstate
    1000,              // spawnhealth
    S_NULL,            // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,            // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,            // meleestate
    S_NULL,            // missilestate
    S_NULL,            // deathstate
    S_NULL,            // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SPECIAL,         // flags
    S_NULL              // raisestate
},

{                      // MT_MISC22
    2008,              // doomednum
    S_SHEL,            // spawnstate
    1000,              // spawnhealth
    S_NULL,            // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,            // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,            // meleestate
    S_NULL,            // missilestate
    S_NULL,            // deathstate
    S_NULL,            // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SPECIAL,         // flags
    S_NULL              // raisestate
},

{                      // MT_MISC23
    2049,              // doomednum

```

```

S_SBOX,          // spawnstate
1000,            // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_NULL,          // painstate
0,              // painchance
sfx_None,        // painsound
S_NULL,          // meleestate
S_NULL,          // missilestate
S_NULL,          // deathstate
S_NULL,          // xdeathstate
sfx_None,        // deathsound
0,              // speed
20*FRACUNIT,     // radius
16*FRACUNIT,     // height
100,             // mass
0,              // damage
sfx_None,        // activesound
MF_SPECIAL,      // flags
S_NULL           // raisestate
},

{                // MT_MISC24
8,              // doomednum
S_BPAK,         // spawnstate
1000,           // spawnhealth
S_NULL,         // seestate
sfx_None,       // seesound
8,              // reactiontime
sfx_None,       // attacksound
S_NULL,         // painstate
0,              // painchance
sfx_None,       // painsound
S_NULL,         // meleestate
S_NULL,         // missilestate
S_NULL,         // deathstate
S_NULL,         // xdeathstate
sfx_None,       // deathsound
0,              // speed
20*FRACUNIT,    // radius
16*FRACUNIT,    // height
100,            // mass
0,              // damage
sfx_None,       // activesound
MF_SPECIAL,     // flags
S_NULL          // raisestate
},

{                // MT_MISC25
2006,           // doomednum
S_BFUG,         // spawnstate
1000,           // spawnhealth
S_NULL,         // seestate
sfx_None,       // seesound
8,              // reactiontime
sfx_None,       // attacksound
S_NULL,         // painstate
0,              // painchance
sfx_None,       // painsound
S_NULL,         // meleestate
S_NULL,         // missilestate
S_NULL,         // deathstate
S_NULL,         // xdeathstate

```

```

    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                  // damage
    sfx_None,           // activesound
    MF_SPECIAL,         // flags
    S_NULL              // raisestate
},

{                      // MT_CHAINGUN
    2002,               // doomednum
    S_MGUN,             // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_None,           // seesound
    8,                  // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                  // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                  // damage
    sfx_None,           // activesound
    MF_SPECIAL,         // flags
    S_NULL              // raisestate
},

{                      // MT_MISC26
    2005,               // doomednum
    S_CSAW,             // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_None,           // seesound
    8,                  // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                  // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                  // damage
    sfx_None,           // activesound
    MF_SPECIAL,         // flags
    S_NULL              // raisestate
},

{                      // MT_MISC27
    2003,               // doomednum

```

```

S_LAUN,           // spawnstate
1000,             // spawnhealth
S_NULL,           // seestate
sfx_None,         // seesound
8,               // reactiontime
sfx_None,         // attacksound
S_NULL,           // painstate
0,               // painchance
sfx_None,         // painsound
S_NULL,           // meleestate
S_NULL,           // missilestate
S_NULL,           // deathstate
S_NULL,           // xdeathstate
sfx_None,         // deathsound
0,               // speed
20*FRACUNIT,      // radius
16*FRACUNIT,      // height
100,             // mass
0,               // damage
sfx_None,         // activesound
MF_SPECIAL,       // flags
S_NULL           // raisestate
},

{                // MT_MISC28
2004,            // doomednum
S_PLAS,          // spawnstate
1000,            // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_NULL,          // painstate
0,              // painchance
sfx_None,        // painsound
S_NULL,          // meleestate
S_NULL,          // missilestate
S_NULL,          // deathstate
S_NULL,          // xdeathstate
sfx_None,        // deathsound
0,              // speed
20*FRACUNIT,     // radius
16*FRACUNIT,     // height
100,            // mass
0,              // damage
sfx_None,        // activesound
MF_SPECIAL,      // flags
S_NULL           // raisestate
},

{                // MT_SHOTGUN
2001,            // doomednum
S_SHOT,          // spawnstate
1000,            // spawnhealth
S_NULL,          // seestate
sfx_None,        // seesound
8,              // reactiontime
sfx_None,        // attacksound
S_NULL,          // painstate
0,              // painchance
sfx_None,        // painsound
S_NULL,          // meleestate
S_NULL,          // missilestate
S_NULL,          // deathstate
S_NULL,          // xdeathstate

```

```

    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                  // damage
    sfx_None,           // activesound
    MF_SPECIAL,         // flags
    S_NULL              // raisestate
},

{                      // MT_SUPERSHOTGUN
    82,                // doomednum
    S_SHOT2,           // spawnstate
    1000,              // spawnhealth
    S_NULL,            // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,            // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,            // meleestate
    S_NULL,            // missilestate
    S_NULL,            // deathstate
    S_NULL,            // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SPECIAL,         // flags
    S_NULL              // raisestate
},

{                      // MT_MISC29
    85,                // doomednum
    S_TECHLAMP,         // spawnstate
    1000,              // spawnhealth
    S_NULL,            // seestate
    sfx_None,          // seesound
    8,                 // reactiontime
    sfx_None,          // attacksound
    S_NULL,            // painstate
    0,                 // painchance
    sfx_None,          // painsound
    S_NULL,            // meleestate
    S_NULL,            // missilestate
    S_NULL,            // deathstate
    S_NULL,            // xdeathstate
    sfx_None,          // deathsound
    0,                 // speed
    16*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SOLID,          // flags
    S_NULL              // raisestate
},

{                      // MT_MISC30
    86,                // doomednum

```

```

S_TECH2LAMP,          // spawnstate
1000,                 // spawnhealth
S_NULL,               // seestate
sfx_None,             // seesound
8,                    // reactiontime
sfx_None,             // attacksound
S_NULL,               // painstate
0,                    // painchance
sfx_None,             // painsound
S_NULL,               // meleestate
S_NULL,               // missilestate
S_NULL,               // deathstate
S_NULL,               // xdeathstate
sfx_None,             // deathsound
0,                    // speed
16*FRACUNIT,          // radius
16*FRACUNIT,          // height
100,                  // mass
0,                    // damage
sfx_None,             // activesound
MF_SOLID,             // flags
S_NULL                // raisestate
},

{                      // MT_MISC31
2028,                 // doomednum
S_COLU,               // spawnstate
1000,                 // spawnhealth
S_NULL,               // seestate
sfx_None,             // seesound
8,                    // reactiontime
sfx_None,             // attacksound
S_NULL,               // painstate
0,                    // painchance
sfx_None,             // painsound
S_NULL,               // meleestate
S_NULL,               // missilestate
S_NULL,               // deathstate
S_NULL,               // xdeathstate
sfx_None,             // deathsound
0,                    // speed
16*FRACUNIT,          // radius
16*FRACUNIT,          // height
100,                  // mass
0,                    // damage
sfx_None,             // activesound
MF_SOLID,             // flags
S_NULL                // raisestate
},

{                      // MT_MISC32
30,                   // doomednum
S_TALLGRNCOL,         // spawnstate
1000,                 // spawnhealth
S_NULL,               // seestate
sfx_None,             // seesound
8,                    // reactiontime
sfx_None,             // attacksound
S_NULL,               // painstate
0,                    // painchance
sfx_None,             // painsound
S_NULL,               // meleestate
S_NULL,               // missilestate
S_NULL,               // deathstate
S_NULL,               // xdeathstate

```



```

sfx_None,          // deathsound
0,                 // speed
16*FRACUNIT,       // radius
16*FRACUNIT,       // height
100,               // mass
0,                 // damage
sfx_None,          // activesound
MF_SOLID,          // flags
S_NULL             // raisestate
},

{                  // MT_MISC33
31,                // doomednum
S_SHRTGRNCOL,      // spawnstate
1000,              // spawnhealth
S_NULL,            // seestate
sfx_None,          // seesound
8,                 // reactiontime
sfx_None,          // attacksound
S_NULL,            // painstate
0,                 // painchance
sfx_None,          // painsound
S_NULL,            // meleestate
S_NULL,            // missilestate
S_NULL,            // deathstate
S_NULL,            // xdeathstate
sfx_None,          // deathsound
0,                 // speed
16*FRACUNIT,       // radius
16*FRACUNIT,       // height
100,               // mass
0,                 // damage
sfx_None,          // activesound
MF_SOLID,          // flags
S_NULL             // raisestate
},

{                  // MT_MISC34
32,                // doomednum
S_TALLREDCOL,      // spawnstate
1000,              // spawnhealth
S_NULL,            // seestate
sfx_None,          // seesound
8,                 // reactiontime
sfx_None,          // attacksound
S_NULL,            // painstate
0,                 // painchance
sfx_None,          // painsound
S_NULL,            // meleestate
S_NULL,            // missilestate
S_NULL,            // deathstate
S_NULL,            // xdeathstate
sfx_None,          // deathsound
0,                 // speed
16*FRACUNIT,       // radius
16*FRACUNIT,       // height
100,               // mass
0,                 // damage
sfx_None,          // activesound
MF_SOLID,          // flags
S_NULL             // raisestate
},

{                  // MT_MISC35
33,                // doomednum

```

```

S_SHRTREDCOL,          // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                    // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                    // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate
sfx_None,              // deathsound
0,                    // speed
16*FRACUNIT,           // radius
16*FRACUNIT,           // height
100,                   // mass
0,                    // damage
sfx_None,              // activesound
MF_SOLID,              // flags
S_NULL                 // raisestate
},

{                      // MT_MISC36
37,                    // doomednum
S_SKULLCOL,            // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                    // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                    // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate
sfx_None,              // deathsound
0,                    // speed
16*FRACUNIT,           // radius
16*FRACUNIT,           // height
100,                   // mass
0,                    // damage
sfx_None,              // activesound
MF_SOLID,              // flags
S_NULL                 // raisestate
},

{                      // MT_MISC37
36,                    // doomednum
S_HEARTCOL,            // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                    // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                    // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate

```

```

sfx_None,          // deathsound
0,                // speed
16*FRACUNIT,      // radius
16*FRACUNIT,      // height
100,              // mass
0,                // damage
sfx_None,          // activesound
MF_SOLID,         // flags
S_NULL            // raisestate
},

{
    // MT_MISC38
    41,            // doomednum
    S_EVILEYE,     // spawnstate
    1000,          // spawnhealth
    S_NULL,        // seestate
    sfx_None,      // seesound
    8,             // reactiontime
    sfx_None,      // attacksound
    S_NULL,        // painstate
    0,             // painchance
    sfx_None,      // painsound
    S_NULL,        // meleestate
    S_NULL,        // missilestate
    S_NULL,        // deathstate
    S_NULL,        // xdeathstate
    sfx_None,      // deathsound
    0,             // speed
    16*FRACUNIT,   // radius
    16*FRACUNIT,   // height
    100,           // mass
    0,             // damage
    sfx_None,      // activesound
    MF_SOLID,      // flags
    S_NULL         // raisestate
},

{
    // MT_MISC39
    42,            // doomednum
    S_FLOATSKULL,  // spawnstate
    1000,          // spawnhealth
    S_NULL,        // seestate
    sfx_None,      // seesound
    8,             // reactiontime
    sfx_None,      // attacksound
    S_NULL,        // painstate
    0,             // painchance
    sfx_None,      // painsound
    S_NULL,        // meleestate
    S_NULL,        // missilestate
    S_NULL,        // deathstate
    S_NULL,        // xdeathstate
    sfx_None,      // deathsound
    0,             // speed
    16*FRACUNIT,   // radius
    16*FRACUNIT,   // height
    100,           // mass
    0,             // damage
    sfx_None,      // activesound
    MF_SOLID,      // flags
    S_NULL         // raisestate
},

{
    // MT_MISC40
    43,            // doomednum

```

```

S_TORCHTREE,          // spawnstate
1000,                 // spawnhealth
S_NULL,               // seestate
sfx_None,             // seesound
8,                   // reactiontime
sfx_None,             // attacksound
S_NULL,               // painstate
0,                   // painchance
sfx_None,             // painsound
S_NULL,               // meleestate
S_NULL,               // missilestate
S_NULL,               // deathstate
S_NULL,               // xdeathstate
sfx_None,             // deathsound
0,                   // speed
16*FRACUNIT,          // radius
16*FRACUNIT,          // height
100,                 // mass
0,                   // damage
sfx_None,             // activesound
MF_SOLID,             // flags
S_NULL               // raisestate
},

{
    // MT_MISC41
    44,                // doomednum
    S_BLUETORCH,        // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_None,           // seesound
    8,                 // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                 // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,           // deathsound
    0,                 // speed
    16*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,               // mass
    0,                 // damage
    sfx_None,           // activesound
    MF_SOLID,           // flags
    S_NULL              // raisestate
},

{
    // MT_MISC42
    45,                // doomednum
    S_GREENTORCH,        // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_None,           // seesound
    8,                 // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                 // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate

```

```

sfx_None,          // deathsound
0,                // speed
16*FRACUNIT,      // radius
16*FRACUNIT,      // height
100,              // mass
0,                // damage
sfx_None,          // activesound
MF_SOLID,          // flags
S_NULL            // raisestate
},

{
    // MT_MISC43
    46,            // doomednum
    S_REDTORCH,    // spawnstate
    1000,          // spawnhealth
    S_NULL,        // seestate
    sfx_None,      // seesound
    8,             // reactiontime
    sfx_None,      // attacksound
    S_NULL,        // painstate
    0,            // painchance
    sfx_None,      // painsound
    S_NULL,        // meleestate
    S_NULL,        // missilestate
    S_NULL,        // deathstate
    S_NULL,        // xdeathstate
    sfx_None,      // deathsound
    0,            // speed
    16*FRACUNIT,   // radius
    16*FRACUNIT,   // height
    100,          // mass
    0,            // damage
    sfx_None,      // activesound
    MF_SOLID,      // flags
    S_NULL        // raisestate
},

{
    // MT_MISC44
    55,            // doomednum
    S_BTORCHSHRT,  // spawnstate
    1000,          // spawnhealth
    S_NULL,        // seestate
    sfx_None,      // seesound
    8,            // reactiontime
    sfx_None,      // attacksound
    S_NULL,        // painstate
    0,            // painchance
    sfx_None,      // painsound
    S_NULL,        // meleestate
    S_NULL,        // missilestate
    S_NULL,        // deathstate
    S_NULL,        // xdeathstate
    sfx_None,      // deathsound
    0,            // speed
    16*FRACUNIT,   // radius
    16*FRACUNIT,   // height
    100,          // mass
    0,            // damage
    sfx_None,      // activesound
    MF_SOLID,      // flags
    S_NULL        // raisestate
},

{
    // MT_MISC45
    56,            // doomednum

```

```

S_GTORCHSHRT,          // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                     // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                     // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate
sfx_None,              // deathsound
0,                     // speed
16*FRACUNIT,           // radius
16*FRACUNIT,           // height
100,                   // mass
0,                     // damage
sfx_None,              // activesound
MF_SOLID,              // flags
S_NULL                 // raisestate
},

{                      // MT_MISC46
57,                    // doomednum
S_RTORCHSHRT,          // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                     // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                     // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate
sfx_None,              // deathsound
0,                     // speed
16*FRACUNIT,           // radius
16*FRACUNIT,           // height
100,                   // mass
0,                     // damage
sfx_None,              // activesound
MF_SOLID,              // flags
S_NULL                 // raisestate
},

{                      // MT_MISC47
47,                    // doomednum
S_STALAGTITE,          // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                     // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                     // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate

```

```

    sfx_None,          // deathsound
    0,                 // speed
    16*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                 // damage
    sfx_None,          // activesound
    MF_SOLID,          // flags
    S_NULL,            // raisestate
},

{
    // MT_MISC48
    48,                // doomednum
    S_TECHPILLAR,       // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_None,           // seesound
    8,                  // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                  // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,           // deathsound
    0,                  // speed
    16*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                  // damage
    sfx_None,           // activesound
    MF_SOLID,          // flags
    S_NULL,            // raisestate
},

{
    // MT_MISC49
    34,                // doomednum
    S_CANDLESTIK,       // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_None,           // seesound
    8,                  // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                  // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,       // radius
    16*FRACUNIT,       // height
    100,               // mass
    0,                  // damage
    sfx_None,           // activesound
    0,                  // flags
    S_NULL,            // raisestate
},

{
    // MT_MISC50
    35,                // doomednum

```

```

S_CANDELABRA,          // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                     // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                     // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate
sfx_None,              // deathsound
0,                     // speed
16*FRACUNIT,           // radius
16*FRACUNIT,           // height
100,                   // mass
0,                     // damage
sfx_None,              // activesound
MF_SOLID,              // flags
S_NULL                 // raisestate
},

{                      // MT_MISC51
49,                    // doomednum
S_BLOODYTWITCH,        // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                     // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                     // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate
sfx_None,              // deathsound
0,                     // speed
16*FRACUNIT,           // radius
68*FRACUNIT,           // height
100,                   // mass
0,                     // damage
sfx_None,              // activesound
MF_SOLID|MF_SPAWNCEILING|MF_NOGRAVITY, // flags
S_NULL                 // raisestate
},

{                      // MT_MISC52
50,                    // doomednum
S_MEAT2,               // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                     // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                     // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate

```



```

sfx_None,          // deathsound
0,                 // speed
16*FRACUNIT,       // radius
84*FRACUNIT,       // height
100,               // mass
0,                 // damage
sfx_None,          // activesound
MF_SOLID|MF_SPAWNCEILING|MF_NOGRAVITY, // flags
S_NULL             // raisestate
},

{                  // MT_MISC53
51,                // doomednum
S_MEAT3,           // spawnstate
1000,              // spawnhealth
S_NULL,            // seestate
sfx_None,          // seesound
8,                 // reactiontime
sfx_None,          // attacksound
S_NULL,            // painstate
0,                 // painchance
sfx_None,          // painsound
S_NULL,            // meleestate
S_NULL,            // missilestate
S_NULL,            // deathstate
S_NULL,            // xdeathstate
sfx_None,          // deathsound
0,                 // speed
16*FRACUNIT,       // radius
84*FRACUNIT,       // height
100,               // mass
0,                 // damage
sfx_None,          // activesound
MF_SOLID|MF_SPAWNCEILING|MF_NOGRAVITY, // flags
S_NULL             // raisestate
},

{                  // MT_MISC54
52,                // doomednum
S_MEAT4,           // spawnstate
1000,              // spawnhealth
S_NULL,            // seestate
sfx_None,          // seesound
8,                 // reactiontime
sfx_None,          // attacksound
S_NULL,            // painstate
0,                 // painchance
sfx_None,          // painsound
S_NULL,            // meleestate
S_NULL,            // missilestate
S_NULL,            // deathstate
S_NULL,            // xdeathstate
sfx_None,          // deathsound
0,                 // speed
16*FRACUNIT,       // radius
68*FRACUNIT,       // height
100,               // mass
0,                 // damage
sfx_None,          // activesound
MF_SOLID|MF_SPAWNCEILING|MF_NOGRAVITY, // flags
S_NULL             // raisestate
},

{                  // MT_MISC55
53,                // doomednum

```

```

S_MEAT5,          // spawnstate
1000,             // spawnhealth
S_NULL,           // seestate
sfx_None,         // seesound
8,               // reactiontime
sfx_None,         // attacksound
S_NULL,           // painstate
0,               // painchance
sfx_None,         // painsound
S_NULL,           // meleestate
S_NULL,           // missilestate
S_NULL,           // deathstate
S_NULL,           // xdeathstate
sfx_None,         // deathsound
0,               // speed
16*FRACUNIT,      // radius
52*FRACUNIT,      // height
100,              // mass
0,               // damage
sfx_None,         // activesound
MF_SOLID|MF_SPAWNCEILING|MF_NOGRAVITY, // flags
S_NULL           // raisestate
},

{                // MT_MISC56
59,              // doomednum
S_MEAT2,          // spawnstate
1000,             // spawnhealth
S_NULL,           // seestate
sfx_None,         // seesound
8,               // reactiontime
sfx_None,         // attacksound
S_NULL,           // painstate
0,               // painchance
sfx_None,         // painsound
S_NULL,           // meleestate
S_NULL,           // missilestate
S_NULL,           // deathstate
S_NULL,           // xdeathstate
sfx_None,         // deathsound
0,               // speed
20*FRACUNIT,      // radius
84*FRACUNIT,      // height
100,              // mass
0,               // damage
sfx_None,         // activesound
MF_SPAWNCEILING|MF_NOGRAVITY, // flags
S_NULL           // raisestate
},

{                // MT_MISC57
60,              // doomednum
S_MEAT4,          // spawnstate
1000,             // spawnhealth
S_NULL,           // seestate
sfx_None,         // seesound
8,               // reactiontime
sfx_None,         // attacksound
S_NULL,           // painstate
0,               // painchance
sfx_None,         // painsound
S_NULL,           // meleestate
S_NULL,           // missilestate
S_NULL,           // deathstate
S_NULL,           // xdeathstate

```

```

sfx_None,          // deathsound
0,                 // speed
20*FRACUNIT,       // radius
68*FRACUNIT,       // height
100,               // mass
0,                 // damage
sfx_None,          // activesound
MF_SPAWNCEILING|MF_NOGRAVITY, // flags
S_NULL             // raisestate
},

{
    // MT_MISC58
    61,             // doomednum
    S_MEAT3,        // spawnstate
    1000,           // spawnhealth
    S_NULL,         // seestate
    sfx_None,       // seesound
    8,              // reactiontime
    sfx_None,       // attacksound
    S_NULL,         // painstate
    0,              // painchance
    sfx_None,       // painsound
    S_NULL,         // meleestate
    S_NULL,         // missilestate
    S_NULL,         // deathstate
    S_NULL,         // xdeathstate
    sfx_None,       // deathsound
    0,              // speed
    20*FRACUNIT,    // radius
    52*FRACUNIT,    // height
    100,            // mass
    0,              // damage
    sfx_None,       // activesound
    MF_SPAWNCEILING|MF_NOGRAVITY, // flags
    S_NULL          // raisestate
},

{
    // MT_MISC59
    62,             // doomednum
    S_MEAT5,        // spawnstate
    1000,           // spawnhealth
    S_NULL,         // seestate
    sfx_None,       // seesound
    8,              // reactiontime
    sfx_None,       // attacksound
    S_NULL,         // painstate
    0,              // painchance
    sfx_None,       // painsound
    S_NULL,         // meleestate
    S_NULL,         // missilestate
    S_NULL,         // deathstate
    S_NULL,         // xdeathstate
    sfx_None,       // deathsound
    0,              // speed
    20*FRACUNIT,    // radius
    52*FRACUNIT,    // height
    100,            // mass
    0,              // damage
    sfx_None,       // activesound
    MF_SPAWNCEILING|MF_NOGRAVITY, // flags
    S_NULL          // raisestate
},

{
    // MT_MISC60
    63,             // doomednum

```

```

S_BLOODYTWITCH,          // spawnstate
1000,                    // spawnhealth
S_NULL,                  // seestate
sfx_None,                // seesound
8,                        // reactiontime
sfx_None,                // attacksound
S_NULL,                  // painstate
0,                        // painchance
sfx_None,                // painsound
S_NULL,                  // meleestate
S_NULL,                  // missilestate
S_NULL,                  // deathstate
S_NULL,                  // xdeathstate
sfx_None,                // deathsound
0,                        // speed
20*FRACUNIT,             // radius
68*FRACUNIT,             // height
100,                     // mass
0,                        // damage
sfx_None,                // activesound
MF_SPAWNCEILING|MF_NOGRAVITY, // flags
S_NULL                   // raisestate
},

{                          // MT_MISC61
22,                       // doomednum
S_HEAD_DIE6,              // spawnstate
1000,                     // spawnhealth
S_NULL,                   // seestate
sfx_None,                 // seesound
8,                         // reactiontime
sfx_None,                 // attacksound
S_NULL,                   // painstate
0,                         // painchance
sfx_None,                 // painsound
S_NULL,                   // meleestate
S_NULL,                   // missilestate
S_NULL,                   // deathstate
S_NULL,                   // xdeathstate
sfx_None,                 // deathsound
0,                         // speed
20*FRACUNIT,              // radius
16*FRACUNIT,              // height
100,                       // mass
0,                         // damage
sfx_None,                 // activesound
0,                         // flags
S_NULL                     // raisestate
},

{                          // MT_MISC62
15,                       // doomednum
S_PLAY_DIE7,              // spawnstate
1000,                     // spawnhealth
S_NULL,                   // seestate
sfx_None,                 // seesound
8,                         // reactiontime
sfx_None,                 // attacksound
S_NULL,                   // painstate
0,                         // painchance
sfx_None,                 // painsound
S_NULL,                   // meleestate
S_NULL,                   // missilestate
S_NULL,                   // deathstate
S_NULL,                   // xdeathstate

```

```

    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                  // damage
    sfx_None,           // activesound
    0,                  // flags
    S_NULL              // raisestate
},

{
    // MT_MISC63
    18,                 // doomednum
    S_POSS_DIE5,        // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_None,           // seesound
    8,                  // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                  // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                  // damage
    sfx_None,           // activesound
    0,                  // flags
    S_NULL              // raisestate
},

{
    // MT_MISC64
    21,                 // doomednum
    S_SARG_DIE6,        // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_None,           // seesound
    8,                  // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                  // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                  // damage
    sfx_None,           // activesound
    0,                  // flags
    S_NULL              // raisestate
},

{
    // MT_MISC65
    23,                 // doomednum

```

```

S_SKULL_DIE6,          // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                     // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                     // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate
sfx_None,              // deathsound
0,                     // speed
20*FRACUNIT,           // radius
16*FRACUNIT,           // height
100,                   // mass
0,                     // damage
sfx_None,              // activesound
0,                     // flags
S_NULL                 // raisestate
},

{                      // MT_MISC66
20,                    // doomednum
S_TROO_DIE5,           // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                     // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                     // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate
sfx_None,              // deathsound
0,                     // speed
20*FRACUNIT,           // radius
16*FRACUNIT,           // height
100,                   // mass
0,                     // damage
sfx_None,              // activesound
0,                     // flags
S_NULL                 // raisestate
},

{                      // MT_MISC67
19,                    // doomednum
S_SPOS_DIE5,           // spawnstate
1000,                  // spawnhealth
S_NULL,                // seestate
sfx_None,              // seesound
8,                     // reactiontime
sfx_None,              // attacksound
S_NULL,                // painstate
0,                     // painchance
sfx_None,              // painsound
S_NULL,                // meleestate
S_NULL,                // missilestate
S_NULL,                // deathstate
S_NULL,                // xdeathstate

```

```

    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                  // damage
    sfx_None,           // activesound
    0,                  // flags
    S_NULL              // raisestate
},

{
    // MT_MISC68
    10,                 // doomednum
    S_PLAY_XDIE9,       // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_None,           // seesound
    8,                  // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                  // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                  // damage
    sfx_None,           // activesound
    0,                  // flags
    S_NULL              // raisestate
},

{
    // MT_MISC69
    12,                 // doomednum
    S_PLAY_XDIE9,       // spawnstate
    1000,               // spawnhealth
    S_NULL,             // seestate
    sfx_None,           // seesound
    8,                  // reactiontime
    sfx_None,           // attacksound
    S_NULL,             // painstate
    0,                  // painchance
    sfx_None,           // painsound
    S_NULL,             // meleestate
    S_NULL,             // missilestate
    S_NULL,             // deathstate
    S_NULL,             // xdeathstate
    sfx_None,           // deathsound
    0,                  // speed
    20*FRACUNIT,        // radius
    16*FRACUNIT,        // height
    100,                // mass
    0,                  // damage
    sfx_None,           // activesound
    0,                  // flags
    S_NULL              // raisestate
},

{
    // MT_MISC70
    28,                 // doomednum

```

```

S_HEADSONSTICK,          // spawnstate
1000,                    // spawnhealth
S_NULL,                  // seestate
sfx_None,                // seesound
8,                        // reactiontime
sfx_None,                // attacksound
S_NULL,                  // painstate
0,                        // painchance
sfx_None,                // painsound
S_NULL,                  // meleestate
S_NULL,                  // missilestate
S_NULL,                  // deathstate
S_NULL,                  // xdeathstate
sfx_None,                // deathsound
0,                        // speed
16*FRACUNIT,             // radius
16*FRACUNIT,             // height
100,                     // mass
0,                        // damage
sfx_None,                // activesound
MF_SOLID,                // flags
S_NULL                   // raisestate
},

{                          // MT_MISC71
24,                       // doomednum
S_GIBS,                   // spawnstate
1000,                     // spawnhealth
S_NULL,                   // seestate
sfx_None,                 // seesound
8,                         // reactiontime
sfx_None,                 // attacksound
S_NULL,                   // painstate
0,                         // painchance
sfx_None,                 // painsound
S_NULL,                   // meleestate
S_NULL,                   // missilestate
S_NULL,                   // deathstate
S_NULL,                   // xdeathstate
sfx_None,                 // deathsound
0,                         // speed
20*FRACUNIT,              // radius
16*FRACUNIT,              // height
100,                       // mass
0,                         // damage
sfx_None,                 // activesound
0,                         // flags
S_NULL                     // raisestate
},

{                          // MT_MISC72
27,                       // doomednum
S_HEADONASTICK,           // spawnstate
1000,                     // spawnhealth
S_NULL,                   // seestate
sfx_None,                 // seesound
8,                         // reactiontime
sfx_None,                 // attacksound
S_NULL,                   // painstate
0,                         // painchance
sfx_None,                 // painsound
S_NULL,                   // meleestate
S_NULL,                   // missilestate
S_NULL,                   // deathstate
S_NULL,                   // xdeathstate

```



```

sfx_None,          // deathsound
0,                // speed
16*FRACUNIT,      // radius
16*FRACUNIT,      // height
100,              // mass
0,                // damage
sfx_None,          // activesound
MF_SOLID,         // flags
S_NULL            // raisestate
},

{
    // MT_MISC73
    29,            // doomednum
    S_HEADCANDLES, // spawnstate
    1000,          // spawnhealth
    S_NULL,        // seestate
    sfx_None,      // seesound
    8,             // reactiontime
    sfx_None,      // attacksound
    S_NULL,        // painstate
    0,             // painchance
    sfx_None,      // painsound
    S_NULL,        // meleestate
    S_NULL,        // missilestate
    S_NULL,        // deathstate
    S_NULL,        // xdeathstate
    sfx_None,      // deathsound
    0,             // speed
    16*FRACUNIT,   // radius
    16*FRACUNIT,   // height
    100,           // mass
    0,             // damage
    sfx_None,      // activesound
    MF_SOLID,      // flags
    S_NULL         // raisestate
},

{
    // MT_MISC74
    25,            // doomednum
    S_DEADSTICK,   // spawnstate
    1000,          // spawnhealth
    S_NULL,        // seestate
    sfx_None,      // seesound
    8,             // reactiontime
    sfx_None,      // attacksound
    S_NULL,        // painstate
    0,             // painchance
    sfx_None,      // painsound
    S_NULL,        // meleestate
    S_NULL,        // missilestate
    S_NULL,        // deathstate
    S_NULL,        // xdeathstate
    sfx_None,      // deathsound
    0,             // speed
    16*FRACUNIT,   // radius
    16*FRACUNIT,   // height
    100,           // mass
    0,             // damage
    sfx_None,      // activesound
    MF_SOLID,      // flags
    S_NULL         // raisestate
},

{
    // MT_MISC75
    26,            // doomednum

```

```

S_LIVESTICK,          // spawnstate
1000,                 // spawnhealth
S_NULL,               // seestate
sfx_None,             // seesound
8,                    // reactiontime
sfx_None,             // attacksound
S_NULL,               // painstate
0,                    // painchance
sfx_None,             // painsound
S_NULL,               // meleestate
S_NULL,               // missilestate
S_NULL,               // deathstate
S_NULL,               // xdeathstate
sfx_None,             // deathsound
0,                    // speed
16*FRACUNIT,          // radius
16*FRACUNIT,          // height
100,                  // mass
0,                    // damage
sfx_None,             // activesound
MF_SOLID,             // flags
S_NULL                // raisestate
},

{                      // MT_MISC76
54,                   // doomednum
S_BIGTREE,            // spawnstate
1000,                 // spawnhealth
S_NULL,               // seestate
sfx_None,             // seesound
8,                    // reactiontime
sfx_None,             // attacksound
S_NULL,               // painstate
0,                    // painchance
sfx_None,             // painsound
S_NULL,               // meleestate
S_NULL,               // missilestate
S_NULL,               // deathstate
S_NULL,               // xdeathstate
sfx_None,             // deathsound
0,                    // speed
32*FRACUNIT,          // radius
16*FRACUNIT,          // height
100,                  // mass
0,                    // damage
sfx_None,             // activesound
MF_SOLID,             // flags
S_NULL                // raisestate
},

{                      // MT_MISC77
70,                   // doomednum
S_BBAR1,              // spawnstate
1000,                 // spawnhealth
S_NULL,               // seestate
sfx_None,             // seesound
8,                    // reactiontime
sfx_None,             // attacksound
S_NULL,               // painstate
0,                    // painchance
sfx_None,             // painsound
S_NULL,               // meleestate
S_NULL,               // missilestate
S_NULL,               // deathstate
S_NULL,               // xdeathstate

```

```

sfx_None,          // deathsound
0,                // speed
16*FRACUNIT,       // radius
16*FRACUNIT,       // height
100,              // mass
0,                // damage
sfx_None,          // activesound
MF_SOLID,          // flags
S_NULL            // raisestate
},

{
    // MT_MISC78
    73,            // doomednum
    S_HANGNOGUTS,  // spawnstate
    1000,          // spawnhealth
    S_NULL,        // seestate
    sfx_None,      // seesound
    8,             // reactiontime
    sfx_None,      // attacksound
    S_NULL,        // painstate
    0,             // painchance
    sfx_None,      // painsound
    S_NULL,        // meleestate
    S_NULL,        // missilestate
    S_NULL,        // deathstate
    S_NULL,        // xdeathstate
    sfx_None,      // deathsound
    0,             // speed
    16*FRACUNIT,   // radius
    88*FRACUNIT,   // height
    100,           // mass
    0,             // damage
    sfx_None,      // activesound
    MF_SOLID|MF_SPAWNCEILING|MF_NOGRAVITY, // flags
    S_NULL        // raisestate
},

{
    // MT_MISC79
    74,            // doomednum
    S_HANGBNOBRAIN, // spawnstate
    1000,          // spawnhealth
    S_NULL,        // seestate
    sfx_None,      // seesound
    8,             // reactiontime
    sfx_None,      // attacksound
    S_NULL,        // painstate
    0,             // painchance
    sfx_None,      // painsound
    S_NULL,        // meleestate
    S_NULL,        // missilestate
    S_NULL,        // deathstate
    S_NULL,        // xdeathstate
    sfx_None,      // deathsound
    0,             // speed
    16*FRACUNIT,   // radius
    88*FRACUNIT,   // height
    100,           // mass
    0,             // damage
    sfx_None,      // activesound
    MF_SOLID|MF_SPAWNCEILING|MF_NOGRAVITY, // flags
    S_NULL        // raisestate
},

{
    // MT_MISC80
    75,            // doomednum

```

```

S_HANGTLOOKDN,          // spawnstate
1000,                   // spawnhealth
S_NULL,                 // seestate
sfx_None,               // seesound
8,                      // reactiontime
sfx_None,               // attacksound
S_NULL,                 // painstate
0,                      // painchance
sfx_None,               // painsound
S_NULL,                 // meleestate
S_NULL,                 // missilestate
S_NULL,                 // deathstate
S_NULL,                 // xdeathstate
sfx_None,               // deathsound
0,                      // speed
16*FRACUNIT,            // radius
64*FRACUNIT,            // height
100,                    // mass
0,                      // damage
sfx_None,               // activesound
MF_SOLID|MF_SPAWNCEILING|MF_NOGRAVITY, // flags
S_NULL                  // raisestate
},

{                      // MT_MISC81
76,                     // doomednum
S_HANGTSKULL,           // spawnstate
1000,                   // spawnhealth
S_NULL,                 // seestate
sfx_None,               // seesound
8,                      // reactiontime
sfx_None,               // attacksound
S_NULL,                 // painstate
0,                      // painchance
sfx_None,               // painsound
S_NULL,                 // meleestate
S_NULL,                 // missilestate
S_NULL,                 // deathstate
S_NULL,                 // xdeathstate
sfx_None,               // deathsound
0,                      // speed
16*FRACUNIT,            // radius
64*FRACUNIT,            // height
100,                    // mass
0,                      // damage
sfx_None,               // activesound
MF_SOLID|MF_SPAWNCEILING|MF_NOGRAVITY, // flags
S_NULL                  // raisestate
},

{                      // MT_MISC82
77,                     // doomednum
S_HANGTLOOKUP,          // spawnstate
1000,                   // spawnhealth
S_NULL,                 // seestate
sfx_None,               // seesound
8,                      // reactiontime
sfx_None,               // attacksound
S_NULL,                 // painstate
0,                      // painchance
sfx_None,               // painsound
S_NULL,                 // meleestate
S_NULL,                 // missilestate
S_NULL,                 // deathstate
S_NULL,                 // xdeathstate

```

```

sfx_None,          // deathsound
0,                // speed
16*FRACUNIT,      // radius
64*FRACUNIT,      // height
100,              // mass
0,                // damage
sfx_None,          // activesound
MF_SOLID|MF_SPAWNCEILING|MF_NOGRAVITY, // flags
S_NULL            // raisestate
},

{
    // MT_MISC83
    78,            // doomednum
    S_HANGTNOBRAIN, // spawnstate
    1000,          // spawnhealth
    S_NULL,         // seestate
    sfx_None,       // seesound
    8,              // reactiontime
    sfx_None,       // attacksound
    S_NULL,         // painstate
    0,              // painchance
    sfx_None,       // painsound
    S_NULL,         // meleestate
    S_NULL,         // missilestate
    S_NULL,         // deathstate
    S_NULL,         // xdeathstate
    sfx_None,       // deathsound
    0,              // speed
    16*FRACUNIT,    // radius
    64*FRACUNIT,    // height
    100,            // mass
    0,              // damage
    sfx_None,       // activesound
    MF_SOLID|MF_SPAWNCEILING|MF_NOGRAVITY, // flags
    S_NULL          // raisestate
},

{
    // MT_MISC84
    79,            // doomednum
    S_COLONGIBS,   // spawnstate
    1000,          // spawnhealth
    S_NULL,         // seestate
    sfx_None,       // seesound
    8,              // reactiontime
    sfx_None,       // attacksound
    S_NULL,         // painstate
    0,              // painchance
    sfx_None,       // painsound
    S_NULL,         // meleestate
    S_NULL,         // missilestate
    S_NULL,         // deathstate
    S_NULL,         // xdeathstate
    sfx_None,       // deathsound
    0,              // speed
    20*FRACUNIT,   // radius
    16*FRACUNIT,   // height
    100,            // mass
    0,              // damage
    sfx_None,       // activesound
    MF_NOBLOCKMAP, // flags
    S_NULL          // raisestate
},

{
    // MT_MISC85
    80,            // doomednum

```

```

    S_SMALLPOOL,          // spawnstate
    1000,                  // spawnhealth
    S_NULL,                // seestate
    sfx_None,              // seesound
    8,                     // reactiontime
    sfx_None,              // attacksound
    S_NULL,                // painstate
    0,                     // painchance
    sfx_None,              // painsound
    S_NULL,                // meleestate
    S_NULL,                // missilestate
    S_NULL,                // deathstate
    S_NULL,                // xdeathstate
    sfx_None,              // deathsound
    0,                     // speed
    20*FRACUNIT,           // radius
    16*FRACUNIT,           // height
    100,                   // mass
    0,                     // damage
    sfx_None,              // activesound
    MF_NOBLOCKMAP,         // flags
    S_NULL                 // raisestate
},

{                          // MT_MISC86
    81,                    // doomednum
    S_BRAINSTEM,           // spawnstate
    1000,                  // spawnhealth
    S_NULL,                // seestate
    sfx_None,              // seesound
    8,                     // reactiontime
    sfx_None,              // attacksound
    S_NULL,                // painstate
    0,                     // painchance
    sfx_None,              // painsound
    S_NULL,                // meleestate
    S_NULL,                // missilestate
    S_NULL,                // deathstate
    S_NULL,                // xdeathstate
    sfx_None,              // deathsound
    0,                     // speed
    20*FRACUNIT,           // radius
    16*FRACUNIT,           // height
    100,                   // mass
    0,                     // damage
    sfx_None,              // activesound
    MF_NOBLOCKMAP,         // flags
    S_NULL                 // raisestate
}
};

```

## 9.2 info.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.

```

```

//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Thing frame/state LUT,
//     generated by multigen utility.
//     This one is the original DOOM version, preserved.
//
//-----

#ifndef __INFO__
#define __INFO__

// Needed for action function pointer handling.
#include "d_think.h"

typedef enum
{
    SPR_TROO,
    SPR_SHTG,
    SPR_PUNG,
    SPR_PISG,
    SPR_PISF,
    SPR_SHTF,
    SPR_SHT2,
    SPR_CHGG,
    SPR_CHGF,
    SPR_MISG,
    SPR_MISF,
    SPR_SAWG,
    SPR_PLSG,
    SPR_PLSF,
    SPR_BFGG,
    SPR_BFGF,
    SPR_BLUD,
    SPR_PUFF,
    SPR_BAL1,
    SPR_BAL2,
    SPR_PLSS,
    SPR_PLSE,
    SPR_MISL,
    SPR_BFS1,
    SPR_BFE1,
    SPR_BFE2,
    SPR_TFOG,
    SPR_IFOG,
    SPR_PLAY,
    SPR_POSS,
    SPR_SPOS,
    SPR_VILE,
    SPR_FIRE,
    SPR_FATB,
    SPR_FBXP,
    SPR_SKEL,
    SPR_MANF,
    SPR_FATT,
    SPR_CPOS,
    SPR_SARG,
    SPR_HEAD,
    SPR_BAL7,
    SPR_BOSS,
    SPR_BOS2,

```

SPR\_SKUL,  
SPR\_SPID,  
SPR\_BSPI,  
SPR\_APLS,  
SPR\_APBX,  
SPR\_CYBR,  
SPR\_PAIN,  
SPR\_SSWV,  
SPR\_KEEN,  
SPR\_BBRN,  
SPR\_BOSF,  
SPR\_ARM1,  
SPR\_ARM2,  
SPR\_BAR1,  
SPR\_BEXP,  
SPR\_FCAN,  
SPR\_BON1,  
SPR\_BON2,  
SPR\_BKEY,  
SPR\_RKEY,  
SPR\_YKEY,  
SPR\_BSKU,  
SPR\_RSKU,  
SPR\_YSKU,  
SPR\_STIM,  
SPR\_MEDI,  
SPR\_SOUL,  
SPR\_PINV,  
SPR\_PSTR,  
SPR\_PINS,  
SPR\_MEGA,  
SPR\_SUIT,  
SPR\_PMAP,  
SPR\_PVIS,  
SPR\_CLIP,  
SPR\_AMMO,  
SPR\_ROCK,  
SPR\_BROK,  
SPR\_CELL,  
SPR\_CELP,  
SPR\_SHEL,  
SPR\_SBOX,  
SPR\_BPAK,  
SPR\_BFUG,  
SPR\_MGUN,  
SPR\_CSAW,  
SPR\_LAUN,  
SPR\_PLAS,  
SPR\_SHOT,  
SPR\_SGN2,  
SPR\_COLU,  
SPR\_SMT2,  
SPR\_GOR1,  
SPR\_POL2,  
SPR\_POL5,  
SPR\_POL4,  
SPR\_POL3,  
SPR\_POL1,  
SPR\_POL6,  
SPR\_GOR2,  
SPR\_GOR3,  
SPR\_GOR4,  
SPR\_GOR5,  
SPR\_SMIT,  
SPR\_COL1,



```

    SPR_COL2,
    SPR_COL3,
    SPR_COL4,
    SPR_CAND,
    SPR_CBRA,
    SPR_COL6,
    SPR_TRE1,
    SPR_TRE2,
    SPR_ELEC,
    SPR_CEYE,
    SPR_FSKU,
    SPR_COL5,
    SPR_TBLU,
    SPR_TGRN,
    SPR_TRED,
    SPR_SMBT,
    SPR_SMG1,
    SPR_SMRT,
    SPR_HDB1,
    SPR_HDB2,
    SPR_HDB3,
    SPR_HDB4,
    SPR_HDB5,
    SPR_HDB6,
    SPR_POB1,
    SPR_POB2,
    SPR_BRS1,
    SPR_TLMP,
    SPR_TLP2,
    NUMSPRITES

} spritenum_t;

typedef enum
{
    S_NULL,
    S_LIGHTDONE,
    S_PUNCH,
    S_PUNCHDOWN,
    S_PUNCHUP,
    S_PUNCH1,
    S_PUNCH2,
    S_PUNCH3,
    S_PUNCH4,
    S_PUNCH5,
    S_PISTOL,
    S_PISTOLDOWN,
    S_PISTOLUP,
    S_PISTOL1,
    S_PISTOL2,
    S_PISTOL3,
    S_PISTOL4,
    S_PISTOLFLASH,
    S_SGUN,
    S_SGUNDOWN,
    S_SGUNUP,
    S_SGUN1,
    S_SGUN2,
    S_SGUN3,
    S_SGUN4,
    S_SGUN5,
    S_SGUN6,
    S_SGUN7,
    S_SGUN8,
    S_SGUN9,

```

S\_SGUNFLASH1,  
S\_SGUNFLASH2,  
S\_DSGUN,  
S\_DSGUNDOWN,  
S\_DSGUNUP,  
S\_DSGUN1,  
S\_DSGUN2,  
S\_DSGUN3,  
S\_DSGUN4,  
S\_DSGUN5,  
S\_DSGUN6,  
S\_DSGUN7,  
S\_DSGUN8,  
S\_DSGUN9,  
S\_DSGUN10,  
S\_DSNR1,  
S\_DSNR2,  
S\_DSGUNFLASH1,  
S\_DSGUNFLASH2,  
S\_CHAIN,  
S\_CHAINDOWN,  
S\_CHAINUP,  
S\_CHAIN1,  
S\_CHAIN2,  
S\_CHAIN3,  
S\_CHAINFLASH1,  
S\_CHAINFLASH2,  
S\_MISSILE,  
S\_MISSILEDOWN,  
S\_MISSILEUP,  
S\_MISSILE1,  
S\_MISSILE2,  
S\_MISSILE3,  
S\_MISSILEFLASH1,  
S\_MISSILEFLASH2,  
S\_MISSILEFLASH3,  
S\_MISSILEFLASH4,  
S\_SAW,  
S\_SAWB,  
S\_SAWDOWN,  
S\_SAWUP,  
S\_SAW1,  
S\_SAW2,  
S\_SAW3,  
S\_PLASMA,  
S\_PLASMADOWN,  
S\_PLASMAUP,  
S\_PLASMA1,  
S\_PLASMA2,  
S\_PLASMAFLASH1,  
S\_PLASMAFLASH2,  
S\_BFG,  
S\_BFGDOWN,  
S\_BFGUP,  
S\_BFG1,  
S\_BFG2,  
S\_BFG3,  
S\_BFG4,  
S\_BFGFLASH1,  
S\_BFGFLASH2,  
S\_BLOOD1,  
S\_BLOOD2,  
S\_BLOOD3,  
S\_PUFF1,  
S\_PUFF2,

S\_PUFF3,  
S\_PUFF4,  
S\_TBALL1,  
S\_TBALL2,  
S\_TBALLX1,  
S\_TBALLX2,  
S\_TBALLX3,  
S\_RBALL1,  
S\_RBALL2,  
S\_RBALLX1,  
S\_RBALLX2,  
S\_RBALLX3,  
S\_PLASBALL,  
S\_PLASBALL2,  
S\_PLASEXP,  
S\_PLASEXP2,  
S\_PLASEXP3,  
S\_PLASEXP4,  
S\_PLASEXP5,  
S\_ROCKET,  
S\_BFGSHOT,  
S\_BFGSHOT2,  
S\_BFGLAND,  
S\_BFGLAND2,  
S\_BFGLAND3,  
S\_BFGLAND4,  
S\_BFGLAND5,  
S\_BFGLAND6,  
S\_BFGEXP,  
S\_BFGEXP2,  
S\_BFGEXP3,  
S\_BFGEXP4,  
S\_EXPLODE1,  
S\_EXPLODE2,  
S\_EXPLODE3,  
S\_TFOG,  
S\_TFOG01,  
S\_TFOG02,  
S\_TFOG2,  
S\_TFOG3,  
S\_TFOG4,  
S\_TFOG5,  
S\_TFOG6,  
S\_TFOG7,  
S\_TFOG8,  
S\_TFOG9,  
S\_TFOG10,  
S\_IFOG,  
S\_IFOG01,  
S\_IFOG02,  
S\_IFOG2,  
S\_IFOG3,  
S\_IFOG4,  
S\_IFOG5,  
S\_PLAY,  
S\_PLAY\_RUN1,  
S\_PLAY\_RUN2,  
S\_PLAY\_RUN3,  
S\_PLAY\_RUN4,  
S\_PLAY\_ATK1,  
S\_PLAY\_ATK2,  
S\_PLAY\_PAIN,  
S\_PLAY\_PAIN2,  
S\_PLAY\_DIE1,  
S\_PLAY\_DIE2,

S\_PLAY\_DIE3,  
S\_PLAY\_DIE4,  
S\_PLAY\_DIE5,  
S\_PLAY\_DIE6,  
S\_PLAY\_DIE7,  
S\_PLAY\_XDIE1,  
S\_PLAY\_XDIE2,  
S\_PLAY\_XDIE3,  
S\_PLAY\_XDIE4,  
S\_PLAY\_XDIE5,  
S\_PLAY\_XDIE6,  
S\_PLAY\_XDIE7,  
S\_PLAY\_XDIE8,  
S\_PLAY\_XDIE9,  
S\_POSS\_STND,  
S\_POSS\_STND2,  
S\_POSS\_RUN1,  
S\_POSS\_RUN2,  
S\_POSS\_RUN3,  
S\_POSS\_RUN4,  
S\_POSS\_RUN5,  
S\_POSS\_RUN6,  
S\_POSS\_RUN7,  
S\_POSS\_RUN8,  
S\_POSS\_ATK1,  
S\_POSS\_ATK2,  
S\_POSS\_ATK3,  
S\_POSS\_PAIN,  
S\_POSS\_PAIN2,  
S\_POSS\_DIE1,  
S\_POSS\_DIE2,  
S\_POSS\_DIE3,  
S\_POSS\_DIE4,  
S\_POSS\_DIE5,  
S\_POSS\_XDIE1,  
S\_POSS\_XDIE2,  
S\_POSS\_XDIE3,  
S\_POSS\_XDIE4,  
S\_POSS\_XDIE5,  
S\_POSS\_XDIE6,  
S\_POSS\_XDIE7,  
S\_POSS\_XDIE8,  
S\_POSS\_XDIE9,  
S\_POSS\_RAISE1,  
S\_POSS\_RAISE2,  
S\_POSS\_RAISE3,  
S\_POSS\_RAISE4,  
S\_SPOS\_STND,  
S\_SPOS\_STND2,  
S\_SPOS\_RUN1,  
S\_SPOS\_RUN2,  
S\_SPOS\_RUN3,  
S\_SPOS\_RUN4,  
S\_SPOS\_RUN5,  
S\_SPOS\_RUN6,  
S\_SPOS\_RUN7,  
S\_SPOS\_RUN8,  
S\_SPOS\_ATK1,  
S\_SPOS\_ATK2,  
S\_SPOS\_ATK3,  
S\_SPOS\_PAIN,  
S\_SPOS\_PAIN2,  
S\_SPOS\_DIE1,  
S\_SPOS\_DIE2,  
S\_SPOS\_DIE3,

S\_SPOS\_DIE4,  
S\_SPOS\_DIE5,  
S\_SPOS\_XDIE1,  
S\_SPOS\_XDIE2,  
S\_SPOS\_XDIE3,  
S\_SPOS\_XDIE4,  
S\_SPOS\_XDIE5,  
S\_SPOS\_XDIE6,  
S\_SPOS\_XDIE7,  
S\_SPOS\_XDIE8,  
S\_SPOS\_XDIE9,  
S\_SPOS\_RAISE1,  
S\_SPOS\_RAISE2,  
S\_SPOS\_RAISE3,  
S\_SPOS\_RAISE4,  
S\_SPOS\_RAISE5,  
S\_VILE\_STND,  
S\_VILE\_STND2,  
S\_VILE\_RUN1,  
S\_VILE\_RUN2,  
S\_VILE\_RUN3,  
S\_VILE\_RUN4,  
S\_VILE\_RUN5,  
S\_VILE\_RUN6,  
S\_VILE\_RUN7,  
S\_VILE\_RUN8,  
S\_VILE\_RUN9,  
S\_VILE\_RUN10,  
S\_VILE\_RUN11,  
S\_VILE\_RUN12,  
S\_VILE\_ATK1,  
S\_VILE\_ATK2,  
S\_VILE\_ATK3,  
S\_VILE\_ATK4,  
S\_VILE\_ATK5,  
S\_VILE\_ATK6,  
S\_VILE\_ATK7,  
S\_VILE\_ATK8,  
S\_VILE\_ATK9,  
S\_VILE\_ATK10,  
S\_VILE\_ATK11,  
S\_VILE\_HEAL1,  
S\_VILE\_HEAL2,  
S\_VILE\_HEAL3,  
S\_VILE\_PAIN,  
S\_VILE\_PAIN2,  
S\_VILE\_DIE1,  
S\_VILE\_DIE2,  
S\_VILE\_DIE3,  
S\_VILE\_DIE4,  
S\_VILE\_DIE5,  
S\_VILE\_DIE6,  
S\_VILE\_DIE7,  
S\_VILE\_DIE8,  
S\_VILE\_DIE9,  
S\_VILE\_DIE10,  
S\_FIRE1,  
S\_FIRE2,  
S\_FIRE3,  
S\_FIRE4,  
S\_FIRE5,  
S\_FIRE6,  
S\_FIRE7,  
S\_FIRE8,  
S\_FIRE9,

S\_FIRE10,  
S\_FIRE11,  
S\_FIRE12,  
S\_FIRE13,  
S\_FIRE14,  
S\_FIRE15,  
S\_FIRE16,  
S\_FIRE17,  
S\_FIRE18,  
S\_FIRE19,  
S\_FIRE20,  
S\_FIRE21,  
S\_FIRE22,  
S\_FIRE23,  
S\_FIRE24,  
S\_FIRE25,  
S\_FIRE26,  
S\_FIRE27,  
S\_FIRE28,  
S\_FIRE29,  
S\_FIRE30,  
S\_SMOKE1,  
S\_SMOKE2,  
S\_SMOKE3,  
S\_SMOKE4,  
S\_SMOKE5,  
S\_TRACER,  
S\_TRACER2,  
S\_TRACEEXP1,  
S\_TRACEEXP2,  
S\_TRACEEXP3,  
S\_SKEL\_STND,  
S\_SKEL\_STND2,  
S\_SKEL\_RUN1,  
S\_SKEL\_RUN2,  
S\_SKEL\_RUN3,  
S\_SKEL\_RUN4,  
S\_SKEL\_RUN5,  
S\_SKEL\_RUN6,  
S\_SKEL\_RUN7,  
S\_SKEL\_RUN8,  
S\_SKEL\_RUN9,  
S\_SKEL\_RUN10,  
S\_SKEL\_RUN11,  
S\_SKEL\_RUN12,  
S\_SKEL\_FIST1,  
S\_SKEL\_FIST2,  
S\_SKEL\_FIST3,  
S\_SKEL\_FIST4,  
S\_SKEL\_MISS1,  
S\_SKEL\_MISS2,  
S\_SKEL\_MISS3,  
S\_SKEL\_MISS4,  
S\_SKEL\_PAIN,  
S\_SKEL\_PAIN2,  
S\_SKEL\_DIE1,  
S\_SKEL\_DIE2,  
S\_SKEL\_DIE3,  
S\_SKEL\_DIE4,  
S\_SKEL\_DIE5,  
S\_SKEL\_DIE6,  
S\_SKEL\_RAISE1,  
S\_SKEL\_RAISE2,  
S\_SKEL\_RAISE3,  
S\_SKEL\_RAISE4,

S\_SKEL\_RAISE5,  
S\_SKEL\_RAISE6,  
S\_FATSHOT1,  
S\_FATSHOT2,  
S\_FATSHOTX1,  
S\_FATSHOTX2,  
S\_FATSHOTX3,  
S\_FATT\_STND,  
S\_FATT\_STND2,  
S\_FATT\_RUN1,  
S\_FATT\_RUN2,  
S\_FATT\_RUN3,  
S\_FATT\_RUN4,  
S\_FATT\_RUN5,  
S\_FATT\_RUN6,  
S\_FATT\_RUN7,  
S\_FATT\_RUN8,  
S\_FATT\_RUN9,  
S\_FATT\_RUN10,  
S\_FATT\_RUN11,  
S\_FATT\_RUN12,  
S\_FATT\_ATK1,  
S\_FATT\_ATK2,  
S\_FATT\_ATK3,  
S\_FATT\_ATK4,  
S\_FATT\_ATK5,  
S\_FATT\_ATK6,  
S\_FATT\_ATK7,  
S\_FATT\_ATK8,  
S\_FATT\_ATK9,  
S\_FATT\_ATK10,  
S\_FATT\_PAIN,  
S\_FATT\_PAIN2,  
S\_FATT\_DIE1,  
S\_FATT\_DIE2,  
S\_FATT\_DIE3,  
S\_FATT\_DIE4,  
S\_FATT\_DIE5,  
S\_FATT\_DIE6,  
S\_FATT\_DIE7,  
S\_FATT\_DIE8,  
S\_FATT\_DIE9,  
S\_FATT\_DIE10,  
S\_FATT\_RAISE1,  
S\_FATT\_RAISE2,  
S\_FATT\_RAISE3,  
S\_FATT\_RAISE4,  
S\_FATT\_RAISE5,  
S\_FATT\_RAISE6,  
S\_FATT\_RAISE7,  
S\_FATT\_RAISE8,  
S\_CPOS\_STND,  
S\_CPOS\_STND2,  
S\_CPOS\_RUN1,  
S\_CPOS\_RUN2,  
S\_CPOS\_RUN3,  
S\_CPOS\_RUN4,  
S\_CPOS\_RUN5,  
S\_CPOS\_RUN6,  
S\_CPOS\_RUN7,  
S\_CPOS\_RUN8,  
S\_CPOS\_ATK1,  
S\_CPOS\_ATK2,  
S\_CPOS\_ATK3,  
S\_CPOS\_ATK4,

S\_CPOS\_PAIN,  
S\_CPOS\_PAIN2,  
S\_CPOS\_DIE1,  
S\_CPOS\_DIE2,  
S\_CPOS\_DIE3,  
S\_CPOS\_DIE4,  
S\_CPOS\_DIE5,  
S\_CPOS\_DIE6,  
S\_CPOS\_DIE7,  
S\_CPOS\_XDIE1,  
S\_CPOS\_XDIE2,  
S\_CPOS\_XDIE3,  
S\_CPOS\_XDIE4,  
S\_CPOS\_XDIE5,  
S\_CPOS\_XDIE6,  
S\_CPOS\_RAISE1,  
S\_CPOS\_RAISE2,  
S\_CPOS\_RAISE3,  
S\_CPOS\_RAISE4,  
S\_CPOS\_RAISE5,  
S\_CPOS\_RAISE6,  
S\_CPOS\_RAISE7,  
S\_TROO\_STND,  
S\_TROO\_STND2,  
S\_TROO\_RUN1,  
S\_TROO\_RUN2,  
S\_TROO\_RUN3,  
S\_TROO\_RUN4,  
S\_TROO\_RUN5,  
S\_TROO\_RUN6,  
S\_TROO\_RUN7,  
S\_TROO\_RUN8,  
S\_TROO\_ATK1,  
S\_TROO\_ATK2,  
S\_TROO\_ATK3,  
S\_TROO\_PAIN,  
S\_TROO\_PAIN2,  
S\_TROO\_DIE1,  
S\_TROO\_DIE2,  
S\_TROO\_DIE3,  
S\_TROO\_DIE4,  
S\_TROO\_DIE5,  
S\_TROO\_XDIE1,  
S\_TROO\_XDIE2,  
S\_TROO\_XDIE3,  
S\_TROO\_XDIE4,  
S\_TROO\_XDIE5,  
S\_TROO\_XDIE6,  
S\_TROO\_XDIE7,  
S\_TROO\_XDIE8,  
S\_TROO\_RAISE1,  
S\_TROO\_RAISE2,  
S\_TROO\_RAISE3,  
S\_TROO\_RAISE4,  
S\_TROO\_RAISE5,  
S\_SARG\_STND,  
S\_SARG\_STND2,  
S\_SARG\_RUN1,  
S\_SARG\_RUN2,  
S\_SARG\_RUN3,  
S\_SARG\_RUN4,  
S\_SARG\_RUN5,  
S\_SARG\_RUN6,  
S\_SARG\_RUN7,  
S\_SARG\_RUN8,



S\_SARG\_ATK1,  
S\_SARG\_ATK2,  
S\_SARG\_ATK3,  
S\_SARG\_PAIN,  
S\_SARG\_PAIN2,  
S\_SARG\_DIE1,  
S\_SARG\_DIE2,  
S\_SARG\_DIE3,  
S\_SARG\_DIE4,  
S\_SARG\_DIE5,  
S\_SARG\_DIE6,  
S\_SARG\_RAISE1,  
S\_SARG\_RAISE2,  
S\_SARG\_RAISE3,  
S\_SARG\_RAISE4,  
S\_SARG\_RAISE5,  
S\_SARG\_RAISE6,  
S\_HEAD\_STND,  
S\_HEAD\_RUN1,  
S\_HEAD\_ATK1,  
S\_HEAD\_ATK2,  
S\_HEAD\_ATK3,  
S\_HEAD\_PAIN,  
S\_HEAD\_PAIN2,  
S\_HEAD\_PAIN3,  
S\_HEAD\_DIE1,  
S\_HEAD\_DIE2,  
S\_HEAD\_DIE3,  
S\_HEAD\_DIE4,  
S\_HEAD\_DIE5,  
S\_HEAD\_DIE6,  
S\_HEAD\_RAISE1,  
S\_HEAD\_RAISE2,  
S\_HEAD\_RAISE3,  
S\_HEAD\_RAISE4,  
S\_HEAD\_RAISE5,  
S\_HEAD\_RAISE6,  
S\_BRBALL1,  
S\_BRBALL2,  
S\_BRBALLX1,  
S\_BRBALLX2,  
S\_BRBALLX3,  
S\_BOSS\_STND,  
S\_BOSS\_STND2,  
S\_BOSS\_RUN1,  
S\_BOSS\_RUN2,  
S\_BOSS\_RUN3,  
S\_BOSS\_RUN4,  
S\_BOSS\_RUN5,  
S\_BOSS\_RUN6,  
S\_BOSS\_RUN7,  
S\_BOSS\_RUN8,  
S\_BOSS\_ATK1,  
S\_BOSS\_ATK2,  
S\_BOSS\_ATK3,  
S\_BOSS\_PAIN,  
S\_BOSS\_PAIN2,  
S\_BOSS\_DIE1,  
S\_BOSS\_DIE2,  
S\_BOSS\_DIE3,  
S\_BOSS\_DIE4,  
S\_BOSS\_DIE5,  
S\_BOSS\_DIE6,  
S\_BOSS\_DIE7,  
S\_BOSS\_RAISE1,

S\_BOSS\_RAISE2,  
S\_BOSS\_RAISE3,  
S\_BOSS\_RAISE4,  
S\_BOSS\_RAISE5,  
S\_BOSS\_RAISE6,  
S\_BOSS\_RAISE7,  
S\_BOS2\_STND,  
S\_BOS2\_STND2,  
S\_BOS2\_RUN1,  
S\_BOS2\_RUN2,  
S\_BOS2\_RUN3,  
S\_BOS2\_RUN4,  
S\_BOS2\_RUN5,  
S\_BOS2\_RUN6,  
S\_BOS2\_RUN7,  
S\_BOS2\_RUN8,  
S\_BOS2\_ATK1,  
S\_BOS2\_ATK2,  
S\_BOS2\_ATK3,  
S\_BOS2\_PAIN,  
S\_BOS2\_PAIN2,  
S\_BOS2\_DIE1,  
S\_BOS2\_DIE2,  
S\_BOS2\_DIE3,  
S\_BOS2\_DIE4,  
S\_BOS2\_DIE5,  
S\_BOS2\_DIE6,  
S\_BOS2\_DIE7,  
S\_BOS2\_RAISE1,  
S\_BOS2\_RAISE2,  
S\_BOS2\_RAISE3,  
S\_BOS2\_RAISE4,  
S\_BOS2\_RAISE5,  
S\_BOS2\_RAISE6,  
S\_BOS2\_RAISE7,  
S\_SKULL\_STND,  
S\_SKULL\_STND2,  
S\_SKULL\_RUN1,  
S\_SKULL\_RUN2,  
S\_SKULL\_ATK1,  
S\_SKULL\_ATK2,  
S\_SKULL\_ATK3,  
S\_SKULL\_ATK4,  
S\_SKULL\_PAIN,  
S\_SKULL\_PAIN2,  
S\_SKULL\_DIE1,  
S\_SKULL\_DIE2,  
S\_SKULL\_DIE3,  
S\_SKULL\_DIE4,  
S\_SKULL\_DIE5,  
S\_SKULL\_DIE6,  
S\_SPID\_STND,  
S\_SPID\_STND2,  
S\_SPID\_RUN1,  
S\_SPID\_RUN2,  
S\_SPID\_RUN3,  
S\_SPID\_RUN4,  
S\_SPID\_RUN5,  
S\_SPID\_RUN6,  
S\_SPID\_RUN7,  
S\_SPID\_RUN8,  
S\_SPID\_RUN9,  
S\_SPID\_RUN10,  
S\_SPID\_RUN11,  
S\_SPID\_RUN12,

S\_SPID\_ATK1,  
S\_SPID\_ATK2,  
S\_SPID\_ATK3,  
S\_SPID\_ATK4,  
S\_SPID\_PAIN,  
S\_SPID\_PAIN2,  
S\_SPID\_DIE1,  
S\_SPID\_DIE2,  
S\_SPID\_DIE3,  
S\_SPID\_DIE4,  
S\_SPID\_DIE5,  
S\_SPID\_DIE6,  
S\_SPID\_DIE7,  
S\_SPID\_DIE8,  
S\_SPID\_DIE9,  
S\_SPID\_DIE10,  
S\_SPID\_DIE11,  
S\_BSPI\_STND,  
S\_BSPI\_STND2,  
S\_BSPI\_SIGHT,  
S\_BSPI\_RUN1,  
S\_BSPI\_RUN2,  
S\_BSPI\_RUN3,  
S\_BSPI\_RUN4,  
S\_BSPI\_RUN5,  
S\_BSPI\_RUN6,  
S\_BSPI\_RUN7,  
S\_BSPI\_RUN8,  
S\_BSPI\_RUN9,  
S\_BSPI\_RUN10,  
S\_BSPI\_RUN11,  
S\_BSPI\_RUN12,  
S\_BSPI\_ATK1,  
S\_BSPI\_ATK2,  
S\_BSPI\_ATK3,  
S\_BSPI\_ATK4,  
S\_BSPI\_PAIN,  
S\_BSPI\_PAIN2,  
S\_BSPI\_DIE1,  
S\_BSPI\_DIE2,  
S\_BSPI\_DIE3,  
S\_BSPI\_DIE4,  
S\_BSPI\_DIE5,  
S\_BSPI\_DIE6,  
S\_BSPI\_DIE7,  
S\_BSPI\_RAISE1,  
S\_BSPI\_RAISE2,  
S\_BSPI\_RAISE3,  
S\_BSPI\_RAISE4,  
S\_BSPI\_RAISE5,  
S\_BSPI\_RAISE6,  
S\_BSPI\_RAISE7,  
S\_ARACH\_PLAZ,  
S\_ARACH\_PLAZ2,  
S\_ARACH\_PLEX,  
S\_ARACH\_PLEX2,  
S\_ARACH\_PLEX3,  
S\_ARACH\_PLEX4,  
S\_ARACH\_PLEX5,  
S\_CYBER\_STND,  
S\_CYBER\_STND2,  
S\_CYBER\_RUN1,  
S\_CYBER\_RUN2,  
S\_CYBER\_RUN3,  
S\_CYBER\_RUN4,

S\_CYBER\_RUN5,  
S\_CYBER\_RUN6,  
S\_CYBER\_RUN7,  
S\_CYBER\_RUN8,  
S\_CYBER\_ATK1,  
S\_CYBER\_ATK2,  
S\_CYBER\_ATK3,  
S\_CYBER\_ATK4,  
S\_CYBER\_ATK5,  
S\_CYBER\_ATK6,  
S\_CYBER\_PAIN,  
S\_CYBER\_DIE1,  
S\_CYBER\_DIE2,  
S\_CYBER\_DIE3,  
S\_CYBER\_DIE4,  
S\_CYBER\_DIE5,  
S\_CYBER\_DIE6,  
S\_CYBER\_DIE7,  
S\_CYBER\_DIE8,  
S\_CYBER\_DIE9,  
S\_CYBER\_DIE10,  
S\_PAIN\_STND,  
S\_PAIN\_RUN1,  
S\_PAIN\_RUN2,  
S\_PAIN\_RUN3,  
S\_PAIN\_RUN4,  
S\_PAIN\_RUN5,  
S\_PAIN\_RUN6,  
S\_PAIN\_ATK1,  
S\_PAIN\_ATK2,  
S\_PAIN\_ATK3,  
S\_PAIN\_ATK4,  
S\_PAIN\_PAIN,  
S\_PAIN\_PAIN2,  
S\_PAIN\_DIE1,  
S\_PAIN\_DIE2,  
S\_PAIN\_DIE3,  
S\_PAIN\_DIE4,  
S\_PAIN\_DIE5,  
S\_PAIN\_DIE6,  
S\_PAIN\_RAISE1,  
S\_PAIN\_RAISE2,  
S\_PAIN\_RAISE3,  
S\_PAIN\_RAISE4,  
S\_PAIN\_RAISE5,  
S\_PAIN\_RAISE6,  
S\_SSWV\_STND,  
S\_SSWV\_STND2,  
S\_SSWV\_RUN1,  
S\_SSWV\_RUN2,  
S\_SSWV\_RUN3,  
S\_SSWV\_RUN4,  
S\_SSWV\_RUN5,  
S\_SSWV\_RUN6,  
S\_SSWV\_RUN7,  
S\_SSWV\_RUN8,  
S\_SSWV\_ATK1,  
S\_SSWV\_ATK2,  
S\_SSWV\_ATK3,  
S\_SSWV\_ATK4,  
S\_SSWV\_ATK5,  
S\_SSWV\_ATK6,  
S\_SSWV\_PAIN,  
S\_SSWV\_PAIN2,  
S\_SSWV\_DIE1,

S\_SSWV\_DIE2,  
S\_SSWV\_DIE3,  
S\_SSWV\_DIE4,  
S\_SSWV\_DIE5,  
S\_SSWV\_XDIE1,  
S\_SSWV\_XDIE2,  
S\_SSWV\_XDIE3,  
S\_SSWV\_XDIE4,  
S\_SSWV\_XDIE5,  
S\_SSWV\_XDIE6,  
S\_SSWV\_XDIE7,  
S\_SSWV\_XDIE8,  
S\_SSWV\_XDIE9,  
S\_SSWV\_RAISE1,  
S\_SSWV\_RAISE2,  
S\_SSWV\_RAISE3,  
S\_SSWV\_RAISE4,  
S\_SSWV\_RAISE5,  
S\_KEENSTND,  
S\_COMMKEEN,  
S\_COMMKEEN2,  
S\_COMMKEEN3,  
S\_COMMKEEN4,  
S\_COMMKEEN5,  
S\_COMMKEEN6,  
S\_COMMKEEN7,  
S\_COMMKEEN8,  
S\_COMMKEEN9,  
S\_COMMKEEN10,  
S\_COMMKEEN11,  
S\_COMMKEEN12,  
S\_KEENPAIN,  
S\_KEENPAIN2,  
S\_BRAIN,  
S\_BRAIN\_PAIN,  
S\_BRAIN\_DIE1,  
S\_BRAIN\_DIE2,  
S\_BRAIN\_DIE3,  
S\_BRAIN\_DIE4,  
S\_BRAINEYE,  
S\_BRAINEYESEE,  
S\_BRAINEYE1,  
S\_SPAWN1,  
S\_SPAWN2,  
S\_SPAWN3,  
S\_SPAWN4,  
S\_SPAWNFIRE1,  
S\_SPAWNFIRE2,  
S\_SPAWNFIRE3,  
S\_SPAWNFIRE4,  
S\_SPAWNFIRE5,  
S\_SPAWNFIRE6,  
S\_SPAWNFIRE7,  
S\_SPAWNFIRE8,  
S\_BRAINEXPLODE1,  
S\_BRAINEXPLODE2,  
S\_BRAINEXPLODE3,  
S\_ARM1,  
S\_ARM1A,  
S\_ARM2,  
S\_ARM2A,  
S\_BAR1,  
S\_BAR2,  
S\_BEXP,  
S\_BEXP2,

S\_BEXP3,  
S\_BEXP4,  
S\_BEXP5,  
S\_BBAR1,  
S\_BBAR2,  
S\_BBAR3,  
S\_BON1,  
S\_BON1A,  
S\_BON1B,  
S\_BON1C,  
S\_BON1D,  
S\_BON1E,  
S\_BON2,  
S\_BON2A,  
S\_BON2B,  
S\_BON2C,  
S\_BON2D,  
S\_BON2E,  
S\_BKEY,  
S\_BKEY2,  
S\_RKEY,  
S\_RKEY2,  
S\_YKEY,  
S\_YKEY2,  
S\_BSKULL,  
S\_BSKULL2,  
S\_RSKULL,  
S\_RSKULL2,  
S\_YSKULL,  
S\_YSKULL2,  
S\_STIM,  
S\_MEDI,  
S\_SOUL,  
S\_SOUL2,  
S\_SOUL3,  
S\_SOUL4,  
S\_SOUL5,  
S\_SOUL6,  
S\_PINV,  
S\_PINV2,  
S\_PINV3,  
S\_PINV4,  
S\_PSTR,  
S\_PINS,  
S\_PINS2,  
S\_PINS3,  
S\_PINS4,  
S\_MEGA,  
S\_MEGA2,  
S\_MEGA3,  
S\_MEGA4,  
S\_SUIT,  
S\_PMAP,  
S\_PMAP2,  
S\_PMAP3,  
S\_PMAP4,  
S\_PMAP5,  
S\_PMAP6,  
S\_PVIS,  
S\_PVIS2,  
S\_CLIP,  
S\_AMMO,  
S\_ROCK,  
S\_BROK,  
S\_CELL,

S\_CELP,  
S\_SHEL,  
S\_SBOX,  
S\_BPAK,  
S\_BFUG,  
S\_MGUN,  
S\_CSAW,  
S\_LAUN,  
S\_PLAS,  
S\_SHOT,  
S\_SHOT2,  
S\_COLU,  
S\_STALAG,  
S\_BLOODYTWITCH,  
S\_BLOODYTWITCH2,  
S\_BLOODYTWITCH3,  
S\_BLOODYTWITCH4,  
S\_DEADTORSO,  
S\_DEADBOTTOM,  
S\_HEADSONSTICK,  
S\_GIBS,  
S\_HEADONASTICK,  
S\_HEADCANDLES,  
S\_HEADCANDLES2,  
S\_DEADSTICK,  
S\_LIVESTICK,  
S\_LIVESTICK2,  
S\_MEAT2,  
S\_MEAT3,  
S\_MEAT4,  
S\_MEAT5,  
S\_STALAGTITE,  
S\_TALLGRNCOL,  
S\_SHRTGRNCOL,  
S\_TALLREDCOL,  
S\_SHRTREDCOL,  
S\_CANDLESTIK,  
S\_CANDELABRA,  
S\_SKULLCOL,  
S\_TORCHTREE,  
S\_BIGTREE,  
S\_TECHPILLAR,  
S\_EVILEYE,  
S\_EVILEYE2,  
S\_EVILEYE3,  
S\_EVILEYE4,  
S\_FLOATSKULL,  
S\_FLOATSKULL2,  
S\_FLOATSKULL3,  
S\_HEARTCOL,  
S\_HEARTCOL2,  
S\_BLUETORCH,  
S\_BLUETORCH2,  
S\_BLUETORCH3,  
S\_BLUETORCH4,  
S\_GREENTORCH,  
S\_GREENTORCH2,  
S\_GREENTORCH3,  
S\_GREENTORCH4,  
S\_REDTORCH,  
S\_REDTORCH2,  
S\_REDTORCH3,  
S\_REDTORCH4,  
S\_BTORCHSHRT,  
S\_BTORCHSHRT2,

```

S_BTORCHSHRT3,
S_BTORCHSHRT4,
S_GTORCHSHRT,
S_GTORCHSHRT2,
S_GTORCHSHRT3,
S_GTORCHSHRT4,
S_RTORCHSHRT,
S_RTORCHSHRT2,
S_RTORCHSHRT3,
S_RTORCHSHRT4,
S_HANGNOGUTS,
S_HANGBNOBRAIN,
S_HANGTLOOKDN,
S_HANGTSKULL,
S_HANGTLOOKUP,
S_HANGTNOBRAIN,
S_COLONGIBS,
S_SMALLPOOL,
S_BRAINSTEM,
S_TECHLAMP,
S_TECHLAMP2,
S_TECHLAMP3,
S_TECHLAMP4,
S_TECH2LAMP,
S_TECH2LAMP2,
S_TECH2LAMP3,
S_TECH2LAMP4,
NUMSTATES
} statenum_t;

```

```

typedef struct
{
    spritenum_t    sprite;
    long           frame;
    long           tics;
    // void         (*action) ();
    actionf_t      action;
    statenum_t     nextstate;
    long           misc1, misc2;
} state_t;

```

```

extern state_t    states[NUMSTATES];
extern char *sprnames[NUMSPRITES];

```

```

typedef enum {
    MT_PLAYER,
    MT_POSSESSED,
    MT_SHOTGUY,
    MT_VILE,
    MT_FIRE,
    MT_UNDEAD,
    MT_TRACER,
    MT_SMOKE,
    MT_FATSO,
    MT_FATSHOT,
    MT_CHAINGUY,
    MT_TROOP,
    MT_SERGEANT,
    MT_SHADOWS,
    MT_HEAD,
    MT_BRUISER,
    MT_BRUISERSHOT,

```



MT\_KNIGHT,  
MT\_SKULL,  
MT\_SPIDER,  
MT\_BABY,  
MT\_CYBORG,  
MT\_PAIN,  
MT\_WOLFSS,  
MT\_KEEN,  
MT\_BOSSBRAIN,  
MT\_BOSSSPIT,  
MT\_BOSSTARGET,  
MT\_SPAWNSHOT,  
MT\_SPAWNFIRE,  
MT\_BARREL,  
MT\_TROOPSHOT,  
MT\_HEADSHOT,  
MT\_ROCKET,  
MT\_PLASMA,  
MT\_BFG,  
MT\_ARACHPLAZ,  
MT\_PUFF,  
MT\_BLOOD,  
MT\_TFOG,  
MT\_IFOG,  
MT\_TELEPORTMAN,  
MT\_EXTRABFG,  
MT\_MISCO,  
MT\_MISC1,  
MT\_MISC2,  
MT\_MISC3,  
MT\_MISC4,  
MT\_MISC5,  
MT\_MISC6,  
MT\_MISC7,  
MT\_MISC8,  
MT\_MISC9,  
MT\_MISC10,  
MT\_MISC11,  
MT\_MISC12,  
MT\_INV,  
MT\_MISC13,  
MT\_INS,  
MT\_MISC14,  
MT\_MISC15,  
MT\_MISC16,  
MT\_MEGA,  
MT\_CLIP,  
MT\_MISC17,  
MT\_MISC18,  
MT\_MISC19,  
MT\_MISC20,  
MT\_MISC21,  
MT\_MISC22,  
MT\_MISC23,  
MT\_MISC24,  
MT\_MISC25,  
MT\_CHAINGUN,  
MT\_MISC26,  
MT\_MISC27,  
MT\_MISC28,  
MT\_SHOTGUN,  
MT\_SUPERSHOTGUN,  
MT\_MISC29,  
MT\_MISC30,  
MT\_MISC31,

```

MT_MISC32,
MT_MISC33,
MT_MISC34,
MT_MISC35,
MT_MISC36,
MT_MISC37,
MT_MISC38,
MT_MISC39,
MT_MISC40,
MT_MISC41,
MT_MISC42,
MT_MISC43,
MT_MISC44,
MT_MISC45,
MT_MISC46,
MT_MISC47,
MT_MISC48,
MT_MISC49,
MT_MISC50,
MT_MISC51,
MT_MISC52,
MT_MISC53,
MT_MISC54,
MT_MISC55,
MT_MISC56,
MT_MISC57,
MT_MISC58,
MT_MISC59,
MT_MISC60,
MT_MISC61,
MT_MISC62,
MT_MISC63,
MT_MISC64,
MT_MISC65,
MT_MISC66,
MT_MISC67,
MT_MISC68,
MT_MISC69,
MT_MISC70,
MT_MISC71,
MT_MISC72,
MT_MISC73,
MT_MISC74,
MT_MISC75,
MT_MISC76,
MT_MISC77,
MT_MISC78,
MT_MISC79,
MT_MISC80,
MT_MISC81,
MT_MISC82,
MT_MISC83,
MT_MISC84,
MT_MISC85,
MT_MISC86,
NUMMOBJTYPES

} mobjtype_t;

typedef struct
{
    int        doomednum;
    int        spawnstate;
    int        spawnhealth;
    int        seestate;

```

```

    int      seesound;
    int      reactiontime;
    int      attacksound;
    int      painstate;
    int      painchance;
    int      painsound;
    int      meleestate;
    int      missilestate;
    int      deathstate;
    int      xdeathstate;
    int      deathsound;
    int      speed;
    int      radius;
    int      height;
    int      mass;
    int      damage;
    int      activesound;
    int      flags;
    int      raisestate;

} mobjinfo_t;

extern mobjinfo_t mobjinfo[NUMMOBJTYPES];

#endif
//-----
//
// $Log:$
//
//-----

```

### 9.3 p\_ceiling.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION: Ceiling animation (lowering, crushing, raising)
//
//-----

static const char
rcsid[] = "$Id: p_ceiling.c,v 1.4 1997/02/03 16:47:53 b1 Exp $";

#include "z_zone.h"
#include "doomdef.h"
#include "p_local.h"

#include "s_sound.h"

```

```

// State.
#include "doomstat.h"
#include "r_state.h"

// Data.
#include "sounds.h"

//
// CEILINGS
//

ceiling_t*      activeceilings[MAXCEILINGS];

//
// T_MoveCeiling
//

void T_MoveCeiling (ceiling_t* ceiling)
{
    result_e      res;

    switch(ceiling->direction)
    {
        case 0:
            // IN STASIS
            break;
        case 1:
            // UP
            res = T_MovePlane(ceiling->sector,
                             ceiling->speed,
                             ceiling->topheight,
                             false,1,ceiling->direction);

            if (!(leveltime&7))
            {
                switch(ceiling->type)
                {
                    case silentCrushAndRaise:
                        break;
                    default:
                        S_StartSound((mobj_t *)&ceiling->sector->soundorg,
                                    sfx_stnmov);

                        // ?
                        break;
                }
            }

            if (res == pastdest)
            {
                switch(ceiling->type)
                {
                    case raiseToHighest:
                        P_RemoveActiveCeiling(ceiling);
                        break;

                    case silentCrushAndRaise:
                        S_StartSound((mobj_t *)&ceiling->sector->soundorg,
                                    sfx_pstop);
                    case fastCrushAndRaise:
                    case crushAndRaise:
                        ceiling->direction = -1;
                        break;
                }
            }
        }
    }

```

```

        default:
            break;
    }

}
break;

case -1:
    // DOWN
    res = T_MovePlane(ceiling->sector,
                      ceiling->speed,
                      ceiling->bottomheight,
                      ceiling->crush,1,ceiling->direction);

    if (!(leveltime&7))
    {
        switch(ceiling->type)
        {
            case silentCrushAndRaise: break;
            default:
                S_StartSound((mobj_t *)&ceiling->sector->soundorg,
                             sfx_stnmov);
        }
    }

    if (res == pastdest)
    {
        switch(ceiling->type)
        {
            case silentCrushAndRaise:
                S_StartSound((mobj_t *)&ceiling->sector->soundorg,
                             sfx_pstop);
            case crushAndRaise:
                ceiling->speed = CEILSPEED;
            case fastCrushAndRaise:
                ceiling->direction = 1;
                break;

            case lowerAndCrush:
            case lowerToFloor:
                P_RemoveActiveCeiling(ceiling);
                break;

            default:
                break;
        }
    }
    else // ( res != pastdest )
    {
        if (res == crushed)
        {
            switch(ceiling->type)
            {
                case silentCrushAndRaise:
                case crushAndRaise:
                case lowerAndCrush:
                    ceiling->speed = CEILSPEED / 8;
                    break;

                default:
                    break;
            }
        }
    }
}

```

```

        break;
    }
}

//
// EV_DoCeiling
// Move a ceiling up/down and all around!
//
int
EV_DoCeiling
( line_t*      line,
  ceiling_e    type )
{
    int          secnum;
    int          rtn;
    sector_t*    sec;
    ceiling_t*    ceiling;

    secnum = -1;
    rtn = 0;

    //      Reactivate in-stasis ceilings...for certain types.
    switch(type)
    {
        case fastCrushAndRaise:
        case silentCrushAndRaise:
        case crushAndRaise:
            P_ActivateInStasisCeiling(line);
        default:
            break;
    }

    while ((secnum = P_FindSectorFromLineTag(line,secnum)) >= 0)
    {
        sec = &sectors[secnum];
        if (sec->specialdata)
            continue;

        // new door thinker
        rtn = 1;
        ceiling = Z_Malloc (sizeof(*ceiling), PU_LEVSPEC, 0);
        P_AddThinker (&ceiling->thinker);
        sec->specialdata = ceiling;
        ceiling->thinker.function.acp1 = (actionf_p1)T_MoveCeiling;
        ceiling->sector = sec;
        ceiling->crush = false;

        switch(type)
        {
            case fastCrushAndRaise:
                ceiling->crush = true;
                ceiling->topheight = sec->ceilingheight;
                ceiling->bottomheight = sec->floorheight + (8*FRACUNIT);
                ceiling->direction = -1;
                ceiling->speed = CEILSPEED * 2;
                break;

            case silentCrushAndRaise:
            case crushAndRaise:
                ceiling->crush = true;
                ceiling->topheight = sec->ceilingheight;
            case lowerAndCrush:
            case lowerToFloor:
                ceiling->bottomheight = sec->floorheight;

```

```

        if (type != lowerToFloor)
            ceiling->bottomheight += 8*FRACUNIT;
        ceiling->direction = -1;
        ceiling->speed = CEILSPEED;
        break;

    case raiseToHighest:
        ceiling->topheight = P_FindHighestCeilingSurrounding(sec);
        ceiling->direction = 1;
        ceiling->speed = CEILSPEED;
        break;
}

    ceiling->tag = sec->tag;
    ceiling->type = type;
    P_AddActiveCeiling(ceiling);
}
return rtn;
}

```

```

//
// Add an active ceiling
//
void P_AddActiveCeiling(ceiling_t* c)
{
    int i;

    for (i = 0; i < MAXCEILINGS;i++)
    {
        if (activeceilings[i] == NULL)
        {
            activeceilings[i] = c;
            return;
        }
    }
}

```

```

//
// Remove a ceiling's thinker
//
void P_RemoveActiveCeiling(ceiling_t* c)
{
    int i;

    for (i = 0;i < MAXCEILINGS;i++)
    {
        if (activeceilings[i] == c)
        {
            activeceilings[i]->sector->specialdata = NULL;
            P_RemoveThinker (&activeceilings[i]->thinker);
            activeceilings[i] = NULL;
            break;
        }
    }
}

```

```

//
// Restart a ceiling that's in-stasis
//
void P_ActivateInStasisCeiling(line_t* line)

```

```

{
    int                i;

    for (i = 0; i < MAXCEILINGS; i++)
    {
        if (activeceilings[i]
            && (activeceilings[i]->tag == line->tag)
            && (activeceilings[i]->direction == 0))
        {
            activeceilings[i]->direction = activeceilings[i]->olddirection;
            activeceilings[i]->thinker.function.acp1
                = (actionf_p1)T_MoveCeiling;
        }
    }
}

//
// EV_CeilingCrushStop
// Stop a ceiling from crushing!
//
int                EV_CeilingCrushStop(line_t        *line)
{
    int                i;
    int                rtn;

    rtn = 0;
    for (i = 0; i < MAXCEILINGS; i++)
    {
        if (activeceilings[i]
            && (activeceilings[i]->tag == line->tag)
            && (activeceilings[i]->direction != 0))
        {
            activeceilings[i]->olddirection = activeceilings[i]->direction;
            activeceilings[i]->thinker.function.acv = (actionf_v)NULL;
            activeceilings[i]->direction = 0;                // in-stasis
            rtn = 1;
        }
    }

    return rtn;
}

```

## 9.4 p\_doors.c

```

// Emacs style mode select    -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$

```



```

//
// DESCRIPTION: Door animation code (opening/closing)
//
//-----

static const char
rcsid[] = "$Id: p_doors.c,v 1.4 1997/02/03 16:47:53 b1 Exp $";

#include "z_zone.h"
#include "doomdef.h"
#include "p_local.h"

#include "s_sound.h"

// State.
#include "doomstat.h"
#include "r_state.h"

// Data.
#include "dstrings.h"
#include "sounds.h"

#if 0
//
// Sliding door frame information
//
slidename_t      slideFrameNames[MAXSLIDEDOORS] =
{
    {"GDOORF1","GDOORF2","GDOORF3","GDOORF4",      // front
     "GDOORB1","GDOORB2","GDOORB3","GDOORB4"},      // back

    {"\0","\0","\0","\0"}
};
#endif

//
// VERTICAL DOORS
//

//
// T_VerticalDoor
//
void T_VerticalDoor (vldoor_t* door)
{
    result_e      res;

    switch(door->direction)
    {
    case 0:
        // WAITING
        if (!--door->topcountdown)
        {
            switch(door->type)
            {
            case blazeRaise:
                door->direction = -1; // time to go back down
                S_StartSound((mobj_t *)&door->sector->soundorg,
                             sfx_bdcls);

                break;

            case normal:
                door->direction = -1; // time to go back down
            }
        }
    }
}

```

```

        S_StartSound((mobj_t *)&door->sector->soundorg,
                    sfx_dorcls);

        break;

    case close30ThenOpen:
        door->direction = 1;
        S_StartSound((mobj_t *)&door->sector->soundorg,
                    sfx_doropn);

        break;

    default:
        break;
}
}
break;

case 2:
    // INITIAL WAIT
    if (!--door->topcountdown)
    {
        switch(door->type)
        {
            case raiseIn5Mins:
                door->direction = 1;
                door->type = normal;
                S_StartSound((mobj_t *)&door->sector->soundorg,
                            sfx_doropn);

                break;

            default:
                break;
        }
    }
    break;

case -1:
    // DOWN
    res = T_MovePlane(door->sector,
                    door->speed,
                    door->sector->floorheight,
                    false,1,door->direction);
    if (res == pastdest)
    {
        switch(door->type)
        {
            case blazeRaise:
            case blazeClose:
                door->sector->specialdata = NULL;
                P_RemoveThinker (&door->thinker); // unlink and free
                S_StartSound((mobj_t *)&door->sector->soundorg,
                            sfx_bdcls);

                break;

            case normal:
            case close:
                door->sector->specialdata = NULL;
                P_RemoveThinker (&door->thinker); // unlink and free
                break;

            case close30ThenOpen:
                door->direction = 0;
                door->topcountdown = 35*30;
                break;

            default:

```

```

        break;
    }
}
else if (res == crushed)
{
    switch(door->type)
    {
        case blazeClose:
        case close:                // DO NOT GO BACK UP!
            break;

        default:
            door->direction = 1;
            S_StartSound((mobj_t *)&door->sector->soundorg,
                        sfx_doropn);
            break;
    }
}
break;

case 1:
    // UP
    res = T_MovePlane(door->sector,
                      door->speed,
                      door->topheight,
                      false,1,door->direction);

    if (res == pastdest)
    {
        switch(door->type)
        {
            case blazeRaise:
            case normal:
                door->direction = 0; // wait at top
                door->topcountdown = door->topwait;
                break;

            case close30ThenOpen:
            case blazeOpen:
            case open:
                door->sector->specialdata = NULL;
                P_RemoveThinker (&door->thinker); // unlink and free
                break;

            default:
                break;
        }
    }
    break;
}
}

//
// EV_DoLockedDoor
// Move a locked door up/down
//

int
EV_DoLockedDoor
( line_t*      line,
  vldoor_e     type,
  mobj_t*      thing )
{
    player_t*   p;

```

```

p = thing->player;

if (!p)
    return 0;

switch(line->special)
{
case 99:          // Blue Lock
case 133:
    if ( !p )
        return 0;
    if (!p->cards[it_bluecard] && !p->cards[it_blueskull])
    {
        p->message = PD_BLUE0;
        S_StartSound(NULL,sfx_oof);
        return 0;
    }
    break;

case 134: // Red Lock
case 135:
    if ( !p )
        return 0;
    if (!p->cards[it_redcard] && !p->cards[it_redskull])
    {
        p->message = PD_REDO;
        S_StartSound(NULL,sfx_oof);
        return 0;
    }
    break;

case 136:          // Yellow Lock
case 137:
    if ( !p )
        return 0;
    if (!p->cards[it_yellowcard] &&
        !p->cards[it_yellowskull])
    {
        p->message = PD_YELLOW0;
        S_StartSound(NULL,sfx_oof);
        return 0;
    }
    break;
}

return EV_DoDoor(line,type);
}

int
EV_DoDoor
( line_t*      line,
  vldoor_e     type )
{
    int          secnum,rtn;
    sector_t*    sec;
    vldoor_t*    door;

    secnum = -1;
    rtn = 0;

    while ((secnum = P_FindSectorFromLineTag(line,secnum)) >= 0)
    {
        sec = &sectors[secnum];

```

```

if (sec->specialdata)
    continue;

// new door thinker
rtn = 1;
door = Z_Malloc (sizeof(*door), PU_LEVSPEC, 0);
P_AddThinker (&door->thinker);
sec->specialdata = door;

door->thinker.function.acp1 = (actionf_p1) T_VerticalDoor;
door->sector = sec;
door->type = type;
door->topwait = VDOORWAIT;
door->speed = VDOORSPEED;

switch(type)
{
case blazeClose:
    door->topheight = P_FindLowestCeilingSurrounding(sec);
    door->topheight -= 4*FRACUNIT;
    door->direction = -1;
    door->speed = VDOORSPEED * 4;
    S_StartSound((mobj_t *)&door->sector->soundorg,
                  sfx_bdcls);

    break;

case close:
    door->topheight = P_FindLowestCeilingSurrounding(sec);
    door->topheight -= 4*FRACUNIT;
    door->direction = -1;
    S_StartSound((mobj_t *)&door->sector->soundorg,
                  sfx_dorcls);

    break;

case close30ThenOpen:
    door->topheight = sec->ceilingheight;
    door->direction = -1;
    S_StartSound((mobj_t *)&door->sector->soundorg,
                  sfx_dorcls);

    break;

case blazeRaise:
case blazeOpen:
    door->direction = 1;
    door->topheight = P_FindLowestCeilingSurrounding(sec);
    door->topheight -= 4*FRACUNIT;
    door->speed = VDOORSPEED * 4;
    if (door->topheight != sec->ceilingheight)
        S_StartSound((mobj_t *)&door->sector->soundorg,
                      sfx_bdopn);

    break;

case normal:
case open:
    door->direction = 1;
    door->topheight = P_FindLowestCeilingSurrounding(sec);
    door->topheight -= 4*FRACUNIT;
    if (door->topheight != sec->ceilingheight)
        S_StartSound((mobj_t *)&door->sector->soundorg,
                      sfx_doropn);

    break;

default:
    break;
}

```

```

    }

    }
    return rtn;
}

//
// EV_VerticalDoor : open a door manually, no tag value
//
void
EV_VerticalDoor
( line_t*      line,
  mobj_t*      thing )
{
    player_t*    player;
    int          secnum;
    sector_t*    sec;
    vldoor_t*    door;
    int          side;

    side = 0;      // only front sides can be used

    //      Check for locks
    player = thing->player;

    switch(line->special)
    {
        case 26: // Blue Lock
        case 32:
            if ( !player )
                return;

            if (!player->cards[it_bluecard] && !player->cards[it_blueskull])
            {
                player->message = PD_BLUEK;
                S_StartSound(NULL,sfx_oof);
                return;
            }
            break;

        case 27: // Yellow Lock
        case 34:
            if ( !player )
                return;

            if (!player->cards[it_yellowcard] &&
                !player->cards[it_yellowskull])
            {
                player->message = PD_YELLOWK;
                S_StartSound(NULL,sfx_oof);
                return;
            }
            break;

        case 28: // Red Lock
        case 33:
            if ( !player )
                return;

            if (!player->cards[it_redcard] && !player->cards[it_redskull])
            {
                player->message = PD_REDK;
                S_StartSound(NULL,sfx_oof);
                return;
            }
    }
}

```

```

    }
    break;
}

// if the sector has an active thinker, use it
sec = sides[ line->sidenum[side^1]] .sector;
secnum = sec-sectors;

if (sec->specialdata)
{
    door = sec->specialdata;
    switch(line->special)
    {
        case 1: // ONLY FOR "RAISE" DOORS, NOT "OPEN"s
        case 26:
        case 27:
        case 28:
        case 117:
            if (door->direction == -1)
                door->direction = 1; // go back up
            else
            {
                if (!thing->player)
                    return; // JDC: bad guys never close doors

                door->direction = -1; // start going down immediately
            }
            return;
        }
    }

// for proper sound
switch(line->special)
{
    case 117: // BLAZING DOOR RAISE
    case 118: // BLAZING DOOR OPEN
        S_StartSound((mobj_t *)&sec->soundorg,sfx_bdopn);
        break;

    case 1: // NORMAL DOOR SOUND
    case 31:
        S_StartSound((mobj_t *)&sec->soundorg,sfx_doropn);
        break;

    default: // LOCKED DOOR SOUND
        S_StartSound((mobj_t *)&sec->soundorg,sfx_doropn);
        break;
}

// new door thinker
door = Z_Malloc (sizeof(*door), PU_LEVSPEC, 0);
P_AddThinker (&door->thinker);
sec->specialdata = door;
door->thinker.function.acp1 = (actionf_p1) T_VerticalDoor;
door->sector = sec;
door->direction = 1;
door->speed = VDOORSPEED;
door->topwait = VDOORWAIT;

switch(line->special)
{
    case 1:
    case 26:
    case 27:

```

```

    case 28:
        door->type = normal;
        break;

    case 31:
    case 32:
    case 33:
    case 34:
        door->type = open;
        line->special = 0;
        break;

    case 117:        // blazing door raise
        door->type = blazeRaise;
        door->speed = VDOORSPEED*4;
        break;
    case 118:        // blazing door open
        door->type = blazeOpen;
        line->special = 0;
        door->speed = VDOORSPEED*4;
        break;
}

// find the top and bottom of the movement range
door->topheight = P_FindLowestCeilingSurrounding(sec);
door->topheight -= 4*FRACUNIT;
}

//
// Spawn a door that closes after 30 seconds
//
void P_SpawnDoorCloseIn30 (sector_t* sec)
{
    vldoor_t*      door;

    door = Z_Malloc ( sizeof(*door), PU_LEVSPEC, 0);

    P_AddThinker (&door->thinker);

    sec->specialdata = door;
    sec->special = 0;

    door->thinker.function.acp1 = (actionf_p1)T_VerticalDoor;
    door->sector = sec;
    door->direction = 0;
    door->type = normal;
    door->speed = VDOORSPEED;
    door->topcountdown = 30 * 35;
}

//
// Spawn a door that opens after 5 minutes
//
void
P_SpawnDoorRaiseIn5Mins
( sector_t*      sec,
  int            secnum )
{
    vldoor_t*      door;

    door = Z_Malloc ( sizeof(*door), PU_LEVSPEC, 0);

    P_AddThinker (&door->thinker);

```



```

sec->specialdata = door;
sec->special = 0;

door->thinker.function.acp1 = (actionf_p1)T_VerticalDoor;
door->sector = sec;
door->direction = 2;
door->type = raiseIn5Mins;
door->speed = VDOORSPEED;
door->topheight = P_FindLowestCeilingSurrounding(sec);
door->topheight -= 4*FRACUNIT;
door->topwait = VDOORWAIT;
door->topcountdown = 5 * 60 * 35;
}

```

```

// UNUSED
// Separate into p_sliddoor.c?

```

```

#if 0 // ABANDONED TO THE MISTS OF TIME!!!
//
// EV_SlidingDoor : slide a door horizontally
// (animate midtexture, then set noblocking line)
//

```

```

slideframe_t slideFrames[MAXSLIDEDOORS];

```

```

void P_InitSlidingDoorFrames(void)
{

```

```

    int          i;
    int          f1;
    int          f2;
    int          f3;
    int          f4;

```

```

    // DOOM II ONLY...
    if ( gamemode != commercial)
        return;

```

```

    for (i = 0; i < MAXSLIDEDOORS; i++)
    {
        if (!slideFrameNames[i].frontFrame1[0])
            break;

```

```

        f1 = R_TextureNumForName(slideFrameNames[i].frontFrame1);
        f2 = R_TextureNumForName(slideFrameNames[i].frontFrame2);
        f3 = R_TextureNumForName(slideFrameNames[i].frontFrame3);
        f4 = R_TextureNumForName(slideFrameNames[i].frontFrame4);

```

```

        slideFrames[i].frontFrames[0] = f1;
        slideFrames[i].frontFrames[1] = f2;
        slideFrames[i].frontFrames[2] = f3;
        slideFrames[i].frontFrames[3] = f4;

```

```

        f1 = R_TextureNumForName(slideFrameNames[i].backFrame1);
        f2 = R_TextureNumForName(slideFrameNames[i].backFrame2);
        f3 = R_TextureNumForName(slideFrameNames[i].backFrame3);
        f4 = R_TextureNumForName(slideFrameNames[i].backFrame4);

```

```

        slideFrames[i].backFrames[0] = f1;
        slideFrames[i].backFrames[1] = f2;
        slideFrames[i].backFrames[2] = f3;
        slideFrames[i].backFrames[3] = f4;

```

```

    }

```

```

}

//
// Return index into "slideFrames" array
// for which door type to use
//
int P_FindSlidingDoorType(line_t*      line)
{
    int          i;
    int          val;

    for (i = 0; i < MAXSLIDEDOORS; i++)
    {
        val = sides[line->sidenum[0]].midtexture;
        if (val == slideFrames[i].frontFrames[0])
            return i;
    }

    return -1;
}

void T_SlidingDoor (slidedoor_t*      door)
{
    switch(door->status)
    {
        case sd_opening:
            if (!door->timer--)
            {
                if (++door->frame == SNUMFRAMES)
                {
                    // IF DOOR IS DONE OPENING...
                    sides[door->line->sidenum[0]].midtexture = 0;
                    sides[door->line->sidenum[1]].midtexture = 0;
                    door->line->flags &= ML_BLOCKING^0xff;

                    if (door->type == sdt_openOnly)
                    {
                        door->frontsector->specialdata = NULL;
                        P_RemoveThinker (&door->thinker);
                        break;
                    }

                    door->timer = SDOORWAIT;
                    door->status = sd_waiting;
                }
            }
            else
            {
                // IF DOOR NEEDS TO ANIMATE TO NEXT FRAME...
                door->timer = SWAITTICS;

                sides[door->line->sidenum[0]].midtexture =
                    slideFrames[door->whichDoorIndex].
                    frontFrames[door->frame];
                sides[door->line->sidenum[1]].midtexture =
                    slideFrames[door->whichDoorIndex].
                    backFrames[door->frame];
            }
        }
        break;

        case sd_waiting:
            // IF DOOR IS DONE WAITING...
            if (!door->timer--)
            {

```

```

        // CAN DOOR CLOSE?
        if (door->frontsector->thinglist != NULL ||
            door->backsector->thinglist != NULL)
        {
            door->timer = SDOORWAIT;
            break;
        }

        //door->frame = SNUMFRAMES-1;
        door->status = sd_closing;
        door->timer = SWAITTICS;
    }
    break;

case sd_closing:
    if (!door->timer--)
    {
        if (--door->frame < 0)
        {
            // IF DOOR IS DONE CLOSING...
            door->line->flags |= ML_BLOCKING;
            door->frontsector->specialdata = NULL;
            P_RemoveThinker (&door->thinker);
            break;
        }
        else
        {
            // IF DOOR NEEDS TO ANIMATE TO NEXT FRAME...
            door->timer = SWAITTICS;

            sides[door->line->sidenum[0]].midtexture =
                slideFrames[door->whichDoorIndex].
                frontFrames[door->frame];
            sides[door->line->sidenum[1]].midtexture =
                slideFrames[door->whichDoorIndex].
                backFrames[door->frame];
        }
    }
    break;
}
}

```

```

void
EV_SlidingDoor
( line_t*      line,
  mobj_t*      thing )
{
    sector_t*      sec;
    slidedoor_t*    door;

    // DOOM II ONLY...
    if (gamemode != commercial)
        return;

    // Make sure door isn't already being animated
    sec = line->frontsector;
    door = NULL;
    if (sec->specialdata)
    {
        if (!thing->player)
            return;

        door = sec->specialdata;
    }
}

```

```

    if (door->type == sdt_openAndClose)
    {
        if (door->status == sd_waiting)
            door->status = sd_closing;
    }
    else
        return;
}

// Init sliding door vars
if (!door)
{
    door = Z_Malloc (sizeof(*door), PU_LEVSPEC, 0);
    P_AddThinker (&door->thinker);
    sec->specialdata = door;

    door->type = sdt_openAndClose;
    door->status = sd_opening;
    door->whichDoorIndex = P_FindSlidingDoorType(line);

    if (door->whichDoorIndex < 0)
        I_Error("EV_SlidingDoor: Can't use texture for sliding door!");

    door->frontsector = sec;
    door->backsector = line->backsector;
    door->thinker.function = T_SlidingDoor;
    door->timer = SWAITTICS;
    door->frame = 0;
    door->line = line;
}
}
#endif

```

## 9.5 p\_enemy.c

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Enemy thinking, AI.
//     Action Pointer Functions
//     that are associated with states/frames.
//
//-----

static const char
rcsid[] = "$Id: p_enemy.c,v 1.5 1997/02/03 22:45:11 b1 Exp $";

#include <stdlib.h>

```

```

#include "m_random.h"
#include "i_system.h"

#include "doomdef.h"
#include "p_local.h"

#include "s_sound.h"

#include "g_game.h"

// State.
#include "doomstat.h"
#include "r_state.h"

// Data.
#include "sounds.h"

typedef enum
{
    DI_EAST,
    DI_NORTHEAST,
    DI_NORTH,
    DI_NORTHWEST,
    DI_WEST,
    DI_SOUTHWEST,
    DI_SOUTH,
    DI_SOUTHEAST,
    DI_NODIR,
    NUMDIRS
} dirtytype_t;

//
// P_NewChaseDir related LUT.
//
dirtytype_t opposite[] =
{
    DI_WEST, DI_SOUTHWEST, DI_SOUTH, DI_SOUTHEAST,
    DI_EAST, DI_NORTHEAST, DI_NORTH, DI_NORTHWEST, DI_NODIR
};

dirtytype_t diags[] =
{
    DI_NORTHWEST, DI_NORTHEAST, DI_SOUTHWEST, DI_SOUTHEAST
};

void A_Fall (mobj_t *actor);

//
// ENEMY THINKING
// Enemies are allways spawned
// with targetplayer = -1, threshold = 0
// Most monsters are spawned unaware of all players,
// but some can be made preaware
//

```

```

//
// Called by P_NoiseAlert.
// Recursively traverse adjacent sectors,
// sound blocking lines cut off traversal.
//

mobj_t*          soundtarget;

void
P_RecursiveSound
( sector_t*      sec,
  int            soundblocks )
{
    int          i;
    line_t*      check;
    sector_t*    other;

    // wake up all monsters in this sector
    if (sec->validcount == validcount
        && sec->soundtraversed <= soundblocks+1)
    {
        return;          // already flooded
    }

    sec->validcount = validcount;
    sec->soundtraversed = soundblocks+1;
    sec->soundtarget = soundtarget;

    for (i=0 ;i<sec->linecount ; i++)
    {
        check = sec->lines[i];
        if (!(check->flags & ML_TWOSIDED) )
            continue;

        P_LineOpening (check);

        if (openrange <= 0)
            continue;      // closed door

        if ( sides[ check->sidenum[0] ].sector == sec)
            other = sides[ check->sidenum[1] ] .sector;
        else
            other = sides[ check->sidenum[0] ] .sector;

        if (check->flags & ML_SOUNDBLOCK)
        {
            if (!soundblocks)
                P_RecursiveSound (other, 1);
        }
        else
            P_RecursiveSound (other, soundblocks);
    }
}

//
// P_NoiseAlert
// If a monster yells at a player,
// it will alert other monsters to the player.
//
void
P_NoiseAlert

```

```

( mobj_t*      target,
  mobj_t*      emitter )
{
    soundtarget = target;
    validcount++;
    P_RecursiveSound (emitter->subsector->sector, 0);
}

//
// P_CheckMeleeRange
//
boolean P_CheckMeleeRange (mobj_t*      actor)
{
    mobj_t*      pl;
    fixed_t      dist;

    if (!actor->target)
        return false;

    pl = actor->target;
    dist = P_AproxDistance (pl->x-actor->x, pl->y-actor->y);

    if (dist >= MELEERANGE-20*FRACUNIT+pl->info->radius)
        return false;

    if (! P_CheckSight (actor, actor->target) )
        return false;

    return true;
}

//
// P_CheckMissileRange
//
boolean P_CheckMissileRange (mobj_t* actor)
{
    fixed_t      dist;

    if (! P_CheckSight (actor, actor->target) )
        return false;

    if ( actor->flags & MF_JUSTHIT )
    {
        // the target just hit the enemy,
        // so fight back!
        actor->flags &= ~MF_JUSTHIT;
        return true;
    }

    if (actor->reactiontime)
        return false;      // do not attack yet

    // OPTIMIZE: get this from a global checksight
    dist = P_AproxDistance ( actor->x-actor->target->x,
                             actor->y-actor->target->y) - 64*FRACUNIT;

    if (!actor->info->meleestate)
        dist -= 128*FRACUNIT;      // no melee attack, so fire more

    dist >>= 16;

    if (actor->type == MT_VILE)

```

```

{
    if (dist > 14*64)
        return false;          // too far away
}

if (actor->type == MT_UNDEAD)
{
    if (dist < 196)
        return false;          // close for fist attack
    dist >>= 1;
}

if (actor->type == MT_CYBORG
    || actor->type == MT_SPIDER
    || actor->type == MT_SKULL)
{
    dist >>= 1;
}

if (dist > 200)
    dist = 200;

if (actor->type == MT_CYBORG && dist > 160)
    dist = 160;

if (P_Random () < dist)
    return false;

return true;
}

//
// P_Move
// Move in the current direction,
// returns false if the move is blocked.
//
fixed_t      xspeed[8] = {FRACUNIT,47000,0,-47000,-FRACUNIT,-47000,0,47000};
fixed_t      yspeed[8] = {0,47000,FRACUNIT,47000,0,-47000,-FRACUNIT,-47000};

#define MAXSPECIALCROSS      8

extern      line_t*      spechit[MAXSPECIALCROSS];
extern      int          numspechit;

boolean P_Move (mobj_t*      actor)
{
    fixed_t      tryx;
    fixed_t      tryy;

    line_t*      ld;

    // warning: 'catch', 'throw', and 'try'
    // are all C++ reserved words
    boolean      try_ok;
    boolean      good;

    if (actor->movedir == DI_NODIR)
        return false;

    if ((unsigned)actor->movedir >= 8)
        I_Error ("Weird actor->movedir!");
}

```



```

tryx = actor->x + actor->info->speed*xspeed[actor->movedir];
tryy = actor->y + actor->info->speed*yspeed[actor->movedir];

try_ok = P_TryMove (actor, tryx, tryy);

if (!try_ok)
{
    // open any specials
    if (actor->flags & MF_FLOAT && floatok)
    {
        // must adjust height
        if (actor->z < tmfloorz)
            actor->z += FLOATSPEED;
        else
            actor->z -= FLOATSPEED;

        actor->flags |= MF_INFLOAT;
        return true;
    }

    if (!numspechit)
        return false;

    actor->movedir = DI_NODIR;
    good = false;
    while (numspechit--)
    {
        ld = spechit[numspechit];
        // if the special is not a door
        // that can be opened,
        // return false
        if (P_UseSpecialLine (actor, ld,0))
            good = true;
    }
    return good;
}
else
{
    actor->flags &= ~MF_INFLOAT;
}

if (! (actor->flags & MF_FLOAT) )
    actor->z = actor->floorz;
return true;
}

//
// TryWalk
// Attempts to move actor on
// in its current (ob->moveangle) direction.
// If blocked by either a wall or an actor
// returns FALSE
// If move is either clear or blocked only by a door,
// returns TRUE and sets...
// If a door is in the way,
// an OpenDoor call is made to start it opening.
//
boolean P_TryWalk (mobj_t* actor)
{
    if (!P_Move (actor))
    {
        return false;
    }
}

```

```

    actor->movecount = P_Random()&15;
    return true;
}

```

```

void P_NewChaseDir (mobj_t*      actor)
{
    fixed_t      deltax;
    fixed_t      deltay;

    dirtype_t      d[3];

    int      tdir;
    dirtype_t      olddir;

    dirtype_t      turnaround;

    if (!actor->target)
        I_Error ("P_NewChaseDir: called with no target");

    olddir = actor->movedir;
    turnaround=opposite[olddir];

    deltax = actor->target->x - actor->x;
    deltay = actor->target->y - actor->y;

    if (deltax>10*FRACUNIT)
        d[1]= DI_EAST;
    else if (deltax<-10*FRACUNIT)
        d[1]= DI_WEST;
    else
        d[1]=DI_NODIR;

    if (deltay<-10*FRACUNIT)
        d[2]= DI_SOUTH;
    else if (deltay>10*FRACUNIT)
        d[2]= DI_NORTH;
    else
        d[2]=DI_NODIR;

    // try direct route
    if (d[1] != DI_NODIR
        && d[2] != DI_NODIR)
    {
        actor->movedir = diags[((deltay<0)<<1)+(deltax>0)];
        if (actor->movedir != turnaround && P_TryWalk(actor))
            return;
    }

    // try other directions
    if (P_Random() > 200
        || abs(deltay)>abs(deltax))
    {
        tdir=d[1];
        d[1]=d[2];
        d[2]=tdir;
    }

    if (d[1]==turnaround)
        d[1]=DI_NODIR;
    if (d[2]==turnaround)
        d[2]=DI_NODIR;
}

```

```

if (d[1]!=DI_NODIR)
{
    actor->movedir = d[1];
    if (P_TryWalk(actor))
    {
        // either moved forward or attacked
        return;
    }
}

if (d[2]!=DI_NODIR)
{
    actor->movedir =d[2];

    if (P_TryWalk(actor))
        return;
}

// there is no direct path to the player,
// so pick another direction.
if (olddir!=DI_NODIR)
{
    actor->movedir =olddir;

    if (P_TryWalk(actor))
        return;
}

// randomly determine direction of search
if (P_Random()&1)
{
    for ( tdir=DI_EAST;
          tdir<=DI_SOUTHEAST;
          tdir++ )
    {
        if (tdir!=turnaround)
        {
            actor->movedir =tdir;

            if ( P_TryWalk(actor) )
                return;
        }
    }
}
else
{
    for ( tdir=DI_SOUTHEAST;
          tdir != (DI_EAST-1);
          tdir-- )
    {
        if (tdir!=turnaround)
        {
            actor->movedir =tdir;

            if ( P_TryWalk(actor) )
                return;
        }
    }
}

if (turnaround != DI_NODIR)
{
    actor->movedir =turnaround;
    if ( P_TryWalk(actor) )

```

```

        return;
    }

    actor->movedir = DI_NODIR;        // can not move
}

//
// P_LookForPlayers
// If allaround is false, only look 180 degrees in front.
// Returns true if a player is targeted.
//
boolean
P_LookForPlayers
( mobj_t*      actor,
  boolean      allaround )
{
    int          c;
    int          stop;
    player_t*     player;
    sector_t*     sector;
    angle_t       an;
    fixed_t       dist;

    sector = actor->subsector->sector;

    c = 0;
    stop = (actor->lastlook-1)&3;

    for ( ; ; actor->lastlook = (actor->lastlook+1)&3 )
    {
        if (!playeringame[actor->lastlook])
            continue;

        if (c++ == 2
            || actor->lastlook == stop)
        {
            // done looking
            return false;
        }

        player = &players[actor->lastlook];

        if (player->health <= 0)
            continue;                // dead

        if (!P_CheckSight (actor, player->mo))
            continue;                // out of sight

        if (!allaround)
        {
            an = R_PointToAngle2 (actor->x,
                                   actor->y,
                                   player->mo->x,
                                   player->mo->y)
                - actor->angle;

            if (an > ANG90 && an < ANG270)
            {
                dist = P_AproxDistance (player->mo->x - actor->x,
                                           player->mo->y - actor->y);
                // if real close, react anyway
                if (dist > MELEERANGE)
                    continue;        // behind back
            }
        }
    }
}

```

```

    }
}

    actor->target = player->mo;
    return true;
}

return false;
}

//
// A_KeenDie
// DOOM II special, map 32.
// Uses special tag 666.
//
void A_KeenDie (mobj_t* mo)
{
    thinker_t*      th;
    mobj_t*         mo2;
    line_t          junk;

    A_Fall (mo);

    // scan the remaining thinkers
    // to see if all Keens are dead
    for (th = thinkercap.next ; th != &thinkercap ; th=th->next)
    {
        if (th->function.acp1 != (actionf_p1)P_MobjThinker)
            continue;

        mo2 = (mobj_t *)th;
        if (mo2 != mo
            && mo2->type == mo->type
            && mo2->health > 0)
        {
            // other Keen not dead
            return;
        }
    }

    junk.tag = 666;
    EV_DoDoor(&junk,open);
}

//
// ACTION ROUTINES
//

//
// A_Look
// Stay in state until a player is sighted.
//
void A_Look (mobj_t* actor)
{
    mobj_t*         targ;

    actor->threshold = 0;          // any shot will wake up
    targ = actor->subsector->sector->soundtarget;

    if (targ
        && (targ->flags & MF_SHOOTABLE) )
    {
        actor->target = targ;
    }
}

```

```

    if ( actor->flags & MF_AMBUSH )
    {
        if (P_CheckSight (actor, actor->target))
            goto seeyou;
    }
    else
        goto seeyou;
}

if (!P_LookForPlayers (actor, false) )
    return;

// go into chase state
seeyou:
if (actor->info->seesound)
{
    int                sound;

    switch (actor->info->seesound)
    {
        case sfx_posit1:
        case sfx_posit2:
        case sfx_posit3:
            sound = sfx_posit1+P_Random()%3;
            break;

        case sfx_bgsit1:
        case sfx_bgsit2:
            sound = sfx_bgsit1+P_Random()%2;
            break;

        default:
            sound = actor->info->seesound;
            break;
    }

    if (actor->type==MT_SPIDER
        || actor->type == MT_CYBORG)
    {
        // full volume
        S_StartSound (NULL, sound);
    }
    else
        S_StartSound (actor, sound);
}

P_SetMobjState (actor, actor->info->seestate);
}

```

```

//
// A_Chase
// Actor has a melee attack,
// so it tries to close as fast as possible
//
void A_Chase (mobj_t*      actor)
{
    int                delta;

    if (actor->reactiontime)
        actor->reactiontime--;
}

```

```

// modify target threshold
if (actor->threshold)
{
    if (!actor->target
        || actor->target->health <= 0)
    {
        actor->threshold = 0;
    }
    else
        actor->threshold--;
}

// turn towards movement direction if not there yet
if (actor->movedir < 8)
{
    actor->angle &= (7<<29);
    delta = actor->angle - (actor->movedir << 29);

    if (delta > 0)
        actor->angle -= ANG90/2;
    else if (delta < 0)
        actor->angle += ANG90/2;
}

if (!actor->target
    || !(actor->target->flags&MF_SHOOTABLE))
{
    // look for a new target
    if (P_LookForPlayers(actor,true))
        return;          // got a new target

    P_SetMobjState (actor, actor->info->spawnstate);
    return;
}

// do not attack twice in a row
if (actor->flags & MF_JUSTATTACKED)
{
    actor->flags &= ~MF_JUSTATTACKED;
    if (gameskill != sk_nightmare && !fastparm)
        P_NewChaseDir (actor);
    return;
}

// check for melee attack
if (actor->info->meleestate
    && P_CheckMeleeRange (actor))
{
    if (actor->info->attacksound)
        S_StartSound (actor, actor->info->attacksound);

    P_SetMobjState (actor, actor->info->meleestate);
    return;
}

// check for missile attack
if (actor->info->missilestate)
{
    if (gameskill < sk_nightmare
        && !fastparm && actor->movecount)
    {
        goto nomissile;
    }

    if (!P_CheckMissileRange (actor))

```

```

        goto nomissile;

        P_SetMobjState (actor, actor->info->missilestate);
        actor->flags |= MF_JUSTATTACKED;
        return;
    }

    // ?
nomissile:
    // possibly choose another target
    if (netgame
        && !actor->threshold
        && !P_CheckSight (actor, actor->target) )
    {
        if (P_LookForPlayers(actor,true))
            return;        // got a new target
    }

    // chase towards player
    if (--actor->movecount<0
        || !P_Move (actor))
    {
        P_NewChaseDir (actor);
    }

    // make active sound
    if (actor->info->activesound
        && P_Random () < 3)
    {
        S_StartSound (actor, actor->info->activesound);
    }
}

//
// A_FaceTarget
//
void A_FaceTarget (mobj_t* actor)
{
    if (!actor->target)
        return;

    actor->flags &= ~MF_AMBUSH;

    actor->angle = R_PointToAngle2 (actor->x,
                                   actor->y,
                                   actor->target->x,
                                   actor->target->y);

    if (actor->target->flags & MF_SHADOW)
        actor->angle += (P_Random()-P_Random())<<21;
}

//
// A_PosAttack
//
void A_PosAttack (mobj_t* actor)
{
    int             angle;
    int             damage;
    int             slope;

    if (!actor->target)
        return;

```



```

A_FaceTarget (actor);
angle = actor->angle;
slope = P_AimLineAttack (actor, angle, MISSILERANGE);

S_StartSound (actor, sfx_pistol);
angle += (P_Random()-P_Random())<<20;
damage = ((P_Random()%5)+1)*3;
P_LineAttack (actor, angle, MISSILERANGE, slope, damage);
}

void A_SPosAttack (mobj_t* actor)
{
    int            i;
    int            angle;
    int            bangle;
    int            damage;
    int            slope;

    if (!actor->target)
        return;

    S_StartSound (actor, sfx_shotgn);
    A_FaceTarget (actor);
    bangle = actor->angle;
    slope = P_AimLineAttack (actor, bangle, MISSILERANGE);

    for (i=0 ; i<3 ; i++)
    {
        angle = bangle + ((P_Random()-P_Random())<<20);
        damage = ((P_Random()%5)+1)*3;
        P_LineAttack (actor, angle, MISSILERANGE, slope, damage);
    }
}

void A_CPosAttack (mobj_t* actor)
{
    int            angle;
    int            bangle;
    int            damage;
    int            slope;

    if (!actor->target)
        return;

    S_StartSound (actor, sfx_shotgn);
    A_FaceTarget (actor);
    bangle = actor->angle;
    slope = P_AimLineAttack (actor, bangle, MISSILERANGE);

    angle = bangle + ((P_Random()-P_Random())<<20);
    damage = ((P_Random()%5)+1)*3;
    P_LineAttack (actor, angle, MISSILERANGE, slope, damage);
}

void A_CPosRefire (mobj_t* actor)
{
    // keep firing unless target got out of sight
    A_FaceTarget (actor);

    if (P_Random () < 40)
        return;

    if (!actor->target
        || actor->target->health <= 0

```

```

        || !P_CheckSight (actor, actor->target) )
    {
        P_SetMobjState (actor, actor->info->seestate);
    }
}

void A_SpidRefire (mobj_t* actor)
{
    // keep firing unless target got out of sight
    A_FaceTarget (actor);

    if (P_Random () < 10)
        return;

    if (!actor->target
        || actor->target->health <= 0
        || !P_CheckSight (actor, actor->target) )
    {
        P_SetMobjState (actor, actor->info->seestate);
    }
}

void A_BspiAttack (mobj_t *actor)
{
    if (!actor->target)
        return;

    A_FaceTarget (actor);

    // launch a missile
    P_SpawnMissile (actor, actor->target, MT_ARACHPLAZ);
}

//
// A_TroopAttack
//
void A_TroopAttack (mobj_t* actor)
{
    int                damage;

    if (!actor->target)
        return;

    A_FaceTarget (actor);
    if (P_CheckMeleeRange (actor))
    {
        S_StartSound (actor, sfx_claw);
        damage = (P_Random()%8+1)*3;
        P_DamageMobj (actor->target, actor, actor, damage);
        return;
    }

    // launch a missile
    P_SpawnMissile (actor, actor->target, MT_TROOPSHOT);
}

void A_SargAttack (mobj_t* actor)
{
    int                damage;

    if (!actor->target)

```

```

        return;

A_FaceTarget (actor);
if (P_CheckMeleeRange (actor))
{
    damage = ((P_Random()%10)+1)*4;
    P_DamageMobj (actor->target, actor, actor, damage);
}
}

void A_HeadAttack (mobj_t* actor)
{
    int                damage;

    if (!actor->target)
        return;

    A_FaceTarget (actor);
    if (P_CheckMeleeRange (actor))
    {
        damage = (P_Random()%6+1)*10;
        P_DamageMobj (actor->target, actor, actor, damage);
        return;
    }

    // launch a missile
    P_SpawnMissile (actor, actor->target, MT_HEADSHOT);
}

void A_CyberAttack (mobj_t* actor)
{
    if (!actor->target)
        return;

    A_FaceTarget (actor);
    P_SpawnMissile (actor, actor->target, MT_ROCKET);
}

void A_BruisAttack (mobj_t* actor)
{
    int                damage;

    if (!actor->target)
        return;

    if (P_CheckMeleeRange (actor))
    {
        S_StartSound (actor, sfx_claw);
        damage = (P_Random()%8+1)*10;
        P_DamageMobj (actor->target, actor, actor, damage);
        return;
    }

    // launch a missile
    P_SpawnMissile (actor, actor->target, MT_BRUISERSHOT);
}

//
// A_SkelMissile
//
void A_SkelMissile (mobj_t* actor)
{
    mobj_t*            mo;

```

```

    if (!actor->target)
        return;

    A_FaceTarget (actor);
    actor->z += 16*FRACUNIT;          // so missile spawns higher
    mo = P_SpawnMissile (actor, actor->target, MT_TRACER);
    actor->z -= 16*FRACUNIT;          // back to normal

    mo->x += mo->momx;
    mo->y += mo->momy;
    mo->tracer = actor->target;
}

int          TRACEANGLE = 0xc000000;

void A_Tracer (mobj_t* actor)
{
    angle_t    exact;
    fixed_t    dist;
    fixed_t    slope;
    mobj_t*    dest;
    mobj_t*    th;

    if (gametic & 3)
        return;

    // spawn a puff of smoke behind the rocket
    P_SpawnPuff (actor->x, actor->y, actor->z);

    th = P_SpawnMobj (actor->x-actor->momx,
                      actor->y-actor->momy,
                      actor->z, MT_SMOKE);

    th->momz = FRACUNIT;
    th->tics -= P_Random()&3;
    if (th->tics < 1)
        th->tics = 1;

    // adjust direction
    dest = actor->tracer;

    if (!dest || dest->health <= 0)
        return;

    // change angle
    exact = R_PointToAngle2 (actor->x,
                             actor->y,
                             dest->x,
                             dest->y);

    if (exact != actor->angle)
    {
        if (exact - actor->angle > 0x80000000)
        {
            actor->angle -= TRACEANGLE;
            if (exact - actor->angle < 0x80000000)
                actor->angle = exact;
        }
        else
        {
            actor->angle += TRACEANGLE;
            if (exact - actor->angle > 0x80000000)
                actor->angle = exact;
        }
    }
}

```

```

}

exact = actor->angle>>ANGLETOFINESHIFT;
actor->momx = FixedMul (actor->info->speed, finecosine[exact]);
actor->momy = FixedMul (actor->info->speed, finesine[exact]);

// change slope
dist = P_AproxDistance (dest->x - actor->x,
                        dest->y - actor->y);

dist = dist / actor->info->speed;

if (dist < 1)
    dist = 1;
slope = (dest->z+40*FRACUNIT - actor->z) / dist;

if (slope < actor->momz)
    actor->momz -= FRACUNIT/8;
else
    actor->momz += FRACUNIT/8;
}

void A_SkelWhoosh (mobj_t*      actor)
{
    if (!actor->target)
        return;
    A_FaceTarget (actor);
    S_StartSound (actor,sfx_skeswg);
}

void A_SkelFist (mobj_t*      actor)
{
    int          damage;

    if (!actor->target)
        return;

    A_FaceTarget (actor);

    if (P_CheckMeleeRange (actor))
    {
        damage = ((P_Random()%10)+1)*6;
        S_StartSound (actor, sfx_skepch);
        P_DamageMobj (actor->target, actor, actor, damage);
    }
}

//
// PIT_VileCheck
// Detect a corpse that could be raised.
//
mobj_t*          corpsehit;
mobj_t*          vileobj;
fixed_t          viletryx;
fixed_t          viletryy;

boolean PIT_VileCheck (mobj_t*      thing)
{
    int          maxdist;
    boolean      check;

    if (!(thing->flags & MF_CORPSE) )

```

```

        return true;           // not a monster

    if (thing->tics != -1)
        return true;           // not lying still yet

    if (thing->info->raisestate == S_NULL)
        return true;           // monster doesn't have a raise state

    maxdist = thing->info->radius + mobjinfo[MT_VILE].radius;

    if ( abs(thing->x - viletryx) > maxdist
        || abs(thing->y - viletryy) > maxdist )
        return true;           // not actually touching

    corpsehit = thing;
    corpsehit->momx = corpsehit->momy = 0;
    corpsehit->height <= 2;
    check = P_CheckPosition (corpsehit, corpsehit->x, corpsehit->y);
    corpsehit->height >= 2;

    if (!check)
        return true;           // doesn't fit here

    return false;               // got one, so stop checking
}

```

```

//
// A_VileChase
// Check for ressurecting a body
//
void A_VileChase (mobj_t* actor)
{
    int                xl;
    int                xh;
    int                yl;
    int                yh;

    int                bx;
    int                by;

    mobjinfo_t*        info;
    mobj_t*            temp;

    if (actor->movedir != DI_NODIR)
    {
        // check for corpses to raise
        viletryx =
            actor->x + actor->info->speed*xspeed[actor->movedir];
        viletryy =
            actor->y + actor->info->speed*yspeed[actor->movedir];

        xl = (viletryx - bmaporgx - MAXRADIUS*2)>>MAPBLOCKSHIFT;
        xh = (viletryx - bmaporgx + MAXRADIUS*2)>>MAPBLOCKSHIFT;
        yl = (viletryy - bmaporgy - MAXRADIUS*2)>>MAPBLOCKSHIFT;
        yh = (viletryy - bmaporgy + MAXRADIUS*2)>>MAPBLOCKSHIFT;

        vileobj = actor;
        for (bx=xl ; bx<=xh ; bx++)
        {
            for (by=yl ; by<=yh ; by++)
            {
                // Call PIT_VileCheck to check
                // whether object is a corpse

```

```

        // that canbe raised.
        if (!P_BlockThingsIterator(bx,by,PIT_VileCheck))
        {
            // got one!
            temp = actor->target;
            actor->target = corpsehit;
            A_FaceTarget (actor);
            actor->target = temp;

            P_SetMobjState (actor, S_VILE_HEAL1);
            S_StartSound (corpsehit, sfx_slop);
            info = corpsehit->info;

            P_SetMobjState (corpsehit,info->raisestate);
            corpsehit->height <= 2;
            corpsehit->flags = info->flags;
            corpsehit->health = info->spawnhealth;
            corpsehit->target = NULL;

            return;
        }
    }
}

// Return to normal attack.
A_Chase (actor);
}

//
// A_VileStart
//
void A_VileStart (mobj_t* actor)
{
    S_StartSound (actor, sfx_vilatk);
}

//
// A_Fire
// Keep fire in front of player unless out of sight
//
void A_Fire (mobj_t* actor);

void A_StartFire (mobj_t* actor)
{
    S_StartSound(actor,sfx_flamst);
    A_Fire(actor);
}

void A_FireCrackle (mobj_t* actor)
{
    S_StartSound(actor,sfx_flame);
    A_Fire(actor);
}

void A_Fire (mobj_t* actor)
{
    mobj_t*      dest;
    unsigned     an;

    dest = actor->tracer;
    if (!dest)
        return;

```

```

// don't move it if the vile lost sight
if (!P_CheckSight (actor->target, dest) )
    return;

an = dest->angle >> ANGLETOFINESHIFT;

P_UnsetThingPosition (actor);
actor->x = dest->x + FixedMul (24*FRACUNIT, finecosine[an]);
actor->y = dest->y + FixedMul (24*FRACUNIT, finesine[an]);
actor->z = dest->z;
P_SetThingPosition (actor);
}

//
// A_VileTarget
// Spawn the hellfire
//
void A_VileTarget (mobj_t* actor)
{
    mobj_t* fog;

    if (!actor->target)
        return;

    A_FaceTarget (actor);

    fog = P_SpawnMobj (actor->target->x,
                      actor->target->y,
                      actor->target->z, MT_FIRE);

    actor->tracer = fog;
    fog->target = actor;
    fog->tracer = actor->target;
    A_Fire (fog);
}

//
// A_VileAttack
//
void A_VileAttack (mobj_t* actor)
{
    mobj_t* fire;
    int an;

    if (!actor->target)
        return;

    A_FaceTarget (actor);

    if (!P_CheckSight (actor, actor->target) )
        return;

    S_StartSound (actor, sfx_barexp);
    P_DamageMobj (actor->target, actor, actor, 20);
    actor->target->momz = 1000*FRACUNIT/actor->target->info->mass;

    an = actor->angle >> ANGLETOFINESHIFT;

    fire = actor->tracer;

```



```

    if (!fire)
        return;

    // move the fire between the vile and the player
    fire->x = actor->target->x - FixedMul (24*FRACUNIT, finecosine[an]);
    fire->y = actor->target->y - FixedMul (24*FRACUNIT, finesine[an]);
    P_RadiusAttack (fire, actor, 70 );
}

//
// Mancubus attack,
// firing three missiles (bruisers)
// in three different directions?
// Doesn't look like it.
//
#define FATSREAD (ANG90/8)

void A_FatRaise (mobj_t *actor)
{
    A_FaceTarget (actor);
    S_StartSound (actor, sfx_manatk);
}

void A_FatAttack1 (mobj_t* actor)
{
    mobj_t* mo;
    int an;

    A_FaceTarget (actor);
    // Change direction to ...
    actor->angle += FATSREAD;
    P_SpawnMissile (actor, actor->target, MT_FATSHOT);

    mo = P_SpawnMissile (actor, actor->target, MT_FATSHOT);
    mo->angle += FATSREAD;
    an = mo->angle >> ANGLETOFINESHIFT;
    mo->momx = FixedMul (mo->info->speed, finecosine[an]);
    mo->momy = FixedMul (mo->info->speed, finesine[an]);
}

void A_FatAttack2 (mobj_t* actor)
{
    mobj_t* mo;
    int an;

    A_FaceTarget (actor);
    // Now here choose opposite deviation.
    actor->angle -= FATSREAD;
    P_SpawnMissile (actor, actor->target, MT_FATSHOT);

    mo = P_SpawnMissile (actor, actor->target, MT_FATSHOT);
    mo->angle -= FATSREAD*2;
    an = mo->angle >> ANGLETOFINESHIFT;
    mo->momx = FixedMul (mo->info->speed, finecosine[an]);
    mo->momy = FixedMul (mo->info->speed, finesine[an]);
}

void A_FatAttack3 (mobj_t* actor)
{
    mobj_t* mo;

```

```

int                an;

A_FaceTarget (actor);

mo = P_SpawnMissile (actor, actor->target, MT_FATSHOT);
mo->angle -= FATSPREAD/2;
an = mo->angle >> ANGLETOFINESHIFT;
mo->momx = FixedMul (mo->info->speed, finecosine[an]);
mo->momy = FixedMul (mo->info->speed, finesine[an]);

mo = P_SpawnMissile (actor, actor->target, MT_FATSHOT);
mo->angle += FATSPREAD/2;
an = mo->angle >> ANGLETOFINESHIFT;
mo->momx = FixedMul (mo->info->speed, finecosine[an]);
mo->momy = FixedMul (mo->info->speed, finesine[an]);
}

//
// SkullAttack
// Fly at the player like a missile.
//
#define            SKULLSPEED            (20*FRACUNIT)

void A_SkullAttack (mobj_t* actor)
{
    mobj_t*        dest;
    angle_t        an;
    int            dist;

    if (!actor->target)
        return;

    dest = actor->target;
    actor->flags |= MF_SKULLFLY;

    S_StartSound (actor, actor->info->attacksound);
    A_FaceTarget (actor);
    an = actor->angle >> ANGLETOFINESHIFT;
    actor->momx = FixedMul (SKULLSPEED, finecosine[an]);
    actor->momy = FixedMul (SKULLSPEED, finesine[an]);
    dist = P_AproxDistance (dest->x - actor->x, dest->y - actor->y);
    dist = dist / SKULLSPEED;

    if (dist < 1)
        dist = 1;
    actor->momz = (dest->z+(dest->height>>1) - actor->z) / dist;
}

//
// A_PainShootSkull
// Spawn a lost soul and launch it at the target
//
void
A_PainShootSkull
( mobj_t*        actor,
  angle_t        angle )
{
    fixed_t        x;
    fixed_t        y;
    fixed_t        z;

    mobj_t*        newmobj;
    angle_t        an;

```

```

int                prestep;
int                count;
thinker_t*        currentthinker;

// count total number of skull currently on the level
count = 0;

currentthinker = thinkercap.next;
while (currentthinker != &thinkercap)
{
    if ( (currentthinker->function.acp1 == (actionf_p1)P_MobjThinker)
        && ((mobj_t *)currentthinker->type == MT_SKULL)
        count++;
    currentthinker = currentthinker->next;
}

// if there are allready 20 skulls on the level,
// don't spit another one
if (count > 20)
    return;

// okay, there's playe for another one
an = angle >> ANGLETOFINESHIFT;

prestep =
    4*FRACUNIT
    + 3*(actor->info->radius + mobjinfo[MT_SKULL].radius)/2;

x = actor->x + FixedMul (prestep, finecosine[an]);
y = actor->y + FixedMul (prestep, finesine[an]);
z = actor->z + 8*FRACUNIT;

newmobj = P_SpawnMobj (x , y, z, MT_SKULL);

// Check for movements.
if (!P_TryMove (newmobj, newmobj->x, newmobj->y))
{
    // kill it immediately
    P_DamageMobj (newmobj,actor,actor,10000);
    return;
}

newmobj->target = actor->target;
A_SkullAttack (newmobj);
}

//
// A_PainAttack
// Spawn a lost soul and launch it at the target
//
void A_PainAttack (mobj_t* actor)
{
    if (!actor->target)
        return;

    A_FaceTarget (actor);
    A_PainShootSkull (actor, actor->angle);
}

void A_PainDie (mobj_t* actor)
{
    A_Fall (actor);
}

```

```

    A_PainShootSkull (actor, actor->angle+ANG90);
    A_PainShootSkull (actor, actor->angle+ANG180);
    A_PainShootSkull (actor, actor->angle+ANG270);
}

```

```

void A_Scream (mobj_t* actor)
{
    int                sound;

    switch (actor->info->deathsound)
    {
        case 0:
            return;

        case sfx_podth1:
        case sfx_podth2:
        case sfx_podth3:
            sound = sfx_podth1 + P_Random ()%3;
            break;

        case sfx_bgdth1:
        case sfx_bgdth2:
            sound = sfx_bgdth1 + P_Random ()%2;
            break;

        default:
            sound = actor->info->deathsound;
            break;
    }

    // Check for bosses.
    if (actor->type==MT_SPIDER
        || actor->type == MT_CYBORG)
    {
        // full volume
        S_StartSound (NULL, sound);
    }
    else
        S_StartSound (actor, sound);
}

```

```

void A_XScream (mobj_t* actor)
{
    S_StartSound (actor, sfx_slop);
}

```

```

void A_Pain (mobj_t* actor)
{
    if (actor->info->painsound)
        S_StartSound (actor, actor->info->painsound);
}

```

```

void A_Fall (mobj_t *actor)
{
    // actor is on ground, it can be walked over
    actor->flags &= ~MF_SOLID;
}

```

```

    // So change this if corpse objects
    // are meant to be obstacles.
}

//
// A_Explode
//
void A_Explode (mobj_t* thingy)
{
    P_RadiusAttack ( thingy, thingy->target, 128 );
}

//
// A_BossDeath
// Possibly trigger special effects
// if on first boss level
//
void A_BossDeath (mobj_t* mo)
{
    thinker_t*      th;
    mobj_t*         mo2;
    line_t          junk;
    int             i;

    if ( gamemode == commercial)
    {
        if (gamemap != 7)
            return;

        if ((mo->type != MT_FATSO)
            && (mo->type != MT_BABY))
            return;
    }
    else
    {
        switch(gameepisode)
        {
            case 1:
                if (gamemap != 8)
                    return;

                if (mo->type != MT_BRUISER)
                    return;
                break;

            case 2:
                if (gamemap != 8)
                    return;

                if (mo->type != MT_CYBORG)
                    return;
                break;

            case 3:
                if (gamemap != 8)
                    return;

                if (mo->type != MT_SPIDER)
                    return;

                break;

            case 4:

```

```

        switch(gamemap)
        {
            case 6:
                if (mo->type != MT_CYBORG)
                    return;
                break;

            case 8:
                if (mo->type != MT_SPIDER)
                    return;
                break;

            default:
                return;
                break;
        }
        break;

    default:
        if (gamemap != 8)
            return;
        break;
}

}

// make sure there is a player alive for victory
for (i=0 ; i<MAXPLAYERS ; i++)
    if (playeringame[i] && players[i].health > 0)
        break;

if (i==MAXPLAYERS)
    return;          // no one left alive, so do not end game

// scan the remaining thinkers to see
// if all bosses are dead
for (th = thinkercap.next ; th != &thinkercap ; th=th->next)
{
    if (th->function.acp1 != (actionf_p1)P_MobjThinker)
        continue;

    mo2 = (mobj_t *)th;
    if (mo2 != mo
        && mo2->type == mo->type
        && mo2->health > 0)
    {
        // other boss not dead
        return;
    }
}

// victory!
if ( gamemode == commercial)
{
    if (gamemap == 7)
    {
        if (mo->type == MT_FATSO)
        {
            junk.tag = 666;
            EV_DoFloor(&junk,lowerFloorToLowest);
            return;
        }

        if (mo->type == MT_BABY)

```

```

        {
            junk.tag = 667;
            EV_DoFloor(&junk,raiseToTexture);
            return;
        }
    }
}
else
{
    switch(gameepisode)
    {
        case 1:
            junk.tag = 666;
            EV_DoFloor (&junk, lowerFloorToLowest);
            return;
            break;

        case 4:
            switch(gamemap)
            {
                case 6:
                    junk.tag = 666;
                    EV_DoDoor (&junk, blazeOpen);
                    return;
                    break;

                case 8:
                    junk.tag = 666;
                    EV_DoFloor (&junk, lowerFloorToLowest);
                    return;
                    break;
            }
        }
    }

    G_ExitLevel ();
}

void A_Hoof (mobj_t* mo)
{
    S_StartSound (mo, sfx_hoof);
    A_Chase (mo);
}

void A_Metal (mobj_t* mo)
{
    S_StartSound (mo, sfx_metal);
    A_Chase (mo);
}

void A_BabyMetal (mobj_t* mo)
{
    S_StartSound (mo, sfx_bspwlk);
    A_Chase (mo);
}

void
A_OpenShotgun2
( player_t*      player,
  pspdef_t*      psp )
{
    S_StartSound (player->mo, sfx_dbopn);
}

```

```

void
A_LoadShotgun2
( player_t*      player,
  pspdef_t*      psp )
{
    S_StartSound (player->mo, sfx_dbload);
}

void
A_ReFire
( player_t*      player,
  pspdef_t*      psp );

void
A_CloseShotgun2
( player_t*      player,
  pspdef_t*      psp )
{
    S_StartSound (player->mo, sfx_dbcls);
    A_ReFire(player,psp);
}


mobj_t*          braintargets[32];
int              numbraintargets;
int              braintargeton;

void A_BrainAwake (mobj_t* mo)
{
    thinker_t*    thinker;
    mobj_t*       m;

    // find all the target spots
    numbraintargets = 0;
    braintargeton = 0;

    thinker = thinkercap.next;
    for (thinker = thinkercap.next ;
        thinker != &thinkercap ;
        thinker = thinker->next)
    {
        if (thinker->function.acp1 != (actionf_p1)P_MobjThinker)
            continue;          // not a mobj

        m = (mobj_t *)thinker;

        if (m->type == MT_BOSSTARGET )
        {
            braintargets[numbraintargets] = m;
            numbraintargets++;
        }
    }

    S_StartSound (NULL,sfx_bossit);
}

void A_BrainPain (mobj_t*      mo)
{
    S_StartSound (NULL,sfx_bospn);
}

void A_BrainScream (mobj_t*      mo)

```



```

{
    int            x;
    int            y;
    int            z;
    mobj_t*        th;

    for (x=mo->x - 196*FRACUNIT ; x< mo->x + 320*FRACUNIT ; x+= FRACUNIT*8)
    {
        y = mo->y - 320*FRACUNIT;
        z = 128 + P_Random()*2*FRACUNIT;
        th = P_SpawnMobj (x,y,z, MT_ROCKET);
        th->momz = P_Random()*512;

        P_SetMobjState (th, S_BRAINEXPLODE1);

        th->tics -= P_Random()&7;
        if (th->tics < 1)
            th->tics = 1;
    }

    S_StartSound (NULL,sfx_bosdth);
}

```

```

void A_BrainExplode (mobj_t* mo)
{
    int            x;
    int            y;
    int            z;
    mobj_t*        th;

    x = mo->x + (P_Random () - P_Random ())*2048;
    y = mo->y;
    z = 128 + P_Random()*2*FRACUNIT;
    th = P_SpawnMobj (x,y,z, MT_ROCKET);
    th->momz = P_Random()*512;

    P_SetMobjState (th, S_BRAINEXPLODE1);

    th->tics -= P_Random()&7;
    if (th->tics < 1)
        th->tics = 1;
}

```

```

void A_BrainDie (mobj_t*        mo)
{
    G_ExitLevel ();
}

```

```

void A_BrainSpit (mobj_t*        mo)
{
    mobj_t*        targ;
    mobj_t*        newmobj;

    static int      easy = 0;

    easy ^= 1;
    if (gameskill <= sk_easy && (!easy))
        return;

    // shoot a cube at current target
    targ = braintargets[braintargeton];
    braintargeton = (braintargeton+1)%numbraintargets;
}

```

```

// spawn brain missile
newmobj = P_SpawnMissile (mo, targ, MT_SPAWNSHOT);
newmobj->target = targ;
newmobj->reactiontime =
    ((targ->y - mo->y)/newmobj->momy) / newmobj->state->tics;

S_StartSound(NULL, sfx_bospit);
}

```

```

void A_SpawnFly (mobj_t* mo);

```

```

// travelling cube sound
void A_SpawnSound (mobj_t* mo)
{
    S_StartSound (mo,sfx_boscub);
    A_SpawnFly(mo);
}

```

```

void A_SpawnFly (mobj_t* mo)
{
    mobj_t*      newmobj;
    mobj_t*      fog;
    mobj_t*      targ;
    int           r;
    mobjtype_t    type;

    if (--mo->reactiontime)
        return;          // still flying

    targ = mo->target;

    // First spawn teleport fog.
    fog = P_SpawnMobj (targ->x, targ->y, targ->z, MT_SPAWNFIRE);
    S_StartSound (fog, sfx_telept);

    // Randomly select monster to spawn.
    r = P_Random ();

    // Probability distribution (kind of :),
    // decreasing likelihood.
    if ( r<50 )
        type = MT_TROOP;
    else if (r<90)
        type = MT_SERGEANT;
    else if (r<120)
        type = MT_SHADOWS;
    else if (r<130)
        type = MT_PAIN;
    else if (r<160)
        type = MT_HEAD;
    else if (r<162)
        type = MT_VILE;
    else if (r<172)
        type = MT_UNDEAD;
    else if (r<192)
        type = MT_BABY;
    else if (r<222)
        type = MT_FATSO;
    else if (r<246)
        type = MT_KNIGHT;
    else
        type = MT_BRUISER;
}

```

```

newmobj      = P_SpawnMobj (targ->x, targ->y, targ->z, type);
if (P_LookForPlayers (newmobj, true) )
    P_SetMobjState (newmobj, newmobj->info->seestate);

// telefrag anything in this spot
P_TeleportMove (newmobj, newmobj->x, newmobj->y);

// remove self (i.e., cube).
P_RemoveMobj (mo);
}

```

```

void A_PlayerScream (mobj_t* mo)
{
    // Default death sound.
    int          sound = sfx_pldeth;

    if ( (gamemode == commercial)
        &&      (mo->health < -50))
    {
        // IF THE PLAYER DIES
        // LESS THAN -50% WITHOUT GIBBING
        sound = sfx_pdiehi;
    }

    S_StartSound (mo, sound);
}

```

## 9.6 p\_floor.c

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Floor animation: raising stairs.
//
//-----

static const char
rcsid[] = "$Id: p_floor.c,v 1.4 1997/02/03 16:47:54 b1 Exp $";

#include "z_zone.h"
#include "doomdef.h"
#include "p_local.h"

#include "s_sound.h"

```

```

// State.
#include "doomstat.h"
#include "r_state.h"
// Data.
#include "sounds.h"

//
// FLOORS
//

//
// Move a plane (floor or ceiling) and check for crushing
//
result_e
T_MovePlane
( sector_t*      sector,
  fixed_t        speed,
  fixed_t        dest,
  boolean        crush,
  int            floorOrCeiling,
  int            direction )
{
    boolean        flag;
    fixed_t        lastpos;

    switch(floorOrCeiling)
    {
    case 0:
        // FLOOR
        switch(direction)
        {
        case -1:
            // DOWN
            if (sector->floorheight - speed < dest)
            {
                lastpos = sector->floorheight;
                sector->floorheight = dest;
                flag = P_ChangeSector(sector,crush);
                if (flag == true)
                {
                    sector->floorheight =lastpos;
                    P_ChangeSector(sector,crush);
                    //return crushed;
                }
                return pastdest;
            }
        else
        {
            lastpos = sector->floorheight;
            sector->floorheight -= speed;
            flag = P_ChangeSector(sector,crush);
            if (flag == true)
            {
                sector->floorheight = lastpos;
                P_ChangeSector(sector,crush);
                return crushed;
            }
        }
        break;

    case 1:
        // UP
        if (sector->floorheight + speed > dest)

```

```

{
    lastpos = sector->floorheight;
    sector->floorheight = dest;
    flag = P_ChangeSector(sector,crush);
    if (flag == true)
    {
        sector->floorheight = lastpos;
        P_ChangeSector(sector,crush);
        //return crushed;
    }
    return pastdest;
}
else
{
    // COULD GET CRUSHED
    lastpos = sector->floorheight;
    sector->floorheight += speed;
    flag = P_ChangeSector(sector,crush);
    if (flag == true)
    {
        if (crush == true)
            return crushed;
        sector->floorheight = lastpos;
        P_ChangeSector(sector,crush);
        return crushed;
    }
}
break;
}
break;

case 1:
// CEILING
switch(direction)
{
    case -1:
        // DOWN
        if (sector->ceilingheight - speed < dest)
        {
            lastpos = sector->ceilingheight;
            sector->ceilingheight = dest;
            flag = P_ChangeSector(sector,crush);

            if (flag == true)
            {
                sector->ceilingheight = lastpos;
                P_ChangeSector(sector,crush);
                //return crushed;
            }
            return pastdest;
        }
    else
    {
        // COULD GET CRUSHED
        lastpos = sector->ceilingheight;
        sector->ceilingheight -= speed;
        flag = P_ChangeSector(sector,crush);

        if (flag == true)
        {
            if (crush == true)
                return crushed;
            sector->ceilingheight = lastpos;
            P_ChangeSector(sector,crush);
            return crushed;
        }
    }
}

```

```

        }
    }
    break;

case 1:
    // UP
    if (sector->ceilingheight + speed > dest)
    {
        lastpos = sector->ceilingheight;
        sector->ceilingheight = dest;
        flag = P_ChangeSector(sector,crush);
        if (flag == true)
        {
            sector->ceilingheight = lastpos;
            P_ChangeSector(sector,crush);
            //return crushed;
        }
        return pastdest;
    }
    else
    {
        lastpos = sector->ceilingheight;
        sector->ceilingheight += speed;
        flag = P_ChangeSector(sector,crush);
// UNUSED
#ifdef 0
        if (flag == true)
        {
            sector->ceilingheight = lastpos;
            P_ChangeSector(sector,crush);
            return crushed;
        }
#endif
    }
    break;
}
break;

}
return ok;
}

//
// MOVE A FLOOR TO IT'S DESTINATION (UP OR DOWN)
//
void T_MoveFloor(floormove_t* floor)
{
    result_e      res;

    res = T_MovePlane(floor->sector,
                      floor->speed,
                      floor->floordestheight,
                      floor->crush,0,floor->direction);

    if (!(leveltime&7))
        S_StartSound((mobj_t *)&floor->sector->soundorg,
                     sfx_stnmov);

    if (res == pastdest)
    {
        floor->sector->specialdata = NULL;

        if (floor->direction == 1)
        {

```

```

        switch(floor->type)
        {
            case donutRaise:
                floor->sector->special = floor->newspecial;
                floor->sector->floorpics = floor->texture;
            default:
                break;
        }
    }
    else if (floor->direction == -1)
    {
        switch(floor->type)
        {
            case lowerAndChange:
                floor->sector->special = floor->newspecial;
                floor->sector->floorpics = floor->texture;
            default:
                break;
        }
    }
    P_RemoveThinker(&floor->thinker);

    S_StartSound((mobj_t *)&floor->sector->soundorg,
                sfx_pstop);
}

}

//
// HANDLE FLOOR TYPES
//
int
EV_DoFloor
( line_t*      line,
  floor_e      floortype )
{
    int          secnum;
    int          rtn;
    int          i;
    sector_t*    sec;
    floormove_t* floor;

    secnum = -1;
    rtn = 0;
    while ((secnum = P_FindSectorFromLineTag(line,secnum)) >= 0)
    {
        sec = &sectors[secnum];

        // ALREADY MOVING?  IF SO, KEEP GOING...
        if (sec->specialdata)
            continue;

        // new floor thinker
        rtn = 1;
        floor = Z_Malloc (sizeof(*floor), PU_LEVSPEC, 0);
        P_AddThinker (&floor->thinker);
        sec->specialdata = floor;
        floor->thinker.function.acp1 = (actionf_p1) T_MoveFloor;
        floor->type = floortype;
        floor->crush = false;

        switch(floortype)
        {
            case lowerFloor:
                floor->direction = -1;

```

```

    floor->sector = sec;
    floor->speed = FLOORSPEED;
    floor->floordestheight =
        P_FindHighestFloorSurrounding(sec);
    break;

case lowerFloorToLowest:
    floor->direction = -1;
    floor->sector = sec;
    floor->speed = FLOORSPEED;
    floor->floordestheight =
        P_FindLowestFloorSurrounding(sec);
    break;

case turboLower:
    floor->direction = -1;
    floor->sector = sec;
    floor->speed = FLOORSPEED * 4;
    floor->floordestheight =
        P_FindHighestFloorSurrounding(sec);
    if (floor->floordestheight != sec->floorheight)
        floor->floordestheight += 8*FRACUNIT;
    break;

case raiseFloorCrush:
    floor->crush = true;
case raiseFloor:
    floor->direction = 1;
    floor->sector = sec;
    floor->speed = FLOORSPEED;
    floor->floordestheight =
        P_FindLowestCeilingSurrounding(sec);
    if (floor->floordestheight > sec->ceilingheight)
        floor->floordestheight = sec->ceilingheight;
    floor->floordestheight -= (8*FRACUNIT)*
        (floortype == raiseFloorCrush);
    break;

case raiseFloorTurbo:
    floor->direction = 1;
    floor->sector = sec;
    floor->speed = FLOORSPEED*4;
    floor->floordestheight =
        P_FindNextHighestFloor(sec, sec->floorheight);
    break;

case raiseFloorToNearest:
    floor->direction = 1;
    floor->sector = sec;
    floor->speed = FLOORSPEED;
    floor->floordestheight =
        P_FindNextHighestFloor(sec, sec->floorheight);
    break;

case raiseFloor24:
    floor->direction = 1;
    floor->sector = sec;
    floor->speed = FLOORSPEED;
    floor->floordestheight = floor->sector->floorheight +
        24 * FRACUNIT;
    break;

case raiseFloor512:
    floor->direction = 1;
    floor->sector = sec;
    floor->speed = FLOORSPEED;

```



```

    floor->floordestheight = floor->sector->floorheight +
        512 * FRACUNIT;
    break;

case raiseFloor24AndChange:
    floor->direction = 1;
    floor->sector = sec;
    floor->speed = FLOORSPEED;
    floor->floordestheight = floor->sector->floorheight +
        24 * FRACUNIT;
    sec->floorpic = line->frontsector->floorpic;
    sec->special = line->frontsector->special;
    break;

case raiseToTexture:
{
    int        minsize = MAXINT;
    side_t*     side;

    floor->direction = 1;
    floor->sector = sec;
    floor->speed = FLOORSPEED;
    for (i = 0; i < sec->linecount; i++)
    {
        if (twoSided (secnum, i) )
        {
            side = getSide(secnum,i,0);
            if (side->bottomtexture >= 0)
                if (textureheight[side->bottomtexture] <
                    minsize)
                    minsize =
                        textureheight[side->bottomtexture];
            side = getSide(secnum,i,1);
            if (side->bottomtexture >= 0)
                if (textureheight[side->bottomtexture] <
                    minsize)
                    minsize =
                        textureheight[side->bottomtexture];
        }
    }
    floor->floordestheight =
        floor->sector->floorheight + minsize;
}
break;

case lowerAndChange:
    floor->direction = -1;
    floor->sector = sec;
    floor->speed = FLOORSPEED;
    floor->floordestheight =
        P_FindLowestFloorSurrounding(sec);
    floor->texture = sec->floorpic;

    for (i = 0; i < sec->linecount; i++)
    {
        if ( twoSided(secnum, i) )
        {
            if (getSide(secnum,i,0)->sector-sectors == secnum)
            {
                sec = getSector(secnum,i,1);

                if (sec->floorheight == floor->floordestheight)
                {
                    floor->texture = sec->floorpic;
                    floor->newspecial = sec->special;
                }
            }
        }
    }
}

```

```

        break;
    }
}
else
{
    sec = getSector(secnum,i,0);

    if (sec->floorheight == floor->floordestheight)
    {
        floor->texture = sec->floorpic;
        floor->newspecial = sec->special;
        break;
    }
}
}
}
default:
    break;
}
}
return rtn;
}

```

```

//
// BUILD A STAIRCASE!
//
int
EV_BuildStairs
( line_t*      line,
  stair_e      type )
{
    int          secnum;
    int          height;
    int          i;
    int          newsecnum;
    int          texture;
    int          ok;
    int          rtn;

    sector_t*    sec;
    sector_t*    tsec;

    floormove_t* floor;

    fixed_t      stairsize;
    fixed_t      speed;

    secnum = -1;
    rtn = 0;
    while ((secnum = P_FindSectorFromLineTag(line,secnum)) >= 0)
    {
        sec = &sectors[secnum];

        // ALREADY MOVING? IF SO, KEEP GOING...
        if (sec->specialdata)
            continue;

        // new floor thinker
        rtn = 1;
        floor = Z_Malloc (sizeof(*floor), PU_LEVSPEC, 0);
        P_AddThinker (&floor->thinker);
        sec->specialdata = floor;
    }
}

```

```

floor->thinker.function.acp1 = (actionf_p1) T_MoveFloor;
floor->direction = 1;
floor->sector = sec;
switch(type)
{
    case build8:
        speed = FLOORSPEED/4;
        stairsize = 8*FRACUNIT;
        break;
    case turbo16:
        speed = FLOORSPEED*4;
        stairsize = 16*FRACUNIT;
        break;
}
floor->speed = speed;
height = sec->floorheight + stairsize;
floor->floordestheight = height;

texture = sec->floorpic;

// Find next sector to raise
// 1.      Find 2-sided line with same sector side[0]
// 2.      Other side is the next sector to raise
do
{
    ok = 0;
    for (i = 0; i < sec->linecount; i++)
    {
        if ( !((sec->lines[i])>->flags & ML_TWOSIDED) )
            continue;

        tsec = (sec->lines[i])>->frontsector;
        newsecnum = tsec->sectors;

        if (secnum != newsecnum)
            continue;

        tsec = (sec->lines[i])>->backsector;
        newsecnum = tsec - sectors;

        if (tsec->floorpic != texture)
            continue;

        height += stairsize;

        if (tsec->specialdata)
            continue;

        sec = tsec;
        secnum = newsecnum;
        floor = Z_Malloc (sizeof(*floor), PU_LEVSPEC, 0);

        P_AddThinker (&floor->thinker);

        sec->specialdata = floor;
        floor->thinker.function.acp1 = (actionf_p1) T_MoveFloor;
        floor->direction = 1;
        floor->sector = sec;
        floor->speed = speed;
        floor->floordestheight = height;
        ok = 1;
        break;
    }
} while(ok);
}

```

```

    return rtn;
}

```

## 9.7 p\_inter.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Handling interactions (i.e., collisions).
//
//-----

static const char
rcsid[] = "$Id: p_inter.c,v 1.4 1997/02/03 22:45:11 b1 Exp $";

// Data.
#include "doomdef.h"
#include "dstrings.h"
#include "sounds.h"

#include "doomstat.h"

#include "m_random.h"
#include "i_system.h"

#include "am_map.h"

#include "p_local.h"

#include "s_sound.h"

#ifdef __GNUG__
#pragma implementation "p_inter.h"
#endif
#include "p_inter.h"

#define BONUSADD      6

// a weapon is found with two clip loads,
// a big item has five clip loads
int      maxammo[NUMAMMO] = {200, 50, 300, 50};

```

```

int          clipammo[NUMAMMO] = {10, 4, 20, 1};

//
// GET STUFF
//

//
// P_GiveAmmo
// Num is the number of clip loads,
// not the individual count (0= 1/2 clip).
// Returns false if the ammo can't be picked up at all
//

boolean
P_GiveAmmo
( player_t*      player,
  ammotype_t     ammo,
  int            num )
{
    int          oldammo;

    if (ammo == am_noammo)
        return false;

    if (ammo < 0 || ammo > NUMAMMO)
        I_Error ("P_GiveAmmo: bad type %i", ammo);

    if ( player->ammo[ammo] == player->maxammo[ammo] )
        return false;

    if (num)
        num *= clipammo[ammo];
    else
        num = clipammo[ammo]/2;

    if (gameskill == sk_baby
        || gameskill == sk_nightmare)
    {
        // give double ammo in trainer mode,
        // you'll need in nightmare
        num <= 1;
    }

    oldammo = player->ammo[ammo];
    player->ammo[ammo] += num;

    if (player->ammo[ammo] > player->maxammo[ammo])
        player->ammo[ammo] = player->maxammo[ammo];

    // If non zero ammo,
    // don't change up weapons,
    // player was lower on purpose.
    if (oldammo)
        return true;

    // We were down to zero,
    // so select a new weapon.
    // Preferences are not user selectable.
    switch (ammo)
    {
    case am_clip:
        if (player->readyweapon == wp_fist)
        {

```

```

        if (player->weaponowned[wp_chaingun])
            player->pendingweapon = wp_chaingun;
        else
            player->pendingweapon = wp_pistol;
    }
    break;

case am_shell:
    if (player->readyweapon == wp_fist
        || player->readyweapon == wp_pistol)
    {
        if (player->weaponowned[wp_shotgun])
            player->pendingweapon = wp_shotgun;
    }
    break;

case am_cell:
    if (player->readyweapon == wp_fist
        || player->readyweapon == wp_pistol)
    {
        if (player->weaponowned[wp_plasma])
            player->pendingweapon = wp_plasma;
    }
    break;

case am_misl:
    if (player->readyweapon == wp_fist)
    {
        if (player->weaponowned[wp_missile])
            player->pendingweapon = wp_missile;
    }
default:
    break;
}

return true;
}

//
// P_GiveWeapon
// The weapon name may have a MF_DROPPED flag ored in.
//
boolean
P_GiveWeapon
( player_t*      player,
  weapontype_t   weapon,
  boolean        dropped )
{
    boolean       gaveammo;
    boolean       gaveweapon;

    if (netgame
        && (deathmatch!=2)
        && !dropped )
    {
        // leave placed weapons forever on net games
        if (player->weaponowned[weapon])
            return false;

        player->bonuscount += BONUSADD;
        player->weaponowned[weapon] = true;

        if (deathmatch)
            P_GiveAmmo (player, weaponinfo[weapon].ammo, 5);
    }
}

```

```

    else
        P_GiveAmmo (player, weaponinfo[weapon].ammo, 2);
    player->pendingweapon = weapon;

    if (player == &players[consoleplayer])
        S_StartSound (NULL, sfx_wpnup);
    return false;
}

if (weaponinfo[weapon].ammo != am_noammo)
{
    // give one clip with a dropped weapon,
    // two clips with a found weapon
    if (dropped)
        gaveammo = P_GiveAmmo (player, weaponinfo[weapon].ammo, 1);
    else
        gaveammo = P_GiveAmmo (player, weaponinfo[weapon].ammo, 2);
}
else
    gaveammo = false;

if (player->weaponowned[weapon])
    gaveweapon = false;
else
{
    gaveweapon = true;
    player->weaponowned[weapon] = true;
    player->pendingweapon = weapon;
}

return (gaveweapon || gaveammo);
}

```

```

//
// P_GiveBody
// Returns false if the body isn't needed at all
//
boolean
P_GiveBody
( player_t*      player,
  int            num )
{
    if (player->health >= MAXHEALTH)
        return false;

    player->health += num;
    if (player->health > MAXHEALTH)
        player->health = MAXHEALTH;
    player->mo->health = player->health;

    return true;
}

```

```

//
// P_GiveArmor
// Returns false if the armor is worse
// than the current armor.
//
boolean
P_GiveArmor
( player_t*      player,

```

```

int          armortype )
{
    int          hits;

    hits = armortype*100;
    if (player->armorpoints >= hits)
        return false;          // don't pick up

    player->armortype = armortype;
    player->armorpoints = hits;

    return true;
}

```

```

//
// P_GiveCard
//
void
P_GiveCard
( player_t*      player,
  card_t        card )
{
    if (player->cards[card])
        return;

    player->bonuscount = BONUSADD;
    player->cards[card] = 1;
}

```

```

//
// P_GivePower
//
boolean
P_GivePower
( player_t*      player,
  int /*powertype_t*/ power )
{
    if (power == pw_invulnerability)
    {
        player->powers[power] = INVULNTICS;
        return true;
    }

    if (power == pw_invisibility)
    {
        player->powers[power] = INVISTICS;
        player->mo->flags |= MF_SHADOW;
        return true;
    }

    if (power == pw_infrared)
    {
        player->powers[power] = INFRATICS;
        return true;
    }

    if (power == pw_ironfeet)
    {
        player->powers[power] = IRONTICS;
        return true;
    }
}

```



```

    if (power == pw_strength)
    {
        P_GiveBody (player, 100);
        player->powers[power] = 1;
        return true;
    }

    if (player->powers[power])
        return false;          // already got it

    player->powers[power] = 1;
    return true;
}

```

```

//
// P_TouchSpecialThing
//
void
P_TouchSpecialThing
( mobj_t*      special,
  mobj_t*      toucher )
{
    player_t*    player;
    int          i;
    fixed_t      delta;
    int          sound;

    delta = special->z - toucher->z;

    if (delta > toucher->height
        || delta < -8*FRACUNIT)
    {
        // out of reach
        return;
    }

    sound = sfx_itemup;
    player = toucher->player;

    // Dead thing touching.
    // Can happen with a sliding player corpse.
    if (toucher->health <= 0)
        return;

    // Identify by sprite.
    switch (special->sprite)
    {
        // armor
        case SPR_ARM1:
            if (!P_GiveArmor (player, 1))
                return;
            player->message = GOTARMOR;
            break;

        case SPR_ARM2:
            if (!P_GiveArmor (player, 2))
                return;
            player->message = GOTMEGA;
            break;

        // bonus items
        case SPR_BON1:

```

```

player->health++; // can go over 100%
if (player->health > 200)
    player->health = 200;
player->mo->health = player->health;
player->message = GOTTHBONUS;
break;

case SPR_BON2:
    player->armorpoints++; // can go over 100%
    if (player->armorpoints > 200)
        player->armorpoints = 200;
    if (!player->armortype)
        player->armortype = 1;
    player->message = GOTARMBONUS;
    break;

case SPR_SOUL:
    player->health += 100;
    if (player->health > 200)
        player->health = 200;
    player->mo->health = player->health;
    player->message = GOTSUPER;
    sound = sfx_getpow;
    break;

case SPR_MEGA:
    if (gamemode != commercial)
        return;
    player->health = 200;
    player->mo->health = player->health;
    P_GiveArmor (player, 2);
    player->message = GOTMSPHERE;
    sound = sfx_getpow;
    break;

    // cards
    // leave cards for everyone
case SPR_BKEY:
    if (!player->cards[it_bluecard])
        player->message = GOTBLUECARD;
    P_GiveCard (player, it_bluecard);
    if (!netgame)
        break;
    return;

case SPR_YKEY:
    if (!player->cards[it_yellowcard])
        player->message = GOTYELWCARD;
    P_GiveCard (player, it_yellowcard);
    if (!netgame)
        break;
    return;

case SPR_RKEY:
    if (!player->cards[it_redcard])
        player->message = GOTREDCARD;
    P_GiveCard (player, it_redcard);
    if (!netgame)
        break;
    return;

case SPR_BSKU:
    if (!player->cards[it_blueskull])
        player->message = GOTBLUESKUL;
    P_GiveCard (player, it_blueskull);

```

```

    if (!netgame)
        break;
    return;

case SPR_YSKU:
    if (!player->cards[it_yellowskull])
        player->message = GOTYELWSKUL;
    P_GiveCard (player, it_yellowskull);
    if (!netgame)
        break;
    return;

case SPR_RSKU:
    if (!player->cards[it_redskull])
        player->message = GOTREDSKULL;
    P_GiveCard (player, it_redskull);
    if (!netgame)
        break;
    return;

    // medikits, heals
case SPR_STIM:
    if (!P_GiveBody (player, 10))
        return;
    player->message = GOTSTIM;
    break;

case SPR_MEDI:
    if (!P_GiveBody (player, 25))
        return;

    if (player->health < 25)
        player->message = GOTMEDINEED;
    else
        player->message = GOTMEDIKIT;
    break;

    // power ups
case SPR_PINV:
    if (!P_GivePower (player, pw_invulnerability))
        return;
    player->message = GOTINVUL;
    sound = sfx_getpow;
    break;

case SPR_PSTR:
    if (!P_GivePower (player, pw_strength))
        return;
    player->message = GOTBERSERK;
    if (player->readyweapon != wp_fist)
        player->pendingweapon = wp_fist;
    sound = sfx_getpow;
    break;

case SPR_PINS:
    if (!P_GivePower (player, pw_invisibility))
        return;
    player->message = GOTINVIS;
    sound = sfx_getpow;
    break;

case SPR_SUIT:
    if (!P_GivePower (player, pw_ironfeet))
        return;

```

```

    player->message = GOTSUIT;
    sound = sfx_getpow;
    break;

case SPR_PMAP:
    if (!P_GivePower (player, pw_allmap))
        return;
    player->message = GOTMAP;
    sound = sfx_getpow;
    break;

case SPR_PVIS:
    if (!P_GivePower (player, pw_infrared))
        return;
    player->message = GOTVISOR;
    sound = sfx_getpow;
    break;

    // ammo
case SPR_CLIP:
    if (special->flags & MF_DROPPED)
    {
        if (!P_GiveAmmo (player, am_clip, 0))
            return;
    }
    else
    {
        if (!P_GiveAmmo (player, am_clip, 1))
            return;
    }
    player->message = GOTCLIP;
    break;

case SPR_AMMO:
    if (!P_GiveAmmo (player, am_clip, 5))
        return;
    player->message = GOTCLIPBOX;
    break;

case SPR_ROCK:
    if (!P_GiveAmmo (player, am_misl, 1))
        return;
    player->message = GOTROCKET;
    break;

case SPR_BROK:
    if (!P_GiveAmmo (player, am_misl, 5))
        return;
    player->message = GOTROCKBOX;
    break;

case SPR_CELL:
    if (!P_GiveAmmo (player, am_cell, 1))
        return;
    player->message = GOTCELL;
    break;

case SPR_CELP:
    if (!P_GiveAmmo (player, am_cell, 5))
        return;
    player->message = GOTCELLBOX;
    break;

case SPR_SHEL:
    if (!P_GiveAmmo (player, am_shell, 1))

```

```

        return;
    player->message = GOTSHELLS;
    break;

case SPR_SBOX:
    if (!P_GiveAmmo (player, am_shell,5))
        return;
    player->message = GOTSHELLBOX;
    break;

case SPR_BPAK:
    if (!player->backpack)
    {
        for (i=0 ; i<NUMAMMO ; i++)
            player->maxammo[i] *= 2;
        player->backpack = true;
    }
    for (i=0 ; i<NUMAMMO ; i++)
        P_GiveAmmo (player, i, 1);
    player->message = GOTBACKPACK;
    break;

    // weapons
case SPR_BFUG:
    if (!P_GiveWeapon (player, wp_bfg, false) )
        return;
    player->message = GOTBFG9000;
    sound = sfx_wpnup;
    break;

case SPR_MGUN:
    if (!P_GiveWeapon (player, wp_chaingun, special->flags&MF_DROPPED) )
        return;
    player->message = GOTCHAINGUN;
    sound = sfx_wpnup;
    break;

case SPR_CSAW:
    if (!P_GiveWeapon (player, wp_chainsaw, false) )
        return;
    player->message = GOTCHAINSAW;
    sound = sfx_wpnup;
    break;

case SPR_LAUN:
    if (!P_GiveWeapon (player, wp_missile, false) )
        return;
    player->message = GOTLAUNCHER;
    sound = sfx_wpnup;
    break;

case SPR_PLAS:
    if (!P_GiveWeapon (player, wp_plasma, false) )
        return;
    player->message = GOTPLASMA;
    sound = sfx_wpnup;
    break;

case SPR_SHOT:
    if (!P_GiveWeapon (player, wp_shotgun, special->flags&MF_DROPPED) )
        return;
    player->message = GOTSHOTGUN;
    sound = sfx_wpnup;
    break;

```

```

case SPR_SGN2:
    if (!P_GiveWeapon (player, wp_supershotgun, special->flags&MF_DROPPED ) )
        return;
    player->message = GOTSHOTGUN2;
    sound = sfx_wpnup;
    break;

default:
    I_Error ("P_SpecialThing: Unknown gettable thing");
}

if (special->flags & MF_COUNTITEM)
    player->itemcount++;
P_RemoveMobj (special);
player->bonuscount += BONUSADD;
if (player == &players[consoleplayer])
    S_StartSound (NULL, sound);
}

//
// KillMobj
//
void
P_KillMobj
( mobj_t*      source,
  mobj_t*      target )
{
    mobjtype_t  item;
    mobj_t*     mo;

    target->flags &= ~(MF_SHOOTABLE|MF_FLOAT|MF_SKULLFLY);

    if (target->type != MT_SKULL)
        target->flags &= ~MF_NOGRAVITY;

    target->flags |= MF_CORPSE|MF_DROPOFF;
    target->height >>= 2;

    if (source && source->player)
    {
        // count for intermission
        if (target->flags & MF_COUNTKILL)
            source->player->killcount++;

        if (target->player)
            source->player->frags[target->player-players]++;
    }
    else if (!netgame && (target->flags & MF_COUNTKILL) )
    {
        // count all monster deaths,
        // even those caused by other monsters
        players[0].killcount++;
    }

    if (target->player)
    {
        // count environment kills against you
        if (!source)
            target->player->frags[target->player-players]++;

        target->flags &= ~MF_SOLID;
        target->player->playerstate = PST_DEAD;
        P_DropWeapon (target->player);
    }
}

```

```

        if (target->player == &players[consoleplayer]
            && automapactive)
        {
            // don't die in auto map,
            // switch view prior to dying
            AM_Stop ();
        }
    }

    if (target->health < -target->info->spawnhealth
        && target->info->xdeathstate)
    {
        P_SetMobjState (target, target->info->xdeathstate);
    }
    else
        P_SetMobjState (target, target->info->deathstate);
    target->tics -= P_Random()&3;

    if (target->tics < 1)
        target->tics = 1;

    //      I_StartSound (&actor->r, actor->info->deathsound);

    // Drop stuff.
    // This determines the kind of object spawned
    // during the death frame of a thing.
    switch (target->type)
    {
        case MT_WOLFSS:
        case MT_POSSESSED:
            item = MT_CLIP;
            break;

        case MT_SHOTGUY:
            item = MT_SHOTGUN;
            break;

        case MT_CHAINGUY:
            item = MT_CHAINGUN;
            break;

        default:
            return;
    }

    mo = P_SpawnMobj (target->x, target->y, ONFLOORZ, item);
    mo->flags |= MF_DROPPED;          // special versions of items
}

//
// P_DamageMobj
// Damages both enemies and players
// "inflictor" is the thing that caused the damage
// creature or missile, can be NULL (slime, etc)
// "source" is the thing to target after taking damage
// creature or NULL
// Source and inflictor are the same for melee attacks.
// Source can be NULL for slime, barrel explosions
// and other environmental stuff.
//

```

```

void
P_DamageMobj
( mobj_t*      target,
  mobj_t*      inflictor,
  mobj_t*      source,
  int          damage )
{
    unsigned    ang;
    int         saved;
    player_t*   player;
    fixed_t     thrust;
    int         temp;

    if ( !(target->flags & MF_SHOOTABLE) )
        return;          // shouldn't happen...

    if (target->health <= 0)
        return;

    if ( target->flags & MF_SKULLFLY )
    {
        target->momx = target->momy = target->momz = 0;
    }

    player = target->player;
    if (player && gameskill == sk_baby)
        damage >>= 1;      // take half damage in trainer mode

    // Some close combat weapons should not
    // inflict thrust and push the victim out of reach,
    // thus kick away unless using the chainsaw.
    if (inflictor
        && !(target->flags & MF_NOCLIP)
        && (!source
            || !source->player
            || source->player->readyweapon != wp_chainsaw))
    {
        ang = R_PointToAngle2 ( inflictor->x,
                                inflictor->y,
                                target->x,
                                target->y);

        thrust = damage*(FRACUNIT>>3)*100/target->info->mass;

        // make fall forwards sometimes
        if ( damage < 40
            && damage > target->health
            && target->z - inflictor->z > 64*FRACUNIT
            && (P_Random ()&1) )
        {
            ang += ANG180;
            thrust *= 4;
        }

        ang >>= ANGLETOfINESHIFT;
        target->momx += FixedMul (thrust, finecosine[ang]);
        target->momy += FixedMul (thrust, finesine[ang]);
    }

    // player specific
    if (player)
    {
        // end of game hell hack
        if (target->subsector->sector->special == 11

```



```

    && damage >= target->health)
{
    damage = target->health - 1;
}

// Below certain threshold,
// ignore damage in GOD mode, or with INVUL power.
if ( damage < 1000
    && ( (player->cheats&CF_GODMODE)
        || player->powers[pw_invulnerability] ) )
{
    return;
}

if (player->armortype)
{
    if (player->armortype == 1)
        saved = damage/3;
    else
        saved = damage/2;

    if (player->armorpoints <= saved)
    {
        // armor is used up
        saved = player->armorpoints;
        player->armortype = 0;
    }
    player->armorpoints -= saved;
    damage -= saved;
}
player->health -= damage;          // mirror mobj health here for Dave
if (player->health < 0)
    player->health = 0;

player->attacker = source;
player->damagecount += damage;    // add damage after armor / invuln

if (player->damagecount > 100)
    player->damagecount = 100;    // teleport stomp does 10k points...

temp = damage < 100 ? damage : 100;

if (player == &players[consoleplayer])
    I_Tactile (40,10,40+temp*2);
}

// do the damage
target->health -= damage;
if (target->health <= 0)
{
    P_KillMobj (source, target);
    return;
}

if ( (P_Random () < target->info->painchance)
    && !(target->flags&MF_SKULLFLY) )
{
    target->flags |= MF_JUSTHIT;    // fight back!

    P_SetMobjState (target, target->info->painstate);
}

target->reactiontime = 0;          // we're awake now...

```

```

if ( (!target->threshold || target->type == MT_VILE)
    && source && source != target
    && source->type != MT_VILE)
{
    // if not intent on another player,
    // chase after this one
    target->target = source;
    target->threshold = BASETHRESHOLD;
    if (target->state == &states[target->info->spawnstate]
        && target->info->seestate != S_NULL)
        P_SetMobjState (target, target->info->seestate);
}
}

```

## 9.8 p\_inter.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//
//-----

#ifndef __P_INTER__
#define __P_INTER__

#ifdef __GNUG__
#pragma interface
#endif

boolean      P_GivePower(player_t*, int);

#endif
//-----
//
// $Log:$
//
//-----

```

## 9.9 p\_lights.c

```

// Emacs style mode select   -*- C++ -*-
//-----

```

```

//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Handle Sector base lighting effects.
//     Muzzle flash?
//
//-----

static const char
rcsid[] = "$Id: p_lights.c,v 1.5 1997/02/03 22:45:11 b1 Exp $";

#include "z_zone.h"
#include "m_random.h"

#include "doomdef.h"
#include "p_local.h"

// State.
#include "r_state.h"

//
// FIRELIGHT FLICKER
//

//
// T_FireFlicker
//
void T_FireFlicker (fireflicker_t* flick)
{
    int         amount;

    if (--flick->count)
        return;

    amount = (P_Random()&3)*16;

    if (flick->sector->lightlevel - amount < flick->minlight)
        flick->sector->lightlevel = flick->minlight;
    else
        flick->sector->lightlevel = flick->maxlight - amount;

    flick->count = 4;
}

//
// P_SpawnFireFlicker

```

```

//
void P_SpawnFireFlicker (sector_t*      sector)
{
    fireflicker_t*      flick;

    // Note that we are resetting sector attributes.
    // Nothing special about it during gameplay.
    sector->special = 0;

    flick = Z_Malloc ( sizeof(*flick), PU_LEVSPEC, 0);

    P_AddThinker (&flick->thinker);

    flick->thinker.function.acp1 = (actionf_p1) T_FireFlicker;
    flick->sector = sector;
    flick->maxlight = sector->lightlevel;
    flick->minlight = P_FindMinSurroundingLight(sector,sector->lightlevel)+16;
    flick->count = 4;
}

```

```

//
// BROKEN LIGHT FLASHING
//

```

```

//
// T_LightFlash
// Do flashing lights.
//
void T_LightFlash (lightflash_t* flash)
{
    if (--flash->count)
        return;

    if (flash->sector->lightlevel == flash->maxlight)
    {
        flash->sector->lightlevel = flash->minlight;
        flash->count = (P_Random()&flash->mintime)+1;
    }
    else
    {
        flash->sector->lightlevel = flash->maxlight;
        flash->count = (P_Random()&flash->maxtime)+1;
    }
}

```

```

//
// P_SpawnLightFlash
// After the map has been loaded, scan each sector
// for specials that spawn thinkers
//
void P_SpawnLightFlash (sector_t*      sector)
{
    lightflash_t*      flash;

    // nothing special about it during gameplay
    sector->special = 0;

    flash = Z_Malloc ( sizeof(*flash), PU_LEVSPEC, 0);

```

```

P_AddThinker (&flash->thinker);

flash->thinker.function.acp1 = (actionf_p1) T_LightFlash;
flash->sector = sector;
flash->maxlight = sector->lightlevel;

flash->minlight = P_FindMinSurroundingLight(sector,sector->lightlevel);
flash->maxtime = 64;
flash->mintime = 7;
flash->count = (P_Random()&flash->maxtime)+1;
}

//
// STROBE LIGHT FLASHING
//

//
// T_StrobeFlash
//
void T_StrobeFlash (strobe_t*          flash)
{
    if (--flash->count)
        return;

    if (flash->sector->lightlevel == flash->minlight)
    {
        flash-> sector->lightlevel = flash->maxlight;
        flash->count = flash->brighttime;
    }
    else
    {
        flash-> sector->lightlevel = flash->minlight;
        flash->count =flash->darktime;
    }
}

}

//
// P_SpawnStrobeFlash
// After the map has been loaded, scan each sector
// for specials that spawn thinkers
//
void
P_SpawnStrobeFlash
( sector_t*          sector,
  int                fastOrSlow,
  int                inSync )
{
    strobe_t*          flash;

    flash = Z_Malloc ( sizeof(*flash), PU_LEVSPEC, 0);

    P_AddThinker (&flash->thinker);

    flash->sector = sector;
    flash->darktime = fastOrSlow;
    flash->brighttime = STROBEBRIGHT;
    flash->thinker.function.acp1 = (actionf_p1) T_StrobeFlash;
    flash->maxlight = sector->lightlevel;

```

```

flash->minlight = P_FindMinSurroundingLight(sector, sector->lightlevel);

if (flash->minlight == flash->maxlight)
    flash->minlight = 0;

// nothing special about it during gameplay
sector->special = 0;

if (!inSync)
    flash->count = (P_Random()&7)+1;
else
    flash->count = 1;
}

//
// Start strobing lights (usually from a trigger)
//
void EV_StartLightStrobing(line_t* line)
{
    int secnum;
    sector_t* sec;

    secnum = -1;
    while ((secnum = P_FindSectorFromLineTag(line, secnum)) >= 0)
    {
        sec = &sectors[secnum];
        if (sec->specialdata)
            continue;

        P_SpawnStrobeFlash (sec, SLOWDARK, 0);
    }
}

//
// TURN LINE'S TAG LIGHTS OFF
//
void EV_TurnTagLightsOff(line_t* line)
{
    int i;
    int j;
    int min;
    sector_t* sector;
    sector_t* tsec;
    line_t* templine;

    sector = sectors;

    for (j = 0; j < numsectors; j++, sector++)
    {
        if (sector->tag == line->tag)
        {
            min = sector->lightlevel;
            for (i = 0; i < sector->linecount; i++)
            {
                templine = sector->lines[i];
                tsec = getNextSector(templine, sector);
                if (!tsec)
                    continue;
                if (tsec->lightlevel < min)
                    min = tsec->lightlevel;
            }
            sector->lightlevel = min;
        }
    }
}

```

```

    }
}

//
// TURN LINE'S TAG LIGHTS ON
//
void
EV_LightTurnOn
( line_t*      line,
  int          bright )
{
    int          i;
    int          j;
    sector_t*    sector;
    sector_t*    temp;
    line_t*      templine;

    sector = sectors;

    for (i=0;i<numsectors;i++, sector++)
    {
        if (sector->tag == line->tag)
        {
            // bright = 0 means to search
            // for highest light level
            // surrounding sector
            if (!bright)
            {
                for (j = 0;j < sector->linecount; j++)
                {
                    templine = sector->lines[j];
                    temp = getNextSector(templine,sector);

                    if (!temp)
                        continue;

                    if (temp->lightlevel > bright)
                        bright = temp->lightlevel;
                }
                sector-> lightlevel = bright;
            }
        }
    }
}

//
// Spawn glowing light
//

void T_Glow(glow_t*      g)
{
    switch(g->direction)
    {
        case -1:
            // DOWN
            g->sector->lightlevel -= GLOWSPEED;
            if (g->sector->lightlevel <= g->minlight)
            {
                g->sector->lightlevel += GLOWSPEED;
                g->direction = 1;
            }
            break;
    }
}

```

```

        case 1:
            // UP
            g->sector->lightlevel += GLOWSPEED;
            if (g->sector->lightlevel >= g->maxlight)
            {
                g->sector->lightlevel -= GLOWSPEED;
                g->direction = -1;
            }
            break;
    }
}

void P_SpawnGlowingLight(sector_t*      sector)
{
    glow_t*      g;

    g = Z_Malloc( sizeof(*g), PU_LEVSPEC, 0);

    P_AddThinker(&g->thinker);

    g->sector = sector;
    g->minlight = P_FindMinSurroundingLight(sector, sector->lightlevel);
    g->maxlight = sector->lightlevel;
    g->thinker.function.acp1 = (actionf_p1) T_Glow;
    g->direction = -1;

    sector->special = 0;
}

```

## 9.10 p\_local.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      Play functions, animation, global header.
//
//-----

#ifdef __P_LOCAL__
#define __P_LOCAL__

#ifdef __R_LOCAL__
#include "r_local.h"
#endif

#define FLOATSPEED          (FRACUNIT*4)

```



```

#define MAXHEALTH                100
#define VIEWHEIGHT                (41*FRACUNIT)

// mapblocks are used to check movement
// against lines and things
#define MAPBLOCKUNITS            128
#define MAPBLOCKSIZE              (MAPBLOCKUNITS*FRACUNIT)
#define MAPBLOCKSHIFT            (FRACBITS+7)
#define MAPBMASK                  (MAPBLOCKSIZE-1)
#define MAPBTOFRAC                (MAPBLOCKSHIFT-FRACBITS)

// player radius for movement checking
#define PLAYERRADIUS              16*FRACUNIT

// MAXRADIUS is for precalculated sector block boxes
// the spider demon is larger,
// but we do not have any moving sectors nearby
#define MAXRADIUS                  32*FRACUNIT

#define GRAVITY                    FRACUNIT
#define MAXMOVE                    (30*FRACUNIT)

#define USERANGE                  (64*FRACUNIT)
#define MELEERANGE                  (64*FRACUNIT)
#define MISSILERANGE              (32*64*FRACUNIT)

// follow a player exclusively for 3 seconds
#define          BASETHRESHOLD              100

//
// P_TICK
//

// both the head and tail of the thinker list
extern          thinker_t          thinkercap;

void P_InitThinkers (void);
void P_AddThinker (thinker_t* thinker);
void P_RemoveThinker (thinker_t* thinker);

//
// P_PSPR
//
void P_SetupPsprites (player_t* curplayer);
void P_MovePsprites (player_t* curplayer);
void P_DropWeapon (player_t* player);

//
// P_USER
//
void          P_PlayerThink (player_t* player);

//
// P_MOBJ
//
#define ONFLOORZ                  MININT

```

```

#define ONCEILLINGZ                MAXINT

// Time interval for item respawning.
#define ITEMQUESIZE                128

extern mapthing_t                  itemrespawnque[ITEMQUESIZE];
extern int                         itemrespawntime[ITEMQUESIZE];
extern int                         iquehead;
extern int                         iquetail;

void P_RespawnSpecials (void);

mobj_t*
P_SpawnMobj
( fixed_t      x,
  fixed_t      y,
  fixed_t      z,
  mobjtype_t    type );

void      P_RemoveMobj (mobj_t* th);
boolean    P_SetMobjState (mobj_t* mobj, statenum_t state);
void      P_MobjThinker (mobj_t* mobj);

void      P_SpawnPuff (fixed_t x, fixed_t y, fixed_t z);
void      P_SpawnBlood (fixed_t x, fixed_t y, fixed_t z, int damage);
mobj_t* P_SpawnMissile (mobj_t* source, mobj_t* dest, mobjtype_t type);
void      P_SpawnPlayerMissile (mobj_t* source, mobjtype_t type);

//
// P_ENEMY
//
void P_NoiseAlert (mobj_t* target, mobj_t* emmitter);

//
// P_MAPUTL
//
typedef struct
{
    fixed_t      x;
    fixed_t      y;
    fixed_t      dx;
    fixed_t      dy;

} divline_t;

typedef struct
{
    fixed_t      frac;                // along trace line
    boolean      isaline;
    union {
        mobj_t*   thing;
        line_t*   line;
    }
    d;
} intercept_t;

#define MAXINTERCEPTS            128

extern intercept_t                  intercepts[MAXINTERCEPTS];
extern intercept_t*                 intercept_p;

typedef boolean (*traverser_t) (intercept_t *in);

```

```

fixed_t P_AproxDistance (fixed_t dx, fixed_t dy);
int      P_PointOnLineSide (fixed_t x, fixed_t y, line_t* line);
int      P_PointOnDivlineSide (fixed_t x, fixed_t y, divline_t* line);
void      P_MakeDivline (line_t* li, divline_t* dl);
fixed_t P_InterceptVector (divline_t* v2, divline_t* v1);
int      P_BoxOnLineSide (fixed_t* tbox, line_t* ld);

extern fixed_t      opentop;
extern fixed_t      openbottom;
extern fixed_t      openrange;
extern fixed_t      lowfloor;

void      P_LineOpening (line_t* linedef);

boolean P_BlockLinesIterator (int x, int y, boolean(*func)(line_t* ));
boolean P_BlockThingsIterator (int x, int y, boolean(*func)(mobj_t* ));

#define PT_ADDLINES      1
#define PT_ADDTHINGS      2
#define PT_EARLYOUT      4

extern divline_t      trace;

boolean
P_PathTraverse
( fixed_t      x1,
  fixed_t      y1,
  fixed_t      x2,
  fixed_t      y2,
  int          flags,
  boolean      (*trav) (intercept_t *));

void P_UnsetThingPosition (mobj_t* thing);
void P_SetThingPosition (mobj_t* thing);

//
// P_MAP
//

// If "floatok" true, move would be ok
// if within "tmfloorz - tmceilingz".
extern boolean      floatok;
extern fixed_t      tmfloorz;
extern fixed_t      tmceilingz;

extern      line_t*      ceilingline;

boolean P_CheckPosition (mobj_t *thing, fixed_t x, fixed_t y);
boolean P_TryMove (mobj_t* thing, fixed_t x, fixed_t y);
boolean P_TeleportMove (mobj_t* thing, fixed_t x, fixed_t y);
void      P_SlideMove (mobj_t* mo);
boolean P_CheckSight (mobj_t* t1, mobj_t* t2);
void      P_UseLines (player_t* player);

boolean P_ChangeSector (sector_t* sector, boolean crunch);

extern mobj_t*      linetarget;      // who got hit (or NULL)

fixed_t
P_AimLineAttack
( mobj_t*      t1,
  angle_t      angle,
  fixed_t      distance );

```

```

void
P_LineAttack
( mobj_t*      t1,
  angle_t      angle,
  fixed_t      distance,
  fixed_t      slope,
  int          damage );

void
P_RadiusAttack
( mobj_t*      spot,
  mobj_t*      source,
  int          damage );

//
// P_SETUP
//
extern byte*      rejectmatrix;      // for fast sight rejection
extern short*     blockmaplump;      // offsets in blockmap are from here
extern short*     blockmap;
extern int        bmapwidth;
extern int        bmapheight;      // in mapblocks
extern fixed_t    bmaporgx;
extern fixed_t    bmaporgy;      // origin of block map
extern mobj_t**   blocklinks;      // for thing chains

//
// P_INTER
//
extern int        maxammo[NUMAMMO];
extern int        clipammo[NUMAMMO];

void
P_TouchSpecialThing
( mobj_t*      special,
  mobj_t*      toucher );

void
P_DamageMobj
( mobj_t*      target,
  mobj_t*      inflictor,
  mobj_t*      source,
  int          damage );

//
// P_SPEC
//
#include "p_spec.h"

#endif      // __P_LOCAL__
//-----
//
// $Log:$
//
//-----

```

## 9.11 p\_map.c

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Movement, collision handling.
//      Shooting and aiming.
//-----

static const char
rcsid[] = "$Id: p_map.c,v 1.5 1997/02/03 22:45:11 b1 Exp $";

#include <stdlib.h>

#include "m_bbox.h"
#include "m_random.h"
#include "i_system.h"

#include "doomdef.h"
#include "p_local.h"

#include "s_sound.h"

// State.
#include "doomstat.h"
#include "r_state.h"
// Data.
#include "sounds.h"

fixed_t          tmbbox[4];
mobj_t*          tmthing;
int              tmflags;
fixed_t          tmx;
fixed_t          tmy;

// If "floatok" true, move would be ok
// if within "tmfloorz - tmceilingz".
boolean          floatok;

fixed_t          tmfloorz;
fixed_t          tmceilingz;
fixed_t          tmdropoffz;

// keep track of the line that lowers the ceiling,
// so missiles don't explode against sky hack walls
line_t*          ceilingline;
```

```
// keep track of special lines as they are hit,
// but don't process them until the move is proven valid
#define MAXSPECIALCROSS 8
```

```
line_t*          spechit[MAXSPECIALCROSS];
int              numspechit;
```

```
//
// TELEPORT MOVE
//
```

```
//
// PIT_StompThing
//
```

```
boolean PIT_StompThing (mobj_t* thing)
{
    fixed_t      blockdist;

    if (!(thing->flags & MF_SHOOTABLE) )
        return true;

    blockdist = thing->radius + tmthing->radius;

    if ( abs(thing->x - tmx) >= blockdist
        || abs(thing->y - tmy) >= blockdist )
    {
        // didn't hit it
        return true;
    }

    // don't clip against self
    if (thing == tmthing)
        return true;

    // monsters don't stomp things except on boss level
    if ( !tmthing->player && gamemap != 30)
        return false;

    P_DamageMobj (thing, tmthing, tmthing, 10000);

    return true;
}
```

```
//
// P_TeleportMove
//
```

```
boolean
P_TeleportMove
( mobj_t*      thing,
  fixed_t      x,
  fixed_t      y )
{
    int          xl;
    int          xh;
    int          yl;
    int          yh;
    int          bx;
    int          by;

    subsector_t* newsubsec;
```

```

// kill anything occupying the position
tmthing = thing;
tmflags = thing->flags;

tmx = x;
tmy = y;

tmbbox[BOXTOP] = y + tmthing->radius;
tmbbox[BOXBOTTOM] = y - tmthing->radius;
tmbbox[BOXRIGHT] = x + tmthing->radius;
tmbbox[BOXLEFT] = x - tmthing->radius;

newsubsec = R_PointInSubsector (x,y);
ceilingline = NULL;

// The base floor/ceiling is from the subsector
// that contains the point.
// Any contacted lines the step closer together
// will adjust them.
tmfloorz = tmdropoffz = newsubsec->sector->floorheight;
tmceilingz = newsubsec->sector->ceilingheight;

validcount++;
numspechit = 0;

// stomp on any things contacted
xl = (tmbbox[BOXLEFT] - bmaporgx - MAXRADIUS)>>MAPBLOCKSHIFT;
xh = (tmbbox[BOXRIGHT] - bmaporgx + MAXRADIUS)>>MAPBLOCKSHIFT;
yl = (tmbbox[BOXBOTTOM] - bmaporgy - MAXRADIUS)>>MAPBLOCKSHIFT;
yh = (tmbbox[BOXTOP] - bmaporgy + MAXRADIUS)>>MAPBLOCKSHIFT;

for (bx=xl ; bx<=xh ; bx++)
    for (by=yl ; by<=yh ; by++)
        if (!P_BlockThingsIterator(bx,by,PIT_StompThing))
            return false;

// the move is ok,
// so link the thing into its new position
P_UnsetThingPosition (thing);

thing->floorz = tmfloorz;
thing->ceilingz = tmceilingz;
thing->x = x;
thing->y = y;

P_SetThingPosition (thing);

return true;
}

//
// MOVEMENT ITERATOR FUNCTIONS
//

//
// PIT_CheckLine
// Adjusts tmfloorz and tmceilingz as lines are contacted
//
boolean PIT_CheckLine (line_t* ld)
{
    if (tmbbox[BOXRIGHT] <= ld->bbox[BOXLEFT]
        || tmbbox[BOXLEFT] >= ld->bbox[BOXRIGHT]
        || tmbbox[BOXTOP] <= ld->bbox[BOXBOTTOM]

```

```

    || tmbbox[BOXBOTTOM] >= ld->bbox[BOXTOP] )
    return true;

if (P_BoxOnLineSide (tmbbox, ld) != -1)
    return true;

// A line has been hit

// The moving thing's destination position will cross
// the given line.
// If this should not be allowed, return false.
// If the line is special, keep track of it
// to process later if the move is proven ok.
// NOTE: specials are NOT sorted by order,
// so two special lines that are only 8 pixels apart
// could be crossed in either order.

if (!ld->backsector)
    return false;                // one sided line

if (!(tmthing->flags & MF_MISSILE) )
{
    if ( ld->flags & ML_BLOCKING )
        return false;           // explicitly blocking everything

    if ( !tmthing->player && ld->flags & ML_BLOCKMONSTERS )
        return false;           // block monsters only
}

// set openrange, opentop, openbottom
P_LineOpening (ld);

// adjust floor / ceiling heights
if (opentop < tmceilingz)
{
    tmceilingz = opentop;
    ceilingline = ld;
}

if (openbottom > tmfloorz)
    tmfloorz = openbottom;

if (lowfloor < tmdropoffz)
    tmdropoffz = lowfloor;

// if contacted a special line, add it to the list
if (ld->special)
{
    spechit[numspechit] = ld;
    numspechit++;
}

return true;
}

//
// PIT_CheckThing
//
boolean PIT_CheckThing (mobj_t* thing)
{
    fixed_t          blockdist;
    boolean           solid;
    int               damage;

    if (!(thing->flags & (MF_SOLID|MF_SPECIAL|MF_SHOOTABLE) ))

```



```

    return true;

blockdist = thing->radius + tmthing->radius;

if ( abs(thing->x - tmx) >= blockdist
    || abs(thing->y - tmy) >= blockdist )
{
    // didn't hit it
    return true;
}

// don't clip against self
if (thing == tmthing)
    return true;

// check for skulls slamming into things
if (tmthing->flags & MF_SKULLFLY)
{
    damage = ((P_Random()%8)+1)*tmthing->info->damage;

    P_DamageMobj (thing, tmthing, tmthing, damage);

    tmthing->flags &= ~MF_SKULLFLY;
    tmthing->momx = tmthing->momy = tmthing->momz = 0;

    P_SetMobjState (tmthing, tmthing->info->spawnstate);

    return false;                // stop moving
}

// missiles can hit other things
if (tmthing->flags & MF_MISSILE)
{
    // see if it went over / under
    if (tmthing->z > thing->z + thing->height)
        return true;                // overhead
    if (tmthing->z+tmthing->height < thing->z)
        return true;                // underneath

    if (tmthing->target && (
        tmthing->target->type == thing->type ||
        (tmthing->target->type == MT_KNIGHT && thing->type == MT_BRUISER)||
        (tmthing->target->type == MT_BRUISER && thing->type == MT_KNIGHT) ) )
    {
        // Don't hit same species as originator.
        if (thing == tmthing->target)
            return true;

        if (thing->type != MT_PLAYER)
        {
            // Explode, but do no damage.
            // Let players missile other players.
            return false;
        }
    }

    if (!(thing->flags & MF_SHOOTABLE) )
    {
        // didn't do any damage
        return !(thing->flags & MF_SOLID);
    }

    // damage / explode
    damage = ((P_Random()%8)+1)*tmthing->info->damage;

```

```

    P_DamageMobj (thing, tmthing, tmthing->target, damage);

    // don't traverse any more
    return false;
}

// check for special pickup
if (thing->flags & MF_SPECIAL)
{
    solid = thing->flags&MF_SOLID;
    if (tmflags&MF_PICKUP)
    {
        // can remove thing
        P_TouchSpecialThing (thing, tmthing);
    }
    return !solid;
}

return !(thing->flags & MF_SOLID);
}

//
// MOVEMENT CLIPPING
//

//
// P_CheckPosition
// This is purely informative, nothing is modified
// (except things picked up).
//
// in:
//  a mobj_t (can be valid or invalid)
//  a position to be checked
//  (doesn't need to be related to the mobj_t->x,y)
//
// during:
//  special things are touched if MF_PICKUP
//  early out on solid lines?
//
// out:
//  newsubsec
//  floorz
//  ceilingz
//  tmdropoffz
//  the lowest point contacted
//  (monsters won't move to a dropoff)
//  speciallines[]
//  numspeciallines
//
boolean
P_CheckPosition
( mobj_t*      thing,
  fixed_t      x,
  fixed_t      y )
{
    int          xl;
    int          xh;
    int          yl;
    int          yh;
    int          bx;
    int          by;
    subsector_t* newsubsec;

    tmthing = thing;

```

```

tmflags = thing->flags;

tmx = x;
tmy = y;

tmbbox[BOXTOP] = y + tmthing->radius;
tmbbox[BOXBOTTOM] = y - tmthing->radius;
tmbbox[BOXRIGHT] = x + tmthing->radius;
tmbbox[BOXLEFT] = x - tmthing->radius;

newsubsec = R_PointInSubsector (x,y);
ceilingline = NULL;

// The base floor / ceiling is from the subsector
// that contains the point.
// Any contacted lines the step closer together
// will adjust them.
tmfloorz = tmdropoffz = newsubsec->sector->floorheight;
tmceilingz = newsubsec->sector->ceilingheight;

validcount++;
numspechit = 0;

if ( tmflags & MF_NOCLIP )
    return true;

// Check things first, possibly picking things up.
// The bounding box is extended by MAXRADIUS
// because mobj_ts are grouped into mapblocks
// based on their origin point, and can overlap
// into adjacent blocks by up to MAXRADIUS units.
xl = (tmbbox[BOXLEFT] - bmaporgx - MAXRADIUS)>>MAPBLOCKSHIFT;
xh = (tmbbox[BOXRIGHT] - bmaporgx + MAXRADIUS)>>MAPBLOCKSHIFT;
yl = (tmbbox[BOXBOTTOM] - bmaporgy - MAXRADIUS)>>MAPBLOCKSHIFT;
yh = (tmbbox[BOXTOP] - bmaporgy + MAXRADIUS)>>MAPBLOCKSHIFT;

for (bx=xl ; bx<=xh ; bx++)
    for (by=yl ; by<=yh ; by++)
        if (!P_BlockThingsIterator(bx,by,PIT_CheckThing))
            return false;

// check lines
xl = (tmbbox[BOXLEFT] - bmaporgx)>>MAPBLOCKSHIFT;
xh = (tmbbox[BOXRIGHT] - bmaporgx)>>MAPBLOCKSHIFT;
yl = (tmbbox[BOXBOTTOM] - bmaporgy)>>MAPBLOCKSHIFT;
yh = (tmbbox[BOXTOP] - bmaporgy)>>MAPBLOCKSHIFT;

for (bx=xl ; bx<=xh ; bx++)
    for (by=yl ; by<=yh ; by++)
        if (!P_BlockLinesIterator (bx,by,PIT_CheckLine))
            return false;

return true;
}

//
// P_TryMove
// Attempt to move to a new position,
// crossing special lines unless MF_TELEPORT is set.
//
boolean
P_TryMove
( mobj_t*      thing,
  fixed_t      x,

```

```

fixed_t      y )
{
    fixed_t      oldx;
    fixed_t      oldy;
    int          side;
    int          oldside;
    line_t*      ld;

    floatok = false;
    if (!P_CheckPosition (thing, x, y))
        return false;          // solid wall or thing

    if ( !(thing->flags & MF_NOCLIP) )
    {
        if (tmceilingz - tmfloorz < thing->height)
            return false;        // doesn't fit

        floatok = true;

        if ( !(thing->flags&MF_TELEPORT)
            &&tmceilingz - thing->z < thing->height)
            return false;        // mobj must lower itself to fit

        if ( !(thing->flags&MF_TELEPORT)
            && tmfloorz - thing->z > 24*FRACUNIT )
            return false;        // too big a step up

        if ( !(thing->flags&(MF_DROPPOFF|MF_FLOAT))
            && tmfloorz - tmdropoffz > 24*FRACUNIT )
            return false;        // don't stand over a dropoff
    }

    // the move is ok,
    // so link the thing into its new position
    P_UnsetThingPosition (thing);

    oldx = thing->x;
    oldy = thing->y;
    thing->floorz = tmfloorz;
    thing->ceilingz = tmceilingz;
    thing->x = x;
    thing->y = y;

    P_SetThingPosition (thing);

    // if any special lines were hit, do the effect
    if ( !(thing->flags&(MF_TELEPORT|MF_NOCLIP)) )
    {
        while (numspechit--)
        {
            // see if the line was crossed
            ld = spechit[numspechit];
            side = P_PointOnLineSide (thing->x, thing->y, ld);
            oldside = P_PointOnLineSide (oldx, oldy, ld);
            if (side != oldside)
            {
                if (ld->special)
                    P_CrossSpecialLine (ld-lines, oldside, thing);
            }
        }
    }

    return true;
}

```

```

//
// P_ThingHeightClip
// Takes a valid thing and adjusts the thing->floorz,
// thing->ceilingz, and possibly thing->z.
// This is called for all nearby monsters
// whenever a sector changes height.
// If the thing doesn't fit,
// the z will be set to the lowest value
// and false will be returned.
//
boolean P_ThingHeightClip (mobj_t* thing)
{
    boolean                onfloor;

    onfloor = (thing->z == thing->floorz);

    P_CheckPosition (thing, thing->x, thing->y);
    // what about stranding a monster partially off an edge?

    thing->floorz = tmfloorz;
    thing->ceilingz = tmceilingz;

    if (onfloor)
    {
        // walking monsters rise and fall with the floor
        thing->z = thing->floorz;
    }
    else
    {
        // don't adjust a floating monster unless forced to
        if (thing->z+thing->height > thing->ceilingz)
            thing->z = thing->ceilingz - thing->height;
    }

    if (thing->ceilingz - thing->floorz < thing->height)
        return false;

    return true;
}

```

```

//
// SLIDE MOVE
// Allows the player to slide along any angled walls.
//

```

```

fixed_t                bestslidefrac;
fixed_t                secondslidefrac;

line_t*                bestslideline;
line_t*                secondslideline;

mobj_t*                slidemo;

fixed_t                tmxmove;
fixed_t                tmymove;

```

```

//
// P_HitSlideLine
// Adjusts the xmove / ymove
// so that the next move will slide along the wall.
//

```

```

void P_HitSlideLine (line_t* ld)
{
    int                side;

    angle_t            lineangle;
    angle_t            moveangle;
    angle_t            deltaangle;

    fixed_t            movelen;
    fixed_t            newlen;

    if (ld->slopetype == ST_HORIZONTAL)
    {
        tmymove = 0;
        return;
    }

    if (ld->slopetype == ST_VERTICAL)
    {
        tmxmove = 0;
        return;
    }

    side = P_PointOnLineSide (slidemo->x, slidemo->y, ld);

    lineangle = R_PointToAngle2 (0,0, ld->dx, ld->dy);

    if (side == 1)
        lineangle += ANG180;

    moveangle = R_PointToAngle2 (0,0, tmxmove, tmymove);
    deltaangle = moveangle-lineangle;

    if (deltaangle > ANG180)
        deltaangle += ANG180;
    //      I_Error ("SlideLine: ang>ANG180");

    lineangle >>= ANGLETOFINESHIFT;
    deltaangle >>= ANGLETOFINESHIFT;

    movelen = P_AproxDistance (tmxmove, tmymove);
    newlen = FixedMul (movelen, finecosine[deltaangle]);

    tmxmove = FixedMul (newlen, finecosine[lineangle]);
    tmymove = FixedMul (newlen, finesine[lineangle]);
}

//
// PTR_SlideTraverse
//
boolean PTR_SlideTraverse (intercept_t* in)
{
    line_t*            li;

    if (!in->isaline)
        I_Error ("PTR_SlideTraverse: not a line?");

    li = in->d.line;

    if ( ! (li->flags & ML_TWOSIDED) )
    {
        if (P_PointOnLineSide (slidemo->x, slidemo->y, li))
        {

```

```

        // don't hit the back side
        return true;
    }
    goto isblocking;
}

// set openrange, opentop, openbottom
P_LineOpening (li);

if (openrange < slidemo->height)
    goto isblocking;           // doesn't fit

if (opentop - slidemo->z < slidemo->height)
    goto isblocking;           // mobj is too high

if (openbottom - slidemo->z > 24*FRACUNIT )
    goto isblocking;           // too big a step up

// this line doesn't block movement
return true;

// the line does block movement,
// see if it is closer than best so far
isblocking:
    if (in->frac < bestslidefrac)
    {
        secondslidefrac = bestslidefrac;
        secondslideline = bestslideline;
        bestslidefrac = in->frac;
        bestslideline = li;
    }

    return false;           // stop
}

```

```

//
// P_SlideMove
// The momx / momy move is bad, so try to slide
// along a wall.
// Find the first line hit, move flush to it,
// and slide along it
//
// This is a kludgy mess.
//
void P_SlideMove (mobj_t* mo)
{
    fixed_t          leadx;
    fixed_t          leady;
    fixed_t          trailx;
    fixed_t          traily;
    fixed_t          newx;
    fixed_t          newy;
    int              hitcount;

    slidemo = mo;
    hitcount = 0;

retry:
    if (++hitcount == 3)
        goto stairstep;           // don't loop forever

    // trace along the three leading corners

```

```

if (mo->momx > 0)
{
    leadx = mo->x + mo->radius;
    trailx = mo->x - mo->radius;
}
else
{
    leadx = mo->x - mo->radius;
    trailx = mo->x + mo->radius;
}

if (mo->momy > 0)
{
    leady = mo->y + mo->radius;
    traily = mo->y - mo->radius;
}
else
{
    leady = mo->y - mo->radius;
    traily = mo->y + mo->radius;
}

bestslidefrac = FRACUNIT+1;

P_PathTraverse ( leadx, leady, leadx+mo->momx, leady+mo->momy,
                PT_ADDLINES, PTR_SlideTraverse );
P_PathTraverse ( trailx, leady, trailx+mo->momx, leady+mo->momy,
                PT_ADDLINES, PTR_SlideTraverse );
P_PathTraverse ( leadx, traily, leadx+mo->momx, traily+mo->momy,
                PT_ADDLINES, PTR_SlideTraverse );

// move up to the wall
if (bestslidefrac == FRACUNIT+1)
{
    // the move most have hit the middle, so stairstep
    stairstep:
    if (!P_TryMove (mo, mo->x, mo->y + mo->momy))
        P_TryMove (mo, mo->x + mo->momx, mo->y);
    return;
}

// fudge a bit to make sure it doesn't hit
bestslidefrac -= 0x800;
if (bestslidefrac > 0)
{
    newx = FixedMul (mo->momx, bestslidefrac);
    newy = FixedMul (mo->momy, bestslidefrac);

    if (!P_TryMove (mo, mo->x+newx, mo->y+newy))
        goto stairstep;
}

// Now continue along the wall.
// First calculate remainder.
bestslidefrac = FRACUNIT-(bestslidefrac+0x800);

if (bestslidefrac > FRACUNIT)
    bestslidefrac = FRACUNIT;

if (bestslidefrac <= 0)
    return;

tmxmove = FixedMul (mo->momx, bestslidefrac);
tmymove = FixedMul (mo->momy, bestslidefrac);

```



```

P_HitSlideLine (bestslideline);          // clip the moves

mo->momx = tmxmove;
mo->momy = tmymove;

if (!P_TryMove (mo, mo->x+tmxmove, mo->y+tmymove))
{
    goto retry;
}
}

//
// P_LineAttack
//
mobj_t*          linetarget;          // who got hit (or NULL)
mobj_t*          shootthing;

// Height if not aiming up or down
// ????: use slope for monsters?
fixed_t          shootz;

int              la_damage;
fixed_t          attackrange;

fixed_t          aimslope;

// slopes to top and bottom of target
extern fixed_t    topslope;
extern fixed_t    bottomslope;

//
// PTR_AimTraverse
// Sets linetaget and aimslope when a target is aimed at.
//
boolean
PTR_AimTraverse (intercept_t* in)
{
    line_t*        li;
    mobj_t*        th;
    fixed_t        slope;
    fixed_t        thingtopslope;
    fixed_t        thingbottomslope;
    fixed_t        dist;

    if (in->isaline)
    {
        li = in->d.line;

        if ( !(li->flags & ML_TWOSIDED) )
            return false;          // stop

        // Crosses a two sided line.
        // A two sided line will restrict
        // the possible target ranges.
        P_LineOpening (li);

        if (openbottom >= opentop)
            return false;          // stop

        dist = FixedMul (attackrange, in->frac);

        if (li->frontsector->floorheight != li->backsector->floorheight)
        {

```

```

        slope = FixedDiv (openbottom - shootz , dist);
        if (slope > bottomslope)
            bottomslope = slope;
    }

    if (li->frontsector->ceilingheight != li->backsector->ceilingheight)
    {
        slope = FixedDiv (opentop - shootz , dist);
        if (slope < topslope)
            topslope = slope;
    }

    if (topslope <= bottomslope)
        return false;                // stop

    return true;                      // shot continues
}

// shoot a thing
th = in->d.thing;
if (th == shootthing)
    return true;                    // can't shoot self

if (!(th->flags&MF_SHOOTABLE))
    return true;                    // corpse or something

// check angles to see if the thing can be aimed at
dist = FixedMul (attackrange, in->frac);
thingtopslope = FixedDiv (th->z+th->height - shootz , dist);

if (thingtopslope < bottomslope)
    return true;                    // shot over the thing

thingbottomslope = FixedDiv (th->z - shootz, dist);

if (thingbottomslope > topslope)
    return true;                    // shot under the thing

// this thing can be hit!
if (thingtopslope > topslope)
    thingtopslope = topslope;

if (thingbottomslope < bottomslope)
    thingbottomslope = bottomslope;

aimslope = (thingtopslope+thingbottomslope)/2;
linetarget = th;

return false;                      // don't go any farther
}

//
// PTR_ShootTraverse
//
boolean PTR_ShootTraverse (intercept_t* in)
{
    fixed_t          x;
    fixed_t          y;
    fixed_t          z;
    fixed_t          frac;

    line_t*          li;

    mobj_t*          th;

```

```

fixed_t          slope;
fixed_t          dist;
fixed_t          thingtopslope;
fixed_t          thingbottomslope;

if (in->isaline)
{
    li = in->d.line;

    if (li->special)
        P_ShootSpecialLine (shootthing, li);

    if ( !(li->flags & ML_TWOSIDED) )
        goto hitline;

    // crosses a two sided line
    P_LineOpening (li);

    dist = FixedMul (attackrange, in->frac);

    if (li->frontsector->floorheight != li->backsector->floorheight)
    {
        slope = FixedDiv (openbottom - shootz , dist);
        if (slope > aimslope)
            goto hitline;
    }

    if (li->frontsector->ceilingheight != li->backsector->ceilingheight)
    {
        slope = FixedDiv (opentop - shootz , dist);
        if (slope < aimslope)
            goto hitline;
    }

    // shot continues
    return true;

    // hit line
hitline:
    // position a bit closer
    frac = in->frac - FixedDiv (4*FRACUNIT,attackrange);
    x = trace.x + FixedMul (trace.dx, frac);
    y = trace.y + FixedMul (trace.dy, frac);
    z = shootz + FixedMul (aimslope, FixedMul(frac, attackrange));

    if (li->frontsector->ceilingpic == skyflatnum)
    {
        // don't shoot the sky!
        if (z > li->frontsector->ceilingheight)
            return false;

        // it's a sky hack wall
        if (li->backsector && li->backsector->ceilingpic == skyflatnum)
            return false;
    }

    // Spawn bullet puffs.
    P_SpawnPuff (x,y,z);

    // don't go any farther
    return false;
}

```

```

// shoot a thing
th = in->d.thing;
if (th == shootthing)
    return true;                // can't shoot self

if (!(th->flags & MF_SHOOTABLE))
    return true;                // corpse or something

// check angles to see if the thing can be aimed at
dist = FixedMul (attackrange, in->frac);
thingtopslope = FixedDiv (th->z+th->height - shootz , dist);

if (thingtopslope < aimslope)
    return true;                // shot over the thing

thingbottomslope = FixedDiv (th->z - shootz, dist);

if (thingbottomslope > aimslope)
    return true;                // shot under the thing

// hit thing
// position a bit closer
frac = in->frac - FixedDiv (10*FRACUNIT, attackrange);

x = trace.x + FixedMul (trace.dx, frac);
y = trace.y + FixedMul (trace.dy, frac);
z = shootz + FixedMul (aimslope, FixedMul(frac, attackrange));

// Spawn bullet puffs or blod spots,
// depending on target type.
if (in->d.thing->flags & MF_NOBLOOD)
    P_SpawnPuff (x,y,z);
else
    P_SpawnBlood (x,y,z, la_damage);

if (la_damage)
    P_DamageMobj (th, shootthing, shootthing, la_damage);

// don't go any farther
return false;
}

//
// P_AimLineAttack
//
fixed_t
P_AimLineAttack
( mobj_t*      t1,
  angle_t      angle,
  fixed_t      distance )
{
    fixed_t      x2;
    fixed_t      y2;

    angle >>= ANGLETOFINESHIFT;
    shootthing = t1;

    x2 = t1->x + (distance>>FRACBITS)*finecosine[angle];
    y2 = t1->y + (distance>>FRACBITS)*finesine[angle];
    shootz = t1->z + (t1->height>>1) + 8*FRACUNIT;

    // can't shoot outside view angles

```

```

    topslope = 100*FRACUNIT/160;
    bottomslope = -100*FRACUNIT/160;

    attackrange = distance;
    linetarget = NULL;

    P_PathTraverse ( t1->x, t1->y,
                    x2, y2,
                    PT_ADDLINES|PT_ADDTHINGS,
                    PTR_AimTraverse );

    if (linetarget)
        return aimslope;

    return 0;
}

//
// P_LineAttack
// If damage == 0, it is just a test trace
// that will leave linetarget set.
//
void
P_LineAttack
( mobj_t*      t1,
  angle_t      angle,
  fixed_t      distance,
  fixed_t      slope,
  int          damage )
{
    fixed_t      x2;
    fixed_t      y2;

    angle >>= ANGLETOFINESHIFT;
    shootthing = t1;
    la_damage = damage;
    x2 = t1->x + (distance>>FRACBITS)*finecosine[angle];
    y2 = t1->y + (distance>>FRACBITS)*finesine[angle];
    shootz = t1->z + (t1->height>>1) + 8*FRACUNIT;
    attackrange = distance;
    aimslope = slope;

    P_PathTraverse ( t1->x, t1->y,
                    x2, y2,
                    PT_ADDLINES|PT_ADDTHINGS,
                    PTR_ShootTraverse );
}

//
// USE LINES
//
mobj_t*          usething;

boolean          PTR_UseTraverse (intercept_t* in)
{
    int          side;

    if (!in->d.line->special)
    {
        P_LineOpening (in->d.line);
        if (openrange <= 0)
        {

```

```

        S_StartSound (usething, sfx_noway);

        // can't use through a wall
        return false;
    }
    // not a special line, but keep checking
    return true ;
}

side = 0;
if (P_PointOnLineSide (usething->x, usething->y, in->d.line) == 1)
    side = 1;

//          return false;          // don't use back side

P_UseSpecialLine (usething, in->d.line, side);

// can't use for than one special line in a row
return false;
}

//
// P_UseLines
// Looks for special lines in front of the player to activate.
//
void P_UseLines (player_t*      player)
{
    int          angle;
    fixed_t      x1;
    fixed_t      y1;
    fixed_t      x2;
    fixed_t      y2;

    usething = player->mo;

    angle = player->mo->angle >> ANGLETOFINESHIFT;

    x1 = player->mo->x;
    y1 = player->mo->y;
    x2 = x1 + (USERANGE>>FRACBITS)*finecosine[angle];
    y2 = y1 + (USERANGE>>FRACBITS)*finesine[angle];

    P_PathTraverse ( x1, y1, x2, y2, PT_ADDLINES, PTR_UseTraverse );
}

//
// RADIUS ATTACK
//
mobj_t*      bombsource;
mobj_t*      bombspot;
int          bombdamage;

//
// PIT_RadiusAttack
// "bombsource" is the creature
// that caused the explosion at "bombspot".
//
boolean PIT_RadiusAttack (mobj_t* thing)
{
    fixed_t      dx;
    fixed_t      dy;
    fixed_t      dist;

```

```

    if (!(thing->flags & MF_SHOOTABLE) )
        return true;

    // Boss spider and cyborg
    // take no damage from concussion.
    if (thing->type == MT_CYBORG
        || thing->type == MT_SPIDER)
        return true;

    dx = abs(thing->x - bombspot->x);
    dy = abs(thing->y - bombspot->y);

    dist = dx>dy ? dx : dy;
    dist = (dist - thing->radius) >> FRACBITS;

    if (dist < 0)
        dist = 0;

    if (dist >= bombdamage)
        return true;          // out of range

    if ( P_CheckSight (thing, bombspot) )
    {
        // must be in direct path
        P_DamageMobj (thing, bombspot, bombsource, bombdamage - dist);
    }

    return true;
}

//
// P_RadiusAttack
// Source is the creature that caused the explosion at spot.
//
void
P_RadiusAttack
( mobj_t*      spot,
  mobj_t*      source,
  int          damage )
{
    int         x;
    int         y;

    int         xl;
    int         xh;
    int         yl;
    int         yh;

    fixed_t     dist;

    dist = (damage+MAXRADIUS)<<FRACBITS;
    yh = (spot->y + dist - bmaporgy)>>MAPBLOCKSHIFT;
    yl = (spot->y - dist - bmaporgy)>>MAPBLOCKSHIFT;
    xh = (spot->x + dist - bmaporgx)>>MAPBLOCKSHIFT;
    xl = (spot->x - dist - bmaporgx)>>MAPBLOCKSHIFT;
    bombspot = spot;
    bombsource = source;
    bombdamage = damage;

    for (y=yl ; y<=yh ; y++)
        for (x=xl ; x<=xh ; x++)
            P_BlockThingsIterator (x, y, PIT_RadiusAttack );
}

```

```

//
// SECTOR HEIGHT CHANGING
// After modifying a sectors floor or ceiling height,
// call this routine to adjust the positions
// of all things that touch the sector.
//
// If anything doesn't fit anymore, true will be returned.
// If crunch is true, they will take damage
// as they are being crushed.
// If Crunch is false, you should set the sector height back
// the way it was and call P_ChangeSector again
// to undo the changes.
//
boolean                crushchange;
boolean                nofit;

//
// PIT_ChangeSector
//
boolean PIT_ChangeSector (mobj_t*      thing)
{
    mobj_t*            mo;

    if (P_ThingHeightClip (thing))
    {
        // keep checking
        return true;
    }

    // crunch bodies to giblets
    if (thing->health <= 0)
    {
        P_SetMobjState (thing, S_GIBS);

        thing->flags &= ~MF_SOLID;
        thing->height = 0;
        thing->radius = 0;

        // keep checking
        return true;
    }

    // crunch dropped items
    if (thing->flags & MF_DROPPED)
    {
        P_RemoveMobj (thing);

        // keep checking
        return true;
    }

    if (! (thing->flags & MF_SHOOTABLE) )
    {
        // assume it is bloody gibs or something
        return true;
    }

    nofit = true;

    if (crushchange && !(leveltime&3) )

```



```

{
    P_DamageMobj(thing,NULL,NULL,10);

    // spray blood in a random direction
    mo = P_SpawnMobj (thing->x,
                      thing->y,
                      thing->z + thing->height/2, MT_BLOOD);

    mo->momx = (P_Random() - P_Random ())<<12;
    mo->momy = (P_Random() - P_Random ())<<12;
}

// keep checking (crush other things)
return true;
}

//
// P_ChangeSector
//
boolean
P_ChangeSector
( sector_t*      sector,
  boolean        crunch )
{
    int          x;
    int          y;

    nofit = false;
    crushchange = crunch;

    // re-check heights for all things near the moving sector
    for (x=sector->blockbox[BOXLEFT] ; x<= sector->blockbox[BOXRIGHT] ; x++)
        for (y=sector->blockbox[BOXBOTTOM];y<= sector->blockbox[BOXTOP] ; y++)
            P_BlockThingsIterator (x, y, PIT_ChangeSector);

    return nofit;
}

```

## 9.12 p\_maputl.c

```

// Emacs style mode select    -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Movement/collision utility functions,

```

```

//      as used by function in p_map.c.
//      BLOCKMAP Iterator functions,
//      and some PIT_* functions to use for iteration.
//
//-----

static const char
rcsid[] = "$Id: p_maputl.c,v 1.5 1997/02/03 22:45:11 b1 Exp $";

#include <stdlib.h>

#include "m_bbox.h"

#include "doomdef.h"
#include "p_local.h"

// State.
#include "r_state.h"

//
// P_AproxDistance
// Gives an estimation of distance (not exact)
//

fixed_t
P_AproxDistance
( fixed_t      dx,
  fixed_t      dy )
{
    dx = abs(dx);
    dy = abs(dy);
    if (dx < dy)
        return dx+dy-(dx>>1);
    return dx+dy-(dy>>1);
}

//
// P_PointOnLineSide
// Returns 0 or 1
//
int
P_PointOnLineSide
( fixed_t      x,
  fixed_t      y,
  line_t*      line )
{
    fixed_t      dx;
    fixed_t      dy;
    fixed_t      left;
    fixed_t      right;

    if (!line->dx)
    {
        if (x <= line->v1->x)
            return line->dy > 0;

        return line->dy < 0;
    }
    if (!line->dy)
    {
        if (y <= line->v1->y)

```

```

        return line->dx < 0;

    return line->dx > 0;
}

dx = (x - line->v1->x);
dy = (y - line->v1->y);

left = FixedMul ( line->dy>>FRACBITS , dx );
right = FixedMul ( dy , line->dx>>FRACBITS );

if (right < left)
    return 0;                // front side
return 1;                    // back side
}

//
// P_BoxOnLineSide
// Considers the line to be infinite
// Returns side 0 or 1, -1 if box crosses the line.
//
int
P_BoxOnLineSide
( fixed_t*      tmbox,
  line_t*      ld )
{
    int          p1;
    int          p2;

    switch (ld->slopetype)
    {
        case ST_HORIZONTAL:
            p1 = tmbox[BOXTOP] > ld->v1->y;
            p2 = tmbox[BOXBOTTOM] > ld->v1->y;
            if (ld->dx < 0)
            {
                p1 ^= 1;
                p2 ^= 1;
            }
            break;

        case ST_VERTICAL:
            p1 = tmbox[BOXRIGHT] < ld->v1->x;
            p2 = tmbox[BOXLEFT] < ld->v1->x;
            if (ld->dy < 0)
            {
                p1 ^= 1;
                p2 ^= 1;
            }
            break;

        case ST_POSITIVE:
            p1 = P_PointOnLineSide (tmbox[BOXLEFT], tmbox[BOXTOP], ld);
            p2 = P_PointOnLineSide (tmbox[BOXRIGHT], tmbox[BOXBOTTOM], ld);
            break;

        case ST_NEGATIVE:
            p1 = P_PointOnLineSide (tmbox[BOXRIGHT], tmbox[BOXTOP], ld);
            p2 = P_PointOnLineSide (tmbox[BOXLEFT], tmbox[BOXBOTTOM], ld);
            break;
    }

    if (p1 == p2)

```

```

        return p1;
    return -1;
}

//
// P_PointOnDivlineSide
// Returns 0 or 1.
//
int
P_PointOnDivlineSide
( fixed_t      x,
  fixed_t      y,
  divline_t*   line )
{
    fixed_t     dx;
    fixed_t     dy;
    fixed_t     left;
    fixed_t     right;

    if (!line->dx)
    {
        if (x <= line->x)
            return line->dy > 0;

        return line->dy < 0;
    }
    if (!line->dy)
    {
        if (y <= line->y)
            return line->dx < 0;

        return line->dx > 0;
    }

    dx = (x - line->x);
    dy = (y - line->y);

    // try to quickly decide by looking at sign bits
    if ( (line->dy ^ line->dx ^ dx ^ dy)&0x80000000 )
    {
        if ( (line->dy ^ dx) & 0x80000000 )
            return 1;                // (left is negative)
        return 0;
    }

    left = FixedMul ( line->dy>>8, dx>>8 );
    right = FixedMul ( dy>>8 , line->dx>>8 );

    if (right < left)
        return 0;                    // front side
    return 1;                        // back side
}

//
// P_MakeDivline
//
void
P_MakeDivline
( line_t*      li,
  divline_t*    dl )
{
    dl->x = li->v1->x;

```

```

    dl->y = li->v1->y;
    dl->dx = li->dx;
    dl->dy = li->dy;
}

//
// P_InterceptVector
// Returns the fractional intercept point
// along the first divline.
// This is only called by the addthings
// and addlines traversers.
//
fixed_t
P_InterceptVector
( divline_t*      v2,
  divline_t*      v1 )
{
#if 1
    fixed_t      frac;
    fixed_t      num;
    fixed_t      den;

    den = FixedMul (v1->dy>>8,v2->dx) - FixedMul(v1->dx>>8,v2->dy);

    if (den == 0)
        return 0;
    //      I_Error ("P_InterceptVector: parallel");

    num =
        FixedMul ( (v1->x - v2->x)>>8 ,v1->dy )
        +FixedMul ( (v2->y - v1->y)>>8, v1->dx );

    frac = FixedDiv (num , den);

    return frac;
#else      // UNUSED, float debug.
    float      frac;
    float      num;
    float      den;
    float      v1x;
    float      v1y;
    float      v1dx;
    float      v1dy;
    float      v2x;
    float      v2y;
    float      v2dx;
    float      v2dy;

    v1x = (float)v1->x/FRACUNIT;
    v1y = (float)v1->y/FRACUNIT;
    v1dx = (float)v1->dx/FRACUNIT;
    v1dy = (float)v1->dy/FRACUNIT;
    v2x = (float)v2->x/FRACUNIT;
    v2y = (float)v2->y/FRACUNIT;
    v2dx = (float)v2->dx/FRACUNIT;
    v2dy = (float)v2->dy/FRACUNIT;

    den = v1dy*v2dx - v1dx*v2dy;

    if (den == 0)
        return 0;      // parallel

    num = (v1x - v2x)*v1dy + (v2y - v1y)*v1dx;

```

```

    frac = num / den;

    return frac*FRACUNIT;
#endif
}

//
// P_LineOpening
// Sets opentop and openbottom to the window
// through a two sided line.
// OPTIMIZE: keep this precalculated
//
fixed_t opentop;
fixed_t openbottom;
fixed_t openrange;
fixed_t      lowfloor;

void P_LineOpening (line_t* linedef)
{
    sector_t*      front;
    sector_t*      back;

    if (linedef->sidenum[1] == -1)
    {
        // single sided line
        openrange = 0;
        return;
    }

    front = linedef->frontsector;
    back = linedef->backsector;

    if (front->ceilingheight < back->ceilingheight)
        opentop = front->ceilingheight;
    else
        opentop = back->ceilingheight;

    if (front->floorheight > back->floorheight)
    {
        openbottom = front->floorheight;
        lowfloor = back->floorheight;
    }
    else
    {
        openbottom = back->floorheight;
        lowfloor = front->floorheight;
    }

    openrange = opentop - openbottom;
}

//
// THING POSITION SETTING
//

//
// P_UnsetThingPosition
// Unlinks a thing from block map and sectors.
// On each position change, BLOCKMAP and other
// lookups maintaining lists of things inside
// these structures need to be updated.

```

```

//
void P_UnsetThingPosition (mobj_t* thing)
{
    int            blockx;
    int            blocky;

    if ( ! (thing->flags & MF_NOSECTOR) )
    {
        // inert things don't need to be in blockmap?
        // unlink from subsector
        if (thing->snext)
            thing->snext->sprev = thing->sprev;

        if (thing->sprev)
            thing->sprev->snext = thing->snext;
        else
            thing->subsector->sector->thinglist = thing->snext;
    }

    if ( ! (thing->flags & MF_NOBLOCKMAP) )
    {
        // inert things don't need to be in blockmap
        // unlink from block map
        if (thing->bnext)
            thing->bnext->bprev = thing->bprev;

        if (thing->bprev)
            thing->bprev->bnext = thing->bnext;
        else
        {
            blockx = (thing->x - bmaporgx)>>MAPBLOCKSHIFT;
            blocky = (thing->y - bmaporgy)>>MAPBLOCKSHIFT;

            if (blockx>=0 && blockx < bmapwidth
                && blocky>=0 && blocky < bmapheight)
            {
                blocklinks[blocky*bmapwidth+blockx] = thing->bnext;
            }
        }
    }
}

```

```

//
// P_SetThingPosition
// Links a thing into both a block and a subsector
// based on it's x y.
// Sets thing->subsector properly
//
void
P_SetThingPosition (mobj_t* thing)
{
    subsector_t*    ss;
    sector_t*       sec;
    int             blockx;
    int             blocky;
    mobj_t**        link;

    // link into subsector
    ss = R_PointInSubsector (thing->x,thing->y);
    thing->subsector = ss;

    if ( ! (thing->flags & MF_NOSECTOR) )
    {

```

```

    // invisible things don't go into the sector links
    sec = ss->sector;

    thing->sprev = NULL;
    thing->snext = sec->thinglist;

    if (sec->thinglist)
        sec->thinglist->sprev = thing;

    sec->thinglist = thing;
}

// link into blockmap
if ( ! (thing->flags & MF_NOBLOCKMAP) )
{
    // inert things don't need to be in blockmap
    blockx = (thing->x - bmaporgx)>>MAPBLOCKSHIFT;
    blocky = (thing->y - bmaporgy)>>MAPBLOCKSHIFT;

    if (blockx>=0
        && blockx < bmapwidth
        && blocky>=0
        && blocky < bmapheight)
    {
        link = &blocklinks[blocky*bmapwidth+blockx];
        thing->bprev = NULL;
        thing->bnext = *link;
        if (*link)
            (*link)->bprev = thing;

        *link = thing;
    }
    else
    {
        // thing is off the map
        thing->bnext = thing->bprev = NULL;
    }
}

}

//
// BLOCK MAP ITERATORS
// For each line/thing in the given mapblock,
// call the passed PIT_* function.
// If the function returns false,
// exit with false without checking anything else.
//

//
// P_BlockLinesIterator
// The validcount flags are used to avoid checking lines
// that are marked in multiple mapblocks,
// so increment validcount before the first call
// to P_BlockLinesIterator, then make one or more calls
// to it.
//
boolean
P_BlockLinesIterator
( int          x,
  int          y,
  boolean(*func)(line_t*) )

```



```

{
    int                offset;
    short*             list;
    line_t*            ld;

    if (x<0
        || y<0
        || x>=bmapwidth
        || y>=bmapheight)
    {
        return true;
    }

    offset = y*bmapwidth+x;

    offset = *(blockmap+offset);

    for ( list = blockmaplump+offset ; *list != -1 ; list++)
    {
        ld = &lines[*list];

        if (ld->validcount == validcount)
            continue;          // line has already been checked

        ld->validcount = validcount;

        if ( !func(ld) )
            return false;
    }
    return true;              // everything was checked
}

//
// P_BlockThingsIterator
//
boolean
P_BlockThingsIterator
( int                x,
  int                y,
  boolean(*func)(mobj_t*) )
{
    mobj_t*          mobj;

    if ( x<0
        || y<0
        || x>=bmapwidth
        || y>=bmapheight)
    {
        return true;
    }

    for (mobj = blocklinks[y*bmapwidth+x] ;
        mobj ;
        mobj = mobj->bnext)
    {
        if (!func( mobj ) )
            return false;
    }
    return true;
}

```

```

//
// INTERCEPT ROUTINES
//
intercept_t      intercepts[MAXINTERCEPTS];
intercept_t*     intercept_p;

divline_t        trace;
boolean          earlyout;
int              ptflags;

//
// PIT_AddLineIntercepts.
// Looks for lines in the given block
// that intercept the given trace
// to add to the intercepts list.
//
// A line is crossed if its endpoints
// are on opposite sides of the trace.
// Returns true if earlyout and a solid line hit.
//
boolean
PIT_AddLineIntercepts (line_t* ld)
{
    int          s1;
    int          s2;
    fixed_t      frac;
    divline_t     dl;

    // avoid precision problems with two routines
    if ( trace.dx > FRACUNIT*16
        || trace.dy > FRACUNIT*16
        || trace.dx < -FRACUNIT*16
        || trace.dy < -FRACUNIT*16)
    {
        s1 = P_PointOnDivlineSide (ld->v1->x, ld->v1->y, &trace);
        s2 = P_PointOnDivlineSide (ld->v2->x, ld->v2->y, &trace);
    }
    else
    {
        s1 = P_PointOnLineSide (trace.x, trace.y, ld);
        s2 = P_PointOnLineSide (trace.x+trace.dx, trace.y+trace.dy, ld);
    }

    if (s1 == s2)
        return true;          // line isn't crossed

    // hit the line
    P_MakeDivline (ld, &dl);
    frac = P_InterceptVector (&trace, &dl);

    if (frac < 0)
        return true;          // behind source

    // try to early out the check
    if (earlyout
        && frac < FRACUNIT
        && !ld->backsector)
    {
        return false;          // stop checking
    }

    intercept_p->frac = frac;
    intercept_p->isaline = true;
    intercept_p->d.line = ld;
}

```

```

    intercept_p++;

    return true;          // continue
}

//
// PIT_AddThingIntercepts
//
boolean PIT_AddThingIntercepts (mobj_t* thing)
{
    fixed_t          x1;
    fixed_t          y1;
    fixed_t          x2;
    fixed_t          y2;

    int              s1;
    int              s2;

    boolean          tracepositive;

    divline_t        dl;

    fixed_t          frac;

    tracepositive = (trace.dx ^ trace.dy)>0;

    // check a corner to corner crossection for hit
    if (tracepositive)
    {
        x1 = thing->x - thing->radius;
        y1 = thing->y + thing->radius;

        x2 = thing->x + thing->radius;
        y2 = thing->y - thing->radius;
    }
    else
    {
        x1 = thing->x - thing->radius;
        y1 = thing->y - thing->radius;

        x2 = thing->x + thing->radius;
        y2 = thing->y + thing->radius;
    }

    s1 = P_PointOnDivlineSide (x1, y1, &trace);
    s2 = P_PointOnDivlineSide (x2, y2, &trace);

    if (s1 == s2)
        return true;          // line isn't crossed

    dl.x = x1;
    dl.y = y1;
    dl.dx = x2-x1;
    dl.dy = y2-y1;

    frac = P_InterceptVector (&trace, &dl);

    if (frac < 0)
        return true;          // behind source

    intercept_p->frac = frac;
    intercept_p->isaline = false;
    intercept_p->d.thing = thing;

```

```

    intercept_p++;

    return true;                // keep going
}

//
// P_TraverseIntercepts
// Returns true if the traverser function returns true
// for all lines.
//
boolean
P_TraverseIntercepts
( traverser_t      func,
  fixed_t          maxfrac )
{
    int             count;
    fixed_t         dist;
    intercept_t*    scan;
    intercept_t*    in;

    count = intercept_p - intercepts;

    in = 0;                // shut up compiler warning

    while (count--)
    {
        dist = MAXINT;
        for (scan = intercepts ; scan < intercept_p ; scan++)
        {
            if (scan->frac < dist)
            {
                dist = scan->frac;
                in = scan;
            }
        }

        if (dist > maxfrac)
            return true;    // checked everything in range
    }

#if 0 // UNUSED
    {
        // don't check these yet, there may be others inserted
        in = scan = intercepts;
        for ( scan = intercepts ; scan < intercept_p ; scan++)
            if (scan->frac > maxfrac)
                *in++ = *scan;
        intercept_p = in;
        return false;
    }
#endif

    if ( !func (in) )
        return false;      // don't bother going farther

    in->frac = MAXINT;
}

return true;                // everything was traversed
}

//

```

```

// P_PathTraverse
// Traces a line from x1,y1 to x2,y2,
// calling the traverser function for each.
// Returns true if the traverser function returns true
// for all lines.
//
boolean
P_PathTraverse
( fixed_t          x1,
  fixed_t          y1,
  fixed_t          x2,
  fixed_t          y2,
  int              flags,
  boolean (*trav) (intercept_t *))
{
    fixed_t        xt1;
    fixed_t        yt1;
    fixed_t        xt2;
    fixed_t        yt2;

    fixed_t        xstep;
    fixed_t        ystep;

    fixed_t        partial;

    fixed_t        xintercept;
    fixed_t        yintercept;

    int            mapx;
    int            mapy;

    int            mapxstep;
    int            mapystep;

    int            count;

    earlyout = flags & PT_EARLYOUT;

    validcount++;
    intercept_p = intercepts;

    if ( ((x1-bmaporgx)&(MAPBLOCKSIZE-1)) == 0 )
        x1 += FRACUNIT;          // don't side exactly on a line

    if ( ((y1-bmaporgy)&(MAPBLOCKSIZE-1)) == 0 )
        y1 += FRACUNIT;          // don't side exactly on a line

    trace.x = x1;
    trace.y = y1;
    trace.dx = x2 - x1;
    trace.dy = y2 - y1;

    x1 -= bmaporgx;
    y1 -= bmaporgy;
    xt1 = x1>>MAPBLOCKSHIFT;
    yt1 = y1>>MAPBLOCKSHIFT;

    x2 -= bmaporgx;
    y2 -= bmaporgy;
    xt2 = x2>>MAPBLOCKSHIFT;
    yt2 = y2>>MAPBLOCKSHIFT;

    if (xt2 > xt1)
    {
        mapxstep = 1;

```

```

    partial = FRACUNIT - ((x1>>MAPBTOFRAC)&(FRACUNIT-1));
    ystep = FixedDiv (y2-y1,abs(x2-x1));
}
else if (xt2 < xt1)
{
    mapxstep = -1;
    partial = (x1>>MAPBTOFRAC)&(FRACUNIT-1);
    ystep = FixedDiv (y2-y1,abs(x2-x1));
}
else
{
    mapxstep = 0;
    partial = FRACUNIT;
    ystep = 256*FRACUNIT;
}

yintercept = (y1>>MAPBTOFRAC) + FixedMul (partial, ystep);

if (yt2 > yt1)
{
    mapystep = 1;
    partial = FRACUNIT - ((y1>>MAPBTOFRAC)&(FRACUNIT-1));
    xstep = FixedDiv (x2-x1,abs(y2-y1));
}
else if (yt2 < yt1)
{
    mapystep = -1;
    partial = (y1>>MAPBTOFRAC)&(FRACUNIT-1);
    xstep = FixedDiv (x2-x1,abs(y2-y1));
}
else
{
    mapystep = 0;
    partial = FRACUNIT;
    xstep = 256*FRACUNIT;
}
xintercept = (x1>>MAPBTOFRAC) + FixedMul (partial, xstep);

// Step through map blocks.
// Count is present to prevent a round off error
// from skipping the break.
mapx = xt1;
mapy = yt1;

for (count = 0 ; count < 64 ; count++)
{
    if (flags & PT_ADDLINES)
    {
        if (!P_BlockLinesIterator (mapx, mapy,PIT_AddLineIntercepts))
            return false;          // early out
    }

    if (flags & PT_ADDTHINGS)
    {
        if (!P_BlockThingsIterator (mapx, mapy,PIT_AddThingIntercepts))
            return false;          // early out
    }

    if (mapx == xt2
        && mapy == yt2)
    {
        break;
    }
}

```

```

    if ( (yintercept >> FRACBITS) == mapy)
    {
        yintercept += ystep;
        mapx += mapxstep;
    }
    else if ( (xintercept >> FRACBITS) == mapx)
    {
        xintercept += xstep;
        mapy += mapystep;
    }

}
// go through the sorted list
return P_TraverseIntercepts ( trav, FRACUNIT );
}

```

### 9.13 p\_mobj.c

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Moving object handling. Spawn functions.
//
//-----

static const char
rcsid[] = "$Id: p_mobj.c,v 1.5 1997/02/03 22:45:12 b1 Exp $";

#include "i_system.h"
#include "z_zone.h"
#include "m_random.h"

#include "doomdef.h"
#include "p_local.h"
#include "sounds.h"

#include "st_stuff.h"
#include "hu_stuff.h"

#include "s_sound.h"

#include "doomstat.h"

void G_PlayerReborn (int player);

```

```

void P_SpawnMapThing (mapthing_t*      mthing);

//
// P_SetMobjState
// Returns true if the mobj is still present.
//
int test;

boolean
P_SetMobjState
( mobj_t*      mobj,
  statenum_t    state )
{
    state_t*      st;

    do
    {
        if (state == S_NULL)
        {
            mobj->state = (state_t *) S_NULL;
            P_RemoveMobj (mobj);
            return false;
        }

        st = &states[state];
        mobj->state = st;
        mobj->tics = st->tics;
        mobj->sprite = st->sprite;
        mobj->frame = st->frame;

        // Modified handling.
        // Call action functions when the state is set
        if (st->action.acp1)
            st->action.acp1(mobj);

        state = st->nextstate;
    } while (!mobj->tics);

    return true;
}

//
// P_ExplodeMissile
//
void P_ExplodeMissile (mobj_t* mo)
{
    mo->momx = mo->momy = mo->momz = 0;

    P_SetMobjState (mo, mobjinfo[mo->type].deathstate);

    mo->tics -= P_Random()&3;

    if (mo->tics < 1)
        mo->tics = 1;

    mo->flags &= ~MF_MISSILE;

    if (mo->info->deathsound)
        S_StartSound (mo, mo->info->deathsound);
}

//

```



```

// P_XYMovement
//
#define STOPSPEED          0x1000
#define FRICTION           0xe800

void P_XYMovement (mobj_t* mo)
{
    fixed_t      ptryx;
    fixed_t      ptryy;
    player_t*     player;
    fixed_t      xmove;
    fixed_t      ymove;

    if (!mo->momx && !mo->momy)
    {
        if (mo->flags & MF_SKULLFLY)
        {
            // the skull slammed into something
            mo->flags &= ~MF_SKULLFLY;
            mo->momx = mo->momy = mo->momz = 0;

            P_SetMobjState (mo, mo->info->spawnstate);
        }
        return;
    }

    player = mo->player;

    if (mo->momx > MAXMOVE)
        mo->momx = MAXMOVE;
    else if (mo->momx < -MAXMOVE)
        mo->momx = -MAXMOVE;

    if (mo->momy > MAXMOVE)
        mo->momy = MAXMOVE;
    else if (mo->momy < -MAXMOVE)
        mo->momy = -MAXMOVE;

    xmove = mo->momx;
    ymove = mo->momy;

    do
    {
        if (xmove > MAXMOVE/2 || ymove > MAXMOVE/2)
        {
            ptryx = mo->x + xmove/2;
            ptryy = mo->y + ymove/2;
            xmove >>= 1;
            ymove >>= 1;
        }
        else
        {
            ptryx = mo->x + xmove;
            ptryy = mo->y + ymove;
            xmove = ymove = 0;
        }

        if (!P_TryMove (mo, ptryx, ptryy))
        {
            // blocked move
            if (mo->player)
            {
                // try to slide along it
                P_SlideMove (mo);
            }
            else if (mo->flags & MF_MISSILE)

```

```

    {
        // explode a missile
        if (ceilingline &&
            ceilingline->backsector &&
            ceilingline->backsector->ceilingpic == skyflatnum)
        {
            // Hack to prevent missiles exploding
            // against the sky.
            // Does not handle sky floors.
            P_RemoveMobj (mo);
            return;
        }
        P_ExplodeMissile (mo);
    }
    else
        mo->momx = mo->momy = 0;
}
} while (xmove || ymove);

// slow down
if (player && player->cheats & CF_NOMOMENTUM)
{
    // debug option for no sliding at all
    mo->momx = mo->momy = 0;
    return;
}

if (mo->flags & (MF_MISSILE | MF_SKULLFLY) )
    return;          // no friction for missiles ever

if (mo->z > mo->floorz)
    return;          // no friction when airborne

if (mo->flags & MF_CORPSE)
{
    // do not stop sliding
    // if halfway off a step with some momentum
    if (mo->momx > FRACUNIT/4
        || mo->momx < -FRACUNIT/4
        || mo->momy > FRACUNIT/4
        || mo->momy < -FRACUNIT/4)
    {
        if (mo->floorz != mo->subsector->sector->floorheight)
            return;
    }
}

if (mo->momx > -STOPSPEED
    && mo->momx < STOPSPEED
    && mo->momy > -STOPSPEED
    && mo->momy < STOPSPEED
    && (!player
        || (player->cmd.forwardmove == 0
            && player->cmd.sidemove == 0) ) )
{
    // if in a walking frame, stop moving
    if ( player&&(unsigned)((player->mo->state - states)- S_PLAY_RUN1) < 4)
        P_SetMobjState (player->mo, S_PLAY);

    mo->momx = 0;
    mo->momy = 0;
}
else
{
    mo->momx = FixedMul (mo->momx, FRICTION);

```

```

        mo->momy = FixedMul (mo->momy, FRICTION);
    }
}

//
// P_ZMovement
//
void P_ZMovement (mobj_t* mo)
{
    fixed_t      dist;
    fixed_t      delta;

    // check for smooth step up
    if (mo->player && mo->z < mo->floorz)
    {
        mo->player->viewheight -= mo->floorz-mo->z;

        mo->player->deltaviewheight
            = (VIEWHEIGHT - mo->player->viewheight)>>3;
    }

    // adjust height
    mo->z += mo->momz;

    if ( mo->flags & MF_FLOAT
        && mo->target)
    {
        // float down towards target if too close
        if ( !(mo->flags & MF_SKULLFLY)
            && !(mo->flags & MF_INFLOAT) )
        {
            dist = P_AproxDistance (mo->x - mo->target->x,
                                    mo->y - mo->target->y);

            delta =(mo->target->z + (mo->height>>1)) - mo->z;

            if (delta<0 && dist < -(delta*3) )
                mo->z -= FLOATSPEED;
            else if (delta>0 && dist < (delta*3) )
                mo->z += FLOATSPEED;
        }
    }

    // clip movement
    if (mo->z <= mo->floorz)
    {
        // hit the floor

        // Note (id):
        // somebody left this after the setting momz to 0,
        // kinda useless there.
        if (mo->flags & MF_SKULLFLY)
        {
            // the skull slammed into something
            mo->momz = -mo->momz;
        }

        if (mo->momz < 0)
        {
            if (mo->player
                && mo->momz < -GRAVITY*8)
            {
                // Squat down.
                // Decrease viewheight for a moment
            }
        }
    }
}

```

```

        // after hitting the ground (hard),
        // and utter appropriate sound.
        mo->player->deltaviewheight = mo->momz>>3;
        S_StartSound (mo, sfx_oof);
    }
    mo->momz = 0;
}
mo->z = mo->floorz;

if ( (mo->flags & MF_MISSILE)
    && !(mo->flags & MF_NOCLIP) )
{
    P_ExplodeMissile (mo);
    return;
}
}
else if (! (mo->flags & MF_NOGRAVITY) )
{
    if (mo->momz == 0)
        mo->momz = -GRAVITY*2;
    else
        mo->momz -= GRAVITY;
}

if (mo->z + mo->height > mo->ceilingz)
{
    // hit the ceiling
    if (mo->momz > 0)
        mo->momz = 0;
    {
        mo->z = mo->ceilingz - mo->height;
    }

    if (mo->flags & MF_SKULLFLY)
    {
        // the skull slammed into something
        mo->momz = -mo->momz;
    }

    if ( (mo->flags & MF_MISSILE)
        && !(mo->flags & MF_NOCLIP) )
    {
        P_ExplodeMissile (mo);
        return;
    }
}
}
}

```

```

//
// P_NightmareRespawn
//
void
P_NightmareRespawn (mobj_t* mobj)
{
    fixed_t          x;
    fixed_t          y;
    fixed_t          z;
    subsector_t*     ss;
    mobj_t*          mo;
    mapthing_t*      mthing;

    x = mobj->spawnpoint.x << FRACBITS;
    y = mobj->spawnpoint.y << FRACBITS;

```

```

// something is occupying it's position?
if (!P_CheckPosition (mobj, x, y) )
    return;          // no respawn

// spawn a teleport fog at old spot
// because of removal of the body?
mo = P_SpawnMobj (mobj->x,
                 mobj->y,
                 mobj->subsector->sector->floorheight , MT_TFOG);
// initiate teleport sound
S_StartSound (mo, sfx_telept);

// spawn a teleport fog at the new spot
ss = R_PointInSubsector (x,y);

mo = P_SpawnMobj (x, y, ss->sector->floorheight , MT_TFOG);

S_StartSound (mo, sfx_telept);

// spawn the new monster
mthing = &mobj->spawnpoint;

// spawn it
if (mobj->info->flags & MF_SPAWNCEILING)
    z = ONCEILINGZ;
else
    z = ONFLOORZ;

// inherit attributes from deceased one
mo = P_SpawnMobj (x,y,z, mobj->type);
mo->spawnpoint = mobj->spawnpoint;
mo->angle = ANG45 * (mthing->angle/45);

if (mthing->options & MTF_AMBUSH)
    mo->flags |= MF_AMBUSH;

mo->reactiontime = 18;

// remove the old monster,
P_RemoveMobj (mobj);
}

//
// P_MobjThinker
//
void P_MobjThinker (mobj_t* mobj)
{
    // momentum movement
    if (mobj->momx
        || mobj->momy
        || (mobj->flags&MF_SKULLFLY) )
    {
        P_XYMovement (mobj);

        // FIXME: decent NOP/NULL/Nil function pointer please.
        if (mobj->thinker.function.acv == (actionf_v) (-1))
            return;          // mobj was removed
    }
    if ( (mobj->z != mobj->floorz)
        || mobj->momz )
    {
        P_ZMovement (mobj);

        // FIXME: decent NOP/NULL/Nil function pointer please.
    }
}

```

```

        if (mobj->thinker.function.acv == (actionf_v) (-1))
            return;                // mobj was removed
    }

    // cycle through states,
    // calling action functions at transitions
    if (mobj->tics != -1)
    {
        mobj->tics--;

        // you can cycle through multiple states in a tic
        if (!mobj->tics)
            if (!P_SetMobjState (mobj, mobj->state->nextstate) )
                return;            // freed itself
    }
    else
    {
        // check for nightmare respawn
        if (! (mobj->flags & MF_COUNTKILL) )
            return;

        if (!respawnmonsters)
            return;

        mobj->movecount++;

        if (mobj->movecount < 12*35)
            return;

        if ( leveltime&31 )
            return;

        if (P_Random () > 4)
            return;

        P_NightmareRespawn (mobj);
    }
}

//
// P_SpawnMobj
//
mobj_t*
P_SpawnMobj
( fixed_t      x,
  fixed_t      y,
  fixed_t      z,
  mobjtype_t    type )
{
    mobj_t*      mobj;
    state_t*     st;
    mobjinfo_t*  info;

    mobj = Z_Malloc (sizeof(*mobj), PU_LEVEL, NULL);
    memset (mobj, 0, sizeof (*mobj));
    info = &mobjinfo[type];

    mobj->type = type;
    mobj->info = info;
    mobj->x = x;
    mobj->y = y;
    mobj->radius = info->radius;

```

```

mobj->height = info->height;
mobj->flags = info->flags;
mobj->health = info->spawnhealth;

if (gameskill != sk_nightmare)
    mobj->reactiontime = info->reactiontime;

mobj->lastlook = P_Random () % MAXPLAYERS;
// do not set the state with P_SetMobjState,
// because action routines can not be called yet
st = &states[info->spawnstate];

mobj->state = st;
mobj->tics = st->tics;
mobj->sprite = st->sprite;
mobj->frame = st->frame;

// set subsector and/or block links
P_SetThingPosition (mobj);

mobj->floorz = mobj->subsector->sector->floorheight;
mobj->ceilingz = mobj->subsector->sector->ceilingheight;

if (z == ONFLOORZ)
    mobj->z = mobj->floorz;
else if (z == ONCEILINGZ)
    mobj->z = mobj->ceilingz - mobj->info->height;
else
    mobj->z = z;

mobj->thinker.function.acp1 = (actionf_p1)P_MobjThinker;

P_AddThinker (&mobj->thinker);

return mobj;
}

//
// P_RemoveMobj
//
mapthing_t      itemrespawnque[ITEMQUESIZE];
int              itemrespawntime[ITEMQUESIZE];
int              iquehead;
int              iquetail;

void P_RemoveMobj (mobj_t* mobj)
{
    if ((mobj->flags & MF_SPECIAL)
        && !(mobj->flags & MF_DROPPED)
        && (mobj->type != MT_INV)
        && (mobj->type != MT_INS))
    {
        itemrespawnque[iquehead] = mobj->spawnpoint;
        itemrespawntime[iquehead] = leveltime;
        iquehead = (iquehead+1)&(ITEMQUESIZE-1);

        // lose one off the end?
        if (iquehead == iquetail)
            iquetail = (iquetail+1)&(ITEMQUESIZE-1);
    }

    // unlink from sector and block lists
    P_UnsetThingPosition (mobj);

```

```

// stop any playing sound
S_StopSound (mobj);

// free block
P_RemoveThinker ((thinker_t*)mobj);
}

//
// P_RespawnSpecials
//
void P_RespawnSpecials (void)
{
    fixed_t          x;
    fixed_t          y;
    fixed_t          z;

    subsector_t*     ss;
    mobj_t*          mo;
    mapthing_t*      mthing;

    int              i;

    // only respawn items in deathmatch
    if (deathmatch != 2)
        return;

    // nothing left to respawn?
    if (iquehead == iquetail)
        return;

    // wait at least 30 seconds
    if (leveltime - itemrespawntime[iquetail] < 30*35)
        return;

    mthing = &itemrespawnque[iquetail];

    x = mthing->x << FRACBITS;
    y = mthing->y << FRACBITS;

    // spawn a teleport fog at the new spot
    ss = R_PointInSubsector (x,y);
    mo = P_SpawnMobj (x, y, ss->sector->floorheight , MT_IFOG);
    S_StartSound (mo, sfx_itmbk);

    // find which type to spawn
    for (i=0 ; i< NUMMOBJTYPES ; i++)
    {
        if (mthing->type == mobjinfo[i].doomednum)
            break;
    }

    // spawn it
    if (mobjinfo[i].flags & MF_SPAWNCEILING)
        z = ONCEILINGZ;
    else
        z = ONFLOORZ;

    mo = P_SpawnMobj (x,y,z, i);
    mo->spawnpoint = *mthing;
    mo->angle = ANG45 * (mthing->angle/45);

```



```

    // pull it from the que
    iquetail = (iquetail+1)&(ITEMQESIZE-1);
}

//
// P_SpawnPlayer
// Called when a player is spawned on the level.
// Most of the player structure stays unchanged
// between levels.
//
void P_SpawnPlayer (mapthing_t* mthing)
{
    player_t*      p;
    fixed_t        x;
    fixed_t        y;
    fixed_t        z;

    mobj_t*        mobj;

    int             i;

    // not playing?
    if (!playeringame[mthing->type-1])
        return;

    p = &players[mthing->type-1];

    if (p->playerstate == PST_REBORN)
        G_PlayerReborn (mthing->type-1);

    x              = mthing->x << FRACBITS;
    y              = mthing->y << FRACBITS;
    z              = ONFLOORZ;
    mobj           = P_SpawnMobj (x,y,z, MT_PLAYER);

    // set color translations for player sprites
    if (mthing->type > 1)
        mobj->flags |= (mthing->type-1)<<MF_TRANSSHIFT;

    mobj->angle     = ANG45 * (mthing->angle/45);
    mobj->player = p;
    mobj->health = p->health;

    p->mo = mobj;
    p->playerstate = PST_LIVE;
    p->refire = 0;
    p->message = NULL;
    p->damagecount = 0;
    p->bonuscount = 0;
    p->extralight = 0;
    p->fixedcolormap = 0;
    p->viewheight = VIEWHEIGHT;

    // setup gun psprite
    P_SetupPsprites (p);

    // give all cards in death match mode
    if (deathmatch)
        for (i=0 ; i<NUMCARDS ; i++)
            p->cards[i] = true;

    if (mthing->type-1 == consoleplayer)

```

```

{
    // wake up the status bar
    ST_Start ();
    // wake up the heads up text
    HU_Start ();
}

}

//
// P_SpawnMapThing
// The fields of the mapthing should
// already be in host byte order.
//
void P_SpawnMapThing (mapthing_t* mthing)
{
    int                i;
    int                bit;
    mobj_t*            mobj;
    fixed_t            x;
    fixed_t            y;
    fixed_t            z;

    // count deathmatch start positions
    if (mthing->type == 11)
    {
        if (deathmatch_p < &deathmatchstarts[10])
        {
            memcpy (deathmatch_p, mthing, sizeof(*mthing));
            deathmatch_p++;
        }
        return;
    }

    // check for players specially
    if (mthing->type <= 4)
    {
        // save spots for respawning in network games
        playerstarts[mthing->type-1] = *mthing;
        if (!deathmatch)
            P_SpawnPlayer (mthing);

        return;
    }

    // check for appropriate skill level
    if (!netgame && (mthing->options & 16) )
        return;

    if (gameskill == sk_baby)
        bit = 1;
    else if (gameskill == sk_nightmare)
        bit = 4;
    else
        bit = 1<<(gameskill-1);

    if (!(mthing->options & bit) )
        return;

    // find which type to spawn
    for (i=0 ; i< NUMMOBJTYPES ; i++)
        if (mthing->type == mobjinfo[i].doomednum)
            break;

    if (i==NUMMOBJTYPES)

```

```

        I_Error ("P_SpawnMapThing: Unknown type %i at (%i, %i)",
                mthing->type,
                mthing->x, mthing->y);

// don't spawn keycards and players in deathmatch
if (deathmatch && mobjinfo[i].flags & MF_NOTDMATCH)
    return;

// don't spawn any monsters if -nomonsters
if (nomonsters
    && ( i == MT_SKULL
        || (mobjinfo[i].flags & MF_COUNTKILL)) )
{
    return;
}

// spawn it
x = mthing->x << FRACBITS;
y = mthing->y << FRACBITS;

if (mobjinfo[i].flags & MF_SPAWNCEILING)
    z = ONCEILINGZ;
else
    z = ONFLOORZ;

mobj = P_SpawnMobj (x,y,z, i);
mobj->spawnpoint = *mthing;

if (mobj->tics > 0)
    mobj->tics = 1 + (P_Random () % mobj->tics);
if (mobj->flags & MF_COUNTKILL)
    totalkills++;
if (mobj->flags & MF_COUNTITEM)
    totalitems++;

mobj->angle = ANG45 * (mthing->angle/45);
if (mthing->options & MTF_AMBUSH)
    mobj->flags |= MF_AMBUSH;
}

//
// GAME SPAWN FUNCTIONS
//

//
// P_SpawnPuff
//
extern fixed_t attackrange;

void
P_SpawnPuff
( fixed_t      x,
  fixed_t      y,
  fixed_t      z )
{
    mobj_t*     th;

    z += ((P_Random()-P_Random())<<10);

    th = P_SpawnMobj (x,y,z, MT_PUFF);
    th->momz = FRACUNIT;
    th->tics -= P_Random()&3;
}

```

```

    if (th->tics < 1)
        th->tics = 1;

    // don't make punches spark on the wall
    if (attackrange == MELEERANGE)
        P_SetMobjState (th, S_PUFF3);
}

//
// P_SpawnBlood
//
void
P_SpawnBlood
( fixed_t      x,
  fixed_t      y,
  fixed_t      z,
  int          damage )
{
    mobj_t*     th;

    z += ((P_Random()-P_Random())<<10);
    th = P_SpawnMobj (x,y,z, MT_BLOOD);
    th->momz = FRACUNIT*2;
    th->tics -= P_Random()&3;

    if (th->tics < 1)
        th->tics = 1;

    if (damage <= 12 && damage >= 9)
        P_SetMobjState (th,S_BLOOD2);
    else if (damage < 9)
        P_SetMobjState (th,S_BLOOD3);
}

//
// P_CheckMissileSpawn
// Moves the missile forward a bit
// and possibly explodes it right there.
//
void P_CheckMissileSpawn (mobj_t* th)
{
    th->tics -= P_Random()&3;
    if (th->tics < 1)
        th->tics = 1;

    // move a little forward so an angle can
    // be computed if it immediately explodes
    th->x += (th->momx>>1);
    th->y += (th->momy>>1);
    th->z += (th->momz>>1);

    if (!P_TryMove (th, th->x, th->y))
        P_ExplodeMissile (th);
}

//
// P_SpawnMissile
//
mobj_t*

```

```

P_SpawnMissile
( mobj_t*      source,
  mobj_t*      dest,
  mobjtype_t    type )
{
    mobj_t*      th;
    angle_t      an;
    int          dist;

    th = P_SpawnMobj (source->x,
                     source->y,
                     source->z + 4*8*FRACUNIT, type);

    if (th->info->seesound)
        S_StartSound (th, th->info->seesound);

    th->target = source;          // where it came from
    an = R_PointToAngle2 (source->x, source->y, dest->x, dest->y);

    // fuzzy player
    if (dest->flags & MF_SHADOW)
        an += (P_Random()-P_Random())<<20;

    th->angle = an;
    an >>= ANGLETOFINESHIFT;
    th->momx = FixedMul (th->info->speed, finecosine[an]);
    th->momy = FixedMul (th->info->speed, finesine[an]);

    dist = P_AproxDistance (dest->x - source->x, dest->y - source->y);
    dist = dist / th->info->speed;

    if (dist < 1)
        dist = 1;

    th->momz = (dest->z - source->z) / dist;
    P_CheckMissileSpawn (th);

    return th;
}

//
// P_SpawnPlayerMissile
// Tries to aim at a nearby monster
//
void
P_SpawnPlayerMissile
( mobj_t*      source,
  mobjtype_t    type )
{
    mobj_t*      th;
    angle_t      an;

    fixed_t      x;
    fixed_t      y;
    fixed_t      z;
    fixed_t      slope;

    // see which target is to be aimed at
    an = source->angle;
    slope = P_AimLineAttack (source, an, 16*64*FRACUNIT);

    if (!linetarget)
    {
        an += 1<<26;
    }
}

```

```

    slope = P_AimLineAttack (source, an, 16*64*FRACUNIT);

    if (!linetarget)
    {
        an -= 2<<26;
        slope = P_AimLineAttack (source, an, 16*64*FRACUNIT);
    }

    if (!linetarget)
    {
        an = source->angle;
        slope = 0;
    }
}

x = source->x;
y = source->y;
z = source->z + 4*8*FRACUNIT;

th = P_SpawnMobj (x,y,z, type);

if (th->info->seesound)
    S_StartSound (th, th->info->seesound);

th->target = source;
th->angle = an;
th->momx = FixedMul( th->info->speed,
                    finecosine[an>>ANGLETOFINESHIFT]);
th->momy = FixedMul( th->info->speed,
                    finesine[an>>ANGLETOFINESHIFT]);
th->momz = FixedMul( th->info->speed, slope);

P_CheckMissileSpawn (th);
}

```

## 9.14 p\_mobj.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      Map Objects, MObj, definition and handling.
//
//-----

#ifndef __P_MOBJ__
#define __P_MOBJ__

// Basics.

```

```

#include "tables.h"
#include "m_fixed.h"

// We need the thinker_t stuff.
#include "d_think.h"

// We need the WAD data structure for Map things,
// from the THINGS lump.
#include "doomdata.h"

// States are tied to finite states are
// tied to animation frames.
// Needs precompiled tables/data structures.
#include "info.h"

#ifdef __GNUG__
#pragma interface
#endif

//
// NOTES: mobj_t
//
// mobj_ts are used to tell the refresh where to draw an image,
// tell the world simulation when objects are contacted,
// and tell the sound driver how to position a sound.
//
// The refresh uses the next and prev links to follow
// lists of things in sectors as they are being drawn.
// The sprite, frame, and angle elements determine which patch_t
// is used to draw the sprite if it is visible.
// The sprite and frame values are allmost allways set
// from state_t structures.
// The statescr.exe utility generates the states.h and states.c
// files that contain the sprite/frame numbers from the
// statescr.txt source file.
// The xyz origin point represents a point at the bottom middle
// of the sprite (between the feet of a biped).
// This is the default origin position for patch_ts grabbed
// with lumpy.exe.
// A walking creature will have its z equal to the floor
// it is standing on.
//
// The sound code uses the x,y, and subsector fields
// to do stereo positioning of any sound effited by the mobj_t.
//
// The play simulation uses the blocklinks, x,y,z, radius, height
// to determine when mobj_ts are touching each other,
// touching lines in the map, or hit by trace lines (gunshots,
// lines of sight, etc).
// The mobj_t->flags element has various bit flags
// used by the simulation.
//
// Every mobj_t is linked into a single sector
// based on its origin coordinates.
// The subsector_t is found with R_PointInSubsector(x,y),
// and the sector_t can be found with subsector->sector.
// The sector links are only used by the rendering code,
// the play simulation does not care about them at all.
//
// Any mobj_t that needs to be acted upon by something else
// in the play world (block movement, be shot, etc) will also

```

```

// need to be linked into the blockmap.
// If the thing has the MF_NOBLOCK flag set, it will not use
// the block links. It can still interact with other things,
// but only as the instigator (missiles will run into other
// things, but nothing can run into a missile).
// Each block in the grid is 128*128 units, and knows about
// every line_t that it contains a piece of, and every
// interactable mobj_t that has its origin contained.
//
// A valid mobj_t is a mobj_t that has the proper subsector_t
// filled in for its xy coordinates and is linked into the
// sector from which the subsector was made, or has the
// MF_NOSECTOR flag set (the subsector_t needs to be valid
// even if MF_NOSECTOR is set), and is linked into a blockmap
// block or has the MF_NOBLOCKMAP flag set.
// Links should only be modified by the P_[Un]SetThingPosition()
// functions.
// Do not change the MF_NO? flags while a thing is valid.
//
// Any questions?
//

//
// Misc. mobj flags
//
typedef enum
{
    // Call P_SpecialThing when touched.
    MF_SPECIAL                = 1,
    // Blocks.
    MF_SOLID                  = 2,
    // Can be hit.
    MF_SHOOTABLE              = 4,
    // Don't use the sector links (invisible but touchable).
    MF_NOSECTOR               = 8,
    // Don't use the blocklinks (inert but displayable)
    MF_NOBLOCKMAP             = 16,

    // Not to be activated by sound, deaf monster.
    MF_AMBUSH                 = 32,
    // Will try to attack right back.
    MF_JUSTHIT                = 64,
    // Will take at least one step before attacking.
    MF_JUSTATTACKED           = 128,
    // On level spawning (initial position),
    // hang from ceiling instead of stand on floor.
    MF_SPAWNCEILING           = 256,
    // Don't apply gravity (every tic),
    // that is, object will float, keeping current height
    // or changing it actively.
    MF_NOGRAVITY              = 512,

    // Movement flags.
    // This allows jumps from high places.
    MF_DROPOFF                = 0x400,
    // For players, will pick up items.
    MF_PICKUP                 = 0x800,
    // Player cheat. ???
    MF_NOCLIP                 = 0x1000,
    // Player: keep info about sliding along walls.
    MF_SLIDE                  = 0x2000,
    // Allow moves to any height, no gravity.
    // For active floaters, e.g. cacodemons, pain elementals.
    MF_FLOAT                  = 0x4000,
    // Don't cross lines

```



```

// ??? or look at heights on teleport.
MF_TELEPORT          = 0x8000,
// Don't hit same species, explode on block.
// Player missiles as well as fireballs of various kinds.
MF_MISSILE           = 0x10000,
// Dropped by a demon, not level spawned.
// E.g. ammo clips dropped by dying former humans.
MF_DROPPED           = 0x20000,
// Use fuzzy draw (shadow demons or spectres),
// temporary player invisibility powerup.
MF_SHADOW            = 0x40000,
// Flag: don't bleed when shot (use puff),
// barrels and shootable furniture shall not bleed.
MF_NOBLOOD           = 0x80000,
// Don't stop moving halfway off a step,
// that is, have dead bodies slide down all the way.
MF_CORPSE            = 0x100000,
// Floating to a height for a move, ???
// don't auto float to target's height.
MF_INFLOAT           = 0x200000,

// On kill, count this enemy object
// towards intermission kill total.
// Happy gathering.
MF_COUNTKILL         = 0x400000,

// On picking up, count this item object
// towards intermission item total.
MF_COUNTITEM         = 0x800000,

// Special handling: skull in flight.
// Neither a cacodemon nor a missile.
MF_SKULLFLY          = 0x1000000,

// Don't spawn this object
// in death match mode (e.g. key cards).
MF_NOTDMATCH         = 0x2000000,

// Player sprites in multiplayer modes are modified
// using an internal color lookup table for re-indexing.
// If 0x4 0x8 or 0xc,
// use a translation table for player colormaps
MF_TRANSLATION        = 0xc000000,
// Hmm ????.
MF_TRANSShift        = 26

} mobjflag_t;

// Map Object definition.
typedef struct mobj_s
{
    // List: thinker links.
    thinker_t          thinker;

    // Info for drawing: position.
    fixed_t             x;
    fixed_t             y;
    fixed_t             z;

    // More list: links in sector (if needed)
    struct mobj_s*       snext;
    struct mobj_s*       sprev;

    //More drawing info: to determine current sprite.

```

```

angle_t            angle;          // orientation
spritenum_t        sprite;         // used to find patch_t and flip value
int                frame;          // might be ORed with FF_FULLBRIGHT

// Interaction info, by BLOCKMAP.
// Links in blocks (if needed).
struct mobj_s*      bnext;
struct mobj_s*      bprev;

struct subsector_s* subsector;

// The closest interval over all contacted Sectors.
fixed_t            floorz;
fixed_t            ceilingz;

// For movement checking.
fixed_t            radius;
fixed_t            height;

// Momentums, used to update position.
fixed_t            momx;
fixed_t            momy;
fixed_t            momz;

// If == validcount, already checked.
int                validcount;

mobjtype_t          type;
mobjinfo_t*         info;          // &mobjinfo[mobj->type]

int                tics;          // state tic counter
state_t*            state;
int                flags;
int                health;

// Movement direction, movement generation (zig-zagging).
int                movedir;        // 0-7
int                movecount;      // when 0, select a new dir

// Thing being chased/attacked (or NULL),
// also the originator for missiles.
struct mobj_s*      target;

// Reaction time: if non 0, don't attack yet.
// Used by player to freeze a bit after teleporting.
int                reactiontime;

// If >0, the target will be chased
// no matter what (even if shot)
int                threshold;

// Additional info record for player avatars only.
// Only valid if type == MT_PLAYER
struct player_s*     player;

// Player number last looked for.
int                lastlook;

// For nightmare respawn.
mapthing_t          spawnpoint;

// Thing being chased/attacked for tracers.
struct mobj_s*      tracer;
} mobj_t;

```

```
#endif
//-----
//
// $Log:$
//
//-----
```

## 9.15 p\_plats.c

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Plats (i.e. elevator platforms) code, raising/lowering.
//
//-----

static const char
rcsid[] = "$Id: p_plats.c,v 1.5 1997/02/03 22:45:12 b1 Exp $";

#include "i_system.h"
#include "z_zone.h"
#include "m_random.h"

#include "doomdef.h"
#include "p_local.h"

#include "s_sound.h"

// State.
#include "doomstat.h"
#include "r_state.h"

// Data.
#include "sounds.h"

plat_t*          activeplats[MAXPLATS];

//
// Move a plat up and down
//
void T_PlatformRaise(plat_t* plat)
```

```

{
    result_e      res;

    switch(plat->status)
    {
        case up:
            res = T_MovePlane(plat->sector,
                              plat->speed,
                              plat->high,
                              plat->crush,0,1);

            if (plat->type == raiseAndChange
                || plat->type == raiseToNearestAndChange)
            {
                if (!(leveltime&7))
                    S_StartSound((mobj_t *)&plat->sector->soundorg,
                                sfx_stnmov);
            }

            if (res == crushed && (!plat->crush))
            {
                plat->count = plat->wait;
                plat->status = down;
                S_StartSound((mobj_t *)&plat->sector->soundorg,
                            sfx_pstart);
            }
            else
            {
                if (res == pastdest)
                {
                    plat->count = plat->wait;
                    plat->status = waiting;
                    S_StartSound((mobj_t *)&plat->sector->soundorg,
                                sfx_pstop);

                    switch(plat->type)
                    {
                        case blazeDWUS:
                        case downWaitUpStay:
                            P_RemoveActivePlat(plat);
                            break;

                        case raiseAndChange:
                        case raiseToNearestAndChange:
                            P_RemoveActivePlat(plat);
                            break;

                        default:
                            break;
                    }
                }
            }
            break;

        case down:
            res = T_MovePlane(plat->sector,plat->speed,plat->low,false,0,-1);

            if (res == pastdest)
            {
                plat->count = plat->wait;
                plat->status = waiting;
                S_StartSound((mobj_t *)&plat->sector->soundorg,sfx_pstop);
            }
            break;
    }
}

```

```

    case        waiting:
        if (!--plat->count)
        {
            if (plat->sector->floorheight == plat->low)
                plat->status = up;
            else
                plat->status = down;
            S_StartSound((mobj_t *)&plat->sector->soundorg,sfx_pstart);
        }
    case        in_stasis:
        break;
}
}

```

```

//
// Do Platforms
// "amount" is only used for SOME platforms.
//

```

```

int
EV_DoPlat
( line_t*      line,
  platttype_e   type,
  int           amount )
{

```

```

    plat_t*      plat;
    int          secnum;
    int          rtn;
    sector_t*    sec;

```

```

    secnum = -1;
    rtn = 0;

```

```

//      Activate all <type> plats that are in_stasis
switch(type)
{
    case perpetualRaise:
        P_ActivateInStasis(line->tag);
        break;

    default:
        break;
}

```

```

while ((secnum = P_FindSectorFromLineTag(line,secnum)) >= 0)
{

```

```

    sec = &sectors[secnum];

```

```

    if (sec->specialdata)
        continue;

```

```

    // Find lowest & highest floors around sector

```

```

    rtn = 1;

```

```

    plat = Z_Malloc( sizeof(*plat), PU_LEVSPEC, 0);

```

```

    P_AddThinker(&plat->thinker);

```

```

    plat->type = type;

```

```

    plat->sector = sec;

```

```

    plat->sector->specialdata = plat;

```

```

    plat->thinker.function.acp1 = (actionf_p1) T_PlatRaise;

```

```

    plat->crush = false;

```

```

    plat->tag = line->tag;

```

```

switch(type)
{
    case raiseToNearestAndChange:
        plat->speed = PLATSPEED/2;
        sec->floorpic = sides[line->sidenum[0]].sector->floorpic;
        plat->high = P_FindNextHighestFloor(sec,sec->floorheight);
        plat->wait = 0;
        plat->status = up;
        // NO MORE DAMAGE, IF APPLICABLE
        sec->special = 0;

        S_StartSound((mobj_t *)&sec->soundorg,sfx_stnmov);
        break;

    case raiseAndChange:
        plat->speed = PLATSPEED/2;
        sec->floorpic = sides[line->sidenum[0]].sector->floorpic;
        plat->high = sec->floorheight + amount*FRACUNIT;
        plat->wait = 0;
        plat->status = up;

        S_StartSound((mobj_t *)&sec->soundorg,sfx_stnmov);
        break;

    case downWaitUpStay:
        plat->speed = PLATSPEED * 4;
        plat->low = P_FindLowestFloorSurrounding(sec);

        if (plat->low > sec->floorheight)
            plat->low = sec->floorheight;

        plat->high = sec->floorheight;
        plat->wait = 35*PLATWAIT;
        plat->status = down;
        S_StartSound((mobj_t *)&sec->soundorg,sfx_pstart);
        break;

    case blazeDWUS:
        plat->speed = PLATSPEED * 8;
        plat->low = P_FindLowestFloorSurrounding(sec);

        if (plat->low > sec->floorheight)
            plat->low = sec->floorheight;

        plat->high = sec->floorheight;
        plat->wait = 35*PLATWAIT;
        plat->status = down;
        S_StartSound((mobj_t *)&sec->soundorg,sfx_pstart);
        break;

    case perpetualRaise:
        plat->speed = PLATSPEED;
        plat->low = P_FindLowestFloorSurrounding(sec);

        if (plat->low > sec->floorheight)
            plat->low = sec->floorheight;

        plat->high = P_FindHighestFloorSurrounding(sec);

        if (plat->high < sec->floorheight)
            plat->high = sec->floorheight;

        plat->wait = 35*PLATWAIT;
        plat->status = P_Random()&1;

```

```

        S_StartSound((mobj_t *)&sec->soundorg,sfx_pstart);
        break;
    }
    P_AddActivePlat(plat);
}
return rtn;
}

void P_ActivateInStasis(int tag)
{
    int i;

    for (i = 0;i < MAXPLATS;i++)
        if (activeplats[i]
            && (activeplats[i])->tag == tag
            && (activeplats[i])->status == in_stasis)
        {
            (activeplats[i])->status = (activeplats[i])->oldstatus;
            (activeplats[i])->thinker.function.acp1
                = (actionf_p1) T_PlatRaise;
        }
}

void EV_StopPlat(line_t* line)
{
    int j;

    for (j = 0;j < MAXPLATS;j++)
        if (activeplats[j]
            && ((activeplats[j])->status != in_stasis)
            && ((activeplats[j])->tag == line->tag))
        {
            (activeplats[j])->oldstatus = (activeplats[j])->status;
            (activeplats[j])->status = in_stasis;
            (activeplats[j])->thinker.function.acv = (actionf_v)NULL;
        }
}

void P_AddActivePlat(plat_t* plat)
{
    int i;

    for (i = 0;i < MAXPLATS;i++)
        if (activeplats[i] == NULL)
        {
            activeplats[i] = plat;
            return;
        }
    I_Error ("P_AddActivePlat: no more plats!");
}

void P_RemoveActivePlat(plat_t* plat)
{
    int i;
    for (i = 0;i < MAXPLATS;i++)
        if (plat == activeplats[i])
        {
            (activeplats[i])->sector->specialdata = NULL;
            P_RemoveThinker(&(activeplats[i])->thinker);
            activeplats[i] = NULL;

            return;
        }
}

```

```

    I_Error ("P_RemoveActivePlat: can't find plat!");
}

```

## 9.16 p\_pspr.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Weapon sprite animation, weapon objects.
//     Action functions for weapons.
//-----

static const char
rcsid[] = "$Id: p_pspr.c,v 1.5 1997/02/03 22:45:12 b1 Exp $";

#include "doomdef.h"
#include "d_event.h"

#include "m_random.h"
#include "p_local.h"
#include "s_sound.h"

// State.
#include "doomstat.h"

// Data.
#include "sounds.h"

#include "p_pspr.h"

#define LOWERSPEED          FRACUNIT*6
#define RAISESPEED          FRACUNIT*6

#define WEAPONBOTTOM        128*FRACUNIT
#define WEAPONTOP           32*FRACUNIT

// plasma cells for a bfg attack
#define BFGCELLS            40

//
// P_SetPsprite
//
void
P_SetPsprite

```



```

( player_t*      player,
  int           position,
  statenum_t    stnum )
{
    pspdef_t*    psp;
    state_t*     state;

    psp = &player->psprites[position];

    do
    {
        if (!stnum)
        {
            // object removed itself
            psp->state = NULL;
            break;
        }

        state = &states[stnum];
        psp->state = state;
        psp->tics = state->tics;          // could be 0

        if (state->misc1)
        {
            // coordinate set
            psp->sx = state->misc1 << FRACBITS;
            psp->sy = state->misc2 << FRACBITS;
        }

        // Call action routine.
        // Modified handling.
        if (state->action.acp2)
        {
            state->action.acp2(player, psp);
            if (!psp->state)
                break;
        }

        stnum = psp->state->nextstate;

    } while (!psp->tics);
    // an initial state of 0 could cycle through
}

```

```

//
// P_CalcSwing
//
fixed_t      swingx;
fixed_t      swingy;

void P_CalcSwing (player_t*      player)
{
    fixed_t    swing;
    int        angle;

    // OPTIMIZE: tablify this.
    // A LUT would allow for different modes,
    // and add flexibility.

    swing = player->bob;

    angle = (FINEANGLES/70*leveltime)&FINEMASK;
    swingx = FixedMul ( swing, finesine[angle]);

```

```

    angle = (FINEANGLES/70*leveltime+FINEANGLES/2)&FINEMASK;
    swingy = -FixedMul ( swingx, finesine[angle]);
}

```

```

//
// P_BringUpWeapon
// Starts bringing the pending weapon up
// from the bottom of the screen.
// Uses player
//
void P_BringUpWeapon (player_t* player)
{
    statenum_t      newstate;

    if (player->pendingweapon == wp_nochange)
        player->pendingweapon = player->readyweapon;

    if (player->pendingweapon == wp_chainsaw)
        S_StartSound (player->mo, sfx_sawup);

    newstate = weaponinfo[player->pendingweapon].upstate;

    player->pendingweapon = wp_nochange;
    player->psprites[ps_weapon].sy = WEAPONBOTTOM;

    P_SetPsprite (player, ps_weapon, newstate);
}

```

```

//
// P_CheckAmmo
// Returns true if there is enough ammo to shoot.
// If not, selects the next weapon to use.
//
boolean P_CheckAmmo (player_t* player)
{
    ammotype_t      ammo;
    int              count;

    ammo = weaponinfo[player->readyweapon].ammo;

    // Minimal amount for one shot varies.
    if (player->readyweapon == wp_bfg)
        count = BFGCELLS;
    else if (player->readyweapon == wp_supershotgun)
        count = 2;          // Double barrel.
    else
        count = 1;          // Regular.

    // Some do not need ammunition anyway.
    // Return if current ammunition sufficient.
    if (ammo == am_noammo || player->ammo[ammo] >= count)
        return true;

    // Out of ammo, pick a weapon to change to.
    // Preferences are set here.
    do
    {
        if (player->weaponowned[wp_plasma]
            && player->ammo[am_cell]
            && (gamemode != shareware) )
        {
            player->pendingweapon = wp_plasma;

```

```

    }
    else if (player->weaponowned[wp_supershotgun]
             && player->ammo[am_shell]>2
             && (gamemode == commercial) )
    {
        player->pendingweapon = wp_supershotgun;
    }
    else if (player->weaponowned[wp_chaingun]
             && player->ammo[am_clip])
    {
        player->pendingweapon = wp_chaingun;
    }
    else if (player->weaponowned[wp_shotgun]
             && player->ammo[am_shell])
    {
        player->pendingweapon = wp_shotgun;
    }
    else if (player->ammo[am_clip])
    {
        player->pendingweapon = wp_pistol;
    }
    else if (player->weaponowned[wp_chainsaw])
    {
        player->pendingweapon = wp_chainsaw;
    }
    else if (player->weaponowned[wp_missile]
             && player->ammo[am_misl])
    {
        player->pendingweapon = wp_missile;
    }
    else if (player->weaponowned[wp_bfg]
             && player->ammo[am_cell]>40
             && (gamemode != shareware) )
    {
        player->pendingweapon = wp_bfg;
    }
    else
    {
        // If everything fails.
        player->pendingweapon = wp_fist;
    }

} while (player->pendingweapon == wp_nochange);

// Now set appropriate weapon overlay.
P_SetPsprite (player,
              ps_weapon,
              weaponinfo[player->readyweapon].downstate);

return false;
}

//
// P_FireWeapon.
//
void P_FireWeapon (player_t* player)
{
    statenum_t      newstate;

    if (!P_CheckAmmo (player))
        return;

    P_SetMobjState (player->mo, S_PLAY_ATK1);
    newstate = weaponinfo[player->readyweapon].atkstate;

```

```

    P_SetPsprite (player, ps_weapon, newstate);
    P_NoiseAlert (player->mo, player->mo);
}

//
// P_DropWeapon
// Player died, so put the weapon away.
//
void P_DropWeapon (player_t* player)
{
    P_SetPsprite (player,
                  ps_weapon,
                  weaponinfo[player->readyweapon].downstate);
}

//
// A_WeaponReady
// The player can fire the weapon
// or change to another weapon at this time.
// Follows after getting weapon up,
// or after previous attack/fire sequence.
//
void
A_WeaponReady
( player_t*      player,
  pspdef_t*      psp )
{
    statenum_t    newstate;
    int           angle;

    // get out of attack state
    if (player->mo->state == &states[S_PLAY_ATK1]
        || player->mo->state == &states[S_PLAY_ATK2] )
    {
        P_SetMobjState (player->mo, S_PLAY);
    }

    if (player->readyweapon == wp_chainsaw
        && psp->state == &states[S_SAW])
    {
        S_StartSound (player->mo, sfx_sawidl);
    }

    // check for change
    // if player is dead, put the weapon away
    if (player->pendingweapon != wp_nochange || !player->health)
    {
        // change weapon
        // (pending weapon should already be validated)
        newstate = weaponinfo[player->readyweapon].downstate;
        P_SetPsprite (player, ps_weapon, newstate);
        return;
    }

    // check for fire
    // the missile launcher and bfg do not auto fire
    if (player->cmd.buttons & BT_ATTACK)
    {
        if ( !player->attackdown
            || (player->readyweapon != wp_missile
                && player->readyweapon != wp_bfg) )

```

```

    {
        player->attackdown = true;
        P_FireWeapon (player);
        return;
    }
}
else
    player->attackdown = false;

// bob the weapon based on movement speed
angle = (128*leveltime)&FINEMASK;
psp->sx = FRACUNIT + FixedMul (player->bob, finecosine[angle]);
angle &= FINEANGLES/2-1;
psp->sy = WEAPONTOP + FixedMul (player->bob, finesine[angle]);
}

```

```

//
// A_ReFire
// The player can re-fire the weapon
// without lowering it entirely.
//
void A_ReFire
( player_t*      player,
  pspdef_t*      psp )
{

```

```

    // check for fire
    // (if a weaponchange is pending, let it go through instead)
    if ( (player->cmd.buttons & BT_ATTACK)
        && player->pendingweapon == wp_nochange
        && player->health)
    {
        player->refire++;
        P_FireWeapon (player);
    }
    else
    {
        player->refire = 0;
        P_CheckAmmo (player);
    }
}

```

```

void
A_CheckReload
( player_t*      player,
  pspdef_t*      psp )
{
    P_CheckAmmo (player);
    #if 0
        if (player->ammo[am_shell]<2)
            P_SetPsprite (player, ps_weapon, S_DSNR1);
    #endif
}

```

```

//
// A_Lower
// Lowers current weapon,
// and changes weapon at bottom.
//
void

```

```

A_Lower
( player_t*      player,
  pspdef_t*      psp )
{
    psp->sy += LOWERSPEED;

    // Is already down.
    if (psp->sy < WEAPONBOTTOM )
        return;

    // Player is dead.
    if (player->playerstate == PST_DEAD)
    {
        psp->sy = WEAPONBOTTOM;

        // don't bring weapon back up
        return;
    }

    // The old weapon has been lowered off the screen,
    // so change the weapon and start raising it
    if (!player->health)
    {
        // Player is dead, so keep the weapon off screen.
        P_SetPsprite (player, ps_weapon, S_NULL);
        return;
    }

    player->readyweapon = player->pendingweapon;

    P_BringUpWeapon (player);
}

//
// A_Raise
//
void
A_Raise
( player_t*      player,
  pspdef_t*      psp )
{
    statenum_t    newstate;

    psp->sy -= RAISESPEED;

    if (psp->sy > WEAPONTOP )
        return;

    psp->sy = WEAPONTOP;

    // The weapon has been raised all the way,
    // so change to the ready state.
    newstate = weaponinfo[player->readyweapon].readystate;

    P_SetPsprite (player, ps_weapon, newstate);
}

//
// A_GunFlash
//
void
A_GunFlash

```

```

( player_t*      player,
  pspdef_t*      psp )
{
    P_SetMobjState (player->mo, S_PLAY_ATK2);
    P_SetPsprite (player,ps_flash,weaponinfo[player->readyweapon].flashstate);
}

```

```

//
// WEAPON ATTACKS
//

```

```

//
// A_Punch
//
void
A_Punch
( player_t*      player,
  pspdef_t*      psp )
{
    angle_t      angle;
    int          damage;
    int          slope;

    damage = (P_Random ()%10+1)<<1;

    if (player->powers[pw_strength])
        damage *= 10;

    angle = player->mo->angle;
    angle += (P_Random()-P_Random())<<18;
    slope = P_AimLineAttack (player->mo, angle, MELEERANGE);
    P_LineAttack (player->mo, angle, MELEERANGE, slope, damage);

    // turn to face target
    if (linetarget)
    {
        S_StartSound (player->mo, sfx_punch);
        player->mo->angle = R_PointToAngle2 (player->mo->x,
                                             player->mo->y,
                                             linetarget->x,
                                             linetarget->y);
    }
}

```

```

//
// A_Saw
//
void
A_Saw
( player_t*      player,
  pspdef_t*      psp )
{
    angle_t      angle;
    int          damage;
    int          slope;

    damage = 2*(P_Random ()%10+1);
    angle = player->mo->angle;
    angle += (P_Random()-P_Random())<<18;

    // use meleerange + 1 se the puff doesn't skip the flash

```

```

slope = P_AimLineAttack (player->mo, angle, MELEERANGE+1);
P_LineAttack (player->mo, angle, MELEERANGE+1, slope, damage);

if (!linetarget)
{
    S_StartSound (player->mo, sfx_sawful);
    return;
}
S_StartSound (player->mo, sfx_sawhit);

// turn to face target
angle = R_PointToAngle2 (player->mo->x, player->mo->y,
                        linetarget->x, linetarget->y);
if (angle - player->mo->angle > ANG180)
{
    if (angle - player->mo->angle < -ANG90/20)
        player->mo->angle = angle + ANG90/21;
    else
        player->mo->angle -= ANG90/20;
}
else
{
    if (angle - player->mo->angle > ANG90/20)
        player->mo->angle = angle - ANG90/21;
    else
        player->mo->angle += ANG90/20;
}
player->mo->flags |= MF_JUSTATTACKED;
}

```

```

//
// A_FireMissile
//
void
A_FireMissile
( player_t*      player,
  pspdef_t*      psp )
{
    player->ammo[weaponinfo[player->readyweapon].ammo]--;
    P_SpawnPlayerMissile (player->mo, MT_ROCKET);
}

```

```

//
// A_FireBFG
//
void
A_FireBFG
( player_t*      player,
  pspdef_t*      psp )
{
    player->ammo[weaponinfo[player->readyweapon].ammo] -= BFGCELLS;
    P_SpawnPlayerMissile (player->mo, MT_BFG);
}

```

```

//
// A_FirePlasma
//
void
A_FirePlasma
( player_t*      player,

```



```

    pspdef_t*      psp )
{
    player->ammo[weaponinfo[player->readyweapon].ammo]--;

    P_SetPsprite (player,
                  ps_flash,
                  weaponinfo[player->readyweapon].flashstate+(P_Random ()&1) );

    P_SpawnPlayerMissile (player->mo, MT_PLASMA);
}

```

```

//
// P_BulletSlope
// Sets a slope so a near miss is at aproximately
// the height of the intended target
//
fixed_t      bulletslope;

```

```

void P_BulletSlope (mobj_t*      mo)
{
    angle_t      an;

    // see which target is to be aimed at
    an = mo->angle;
    bulletslope = P_AimLineAttack (mo, an, 16*64*FRACUNIT);

    if (!linetarget)
    {
        an += 1<<26;
        bulletslope = P_AimLineAttack (mo, an, 16*64*FRACUNIT);
        if (!linetarget)
        {
            an -= 2<<26;
            bulletslope = P_AimLineAttack (mo, an, 16*64*FRACUNIT);
        }
    }
}

```

```

//
// P_GunShot
//
void
P_GunShot
( mobj_t*      mo,
  boolean      accurate )
{
    angle_t      angle;
    int          damage;

    damage = 5*(P_Random ()%3+1);
    angle = mo->angle;

    if (!accurate)
        angle += (P_Random()-P_Random())<<18;

    P_LineAttack (mo, angle, MISSILERANGE, bulletslope, damage);
}

```

```

//
// A_FirePistol

```

```

//
void
A_FirePistol
( player_t*      player,
  pspdef_t*      psp )
{
    S_StartSound (player->mo, sfx_pistol);

    P_SetMobjState (player->mo, S_PLAY_ATK2);
    player->ammo[weaponinfo[player->readyweapon].ammo]--;

    P_SetPsprite (player,
                  ps_flash,
                  weaponinfo[player->readyweapon].flashstate);

    P_BulletSlope (player->mo);
    P_GunShot (player->mo, !player->refire);
}

```

```

//
// A_FireShotgun
//
void
A_FireShotgun
( player_t*      player,
  pspdef_t*      psp )
{
    int          i;

    S_StartSound (player->mo, sfx_shotgn);
    P_SetMobjState (player->mo, S_PLAY_ATK2);

    player->ammo[weaponinfo[player->readyweapon].ammo]--;

    P_SetPsprite (player,
                  ps_flash,
                  weaponinfo[player->readyweapon].flashstate);

    P_BulletSlope (player->mo);

    for (i=0 ; i<7 ; i++)
        P_GunShot (player->mo, false);
}

```

```

//
// A_FireShotgun2
//
void
A_FireShotgun2
( player_t*      player,
  pspdef_t*      psp )
{
    int          i;
    angle_t      angle;
    int          damage;

    S_StartSound (player->mo, sfx_dshtgn);
    P_SetMobjState (player->mo, S_PLAY_ATK2);

    player->ammo[weaponinfo[player->readyweapon].ammo]-=2;
}

```

```

P_SetPsprite (player,
              ps_flash,
              weaponinfo[player->readyweapon].flashstate);

P_BulletSlope (player->mo);

for (i=0 ; i<20 ; i++)
{
    damage = 5*(P_Random ()%3+1);
    angle = player->mo->angle;
    angle += (P_Random()-P_Random())<<19;
    P_LineAttack (player->mo,
                  angle,
                  MISSILERANGE,
                  bulletslope + ((P_Random()-P_Random())<<5), damage);
}
}

```

```

//
// A_FireCGun
//
void
A_FireCGun
( player_t*      player,
  pspdef_t*      psp )
{
    S_StartSound (player->mo, sfx_pistol);

    if (!player->ammo[weaponinfo[player->readyweapon].ammo])
        return;

    P_SetMobjState (player->mo, S_PLAY_ATK2);
    player->ammo[weaponinfo[player->readyweapon].ammo]--;

    P_SetPsprite (player,
                  ps_flash,
                  weaponinfo[player->readyweapon].flashstate
                  + psp->state
                  - &states[S_CHAIN1] );

    P_BulletSlope (player->mo);

    P_GunShot (player->mo, !player->refire);
}

```

```

//
// ?
//
void A_Light0 (player_t *player, pspdef_t *psp)
{
    player->extralight = 0;
}

```

```

void A_Light1 (player_t *player, pspdef_t *psp)
{
    player->extralight = 1;
}

```

```

void A_Light2 (player_t *player, pspdef_t *psp)
{
    player->extralight = 2;
}

```

```

//
// A_BFGSpray
// Spawn a BFG explosion on every monster in view
//
void A_BFGSpray (mobj_t* mo)
{
    int                i;
    int                j;
    int                damage;
    angle_t            an;

    // offset angles from its attack angle
    for (i=0 ; i<40 ; i++)
    {
        an = mo->angle - ANG90/2 + ANG90/40*i;

        // mo->target is the originator (player)
        // of the missile
        P_AimLineAttack (mo->target, an, 16*64*FRACUNIT);

        if (!linetarget)
            continue;

        P_SpawnMobj (linetarget->x,
                    linetarget->y,
                    linetarget->z + (linetarget->height>>2),
                    MT_EXTRABFG);

        damage = 0;
        for (j=0;j<15;j++)
            damage += (P_Random()&7) + 1;

        P_DamageMobj (linetarget, mo->target,mo->target, damage);
    }
}

```

```

//
// A_BFGsound
//
void
A_BFGsound
( player_t*          player,
  pspdef_t*          psp )
{
    S_StartSound (player->mo, sfx_bfg);
}

```

```

//
// P_SetupPspprites
// Called at start of level for each player.
//
void P_SetupPspprites (player_t* player)
{
    int                i;

    // remove all psprites
    for (i=0 ; i<NUMPSPRITES ; i++)
        player->psprites[i].state = NULL;

    // spawn the gun

```

```

    player->pendingweapon = player->readyweapon;
    P_BringUpWeapon (player);
}

//
// P_MovePsprites
// Called every tic by player thinking routine.
//
void P_MovePsprites (player_t* player)
{
    int            i;
    pspdef_t*      psp;
    state_t*       state;

    psp = &player->psprites[0];
    for (i=0 ; i<NUMPSPRITES ; i++, psp++)
    {
        // a null state means not active
        if ( (state = psp->state) )
        {
            // drop tic count and possibly change state

            // a -1 tic count never changes
            if (psp->tics != -1)
            {
                psp->tics--;
                if (!psp->tics)
                    P_SetPsprite (player, i, psp->state->nextstate);
            }
        }
    }

    player->psprites[ps_flash].sx = player->psprites[ps_weapon].sx;
    player->psprites[ps_flash].sy = player->psprites[ps_weapon].sy;
}

```

## 9.17 p\_pspr.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
// Sprite animation.
//-----

```

```

#ifndef __P_PSPR__
#define __P_PSPR__

// Basic data types.
// Needs fixed point, and BAM angles.
#include "m_fixed.h"
#include "tables.h"

//
// Needs to include the precompiled
// sprite animation tables.
// Header generated by multigen utility.
// This includes all the data for thing animation,
// i.e. the Thing Attributes table
// and the Frame Sequence table.
#include "info.h"

#ifdef __GNUG__
#pragma interface
#endif

//
// Frame flags:
// handles maximum brightness (torches, muzzle flare, light sources)
//
#define FF_FULLBRIGHT      0x8000      // flag in thing->frame
#define FF_FRAMEMASK      0x7fff

//
// Overlay psprites are scaled shapes
// drawn directly on the view screen,
// coordinates are given for a 320*200 view screen.
//
typedef enum
{
    ps_weapon,
    ps_flash,
    NUMPSPRITES
} psprnum_t;

typedef struct
{
    state_t*      state;      // a NULL state means not active
    int           tics;
    fixed_t      sx;
    fixed_t      sy;
} pspdef_t;

#endif
//-----
//
// $Log:$
//
//-----

```

## 9.18 p\_saveg.c

```
// Emacs style mode select  -*- C++ -*-
```

```

//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Archiving: SaveGame I/O.
//-----

static const char
rcsid[] = "$Id: p_tick.c,v 1.4 1997/02/03 16:47:55 b1 Exp $";

#include "i_system.h"
#include "z_zone.h"
#include "p_local.h"

// State.
#include "doomstat.h"
#include "r_state.h"

byte*          save_p;

// Pads save_p to a 4-byte boundary
// so that the load/save works on SGI&Gecko.
#define PADSAVEP()      save_p += (4 - ((int) save_p & 3)) & 3

//
// P_ArchivePlayers
//
void P_ArchivePlayers (void)
{
    int          i;
    int          j;
    player_t*    dest;

    for (i=0 ; i<MAXPLAYERS ; i++)
    {
        if (!playeringame[i])
            continue;

        PADSAVEP();

        dest = (player_t *)save_p;
        memcpy (dest,&players[i],sizeof(player_t));
        save_p += sizeof(player_t);
        for (j=0 ; j<NUMPSPRITES ; j++)
        {
            if (dest->psprites[j].state)

```

```

        {
            dest->psprites[j].state
                = (state_t *) (dest->psprites[j].state-states);
        }
    }
}

```

```

//
// P_UnArchivePlayers
//
void P_UnArchivePlayers (void)
{
    int            i;
    int            j;

    for (i=0 ; i<MAXPLAYERS ; i++)
    {
        if (!playeringame[i])
            continue;

        PADSAVEP();

        memcpy (&players[i],save_p, sizeof(player_t));
        save_p += sizeof(player_t);

        // will be set when unarc thinker
        players[i].mo = NULL;
        players[i].message = NULL;
        players[i].attacker = NULL;

        for (j=0 ; j<NUMPSPRITES ; j++)
        {
            if (players[i]. psprites[j].state)
            {
                players[i]. psprites[j].state
                    = &states[ (int)players[i].psprites[j].state ];
            }
        }
    }
}

```

```

//
// P_ArchiveWorld
//
void P_ArchiveWorld (void)
{
    int            i;
    int            j;
    sector_t*      sec;
    line_t*        li;
    side_t*        si;
    short*         put;

    put = (short *)save_p;

    // do sectors
    for (i=0, sec = sectors ; i<numsectors ; i++,sec++)
    {
        *put++ = sec->floorheight >> FRACBITS;
        *put++ = sec->ceilingheight >> FRACBITS;
        *put++ = sec->floorpic;
    }
}

```



```

        *put++ = sec->ceilingpic;
        *put++ = sec->lightlevel;
        *put++ = sec->special;           // needed?
        *put++ = sec->tag;               // needed?
    }

// do lines
for (i=0, li = lines ; i<numlines ; i++,li++)
{
    *put++ = li->flags;
    *put++ = li->special;
    *put++ = li->tag;
    for (j=0 ; j<2 ; j++)
    {
        if (li->sidenum[j] == -1)
            continue;

        si = &sides[li->sidenum[j]];

        *put++ = si->textureoffset >> FRACBITS;
        *put++ = si->rowoffset >> FRACBITS;
        *put++ = si->toptexture;
        *put++ = si->bottomtexture;
        *put++ = si->midtexture;
    }
}

save_p = (byte *)put;
}

//
// P_UnArchiveWorld
//
void P_UnArchiveWorld (void)
{
    int                i;
    int                j;
    sector_t*          sec;
    line_t*            li;
    side_t*            si;
    short*             get;

    get = (short *)save_p;

// do sectors
for (i=0, sec = sectors ; i<numsectors ; i++,sec++)
{
    sec->floorheight = *get++ << FRACBITS;
    sec->ceilingheight = *get++ << FRACBITS;
    sec->floorpic = *get++;
    sec->ceilingpic = *get++;
    sec->lightlevel = *get++;
    sec->special = *get++;           // needed?
    sec->tag = *get++;              // needed?
    sec->specialdata = 0;
    sec->soundtarget = 0;
}

// do lines
for (i=0, li = lines ; i<numlines ; i++,li++)
{
    li->flags = *get++;

```

```

    li->special = *get++;
    li->tag = *get++;
    for (j=0 ; j<2 ; j++)
    {
        if (li->sidenum[j] == -1)
            continue;
        si = &sides[li->sidenum[j]];
        si->textureoffset = *get++ << FRACBITS;
        si->rowoffset = *get++ << FRACBITS;
        si->toptexture = *get++;
        si->bottomtexture = *get++;
        si->midtexture = *get++;
    }
}
save_p = (byte *)get;
}

//
// Thinkers
//
typedef enum
{
    tc_end,
    tc_mobj
} thinkerclass_t;

//
// P_ArchiveThinkers
//
void P_ArchiveThinkers (void)
{
    thinker_t*      th;
    mobj_t*         mobj;

    // save off the current thinkers
    for (th = thinkercap.next ; th != &thinkercap ; th=th->next)
    {
        if (th->function.acp1 == (actionf_p1)P_MobjThinker)
        {
            *save_p++ = tc_mobj;
            PADSAVEP();
            mobj = (mobj_t *)save_p;
            memcpy (mobj, th, sizeof(*mobj));
            save_p += sizeof(*mobj);
            mobj->state = (state_t *) (mobj->state - states);

            if (mobj->player)
                mobj->player = (player_t *) ((mobj->player-players) + 1);
            continue;
        }

        // I_Error ("P_ArchiveThinkers: Unknown thinker function");
    }

    // add a terminating marker
    *save_p++ = tc_end;
}

```

```

//
// P_UnArchiveThinkers
//
void P_UnArchiveThinkers (void)
{
    byte                tclass;
    thinker_t*          currentthinker;
    thinker_t*          next;
    mobj_t*             mobj;

    // remove all the current thinkers
    currentthinker = thinkercap.next;
    while (currentthinker != &thinkercap)
    {
        next = currentthinker->next;

        if (currentthinker->function.acp1 == (actionf_p1)P_MobjThinker)
            P_RemoveMobj ((mobj_t *)currentthinker);
        else
            Z_Free (currentthinker);

        currentthinker = next;
    }
    P_InitThinkers ();

    // read in saved thinkers
    while (1)
    {
        tclass = *save_p++;
        switch (tclass)
        {
            case tc_end:
                return;                // end of list

            case tc_mobj:
                PADSAVEP();
                mobj = Z_Malloc (sizeof(*mobj), PU_LEVEL, NULL);
                memcpy (mobj, save_p, sizeof(*mobj));
                save_p += sizeof(*mobj);
                mobj->state = &states[(int)mobj->state];
                mobj->target = NULL;
                if (mobj->player)
                {
                    mobj->player = &players[(int)mobj->player-1];
                    mobj->player->mo = mobj;
                }
                P_SetThingPosition (mobj);
                mobj->info = &mobjinfo[mobj->type];
                mobj->floorz = mobj->subsector->sector->floorheight;
                mobj->ceilingz = mobj->subsector->sector->ceilingheight;
                mobj->thinker.function.acp1 = (actionf_p1)P_MobjThinker;
                P_AddThinker (&mobj->thinker);
                break;

            default:
                I_Error ("Unknown tclass %i in savegame",tclass);
        }
    }
}

```

```

//
// P_ArchiveSpecials
//
enum
{
    tc_ceiling,
    tc_door,
    tc_floor,
    tc_plat,
    tc_flash,
    tc_strobe,
    tc_glow,
    tc_endspecials
} specials_e;


//
// Things to handle:
//
// T_MoveCeiling, (ceiling_t: sector_t * swizzle), - active list
// T_VerticalDoor, (vldoor_t: sector_t * swizzle),
// T_MoveFloor, (floormove_t: sector_t * swizzle),
// T_LightFlash, (lightflash_t: sector_t * swizzle),
// T_StrobeFlash, (strobe_t: sector_t *),
// T_Glow, (glow_t: sector_t *),
// T_PlatRaise, (plat_t: sector_t *), - active list
//
void P_ArchiveSpecials (void)
{
    thinker_t*      th;
    ceiling_t*       ceiling;
    vldoor_t*        door;
    floormove_t*     floor;
    plat_t*          plat;
    lightflash_t*    flash;
    strobe_t*         strobe;
    glow_t*          glow;
    int              i;

    // save off the current thinkers
    for (th = thinkercap.next ; th != &thinkercap ; th=th->next)
    {
        if (th->function.acv == (actionf_v)NULL)
        {
            for (i = 0; i < MAXCEILINGS;i++)
                if (activeceilings[i] == (ceiling_t *)th)
                    break;

            if (i<MAXCEILINGS)
            {
                *save_p++ = tc_ceiling;
                PADSAVEP();
                ceiling = (ceiling_t *)save_p;
                memcpy (ceiling, th, sizeof(*ceiling));
                save_p += sizeof(*ceiling);
                ceiling->sector = (sector_t *) (ceiling->sector - sectors);
            }
            continue;
        }

        if (th->function.acp1 == (actionf_p1)T_MoveCeiling)
        {
            *save_p++ = tc_ceiling;

```

```

    PADSAVEP();
    ceiling = (ceiling_t *)save_p;
    memcpy (ceiling, th, sizeof(*ceiling));
    save_p += sizeof(*ceiling);
    ceiling->sector = (sector_t *) (ceiling->sector - sectors);
    continue;
}

if (th->function.acp1 == (actionf_p1)T_VerticalDoor)
{
    *save_p++ = tc_door;
    PADSAVEP();
    door = (vldoor_t *)save_p;
    memcpy (door, th, sizeof(*door));
    save_p += sizeof(*door);
    door->sector = (sector_t *) (door->sector - sectors);
    continue;
}

if (th->function.acp1 == (actionf_p1)T_MoveFloor)
{
    *save_p++ = tc_floor;
    PADSAVEP();
    floor = (floormove_t *)save_p;
    memcpy (floor, th, sizeof(*floor));
    save_p += sizeof(*floor);
    floor->sector = (sector_t *) (floor->sector - sectors);
    continue;
}

if (th->function.acp1 == (actionf_p1)T_PlatformRaise)
{
    *save_p++ = tc_plat;
    PADSAVEP();
    plat = (plat_t *)save_p;
    memcpy (plat, th, sizeof(*plat));
    save_p += sizeof(*plat);
    plat->sector = (sector_t *) (plat->sector - sectors);
    continue;
}

if (th->function.acp1 == (actionf_p1)T_LightFlash)
{
    *save_p++ = tc_flash;
    PADSAVEP();
    flash = (lightflash_t *)save_p;
    memcpy (flash, th, sizeof(*flash));
    save_p += sizeof(*flash);
    flash->sector = (sector_t *) (flash->sector - sectors);
    continue;
}

if (th->function.acp1 == (actionf_p1)T_StrobeFlash)
{
    *save_p++ = tc_strobe;
    PADSAVEP();
    strobe = (strobe_t *)save_p;
    memcpy (strobe, th, sizeof(*strobe));
    save_p += sizeof(*strobe);
    strobe->sector = (sector_t *) (strobe->sector - sectors);
    continue;
}

if (th->function.acp1 == (actionf_p1)T_Glow)
{

```

```

        *save_p++ = tc_glow;
        PADSAVEP();
        glow = (glow_t *)save_p;
        memcpy (glow, th, sizeof(*glow));
        save_p += sizeof(*glow);
        glow->sector = (sector_t *) (glow->sector - sectors);
        continue;
    }
}

// add a terminating marker
*save_p++ = tc_endspecials;

}

//
// P_UnArchiveSpecials
//
void P_UnArchiveSpecials (void)
{
    byte          tclass;
    ceiling_t*    ceiling;
    vldoor_t*     door;
    floormove_t*  floor;
    plat_t*       plat;
    lightflash_t* flash;
    strobe_t*     strobe;
    glow_t*       glow;

    // read in saved thinkers
    while (1)
    {
        tclass = *save_p++;
        switch (tclass)
        {
            case tc_endspecials:
                return;          // end of list

            case tc_ceiling:
                PADSAVEP();
                ceiling = Z_Malloc (sizeof(*ceiling), PU_LEVEL, NULL);
                memcpy (ceiling, save_p, sizeof(*ceiling));
                save_p += sizeof(*ceiling);
                ceiling->sector = &sectors[(int)ceiling->sector];
                ceiling->sector->specialdata = ceiling;

                if (ceiling->thinker.function.acp1)
                    ceiling->thinker.function.acp1 = (actionf_p1)T_MoveCeiling;

                P_AddThinker (&ceiling->thinker);
                P_AddActiveCeiling(ceiling);
                break;

            case tc_door:
                PADSAVEP();
                door = Z_Malloc (sizeof(*door), PU_LEVEL, NULL);
                memcpy (door, save_p, sizeof(*door));
                save_p += sizeof(*door);
                door->sector = &sectors[(int)door->sector];
                door->sector->specialdata = door;
                door->thinker.function.acp1 = (actionf_p1)T_VerticalDoor;
                P_AddThinker (&door->thinker);
                break;
        }
    }
}

```

```

case tc_floor:
    PADSAVEP();
    floor = Z_Malloc (sizeof(*floor), PU_LEVEL, NULL);
    memcpy (floor, save_p, sizeof(*floor));
    save_p += sizeof(*floor);
    floor->sector = &sectors[(int)floor->sector];
    floor->sector->specialdata = floor;
    floor->thinker.function.acp1 = (actionf_p1)T_MoveFloor;
    P_AddThinker (&floor->thinker);
    break;

case tc_plat:
    PADSAVEP();
    plat = Z_Malloc (sizeof(*plat), PU_LEVEL, NULL);
    memcpy (plat, save_p, sizeof(*plat));
    save_p += sizeof(*plat);
    plat->sector = &sectors[(int)plat->sector];
    plat->sector->specialdata = plat;

    if (plat->thinker.function.acp1)
        plat->thinker.function.acp1 = (actionf_p1)T_PlatRaise;

    P_AddThinker (&plat->thinker);
    P_AddActivePlat(plat);
    break;

case tc_flash:
    PADSAVEP();
    flash = Z_Malloc (sizeof(*flash), PU_LEVEL, NULL);
    memcpy (flash, save_p, sizeof(*flash));
    save_p += sizeof(*flash);
    flash->sector = &sectors[(int)flash->sector];
    flash->thinker.function.acp1 = (actionf_p1)T_LightFlash;
    P_AddThinker (&flash->thinker);
    break;

case tc_strobe:
    PADSAVEP();
    strobe = Z_Malloc (sizeof(*strobe), PU_LEVEL, NULL);
    memcpy (strobe, save_p, sizeof(*strobe));
    save_p += sizeof(*strobe);
    strobe->sector = &sectors[(int)strobe->sector];
    strobe->thinker.function.acp1 = (actionf_p1)T_StrobeFlash;
    P_AddThinker (&strobe->thinker);
    break;

case tc_glow:
    PADSAVEP();
    glow = Z_Malloc (sizeof(*glow), PU_LEVEL, NULL);
    memcpy (glow, save_p, sizeof(*glow));
    save_p += sizeof(*glow);
    glow->sector = &sectors[(int)glow->sector];
    glow->thinker.function.acp1 = (actionf_p1)T_Glow;
    P_AddThinker (&glow->thinker);
    break;

default:
    I_Error ("P_UnarchiveSpecials:Unknown tclass %i "
            "in savegame",tclass);
}

}

}

```

## 9.19 p\_saveg.h

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Savegame I/O, archiving, persistence.
//-----

#ifndef __P_SAVEG__
#define __P_SAVEG__

#ifdef __GNUG__
#pragma interface
#endif

// Persistent storage/archiving.
// These are the load / save game routines.
void P_ArchivePlayers (void);
void P_UnArchivePlayers (void);
void P_ArchiveWorld (void);
void P_UnArchiveWorld (void);
void P_ArchiveThinkers (void);
void P_UnArchiveThinkers (void);
void P_ArchiveSpecials (void);
void P_UnArchiveSpecials (void);

extern byte*          save_p;

#endif
//-----
//
// $Log:$
//
//-----
```

## 9.20 p\_setup.c

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
```



```

//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Do all the WAD I/O, get map description,
//      set up initial state and misc. LUTs.
//
//-----

static const char
rcsid[] = "$Id: p_setup.c,v 1.5 1997/02/03 22:45:12 b1 Exp $";

#include <math.h>

#include "z_zone.h"

#include "m_swap.h"
#include "m_bbox.h"

#include "g_game.h"

#include "i_system.h"
#include "w_wad.h"

#include "doomdef.h"
#include "p_local.h"

#include "s_sound.h"

#include "doomstat.h"

void      P_SpawnMapThing (mapthing_t*      mthing);

//
// MAP related Lookup tables.
// Store VERTEXES, LINEDEFS, SIDEDEFS, etc.
//
int          numvertexes;
vertex_t*    vertexes;

int          numsegs;
seg_t*       segs;

int          numsectors;
sector_t*    sectors;

int          numsubsectors;
subsector_t* subsectors;

int          numnodes;
node_t*      nodes;

```

```

int                numlines;
line_t*            lines;

int                numsides;
side_t*            sides;

// BLOCKMAP
// Created from axis aligned bounding box
// of the map, a rectangular array of
// blocks of size ...
// Used to speed up collision detection
// by spatial subdivision in 2D.
//
// Blockmap size.
int                bmapwidth;
int                bmapheight;        // size in mapblocks
short*             blockmap;          // int for larger maps
// offsets in blockmap are from here
short*             blockmaplump;
// origin of block map
fixed_t            bmaporgx;
fixed_t            bmaporgy;
// for thing chains
mobj_t**           blocklinks;

// REJECT
// For fast sight rejection.
// Speeds up enemy AI by skipping detailed
// LineOf Sight calculation.
// Without special effect, this could be
// used as a PVS lookup as well.
//
byte*              rejectmatrix;

// Maintain single and multi player starting spots.
#define MAX_DEATHMATCH_STARTS    10

mapthing_t         deathmatchstarts[MAX_DEATHMATCH_STARTS];
mapthing_t*        deathmatch_p;
mapthing_t         playerstarts[MAXPLAYERS];

//
// P_LoadVertexes
//
void P_LoadVertexes (int lump)
{
    byte*            data;
    int              i;
    mapvertex_t*     ml;
    vertex_t*        li;

    // Determine number of lumps:
    // total lump length / vertex record length.
    numvertexes = W_LumpLength (lump) / sizeof(mapvertex_t);

    // Allocate zone memory for buffer.
    vertexes = Z_Malloc (numvertexes*sizeof(vertex_t),PU_LEVEL,0);

```

```

// Load data into cache.
data = W_CacheLumpNum (lump,PU_STATIC);

ml = (mapvertex_t *)data;
li = vertexes;

// Copy and convert vertex coordinates,
// internal representation as fixed.
for (i=0 ; i<numvertexes ; i++, li++, ml++)
{
    li->x = SHORT(ml->x)<<FRACBITS;
    li->y = SHORT(ml->y)<<FRACBITS;
}

// Free buffer memory.
Z_Free (data);
}

//
// P_LoadSegs
//
void P_LoadSegs (int lump)
{
    byte*          data;
    int             i;
    mapseg_t*      ml;
    seg_t*         li;
    line_t*        ldef;
    int            linedef;
    int            side;

    numsegs = W_LumpLength (lump) / sizeof(mapseg_t);
    segs = Z_Malloc (numsegs*sizeof(seg_t),PU_LEVEL,0);
    memset (segs, 0, numsegs*sizeof(seg_t));
    data = W_CacheLumpNum (lump,PU_STATIC);

    ml = (mapseg_t *)data;
    li = segs;
    for (i=0 ; i<numsegs ; i++, li++, ml++)
    {
        li->v1 = &vertexes[SHORT(ml->v1)];
        li->v2 = &vertexes[SHORT(ml->v2)];

        li->angle = (SHORT(ml->angle))<<16;
        li->offset = (SHORT(ml->offset))<<16;
        linedef = SHORT(ml->linedef);
        ldef = &lines[linedef];
        li->linedef = ldef;
        side = SHORT(ml->side);
        li->sidedef = &sides[ldef->sidenum[side]];
        li->frontsector = sides[ldef->sidenum[side]].sector;
        if (ldef-> flags & ML_TWOSIDED)
            li->backsector = sides[ldef->sidenum[side^1]].sector;
        else
            li->backsector = 0;
    }

    Z_Free (data);
}

//
// P_LoadSubsectors

```

```

//
void P_LoadSubsectors (int lump)
{
    byte*          data;
    int            i;
    mapsubsector_t* ms;
    subsector_t*   ss;

    numsubsectors = W_LumpLength (lump) / sizeof(mapsubsector_t);
    subsectors = Z_Malloc (numsubsectors*sizeof(subsector_t),PU_LEVEL,0);
    data = W_CacheLumpNum (lump,PU_STATIC);

    ms = (mapsubsector_t *)data;
    memset (subsectors,0, numsubsectors*sizeof(subsector_t));
    ss = subsectors;

    for (i=0 ; i<numsubsectors ; i++, ss++, ms++)
    {
        ss->numlines = SHORT(ms->numsegs);
        ss->firstline = SHORT(ms->firstseg);
    }

    Z_Free (data);
}

```

```

//
// P_LoadSectors
//
void P_LoadSectors (int lump)
{
    byte*          data;
    int            i;
    mapsector_t*   ms;
    sector_t*      ss;

    numsectors = W_LumpLength (lump) / sizeof(mapsector_t);
    sectors = Z_Malloc (numsectors*sizeof(sector_t),PU_LEVEL,0);
    memset (sectors, 0, numsectors*sizeof(sector_t));
    data = W_CacheLumpNum (lump,PU_STATIC);

    ms = (mapsector_t *)data;
    ss = sectors;
    for (i=0 ; i<numsectors ; i++, ss++, ms++)
    {
        ss->floorheight = SHORT(ms->floorheight)<<FRACBITS;
        ss->ceilingheight = SHORT(ms->ceilingheight)<<FRACBITS;
        ss->floorpic = R_FlatNumForName(ms->floorpic);
        ss->ceilingpic = R_FlatNumForName(ms->ceilingpic);
        ss->lightlevel = SHORT(ms->lightlevel);
        ss->special = SHORT(ms->special);
        ss->tag = SHORT(ms->tag);
        ss->thinglist = NULL;
    }

    Z_Free (data);
}

```

```

//
// P_LoadNodes
//
void P_LoadNodes (int lump)
{

```

```

byte*      data;
int        i;
int        j;
int        k;
mapnode_t* mn;
node_t*    no;

numnodes = W_LumpLength (lump) / sizeof(mapnode_t);
nodes = Z_Malloc (numnodes*sizeof(node_t),PU_LEVEL,0);
data = W_CacheLumpNum (lump,PU_STATIC);

mn = (mapnode_t *)data;
no = nodes;

for (i=0 ; i<numnodes ; i++, no++, mn++)
{
    no->x = SHORT(mn->x)<<FRACBITS;
    no->y = SHORT(mn->y)<<FRACBITS;
    no->dx = SHORT(mn->dx)<<FRACBITS;
    no->dy = SHORT(mn->dy)<<FRACBITS;
    for (j=0 ; j<2 ; j++)
    {
        no->children[j] = SHORT(mn->children[j]);
        for (k=0 ; k<4 ; k++)
            no->bbox[j][k] = SHORT(mn->bbox[j][k])<<FRACBITS;
    }
}

Z_Free (data);
}

```

```

//
// P_LoadThings
//
void P_LoadThings (int lump)
{
    byte*      data;
    int        i;
    mapthing_t* mt;
    int        numthings;
    boolean    spawn;

    data = W_CacheLumpNum (lump,PU_STATIC);
    numthings = W_LumpLength (lump) / sizeof(mapthing_t);

    mt = (mapthing_t *)data;
    for (i=0 ; i<numthings ; i++, mt++)
    {
        spawn = true;

        // Do not spawn cool, new monsters if !commercial
        if ( gamemode != commercial)
        {
            switch(mt->type)
            {
                case 68:      // Arachnotron
                case 64:      // Archvile
                case 88:      // Boss Brain
                case 89:      // Boss Shooter
                case 69:      // Hell Knight
                case 67:      // Mancubus
                case 71:      // Pain Elemental
                case 65:      // Former Human Commando
                case 66:      // Revenant

```

```

        case 84:          // Wolf SS
            spawn = false;
            break;
    }
}
if (spawn == false)
    break;

// Do spawn all other stuff.
mt->x = SHORT(mt->x);
mt->y = SHORT(mt->y);
mt->angle = SHORT(mt->angle);
mt->type = SHORT(mt->type);
mt->options = SHORT(mt->options);

P_SpawnMapThing (mt);
}

Z_Free (data);
}

//
// P_LoadLineDefs
// Also counts secret lines for intermissions.
//
void P_LoadLineDefs (int lump)
{
    byte*          data;
    int            i;
    maplinedef_t*  mld;
    line_t*        ld;
    vertex_t*      v1;
    vertex_t*      v2;

    numlines = W_LumpLength (lump) / sizeof(maplinedef_t);
    lines = Z_Malloc (numlines*sizeof(line_t),PU_LEVEL,0);
    memset (lines, 0, numlines*sizeof(line_t));
    data = W_CacheLumpNum (lump,PU_STATIC);

    mld = (maplinedef_t *)data;
    ld = lines;
    for (i=0 ; i<numlines ; i++, mld++, ld++)
    {
        ld->flags = SHORT(mld->flags);
        ld->special = SHORT(mld->special);
        ld->tag = SHORT(mld->tag);
        v1 = ld->v1 = &vertexes[SHORT(mld->v1)];
        v2 = ld->v2 = &vertexes[SHORT(mld->v2)];
        ld->dx = v2->x - v1->x;
        ld->dy = v2->y - v1->y;

        if (!ld->dx)
            ld->slopetype = ST_VERTICAL;
        else if (!ld->dy)
            ld->slopetype = ST_HORIZONTAL;
        else
        {
            if (FixedDiv (ld->dy , ld->dx) > 0)
                ld->slopetype = ST_POSITIVE;
            else
                ld->slopetype = ST_NEGATIVE;
        }

        if (v1->x < v2->x)

```

```

    {
        ld->bbox[BOXLEFT] = v1->x;
        ld->bbox[BOXRIGHT] = v2->x;
    }
    else
    {
        ld->bbox[BOXLEFT] = v2->x;
        ld->bbox[BOXRIGHT] = v1->x;
    }

    if (v1->y < v2->y)
    {
        ld->bbox[BOXBOTTOM] = v1->y;
        ld->bbox[BOXTOP] = v2->y;
    }
    else
    {
        ld->bbox[BOXBOTTOM] = v2->y;
        ld->bbox[BOXTOP] = v1->y;
    }

    ld->sidenum[0] = SHORT(mld->sidenum[0]);
    ld->sidenum[1] = SHORT(mld->sidenum[1]);

    if (ld->sidenum[0] != -1)
        ld->frontsector = sides[ld->sidenum[0]].sector;
    else
        ld->frontsector = 0;

    if (ld->sidenum[1] != -1)
        ld->backsector = sides[ld->sidenum[1]].sector;
    else
        ld->backsector = 0;
}

Z_Free (data);
}

//
// P_LoadSideDefs
//
void P_LoadSideDefs (int lump)
{
    byte*          data;
    int            i;
    mapsidedef_t*  msd;
    side_t*        sd;

    numsides = W_LumpLength (lump) / sizeof(mapsidedef_t);
    sides = Z_Malloc (numsides*sizeof(side_t),PU_LEVEL,0);
    memset (sides, 0, numsides*sizeof(side_t));
    data = W_CacheLumpNum (lump,PU_STATIC);

    msd = (mapsidedef_t *)data;
    sd = sides;
    for (i=0 ; i<numsides ; i++, msd++, sd++)
    {
        sd->textureoffset = SHORT(msd->textureoffset)<<FRACBITS;
        sd->rowoffset = SHORT(msd->rowoffset)<<FRACBITS;
        sd->toptexture = R_TextureNumForName(msd->toptexture);
        sd->bottomtexture = R_TextureNumForName(msd->bottomtexture);
        sd->midtexture = R_TextureNumForName(msd->midtexture);
        sd->sector = &sectors[SHORT(msd->sector)];
    }
}

```

```

    Z_Free (data);
}

//
// P_LoadBlockMap
//
void P_LoadBlockMap (int lump)
{
    int            i;
    int            count;

    blockmaplump = W_CacheLumpNum (lump,PU_LEVEL);
    blockmap = blockmaplump+4;
    count = W_LumpLength (lump)/2;

    for (i=0 ; i<count ; i++)
        blockmaplump[i] = SHORT(blockmaplump[i]);

    bmaporgx = blockmaplump[0]<<FRACBITS;
    bmaporgy = blockmaplump[1]<<FRACBITS;
    bmapwidth = blockmaplump[2];
    bmapheight = blockmaplump[3];

    // clear out mobj chains
    count = sizeof(*blocklinks)* bmapwidth*bmapheight;
    blocklinks = Z_Malloc (count,PU_LEVEL, 0);
    memset (blocklinks, 0, count);
}

//
// P_GroupLines
// Builds sector line lists and subsector sector numbers.
// Finds block bounding boxes for sectors.
//
void P_GroupLines (void)
{
    line_t**        linebuffer;
    int             i;
    int             j;
    int             total;
    line_t*         li;
    sector_t*       sector;
    subsector_t*    ss;
    seg_t*          seg;
    fixed_t         bbox[4];
    int             block;

    // look up sector number for each subsector
    ss = subsectors;
    for (i=0 ; i<numsubsectors ; i++, ss++)
    {
        seg = &segs[ss->firstline];
        ss->sector = seg->sidedef->sector;
    }

    // count number of lines in each sector
    li = lines;
    total = 0;
    for (i=0 ; i<numlines ; i++, li++)
    {
        total++;
    }
}

```



```

    li->frontsector->linecount++;

    if (li->backsector && li->backsector != li->frontsector)
    {
        li->backsector->linecount++;
        total++;
    }
}

// build line tables for each sector
linebuffer = Z_Malloc (total*4, PU_LEVEL, 0);
sector = sectors;
for (i=0 ; i<numsectors ; i++, sector++)
{
    M_ClearBox (bbox);
    sector->lines = linebuffer;
    li = lines;
    for (j=0 ; j<numlines ; j++, li++)
    {
        if (li->frontsector == sector || li->backsector == sector)
        {
            *linebuffer++ = li;
            M_AddToBox (bbox, li->v1->x, li->v1->y);
            M_AddToBox (bbox, li->v2->x, li->v2->y);
        }
    }
    if (linebuffer - sector->lines != sector->linecount)
        I_Error ("P_GroupLines: miscounted");

    // set the degenmobj_t to the middle of the bounding box
    sector->soundorg.x = (bbox[BOXRIGHT]+bbox[BOXLEFT])/2;
    sector->soundorg.y = (bbox[BOXTOP]+bbox[BOXBOTTOM])/2;

    // adjust bounding box to map blocks
    block = (bbox[BOXTOP]-bmaporgy+MAXRADIUS)>>MAPBLOCKSHIFT;
    block = block >= bmapheight ? bmapheight-1 : block;
    sector->blockbox[BOXTOP]=block;

    block = (bbox[BOXBOTTOM]-bmaporgy-MAXRADIUS)>>MAPBLOCKSHIFT;
    block = block < 0 ? 0 : block;
    sector->blockbox[BOXBOTTOM]=block;

    block = (bbox[BOXRIGHT]-bmaporgx+MAXRADIUS)>>MAPBLOCKSHIFT;
    block = block >= bmapwidth ? bmapwidth-1 : block;
    sector->blockbox[BOXRIGHT]=block;

    block = (bbox[BOXLEFT]-bmaporgx-MAXRADIUS)>>MAPBLOCKSHIFT;
    block = block < 0 ? 0 : block;
    sector->blockbox[BOXLEFT]=block;
}

}

//
// P_SetupLevel
//
void
P_SetupLevel
( int          episode,
  int          map,
  int          playermask,
  skill_t      skill)
{
    int          i;

```

```

char      lumpname[9];
int       lumpnum;

totalkills = totalitems = totalsecret = wminfo.maxfrags = 0;
wminfo.partime = 180;
for (i=0 ; i<MAXPLAYERS ; i++)
{
    players[i].killcount = players[i].secretcount
        = players[i].itemcount = 0;
}

// Initial height of PointOfView
// will be set by player think.
players[consoleplayer].viewz = 1;

// Make sure all sounds are stopped before Z_FreeTags.
S_Start ();

#if 0 // UNUSED
    if (debugfile)
    {
        Z_FreeTags (PU_LEVEL, MAXINT);
        Z_FileDumpHeap (debugfile);
    }
    else
#endif
    Z_FreeTags (PU_LEVEL, PU_PURGELEVEL-1);

// UNUSED W_Profile ();
P_InitThinkers ();

// if working with a development map, reload it
W_Reload ();

// find map name
if ( gamemode == commercial)
{
    if (map<10)
        sprintf (lumpname,"map0%i", map);
    else
        sprintf (lumpname,"map%i", map);
}
else
{
    lumpname[0] = 'E';
    lumpname[1] = '0' + episode;
    lumpname[2] = 'M';
    lumpname[3] = '0' + map;
    lumpname[4] = 0;
}

lumpnum = W_GetNumForName (lumpname);

leveltime = 0;

// note: most of this ordering is important
P_LoadBlockMap (lumpnum+ML_BLOCKMAP);
P_LoadVertexes (lumpnum+ML_VERTEXES);
P_LoadSectors (lumpnum+ML_SECTORS);
P_LoadSideDefs (lumpnum+ML_SIDEDEFS);

P_LoadLineDefs (lumpnum+ML_LINEDEFS);
P_LoadSubsectors (lumpnum+ML_SSECTORS);

```

```

P_LoadNodes (lumpnum+ML_NODES);
P_LoadSegs (lumpnum+ML_SEGS);

rejectmatrix = W_CacheLumpNum (lumpnum+ML_REJECT,PU_LEVEL);
P_GroupLines ();

bodyqueslot = 0;
deathmatch_p = deathmatchstarts;
P_LoadThings (lumpnum+ML_THINGS);

// if deathmatch, randomly spawn the active players
if (deathmatch)
{
    for (i=0 ; i<MAXPLAYERS ; i++)
        if (playeringame[i])
        {
            players[i].mo = NULL;
            G_DeathMatchSpawnPlayer (i);
        }
}

// clear special respawning que
iquehead = iquetail = 0;

// set up world state
P_SpawnSpecials ();

// build subsector connect matrix
//      UNUSED P_ConnectSubsectors ();

// preload graphics
if (precache)
    R_PrecacheLevel ();

//printf ("free memory: 0x%x\n", Z_FreeMemory());
}

//
// P_Init
//
void P_Init (void)
{
    P_InitSwitchList ();
    P_InitPicAnims ();
    R_InitSprites (sprnames);
}

```

## 9.21 p\_setup.h

```

// Emacs style mode select    -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License

```

```

// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
// Setup a game, startup stuff.
//
//-----

#ifndef __P_SETUP__
#define __P_SETUP__

#ifdef __GNUG__
#pragma interface
#endif

// NOT called by W_Ticker. Fixme.
void
P_SetupLevel
( int          episode,
  int          map,
  int          playermask,
  skill_t      skill);

// Called by startup code.
void P_Init (void);

#endif
//-----
//
// $Log:$
//
//-----

```

## 9.22 p\_sight.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
// LineOfSight/Visibility checks, uses REJECT Lookup Table.
//

```

```

//-----

static const char
rcsid[] = "$Id: p_sight.c,v 1.3 1997/01/28 22:08:28 b1 Exp $";

#include "doomdef.h"

#include "i_system.h"
#include "p_local.h"

// State.
#include "r_state.h"

//
// P_CheckSight
//
fixed_t      sightzstart;           // eye z of looker
fixed_t      topslope;
fixed_t      bottomslope;          // slopes to top and bottom of target

divline_t    strace;                // from t1 to t2
fixed_t      t2x;
fixed_t      t2y;

int           sightcounts[2];

//
// P_DivlineSide
// Returns side 0 (front), 1 (back), or 2 (on).
//
int
P_DivlineSide
( fixed_t      x,
  fixed_t      y,
  divline_t*   node )
{
    fixed_t      dx;
    fixed_t      dy;
    fixed_t      left;
    fixed_t      right;

    if (!node->dx)
    {
        if (x==node->x)
            return 2;

        if (x <= node->x)
            return node->dy > 0;

        return node->dy < 0;
    }

    if (!node->dy)
    {
        if (x==node->y)
            return 2;

        if (y <= node->y)
            return node->dx < 0;

        return node->dx > 0;
    }
}

```

```

    dx = (x - node->x);
    dy = (y - node->y);

    left = (node->dy>>FRACBITS) * (dx>>FRACBITS);
    right = (dy>>FRACBITS) * (node->dx>>FRACBITS);

    if (right < left)
        return 0;          // front side

    if (left == right)
        return 2;
    return 1;              // back side
}

//
// P_InterceptVector2
// Returns the fractional intercept point
// along the first divline.
// This is only called by the addthings and addlines traversers.
//
fixed_t
P_InterceptVector2
( divline_t*      v2,
  divline_t*      v1 )
{
    fixed_t      frac;
    fixed_t      num;
    fixed_t      den;

    den = FixedMul (v1->dy>>8,v2->dx) - FixedMul(v1->dx>>8,v2->dy);

    if (den == 0)
        return 0;
    //      I_Error ("P_InterceptVector: parallel");

    num = FixedMul ( (v1->x - v2->x)>>8 ,v1->dy) +
        FixedMul ( (v2->y - v1->y)>>8 , v1->dx);
    frac = FixedDiv (num , den);

    return frac;
}

//
// P_CrossSubsector
// Returns true
// if strace crosses the given subsector successfully.
//
boolean P_CrossSubsector (int num)
{
    seg_t*        seg;
    line_t*        line;
    int            s1;
    int            s2;
    int            count;
    subsector_t*   sub;
    sector_t*      front;
    sector_t*      back;
    fixed_t        opentop;
    fixed_t        openbottom;
    divline_t      div1;
    vertex_t*      v1;
    vertex_t*      v2;
    fixed_t        frac;
    fixed_t        slope;

```

```

#ifdef RANGECHECK
    if (num>=numsubsectors)
        I_Error ("P_CrossSubsector: ss %i with numss = %i",
            num,
            numsubsectors);
#endif

    sub = &subsectors[num];

    // check lines
    count = sub->numlines;
    seg = &segs[sub->firstline];

    for ( ; count ; seg++, count--)
    {
        line = seg->linedef;

        // already checked other side?
        if (line->validcount == validcount)
            continue;

        line->validcount = validcount;

        v1 = line->v1;
        v2 = line->v2;
        s1 = P_DivlineSide (v1->x,v1->y, &strace);
        s2 = P_DivlineSide (v2->x, v2->y, &strace);

        // line isn't crossed?
        if (s1 == s2)
            continue;

        divl.x = v1->x;
        divl.y = v1->y;
        divl.dx = v2->x - v1->x;
        divl.dy = v2->y - v1->y;
        s1 = P_DivlineSide (strace.x, strace.y, &divl);
        s2 = P_DivlineSide (t2x, t2y, &divl);

        // line isn't crossed?
        if (s1 == s2)
            continue;

        // stop because it is not two sided anyway
        // might do this after updating validcount?
        if ( !(line->flags & ML_TWOSIDED) )
            return false;

        // crosses a two sided line
        front = seg->frontsector;
        back = seg->backsector;

        // no wall to block sight with?
        if (front->floorheight == back->floorheight
            && front->ceilingheight == back->ceilingheight)
            continue;

        // possible occluder
        // because of ceiling height differences
        if (front->ceilingheight < back->ceilingheight)
            opentop = front->ceilingheight;
        else
            opentop = back->ceilingheight;
    }

```

```

// because of ceiling height differences
if (front->floorheight > back->floorheight)
    openbottom = front->floorheight;
else
    openbottom = back->floorheight;

// quick test for totally closed doors
if (openbottom >= opentop)
    return false;          // stop

frac = P_InterceptVector2 (&strace, &divl);

if (front->floorheight != back->floorheight)
{
    slope = FixedDiv (openbottom - sightzstart , frac);
    if (slope > bottomslope)
        bottomslope = slope;
}

if (front->ceilingheight != back->ceilingheight)
{
    slope = FixedDiv (opentop - sightzstart , frac);
    if (slope < topslope)
        topslope = slope;
}

if (topslope <= bottomslope)
    return false;          // stop
}
// passed the subsector ok
return true;
}

```

```

//
// P_CrossBSPNode
// Returns true
// if strace crosses the given node successfully.
//
boolean P_CrossBSPNode (int bspnum)
{
    node_t*      bsp;
    int          side;

    if (bspnum & NF_SUBSECTOR)
    {
        if (bspnum == -1)
            return P_CrossSubsector (0);
        else
            return P_CrossSubsector (bspnum&(~NF_SUBSECTOR));
    }

    bsp = &nodes[bspnum];

    // decide which side the start point is on
    side = P_DivlineSide (strace.x, strace.y, (divline_t *)bsp);
    if (side == 2)
        side = 0;          // an "on" should cross both sides

    // cross the starting side
    if (!P_CrossBSPNode (bsp->children[side]))
        return false;

    // the partition plane is crossed here

```



```

    if (side == P_DivlineSide (t2x, t2y,(divline_t *)bsp))
    {
        // the line doesn't touch the other side
        return true;
    }

    // cross the ending side
    return P_CrossBSPNode (bsp->children[side^1]);
}

//
// P_CheckSight
// Returns true
// if a straight line between t1 and t2 is unobstructed.
// Uses REJECT.
//
boolean
P_CheckSight
( mobj_t*      t1,
  mobj_t*      t2 )
{
    int          s1;
    int          s2;
    int          pnum;
    int          bytenum;
    int          bitnum;

    // First check for trivial rejection.

    // Determine subsector entries in REJECT table.
    s1 = (t1->subsector->sector - sectors);
    s2 = (t2->subsector->sector - sectors);
    pnum = s1*numsectors + s2;
    bytenum = pnum>>3;
    bitnum = 1 << (pnum&7);

    // Check in REJECT table.
    if (rejectmatrix[bytenum]&bitnum)
    {
        sightcounts[0]++;

        // can't possibly be connected
        return false;
    }

    // An unobstructed LOS is possible.
    // Now look from eyes of t1 to any part of t2.
    sightcounts[1]++;

    validcount++;

    sightzstart = t1->z + t1->height - (t1->height>>2);
    topslope = (t2->z+t2->height) - sightzstart;
    bottomslope = (t2->z) - sightzstart;

    strace.x = t1->x;
    strace.y = t1->y;
    t2x = t2->x;
    t2y = t2->y;
    strace.dx = t2->x - t1->x;
    strace.dy = t2->y - t1->y;

    // the head node is the last node output
    return P_CrossBSPNode (numnodes-1);
}

```

```
}
```

## 9.23 p\_spec.c

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Implements special effects:
//     Texture animation, height or lighting changes
//     according to adjacent sectors, respective
//     utility functions, etc.
//     Line Tag handling. Line and Sector triggers.
//
//-----

static const char
rcsid[] = "$Id: p_spec.c,v 1.6 1997/02/03 22:45:12 b1 Exp $";

#include <stdlib.h>

#include "doomdef.h"
#include "doomstat.h"

#include "i_system.h"
#include "z_zone.h"
#include "m_argv.h"
#include "m_random.h"
#include "w_wad.h"

#include "r_local.h"
#include "p_local.h"

#include "g_game.h"

#include "s_sound.h"

// State.
#include "r_state.h"

// Data.
#include "sounds.h"

//
// Animating textures and planes
// There is another anim_t used in wi_stuff, unrelated.
```

```

//
typedef struct
{
    boolean    istexture;
    int        picnum;
    int        basepic;
    int        numpics;
    int        speed;

} anim_t;

//
//      source animation definition
//
typedef struct
{
    boolean    istexture;           // if false, it is a flat
    char        endname[9];
    char        startname[9];
    int        speed;
} animdef_t;

#define MAXANIMS          32

extern anim_t      anims[MAXANIMS];
extern anim_t*     lastanim;

//
// P_InitPicAnims
//

// Floor/ceiling animation sequences,
// defined by first and last frame,
// i.e. the flat (64x64 tile) name to
// be used.
// The full animation sequence is given
// using all the flats between the start
// and end entry, in the order found in
// the WAD file.
//
animdef_t          animdefs[] =
{
    {false,        "NUKAGE3",        "NUKAGE1",        8},
    {false,        "FWATER4",        "FWATER1",        8},
    {false,        "SWATER4",        "SWATER1",        8},
    {false,        "LAVA4",          "LAVA1",          8},
    {false,        "BLOOD3",         "BLOOD1",         8},

    // DOOM II flat animations.
    {false,        "RROCK08",        "RROCK05",        8},
    {false,        "SLIME04",        "SLIME01",        8},
    {false,        "SLIME08",        "SLIME05",        8},
    {false,        "SLIME12",        "SLIME09",        8},

    {true,         "BLODGR4",        "BLODGR1",        8},
    {true,         "SLADRIP3",        "SLADRIP1",        8},

    {true,         "BLODRIP4",        "BLODRIP1",        8},
    {true,         "FIREWALL",        "FIREWALA",        8},
    {true,         "GSTFONT3",        "GSTFONT1",        8},
    {true,         "FIRELAVA",        "FIRELAV3",        8},
    {true,         "FIREMAG3",        "FIREMAG1",        8},
    {true,         "FIREBLU2",        "FIREBLU1",        8},

```

```

    {true,      "ROCKRED3",      "ROCKRED1",      8},

    {true,      "BFALL4",      "BFALL1",      8},
    {true,      "SFALL4",      "SFALL1",      8},
    {true,      "WFALL4",      "WFALL1",      8},
    {true,      "DBRAIN4",      "DBRAIN1",      8},

    {-1}
};

anim_t      anims[MAXANIMS];
anim_t*     lastanim;

//
//      Animating line specials
//
#define MAXLINEANIMS      64

extern short      numlinespecials;
extern line_t*    linespeciallist[MAXLINEANIMS];

void P_InitPicAnims (void)
{
    int      i;

    //      Init animation
    lastanim = anims;
    for (i=0 ; animdefs[i].istexture != -1 ; i++)
    {
        if (animdefs[i].istexture)
        {
            // different episode ?
            if (R_CheckTextureNumForName(animdefs[i].startname) == -1)
                continue;

            lastanim->picnum = R_TextureNumForName (animdefs[i].endname);
            lastanim->basepic = R_TextureNumForName (animdefs[i].startname);
        }
        else
        {
            if (W_CheckNumForName(animdefs[i].startname) == -1)
                continue;

            lastanim->picnum = R_FlatNumForName (animdefs[i].endname);
            lastanim->basepic = R_FlatNumForName (animdefs[i].startname);
        }

        lastanim->istexture = animdefs[i].istexture;
        lastanim->numpics = lastanim->picnum - lastanim->basepic + 1;

        if (lastanim->numpics < 2)
            I_Error ("P_InitPicAnims: bad cycle from %s to %s",
                    animdefs[i].startname,
                    animdefs[i].endname);

        lastanim->speed = animdefs[i].speed;
        lastanim++;
    }
}

```

```

//
// UTILITIES
//

//
// getSide()
// Will return a side_t*
// given the number of the current sector,
// the line number, and the side (0/1) that you want.
//
side_t*
getSide
( int          currentSector,
  int          line,
  int          side )
{
    return &sides[ (sectors[currentSector].lines[line])->sidenum[side] ];
}

//
// getSector()
// Will return a sector_t*
// given the number of the current sector,
// the line number and the side (0/1) that you want.
//
sector_t*
getSector
( int          currentSector,
  int          line,
  int          side )
{
    return sides[ (sectors[currentSector].lines[line])->sidenum[side] ].sector;
}

//
// twoSided()
// Given the sector number and the line number,
// it will tell you whether the line is two-sided or not.
//
int
twoSided
( int          sector,
  int          line )
{
    return (sectors[sector].lines[line])->flags & ML_TWOSIDED;
}

//
// getNextSector()
// Return sector_t * of sector next to current.
// NULL if not two-sided line
//
sector_t*
getNextSector
( line_t*      line,
  sector_t*    sec )

```

```

{
    if (!(line->flags & ML_TWOSIDED))
        return NULL;

    if (line->frontsector == sec)
        return line->backsector;

    return line->frontsector;
}

//
// P_FindLowestFloorSurrounding()
// FIND LOWEST FLOOR HEIGHT IN SURROUNDING SECTORS
//
fixed_t      P_FindLowestFloorSurrounding(sector_t* sec)
{
    int                i;
    line_t*            check;
    sector_t*          other;
    fixed_t            floor = sec->floorheight;

    for (i=0 ;i < sec->linecount ; i++)
    {
        check = sec->lines[i];
        other = getNextSector(check,sec);

        if (!other)
            continue;

        if (other->floorheight < floor)
            floor = other->floorheight;
    }
    return floor;
}

//
// P_FindHighestFloorSurrounding()
// FIND HIGHEST FLOOR HEIGHT IN SURROUNDING SECTORS
//
fixed_t      P_FindHighestFloorSurrounding(sector_t *sec)
{
    int                i;
    line_t*            check;
    sector_t*          other;
    fixed_t            floor = -500*FRACUNIT;

    for (i=0 ;i < sec->linecount ; i++)
    {
        check = sec->lines[i];
        other = getNextSector(check,sec);

        if (!other)
            continue;

        if (other->floorheight > floor)
            floor = other->floorheight;
    }
    return floor;
}

```

```

//
// P_FindNextHighestFloor
// FIND NEXT HIGHEST FLOOR IN SURROUNDING SECTORS
// Note: this should be doable w/o a fixed array.

// 20 adjoining sectors max!
#define MAX_ADJOINING_SECTORS          20

fixed_t
P_FindNextHighestFloor
( sector_t*          sec,
  int                currentheight )
{
    int              i;
    int              h;
    int              min;
    line_t*          check;
    sector_t*         other;
    fixed_t           height = currentheight;

    fixed_t           heightlist[MAX_ADJOINING_SECTORS];

    for (i=0, h=0 ; i < sec->linecount ; i++)
    {
        check = sec->lines[i];
        other = getNextSector(check,sec);

        if (!other)
            continue;

        if (other->floorheight > height)
            heightlist[h++] = other->floorheight;

        // Check for overflow. Exit.
        if ( h >= MAX_ADJOINING_SECTORS )
        {
            fprintf( stderr,
                    "Sector with more than 20 adjoining sectors\n" );
            break;
        }
    }

    // Find lowest height in list
    if (!h)
        return currentheight;

    min = heightlist[0];

    // Range checking?
    for (i = 1; i < h; i++)
        if (heightlist[i] < min)
            min = heightlist[i];

    return min;
}

//
// FIND LOWEST CEILING IN THE SURROUNDING SECTORS
//
fixed_t
P_FindLowestCeilingSurrounding(sector_t* sec)
{

```

```

int                i;
line_t*            check;
sector_t*          other;
fixed_t            height = MAXINT;

for (i=0 ;i < sec->linecount ; i++)
{
    check = sec->lines[i];
    other = getNextSector(check,sec);

    if (!other)
        continue;

    if (other->ceilingheight < height)
        height = other->ceilingheight;
}
return height;
}

//
// FIND HIGHEST CEILING IN THE SURROUNDING SECTORS
//
fixed_t            P_FindHighestCeilingSurrounding(sector_t* sec)
{
    int                i;
    line_t*            check;
    sector_t*          other;
    fixed_t            height = 0;

    for (i=0 ;i < sec->linecount ; i++)
    {
        check = sec->lines[i];
        other = getNextSector(check,sec);

        if (!other)
            continue;

        if (other->ceilingheight > height)
            height = other->ceilingheight;
    }
    return height;
}

//
// RETURN NEXT SECTOR # THAT LINE TAG REFERS TO
//
int
P_FindSectorFromLineTag
( line_t*            line,
  int                start )
{
    int                i;

    for (i=start+1;i<numsectors;i++)
        if (sectors[i].tag == line->tag)
            return i;

    return -1;
}

```



```

//
// Find minimum light from an adjacent sector
//
int
P_FindMinSurroundingLight
( sector_t*      sector,
  int            max )
{
    int          i;
    int          min;
    line_t*      line;
    sector_t*    check;

    min = max;
    for (i=0 ; i < sector->linecount ; i++)
    {
        line = sector->lines[i];
        check = getNextSector(line,sector);

        if (!check)
            continue;

        if (check->lightlevel < min)
            min = check->lightlevel;
    }
    return min;
}

//
// EVENTS
// Events are operations triggered by using, crossing,
// or shooting special lines, or by timed thinkers.
//

//
// P_CrossSpecialLine - TRIGGER
// Called every time a thing origin is about
// to cross a line with a non 0 special.
//
void
P_CrossSpecialLine
( int          linenum,
  int          side,
  mobj_t*      thing )
{
    line_t*     line;
    int         ok;

    line = &lines[linenum];

    //          Triggers that other things can activate
    if (!thing->player)
    {
        // Things that should NOT trigger specials...
        switch(thing->type)
        {
            case MT_ROCKET:
            case MT_PLASMA:
            case MT_BFG:
            case MT_TROOPSHOT:
            case MT_HEADSHOT:
            case MT_BRUISERSHOT:

```

```

        return;
        break;

    default: break;
}

ok = 0;
switch(line->special)
{
    case 39:          // TELEPORT TRIGGER
    case 97:          // TELEPORT RETRIGGER
    case 125:         // TELEPORT MONSTERONLY TRIGGER
    case 126:         // TELEPORT MONSTERONLY RETRIGGER
    case 4:           // RAISE DOOR
    case 10:          // PLAT DOWN-WAIT-UP-STAY TRIGGER
    case 88:          // PLAT DOWN-WAIT-UP-STAY RETRIGGER
        ok = 1;
        break;
}
if (!ok)
    return;
}

```

```

// Note: could use some const's here.
switch (line->special)
{
    // TRIGGERS.
    // All from here to RETRIGGERS.
case 2:
    // Open Door
    EV_DoDoor(line,open);
    line->special = 0;
    break;

case 3:
    // Close Door
    EV_DoDoor(line,close);
    line->special = 0;
    break;

case 4:
    // Raise Door
    EV_DoDoor(line,normal);
    line->special = 0;
    break;

case 5:
    // Raise Floor
    EV_DoFloor(line,raiseFloor);
    line->special = 0;
    break;

case 6:
    // Fast Ceiling Crush & Raise
    EV_DoCeiling(line,fastCrushAndRaise);
    line->special = 0;
    break;

case 8:
    // Build Stairs
    EV_BuildStairs(line,build8);
    line->special = 0;
    break;

```

```

case 10:
    // PlatDownWaitUp
    EV_DoPlat(line,downWaitUpStay,0);
    line->special = 0;
    break;

case 12:
    // Light Turn On - brightest near
    EV_LightTurnOn(line,0);
    line->special = 0;
    break;

case 13:
    // Light Turn On 255
    EV_LightTurnOn(line,255);
    line->special = 0;
    break;

case 16:
    // Close Door 30
    EV_DoDoor(line,close30ThenOpen);
    line->special = 0;
    break;

case 17:
    // Start Light Strobing
    EV_StartLightStrobing(line);
    line->special = 0;
    break;

case 19:
    // Lower Floor
    EV_DoFloor(line,lowerFloor);
    line->special = 0;
    break;

case 22:
    // Raise floor to nearest height and change texture
    EV_DoPlat(line,raiseToNearestAndChange,0);
    line->special = 0;
    break;

case 25:
    // Ceiling Crush and Raise
    EV_DoCeiling(line,crushAndRaise);
    line->special = 0;
    break;

case 30:
    // Raise floor to shortest texture height
    // on either side of lines.
    EV_DoFloor(line,raiseToTexture);
    line->special = 0;
    break;

case 35:
    // Lights Very Dark
    EV_LightTurnOn(line,35);
    line->special = 0;
    break;

case 36:
    // Lower Floor (TURBO)
    EV_DoFloor(line,turboLower);
    line->special = 0;

```

```

        break;

case 37:
    // LowerAndChange
    EV_DoFloor(line,lowerAndChange);
    line->special = 0;
    break;

case 38:
    // Lower Floor To Lowest
    EV_DoFloor( line, lowerFloorToLowest );
    line->special = 0;
    break;

case 39:
    // TELEPORT!
    EV_Teleport( line, side, thing );
    line->special = 0;
    break;

case 40:
    // RaiseCeilingLowerFloor
    EV_DoCeiling( line, raiseToHighest );
    EV_DoFloor( line, lowerFloorToLowest );
    line->special = 0;
    break;

case 44:
    // Ceiling Crush
    EV_DoCeiling( line, lowerAndCrush );
    line->special = 0;
    break;

case 52:
    // EXIT!
    G_ExitLevel ();
    break;

case 53:
    // Perpetual Platform Raise
    EV_DoPlat(line,perpetualRaise,0);
    line->special = 0;
    break;

case 54:
    // Platform Stop
    EV_StopPlat(line);
    line->special = 0;
    break;

case 56:
    // Raise Floor Crush
    EV_DoFloor(line,raiseFloorCrush);
    line->special = 0;
    break;

case 57:
    // Ceiling Crush Stop
    EV_CeilingCrushStop(line);
    line->special = 0;
    break;

case 58:
    // Raise Floor 24
    EV_DoFloor(line,raiseFloor24);

```

```

    line->special = 0;
    break;

case 59:
    // Raise Floor 24 And Change
    EV_DoFloor(line,raiseFloor24AndChange);
    line->special = 0;
    break;

case 104:
    // Turn lights off in sector(tag)
    EV_TurnTagLightsOff(line);
    line->special = 0;
    break;

case 108:
    // Blazing Door Raise (faster than TURBO!)
    EV_DoDoor (line,blazeRaise);
    line->special = 0;
    break;

case 109:
    // Blazing Door Open (faster than TURBO!)
    EV_DoDoor (line,blazeOpen);
    line->special = 0;
    break;

case 100:
    // Build Stairs Turbo 16
    EV_BuildStairs(line,turbo16);
    line->special = 0;
    break;

case 110:
    // Blazing Door Close (faster than TURBO!)
    EV_DoDoor (line,blazeClose);
    line->special = 0;
    break;

case 119:
    // Raise floor to nearest surr. floor
    EV_DoFloor(line,raiseFloorToNearest);
    line->special = 0;
    break;

case 121:
    // Blazing PlatDownWaitUpStay
    EV_DoPlat(line,blazeDWUS,0);
    line->special = 0;
    break;

case 124:
    // Secret EXIT
    G_SecretExitLevel ();
    break;

case 125:
    // TELEPORT MonsterONLY
    if (!thing->player)
    {
        EV_Teleport( line, side, thing );
        line->special = 0;
    }
    break;

```

```

case 130:
    // Raise Floor Turbo
    EV_DoFloor(line,raiseFloorTurbo);
    line->special = 0;
    break;

case 141:
    // Silent Ceiling Crush & Raise
    EV_DoCeiling(line,silentCrushAndRaise);
    line->special = 0;
    break;

    // RETRIGGERS. All from here till end.
case 72:
    // Ceiling Crush
    EV_DoCeiling( line, lowerAndCrush );
    break;

case 73:
    // Ceiling Crush and Raise
    EV_DoCeiling(line,crushAndRaise);
    break;

case 74:
    // Ceiling Crush Stop
    EV_CeilingCrushStop(line);
    break;

case 75:
    // Close Door
    EV_DoDoor(line,close);
    break;

case 76:
    // Close Door 30
    EV_DoDoor(line,close30ThenOpen);
    break;

case 77:
    // Fast Ceiling Crush & Raise
    EV_DoCeiling(line,fastCrushAndRaise);
    break;

case 79:
    // Lights Very Dark
    EV_LightTurnOn(line,35);
    break;

case 80:
    // Light Turn On - brightest near
    EV_LightTurnOn(line,0);
    break;

case 81:
    // Light Turn On 255
    EV_LightTurnOn(line,255);
    break;

case 82:
    // Lower Floor To Lowest
    EV_DoFloor( line, lowerFloorToLowest );
    break;

case 83:
    // Lower Floor

```

```

    EV_DoFloor(line,lowerFloor);
    break;

case 84:
    // LowerAndChange
    EV_DoFloor(line,lowerAndChange);
    break;

case 86:
    // Open Door
    EV_DoDoor(line,open);
    break;

case 87:
    // Perpetual Platform Raise
    EV_DoPlat(line,perpetualRaise,0);
    break;

case 88:
    // PlatDownWaitUp
    EV_DoPlat(line,downWaitUpStay,0);
    break;

case 89:
    // Platform Stop
    EV_StopPlat(line);
    break;

case 90:
    // Raise Door
    EV_DoDoor(line,normal);
    break;

case 91:
    // Raise Floor
    EV_DoFloor(line,raiseFloor);
    break;

case 92:
    // Raise Floor 24
    EV_DoFloor(line,raiseFloor24);
    break;

case 93:
    // Raise Floor 24 And Change
    EV_DoFloor(line,raiseFloor24AndChange);
    break;

case 94:
    // Raise Floor Crush
    EV_DoFloor(line,raiseFloorCrush);
    break;

case 95:
    // Raise floor to nearest height
    // and change texture.
    EV_DoPlat(line,raiseToNearestAndChange,0);
    break;

case 96:
    // Raise floor to shortest texture height
    // on either side of lines.
    EV_DoFloor(line,raiseToTexture);
    break;

```

```

case 97:
    // TELEPORT!
    EV_Teleport( line, side, thing );
    break;

case 98:
    // Lower Floor (TURBO)
    EV_DoFloor(line,turboLower);
    break;

case 105:
    // Blazing Door Raise (faster than TURBO!)
    EV_DoDoor (line,blazeRaise);
    break;

case 106:
    // Blazing Door Open (faster than TURBO!)
    EV_DoDoor (line,blazeOpen);
    break;

case 107:
    // Blazing Door Close (faster than TURBO!)
    EV_DoDoor (line,blazeClose);
    break;

case 120:
    // Blazing PlatDownWaitUpStay.
    EV_DoPlat(line,blazeDWUS,0);
    break;

case 126:
    // TELEPORT MonsterONLY.
    if (!thing->player)
        EV_Teleport( line, side, thing );
    break;

case 128:
    // Raise To Nearest Floor
    EV_DoFloor(line,raiseFloorToNearest);
    break;

case 129:
    // Raise Floor Turbo
    EV_DoFloor(line,raiseFloorTurbo);
    break;
}
}

//
// P_ShootSpecialLine - IMPACT SPECIALS
// Called when a thing shoots a special line.
//
void
P_ShootSpecialLine
( mobj_t*      thing,
  line_t*      line )
{
    int          ok;

    //          Impacts that other things can activate.
    if (!thing->player)
    {
        ok = 0;

```



```

    switch(line->special)
    {
        case 46:
            // OPEN DOOR IMPACT
            ok = 1;
            break;
    }
    if (!ok)
        return;
}

switch(line->special)
{
    case 24:
        // RAISE FLOOR
        EV_DoFloor(line,raiseFloor);
        P_ChangeSwitchTexture(line,0);
        break;

    case 46:
        // OPEN DOOR
        EV_DoDoor(line,open);
        P_ChangeSwitchTexture(line,1);
        break;

    case 47:
        // RAISE FLOOR NEAR AND CHANGE
        EV_DoPlat(line,raiseToNearestAndChange,0);
        P_ChangeSwitchTexture(line,0);
        break;
}
}

//
// P_PlayerInSpecialSector
// Called every tic frame
// that the player origin is in a special sector
//
void P_PlayerInSpecialSector (player_t* player)
{
    sector_t*      sector;

    sector = player->mo->subsector->sector;

    // Falling, not all the way down yet?
    if (player->mo->z != sector->floorheight)
        return;

    // Has hitten ground.
    switch (sector->special)
    {
        case 5:
            // HELLSLIME DAMAGE
            if (!player->powers[pw_ironfeet])
                if (!(leveltime&0x1f))
                    P_DamageMobj (player->mo, NULL, NULL, 10);
            break;

        case 7:
            // NUKAGE DAMAGE
            if (!player->powers[pw_ironfeet])
                if (!(leveltime&0x1f))
                    P_DamageMobj (player->mo, NULL, NULL, 5);
    }
}

```

```

        break;

case 16:
    // SUPER HELLSLIME DAMAGE
case 4:
    // STROBE HURT
    if (!player->powers[pw_ironfeet]
        || (P_Random()<5) )
    {
        if (!(leveltime&0x1f))
            P_DamageMobj (player->mo, NULL, NULL, 20);
    }
    break;

case 9:
    // SECRET SECTOR
    player->secretcount++;
    sector->special = 0;
    break;

case 11:
    // EXIT SUPER DAMAGE! (for E1M8 finale)
    player->cheats &= ~CF_GODMODE;

    if (!(leveltime&0x1f))
        P_DamageMobj (player->mo, NULL, NULL, 20);

    if (player->health <= 10)
        G_ExitLevel();
    break;

default:
    I_Error ("P_PlayerInSpecialSector: "
            "unknown special %i",
            sector->special);
    break;
};
}

```

```

//
// P_UpdateSpecials
// Animate planes, scroll walls, etc.
//
boolean          levelTimer;
int              levelTimeCount;

```

```

void P_UpdateSpecials (void)
{
    anim_t*       anim;
    int           pic;
    int           i;
    line_t*       line;

```

```

//          LEVEL TIMER
if (levelTimer == true)
{
    levelTimeCount--;
    if (!levelTimeCount)
        G_ExitLevel();
}

```

```

//      ANIMATE FLATS AND TEXTURES GLOBALLY
for (anim = anims ; anim < lastanim ; anim++)
{
    for (i=anim->basepic ; i<anim->basepic+anim->numpics ; i++)
    {
        pic = anim->basepic + ( (leveltime/anim->speed + i)%anim->numpics );
        if (anim->istexture)
            texturetranslation[i] = pic;
        else
            flattranslation[i] = pic;
    }
}

//      ANIMATE LINE SPECIALS
for (i = 0; i < numlinespecials; i++)
{
    line = linespeciallist[i];
    switch(line->special)
    {
        case 48:
            // EFFECT FIRSTCOL SCROLL +
            sides[line->sidenum[0]].textureoffset += FRACUNIT;
            break;
    }
}

//      DO BUTTONS
for (i = 0; i < MAXBUTTONS; i++)
    if (buttonlist[i].btimer)
    {
        buttonlist[i].btimer--;
        if (!buttonlist[i].btimer)
        {
            switch(buttonlist[i].where)
            {
                case top:
                    sides[buttonlist[i].line->sidenum[0]].toptexture =
                        buttonlist[i].btexture;
                    break;

                case middle:
                    sides[buttonlist[i].line->sidenum[0]].midtexture =
                        buttonlist[i].btexture;
                    break;

                case bottom:
                    sides[buttonlist[i].line->sidenum[0]].bottomtexture =
                        buttonlist[i].btexture;
                    break;
            }
            S_StartSound((mobj_t *)&buttonlist[i].soundorg,sfx_swtn);
            memset(&buttonlist[i],0,sizeof(button_t));
        }
    }
}

//
// Special Stuff that can not be categorized
//
int EV_DoDonut(line_t*      line)

```

```

{
    sector_t*      s1;
    sector_t*      s2;
    sector_t*      s3;
    int            secnum;
    int            rtn;
    int            i;
    floormove_t*    floor;

    secnum = -1;
    rtn = 0;
    while ((secnum = P_FindSectorFromLineTag(line,secnum)) >= 0)
    {
        s1 = &sectors[secnum];

        // ALREADY MOVING? IF SO, KEEP GOING...
        if (s1->specialdata)
            continue;

        rtn = 1;
        s2 = getNextSector(s1->lines[0],s1);
        for (i = 0;i < s2->linecount;i++)
        {
            if ((!s2->lines[i]->flags & ML_TWOSIDED) ||
                (s2->lines[i]->backsector == s1))
                continue;
            s3 = s2->lines[i]->backsector;

            //      Spawn rising slime
            floor = Z_Malloc (sizeof(*floor), PU_LEVSPEC, 0);
            P_AddThinker (&floor->thinker);
            s2->specialdata = floor;
            floor->thinker.function.acp1 = (actionf_p1) T_MoveFloor;
            floor->type = donutRaise;
            floor->crush = false;
            floor->direction = 1;
            floor->sector = s2;
            floor->speed = FLOORSPEED / 2;
            floor->texture = s3->floorpic;
            floor->newspecial = 0;
            floor->floordestheight = s3->floorheight;

            //      Spawn lowering donut-hole
            floor = Z_Malloc (sizeof(*floor), PU_LEVSPEC, 0);
            P_AddThinker (&floor->thinker);
            s1->specialdata = floor;
            floor->thinker.function.acp1 = (actionf_p1) T_MoveFloor;
            floor->type = lowerFloor;
            floor->crush = false;
            floor->direction = -1;
            floor->sector = s1;
            floor->speed = FLOORSPEED / 2;
            floor->floordestheight = s3->floorheight;
            break;
        }
    }
    return rtn;
}

//
// SPECIAL SPAWNING
//

```

```

//
// P_SpawnSpecials
// After the map has been loaded, scan for specials
// that spawn thinkers
//
short                numlinespecials;
line_t*              linespeciallist[MAXLINEANIMS];

// Parses command line parameters.
void P_SpawnSpecials (void)
{
    sector_t*         sector;
    int                i;
    int                episode;

    episode = 1;
    if (W_CheckNumForName("texture2") >= 0)
        episode = 2;

    // See if -TIMER needs to be used.
    levelTimer = false;

    i = M_CheckParm("-avg");
    if (i && deathmatch)
    {
        levelTimer = true;
        levelTimeCount = 20 * 60 * 35;
    }

    i = M_CheckParm("-timer");
    if (i && deathmatch)
    {
        int            time;
        time = atoi(myargv[i+1]) * 60 * 35;
        levelTimer = true;
        levelTimeCount = time;
    }

    //      Init special SECTORS.
    sector = sectors;
    for (i=0 ; i<numsectors ; i++, sector++)
    {
        if (!sector->special)
            continue;

        switch (sector->special)
        {
            case 1:
                // FLICKERING LIGHTS
                P_SpawnLightFlash (sector);
                break;

            case 2:
                // STROBE FAST
                P_SpawnStrobeFlash(sector,FASTDARK,0);
                break;

            case 3:
                // STROBE SLOW
                P_SpawnStrobeFlash(sector,SLOWDARK,0);
                break;

            case 4:

```

```

        // STROBE FAST/DEATH SLIME
        P_SpawnStrobeFlash(sectors,FASTDARK,0);
        sectors->special = 4;
        break;

    case 8:
        // GLOWING LIGHT
        P_SpawnGlowingLight(sectors);
        break;
    case 9:
        // SECRET SECTOR
        totalsecret++;
        break;

    case 10:
        // DOOR CLOSE IN 30 SECONDS
        P_SpawnDoorCloseIn30 (sectors);
        break;

    case 12:
        // SYNC STROBE SLOW
        P_SpawnStrobeFlash (sectors, SLOWDARK, 1);
        break;

    case 13:
        // SYNC STROBE FAST
        P_SpawnStrobeFlash (sectors, FASTDARK, 1);
        break;

    case 14:
        // DOOR RAISE IN 5 MINUTES
        P_SpawnDoorRaiseIn5Mins (sectors, i);
        break;

    case 17:
        P_SpawnFireFlicker(sectors);
        break;
}

}

//      Init line EFFECTs
numlinespecials = 0;
for (i = 0;i < numlines; i++)
{
    switch(lines[i].special)
    {
        case 48:
            // EFFECT FIRSTCOL SCROLL+
            linespeciallist[numlinespecials] = &lines[i];
            numlinespecials++;
            break;
    }
}

//      Init other misc stuff
for (i = 0;i < MAXCEILINGS;i++)
    activeceilings[i] = NULL;

for (i = 0;i < MAXPLATS;i++)
    activeplats[i] = NULL;

for (i = 0;i < MAXBUTTONS;i++)
    memset(&buttonlist[i],0,sizeof(button_t));

```

```

    // UNUSED: no horizontal sliders.
    //      P_InitSlidingDoorFrames();
}

```

## 9.24 p\_spec.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:  none
//      Implements special effects:
//      Texture animation, height or lighting changes
//      according to adjacent sectors, respective
//      utility functions, etc.
//-----

#ifndef __P_SPEC__
#define __P_SPEC__

//
// End-level timer (-TIMER option)
//
extern      boolean levelTimer;
extern      int      levelTimeCount;

//      Define values for map objects
#define MO_TELEPORTMAN      14

// at game start
void      P_InitPicAnims (void);

// at map load
void      P_SpawnSpecials (void);

// every tic
void      P_UpdateSpecials (void);

// when needed
boolean
P_UseSpecialLine
( mobj_t*      thing,
  line_t*      line,
  int          side );

void

```

```

P_ShootSpecialLine
( mobj_t*      thing,
  line_t*      line );

void
P_CrossSpecialLine
( int          linenum,
  int          side,
  mobj_t*      thing );

void    P_PlayerInSpecialSector (player_t* player);

int
twoSided
( int          sector,
  int          line );

sector_t*
getSector
( int          currentSector,
  int          line,
  int          side );

side_t*
getSide
( int          currentSector,
  int          line,
  int          side );

fixed_t P_FindLowestFloorSurrounding(sector_t* sec);
fixed_t P_FindHighestFloorSurrounding(sector_t* sec);

fixed_t
P_FindNextHighestFloor
( sector_t*    sec,
  int          currentheight );

fixed_t P_FindLowestCeilingSurrounding(sector_t* sec);
fixed_t P_FindHighestCeilingSurrounding(sector_t* sec);

int
P_FindSectorFromLineTag
( line_t*      line,
  int          start );

int
P_FindMinSurroundingLight
( sector_t*    sector,
  int          max );

sector_t*
getNextSector
( line_t*      line,
  sector_t*    sec );

//
// SPECIAL
//
int EV_DoDonut(line_t* line);

//
// P_LIGHTS

```



```

//
typedef struct
{
    thinker_t      thinker;
    sector_t*      sector;
    int             count;
    int             maxlight;
    int             minlight;

} fireflicker_t;


typedef struct
{
    thinker_t      thinker;
    sector_t*      sector;
    int             count;
    int             maxlight;
    int             minlight;
    int             maxtime;
    int             mintime;

} lightflash_t;


typedef struct
{
    thinker_t      thinker;
    sector_t*      sector;
    int             count;
    int             minlight;
    int             maxlight;
    int             darktime;
    int             brighttime;

} strobe_t;


typedef struct
{
    thinker_t      thinker;
    sector_t*      sector;
    int             minlight;
    int             maxlight;
    int             direction;

} glow_t;


#define GLOWSPEED          8
#define STROBEBRIGHT      5
#define FASTDARK           15
#define SLOWDARK           35


void    P_SpawnFireFlicker (sector_t* sector);
void    T_LightFlash (lightflash_t* flash);
void    P_SpawnLightFlash (sector_t* sector);
void    T_StrobeFlash (strobe_t* flash);


void
P_SpawnStrobeFlash

```

```

( sector_t*      sector,
  int            fastOrSlow,
  int            inSync );

void    EV_StartLightStrobing(line_t* line);
void    EV_TurnTagLightsOff(line_t* line);

void
EV_LightTurnOn
( line_t*      line,
  int          bright );

void    T_Glow(glow_t* g);
void    P_SpawnGlowingLight(sector_t* sector);


//
// P_SWITCH
//
typedef struct
{
    char        name1[9];
    char        name2[9];
    short       episode;

} switchlist_t;

typedef enum
{
    top,
    middle,
    bottom

} bwhere_e;

typedef struct
{
    line_t*      line;
    bwhere_e     where;
    int          btexture;
    int          btimer;
    mobj_t*      soundorg;

} button_t;


// max # of wall switches in a level
#define MAXSWITCHES      50

// 4 players, 4 buttons each at once, max.
#define MAXBUTTONS      16

// 1 second, in ticks.
#define BUTTONTIME      35

extern button_t      buttonlist[MAXBUTTONS];

void
P_ChangeSwitchTexture

```

```

( line_t*      line,
  int          useAgain );

void P_InitSwitchList(void);

//
// P_PLATS
//
typedef enum
{
    up,
    down,
    waiting,
    in_stasis
} plat_e;

typedef enum
{
    perpetualRaise,
    downWaitUpStay,
    raiseAndChange,
    raiseToNearestAndChange,
    blazeDWUS
} plattype_e;

typedef struct
{
    thinker_t      thinker;
    sector_t*      sector;
    fixed_t        speed;
    fixed_t        low;
    fixed_t        high;
    int             wait;
    int             count;
    plat_e          status;
    plat_e          oldstatus;
    boolean         crush;
    int             tag;
    plattype_e      type;
} plat_t;

#define PLATWAIT          3
#define PLATSPEED         FRACUNIT
#define MAXPLATS         30

extern plat_t*          activeplats[MAXPLATS];

void    T_PlatformRaise(plat_t*      plat);

int
EV_DoPlatform
( line_t*      line,
  plattype_e    type,
  int           amount );

```

```

void    P_AddActivePlat(plat_t* plat);
void    P_RemoveActivePlat(plat_t* plat);
void    EV_StopPlat(line_t* line);
void    P_ActivateInStasis(int tag);

//
// P_DOORS
//
typedef enum
{
    normal,
    close30ThenOpen,
    close,
    open,
    raiseIn5Mins,
    blazeRaise,
    blazeOpen,
    blazeClose
} vldoor_e;

typedef struct
{
    thinker_t      thinker;
    vldoor_e       type;
    sector_t*      sector;
    fixed_t        topheight;
    fixed_t        speed;

    // 1 = up, 0 = waiting at top, -1 = down
    int            direction;

    // tics to wait at the top
    int            topwait;
    // (keep in case a door going down is reset)
    // when it reaches 0, start going down
    int            topcountdown;
} vldoor_t;

#define VDOORSPEED          FRACUNIT*2
#define VDOORWAIT           150

void
EV_VerticalDoor
( line_t*         line,
  mobj_t*         thing );

int
EV_DoDoor
( line_t*         line,
  vldoor_e        type );

int
EV_DoLockedDoor
( line_t*         line,
  vldoor_e        type,
  mobj_t*         thing );

```

```

void    T_VerticalDoor (vldoor_t* door);
void    P_SpawnDoorCloseIn30 (sector_t* sec);

```

```

void
P_SpawnDoorRaiseIn5Mins
( sector_t*      sec,
  int            secnum );

```

```

#if 0 // UNUSED
//
//      Sliding doors...
//
typedef enum
{
    sd_opening,
    sd_waiting,
    sd_closing

```

```

} sd_e;

```

```

typedef enum
{
    sdt_openOnly,
    sdt_closeOnly,
    sdt_openAndClose

```

```

} sdt_e;

```

```

typedef struct
{
    thinker_t      thinker;
    sdt_e          type;
    line_t*        line;
    int            frame;
    int            whichDoorIndex;
    int            timer;
    sector_t*      frontsector;
    sector_t*      backsector;
    sd_e          status;

```

```

} slidedoor_t;

```

```

typedef struct
{
    char          frontFrame1[9];
    char          frontFrame2[9];
    char          frontFrame3[9];
    char          frontFrame4[9];
    char          backFrame1[9];
    char          backFrame2[9];
    char          backFrame3[9];
    char          backFrame4[9];

```

```

} slidename_t;

```

```

typedef struct
{
    int          frontFrames[4];
    int          backFrames[4];

} slideframe_t;

// how many frames of animation
#define SNUMFRAMES          4

#define SDOORWAIT           35*3
#define SWAITTICS           4

// how many diff. types of anims
#define MAXSLIDEDOORS       5

void P_InitSlidingDoorFrames(void);

void
EV_SlidingDoor
( line_t*        line,
  mobj_t*        thing );
#endif

//
// P_CEILING
//
typedef enum
{
    lowerToFloor,
    raiseToHighest,
    lowerAndCrush,
    crushAndRaise,
    fastCrushAndRaise,
    silentCrushAndRaise

} ceiling_e;

typedef struct
{
    thinker_t      thinker;
    ceiling_e      type;
    sector_t*      sector;
    fixed_t        bottomheight;
    fixed_t        topheight;
    fixed_t        speed;
    boolean        crush;

    // 1 = up, 0 = waiting, -1 = down
    int            direction;

    // ID
    int            tag;
    int            olddirection;

} ceiling_t;

```

```

#define CEILSPEED                FRACUNIT
#define CEILWAIT                150
#define MAXCEILINGS              30

extern ceiling_t*                activeceilings[MAXCEILINGS];

int
EV_DoCeiling
( line_t*            line,
  ceiling_e          type );

void  T_MoveCeiling (ceiling_t* ceiling);
void  P_AddActiveCeiling(ceiling_t* c);
void  P_RemoveActiveCeiling(ceiling_t* c);
int    EV_CeilingCrushStop(line_t* line);
void  P_ActivateInStasisCeiling(line_t* line);

//
// P_FLOOR
//
typedef enum
{
    // lower floor to highest surrounding floor
    lowerFloor,

    // lower floor to lowest surrounding floor
    lowerFloorToLowest,

    // lower floor to highest surrounding floor VERY FAST
    turboLower,

    // raise floor to lowest surrounding CEILING
    raiseFloor,

    // raise floor to next highest surrounding floor
    raiseFloorToNearest,

    // raise floor to shortest height texture around it
    raiseToTexture,

    // lower floor to lowest surrounding floor
    // and change floorpic
    lowerAndChange,

    raiseFloor24,
    raiseFloor24AndChange,
    raiseFloorCrush,

    // raise to next highest floor, turbo-speed
    raiseFloorTurbo,
    donutRaise,
    raiseFloor512
} floor_e;

typedef enum
{
    build8,          // slowly build by 8

```

```

        turbo16        // quickly build by 16

} stair_e;

typedef struct
{
    thinker_t        thinker;
    floor_e          type;
    boolean          crush;
    sector_t*        sector;
    int              direction;
    int              newspecial;
    short            texture;
    fixed_t          floordestheight;
    fixed_t          speed;

} floormove_t;

#define FLOORSPEED          FRACUNIT

typedef enum
{
    ok,
    crushed,
    pastdest

} result_e;

result_e
T_MovePlane
( sector_t*          sector,
  fixed_t            speed,
  fixed_t            dest,
  boolean            crush,
  int                floorOrCeiling,
  int                direction );

int
EV_BuildStairs
( line_t*            line,
  stair_e            type );

int
EV_DoFloor
( line_t*            line,
  floor_e            floortype );

void T_MoveFloor( floormove_t* floor);

//
// P_TELEPT
//
int
EV_Teleport
( line_t*            line,
  int                side,
  mobj_t*            thing );

#endif
//-----
//

```



```

// $Log:$
//
//-----

9.25  p_switch.c

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
//
// $Log:$
//
// DESCRIPTION:
//      Switches, buttons. Two-state animation. Exits.
//
//-----

static const char
rcsid[] = "$Id: p_switch.c,v 1.3 1997/01/28 22:08:29 b1 Exp $";

#include "i_system.h"
#include "doomdef.h"
#include "p_local.h"

#include "g_game.h"

#include "s_sound.h"

// Data.
#include "sounds.h"

// State.
#include "doomstat.h"
#include "r_state.h"

//
// CHANGE THE TEXTURE OF A WALL SWITCH TO ITS OPPOSITE
//
switchlist_t alphSwitchList[] =
{
    // Doom shareware episode 1 switches
    {"SW1BRCOM",      "SW2BRCOM",      1},
    {"SW1BRN1",       "SW2BRN1",       1},
    {"SW1BRN2",       "SW2BRN2",       1},
    {"SW1BRNGN",      "SW2BRNGN",      1},
    {"SW1BROWN",      "SW2BROWN",      1},
    {"SW1COMM",       "SW2COMM",       1},
    {"SW1COMP",       "SW2COMP",       1},
    {"SW1DIRT",       "SW2DIRT",       1},

```

```

{"SW1EXIT",      "SW2EXIT",      1},
{"SW1GRAY",      "SW2GRAY",      1},
{"SW1GRAY1",     "SW2GRAY1",     1},
{"SW1METAL",     "SW2METAL",     1},
{"SW1PIPE",      "SW2PIPE",      1},
{"SW1SLAD",      "SW2SLAD",      1},
{"SW1STARG",     "SW2STARG",     1},
{"SW1STON1",     "SW2STON1",     1},
{"SW1STON2",     "SW2STON2",     1},
{"SW1STONE",     "SW2STONE",     1},
{"SW1STRTN",     "SW2STRTN",     1},

// Doom registered episodes 2&3 switches
{"SW1BLUE",      "SW2BLUE",      2},
{"SW1CMT",       "SW2CMT",      2},
{"SW1GARG",      "SW2GARG",      2},
{"SW1GSTON",     "SW2GSTON",     2},
{"SW1HOT",       "SW2HOT",      2},
{"SW1LION",      "SW2LION",      2},
{"SW1SATYR",     "SW2SATYR",     2},
{"SW1SKIN",      "SW2SKIN",      2},
{"SW1VINE",      "SW2VINE",      2},
{"SW1WOOD",      "SW2WOOD",      2},

// Doom II switches
{"SW1PANEL",     "SW2PANEL",     3},
{"SW1ROCK",      "SW2ROCK",      3},
{"SW1MET2",      "SW2MET2",      3},
{"SW1WDMET",     "SW2WDMET",     3},
{"SW1BRIK",      "SW2BRIK",      3},
{"SW1MOD1",      "SW2MOD1",      3},
{"SW1ZIM",       "SW2ZIM",      3},
{"SW1STON6",     "SW2STON6",     3},
{"SW1TEK",       "SW2TEK",      3},
{"SW1MARB",      "SW2MARB",      3},
{"SW1SKULL",     "SW2SKULL",     3},

{"\0",           "\0",           0}
};

int          switchlist[MAXSWITCHES * 2];
int          numswitches;
button_t     buttonlist[MAXBUTTONS];

//
// P_InitSwitchList
// Only called at game initialization.
//
void P_InitSwitchList(void)
{
    int        i;
    int        index;
    int        episode;

    episode = 1;

    if (gamemode == registered)
        episode = 2;
    else
        if ( gamemode == commercial )
            episode = 3;

    for (index = 0, i = 0; i < MAXSWITCHES; i++)
    {
        if (!alphSwitchList[i].episode)

```

```

    {
        numswitches = index/2;
        switchlist[index] = -1;
        break;
    }

    if (alphSwitchList[i].episode <= episode)
    {
#ifdef 0
        // UNUSED - debug?
        int          value;

        if (R_CheckTextureNumForName(alphSwitchList[i].name1) < 0)
        {
            I_Error("Can't find switch texture '%s'!",
                    alphSwitchList[i].name1);
            continue;
        }

        value = R_TextureNumForName(alphSwitchList[i].name1);
#endif

        switchlist[index++] = R_TextureNumForName(alphSwitchList[i].name1);
        switchlist[index++] = R_TextureNumForName(alphSwitchList[i].name2);
    }
}

//
// Start a button counting down till it turns off.
//
void
P_StartButton
( line_t*      line,
  bwhere_e     w,
  int          texture,
  int          time )
{
    int          i;

    // See if button is already pressed
    for (i = 0; i < MAXBUTTONS; i++)
    {
        if (buttonlist[i].btimer
            && buttonlist[i].line == line)
        {
            return;
        }
    }

    for (i = 0; i < MAXBUTTONS; i++)
    {
        if (!buttonlist[i].btimer)
        {
            buttonlist[i].line = line;
            buttonlist[i].where = w;
            buttonlist[i].btexture = texture;
            buttonlist[i].btimer = time;
            buttonlist[i].soundorg = (mobj_t *)&line->frontsector->soundorg;
            return;
        }
    }
}

```

```

    I_Error("P_StartButton: no button slots left!");
}

//
// Function that changes wall texture.
// Tell it if switch is ok to use again (1=yes, it's a button).
//
void
P_ChangeSwitchTexture
( line_t*      line,
  int          useAgain )
{
    int      texTop;
    int      texMid;
    int      texBot;
    int      i;
    int      sound;

    if (!useAgain)
        line->special = 0;

    texTop = sides[line->sidenum[0]].toptexture;
    texMid = sides[line->sidenum[0]].midtexture;
    texBot = sides[line->sidenum[0]].bottomtexture;

    sound = sfx_swtn;

    // EXIT SWITCH?
    if (line->special == 11)
        sound = sfx_swtn;

    for (i = 0; i < numswitches*2; i++)
    {
        if (switchlist[i] == texTop)
        {
            S_StartSound(buttonlist->soundorg, sound);
            sides[line->sidenum[0]].toptexture = switchlist[i^1];

            if (useAgain)
                P_StartButton(line, top, switchlist[i], BUTTONTIME);

            return;
        }
        else
        {
            if (switchlist[i] == texMid)
            {
                S_StartSound(buttonlist->soundorg, sound);
                sides[line->sidenum[0]].midtexture = switchlist[i^1];

                if (useAgain)
                    P_StartButton(line, middle, switchlist[i], BUTTONTIME);

                return;
            }
            else
            {
                if (switchlist[i] == texBot)
                {
                    S_StartSound(buttonlist->soundorg, sound);
                    sides[line->sidenum[0]].bottomtexture = switchlist[i^1];

```

```

        if (useAgain)
            P_StartButton(line, bottom,switchlist[i],BUTTONTIME);

        return;
    }
}
}
}
}
}

```

```

//
// P_UseSpecialLine
// Called when a thing uses a special line.
// Only the front sides of lines are usable.
//
boolean
P_UseSpecialLine
( mobj_t*      thing,
  line_t*      line,
  int          side )
{
    // Err...
    // Use the back sides of VERY SPECIAL lines...
    if (side)
    {
        switch(line->special)
        {
            case 124:
                // Sliding door open&close
                // UNUSED?
                break;

            default:
                return false;
                break;
        }
    }

    // Switches that other things can activate.
    if (!thing->player)
    {
        // never open secret doors
        if (line->flags & ML_SECRET)
            return false;

        switch(line->special)
        {
            case 1:          // MANUAL DOOR RAISE
            case 32:         // MANUAL BLUE
            case 33:         // MANUAL RED
            case 34:         // MANUAL YELLOW
                break;

            default:
                return false;
                break;
        }
    }
}

```

```

}

// do something
switch (line->special)
{
    // MANUALS
    case 1:                // Vertical Door
    case 26:                // Blue Door/Locked
    case 27:                // Yellow Door /Locked
    case 28:                // Red Door /Locked

    case 31:                // Manual door open
    case 32:                // Blue locked door open
    case 33:                // Red locked door open
    case 34:                // Yellow locked door open

    case 117:               // Blazing door raise
    case 118:               // Blazing door open
        EV_VerticalDoor (line, thing);
        break;

    //UNUSED - Door Slide Open&Close
    // case 124:
    // EV_SlidingDoor (line, thing);
    // break;

    // SWITCHES
    case 7:
        // Build Stairs
        if (EV_BuildStairs(line,build8))
            P_ChangeSwitchTexture(line,0);
        break;

    case 9:
        // Change Donut
        if (EV_DoDonut(line))
            P_ChangeSwitchTexture(line,0);
        break;

    case 11:
        // Exit level
        P_ChangeSwitchTexture(line,0);
        G_ExitLevel ();
        break;

    case 14:
        // Raise Floor 32 and change texture
        if (EV_DoPlat(line,raiseAndChange,32))
            P_ChangeSwitchTexture(line,0);
        break;

    case 15:
        // Raise Floor 24 and change texture
        if (EV_DoPlat(line,raiseAndChange,24))
            P_ChangeSwitchTexture(line,0);
        break;

    case 18:
        // Raise Floor to next highest floor
        if (EV_DoFloor(line, raiseFloorToNearest))
            P_ChangeSwitchTexture(line,0);
        break;

    case 20:

```

```

// Raise Plat next highest floor and change texture
if (EV_DoPlat(line,raiseToNearestAndChange,0))
    P_ChangeSwitchTexture(line,0);
break;

case 21:
    // PlatDownWaitUpStay
    if (EV_DoPlat(line,downWaitUpStay,0))
        P_ChangeSwitchTexture(line,0);
    break;

case 23:
    // Lower Floor to Lowest
    if (EV_DoFloor(line,lowerFloorToLowest))
        P_ChangeSwitchTexture(line,0);
    break;

case 29:
    // Raise Door
    if (EV_DoDoor(line,normal))
        P_ChangeSwitchTexture(line,0);
    break;

case 41:
    // Lower Ceiling to Floor
    if (EV_DoCeiling(line,lowerToFloor))
        P_ChangeSwitchTexture(line,0);
    break;

case 71:
    // Turbo Lower Floor
    if (EV_DoFloor(line,turboLower))
        P_ChangeSwitchTexture(line,0);
    break;

case 49:
    // Ceiling Crush And Raise
    if (EV_DoCeiling(line,crushAndRaise))
        P_ChangeSwitchTexture(line,0);
    break;

case 50:
    // Close Door
    if (EV_DoDoor(line,close))
        P_ChangeSwitchTexture(line,0);
    break;

case 51:
    // Secret EXIT
    P_ChangeSwitchTexture(line,0);
    G_SecretExitLevel ();
    break;

case 55:
    // Raise Floor Crush
    if (EV_DoFloor(line,raiseFloorCrush))
        P_ChangeSwitchTexture(line,0);
    break;

case 101:
    // Raise Floor
    if (EV_DoFloor(line,raiseFloor))
        P_ChangeSwitchTexture(line,0);
    break;

```

```

case 102:
    // Lower Floor to Surrounding floor height
    if (EV_DoFloor(line,lowerFloor))
        P_ChangeSwitchTexture(line,0);
    break;

case 103:
    // Open Door
    if (EV_DoDoor(line,open))
        P_ChangeSwitchTexture(line,0);
    break;

case 111:
    // Blazing Door Raise (faster than TURBO!)
    if (EV_DoDoor (line,blazeRaise))
        P_ChangeSwitchTexture(line,0);
    break;

case 112:
    // Blazing Door Open (faster than TURBO!)
    if (EV_DoDoor (line,blazeOpen))
        P_ChangeSwitchTexture(line,0);
    break;

case 113:
    // Blazing Door Close (faster than TURBO!)
    if (EV_DoDoor (line,blazeClose))
        P_ChangeSwitchTexture(line,0);
    break;

case 122:
    // Blazing PlatDownWaitUpStay
    if (EV_DoPlat(line,blazeDWUS,0))
        P_ChangeSwitchTexture(line,0);
    break;

case 127:
    // Build Stairs Turbo 16
    if (EV_BuildStairs(line,turbo16))
        P_ChangeSwitchTexture(line,0);
    break;

case 131:
    // Raise Floor Turbo
    if (EV_DoFloor(line,raiseFloorTurbo))
        P_ChangeSwitchTexture(line,0);
    break;

case 133:
    // BlzOpenDoor BLUE
case 135:
    // BlzOpenDoor RED
case 137:
    // BlzOpenDoor YELLOW
    if (EV_DoLockedDoor (line,blazeOpen,thing))
        P_ChangeSwitchTexture(line,0);
    break;

case 140:
    // Raise Floor 512
    if (EV_DoFloor(line,raiseFloor512))
        P_ChangeSwitchTexture(line,0);
    break;

// BUTTONS

```



```

case 42:
    // Close Door
    if (EV_DoDoor(line,close))
        P_ChangeSwitchTexture(line,1);
    break;

case 43:
    // Lower Ceiling to Floor
    if (EV_DoCeiling(line,lowerToFloor))
        P_ChangeSwitchTexture(line,1);
    break;

case 45:
    // Lower Floor to Surrounding floor height
    if (EV_DoFloor(line,lowerFloor))
        P_ChangeSwitchTexture(line,1);
    break;

case 60:
    // Lower Floor to Lowest
    if (EV_DoFloor(line,lowerFloorToLowest))
        P_ChangeSwitchTexture(line,1);
    break;

case 61:
    // Open Door
    if (EV_DoDoor(line,open))
        P_ChangeSwitchTexture(line,1);
    break;

case 62:
    // PlatDownWaitUpStay
    if (EV_DoPlat(line,downWaitUpStay,1))
        P_ChangeSwitchTexture(line,1);
    break;

case 63:
    // Raise Door
    if (EV_DoDoor(line,normal))
        P_ChangeSwitchTexture(line,1);
    break;

case 64:
    // Raise Floor to ceiling
    if (EV_DoFloor(line,raiseFloor))
        P_ChangeSwitchTexture(line,1);
    break;

case 66:
    // Raise Floor 24 and change texture
    if (EV_DoPlat(line,raiseAndChange,24))
        P_ChangeSwitchTexture(line,1);
    break;

case 67:
    // Raise Floor 32 and change texture
    if (EV_DoPlat(line,raiseAndChange,32))
        P_ChangeSwitchTexture(line,1);
    break;

case 65:
    // Raise Floor Crush
    if (EV_DoFloor(line,raiseFloorCrush))
        P_ChangeSwitchTexture(line,1);
    break;

```

```

case 68:
    // Raise Plat to next highest floor and change texture
    if (EV_DoPlat(line,raiseToNearestAndChange,0))
        P_ChangeSwitchTexture(line,1);
    break;

case 69:
    // Raise Floor to next highest floor
    if (EV_DoFloor(line, raiseFloorToNearest))
        P_ChangeSwitchTexture(line,1);
    break;

case 70:
    // Turbo Lower Floor
    if (EV_DoFloor(line,turboLower))
        P_ChangeSwitchTexture(line,1);
    break;

case 114:
    // Blazing Door Raise (faster than TURBO!)
    if (EV_DoDoor (line,blazeRaise))
        P_ChangeSwitchTexture(line,1);
    break;

case 115:
    // Blazing Door Open (faster than TURBO!)
    if (EV_DoDoor (line,blazeOpen))
        P_ChangeSwitchTexture(line,1);
    break;

case 116:
    // Blazing Door Close (faster than TURBO!)
    if (EV_DoDoor (line,blazeClose))
        P_ChangeSwitchTexture(line,1);
    break;

case 123:
    // Blazing PlatDownWaitUpStay
    if (EV_DoPlat(line,blazeDWUS,0))
        P_ChangeSwitchTexture(line,1);
    break;

case 132:
    // Raise Floor Turbo
    if (EV_DoFloor(line,raiseFloorTurbo))
        P_ChangeSwitchTexture(line,1);
    break;

case 99:
    // BlzOpenDoor BLUE
case 134:
    // BlzOpenDoor RED
case 136:
    // BlzOpenDoor YELLOW
    if (EV_DoLockedDoor (line,blazeOpen,thing))
        P_ChangeSwitchTexture(line,1);
    break;

case 138:
    // Light Turn On
    EV_LightTurnOn(line,255);
    P_ChangeSwitchTexture(line,1);
    break;

```

```

        case 139:
            // Light Turn Off
            EV_LightTurnOn(line,35);
            P_ChangeSwitchTexture(line,1);
            break;

    }

    return true;
}

```

## 9.26 p\_telept.c

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Teleportation.
//-----

static const char
rcsid[] = "$Id: p_telept.c,v 1.3 1997/01/28 22:08:29 b1 Exp $";

#include "doomdef.h"

#include "s_sound.h"

#include "p_local.h"

// Data.
#include "sounds.h"

// State.
#include "r_state.h"

//
// TELEPORTATION
//
int
EV_Teleport
( line_t*      line,
  int          side,

```

```

mobj_t*      thing )
{
    int        i;
    int        tag;
    mobj_t*    m;
    mobj_t*    fog;
    unsigned   an;
    thinker_t* thinker;
    sector_t*  sector;
    fixed_t    oldx;
    fixed_t    oldy;
    fixed_t    oldz;

    // don't teleport missiles
    if (thing->flags & MF_MISSILE)
        return 0;

    // Don't teleport if hit back of line,
    // so you can get out of teleporter.
    if (side == 1)
        return 0;

    tag = line->tag;
    for (i = 0; i < numsectors; i++)
    {
        if (sectors[ i ].tag == tag )
        {
            thinker = thinkercap.next;
            for (thinker = thinkercap.next;
                 thinker != &thinkercap;
                 thinker = thinker->next)
            {
                // not a mobj
                if (thinker->function.acp1 != (actionf_p1)P_MobjThinker)
                    continue;

                m = (mobj_t *)thinker;

                // not a teleportman
                if (m->type != MT_TELEPORTMAN )
                    continue;

                sector = m->subsector->sector;
                // wrong sector
                if (sector-sectors != i )
                    continue;

                oldx = thing->x;
                oldy = thing->y;
                oldz = thing->z;

                if (!P_TeleportMove (thing, m->x, m->y))
                    return 0;

                thing->z = thing->floorz; //fixme: not needed?
                if (thing->player)
                    thing->player->viewz = thing->z+thing->player->viewheight;

                // spawn teleport fog at source and destination
                fog = P_SpawnMobj (oldx, oldy, oldz, MT_TFOG);
                S_StartSound (fog, sfx_telept);
                an = m->angle >> ANGLETOFINESHIFT;
                fog = P_SpawnMobj (m->x+20*finecosine[an], m->y+20*finesine[an]
                                   , thing->z, MT_TFOG);
            }
        }
    }
}

```

```

        // emit sound, where?
        S_StartSound (fog, sfx_telept);

        // don't move for a bit
        if (thing->player)
            thing->reactiontime = 18;

        thing->angle = m->angle;
        thing->momx = thing->momy = thing->momz = 0;
        return 1;
    }
}
return 0;
}

```

## 9.27 p\_tick.c

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Archiving: SaveGame I/O.
//     Thinker, Ticker.
//
//-----

static const char
rcsid[] = "$Id: p_tick.c,v 1.4 1997/02/03 16:47:55 b1 Exp $";

#include "z_zone.h"
#include "p_local.h"

#include "doomstat.h"

int         leveltime;

//
// THINKERS
// All thinkers should be allocated by Z_Malloc
// so they can be operated on uniformly.
// The actual structures will vary in size,
// but the first element must be thinker_t.
//

```

```

// Both the head and tail of the thinker list.
thinker_t      thinkercap;

//
// P_InitThinkers
//
void P_InitThinkers (void)
{
    thinkercap.prev = thinkercap.next = &thinkercap;
}

//
// P_AddThinker
// Adds a new thinker at the end of the list.
//
void P_AddThinker (thinker_t* thinker)
{
    thinkercap.prev->next = thinker;
    thinker->next = &thinkercap;
    thinker->prev = thinkercap.prev;
    thinkercap.prev = thinker;
}

//
// P_RemoveThinker
// Deallocation is lazy -- it will not actually be freed
// until its thinking turn comes up.
//
void P_RemoveThinker (thinker_t* thinker)
{
    // FIXME: NOP.
    thinker->function.acv = (actionf_v)(-1);
}

//
// P_AllocateThinker
// Allocates memory and adds a new thinker at the end of the list.
//
void P_AllocateThinker (thinker_t*      thinker)
{
}

//
// P_RunThinkers
//
void P_RunThinkers (void)
{
    thinker_t*      currentthinker;

    currentthinker = thinkercap.next;
    while (currentthinker != &thinkercap)
    {
        if ( currentthinker->function.acv == (actionf_v)(-1) )
        {

```

```

        // time to remove it
        currentthinker->next->prev = currentthinker->prev;
        currentthinker->prev->next = currentthinker->next;
        Z_Free (currentthinker);
    }
    else
    {
        if (currentthinker->function.acp1)
            currentthinker->function.acp1 (currentthinker);
    }
    currentthinker = currentthinker->next;
}
}

```

```

//
// P_Ticker
//

void P_Ticker (void)
{
    int            i;

    // run the tic
    if (paused)
        return;

    // pause if in menu and at least one tic has been run
    if ( !netgame
        && menuactive
        && !demoplayback
        && players[consoleplayer].viewz != 1)
    {
        return;
    }

    for (i=0 ; i<MAXPLAYERS ; i++)
        if (playeringame[i])
            P_PlayerThink (&players[i]);

    P_RunThinkers ();
    P_UpdateSpecials ();
    P_RespawnSpecials ();

    // for par times
    leveltime++;
}

```

## 9.28 p\_tick.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,

```

```

// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      ?
//
//-----

#ifndef __P_TICK__
#define __P_TICK__

#ifdef __GNUG__
#pragma interface
#endif

// Called by C_Ticker,
// can call G_PlayerExited.
// Carries out all thinking of monsters and players.
void P_Ticker (void);

#endif
//-----
//
// $Log:$
//
//-----

```

## 9.29 p\_user.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Player related stuff.
//      Bobbing POV/weapon, movement.
//      Pending weapon.
//
//-----

static const char
rcsid[] = "$Id: p_user.c,v 1.3 1997/01/28 22:08:29 b1 Exp $";

```



```

#include "doomdef.h"
#include "d_event.h"

#include "p_local.h"

#include "doomstat.h"

// Index of the special effects (INVUL inverse) map.
#define INVERSECOLORMAP          32

//
// Movement.
//

// 16 pixels of bob
#define MAXBOB          0x100000

boolean          onground;

//
// P_Thrust
// Moves the given origin along a given angle.
//
void
P_Thrust
( player_t*      player,
  angle_t        angle,
  fixed_t        move )
{
    angle >>= ANGLETOFINESHIFT;

    player->mo->momx += FixedMul(move,finecosine[angle]);
    player->mo->momy += FixedMul(move,finesine[angle]);
}

//
// P_CalcHeight
// Calculate the walking / running height adjustment
//
void P_CalcHeight (player_t* player)
{
    int          angle;
    fixed_t      bob;

    // Regular movement bobbing
    // (needs to be calculated for gun swing
    // even if not on ground)
    // OPTIMIZE: tablify angle
    // Note: a LUT allows for effects
    // like a ramp with low health.
    player->bob =
        FixedMul (player->mo->momx, player->mo->momx)
        + FixedMul (player->mo->momy,player->mo->momy);

    player->bob >>= 2;

    if (player->bob>MAXBOB)

```

```

    player->bob = MAXBOB;

    if ((player->cheats & CF_NOMOMENTUM) || !onground)
    {
        player->viewz = player->mo->z + VIEWHEIGHT;

        if (player->viewz > player->mo->ceilingz-4*FRACUNIT)
            player->viewz = player->mo->ceilingz-4*FRACUNIT;

        player->viewz = player->mo->z + player->viewheight;
        return;
    }

    angle = (FINEANGLES/20*leveltime)&FINEMASK;
    bob = FixedMul ( player->bob/2, finesine[angle]);

    // move viewheight
    if (player->playerstate == PST_LIVE)
    {
        player->viewheight += player->deltaviewheight;

        if (player->viewheight > VIEWHEIGHT)
        {
            player->viewheight = VIEWHEIGHT;
            player->deltaviewheight = 0;
        }

        if (player->viewheight < VIEWHEIGHT/2)
        {
            player->viewheight = VIEWHEIGHT/2;
            if (player->deltaviewheight <= 0)
                player->deltaviewheight = 1;
        }

        if (player->deltaviewheight)
        {
            player->deltaviewheight += FRACUNIT/4;
            if (!player->deltaviewheight)
                player->deltaviewheight = 1;
        }
    }
    player->viewz = player->mo->z + player->viewheight + bob;

    if (player->viewz > player->mo->ceilingz-4*FRACUNIT)
        player->viewz = player->mo->ceilingz-4*FRACUNIT;
}

//
// P_MovePlayer
//
void P_MovePlayer (player_t* player)
{
    ticcmd_t*      cmd;

    cmd = &player->cmd;

    player->mo->angle += (cmd->angleturn<<16);

    // Do not let the player control movement
    // if not onground.
    onground = (player->mo->z <= player->mo->floorz);

```

```

    if (cmd->forwardmove && onground)
        P_Thrust (player, player->mo->angle, cmd->forwardmove*2048);

    if (cmd->sidemove && onground)
        P_Thrust (player, player->mo->angle-ANG90, cmd->sidemove*2048);

    if ( (cmd->forwardmove || cmd->sidemove)
        && player->mo->state == &states[S_PLAY] )
    {
        P_SetMobjState (player->mo, S_PLAY_RUN1);
    }
}

//
// P_DeathThink
// Fall on your face when dying.
// Decrease POV height to floor height.
//
#define ANG5          (ANG90/18)

void P_DeathThink (player_t* player)
{
    angle_t          angle;
    angle_t          delta;

    P_MovePsprites (player);

    // fall to the ground
    if (player->viewheight > 6*FRACUNIT)
        player->viewheight -= FRACUNIT;

    if (player->viewheight < 6*FRACUNIT)
        player->viewheight = 6*FRACUNIT;

    player->deltaviewheight = 0;
    onground = (player->mo->z <= player->mo->floorz);
    P_CalcHeight (player);

    if (player->attacker && player->attacker != player->mo)
    {
        angle = R_PointToAngle2 (player->mo->x,
                                player->mo->y,
                                player->attacker->x,
                                player->attacker->y);

        delta = angle - player->mo->angle;

        if (delta < ANG5 || delta > (unsigned)-ANG5)
        {
            // Looking at killer,
            // so fade damage flash down.
            player->mo->angle = angle;

            if (player->damagecount)
                player->damagecount--;
        }
        else if (delta < ANG180)
            player->mo->angle += ANG5;
        else
            player->mo->angle -= ANG5;
    }
    else if (player->damagecount)
        player->damagecount--;
}

```

```

    if (player->cmd.buttons & BT_USE)
        player->playerstate = PST_REBORN;
}

//
// P_PlayerThink
//
void P_PlayerThink (player_t* player)
{
    ticcmd_t*      cmd;
    weapontype_t   newweapon;

    // fixme: do this in the cheat code
    if (player->cheats & CF_NOCLIP)
        player->mo->flags |= MF_NOCLIP;
    else
        player->mo->flags &= ~MF_NOCLIP;

    // chain saw run forward
    cmd = &player->cmd;
    if (player->mo->flags & MF_JUSTATTACKED)
    {
        cmd->angleturn = 0;
        cmd->forwardmove = 0xc800/512;
        cmd->sidemove = 0;
        player->mo->flags &= ~MF_JUSTATTACKED;
    }

    if (player->playerstate == PST_DEAD)
    {
        P_DeathThink (player);
        return;
    }

    // Move around.
    // Reactiontime is used to prevent movement
    // for a bit after a teleport.
    if (player->mo->reactiontime)
        player->mo->reactiontime--;
    else
        P_MovePlayer (player);

    P_CalcHeight (player);

    if (player->mo->subsector->sector->special)
        P_PlayerInSpecialSector (player);

    // Check for weapon change.

    // A special event has no other buttons.
    if (cmd->buttons & BT_SPECIAL)
        cmd->buttons = 0;

    if (cmd->buttons & BT_CHANGE)
    {
        // The actual changing of the weapon is done
        // when the weapon psprite can do it
        // (read: not in the middle of an attack).
        newweapon = (cmd->buttons & BT_WEAPONMASK) >> BT_WEAPONSHIFT;
    }
}

```

```

    if (newweapon == wp_fist
        && player->weaponowned[wp_chainsaw]
        && !(player->readyweapon == wp_chainsaw
            && player->powers[pw_strength]))
    {
        newweapon = wp_chainsaw;
    }

    if ( (gamemode == commercial)
        && newweapon == wp_shotgun
        && player->weaponowned[wp_supershotgun]
        && player->readyweapon != wp_supershotgun)
    {
        newweapon = wp_supershotgun;
    }

    if (player->weaponowned[newweapon]
        && newweapon != player->readyweapon)
    {
        // Do not go to plasma or BFG in shareware,
        // even if cheated.
        if ((newweapon != wp_plasma
            && newweapon != wp_bfg)
            || (gamemode != shareware) )
        {
            player->pendingweapon = newweapon;
        }
    }
}

// check for use
if (cmd->buttons & BT_USE)
{
    if (!player->usedown)
    {
        P_UseLines (player);
        player->usedown = true;
    }
}
else
    player->usedown = false;

// cycle psprites
P_MovePsprites (player);

// Counters, time dependend power ups.

// Strength counts up to diminish fade.
if (player->powers[pw_strength])
    player->powers[pw_strength]++;

if (player->powers[pw_invulnerability])
    player->powers[pw_invulnerability]--;

if (player->powers[pw_invisibility])
    if (! --player->powers[pw_invisibility] )
        player->mo->flags &= ~MF_SHADOW;

if (player->powers[pw_infrared])
    player->powers[pw_infrared]--;

if (player->powers[pw_ironfeet])
    player->powers[pw_ironfeet]--;

```

```

if (player->damagecount)
    player->damagecount--;

if (player->bonuscount)
    player->bonuscount--;

// Handling colormaps.
if (player->powers[pw_invulnerability])
{
    if (player->powers[pw_invulnerability] > 4*32
        || (player->powers[pw_invulnerability]&8) )
        player->fixedcolormap = INVERSECOLORMAP;
    else
        player->fixedcolormap = 0;
}
else if (player->powers[pw_infrared])
{
    if (player->powers[pw_infrared] > 4*32
        || (player->powers[pw_infrared]&8) )
    {
        // almost full bright
        player->fixedcolormap = 1;
    }
    else
        player->fixedcolormap = 0;
}
else
    player->fixedcolormap = 0;
}

```

## 10 Rendering engine

### 10.1 r\_bsp.c

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     BSP traversal, handling of LineSegs for rendering.
//
//-----

static const char
rcsid[] = "$Id: r_bsp.c,v 1.4 1997/02/03 22:45:12 b1 Exp $";

```

```

#include "doomdef.h"

#include "m_bbox.h"

#include "i_system.h"

#include "r_main.h"
#include "r_plane.h"
#include "r_things.h"

// State.
#include "doomstat.h"
#include "r_state.h"

// #include "r_local.h"

seg_t*          curline;
side_t*         sidedef;
line_t*         linedef;
sector_t*       frontsector;
sector_t*       backsector;

drawseg_t       drawsegs[MAXDRAWSEGS];
drawseg_t*      ds_p;

void
R_StoreWallRange
( int          start,
  int          stop );

//
// R_ClearDrawSegs
//
void R_ClearDrawSegs (void)
{
    ds_p = drawsegs;
}

//
// ClipWallSegment
// Clips the given range of columns
// and includes it in the new clip list.
//
typedef          struct
{
    int          first;
    int          last;
} cliprange_t;

#define MAXSEGS          32

// newend is one past the last valid seg
cliprange_t*      newend;
cliprange_t       solidsegs[MAXSEGS];

```

```

//
// R_ClipSolidWallSegment
// Does handle solid walls,
// e.g. single sided LineDefs (middle texture)
// that entirely block the view.
//
void
R_ClipSolidWallSegment
( int                first,
  int                last )
{
    cliprange_t*      next;
    cliprange_t*      start;

    // Find the first range that touches the range
    // (adjacent pixels are touching).
    start = solidsegs;
    while (start->last < first-1)
        start++;

    if (first < start->first)
    {
        if (last < start->first-1)
        {
            // Post is entirely visible (above start),
            // so insert a new clippost.
            R_StoreWallRange (first, last);
            next = newend;
            newend++;

            while (next != start)
            {
                *next = *(next-1);
                next--;
            }
            next->first = first;
            next->last = last;
            return;
        }

        // There is a fragment above *start.
        R_StoreWallRange (first, start->first - 1);
        // Now adjust the clip size.
        start->first = first;
    }

    // Bottom contained in start?
    if (last <= start->last)
        return;

    next = start;
    while (last >= (next+1)->first-1)
    {
        // There is a fragment between two posts.
        R_StoreWallRange (next->last + 1, (next+1)->first - 1);
        next++;

        if (last <= next->last)
        {
            // Bottom is contained in next.
            // Adjust the clip size.

```



```

        start->last = next->last;
        goto crunch;
    }
}

// There is a fragment after *next.
R_StoreWallRange (next->last + 1, last);
// Adjust the clip size.
start->last = last;

// Remove start+1 to next from the clip list,
// because start now covers their area.
crunch:
    if (next == start)
    {
        // Post just extended past the bottom of one post.
        return;
    }

    while (next++ != newend)
    {
        // Remove a post.
        *++start = *next;
    }

    newend = start+1;
}

```

```

//
// R_ClipPassWallSegment
// Clips the given range of columns,
// but does not includes it in the clip list.
// Does handle windows,
// e.g. LineDefs with upper and lower texture.
//
void
R_ClipPassWallSegment
( int      first,
  int      last )
{
    cliprange_t*      start;

    // Find the first range that touches the range
    // (adjacent pixels are touching).
    start = solidsegs;
    while (start->last < first-1)
        start++;

    if (first < start->first)
    {
        if (last < start->first-1)
        {
            // Post is entirely visible (above start).
            R_StoreWallRange (first, last);
            return;
        }

        // There is a fragment above *start.
        R_StoreWallRange (first, start->first - 1);
    }

    // Bottom contained in start?

```

```

    if (last <= start->last)
        return;

    while (last >= (start+1)->first-1)
    {
        // There is a fragment between two posts.
        R_StoreWallRange (start->last + 1, (start+1)->first - 1);
        start++;

        if (last <= start->last)
            return;
    }

    // There is a fragment after *next.
    R_StoreWallRange (start->last + 1, last);
}

```

```

//
// R_ClearClipSegs
//
void R_ClearClipSegs (void)
{
    solidsegs[0].first = -0x7fffffff;
    solidsegs[0].last = -1;
    solidsegs[1].first = viewwidth;
    solidsegs[1].last = 0x7fffffff;
    newend = solidsegs+2;
}

//
// R_AddLine
// Clips the given segment
// and adds any visible pieces to the line list.
//
void R_AddLine (seg_t*      line)
{
    int                x1;
    int                x2;
    angle_t            angle1;
    angle_t            angle2;
    angle_t            span;
    angle_t            tspan;

    curline = line;

    // OPTIMIZE: quickly reject orthogonal back sides.
    angle1 = R_PointToAngle (line->v1->x, line->v1->y);
    angle2 = R_PointToAngle (line->v2->x, line->v2->y);

    // Clip to view edges.
    // OPTIMIZE: make constant out of 2*clipangle (FIELDOFVIEW).
    span = angle1 - angle2;

    // Back side? I.e. backface culling?
    if (span >= ANG180)
        return;

    // Global angle needed by segcalc.
    rw_angle1 = angle1;
    angle1 -= viewangle;
    angle2 -= viewangle;

    tspan = angle1 + clipangle;
}

```

```

if (tspan > 2*clipangle)
{
    tspan -= 2*clipangle;

    // Totally off the left edge?
    if (tspan >= span)
        return;

    angle1 = clipangle;
}
tspan = clipangle - angle2;
if (tspan > 2*clipangle)
{
    tspan -= 2*clipangle;

    // Totally off the left edge?
    if (tspan >= span)
        return;
    angle2 = -clipangle;
}

// The seg is in the view range,
// but not necessarily visible.
angle1 = (angle1+ANG90)>>ANGLETOFINESHIFT;
angle2 = (angle2+ANG90)>>ANGLETOFINESHIFT;
x1 = viewangletox[angle1];
x2 = viewangletox[angle2];

// Does not cross a pixel?
if (x1 == x2)
    return;

backsector = line->backsector;

// Single sided line?
if (!backsector)
    goto clipsolid;

// Closed door.
if (backsector->ceilingheight <= frontsector->floorheight
    || backsector->floorheight >= frontsector->ceilingheight)
    goto clipsolid;

// Window.
if (backsector->ceilingheight != frontsector->ceilingheight
    || backsector->floorheight != frontsector->floorheight)
    goto clippass;

// Reject empty lines used for triggers
// and special events.
// Identical floor and ceiling on both sides,
// identical light levels on both sides,
// and no middle texture.
if (backsector->ceilingpic == frontsector->ceilingpic
    && backsector->floorpic == frontsector->floorpic
    && backsector->lightlevel == frontsector->lightlevel
    && curline->sidedef->midtexture == 0)
{
    return;
}

clippass:
R_ClipPassWallSegment (x1, x2-1);
return;

```

```

clipsolid:
    R_ClipSolidWallSegment (x1, x2-1);
}

//
// R_CheckBBox
// Checks BSP node/subtree bounding box.
// Returns true
// if some part of the bbox might be visible.
//
int      checkcoord[12][4] =
{
    {3,0,2,1},
    {3,0,2,0},
    {3,1,2,0},
    {0},
    {2,0,2,1},
    {0,0,0,0},
    {3,1,3,0},
    {0},
    {2,0,3,1},
    {2,1,3,1},
    {2,1,3,0}
};

boolean R_CheckBBox (fixed_t*      bspcoord)
{
    int      boxx;
    int      boxy;
    int      boxpos;

    fixed_t  x1;
    fixed_t  y1;
    fixed_t  x2;
    fixed_t  y2;

    angle_t  angle1;
    angle_t  angle2;
    angle_t  span;
    angle_t  tspan;

    cliprange_t*  start;

    int      sx1;
    int      sx2;

    // Find the corners of the box
    // that define the edges from current viewpoint.
    if (viewx <= bspcoord[BOXLEFT])
        boxx = 0;
    else if (viewx < bspcoord[BOXRIGHT])
        boxx = 1;
    else
        boxx = 2;

    if (viewy >= bspcoord[BOXTOP])
        boxy = 0;
    else if (viewy > bspcoord[BOXBOTTOM])
        boxy = 1;
    else
        boxy = 2;

```

```

boxpos = (boxy<<2)+boxx;
if (boxpos == 5)
    return true;

x1 = bspcoord[checkcoord[boxpos][0]];
y1 = bspcoord[checkcoord[boxpos][1]];
x2 = bspcoord[checkcoord[boxpos][2]];
y2 = bspcoord[checkcoord[boxpos][3]];

// check clip list for an open space
angle1 = R_PointToAngle (x1, y1) - viewangle;
angle2 = R_PointToAngle (x2, y2) - viewangle;

span = angle1 - angle2;

// Sitting on a line?
if (span >= ANG180)
    return true;

tspan = angle1 + clipangle;

if (tspan > 2*clipangle)
{
    tspan -= 2*clipangle;

    // Totally off the left edge?
    if (tspan >= span)
        return false;

    angle1 = clipangle;
}
tspan = clipangle - angle2;
if (tspan > 2*clipangle)
{
    tspan -= 2*clipangle;

    // Totally off the left edge?
    if (tspan >= span)
        return false;

    angle2 = -clipangle;
}

// Find the first clippost
// that touches the source post
// (adjacent pixels are touching).
angle1 = (angle1+ANG90)>>ANGLETOFINESHIFT;
angle2 = (angle2+ANG90)>>ANGLETOFINESHIFT;
sx1 = viewangletox[angle1];
sx2 = viewangletox[angle2];

// Does not cross a pixel.
if (sx1 == sx2)
    return false;
sx2--;

start = solidsegs;
while (start->last < sx2)
    start++;

if (sx1 >= start->first
    && sx2 <= start->last)
{
    // The clippost contains the new span.

```

```

        return false;
    }

    return true;
}

//
// R_Subsector
// Determine floor/ceiling planes.
// Add sprites of things in sector.
// Draw one or more line segments.
//
void R_Subsector (int num)
{
    int                count;
    seg_t*             line;
    subsector_t*       sub;

#ifdef RANGECHECK
    if (num>=numsubsectors)
        I_Error ("R_Subsector: ss %i with numss = %i",
                num,
                numsubsectors);
#endif

    sscount++;
    sub = &subsectors[num];
    frontsector = sub->sector;
    count = sub->numlines;
    line = &segs[sub->firstline];

    if (frontsector->floorheight < viewz)
    {
        floorplane = R_FindPlane (frontsector->floorheight,
                                   frontsector->floorpic,
                                   frontsector->lightlevel);
    }
    else
        floorplane = NULL;

    if (frontsector->ceilingheight > viewz
        || frontsector->ceilingpic == skyflatnum)
    {
        ceilingplane = R_FindPlane (frontsector->ceilingheight,
                                     frontsector->ceilingpic,
                                     frontsector->lightlevel);
    }
    else
        ceilingplane = NULL;

    R_AddSprites (frontsector);

    while (count--)
    {
        R_AddLine (line);
        line++;
    }
}

//

```

```

// RenderBSPNode
// Renders all subsectors below a given node,
// traversing subtree recursively.
// Just call with BSP root.
void R_RenderBSPNode (int bspnum)
{
    node_t*      bsp;
    int          side;

    // Found a subsector?
    if (bspnum & NF_SUBSECTOR)
    {
        if (bspnum == -1)
            R_Subsector (0);
        else
            R_Subsector (bspnum & (~NF_SUBSECTOR));
        return;
    }

    bsp = &nodes[bspnum];

    // Decide which side the view point is on.
    side = R_PointOnSide (viewx, viewy, bsp);

    // Recursively divide front space.
    R_RenderBSPNode (bsp->children[side]);

    // Possibly divide back space.
    if (R_CheckBBox (bsp->bbox[side^1]))
        R_RenderBSPNode (bsp->children[side^1]);
}

```

## 10.2 r\_bsp.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Refresh module, BSP traversal and handling.
//
//-----

#ifndef __R_BSP__
#define __R_BSP__

#ifdef __GNUG__
#pragma interface
#endif

```

```

extern seg_t*          curline;
extern side_t*         sidedef;
extern line_t*         linedef;
extern sector_t*       frontsector;
extern sector_t*       backsector;

extern int             rw_x;
extern int             rw_stopx;

extern boolean         segtextured;

// false if the back side is the same plane
extern boolean         markfloor;
extern boolean         markceiling;

extern boolean         skymap;

extern drawseg_t       drawsegs[MAXDRAWSEGS];
extern drawseg_t*      ds_p;

extern lighttable_t**  hscalelight;
extern lighttable_t**  vscalelight;
extern lighttable_t**  dscalelight;

typedef void (*drawfunc_t) (int start, int stop);

// BSP?
void R_ClearClipSegs (void);
void R_ClearDrawSegs (void);

```

```

void R_RenderBSPNode (int bspnum);

```

```

#endif
//-----
//
// $Log:$
//
//-----

```

### 10.3 r\_data.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$

```



```

//
// Revision 1.3 1997/01/29 20:10
// DESCRIPTION:
//      Preparation of data for rendering,
//      generation of lookups, caching, retrieval by name.
//
//-----

static const char
rcsid[] = "$Id: r_data.c,v 1.4 1997/02/03 16:47:55 b1 Exp $";

#include "i_system.h"
#include "z_zone.h"

#include "m_swap.h"

#include "w_wad.h"

#include "doomdef.h"
#include "r_local.h"
#include "p_local.h"

#include "doomstat.h"
#include "r_sky.h"

#ifdef LINUX
#include <alloca.h>
#endif

#include "r_data.h"

//
// Graphics.
// DOOM graphics for walls and sprites
// is stored in vertical runs of opaque pixels (posts).
// A column is composed of zero or more posts,
// a patch or sprite is composed of zero or more columns.
//

//
// Texture definition.
// Each texture is composed of one or more patches,
// with patches being lumps stored in the WAD.
// The lumps are referenced by number, and patched
// into the rectangular texture space using origin
// and possibly other attributes.
//
typedef struct
{
    short      originx;
    short      originy;
    short      patch;
    short      stepdir;
    short      colormap;
} mappatch_t;

//
// Texture definition.
// A DOOM wall texture is a list of patches
// which are to be combined in a predefined order.

```

```
//
typedef struct
{
    char            name[8];
    boolean         masked;
    short           width;
    short           height;
    void            **columnndirectory;    // OBSOLETE
    short           patchcount;
    mappatch_t      patches[1];
} maptexture_t;
```

```
// A single patch from a texture definition,
// basically a rectangular area within
// the texture rectangle.
```

```
typedef struct
{
    // Block origin (allways UL),
    // which has allready accounted
    // for the internal origin of the patch.
    int             originx;
    int             originy;
    int             patch;
} texpatch_t;
```

```
// A maptexturedef_t describes a rectangular texture,
// which is composed of one or more mappatch_t structures
// that arrange graphic patches.
```

```
typedef struct
{
    // Keep name for switch changing, etc.
    char            name[8];
    short           width;
    short           height;

    // All the patches[patchcount]
    // are drawn back to front into the cached texture.
    short           patchcount;
    texpatch_t      patches[1];
} texture_t;
```

```
int             firstflat;
int             lastflat;
int             numflats;
```

```
int             firstpatch;
int             lastpatch;
int             numpatches;
```

```
int             firstspritelump;
int             lastspritelump;
int             numspritelumps;
```

```
int             numtextures;
texture_t**     textures;
```

```
int*            texturewidthmask;
// needed for texture pegging
fixed_t*        textureheight;
```

```

int*                texturecompositesize;
short**             texturecolumnlump;
unsigned short**    texturecolumnofs;
byte**              texturecomposite;

// for global animation
int*                flattranslation;
int*                texturetranslation;

// needed for pre rendering
fixed_t*            spritewidth;
fixed_t*            spriteoffset;
fixed_t*            spritetopoffset;

lighttable_t        *colormaps;

//
// MAPTEXTURE_T CACHING
// When a texture is first needed,
// it counts the number of composite columns
// required in the texture and allocates space
// for a column directory and any new columns.
// The directory will simply point inside other patches
// if there is only one patch in a given column,
// but any columns with multiple patches
// will have new column_ts generated.
//

//
// R_DrawColumnInCache
// Clip and draw a column
// from a patch into a cached post.
//
void
R_DrawColumnInCache
( column_t*          patch,
  byte*              cache,
  int                 originy,
  int                 cacheheight )
{
    int               count;
    int               position;
    byte*             source;
    byte*             dest;

    dest = (byte *)cache + 3;

    while (patch->topdelta != 0xff)
    {
        source = (byte *)patch + 3;
        count = patch->length;
        position = originy + patch->topdelta;

        if (position < 0)
        {
            count += position;
            position = 0;
        }

        if (position + count > cacheheight)
            count = cacheheight - position;
    }
}

```

```

        if (count > 0)
            memcpy (cache + position, source, count);

        patch = (column_t *) ( (byte *)patch + patch->length + 4);
    }
}

//
// R_GenerateComposite
// Using the texture definition,
// the composite texture is created from the patches,
// and each column is cached.
//
void R_GenerateComposite (int texnum)
{
    byte*          block;
    texture_t*      texture;
    texpatch_t*     patch;
    patch_t*        realpatch;
    int             x;
    int             x1;
    int             x2;
    int             i;
    column_t*       patchcol;
    short*          collump;
    unsigned short* colofs;

    texture = textures[texnum];

    block = Z_Malloc (texturecompositesize[texnum],
                     PU_STATIC,
                     &texturecomposite[texnum]);

    collump = texturecolumnlump[texnum];
    colofs = texturecolumnofs[texnum];

    // Composite the columns together.
    patch = texture->patches;

    for (i=0 , patch = texture->patches;
         i<texture->patchcount;
         i++, patch++)
    {
        realpatch = W_CacheLumpNum (patch->patch, PU_CACHE);
        x1 = patch->originx;
        x2 = x1 + SHORT(realpatch->width);

        if (x1<0)
            x = 0;
        else
            x = x1;

        if (x2 > texture->width)
            x2 = texture->width;

        for ( ; x<x2 ; x++)
        {
            // Column does not have multiple patches?
            if (collump[x] >= 0)
                continue;

            patchcol = (column_t *) ((byte *)realpatch
                                     + LONG(realpatch->columnofs[x-x1]));
        }
    }
}

```

```

        R_DrawColumnInCache (patchcol,
                             block + colofs[x],
                             patch->originy,
                             texture->height);
    }

}

// Now that the texture has been built in column cache,
// it is purgable from zone memory.
Z_ChangeTag (block, PU_CACHE);
}

//
// R_GenerateLookup
//
void R_GenerateLookup (int texnum)
{
    texture_t*          texture;
    byte*               patchcount;          // patchcount[texture->width]
    texpatch_t*         patch;
    patch_t*            realpatch;
    int                 x;
    int                 x1;
    int                 x2;
    int                 i;
    short*              collump;
    unsigned short*     colofs;

    texture = textures[texnum];

    // Composited texture not created yet.
    texturecomposite[texnum] = 0;

    texturecompositesize[texnum] = 0;
    collump = texturecolumnlump[texnum];
    colofs = texturecolumnofs[texnum];

    // Now count the number of columns
    // that are covered by more than one patch.
    // Fill in the lump / offset, so columns
    // with only a single patch are all done.
    patchcount = (byte *)alloca (texture->width);
    memset (patchcount, 0, texture->width);
    patch = texture->patches;

    for (i=0 , patch = texture->patches;
         i<texture->patchcount;
         i++, patch++)
    {
        realpatch = W_CacheLumpNum (patch->patch, PU_CACHE);
        x1 = patch->originx;
        x2 = x1 + SHORT(realpatch->width);

        if (x1 < 0)
            x = 0;
        else
            x = x1;

        if (x2 > texture->width)
            x2 = texture->width;
        for ( ; x<x2 ; x++)
        {

```

```

        patchcount[x]++;
        collump[x] = patch->patch;
        colofs[x] = LONG(realpatch->columnofs[x-x1])+3;
    }
}

for (x=0 ; x<texture->width ; x++)
{
    if (!patchcount[x])
    {
        printf ("R_GenerateLookup: column without a patch (%s)\n",
            texture->name);
        return;
    }
    // I_Error ("R_GenerateLookup: column without a patch");

    if (patchcount[x] > 1)
    {
        // Use the cached block.
        collump[x] = -1;
        colofs[x] = texturecompositesize[texture->texnum];

        if (texturecompositesize[texture->texnum] > 0x10000-texture->height)
        {
            I_Error ("R_GenerateLookup: texture %i is >64k",
                texture->texnum);
        }

        texturecompositesize[texture->texnum] += texture->height;
    }
}
}

```

```

//
// R_GetColumn
//
byte*
R_GetColumn
( int          tex,
  int          col )
{
    int          lump;
    int          ofs;

    col &= texturewidthmask[tex];
    lump = texturecolumnlump[tex][col];
    ofs = texturecolumnofs[tex][col];

    if (lump > 0)
        return (byte *)W_CacheLumpNum(lump,PU_CACHE)+ofs;

    if (!texturecomposite[tex])
        R_GenerateComposite (tex);

    return texturecomposite[tex] + ofs;
}

```

```

//
// R_InitTextures

```

```

// Initializes the texture list
// with the textures from the world map.
//
void R_InitTextures (void)
{
    maptexture_t*      mtexture;
    texture_t*         texture;
    mappatch_t*        mpatch;
    texpatch_t*        patch;

    int                i;
    int                j;

    int*               maptex;
    int*               maptex2;
    int*               maptex1;

    char               name[9];
    char*              names;
    char*              name_p;

    int*               patchlookup;

    int                totalwidth;
    int                nummappatches;
    int                offset;
    int                maxoff;
    int                maxoff2;
    int                numtextures1;
    int                numtextures2;

    int*               directory;

    int                temp1;
    int                temp2;
    int                temp3;

    // Load the patch names from pnames.lmp.
    name[8] = 0;
    names = W_CacheLumpName ("PNAMES", PU_STATIC);
    nummappatches = LONG ( *((int *)names) );
    name_p = names+4;
    patchlookup = alloca (nummappatches*sizeof(*patchlookup));

    for (i=0 ; i<nummappatches ; i++)
    {
        strncpy (name,name_p+i*8, 8);
        patchlookup[i] = W_CheckNumForName (name);
    }
    Z_Free (names);

    // Load the map texture definitions from textures.lmp.
    // The data is contained in one or two lumps,
    // TEXTURE1 for shareware, plus TEXTURE2 for commercial.
    maptex = maptex1 = W_CacheLumpName ("TEXTURE1", PU_STATIC);
    numtextures1 = LONG(*maptex);
    maxoff = W_LumpLength (W_GetNumForName ("TEXTURE1"));
    directory = maptex+1;

    if (W_CheckNumForName ("TEXTURE2") != -1)
    {
        maptex2 = W_CacheLumpName ("TEXTURE2", PU_STATIC);
        numtextures2 = LONG(*maptex2);
        maxoff2 = W_LumpLength (W_GetNumForName ("TEXTURE2"));
    }
}

```

```

}
else
{
    maptex2 = NULL;
    numtextures2 = 0;
    maxoff2 = 0;
}
numtextures = numtextures1 + numtextures2;

textures = Z_Malloc (numtextures*4, PU_STATIC, 0);
texturecolumnlump = Z_Malloc (numtextures*4, PU_STATIC, 0);
texturecolumnofs = Z_Malloc (numtextures*4, PU_STATIC, 0);
texturecomposite = Z_Malloc (numtextures*4, PU_STATIC, 0);
texturecompositesize = Z_Malloc (numtextures*4, PU_STATIC, 0);
texturewidthmask = Z_Malloc (numtextures*4, PU_STATIC, 0);
textureheight = Z_Malloc (numtextures*4, PU_STATIC, 0);

totalwidth = 0;

//      Really complex printing shit...
temp1 = W_GetNumForName ("S_START"); // P_??????
temp2 = W_GetNumForName ("S_END") - 1;
temp3 = ((temp2-temp1+63)/64) + ((numtextures+63)/64);
printf("[");
for (i = 0; i < temp3; i++)
    printf(" ");
printf("      ]");
for (i = 0; i < temp3; i++)
    printf("\x8");
printf("\x8\x8\x8\x8\x8\x8\x8\x8\x8\x8");

for (i=0 ; i<numtextures ; i++, directory++)
{
    if (!(i&63))
        printf (".");

    if (i == numtextures1)
    {
        // Start looking in second texture file.
        maptex = maptex2;
        maxoff = maxoff2;
        directory = maptex+1;
    }

    offset = LONG(*directory);

    if (offset > maxoff)
        I_Error ("R_InitTextures: bad texture directory");

    mtexture = (maptexture_t *) ( (byte *)maptex + offset);

    texture = textures[i] =
        Z_Malloc (sizeof(texture_t)
            + sizeof(texpatch_t)*(SHORT(mtexture->patchcount)-1),
            PU_STATIC, 0);

    texture->width = SHORT(mtexture->width);
    texture->height = SHORT(mtexture->height);
    texture->patchcount = SHORT(mtexture->patchcount);

    memcpy (texture->name, mtexture->name, sizeof(texture->name));
    mpatch = &mtexture->patches[0];
    patch = &texture->patches[0];

    for (j=0 ; j<texture->patchcount ; j++, mpatch++, patch++)

```



```

{
    patch->originx = SHORT(mpatch->originx);
    patch->originy = SHORT(mpatch->originy);
    patch->patch = patchlookup[SHORT(mpatch->patch)];
    if (patch->patch == -1)
    {
        I_Error ("R_InitTextures: Missing patch in texture %s",
                texture->name);
    }
}
texturecolumnlump[i] = Z_Malloc (texture->width*2, PU_STATIC,0);
texturecolumnofs[i] = Z_Malloc (texture->width*2, PU_STATIC,0);

j = 1;
while (j*2 <= texture->width)
    j<<=1;

texturewidthmask[i] = j-1;
textureheight[i] = texture->height<<FRACBITS;

totalwidth += texture->width;
}

Z_Free (maptex1);
if (maptex2)
    Z_Free (maptex2);

// Precalculate whatever possible.
for (i=0 ; i<numtextures ; i++)
    R_GenerateLookup (i);

// Create translation table for global animation.
texturetranslation = Z_Malloc ((numtextures+1)*4, PU_STATIC, 0);

for (i=0 ; i<numtextures ; i++)
    texturetranslation[i] = i;
}

//
// R_InitFlats
//
void R_InitFlats (void)
{
    int i;

    firstflat = W_GetNumForName ("F_START") + 1;
    lastflat = W_GetNumForName ("F_END") - 1;
    numflats = lastflat - firstflat + 1;

    // Create translation table for global animation.
    flattranslation = Z_Malloc ((numflats+1)*4, PU_STATIC, 0);

    for (i=0 ; i<numflats ; i++)
        flattranslation[i] = i;
}

//
// R_InitSpriteLumps
// Finds the width and hoffset of all sprites in the wad,
// so the sprite does not need to be cached completely
// just for having the header info ready during rendering.
//

```

```

void R_InitSpriteLumps (void)
{
    int            i;
    patch_t        *patch;

    firstspritelump = W_GetNumForName ("S_START") + 1;
    lastspritelump = W_GetNumForName ("S_END") - 1;

    numspritelumps = lastspritelump - firstspritelump + 1;
    spritewidth = Z_Malloc (numspritelumps*4, PU_STATIC, 0);
    spriteoffset = Z_Malloc (numspritelumps*4, PU_STATIC, 0);
    spritetopoffset = Z_Malloc (numspritelumps*4, PU_STATIC, 0);

    for (i=0 ; i< numspritelumps ; i++)
    {
        if (!(i&63))
            printf (".");

        patch = W_CacheLumpNum (firstspritelump+i, PU_CACHE);
        spritewidth[i] = SHORT(patch->width)<<FRACBITS;
        spriteoffset[i] = SHORT(patch->leftoffset)<<FRACBITS;
        spritetopoffset[i] = SHORT(patch->topoffset)<<FRACBITS;
    }
}

```

```

//
// R_InitColormaps
//
void R_InitColormaps (void)
{
    int            lump, length;

    // Load in the light tables,
    // 256 byte align tables.
    lump = W_GetNumForName("COLORMAP");
    length = W_LumpLength (lump) + 255;
    colormaps = Z_Malloc (length, PU_STATIC, 0);
    colormaps = (byte *)(((int)colormaps + 255)&~0xff);
    W_ReadLump (lump,colormaps);
}

```

```

//
// R_InitData
// Locates all the lumps
// that will be used by all views
// Must be called after W_Init.
//
void R_InitData (void)
{
    R_InitTextures ();
    printf ("\nInitTextures");
    R_InitFlats ();
    printf ("\nInitFlats");
    R_InitSpriteLumps ();
    printf ("\nInitSprites");
    R_InitColormaps ();
    printf ("\nInitColormaps");
}

```

```

//
// R_FlatNumForName
// Retrieval, get a flat number for a flat name.
//
int R_FlatNumForName (char* name)
{
    int            i;
    char           namet[9];

    i = W_CheckNumForName (name);

    if (i == -1)
    {
        namet[8] = 0;
        memcpy (namet, name,8);
        I_Error ("R_FlatNumForName: %s not found",namet);
    }
    return i - firstflat;
}

```

```

//
// R_CheckTextureNumForName
// Check whether texture is available.
// Filter out NoTexture indicator.
//
int      R_CheckTextureNumForName (char *name)
{
    int            i;

    // "NoTexture" marker.
    if (name[0] == '-')
        return 0;

    for (i=0 ; i<numtextures ; i++)
        if (!strncasecmp (textures[i]->name, name, 8) )
            return i;

    return -1;
}

```

```

//
// R_TextureNumForName
// Calls R_CheckTextureNumForName,
// aborts with error message.
//
int      R_TextureNumForName (char* name)
{
    int            i;

    i = R_CheckTextureNumForName (name);

    if (i== -1)
    {
        I_Error ("R_TextureNumForName: %s not found",
                name);
    }
    return i;
}

```

```

//
// R_PrecacheLevel
// Preloads all relevant graphics for the level.
//
int          flatmemory;
int          texturememory;
int          spritememory;

void R_PrecacheLevel (void)
{
    char*          flatpresent;
    char*          texturepresent;
    char*          spritepresent;

    int            i;
    int            j;
    int            k;
    int            lump;

    texture_t*      texture;
    thinker_t*      th;
    spriteframe_t*  sf;

    if (demoplayback)
        return;

    // Precache flats.
    flatpresent = alloca(numflats);
    memset (flatpresent,0,numflats);

    for (i=0 ; i<numsectors ; i++)
    {
        flatpresent[sectors[i].floorpic] = 1;
        flatpresent[sectors[i].ceilingpic] = 1;
    }

    flatmemory = 0;

    for (i=0 ; i<numflats ; i++)
    {
        if (flatpresent[i])
        {
            lump = firstflat + i;
            flatmemory += lumpinfo[lump].size;
            W_CacheLumpNum(lump, PU_CACHE);
        }
    }

    // Precache textures.
    texturepresent = alloca(numtextures);
    memset (texturepresent,0, numtextures);

    for (i=0 ; i<numsides ; i++)
    {
        texturepresent[sides[i].toptexture] = 1;
        texturepresent[sides[i].midtexture] = 1;
        texturepresent[sides[i].bottomtexture] = 1;
    }

    // Sky texture is always present.
    // Note that F_SKY1 is the name used to
    // indicate a sky floor/ceiling as a flat,
    // while the sky texture is stored like

```

```

// a wall texture, with an episode dependend
// name.
texturepresent[skytexture] = 1;

texturememory = 0;
for (i=0 ; i<numtextures ; i++)
{
    if (!texturepresent[i])
        continue;

    texture = textures[i];

    for (j=0 ; j<texture->patchcount ; j++)
    {
        lump = texture->patches[j].patch;
        texturememory += lumpinfo[lump].size;
        W_CacheLumpNum(lump , PU_CACHE);
    }
}

// Precache sprites.
spritepresent = alloca(numsprites);
memset (spritepresent,0, numsprites);

for (th = thinkercap.next ; th != &thinkercap ; th=th->next)
{
    if (th->function.acp1 == (actionf_p1)P_MobjThinker)
        spritepresent[((mobj_t *)th)->sprite] = 1;
}

spritememory = 0;
for (i=0 ; i<numsprites ; i++)
{
    if (!spritepresent[i])
        continue;

    for (j=0 ; j<sprites[i].numframes ; j++)
    {
        sf = &sprites[i].spriteframes[j];
        for (k=0 ; k<8 ; k++)
        {
            lump = firstspritelump + sf->lump[k];
            spritememory += lumpinfo[lump].size;
            W_CacheLumpNum(lump , PU_CACHE);
        }
    }
}
}

```

## 10.4 r\_data.h

```

// Emacs style mode select    -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2

```

```

// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
// Refresh module, data I/O, caching, retrieval of graphics
// by name.
//
//-----

#ifndef __R_DATA__
#define __R_DATA__

#include "r_defs.h"
#include "r_state.h"

#ifdef __GNUG__
#pragma interface
#endif

// Retrieve column data for span blitting.
byte*
R_GetColumn
( int          tex,
  int          col );

// I/O, setting up the stuff.
void R_InitData (void);
void R_PrecacheLevel (void);

// Retrieval.
// Floor/ceiling opaque texture tiles,
// lookup by name. For animation?
int R_FlatNumForName (char* name);

// Called by P_Ticker for switches and animations,
// returns the texture number for the texture name.
int R_TextureNumForName (char *name);
int R_CheckTextureNumForName (char *name);

#endif
//-----
//
// $Log:$
//
//-----

```

## 10.5 r\_defs.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License

```

```

// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
// Refresh/rendering module, shared data struct definitions.
//
//-----

#ifndef __R_DEFS__
#define __R_DEFS__

// Screenwidth.
#include "doomdef.h"

// Some more or less basic data types
// we depend on.
#include "m_fixed.h"

// We rely on the thinker data struct
// to handle sound origins in sectors.
#include "d_think.h"
// SECTORS do store MObjs anyway.
#include "p_mobj.h"

#ifdef __GNUG__
#pragma interface
#endif

// Silhouette, needed for clipping Segs (mainly)
// and sprites representing things.
#define SIL_NONE 0
#define SIL_BOTTOM 1
#define SIL_TOP 2
#define SIL_BOTH 3

#define MAXDRAWSEGS 256

//
// INTERNAL MAP TYPES
// used by play and refresh
//
//
// Your plain vanilla vertex.
// Note: transformed values not buffered locally,
// like some DOOM-alikes ("wt", "WebView") did.
//
typedef struct
{
    fixed_t x;

```

```

        fixed_t          y;

} vertex_t;

// Forward of LineDefs, for Sectors.
struct line_s;

// Each sector has a degenmobj_t in its center
// for sound origin purposes.
// I suppose this does not handle sound from
// moving objects (doppler), because
// position is prolly just buffered, not
// updated.
typedef struct
{
    thinker_t          thinker;          // not used for anything
    fixed_t            x;
    fixed_t            y;
    fixed_t            z;

} degenmobj_t;

//
// The SECTORS record, at runtime.
// Stores things/mobj_s.
//
typedef          struct
{
    fixed_t        floorheight;
    fixed_t        ceilingheight;
    short          floorpic;
    short          ceilingpic;
    short          lightlevel;
    short          special;
    short          tag;

    // 0 = untraversed, 1,2 = sndlines -1
    int            soundtraversed;

    // thing that made a sound (or null)
    mobj_t*        soundtarget;

    // mapblock bounding box for height changes
    int            blockbox[4];

    // origin for any sounds played by the sector
    degenmobj_t    soundorg;

    // if == validcount, already checked
    int            validcount;

    // list of mobj_s in sector
    mobj_t*        thinglist;

    // thinker_t for reversable actions
    void*          specialdata;

    int            linecount;
    struct line_s** lines;          // [linecount] size

} sector_t;

```



```

//
// The SideDef.
//

typedef struct
{
    // add this to the calculated texture column
    fixed_t      textureoffset;

    // add this to the calculated texture top
    fixed_t      rowoffset;

    // Texture indices.
    // We do not maintain names here.
    short        toptexture;
    short        bottomtexture;
    short        midtexture;

    // Sector the SideDef is facing.
    sector_t*     sector;
} side_t;

//
// Move clipping aid for LineDefs.
//
typedef enum
{
    ST_HORIZONTAL,
    ST_VERTICAL,
    ST_POSITIVE,
    ST_NEGATIVE
} slopetype_t;

typedef struct line_s
{
    // Vertices, from v1 to v2.
    vertex_t*     v1;
    vertex_t*     v2;

    // Precalculated v2 - v1 for side checking.
    fixed_t      dx;
    fixed_t      dy;

    // Animation related.
    short        flags;
    short        special;
    short        tag;

    // Visual appearance: SideDefs.
    // sidenum[1] will be -1 if one sided
    short        sidenum[2];

    // Neat. Another bounding box, for the extent
    // of the LineDef.
    fixed_t      bbox[4];

    // To aid move clipping.
    slopetype_t   slopetype;
}

```

```

// Front and back sector.
// Note: redundant? Can be retrieved from SideDefs.
sector_t*      frontsector;
sector_t*      backsector;

// if == validcount, already checked
int            validcount;

// thinker_t for reversible actions
void*          specialdata;
} line_t;

```

```

//
// A SubSector.
// References a Sector.
// Basically, this is a list of LineSegs,
// indicating the visible walls that define
// (all or some) sides of a convex BSP leaf.
//
typedef struct subsector_s
{
    sector_t*      sector;
    short          numlines;
    short          firstline;
} subsector_t;

```

```

//
// The LineSeg.
//
typedef struct
{
    vertex_t*      v1;
    vertex_t*      v2;

    fixed_t        offset;

    angle_t        angle;

    side_t*        sidedef;
    line_t*        linedef;

    // Sector references.
    // Could be retrieved from linedef, too.
    // backsector is NULL for one sided lines
    sector_t*      frontsector;
    sector_t*      backsector;
} seg_t;

```

```

//
// BSP node.
//
typedef struct
{
    // Partition line.
    fixed_t        x;

```

```

    fixed_t      y;
    fixed_t      dx;
    fixed_t      dy;

    // Bounding box for each child.
    fixed_t      bbox[2][4];

    // If NF_SUBSECTOR its a subsector.
    unsigned short children[2];

} node_t;


// posts are runs of non masked source pixels
typedef struct
{
    byte          topdelta;      // -1 is the last post in a column
    byte          length;        // length data bytes follows
} post_t;

// column_t is a list of 0 or more post_t, (byte)-1 terminated
typedef post_t   column_t;


// PC direct to screen pointers
//B UNUSED - keep till detailshift in r_draw.c resolved
//extern byte*    destview;
//extern byte*    destscreen;


//
// OTHER TYPES
//

// This could be wider for >8 bit display.
// Indeed, true color support is possible
// precalculating 24bpp lightmap/colormap LUT.
// from darkening PLAYPAL to all black.
// Could even use more than 32 levels.
typedef byte     lighttable_t;


//
// ?
//
typedef struct drawseg_s
{
    seg_t*        curline;
    int           x1;
    int           x2;

    fixed_t       scale1;
    fixed_t       scale2;
    fixed_t       scalestep;

    // 0=none, 1=bottom, 2=top, 3=both
    int           silhouette;

```

```

// do not clip sprites above this
fixed_t          bsilheight;

// do not clip sprites below this
fixed_t          tsilheight;

// Pointers to lists for sprite clipping,
// all three adjusted so [x1] is first value.
short*           sprtopclip;
short*           sprbottomclip;
short*           maskedtexturecol;

} drawseg_t;


// Patches.
// A patch holds one or more columns.
// Patches are used for sprites and all masked pictures,
// and we compose textures from the TEXTURE1/2 lists
// of patches.
typedef struct
{
    short          width;           // bounding box size
    short          height;
    short          leftoffset;      // pixels to the left of origin
    short          topoffset;       // pixels below the origin
    int            columnofs[8];    // only [width] used
    // the [0] is &columnofs[width]
} patch_t;


// A vissprite_t is a thing
// that will be drawn during a refresh.
// I.e. a sprite object that is partly visible.
typedef struct vissprite_s
{
    // Doubly linked list.
    struct vissprite_s* prev;
    struct vissprite_s* next;

    int            x1;
    int            x2;

    // for line side calculation
    fixed_t        gx;
    fixed_t        gy;

    // global bottom / top for silhouette clipping
    fixed_t        gz;
    fixed_t        gzt;

    // horizontal position of x1
    fixed_t        startfrac;

    fixed_t        scale;

    // negative if flipped
    fixed_t        xiscale;

```

```

    fixed_t            texturemid;
    int                patch;

    // for color translation and shadow draw,
    // maxbright frames as well
    lighttable_t*      colormap;

    int                mobjflags;
} vissprite_t;

//
// Sprites are patches with a special naming convention
// so they can be recognized by R_InitSprites.
// The base name is NNNNFx or NNNNFxFx, with
// x indicating the rotation, x = 0, 1-7.
// The sprite and frame specified by a thing_t
// is range checked at run time.
// A sprite is a patch_t that is assumed to represent
// a three dimensional object and may have multiple
// rotations pre drawn.
// Horizontal flipping is used to save space,
// thus NNNNF2F5 defines a mirrored patch.
// Some sprites will only have one picture used
// for all views: NNNNF0
//
typedef struct
{
    // If false use 0 for any position.
    // Note: as eight entries are available,
    // we might as well insert the same name eight times.
    boolean            rotate;

    // Lump to use for view angles 0-7.
    short              lump[8];

    // Flip bit (1 = flip) to use for view angles 0-7.
    byte               flip[8];
} spriteframe_t;

//
// A sprite definition:
// a number of animation frames.
//
typedef struct
{
    int                numframes;
    spriteframe_t*     spriteframes;
} spritedef_t;

//
// Now what is a visplane, anyway?
//
typedef struct
{
    fixed_t            height;
    int                picnum;

```

```

int                lightlevel;
int                minx;
int                maxx;

// leave pads for [minx-1]/[maxx+1]

byte               pad1;
// Here lies the rub for all
// dynamic resize/change of resolution.
byte               top[SCREENWIDTH];
byte               pad2;
byte               pad3;
// See above.
byte               bottom[SCREENWIDTH];
byte               pad4;

} visplane_t;

#endif
//-----
//
// $Log:$
//
//-----

```

## 10.6 r\_draw.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     The actual span/column drawing functions.
//     Here find the main potential for optimization,
//     e.g. inline assembly, different algorithms.
//-----

static const char
rcsid[] = "$Id: r_draw.c,v 1.4 1997/02/03 16:47:55 b1 Exp $";

#include "doomdef.h"

#include "i_system.h"
#include "z_zone.h"

```

```

#include "w_wad.h"

#include "r_local.h"

// Needs access to LFB (guess what).
#include "v_video.h"

// State.
#include "doomstat.h"

// ?
#define MAXWIDTH          1120
#define MAXHEIGHT         832

// status bar height at bottom of screen
#define SBARHEIGHT        32

//
// All drawing to the view buffer is accomplished in this file.
// The other refresh files only know about coordinates,
// not the architecture of the frame buffer.
// Conveniently, the frame buffer is a linear one,
// and we need only the base address,
// and the total size == width*height*depth/8.,
//

byte*          viewimage;
int            viewwidth;
int            scaledviewwidth;
int            viewheight;
int            viewwindowx;
int            viewwindowy;
byte*          ylookup[MAXHEIGHT];
int            columnofs[MAXWIDTH];

// Color tables for different players,
// translate a limited part to another
// (color ramps used for suit colors).
//
byte            translations[3][256];

//
// R_DrawColumn
// Source is the top of the column to scale.
//
lighttable_t*  dc_colormap;
int            dc_x;
int            dc_y1;
int            dc_yh;
fixed_t        dc_iscale;
fixed_t        dc_texturemid;

// first pixel in a column (possibly virtual)
byte*          dc_source;

// just for profiling
int            dccount;

//
// A column is a vertical slice/span from a wall texture that,

```

```

// given the DOOM style restrictions on the view orientation,
// will always have constant z depth.
// Thus a special case loop for very fast rendering can
// be used. It has also been used with Wolfenstein 3D.
//
void R_DrawColumn (void)
{
    int                count;
    byte*              dest;
    fixed_t             frac;
    fixed_t             fracstep;

    count = dc_yh - dc_yl;

    // Zero length, column does not exceed a pixel.
    if (count < 0)
        return;

#ifdef RANGECHECK
    if ((unsigned)dc_x >= SCREENWIDTH
        || dc_yl < 0
        || dc_yh >= SCREENHEIGHT)
        I_Error ("R_DrawColumn: %i to %i at %i", dc_yl, dc_yh, dc_x);
#endif

    // Framebuffer destination address.
    // Use ylookup LUT to avoid multiply with ScreenWidth.
    // Use columnofs LUT for subwindows?
    dest = ylookup[dc_yl] + columnofs[dc_x];

    // Determine scaling,
    // which is the only mapping to be done.
    fracstep = dc_yscale;
    frac = dc_texturemid + (dc_yl - centery) * fracstep;

    // Inner loop that does the actual texture mapping,
    // e.g. a DDA-like scaling.
    // This is as fast as it gets.
    do
    {
        // Re-map color indices from wall texture column
        // using a lighting/special effects LUT.
        *dest = dc_colormap[dc_source[(frac >> FRACBITS) & 127]];

        dest += SCREENWIDTH;
        frac += fracstep;
    } while (count--);
}

// UNUSED.
// Loop unrolled.
#if 0
void R_DrawColumn (void)
{
    int                count;
    byte*              source;
    byte*              dest;
    byte*              colormap;

    unsigned           frac;
    unsigned           fracstep;
    unsigned           fracstep2;

```



```

unsigned          fracstep3;
unsigned          fracstep4;

count = dc_yh - dc_y1 + 1;

source = dc_source;
colormap = dc_colormap;
dest = ylookup[dc_y1] + columnofs[dc_x];

fracstep = dc_iscale<<9;
frac = (dc_texturemid + (dc_y1-centery)*dc_iscale)<<9;

fracstep2 = fracstep+fracstep;
fracstep3 = fracstep2+fracstep;
fracstep4 = fracstep3+fracstep;

while (count >= 8)
{
    dest[0] = colormap[source[frac>>25]];
    dest[SCREENWIDTH] = colormap[source[(frac+fracstep)>>25]];
    dest[SCREENWIDTH*2] = colormap[source[(frac+fracstep2)>>25]];
    dest[SCREENWIDTH*3] = colormap[source[(frac+fracstep3)>>25]];

    frac += fracstep4;

    dest[SCREENWIDTH*4] = colormap[source[frac>>25]];
    dest[SCREENWIDTH*5] = colormap[source[(frac+fracstep)>>25]];
    dest[SCREENWIDTH*6] = colormap[source[(frac+fracstep2)>>25]];
    dest[SCREENWIDTH*7] = colormap[source[(frac+fracstep3)>>25]];

    frac += fracstep4;
    dest += SCREENWIDTH*8;
    count -= 8;
}

while (count > 0)
{
    *dest = colormap[source[frac>>25]];
    dest += SCREENWIDTH;
    frac += fracstep;
    count--;
}
}
#endif

void R_DrawColumnLow (void)
{
    int          count;
    byte*        dest;
    byte*        dest2;
    fixed_t      frac;
    fixed_t      fracstep;

    count = dc_yh - dc_y1;

    // Zero length.
    if (count < 0)
        return;

#ifdef RANGECHECK
    if ((unsigned)dc_x >= SCREENWIDTH
        || dc_y1 < 0
        || dc_yh >= SCREENHEIGHT)
    {

```

```

        I_Error ("R_DrawColumn: %i to %i at %i", dc_yl, dc_yh, dc_x);
    }
    //      dccount++;
#endif
    // Blocky mode, need to multiply by 2.
    dc_x <= 1;

    dest = ylookup[dc_yl] + columnofs[dc_x];
    dest2 = ylookup[dc_yl] + columnofs[dc_x+1];

    fracstep = dc_yscale;
    frac = dc_texturemid + (dc_yl-center)*fracstep;

    do
    {
        // Hack. Does not work corretly.
        *dest2 = *dest = dc_colormap[dc_source[(frac>>FRACBITS)&127]];
        dest += SCREENWIDTH;
        dest2 += SCREENWIDTH;
        frac += fracstep;
    } while (count--);
}

//
// Spectre/Invisibility.
//
#define FUZZTABLE          50
#define FUZZOFF            (SCREENWIDTH)

int      fuzzoffset[FUZZTABLE] =
{
    FUZZOFF,-FUZZOFF,FUZZOFF,-FUZZOFF,FUZZOFF,FUZZOFF,-FUZZOFF,
    FUZZOFF,FUZZOFF,-FUZZOFF,FUZZOFF,FUZZOFF,FUZZOFF,-FUZZOFF,
    FUZZOFF,FUZZOFF,FUZZOFF,-FUZZOFF,-FUZZOFF,-FUZZOFF,-FUZZOFF,
    FUZZOFF,-FUZZOFF,-FUZZOFF,FUZZOFF,FUZZOFF,FUZZOFF,FUZZOFF,-FUZZOFF,
    FUZZOFF,-FUZZOFF,FUZZOFF,FUZZOFF,-FUZZOFF,-FUZZOFF,FUZZOFF,
    FUZZOFF,-FUZZOFF,-FUZZOFF,-FUZZOFF,-FUZZOFF,FUZZOFF,FUZZOFF,
    FUZZOFF,FUZZOFF,-FUZZOFF,FUZZOFF,FUZZOFF,-FUZZOFF,FUZZOFF
};

int      fuzzpos = 0;

//
// Framebuffer postprocessing.
// Creates a fuzzy image by copying pixels
// from adjacent ones to left and right.
// Used with an all black colormap, this
// could create the SHADOW effect,
// i.e. spectres and invisible players.
//
void R_DrawFuzzColumn (void)
{
    int      count;
    byte*    dest;
    fixed_t  frac;
    fixed_t  fracstep;

    // Adjust borders. Low...
    if (!dc_yl)
        dc_yl = 1;

```

```

// .. and high.
if (dc_yh == viewheight-1)
    dc_yh = viewheight - 2;

count = dc_yh - dc_yl;

// Zero length.
if (count < 0)
    return;

#ifdef RANGECHECK
    if ((unsigned)dc_x >= SCREENWIDTH
        || dc_yl < 0 || dc_yh >= SCREENHEIGHT)
    {
        I_Error ("R_DrawFuzzColumn: %i to %i at %i",
                dc_yl, dc_yh, dc_x);
    }
#endif

// Keep till detailshift bug in blocky mode fixed,
// or blocky mode removed.
/* WATCOM code
if (detailshift)
{
    if (dc_x & 1)
    {
        outpw (GC_INDEX,GC_READMAP+(2<<8) );
        outp (SC_INDEX+1,12);
    }
    else
    {
        outpw (GC_INDEX,GC_READMAP);
        outp (SC_INDEX+1,3);
    }
    dest = destview + dc_yl*80 + (dc_x>>1);
}
else
{
    outpw (GC_INDEX,GC_READMAP+((dc_x&3)<<8) );
    outp (SC_INDEX+1,1<<(dc_x&3));
    dest = destview + dc_yl*80 + (dc_x>>2);
}*/

// Does not work with blocky mode.
dest = ylookup[dc_yl] + columnofs[dc_x];

// Looks familiar.
fracstep = dc_iscalc;
frac = dc_texturemid + (dc_yl-center)*fracstep;

// Looks like an attempt at dithering,
// using the colormap #6 (of 0-31, a bit
// brighter than average).
do
{
    // Lookup framebuffer, and retrieve
    // a pixel that is either one column
    // left or right of the current one.
    // Add index from colormap to index.
    *dest = colormaps[6*256+dest[fuzzoffset[fuzzpos]]];

```

```

        // Clamp table lookup index.
        if (++fuzzpos == FUZZTABLE)
            fuzzpos = 0;

        dest += SCREENWIDTH;

        frac += fracstep;
    } while (count--);
}

//
// R_DrawTranslatedColumn
// Used to draw player sprites
// with the green colorramp mapped to others.
// Could be used with different translation
// tables, e.g. the lighter colored version
// of the BaronOfHell, the HellKnight, uses
// identical sprites, kinda brightened up.
//
byte*      dc_translation;
byte*      translationtables;

void R_DrawTranslatedColumn (void)
{
    int          count;
    byte*        dest;
    fixed_t      frac;
    fixed_t      fracstep;

    count = dc_yh - dc_y1;
    if (count < 0)
        return;

#ifdef RANGECHECK
    if ((unsigned)dc_x >= SCREENWIDTH
        || dc_y1 < 0
        || dc_yh >= SCREENHEIGHT)
    {
        I_Error ( "R_DrawColumn: %i to %i at %i",
                  dc_y1, dc_yh, dc_x);
    }
#endif

    // WATCOM VGA specific.
    /* Keep for fixing.
    if (detailshift)
    {
        if (dc_x & 1)
            outp (SC_INDEX+1,12);
        else
            outp (SC_INDEX+1,3);

        dest = destview + dc_y1*80 + (dc_x>>1);
    }
    else
    {
        outp (SC_INDEX+1,1<<(dc_x&3));

        dest = destview + dc_y1*80 + (dc_x>>2);
    }*/
}

```

```

// FIXME. As above.
dest = ylookup[dc_yl] + columnofs[dc_x];

// Looks familiar.
fracstep = dc_iscale;
frac = dc_texturemid + (dc_yl-centery)*fracstep;

// Here we do an additional index re-mapping.
do
{
    // Translation tables are used
    // to map certain colorramps to other ones,
    // used with PLAY sprites.
    // Thus the "green" ramp of the player 0 sprite
    // is mapped to gray, red, black/indigo.
    *dest = dc_colormap[dc_translation[dc_source[frac>>FRACBITS]]];
    dest += SCREENWIDTH;

    frac += fracstep;
} while (count--);
}

//
// R_InitTranslationTables
// Creates the translation tables to map
// the green color ramp to gray, brown, red.
// Assumes a given structure of the PLAYPAL.
// Could be read from a lump instead.
//
void R_InitTranslationTables (void)
{
    int i;

    translationtables = Z_Malloc (256*3+255, PU_STATIC, 0);
    translationtables = (byte *)(( (int)translationtables + 255 )& ~255);

    // translate just the 16 green colors
    for (i=0 ; i<256 ; i++)
    {
        if (i >= 0x70 && i<= 0x7f)
        {
            // map green ramp to gray, brown, red
            translationtables[i] = 0x60 + (i&0xf);
            translationtables [i+256] = 0x40 + (i&0xf);
            translationtables [i+512] = 0x20 + (i&0xf);
        }
        else
        {
            // Keep all other colors as is.
            translationtables[i] = translationtables[i+256]
                = translationtables[i+512] = i;
        }
    }
}

//
// R_DrawSpan

```

```

// With DOOM style restrictions on view orientation,
// the floors and ceilings consist of horizontal slices
// or spans with constant z depth.
// However, rotation around the world z axis is possible,
// thus this mapping, while simpler and faster than
// perspective correct texture mapping, has to traverse
// the texture at an angle in all but a few cases.
// In consequence, flats are not stored by column (like walls),
// and the inner loop has to step in texture space u and v.
//
int                ds_y;
int                ds_x1;
int                ds_x2;

lighttable_t*     ds_colormap;

fixed_t           ds_xfrac;
fixed_t           ds_yfrac;
fixed_t           ds_xstep;
fixed_t           ds_ystep;

// start of a 64*64 tile image
byte*             ds_source;

// just for profiling
int               dscount;

//
// Draws the actual span.
void R_DrawSpan (void)
{
    fixed_t        xfrac;
    fixed_t        yfrac;
    byte*          dest;
    int            count;
    int            spot;

#ifdef RANGECHECK
    if (ds_x2 < ds_x1
        || ds_x1<0
        || ds_x2>=SCREENWIDTH
        || (unsigned)ds_y>SCREENHEIGHT)
    {
        I_Error( "R_DrawSpan: %i to %i at %i",
                  ds_x1,ds_x2,ds_y);
    }
//    dscount++;
#endif

    xfrac = ds_xfrac;
    yfrac = ds_yfrac;

    dest = ylookup[ds_y] + columnofs[ds_x1];

    // We do not check for zero spans here?
    count = ds_x2 - ds_x1;

    do
    {
        // Current texture index in u,v.
        spot = ((yfrac>>(16-6))&(63*64)) + ((xfrac>>16)&63);

        // Lookup pixel from flat texture tile,

```

```

    // re-index using light/colormap.
    *dest++ = ds_colormap[ds_source[spot]];

    // Next step in u,v.
    xfrac += ds_xstep;
    yfrac += ds_ystep;
} while (count--);
}

// UNUSED.
// Loop unrolled by 4.
#if 0
void R_DrawSpan (void)
{
    unsigned    position, step;

    byte*       source;
    byte*       colormap;
    byte*       dest;

    unsigned    count;
    unsigned    spot;
    unsigned    value;
    unsigned    temp;
    unsigned    xtemp;
    unsigned    ytemp;

    position = ((ds_xfrac<<10)&0xffff0000) | ((ds_yfrac>>6)&0xffff);
    step = ((ds_xstep<<10)&0xffff0000) | ((ds_ystep>>6)&0xffff);

    source = ds_source;
    colormap = ds_colormap;
    dest = ylookup[ds_y] + columnofs[ds_x1];
    count = ds_x2 - ds_x1 + 1;

    while (count >= 4)
    {
        ytemp = position>>4;
        ytemp = ytemp & 4032;
        xtemp = position>>26;
        spot = xtemp | ytemp;
        position += step;
        dest[0] = colormap[source[spot]];

        ytemp = position>>4;
        ytemp = ytemp & 4032;
        xtemp = position>>26;
        spot = xtemp | ytemp;
        position += step;
        dest[1] = colormap[source[spot]];

        ytemp = position>>4;
        ytemp = ytemp & 4032;
        xtemp = position>>26;
        spot = xtemp | ytemp;
        position += step;
        dest[2] = colormap[source[spot]];

        ytemp = position>>4;
        ytemp = ytemp & 4032;
        xtemp = position>>26;
        spot = xtemp | ytemp;

```

```

    position += step;
    dest[3] = colormap[source[spot]];

    count -= 4;
    dest += 4;
}
while (count > 0)
{
    ytemp = position>>4;
    ytemp = ytemp & 4032;
    xtemp = position>>26;
    spot = xtemp | ytemp;
    position += step;
    *dest++ = colormap[source[spot]];
    count--;
}
}
#endif

//
// Again..
//
void R_DrawSpanLow (void)
{
    fixed_t          xfrac;
    fixed_t          yfrac;
    byte*            dest;
    int               count;
    int               spot;

#ifdef RANGECHECK
    if (ds_x2 < ds_x1
        || ds_x1<0
        || ds_x2>=SCREENWIDTH
        || (unsigned)ds_y>SCREENHEIGHT)
    {
        I_Error( "R_DrawSpan: %i to %i at %i",
                  ds_x1,ds_x2,ds_y);
    }
//    dscount++;
#endif

    xfrac = ds_xfrac;
    yfrac = ds_yfrac;

    // Blocky mode, need to multiply by 2.
    ds_x1 <= 1;
    ds_x2 <= 1;

    dest = ylookup[ds_y] + columnofs[ds_x1];

    count = ds_x2 - ds_x1;
    do
    {
        spot = ((yfrac>>(16-6))&(63*64)) + ((xfrac>>16)&63);
        // Lowres/blocky mode does it twice,
        // while scale is adjusted appropriately.
        *dest++ = ds_colormap[ds_source[spot]];
        *dest++ = ds_colormap[ds_source[spot]];

        xfrac += ds_xstep;
        yfrac += ds_ystep;
    }

```



```

    } while (count--);
}

//
// R_InitBuffer
// Creates lookup tables that avoid
// multiplies and other hazzles
// for getting the framebuffer address
// of a pixel to draw.
//
void
R_InitBuffer
( int          width,
  int          height )
{
    int          i;

    // Handle resize,
    // e.g. smaller view windows
    // with border and/or status bar.
    viewwindowx = (SCREENWIDTH-width) >> 1;

    // Column offset. For windows.
    for (i=0 ; i<width ; i++)
        columnofs[i] = viewwindowx + i;

    // Samw with base row offset.
    if (width == SCREENWIDTH)
        viewwindowy = 0;
    else
        viewwindowy = (SCREENHEIGHT-SBARHEIGHT-height) >> 1;

    // Preclaculate all row offsets.
    for (i=0 ; i<height ; i++)
        ylookup[i] = screens[0] + (i+viewwindowy)*SCREENWIDTH;
}

```

```

//
// R_FillBackScreen
// Fills the back screen with a pattern
// for variable screen sizes
// Also draws a beveled edge.
//
void R_FillBackScreen (void)
{
    byte*        src;
    byte*        dest;
    int          x;
    int          y;
    patch_t*     patch;

    // DOOM border patch.
    char          name1[] = "FLOOR7_2";

    // DOOM II border patch.
    char          name2[] = "GRNROCK";

    char*        name;

    if (scaledviewwidth == 320)
        return;
}

```

```

if ( gamemode == commercial)
    name = name2;
else
    name = name1;

src = W_CacheLumpName (name, PU_CACHE);
dest = screens[1];

for (y=0 ; y<SCREENHEIGHT-SBARHEIGHT ; y++)
{
    for (x=0 ; x<SCREENWIDTH/64 ; x++)
    {
        memcpy (dest, src+((y&63)<<6), 64);
        dest += 64;
    }

    if (SCREENWIDTH&63)
    {
        memcpy (dest, src+((y&63)<<6), SCREENWIDTH&63);
        dest += (SCREENWIDTH&63);
    }
}

patch = W_CacheLumpName ("brdr_t",PU_CACHE);

for (x=0 ; x<scaledviewwidth ; x+=8)
    V_DrawPatch (viewwindowx+x,viewwindowy-8,1,patch);
patch = W_CacheLumpName ("brdr_b",PU_CACHE);

for (x=0 ; x<scaledviewwidth ; x+=8)
    V_DrawPatch (viewwindowx+x,viewwindowy+viewheight,1,patch);
patch = W_CacheLumpName ("brdr_l",PU_CACHE);

for (y=0 ; y<viewheight ; y+=8)
    V_DrawPatch (viewwindowx-8,viewwindowy+y,1,patch);
patch = W_CacheLumpName ("brdr_r",PU_CACHE);

for (y=0 ; y<viewheight ; y+=8)
    V_DrawPatch (viewwindowx+scaledviewwidth,viewwindowy+y,1,patch);

// Draw beveled edge.
V_DrawPatch (viewwindowx-8,
             viewwindowy-8,
             1,
             W_CacheLumpName ("brdr_t1",PU_CACHE));

V_DrawPatch (viewwindowx+scaledviewwidth,
             viewwindowy-8,
             1,
             W_CacheLumpName ("brdr_tr",PU_CACHE));

V_DrawPatch (viewwindowx-8,
             viewwindowy+viewheight,
             1,
             W_CacheLumpName ("brdr_bl",PU_CACHE));

V_DrawPatch (viewwindowx+scaledviewwidth,
             viewwindowy+viewheight,
             1,
             W_CacheLumpName ("brdr_br",PU_CACHE));
}

//

```

```

// Copy a screen buffer.
//
void
R_VideoErase
( unsigned      ofs,
  int           count )
{
  // LFB copy.
  // This might not be a good idea if memcpy
  // is not optional, e.g. byte by byte on
  // a 32bit CPU, as GNU GCC/Linux libc did
  // at one point.
  memcpy (screens[0]+ofs, screens[1]+ofs, count);
}

//
// R_DrawViewBorder
// Draws the border around the view
// for different size windows?
//
void
V_MarkRect
( int          x,
  int          y,
  int          width,
  int          height );

void R_DrawViewBorder (void)
{
  int          top;
  int          side;
  int          ofs;
  int          i;

  if (scaledviewwidth == SCREENWIDTH)
    return;

  top = ((SCREENHEIGHT-SBARHEIGHT)-viewheight)/2;
  side = (SCREENWIDTH-scaledviewwidth)/2;

  // copy top and one line of left side
  R_VideoErase (0, top*SCREENWIDTH+side);

  // copy one line of right side and bottom
  ofs = (viewheight+top)*SCREENWIDTH-side;
  R_VideoErase (ofs, top*SCREENWIDTH+side);

  // copy sides using wraparound
  ofs = top*SCREENWIDTH + SCREENWIDTH-side;
  side <= 1;

  for (i=1 ; i<viewheight ; i++)
  {
    R_VideoErase (ofs, side);
    ofs += SCREENWIDTH;
  }

  // ?
  V_MarkRect (0,0,SCREENWIDTH, SCREENHEIGHT-SBARHEIGHT);
}

```

## 10.7 r\_draw.h

```
// Emacs style mode select    -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      System specific interface stuff.
//-----

#ifndef __R_DRAW__
#define __R_DRAW__

#ifdef __GNUG__
#pragma interface
#endif

extern lighttable_t*      dc_colormap;
extern int                dc_x;
extern int                dc_yl;
extern int                dc_yh;
extern fixed_t            dc_yscale;
extern fixed_t            dc_texturemid;

// first pixel in a column
extern byte*              dc_source;

// The span blitting interface.
// Hook in assembler or system specific BLT
// here.
void      R_DrawColumn (void);
void      R_DrawColumnLow (void);

// The Spectre/Invisibility effect.
void      R_DrawFuzzColumn (void);
void      R_DrawFuzzColumnLow (void);

// Draw with color translation tables,
// for player sprite rendering,
// Green/Red/Blue/Indigo shirts.
void      R_DrawTranslatedColumn (void);
void      R_DrawTranslatedColumnLow (void);

void
R_VideoErase
( unsigned      ofs,
  int           count );
```

```

extern int          ds_y;
extern int          ds_x1;
extern int          ds_x2;

extern lighttable_t* ds_colormap;

extern fixed_t      ds_xfrac;
extern fixed_t      ds_yfrac;
extern fixed_t      ds_xstep;
extern fixed_t      ds_ystep;

// start of a 64*64 tile image
extern byte*        ds_source;

extern byte*        translationtables;
extern byte*        dc_translation;

// Span blitting for rows, floor/ceiling.
// No Sepctre effect needed.
void                R_DrawSpan (void);

// Low resolution mode, 160x200?
void                R_DrawSpanLow (void);

void
R_InitBuffer
( int                width,
  int                height );

// Initialize color translation tables,
// for player rendering etc.
void                R_InitTranslationTables (void);

// Rendering function.
void R_FillBackScreen (void);

// If the view size is not full screen, draws a border around it.
void R_DrawViewBorder (void);

#endif
//-----
//
// $Log:$
//
//-----

```

## 10.8 r\_local.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2

```

```

// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Refresh (R_*) module, global header.
//     All the rendering/drawing stuff is here.
//
//-----

#ifndef __R_LOCAL__
#define __R_LOCAL__

// Binary Angles, sine/cosine/atan lookups.
#include "tables.h"

// Screen size related parameters.
#include "doomdef.h"

// Include the refresh/render data structs.
#include "r_data.h"

//
// Separate header file for each module.
//
#include "r_main.h"
#include "r_bsp.h"
#include "r_segs.h"
#include "r_plane.h"
#include "r_data.h"
#include "r_things.h"
#include "r_draw.h"

#endif // __R_LOCAL__
//-----
//
// $Log:$
//
//-----

```

## 10.9 r\_main.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$

```

```

//
// DESCRIPTION:
//      Rendering main loop and setup functions,
//      utility functions (BSP, geometry, trigonometry).
//      See tables.c, too.
//
//-----

static const char rcsid[] = "$Id: r_main.c,v 1.5 1997/02/03 22:45:12 b1 Exp $";


#include <stdlib.h>
#include <math.h>


#include "doomdef.h"
#include "d_net.h"


#include "m_bbox.h"


#include "r_local.h"
#include "r_sky.h"


// Fineangles in the SCREENWIDTH wide window.
#define FIELDOFVIEW          2048


int                viewangleoffset;

// increment every time a check is made
int                validcount = 1;


lighttable_t*      fixedcolormap;
extern lighttable_t** walllights;


int                centerx;
int                centery;


fixed_t            centerxfrac;
fixed_t            centeryfrac;
fixed_t            projection;


// just for profiling purposes
int                framecount;


int                sscount;
int                linecount;
int                loopcount;


fixed_t            viewx;
fixed_t            viewy;
fixed_t            viewz;


angle_t            viewangle;


fixed_t            viewcos;
fixed_t            viewsin;

```

```

player_t*          viewplayer;

// 0 = high, 1 = low
int                detailshift;

//
// precalculated math tables
//
angle_t            clipangle;

// The viewangletox[viewangle + FINEANGLES/4] lookup
// maps the visible view angles to screen X coordinates,
// flattening the arc to a flat projection plane.
// There will be many angles mapped to the same X.
int                viewangletox[FINEANGLES/2];

// The xtoviewangleangle[] table maps a screen pixel
// to the lowest viewangle that maps back to x ranges
// from clipangle to -clipangle.
angle_t            xtoviewangle[SCREENWIDTH+1];

// UNUSED.
// The finetangentgent[angle+FINEANGLES/4] table
// holds the fixed_t tangent values for view angles,
// ranging from MININT to 0 to MAXINT.
// fixed_t          finetangent[FINEANGLES/2];

// fixed_t          finesine[5*FINEANGLES/4];
fixed_t*           finecosine = &finesine[FINEANGLES/4];

lighttable_t*      scalelight[LIGHTLEVELS][MAXLIGHTSCALE];
lighttable_t*      scalelightfixed[MAXLIGHTSCALE];
lighttable_t*      zlight[LIGHTLEVELS][MAXLIGHTZ];

// bumped light from gun blasts
int                extralight;

void (*colfunc) (void);
void (*basecolfunc) (void);
void (*fuzzcolfunc) (void);
void (*transcolfunc) (void);
void (*spanfunc) (void);

//
// R_AddPointToBox
// Expand a given bbox
// so that it encloses a given point.
//
void
R_AddPointToBox
( int          x,
  int          y,
  fixed_t*     box )
{
    if (x< box[BOXLEFT])
        box[BOXLEFT] = x;
    if (x> box[BOXRIGHT])
        box[BOXRIGHT] = x;
}

```



```

    if (y< box[BOXBOTTOM])
        box[BOXBOTTOM] = y;
    if (y> box[BOXTOP])
        box[BOXTOP] = y;
}

//
// R_PointOnSide
// Traverse BSP (sub) tree,
// check point against partition plane.
// Returns side 0 (front) or 1 (back).
//
int
R_PointOnSide
( fixed_t      x,
  fixed_t      y,
  node_t*      node )
{
    fixed_t      dx;
    fixed_t      dy;
    fixed_t      left;
    fixed_t      right;

    if (!node->dx)
    {
        if (x <= node->x)
            return node->dy > 0;

        return node->dy < 0;
    }
    if (!node->dy)
    {
        if (y <= node->y)
            return node->dx < 0;

        return node->dx > 0;
    }

    dx = (x - node->x);
    dy = (y - node->y);

    // Try to quickly decide by looking at sign bits.
    if ( (node->dy ^ node->dx ^ dx ^ dy)&0x80000000 )
    {
        if ( (node->dy ^ dx) & 0x80000000 )
        {
            // (left is negative)
            return 1;
        }
        return 0;
    }

    left = FixedMul ( node->dy>>FRACBITS , dx );
    right = FixedMul ( dy , node->dx>>FRACBITS );

    if (right < left)
    {
        // front side
        return 0;
    }
    // back side
    return 1;
}

```

```

int
R_PointOnSegSide
( fixed_t      x,
  fixed_t      y,
  seg_t*       line )
{
    fixed_t      lx;
    fixed_t      ly;
    fixed_t      ldx;
    fixed_t      ldy;
    fixed_t      dx;
    fixed_t      dy;
    fixed_t      left;
    fixed_t      right;

    lx = line->v1->x;
    ly = line->v1->y;

    ldx = line->v2->x - lx;
    ldy = line->v2->y - ly;

    if (!ldx)
    {
        if (x <= lx)
            return ldy > 0;

        return ldy < 0;
    }
    if (!ldy)
    {
        if (y <= ly)
            return ldx < 0;

        return ldx > 0;
    }

    dx = (x - lx);
    dy = (y - ly);

    // Try to quickly decide by looking at sign bits.
    if ( (ldy ^ ldx ^ dx ^ dy) & 0x80000000 )
    {
        if ( (ldy ^ dx) & 0x80000000 )
        {
            // (left is negative)
            return 1;
        }
        return 0;
    }

    left = FixedMul ( ldy>>FRACBITS , dx );
    right = FixedMul ( dy , ldx>>FRACBITS );

    if (right < left)
    {
        // front side
        return 0;
    }
    // back side
    return 1;
}

//

```

```

// R_PointToAngle
// To get a global angle from cartesian coordinates,
// the coordinates are flipped until they are in
// the first octant of the coordinate system, then
// the y (<=x) is scaled and divided by x to get a
// tangent (slope) value which is looked up in the
// tantoangle[] table.

//

angle_t
R_PointToAngle
( fixed_t      x,
  fixed_t      y )
{
    x -= viewx;
    y -= viewy;

    if ( (!x) && (!y) )
        return 0;

    if (x>= 0)
    {
        // x >=0
        if (y>= 0)
        {
            // y>= 0

            if (x>y)
            {
                // octant 0
                return tantoangle[ SlopeDiv(y,x)];
            }
            else
            {
                // octant 1
                return ANG90-1-tantoangle[ SlopeDiv(x,y)];
            }
        }
        else
        {
            // y<0
            y = -y;

            if (x>y)
            {
                // octant 8
                return -tantoangle[SlopeDiv(y,x)];
            }
            else
            {
                // octant 7
                return ANG270+tantoangle[ SlopeDiv(x,y)];
            }
        }
    }
    else
    {
        // x<0
        x = -x;

        if (y>= 0)

```

```

{
    // y>= 0
    if (x>y)
    {
        // octant 3
        return ANG180-1-tantoangle[ SlopeDiv(y,x)];
    }
    else
    {
        // octant 2
        return ANG90+ tantoangle[ SlopeDiv(x,y)];
    }
}
else
{
    // y<0
    y = -y;

    if (x>y)
    {
        // octant 4
        return ANG180+tantoangle[ SlopeDiv(y,x)];
    }
    else
    {
        // octant 5
        return ANG270-1-tantoangle[ SlopeDiv(x,y)];
    }
}
}
return 0;
}

```

```

angle_t
R_PointToAngle2
( fixed_t      x1,
  fixed_t      y1,
  fixed_t      x2,
  fixed_t      y2 )
{
    viewx = x1;
    viewy = y1;

    return R_PointToAngle (x2, y2);
}

```

```

fixed_t
R_PointToDist
( fixed_t      x,
  fixed_t      y )
{
    int          angle;
    fixed_t      dx;
    fixed_t      dy;
    fixed_t      temp;
    fixed_t      dist;

    dx = abs(x - viewx);
    dy = abs(y - viewy);

    if (dy>dx)
    {
        temp = dx;

```

```

        dx = dy;
        dy = temp;
    }

    angle = (tantoangle[ FixedDiv(dy,dx)>>DBITS ]+ANG90) >> ANGLETOFINESHIFT;

    // use as cosine
    dist = FixedDiv (dx, finesine[angle] );

    return dist;
}

//
// R_InitPointToAngle
//
void R_InitPointToAngle (void)
{
    // UNUSED - now getting from tables.c
#ifdef 0
    int      i;
    long      t;
    float      f;
//
// slope (tangent) to angle lookup
//
    for (i=0 ; i<=SLOPERANGE ; i++)
    {
        f = atan( (float)i/SLOPERANGE )/(3.141592657*2);
        t = 0xffffffff*f;
        tantoangle[i] = t;
    }
#endif
}

//
// R_ScaleFromGlobalAngle
// Returns the texture mapping scale
// for the current line (horizontal span)
// at the given angle.
// rw_distance must be calculated first.
//
fixed_t R_ScaleFromGlobalAngle (angle_t visangle)
{
    fixed_t      scale;
    int          anglea;
    int          angleb;
    int          sinea;
    int          sineb;
    fixed_t      num;
    int          den;

    // UNUSED
#ifdef 0
    {
        fixed_t      dist;
        fixed_t      z;
        fixed_t      sinv;
        fixed_t      cosv;

        sinv = finesine[(visangle-rw_normalangle)>>ANGLETOFINESHIFT];
        dist = FixedDiv (rw_distance, sinv);
    }
#endif
}

```

```

    cosv = finecosine[(viewangle-visangle)>>ANGLETOFINESHIFT];
    z = abs(FixedMul (dist, cosv));
    scale = FixedDiv(projection, z);
    return scale;
}
#endif

anglea = ANG90 + (visangle-viewangle);
angleb = ANG90 + (visangle-rw_normalangle);

// both sines are allways positive
sinea = finesine[anglea>>ANGLETOFINESHIFT];
sineb = finesine[angleb>>ANGLETOFINESHIFT];
num = FixedMul(projection,sineb)<<detailshift;
den = FixedMul(rw_distance,sinea);

if (den > num>>16)
{
    scale = FixedDiv (num, den);

    if (scale > 64*FRACUNIT)
        scale = 64*FRACUNIT;
    else if (scale < 256)
        scale = 256;
}
else
    scale = 64*FRACUNIT;

return scale;
}

//
// R_InitTables
//
void R_InitTables (void)
{
    // UNUSED: now getting from tables.c
#ifdef 0
    int          i;
    float        a;
    float        fv;
    int          t;

    // viewangle tangent table
    for (i=0 ; i<FINEANGLES/2 ; i++)
    {
        a = (i-FINEANGLES/4+0.5)*PI*2/FINEANGLES;
        fv = FRACUNIT*tan (a);
        t = fv;
        finetangent[i] = t;
    }

    // finesine table
    for (i=0 ; i<5*FINEANGLES/4 ; i++)
    {
        // OPTIMIZE: mirror...
        a = (i+0.5)*PI*2/FINEANGLES;
        t = FRACUNIT*sin (a);
        finesine[i] = t;
    }
#endif
}

```

```

//
// R_InitTextureMapping
//
void R_InitTextureMapping (void)
{
    int                i;
    int                x;
    int                t;
    fixed_t            focallength;

    // Use tangent table to generate viewangletox:
    // viewangletox will give the next greatest x
    // after the view angle.
    //
    // Calc focallength
    // so FIELDOFVIEW angles covers SCREENWIDTH.
    focallength = FixedDiv (centerxfrac,
                            finetangent[FINEANGLES/4+FIELDOFVIEW/2] );

    for (i=0 ; i<FINEANGLES/2 ; i++)
    {
        if (finetangent[i] > FRACUNIT*2)
            t = -1;
        else if (finetangent[i] < -FRACUNIT*2)
            t = viewwidth+1;
        else
        {
            t = FixedMul (finetangent[i], focallength);
            t = (centerxfrac - t+FRACUNIT-1)>>FRACBITS;

            if (t < -1)
                t = -1;
            else if (t>viewwidth+1)
                t = viewwidth+1;
        }
        viewangletox[i] = t;
    }

    // Scan viewangletox[] to generate xtoviewangle[]:
    // xtoviewangle will give the smallest view angle
    // that maps to x.
    for (x=0;x<=viewwidth;x++)
    {
        i = 0;
        while (viewangletox[i]>x)
            i++;
        xtoviewangle[x] = (i<<ANGLETOFINESHIFT)-ANG90;
    }

    // Take out the fencepost cases from viewangletox.
    for (i=0 ; i<FINEANGLES/2 ; i++)
    {
        t = FixedMul (finetangent[i], focallength);
        t = centerx - t;

        if (viewangletox[i] == -1)
            viewangletox[i] = 0;
        else if (viewangletox[i] == viewwidth+1)
            viewangletox[i] = viewwidth;
    }

    clipangle = xtoviewangle[0];
}

```

```

}

//
// R_InitLightTables
// Only inits the zlight table,
// because the scalelight table changes with view size.
//
#define DISTMAP                2

void R_InitLightTables (void)
{
    int            i;
    int            j;
    int            level;
    int            startmap;
    int            scale;

    // Calculate the light levels to use
    // for each level / distance combination.
    for (i=0 ; i< LIGHTLEVELS ; i++)
    {
        startmap = ((LIGHTLEVELS-1-i)*2)*NUMCOLORMAPS/LIGHTLEVELS;
        for (j=0 ; j<MAXLIGHTZ ; j++)
        {
            scale = FixedDiv ((SCREENWIDTH/2*FRACUNIT), (j+1)<<LIGHTZSHIFT);
            scale >>= LIGHTSCAleshift;
            level = startmap - scale/DISTMAP;

            if (level < 0)
                level = 0;

            if (level >= NUMCOLORMAPS)
                level = NUMCOLORMAPS-1;

            zlight[i][j] = colormaps + level*256;
        }
    }
}

//
// R_SetViewSize
// Do not really change anything here,
// because it might be in the middle of a refresh.
// The change will take effect next refresh.
//
boolean            setsizeneeded;
int                setblocks;
int                setdetail;

void
R_SetViewSize
( int                blocks,
  int                detail )
{
    setsizeneeded = true;
    setblocks = blocks;
    setdetail = detail;
}

```



```

//
// R_ExecuteSetViewSize
//
void R_ExecuteSetViewSize (void)
{
    fixed_t      cosadj;
    fixed_t      dy;
    int          i;
    int          j;
    int          level;
    int          startmap;

    setsizeneeded = false;

    if (setblocks == 11)
    {
        scaledviewwidth = SCREENWIDTH;
        viewheight = SCREENHEIGHT;
    }
    else
    {
        scaledviewwidth = setblocks*32;
        viewheight = (setblocks*168/10)&~7;
    }

    detailshift = setdetail;
    viewwidth = scaledviewwidth>>detailshift;

    centery = viewheight/2;
    centerx = viewwidth/2;
    centerxfrac = centerx<<FRACBITS;
    centeryfrac = centery<<FRACBITS;
    projection = centerxfrac;

    if (!detailshift)
    {
        colfunc = basecolfunc = R_DrawColumn;
        fuzzcolfunc = R_DrawFuzzColumn;
        transcolfunc = R_DrawTranslatedColumn;
        spanfunc = R_DrawSpan;
    }
    else
    {
        colfunc = basecolfunc = R_DrawColumnLow;
        fuzzcolfunc = R_DrawFuzzColumn;
        transcolfunc = R_DrawTranslatedColumn;
        spanfunc = R_DrawSpanLow;
    }

    R_InitBuffer (scaledviewwidth, viewheight);

    R_InitTextureMapping ();

    // psprite scales
    pspritescale = FRACUNIT*viewwidth/SCREENWIDTH;
    pspriteiscale = FRACUNIT*SCREENWIDTH/viewwidth;

    // thing clipping
    for (i=0 ; i<viewwidth ; i++)
        screenheightarray[i] = viewheight;

    // planes
    for (i=0 ; i<viewheight ; i++)
    {
        dy = ((i-viewheight/2)<<FRACBITS)+FRACUNIT/2;

```

```

        dy = abs(dy);
        yslope[i] = FixedDiv ( (viewwidth<<detailshift)/2*FRACUNIT, dy);
    }

    for (i=0 ; i<viewwidth ; i++)
    {
        cosadj = abs(finecosine[xtoviewangle[i]>>ANGLETOFINESHIFT]);
        distscale[i] = FixedDiv (FRACUNIT,cosadj);
    }

    // Calculate the light levels to use
    // for each level / scale combination.
    for (i=0 ; i< LIGHTLEVELS ; i++)
    {
        startmap = ((LIGHTLEVELS-1-i)*2)*NUMCOLORMAPS/LIGHTLEVELS;
        for (j=0 ; j<MAXLIGHTSCALE ; j++)
        {
            level = startmap - j*SCREENWIDTH/(viewwidth<<detailshift)/DISTMAP;

            if (level < 0)
                level = 0;

            if (level >= NUMCOLORMAPS)
                level = NUMCOLORMAPS-1;

            scalelight[i][j] = colormaps + level*256;
        }
    }
}

```

```

//
// R_Init
//
extern int      detailLevel;
extern int      screenblocks;

```

```

void R_Init (void)
{
    R_InitData ();
    printf ("\nR_InitData");
    R_InitPointToAngle ();
    printf ("\nR_InitPointToAngle");
    R_InitTables ();
    // viewwidth / viewheight / detailLevel are set by the defaults
    printf ("\nR_InitTables");

    R_SetViewSize (screenblocks, detailLevel);
    R_InitPlanes ();
    printf ("\nR_InitPlanes");
    R_InitLightTables ();
    printf ("\nR_InitLightTables");
    R_InitSkyMap ();
    printf ("\nR_InitSkyMap");
    R_InitTranslationTables ();
    printf ("\nR_InitTranslationsTables");

    framecount = 0;
}

```

```

//

```

```

// R_PointInSubsector
//
subsector_t*
R_PointInSubsector
( fixed_t      x,
  fixed_t      y )
{
    node_t*      node;
    int          side;
    int          nodenum;

    // single subsector is a special case
    if (!numnodes)
        return subsectors;

    nodenum = numnodes-1;

    while (! (nodenum & NF_SUBSECTOR) )
    {
        node = &nodes[nodenum];
        side = R_PointOnSide (x, y, node);
        nodenum = node->children[side];
    }

    return &subsectors[nodenum & ~NF_SUBSECTOR];
}

//
// R_SetupFrame
//
void R_SetupFrame (player_t* player)
{
    int          i;

    viewplayer = player;
    viewx = player->mo->x;
    viewy = player->mo->y;
    viewangle = player->mo->angle + viewangleoffset;
    extralight = player->extralight;

    viewz = player->viewz;

    viewsin = finesine[viewangle>>ANGLETOFINESHIFT];
    viewcos = finecospine[viewangle>>ANGLETOFINESHIFT];

    sscount = 0;

    if (player->fixedcolormap)
    {
        fixedcolormap =
            colormaps
            + player->fixedcolormap*256*sizeof(lighttable_t);

        walllights = scalelightfixed;

        for (i=0 ; i<MAXLIGHTSCALE ; i++)
            scalelightfixed[i] = fixedcolormap;
    }
    else
        fixedcolormap = 0;

    framecount++;
    validcount++;
}

```

```

}

//
// R_RenderView
//
void R_RenderPlayerView (player_t* player)
{
    R_SetupFrame (player);

    // Clear buffers.
    R_ClearClipSegs ();
    R_ClearDrawSegs ();
    R_ClearPlanes ();
    R_ClearSprites ();

    // check for new console commands.
    NetUpdate ();

    // The head node is the last node output.
    R_RenderBSPNode (numnodes-1);

    // Check for new console commands.
    NetUpdate ();

    R_DrawPlanes ();

    // Check for new console commands.
    NetUpdate ();

    R_DrawMasked ();

    // Check for new console commands.
    NetUpdate ();
}

```

## 10.10 r\_main.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      System specific interface stuff.
//
//-----

#ifdef __R_MAIN__
#define __R_MAIN__

```

```

#include "d_player.h"
#include "r_data.h"

#ifdef __GNUG__
#pragma interface
#endif

//
// POV related.
//
extern fixed_t          viewcos;
extern fixed_t          viewsin;

extern int              viewwidth;
extern int              viewheight;
extern int              viewwindowx;
extern int              viewwindowy;

extern int              centerx;
extern int              centery;

extern fixed_t          centerxfrac;
extern fixed_t          centeryfrac;
extern fixed_t          projection;

extern int              validcount;

extern int              linecount;
extern int              loopcount;

//
// Lighting LUT.
// Used for z-depth cuing per column/row,
// and other lighting effects (sector ambient, flash).
//

// Lighting constants.
// Now why not 32 levels here?
#define LIGHTLEVELS      16
#define LIGHTSEGSHIFT    4

#define MAXLIGHTSCALE     48
#define LIGHTSCALESHIFT  12
#define MAXLIGHTZ        128
#define LIGHTZSHIFT      20

extern lighttable_t*     scalelight[LIGHTLEVELS][MAXLIGHTSCALE];
extern lighttable_t*     scalelightfixed[MAXLIGHTSCALE];
extern lighttable_t*     zlight[LIGHTLEVELS][MAXLIGHTZ];

extern int               extralight;
extern lighttable_t*     fixedcolormap;

// Number of diminishing brightness levels.
// There a 0-31, i.e. 32 LUT in the COLORMAP lump.
#define NUMCOLORMAPS     32

// Blocky/low detail mode.

```

```

//B remove this?
// 0 = high, 1 = low
extern      int          detailshift;

//
// Function pointers to switch refresh/drawing functions.
// Used to select shadow mode etc.
//
extern void          (*colfunc) (void);
extern void          (*basecolfunc) (void);
extern void          (*fuzzcolfunc) (void);
// No shadow effects on floors.
extern void          (*spanfunc) (void);

//
// Utility functions.
int
R_PointOnSide
( fixed_t          x,
  fixed_t          y,
  node_t*          node );

int
R_PointOnSegSide
( fixed_t          x,
  fixed_t          y,
  seg_t*          line );

angle_t
R_PointToAngle
( fixed_t          x,
  fixed_t          y );

angle_t
R_PointToAngle2
( fixed_t          x1,
  fixed_t          y1,
  fixed_t          x2,
  fixed_t          y2 );

fixed_t
R_PointToDist
( fixed_t          x,
  fixed_t          y );

fixed_t R_ScaleFromGlobalAngle (angle_t visangle);

subsector_t*
R_PointInSubsector
( fixed_t          x,
  fixed_t          y );

void
R_AddPointToBox
( int              x,
  int              y,
  fixed_t*         box );

//
// REFRESH - the actual rendering functions.

```

```
//

// Called by G_Drawer.
void R_RenderPlayerView (player_t *player);

// Called by startup code.
void R_Init (void);

// Called by M_Responder.
void R_SetViewSize (int blocks, int detail);

#endif
//-----
//
// $Log:$
//
//-----
```

## 10.11 r\_plane.c

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Here is a core component: drawing the floors and ceilings,
//     while maintaining a per column clipping list only.
//     Moreover, the sky areas have to be determined.
//
//-----

static const char
rcsid[] = "$Id: r_plane.c,v 1.4 1997/02/03 16:47:55 b1 Exp $";

#include <stdlib.h>

#include "i_system.h"
#include "z_zone.h"
#include "w_wad.h"

#include "doomdef.h"
#include "doomstat.h"

#include "r_local.h"
#include "r_sky.h"

planefunction_t          floorfunc;
```

```

planefunction_t          ceilingfunc;

//
// opening
//

// Here comes the obnoxious "visplane".
#define MAXVISPLANES      128
visplane_t               visplanes[MAXVISPLANES];
visplane_t*               lastvisplane;
visplane_t*               floorplane;
visplane_t*               ceilingplane;

// ?
#define MAXOPENINGS       SCREENWIDTH*64
short                     openings[MAXOPENINGS];
short*                     lastopening;

//
// Clip values are the solid pixel bounding the range.
// floorclip starts out SCREENHEIGHT
// ceilingclip starts out -1
//
short                      floorclip[SCREENWIDTH];
short                      ceilingclip[SCREENWIDTH];

//
// spanstart holds the start of a plane span
// initialized to 0 at start
//
int                        spanstart[SCREENHEIGHT];
int                        spanstop[SCREENHEIGHT];

//
// texture mapping
//
lighttable_t**            planezlight;
fixed_t                   planeheight;

fixed_t                   yslope[SCREENHEIGHT];
fixed_t                   distscale[SCREENWIDTH];
fixed_t                   basexscale;
fixed_t                   baseyscale;

fixed_t                   cachedheight[SCREENHEIGHT];
fixed_t                   cacheddistance[SCREENHEIGHT];
fixed_t                   cachedxstep[SCREENHEIGHT];
fixed_t                   cachedystep[SCREENHEIGHT];

//
// R_InitPlanes
// Only at game startup.
//
void R_InitPlanes (void)
{
    // Doh!
}

//
// R_MapPlane
//

```



```

// Uses global vars:
// planeheight
// ds_source
// basexscale
// baseyscale
// viewx
// viewy
//
// BASIC PRIMITIVE
//
void
R_MapPlane
( int          y,
  int          x1,
  int          x2 )
{
    angle_t     angle;
    fixed_t     distance;
    fixed_t     length;
    unsigned    index;

#ifdef RANGECHECK
    if (x2 < x1
        || x1<0
        || x2>=viewwidth
        || (unsigned)y>viewheight)
    {
        I_Error ("R_MapPlane: %i, %i at %i",x1,x2,y);
    }
#endif

    if (planeheight != cachedheight[y])
    {
        cachedheight[y] = planeheight;
        distance = cacheddistance[y] = FixedMul (planeheight, yslope[y]);
        ds_xstep = cachedxstep[y] = FixedMul (distance,basexscale);
        ds_ystep = cachedystep[y] = FixedMul (distance,baseyscale);
    }
    else
    {
        distance = cacheddistance[y];
        ds_xstep = cachedxstep[y];
        ds_ystep = cachedystep[y];
    }

    length = FixedMul (distance,distscale[x1]);
    angle = (viewangle + xtoviewangle[x1])>>ANGLETOFINESHIFT;
    ds_xfrac = viewx + FixedMul(finecosine[angle], length);
    ds_yfrac = -viewy - FixedMul(finesine[angle], length);

    if (fixedcolormap)
        ds_colormap = fixedcolormap;
    else
    {
        index = distance >> LIGHTZSHIFT;

        if (index >= MAXLIGHTZ )
            index = MAXLIGHTZ-1;

        ds_colormap = planezlight[index];
    }

    ds_y = y;
    ds_x1 = x1;
    ds_x2 = x2;

```

```

    // high or low detail
    spanfunc ();
}

//
// R_ClearPlanes
// At beginning of frame.
//
void R_ClearPlanes (void)
{
    int            i;
    angle_t        angle;

    // opening / clipping determination
    for (i=0 ; i<viewwidth ; i++)
    {
        floorclip[i] = viewheight;
        ceilingclip[i] = -1;
    }

    lastvisplane = visplanes;
    lastopening = openings;

    // texture calculation
    memset (cachedheight, 0, sizeof(cachedheight));

    // left to right mapping
    angle = (viewangle-ANG90)>>ANGLETOFINESHIFT;

    // scale will be unit scale at SCREENWIDTH/2 distance
    basexscale = FixedDiv (finecosine[angle],centerxfrac);
    baseyscale = -FixedDiv (finesine[angle],centerxfrac);
}

//
// R_FindPlane
//
visplane_t*
R_FindPlane
( fixed_t        height,
  int            picnum,
  int            lightlevel )
{
    visplane_t*   check;

    if (picnum == skyflatnum)
    {
        height = 0;
        lightlevel = 0;
        // all skys map together
    }

    for (check=visplanes; check<lastvisplane; check++)
    {
        if (height == check->height
            && picnum == check->picnum
            && lightlevel == check->lightlevel)
        {
            break;
        }
    }
}

```

```

    if (check < lastvisplane)
        return check;

    if (lastvisplane - visplanes == MAXVISPLANES)
        I_Error ("R_FindPlane: no more visplanes");

    lastvisplane++;

    check->height = height;
    check->picnum = picnum;
    check->lightlevel = lightlevel;
    check->minx = SCREENWIDTH;
    check->maxx = -1;

    memset (check->top, 0xff, sizeof(check->top));

    return check;
}

```

```

//
// R_CheckPlane
//
visplane_t*
R_CheckPlane
( visplane_t*    pl,
  int            start,
  int            stop )
{
    int            intrl;
    int            intrh;
    int            unionl;
    int            unionh;
    int            x;

    if (start < pl->minx)
    {
        intrl = pl->minx;
        unionl = start;
    }
    else
    {
        unionl = pl->minx;
        intrl = start;
    }

    if (stop > pl->maxx)
    {
        intrh = pl->maxx;
        unionh = stop;
    }
    else
    {
        unionh = pl->maxx;
        intrh = stop;
    }

    for (x=intrl ; x<= intrh ; x++)
        if (pl->top[x] != 0xff)
            break;

    if (x > intrh)
    {

```

```

    pl->minx = unionl;
    pl->maxx = unionh;

    // use the same one
    return pl;
}

// make a new visplane
lastvisplane->height = pl->height;
lastvisplane->picnum = pl->picnum;
lastvisplane->lightlevel = pl->lightlevel;

pl = lastvisplane++;
pl->minx = start;
pl->maxx = stop;

memset (pl->top,0xff,sizeof(pl->top));

return pl;
}

```

```

//
// R_MakeSpans
//
void
R_MakeSpans
( int          x,
  int          t1,
  int          b1,
  int          t2,
  int          b2 )
{
    while (t1 < t2 && t1<=b1)
    {
        R_MapPlane (t1,spanstart[t1],x-1);
        t1++;
    }
    while (b1 > b2 && b1>=t1)
    {
        R_MapPlane (b1,spanstart[b1],x-1);
        b1--;
    }

    while (t2 < t1 && t2<=b2)
    {
        spanstart[t2] = x;
        t2++;
    }
    while (b2 > b1 && b2>=t2)
    {
        spanstart[b2] = x;
        b2--;
    }
}

```

```

//
// R_DrawPlanes
// At the end of each frame.
//
void R_DrawPlanes (void)
{
    visplane_t*      pl;

```

```

int                light;
int                x;
int                stop;
int                angle;

#ifdef RANGECHECK
    if (ds_p - drawsegs > MAXDRAWSEGS)
        I_Error ("R_DrawPlanes: drawsegs overflow (%i)",
            ds_p - drawsegs);

    if (lastvisplane - visplanes > MAXVISPLANES)
        I_Error ("R_DrawPlanes: visplane overflow (%i)",
            lastvisplane - visplanes);

    if (lastopening - openings > MAXOPENINGS)
        I_Error ("R_DrawPlanes: opening overflow (%i)",
            lastopening - openings);
#endif

for (pl = visplanes ; pl < lastvisplane ; pl++)
{
    if (pl->minx > pl->maxx)
        continue;

    // sky flat
    if (pl->picnum == skyflatnum)
    {
        dc_iscale = pspriteiscale>>detailshift;

        // Sky is allways drawn full bright,
        // i.e. colormaps[0] is used.
        // Because of this hack, sky is not affected
        // by INVUL inverse mapping.
        dc_colormap = colormaps;
        dc_texturemid = skytexturemid;
        for (x=pl->minx ; x <= pl->maxx ; x++)
        {
            dc_y1 = pl->top[x];
            dc_yh = pl->bottom[x];

            if (dc_y1 <= dc_yh)
            {
                angle = (viewangle + xtoviewangle[x])>>ANGLETOSKYSHIFT;
                dc_x = x;
                dc_source = R_GetColumn(skytexture, angle);
                colfunc ();
            }
        }
        continue;
    }

    // regular flat
    ds_source = W_CacheLumpNum(firstflat +
                                flattranslation[pl->picnum],
                                PU_STATIC);

    planeheight = abs(pl->height-viewz);
    light = (pl->lightlevel >> LIGHTSEGS SHIFT)+extralight;

    if (light >= LIGHTLEVELS)
        light = LIGHTLEVELS-1;

    if (light < 0)
        light = 0;
}

```

```

    planezlight = zlight[light];

    pl->top[pl->maxx+1] = 0xff;
    pl->top[pl->minx-1] = 0xff;

    stop = pl->maxx + 1;

    for (x=pl->minx ; x<= stop ; x++)
    {
        R_MakeSpans(x,pl->top[x-1],
                    pl->bottom[x-1],
                    pl->top[x],
                    pl->bottom[x]);
    }

    Z_ChangeTag (ds_source, PU_CACHE);
}
}

```

## 10.12 r\_plane.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Refresh, visplane stuff (floor, ceilings).
//
//-----

#ifndef __R_PLANE__
#define __R_PLANE__

#include "r_data.h"

#ifdef __GNUG__
#pragma interface
#endif

// Visplane related.
extern short*          lastopening;

typedef void (*planefunction_t) (int top, int bottom);

extern planefunction_t  floorfunc;
extern planefunction_t  ceilingfunc_t;

```

```

extern short          floorclip[SCREENWIDTH];
extern short          ceilingclip[SCREENWIDTH];

extern fixed_t        yslope[SCREENHEIGHT];
extern fixed_t        distscale[SCREENWIDTH];

void R_InitPlanes (void);
void R_ClearPlanes (void);

void
R_MapPlane
( int          y,
  int          x1,
  int          x2 );

void
R_MakeSpans
( int          x,
  int          t1,
  int          b1,
  int          t2,
  int          b2 );

void R_DrawPlanes (void);

visplane_t*
R_FindPlane
( fixed_t      height,
  int          picnum,
  int          lightlevel );

visplane_t*
R_CheckPlane
( visplane_t*  pl,
  int          start,
  int          stop );

#endif
//-----
//
// $Log:$
//
//-----

```

## 10.13 r\_segs.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//

```

```

// $Log:$
//
// DESCRIPTION:
//      All the clipping: columns, horizontal spans, sky columns.
//
//-----

static const char
rcsid[] = "$Id: r_segs.c,v 1.3 1997/01/29 20:10:19 b1 Exp $";


#include <stdlib.h>

#include "i_system.h"

#include "doomdef.h"
#include "doomstat.h"

#include "r_local.h"
#include "r_sky.h"


// OPTIMIZE: closed two sided lines as single sided

// True if any of the segs textures might be visible.
boolean          segtextured;

// False if the back side is the same plane.
boolean          markfloor;
boolean          markceiling;

boolean          maskedtexture;
int              toptexture;
int              bottomtexture;
int              midtexture;


angle_t          rw_normalangle;
// angle to line origin
int              rw_angle1;


//
// regular wall
//
int              rw_x;
int              rw_stopx;
angle_t          rw_centerangle;
fixed_t          rw_offset;
fixed_t          rw_distance;
fixed_t          rw_scale;
fixed_t          rw_scalestep;
fixed_t          rw_midtexturemid;
fixed_t          rw_toptexturemid;
fixed_t          rw_bottomtexturemid;


int              worldtop;
int              worldbottom;
int              worldhigh;
int              worldlow;

fixed_t          pixhigh;

```



```

fixed_t          pixlow;
fixed_t          pixhighstep;
fixed_t          pixlowstep;

fixed_t          topfrac;
fixed_t          topstep;

fixed_t          bottomfrac;
fixed_t          bottomstep;

lighttable_t**   walllights;

short*           maskedtexturecol;

//
// R_RenderMaskedSegRange
//
void
R_RenderMaskedSegRange
( drawseg_t*      ds,
  int             x1,
  int             x2 )
{
    unsigned      index;
    column_t*     col;
    int           lightnum;
    int           texnum;

    // Calculate light table.
    // Use different light tables
    //   for horizontal / vertical / diagonal. Diagonal?
    // OPTIMIZE: get rid of LIGHTSEGS_SHIFT globally
    curline = ds->curline;
    frontsector = curline->frontsector;
    backsector = curline->backsector;
    texnum = texturetranslation[curline->sidedef->midtexture];

    lightnum = (frontsector->lightlevel >> LIGHTSEGS_SHIFT)+extralight;

    if (curline->v1->y == curline->v2->y)
        lightnum--;
    else if (curline->v1->x == curline->v2->x)
        lightnum++;

    if (lightnum < 0)
        walllights = scalelight[0];
    else if (lightnum >= LIGHTLEVELS)
        walllights = scalelight[LIGHTLEVELS-1];
    else
        walllights = scalelight[lightnum];

    maskedtexturecol = ds->maskedtexturecol;

    rw_scalestep = ds->scalestep;
    spryscale = ds->scale1 + (x1 - ds->x1)*rw_scalestep;
    mfloorclip = ds->sprbottomclip;
    mceilingclip = ds->sprtopclip;

    // find positioning
    if (curline->linedef->flags & ML_DONTPEGBOTTOM)
    {
        dc_texturemid = frontsector->floorheight > backsector->floorheight

```

```

        ? frontsector->floorheight : backsector->floorheight;
    dc_texturemid = dc_texturemid + textureheight[texturenum] - viewz;
}
else
{
    dc_texturemid = frontsector->ceilingheight < backsector->ceilingheight
        ? frontsector->ceilingheight : backsector->ceilingheight;
    dc_texturemid = dc_texturemid - viewz;
}
dc_texturemid += curline->sidedef->rowoffset;

if (fixedcolormap)
    dc_colormap = fixedcolormap;

// draw the columns
for (dc_x = x1 ; dc_x <= x2 ; dc_x++)
{
    // calculate lighting
    if (maskedtexturecol[dc_x] != MAXSHORT)
    {
        if (!fixedcolormap)
        {
            index = spryscale >> LIGHTSCAleshift;

            if (index >= MAXLIGHTSCALE)
                index = MAXLIGHTSCALE-1;

            dc_colormap = walllights[index];
        }

        sprtopscreen = centeryfrac - FixedMul(dc_texturemid, spryscale);
        dc_yscale = 0xffffffffu / (unsigned)spryscale;

        // draw the texture
        col = (column_t *) (
            (byte *) R_GetColumn(texturenum, maskedtexturecol[dc_x]) - 3);

        R_DrawMaskedColumn (col);
        maskedtexturecol[dc_x] = MAXSHORT;
    }
    spryscale += rw_scalestep;
}
}

```

```

//
// R_RenderSegLoop
// Draws zero, one, or two textures (and possibly a masked
// texture) for walls.
// Can draw or mark the starting pixel of floor and ceiling
// textures.
// CALLED: CORE LOOPING ROUTINE.
//
#define HEIGHTBITS          12
#define HEIGHTUNIT          (1<<HEIGHTBITS)

void R_RenderSegLoop (void)
{
    angle_t          angle;
    unsigned          index;
    int               yl;
    int               yh;

```

```

int                mid;
fixed_t           texturecolumn;
int               top;
int               bottom;

//texturecolumn = 0;                                // shut up compiler warning

for ( ; rw_x < rw_stopx ; rw_x++)
{
    // mark floor / ceiling areas
    yl = (topfrac+HEIGHTUNIT-1)>>HEIGHTBITS;

    // no space above wall?
    if (yl < ceilingclip[rw_x]+1)
        yl = ceilingclip[rw_x]+1;

    if (markceiling)
    {
        top = ceilingclip[rw_x]+1;
        bottom = yl-1;

        if (bottom >= floorclip[rw_x])
            bottom = floorclip[rw_x]-1;

        if (top <= bottom)
        {
            ceilingplane->top[rw_x] = top;
            ceilingplane->bottom[rw_x] = bottom;
        }
    }

    yh = bottomfrac>>HEIGHTBITS;

    if (yh >= floorclip[rw_x])
        yh = floorclip[rw_x]-1;

    if (markfloor)
    {
        top = yh+1;
        bottom = floorclip[rw_x]-1;
        if (top <= ceilingclip[rw_x])
            top = ceilingclip[rw_x]+1;
        if (top <= bottom)
        {
            floorplane->top[rw_x] = top;
            floorplane->bottom[rw_x] = bottom;
        }
    }

    // texturecolumn and lighting are independent of wall tiers
    if (segtextured)
    {
        // calculate texture offset
        angle = (rw_centerangle + xtoviewangle[rw_x])>>ANGLETOFINESHIFT;
        texturecolumn = rw_offset-FixedMul(finetangent[angle],rw_distance);
        texturecolumn >>= FRACBITS;
        // calculate lighting
        index = rw_scale>>LIGHTSCAleshift;

        if (index >= MAXLIGHTSCALE )
            index = MAXLIGHTSCALE-1;

        dc_colormap = walllights[index];
        dc_x = rw_x;
        dc_iscale = 0xffffffffu / (unsigned)rw_scale;
    }
}

```

```

}

// draw the wall tiers
if (midtexture)
{
    // single sided line
    dc_yl = yl;
    dc_yh = yh;
    dc_texturemid = rw_midtexturemid;
    dc_source = R_GetColumn(midtexture,texturecolumn);
    colfunc ();
    ceilingclip[rw_x] = viewheight;
    floorclip[rw_x] = -1;
}
else
{
    // two sided line
    if (toptexture)
    {
        // top wall
        mid = pixhigh>>HEIGHTBITS;
        pixhigh += pixhighstep;

        if (mid >= floorclip[rw_x])
            mid = floorclip[rw_x]-1;

        if (mid >= yl)
        {
            dc_yl = yl;
            dc_yh = mid;
            dc_texturemid = rw_toptexturemid;
            dc_source = R_GetColumn(toptexture,texturecolumn);
            colfunc ();
            ceilingclip[rw_x] = mid;
        }
        else
            ceilingclip[rw_x] = yl-1;
    }
    else
    {
        // no top wall
        if (markceiling)
            ceilingclip[rw_x] = yl-1;
    }
}

if (bottomtexture)
{
    // bottom wall
    mid = (pixlow+HEIGHTUNIT-1)>>HEIGHTBITS;
    pixlow += pixlowstep;

    // no space above wall?
    if (mid <= ceilingclip[rw_x])
        mid = ceilingclip[rw_x]+1;

    if (mid <= yh)
    {
        dc_yl = mid;
        dc_yh = yh;
        dc_texturemid = rw_bottomtexturemid;
        dc_source = R_GetColumn(bottomtexture,
                                texturecolumn);

        colfunc ();
        floorclip[rw_x] = mid;
    }
}

```

```

        else
            floorclip[rw_x] = yh+1;
    }
    else
    {
        // no bottom wall
        if (markfloor)
            floorclip[rw_x] = yh+1;
    }

    if (maskedtexture)
    {
        // save texturecol
        // for backdrawing of masked mid texture
        maskedtexturecol[rw_x] = texturecolumn;
    }
}

rw_scale += rw_scalestep;
topfrac += topstep;
bottomfrac += bottomstep;
}
}

```

```

//
// R_StoreWallRange
// A wall segment will be drawn
// between start and stop pixels (inclusive).
//
void
R_StoreWallRange
( int      start,
  int      stop )
{
    fixed_t      hyp;
    fixed_t      sineval;
    angle_t      distangle, offsetangle;
    fixed_t      vtop;
    int           lightnum;

    // don't overflow and crash
    if (ds_p == &drawsegs[MAXDRAWSEGS])
        return;

#ifdef RANGECHECK
    if (start >=viewwidth || start > stop)
        I_Error ("Bad R_RenderWallRange: %i to %i", start , stop);
#endif

    sidedef = curline->sidedef;
    linedef = curline->linedef;

    // mark the segment as visible for auto map
    linedef->flags |= ML_MAPPED;

    // calculate rw_distance for scale calculation
    rw_normalangle = curline->angle + ANG90;
    offsetangle = abs(rw_normalangle-rw_angle1);

    if (offsetangle > ANG90)
        offsetangle = ANG90;

```

```

distangle = ANG90 - offsetangle;
hyp = R_PointToDist (curline->v1->x, curline->v1->y);
sineval = finesine[distangle>>ANGLETOFINESHIFT];
rw_distance = FixedMul (hyp, sineval);

ds_p->x1 = rw_x = start;
ds_p->x2 = stop;
ds_p->curline = curline;
rw_stopx = stop+1;

// calculate scale at both ends and step
ds_p->scale1 = rw_scale =
    R_ScaleFromGlobalAngle (viewangle + xtoviewangle[start]);

if (stop > start )
{
    ds_p->scale2 = R_ScaleFromGlobalAngle (viewangle + xtoviewangle[stop]);
    ds_p->scaletestep = rw_scaletestep =
        (ds_p->scale2 - rw_scale) / (stop-start);
}
else
{
    // UNUSED: try to fix the stretched line bug
#if 0
    if (rw_distance < FRACUNIT/2)
    {
        fixed_t            trx,try;
        fixed_t            gxt,gyt;

        trx = curline->v1->x - viewx;
        try = curline->v1->y - viewy;

        gxt = FixedMul(trx,viewcos);
        gyt = -FixedMul(try,viewsin);
        ds_p->scale1 = FixedDiv(projection, gxt-gyt)<<detailshift;
    }
#endif
    ds_p->scale2 = ds_p->scale1;
}

// calculate texture boundaries
// and decide if floor / ceiling marks are needed
worldtop = frontsector->ceilingheight - viewz;
worldbottom = frontsector->floorheight - viewz;

midtexture = toptexture = bottomtexture = maskedtexture = 0;
ds_p->maskedtexturecol = NULL;

if (!backsector)
{
    // single sided line
    midtexture = texturetranslation[sidedef->midtexture];
    // a single sided line is terminal, so it must mark ends
    markfloor = markceiling = true;
    if (linedef->flags & ML_DONTPEGBOTTOM)
    {
        vtop = frontsector->floorheight +
            textureheight[sidedef->midtexture];
        // bottom of texture at bottom
        rw_midtexturemid = vtop - viewz;
    }
    else
    {
        // top of texture at top

```

```

        rw_midtexturemid = worldtop;
    }
    rw_midtexturemid += sidedef->rowoffset;

    ds_p->silhouette = SIL_BOTH;
    ds_p->sprtopclip = screenheightarray;
    ds_p->sprbottomclip = negonearray;
    ds_p->bsilheight = MAXINT;
    ds_p->tsilheight = MININT;
}
else
{
    // two sided line
    ds_p->sprtopclip = ds_p->sprbottomclip = NULL;
    ds_p->silhouette = 0;

    if (frontsector->floorheight > backsector->floorheight)
    {
        ds_p->silhouette = SIL_BOTTOM;
        ds_p->bsilheight = frontsector->floorheight;
    }
    else if (backsector->floorheight > viewz)
    {
        ds_p->silhouette = SIL_BOTTOM;
        ds_p->bsilheight = MAXINT;
        // ds_p->sprbottomclip = negonearray;
    }

    if (frontsector->ceilingheight < backsector->ceilingheight)
    {
        ds_p->silhouette |= SIL_TOP;
        ds_p->tsilheight = frontsector->ceilingheight;
    }
    else if (backsector->ceilingheight < viewz)
    {
        ds_p->silhouette |= SIL_TOP;
        ds_p->tsilheight = MININT;
        // ds_p->sprtopclip = screenheightarray;
    }

    if (backsector->ceilingheight <= frontsector->floorheight)
    {
        ds_p->sprbottomclip = negonearray;
        ds_p->bsilheight = MAXINT;
        ds_p->silhouette |= SIL_BOTTOM;
    }

    if (backsector->floorheight >= frontsector->ceilingheight)
    {
        ds_p->sprtopclip = screenheightarray;
        ds_p->tsilheight = MININT;
        ds_p->silhouette |= SIL_TOP;
    }

    worldhigh = backsector->ceilingheight - viewz;
    worldlow = backsector->floorheight - viewz;

    // hack to allow height changes in outdoor areas
    if (frontsector->ceilingpic == skyflatnum
        && backsector->ceilingpic == skyflatnum)
    {
        worldtop = worldhigh;
    }
}

```

```

if (worldlow != worldbottom
    || backsector->floorpic != frontsector->floorpic
    || backsector->lightlevel != frontsector->lightlevel)
{
    markfloor = true;
}
else
{
    // same plane on both sides
    markfloor = false;
}

if (worldhigh != worldtop
    || backsector->ceilingpic != frontsector->ceilingpic
    || backsector->lightlevel != frontsector->lightlevel)
{
    markceiling = true;
}
else
{
    // same plane on both sides
    markceiling = false;
}

if (backsector->ceilingheight <= frontsector->floorheight
    || backsector->floorheight >= frontsector->ceilingheight)
{
    // closed door
    markceiling = markfloor = true;
}

if (worldhigh < worldtop)
{
    // top texture
    toptexture = texturetranslation[sidedef->toptexture];
    if (linedef->flags & ML_DONTPEGTOP)
    {
        // top of texture at top
        rw_toptexturemid = worldtop;
    }
    else
    {
        vtop =
            backsector->ceilingheight
            + textureheight[sidedef->toptexture];

        // bottom of texture
        rw_toptexturemid = vtop - viewz;
    }
}
if (worldlow > worldbottom)
{
    // bottom texture
    bottomtexture = texturetranslation[sidedef->bottomtexture];

    if (linedef->flags & ML_DONTPEGBOTTOM )
    {
        // bottom of texture at bottom
        // top of texture at top
        rw_bottomtexturemid = worldtop;
    }
    else
        // top of texture at top
        rw_bottomtexturemid = worldlow;
}

```



```

}
rw_toptexturemid += sidedef->rowoffset;
rw_bottomtexturemid += sidedef->rowoffset;

// allocate space for masked texture tables
if (sidedef->midtexture)
{
    // masked midtexture
    maskedtexture = true;
    ds_p->maskedtexturecol = maskedtexturecol = lastopening - rw_x;
    lastopening += rw_stopx - rw_x;
}
}

// calculate rw_offset (only needed for textured lines)
segtextured = midtexture | toptexture | bottomtexture | maskedtexture;

if (segtextured)
{
    offsetangle = rw_normalangle-rw_angle1;

    if (offsetangle > ANG180)
        offsetangle = -offsetangle;

    if (offsetangle > ANG90)
        offsetangle = ANG90;

    sineval = finesine[offsetangle >> ANGLETOFINESHIFT];
    rw_offset = FixedMul (hyp, sineval);

    if (rw_normalangle-rw_angle1 < ANG180)
        rw_offset = -rw_offset;

    rw_offset += sidedef->textureoffset + curline->offset;
    rw_centerangle = ANG90 + viewangle - rw_normalangle;

    // calculate light table
    // use different light tables
    // for horizontal / vertical / diagonal
    // OPTIMIZE: get rid of LIGHTSEGSHIFT globally
    if (!fixedcolormap)
    {
        lightnum = (frontsector->lightlevel >> LIGHTSEGSHIFT)+extralight;

        if (curline->v1->y == curline->v2->y)
            lightnum--;
        else if (curline->v1->x == curline->v2->x)
            lightnum++;

        if (lightnum < 0)
            walllights = scalelight[0];
        else if (lightnum >= LIGHTLEVELS)
            walllights = scalelight[LIGHTLEVELS-1];
        else
            walllights = scalelight[lightnum];
    }
}

// if a floor / ceiling plane is on the wrong side
// of the view plane, it is definitely invisible
// and doesn't need to be marked.

if (frontsector->floorheight >= viewz)
{

```

```

    // above view plane
    markfloor = false;
}

if (frontsector->ceilingheight <= viewz
    && frontsector->ceilingpic != skyflatnum)
{
    // below view plane
    markceiling = false;
}

// calculate incremental stepping values for texture edges
worldtop >>= 4;
worldbottom >>= 4;

topstep = -FixedMul (rw_scalestep, worldtop);
topfrac = (centeryfrac>>4) - FixedMul (worldtop, rw_scale);

bottomstep = -FixedMul (rw_scalestep, worldbottom);
bottomfrac = (centeryfrac>>4) - FixedMul (worldbottom, rw_scale);

if (backsector)
{
    worldhigh >>= 4;
    worldlow >>= 4;

    if (worldhigh < worldtop)
    {
        pixhigh = (centeryfrac>>4) - FixedMul (worldhigh, rw_scale);
        pixhighstep = -FixedMul (rw_scalestep, worldhigh);
    }

    if (worldlow > worldbottom)
    {
        pixlow = (centeryfrac>>4) - FixedMul (worldlow, rw_scale);
        pixlowstep = -FixedMul (rw_scalestep, worldlow);
    }
}

// render it
if (markceiling)
    ceilingplane = R_CheckPlane (ceilingplane, rw_x, rw_stopx-1);

if (markfloor)
    floorplane = R_CheckPlane (floorplane, rw_x, rw_stopx-1);

R_RenderSegLoop ();

// save sprite clipping info
if ( ((ds_p->silhouette & SIL_TOP) || maskedtexture)
    && !ds_p->sprtopclip)
{
    memcpy (lastopening, ceilingclip+start, 2*(rw_stopx-start));
    ds_p->sprtopclip = lastopening - start;
    lastopening += rw_stopx - start;
}

if ( ((ds_p->silhouette & SIL_BOTTOM) || maskedtexture)
    && !ds_p->sprbottomclip)
{
    memcpy (lastopening, floorclip+start, 2*(rw_stopx-start));
    ds_p->sprbottomclip = lastopening - start;
    lastopening += rw_stopx - start;
}

```

```

}

if (maskedtexture && !(ds_p->silhouette&SIL_TOP))
{
    ds_p->silhouette |= SIL_TOP;
    ds_p->tsilheight = MININT;
}
if (maskedtexture && !(ds_p->silhouette&SIL_BOTTOM))
{
    ds_p->silhouette |= SIL_BOTTOM;
    ds_p->bsilheight = MAXINT;
}
ds_p++;
}

```

## 10.14 r\_segs.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Refresh module, drawing LineSegs from BSP.
//
//-----

#ifdef __R_SEGS__
#define __R_SEGS__

#ifdef __GNUG__
#pragma interface
#endif

void
R_RenderMaskedSegRange
( drawseg_t*      ds,
  int              x1,
  int              x2 );

#endif
//-----
//
// $Log:$
//
//-----

```

## 10.15 r\_sky.c

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
// Sky rendering. The DOOM sky is a texture map like any
// wall, wrapping around. A 1024 columns equal 360 degrees.
// The default sky map is 256 columns and repeats 4 times
// on a 320 screen?
//
//
//-----

static const char
rcsid[] = "$Id: m_bbox.c,v 1.1 1997/02/03 22:45:10 b1 Exp $";

// Needed for FRACUNIT.
#include "m_fixed.h"

// Needed for Flat retrieval.
#include "r_data.h"

#ifdef __GNUG__
#pragma implementation "r_sky.h"
#endif
#include "r_sky.h"

//
// sky mapping
//
int          skyflatnum;
int          skytexture;
int          skytexturemid;

//
// R_InitSkyMap
// Called whenever the view size changes.
//
void R_InitSkyMap (void)
{
    // skyflatnum = R_FlatNumForName ( SKYFLATNAME );
    skytexturemid = 100*FRACUNIT;
}
```

## 10.16 r\_sky.h

```
// Emacs style mode select    -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Sky rendering.
//-----

#ifdef __R_SKY__
#define __R_SKY__

#ifdef __GNUG__
#pragma interface
#endif

// SKY, store the number for name.
#define SKYFLATNAME "F_SKY1"

// The sky map is 256*128*4 maps.
#define ANGLETOSKYSHIFT 22

extern int skytexture;
extern int skytexturemid;

// Called whenever the view size changes.
void R_InitSkyMap (void);

#endif
//-----
//
// $Log:$
//-----
```

## 10.17 r\_state.h

```
// Emacs style mode select    -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
```

```

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
// Refresh/render internal state variables (global).
//
//-----

#ifndef __R_STATE__
#define __R_STATE__

// Need data structure definitions.
#include "d_player.h"
#include "r_data.h"

#ifdef __GNUG__
#pragma interface
#endif

//
// Refresh internal data structures,
// for rendering.
//

// needed for texture pegging
extern fixed_t* textureheight;

// needed for pre rendering (fracs)
extern fixed_t* spritewidth;

extern fixed_t* spriteoffset;
extern fixed_t* spritetopoffset;

extern lighttable_t* colormaps;

extern int viewwidth;
extern int scaledviewwidth;
extern int viewheight;

extern int firstflat;

// for global animation
extern int* flattranslation;
extern int* texturetranslation;

// Sprite....
extern int firstspritelump;
extern int lastspritelump;
extern int numspritelumps;

//
// Lookup tables for map data.
//
extern int numsprites;
extern spritedef_t* sprites;

```

```

extern int          numvertexes;
extern vertex_t*    vertexes;

extern int          numsegs;
extern seg_t*       segs;

extern int          numsectors;
extern sector_t*    sectors;

extern int          numsubsectors;
extern subsector_t* subsectors;

extern int          numnodes;
extern node_t*      nodes;

extern int          numlines;
extern line_t*      lines;

extern int          numsides;
extern side_t*      sides;

//
// POV data.
//
extern fixed_t      viewx;
extern fixed_t      viewy;
extern fixed_t      viewz;

extern angle_t      viewangle;
extern player_t*    viewplayer;

// ?
extern angle_t      clipangle;

extern int          viewangletox[FINEANGLES/2];
extern angle_t      xtoviewangle[SCREENWIDTH+1];
//extern fixed_t      finetangent[FINEANGLES/2];

extern fixed_t      rw_distance;
extern angle_t      rw_normalangle;

// angle to line origin
extern int          rw_angle1;

// Segs count?
extern int          sscount;

extern visplane_t*  floorplane;
extern visplane_t*  ceilingplane;

#endif
//-----
//
// $Log:$
//
//-----

```

## 10.18 r\_things.c

```
// Emacs style mode select    -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Refresh of things, i.e. objects represented by sprites.
//-----

static const char
rcsid[] = "$Id: r_things.c,v 1.5 1997/02/03 16:47:56 b1 Exp $";

#include <stdio.h>
#include <stdlib.h>

#include "doomdef.h"
#include "m_swap.h"

#include "i_system.h"
#include "z_zone.h"
#include "w_wad.h"

#include "r_local.h"

#include "doomstat.h"

#define MINZ                                (FRACUNIT*4)
#define BASEYCENTER                        100

//void R_DrawColumn (void);
//void R_DrawFuzzColumn (void);

typedef struct
{
    int                x1;
    int                x2;

    int                column;
    int                topclip;
    int                bottomclip;
} maskdraw_t;
```



```

//
// Sprite rotation 0 is facing the viewer,
// rotation 1 is one angle turn CLOCKWISE around the axis.
// This is not the same as the angle,
// which increases counter clockwise (protractor).
// There was a lot of stuff grabbed wrong, so I changed it...
//
fixed_t            pspritescale;
fixed_t            pspriteiscale;

lighttable_t**     spritelights;

// constant arrays
// used for psprite clipping and initializing clipping
short              negonearray[SCREENWIDTH];
short              screenheightarray[SCREENWIDTH];

//
// INITIALIZATION FUNCTIONS
//

// variables used to look up
// and range check thing_t sprites patches
spritedef_t*       sprites;
int                numsprites;

spriteframe_t      sprtemp[29];
int                maxframe;
char*              spritename;

//
// R_InstallSpriteLump
// Local function for R_InitSprites.
//
void
R_InstallSpriteLump
( int                lump,
  unsigned           frame,
  unsigned           rotation,
  boolean           flipped )
{
    int              r;

    if (frame >= 29 || rotation > 8)
        I_Error("R_InstallSpriteLump: "
                "Bad frame characters in lump %i", lump);

    if ((int)frame > maxframe)
        maxframe = frame;

    if (rotation == 0)
    {
        // the lump should be used for all rotations
        if (sprtemp[frame].rotate == false)
            I_Error ("R_InitSprites: Sprite %s frame %c has "
                    "multip rot=0 lump", spritename, 'A'+frame);

        if (sprtemp[frame].rotate == true)

```

```

        I_Error ("R_InitSprites: Sprite %s frame %c has rotations "
                "and a rot=0 lump", spritename, 'A'+frame);

    sprtemp[frame].rotate = false;
    for (r=0 ; r<8 ; r++)
    {
        sprtemp[frame].lump[r] = lump - firstspritelump;
        sprtemp[frame].flip[r] = (byte)flipped;
    }
    return;
}

// the lump is only used for one rotation
if (sprtemp[frame].rotate == false)
    I_Error ("R_InitSprites: Sprite %s frame %c has rotations "
            "and a rot=0 lump", spritename, 'A'+frame);

sprtemp[frame].rotate = true;

// make 0 based
rotation--;
if (sprtemp[frame].lump[rotation] != -1)
    I_Error ("R_InitSprites: Sprite %s : %c : %c "
            "has two lumps mapped to it",
            spritename, 'A'+frame, '1'+rotation);

sprtemp[frame].lump[rotation] = lump - firstspritelump;
sprtemp[frame].flip[rotation] = (byte)flipped;
}

//
// R_InitSpriteDefs
// Pass a null terminated list of sprite names
// (4 chars exactly) to be used.
// Builds the sprite rotation matrixes to account
// for horizontally flipped sprites.
// Will report an error if the lumps are inconsistent.
// Only called at startup.
//
// Sprite lump names are 4 characters for the actor,
// a letter for the frame, and a number for the rotation.
// A sprite that is flippable will have an additional
// letter/number appended.
// The rotation character can be 0 to signify no rotations.
//
void R_InitSpriteDefs (char** namelist)
{
    char**      check;
    int         i;
    int         l;
    int         inthname;
    int         frame;
    int         rotation;
    int         start;
    int         end;
    int         patched;

    // count the number of sprite names
    check = namelist;
    while (*check != NULL)
        check++;

```

```

numsprites = check-namelist;

if (!numsprites)
    return;

sprites = Z_Malloc(numsprites * sizeof(*sprites), PU_STATIC, NULL);

start = firstspritelump-1;
end = lastspritelump+1;

// scan all the lump names for each of the names,
// noting the highest frame letter.
// Just compare 4 characters as ints
for (i=0 ; i<numsprites ; i++)
{
    spritename = namelist[i];
    memset (sprtemp,-1, sizeof(sprtemp));

    maxframe = -1;
    intname = *(int *)namelist[i];

    // scan the lumps,
    // filling in the frames for whatever is found
    for (l=start+1 ; l<end ; l++)
    {
        if (*(int *)lumpinfo[l].name == intname)
        {
            frame = lumpinfo[l].name[4] - 'A';
            rotation = lumpinfo[l].name[5] - '0';

            if (modifiedgame)
                patched = W_GetNumForName (lumpinfo[l].name);
            else
                patched = l;

            R_InstallSpriteLump (patched, frame, rotation, false);

            if (lumpinfo[l].name[6])
            {
                frame = lumpinfo[l].name[6] - 'A';
                rotation = lumpinfo[l].name[7] - '0';
                R_InstallSpriteLump (l, frame, rotation, true);
            }
        }
    }

    // check the frames that were found for completeness
    if (maxframe == -1)
    {
        sprites[i].numframes = 0;
        continue;
    }

    maxframe++;

    for (frame = 0 ; frame < maxframe ; frame++)
    {
        switch ((int)sprtemp[frame].rotate)
        {
            case -1:
                // no rotations were found for that frame at all
                I_Error ("R_InitSprites: No patches found "
                    "for %s frame %c", namelist[i], frame+'A');
                break;
        }
    }
}

```

```

        case 0:
            // only the first rotation is needed
            break;

        case 1:
            // must have all 8 frames
            for (rotation=0 ; rotation<8 ; rotation++)
                if (sprtemp[frame].lump[rotation] == -1)
                    I_Error ("R_InitSprites: Sprite %s frame %c "
                            "is missing rotations",
                            namelist[i], frame+'A');
            break;
    }
}

// allocate space for the frames present and copy sprtemp to it
sprites[i].numframes = maxframe;
sprites[i].spriteframes =
    Z_Malloc (maxframe * sizeof(spriteframe_t), PU_STATIC, NULL);
memcpy (sprites[i].spriteframes, sprtemp, maxframe*sizeof(spriteframe_t));
}
}

```

```

//
// GAME FUNCTIONS
//
vissprite_t      vissprites[MAXVISSPRITES];
vissprite_t*     vissprite_p;
int              newvissprite;

```

```

//
// R_InitSprites
// Called at program start.
//
void R_InitSprites (char** namelist)
{
    int          i;

    for (i=0 ; i<SCREENWIDTH ; i++)
    {
        negonearray[i] = -1;
    }

    R_InitSpriteDefs (namelist);
}

```

```

//
// R_ClearSprites
// Called at frame start.
//
void R_ClearSprites (void)
{
    vissprite_p = vissprites;
}

```

```

//

```

```

// R_NewVisSprite
//
vissprite_t      overflowsprite;

vissprite_t* R_NewVisSprite (void)
{
    if (vissprite_p == &vissprites[MAXVISSPRITES])
        return &overflowsprite;

    vissprite_p++;
    return vissprite_p-1;
}

//
// R_DrawMaskedColumn
// Used for sprites and masked mid textures.
// Masked means: partly transparent, i.e. stored
// in posts/runs of opaque pixels.
//
short*           mfloorclip;
short*           mceilingclip;

fixed_t          spryscale;
fixed_t          sprtopscreen;

void R_DrawMaskedColumn (column_t* column)
{
    int            topscreen;
    int            bottomscreen;
    fixed_t        basetexturemid;

    basetexturemid = dc_texturemid;

    for ( ; column->topdelta != 0xff ; )
    {
        // calculate unclipped screen coordinates
        // for post
        topscreen = sprtopscreen + spryscale*column->topdelta;
        bottomscreen = topscreen + spryscale*column->length;

        dc_yl = (topscreen+FRACUNIT-1)>>FRACBITS;
        dc_yh = (bottomscreen-1)>>FRACBITS;

        if (dc_yh >= mfloorclip[dc_x])
            dc_yh = mfloorclip[dc_x]-1;
        if (dc_yl <= mceilingclip[dc_x])
            dc_yl = mceilingclip[dc_x]+1;

        if (dc_yl <= dc_yh)
        {
            dc_source = (byte *)column + 3;
            dc_texturemid = basetexturemid - (column->topdelta<<FRACBITS);
            // dc_source = (byte *)column + 3 - column->topdelta;

            // Drawn by either R_DrawColumn
            // or (SHADOW) R_DrawFuzzColumn.
            colfunc ();
        }
        column = (column_t *)((byte *)column + column->length + 4);
    }

    dc_texturemid = basetexturemid;
}

```

```

//
// R_DrawVisSprite
// mfloorclip and mceilingclip should also be set.
//
void
R_DrawVisSprite
( vissprite_t*      vis,
  int               x1,
  int               x2 )
{
    column_t*        column;
    int              texturecolumn;
    fixed_t          frac;
    patch_t*         patch;

    patch = W_CacheLumpNum (vis->patch+firstspritelump, PU_CACHE);

    dc_colormap = vis->colormap;

    if (!dc_colormap)
    {
        // NULL colormap = shadow draw
        colfunc = fuzzcolfunc;
    }
    else if (vis->mobjflags & MF_TRANSLATION)
    {
        colfunc = R_DrawTranslatedColumn;
        dc_translation = translationtables - 256 +
            ( (vis->mobjflags & MF_TRANSLATION) >> (MF_TRANSShift-8) );
    }

    dc_iscale = abs(vis->xiscale)>>detailshift;
    dc_texturemid = vis->texturemid;
    frac = vis->startfrac;
    spryscale = vis->scale;
    sprtopscreen = centeryfrac - FixedMul(dc_texturemid,spryscale);

    for (dc_x=vis->x1 ; dc_x<=vis->x2 ; dc_x++, frac += vis->xiscale)
    {
        texturecolumn = frac>>FRACBITS;
#ifdef RANGECHECK
        if (texturecolumn < 0 || texturecolumn >= SHORT(patch->width))
            I_Error ("R_DrawSpriteRange: bad texturecolumn");
#endif
        column = (column_t *) ((byte *)patch +
                                LONG(patch->columnofs[texturecolumn]));
        R_DrawMaskedColumn (column);
    }

    colfunc = basecolfunc;
}

//
// R_ProjectSprite
// Generates a vissprite for a thing
// if it might be visible.
//
void R_ProjectSprite (mobj_t* thing)
{

```

```

fixed_t          tr_x;
fixed_t          tr_y;

fixed_t          gxt;
fixed_t          gyt;

fixed_t          tx;
fixed_t          tz;

fixed_t          xscale;

int              x1;
int              x2;

spritedef_t*     sprdef;
spriteframe_t*   sprframe;
int              lump;

unsigned         rot;
boolean          flip;

int              index;

vissprite_t*     vis;

angle_t          ang;
fixed_t          iscale;

// transform the origin point
tr_x = thing->x - viewx;
tr_y = thing->y - viewy;

gxt = FixedMul(tr_x,viewcos);
gyt = -FixedMul(tr_y,viewsin);

tz = gxt-gyt;

// thing is behind view plane?
if (tz < MINZ)
    return;

xscale = FixedDiv(projection, tz);

gxt = -FixedMul(tr_x,viewsin);
gyt = FixedMul(tr_y,viewcos);
tx = -(gyt+gxt);

// too far off the side?
if (abs(tx)>(tz<<2))
    return;

// decide which patch to use for sprite relative to player
#ifdef RANGECHECK
    if ((unsigned)thing->sprite >= numsprites)
        I_Error ("R_ProjectSprite: invalid sprite number %i ",
            thing->sprite);
#endif
    sprdef = &sprites[thing->sprite];
#ifdef RANGECHECK
    if ( (thing->frame&FF_FRAMEMASK) >= sprdef->numframes )
        I_Error ("R_ProjectSprite: invalid sprite frame %i : %i ",
            thing->sprite, thing->frame);
#endif
    sprframe = &sprdef->spriteframes[ thing->frame & FF_FRAMEMASK];

```

```

if (sprframe->rotate)
{
    // choose a different rotation based on player view
    ang = R_PointToAngle (thing->x, thing->y);
    rot = (ang-thing->angle+(unsigned)(ANG45/2)*9)>>29;
    lump = sprframe->lump[rot];
    flip = (boolean)sprframe->flip[rot];
}
else
{
    // use single rotation for all views
    lump = sprframe->lump[0];
    flip = (boolean)sprframe->flip[0];
}

// calculate edges of the shape
tx -= spriteoffset[lump];
x1 = (centerxfrac + FixedMul (tx,xscale) ) >>FRACBITS;

// off the right side?
if (x1 > viewwidth)
    return;

tx += spritewidth[lump];
x2 = ((centerxfrac + FixedMul (tx,xscale) ) >>FRACBITS) - 1;

// off the left side
if (x2 < 0)
    return;

// store information in a vissprite
vis = R_NewVisSprite ();
vis->mobjflags = thing->flags;
vis->scale = xscale<<detailshift;
vis->gx = thing->x;
vis->gy = thing->y;
vis->gz = thing->z;
vis->gzt = thing->z + spritetopoffset[lump];
vis->texturemid = vis->gzt - viewz;
vis->x1 = x1 < 0 ? 0 : x1;
vis->x2 = x2 >= viewwidth ? viewwidth-1 : x2;
iscale = FixedDiv (FRACUNIT, xscale);

if (flip)
{
    vis->startfrac = spritewidth[lump]-1;
    vis->xiscale = -iscale;
}
else
{
    vis->startfrac = 0;
    vis->xiscale = iscale;
}

if (vis->x1 > x1)
    vis->startfrac += vis->xiscale*(vis->x1-x1);
vis->patch = lump;

// get light level
if (thing->flags & MF_SHADOW)
{
    // shadow draw
    vis->colormap = NULL;
}
else if (fixedcolormap)

```



```

{
    // fixed map
    vis->colormap = fixedcolormap;
}
else if (thing->frame & FF_FULLBRIGHT)
{
    // full bright
    vis->colormap = colormap;
}

else
{
    // diminished light
    index = xscale>>(LIGHTSCAleshift-detailshift);

    if (index >= MAXLIGHTSCALE)
        index = MAXLIGHTSCALE-1;

    vis->colormap = spritelights[index];
}
}

//
// R_AddSprites
// During BSP traversal, this adds sprites by sector.
//
void R_AddSprites (sector_t* sec)
{
    mobj_t*      thing;
    int          lightnum;

    // BSP is traversed by subsector.
    // A sector might have been split into several
    // subsectors during BSP building.
    // Thus we check whether its already added.
    if (sec->validcount == validcount)
        return;

    // Well, now it will be done.
    sec->validcount = validcount;

    lightnum = (sec->lightlevel >> LIGHTSEGSshift)+extralight;

    if (lightnum < 0)
        spritelights = scalelight[0];
    else if (lightnum >= LIGHTLEVELS)
        spritelights = scalelight[LIGHTLEVELS-1];
    else
        spritelights = scalelight[lightnum];

    // Handle all things in sector.
    for (thing = sec->thinglist ; thing ; thing = thing->snext)
        R_ProjectSprite (thing);
}

//
// R_DrawPSprite
//
void R_DrawPSprite (pspdef_t* psp)
{
    fixed_t      tx;

```

```

int                x1;
int                x2;
spritedef_t*      sprdef;
spriteframe_t*    sprframe;
int               lump;
boolean           flip;
vissprite_t*      vis;
vissprite_t       avis;

// decide which patch to use
#ifdef RANGECHECK
    if ( (unsigned)psp->state->sprite >= numsprites)
        I_Error ("R_ProjectSprite: invalid sprite number %i ",
            psp->state->sprite);
#endif
    sprdef = &sprites[psp->state->sprite];
#ifdef RANGECHECK
    if ( (psp->state->frame & FF_FRAMEMASK) >= sprdef->numframes)
        I_Error ("R_ProjectSprite: invalid sprite frame %i : %i ",
            psp->state->sprite, psp->state->frame);
#endif
    sprframe = &sprdef->spriteframes[ psp->state->frame & FF_FRAMEMASK ];

    lump = sprframe->lump[0];
    flip = (boolean)sprframe->flip[0];

// calculate edges of the shape
tx = psp->sx-160*FRACUNIT;

tx -= spriteoffset[lump];
x1 = (centerxfrac + FixedMul (tx,pspritescale) ) >>FRACBITS;

// off the right side
if (x1 > viewwidth)
    return;

tx += spritewidth[lump];
x2 = ((centerxfrac + FixedMul (tx, pspritescale) ) >>FRACBITS) - 1;

// off the left side
if (x2 < 0)
    return;

// store information in a vissprite
vis = &avis;
vis->objjflags = 0;
vis->texturemid = (BASEYCENTER<<FRACBITS)+FRACUNIT/2-(psp->sy-spritetopoffset[lump]);
vis->x1 = x1 < 0 ? 0 : x1;
vis->x2 = x2 >= viewwidth ? viewwidth-1 : x2;
vis->scale = pspritescale<<detailshift;

if (flip)
{
    vis->xiscale = -pspriteiscale;
    vis->startfrac = spritewidth[lump]-1;
}
else
{
    vis->xiscale = pspriteiscale;
    vis->startfrac = 0;
}

if (vis->x1 > x1)
    vis->startfrac += vis->xiscale*(vis->x1-x1);

```

```

vis->patch = lump;

if (viewplayer->powers[pw_invisibility] > 4*32
    || viewplayer->powers[pw_invisibility] & 8)
{
    // shadow draw
    vis->colormap = NULL;
}
else if (fixedcolormap)
{
    // fixed color
    vis->colormap = fixedcolormap;
}
else if (psp->state->frame & FF_FULLBRIGHT)
{
    // full bright
    vis->colormap = colormap;
}
else
{
    // local light
    vis->colormap = spritelights[MAXLIGHTSCALE-1];
}

R_DrawVisSprite (vis, vis->x1, vis->x2);
}

//
// R_DrawPlayerSprites
//
void R_DrawPlayerSprites (void)
{
    int i;
    int lightnum;
    pspdef_t* psp;

    // get light level
    lightnum =
        (viewplayer->mo->subsector->sector->lightlevel >> LIGHTSEGSHIFT)
        +extralight;

    if (lightnum < 0)
        spritelights = scalelight[0];
    else if (lightnum >= LIGHTLEVELS)
        spritelights = scalelight[LIGHTLEVELS-1];
    else
        spritelights = scalelight[lightnum];

    // clip to screen bounds
    mfloorclip = screenheightarray;
    mceilingclip = negonearray;

    // add all active psprites
    for (i=0, psp=viewplayer->psprites;
        i<NUMPSPRITES;
        i++,psp++)
    {
        if (psp->state)
            R_DrawPSprite (psp);
    }
}

```

```

//
// R_SortVisSprites
//
vissprite_t      vsprsortedhead;

void R_SortVisSprites (void)
{
    int                i;
    int                count;
    vissprite_t*       ds;
    vissprite_t*       best;
    vissprite_t        unsorted;
    fixed_t            bestscale;

    count = vissprite_p - vissprites;

    unsorted.next = unsorted.prev = &unsorted;

    if (!count)
        return;

    for (ds=vissprites ; ds<vissprite_p ; ds++)
    {
        ds->next = ds+1;
        ds->prev = ds-1;
    }

    vissprites[0].prev = &unsorted;
    unsorted.next = &vissprites[0];
    (vissprite_p-1)->next = &unsorted;
    unsorted.prev = vissprite_p-1;

    // pull the vissprites out by scale
    //best = 0;                // shut up the compiler warning
    vsprsortedhead.next = vsprsortedhead.prev = &vsprsortedhead;
    for (i=0 ; i<count ; i++)
    {
        bestscale = MAXINT;
        for (ds=unsorted.next ; ds!= &unsorted ; ds=ds->next)
        {
            if (ds->scale < bestscale)
            {
                bestscale = ds->scale;
                best = ds;
            }
        }
        best->next->prev = best->prev;
        best->prev->next = best->next;
        best->next = &vsprsortedhead;
        best->prev = vsprsortedhead.prev;
        vsprsortedhead.prev->next = best;
        vsprsortedhead.prev = best;
    }
}

//
// R_DrawSprite
//
void R_DrawSprite (vissprite_t* spr)
{

```

```

drawseg_t*          ds;
short               clipbot[SCREENWIDTH];
short               cliptop[SCREENWIDTH];
int                 x;
int                 r1;
int                 r2;
fixed_t             scale;
fixed_t             lowscale;
int                 silhouette;

for (x = spr->x1 ; x<=spr->x2 ; x++)
    clipbot[x] = cliptop[x] = -2;

// Scan drawsegs from end to start for obscuring segs.
// The first drawseg that has a greater scale
// is the clip seg.
for (ds=ds_p-1 ; ds >= drawsegs ; ds--)
{
    // determine if the drawseg obscures the sprite
    if (ds->x1 > spr->x2
        || ds->x2 < spr->x1
        || (!ds->silhouette
            && !ds->maskedtexturecol) )
    {
        // does not cover sprite
        continue;
    }

    r1 = ds->x1 < spr->x1 ? spr->x1 : ds->x1;
    r2 = ds->x2 > spr->x2 ? spr->x2 : ds->x2;

    if (ds->scale1 > ds->scale2)
    {
        lowscale = ds->scale2;
        scale = ds->scale1;
    }
    else
    {
        lowscale = ds->scale1;
        scale = ds->scale2;
    }

    if (scale < spr->scale
        || ( lowscale < spr->scale
            && !R_PointOnSegSide (spr->gx, spr->gy, ds->curline) ) )
    {
        // masked mid texture?
        if (ds->maskedtexturecol)
            R_RenderMaskedSegRange (ds, r1, r2);
        // seg is behind sprite
        continue;
    }

    // clip this piece of the sprite
    silhouette = ds->silhouette;

    if (spr->gz >= ds->bsilheight)
        silhouette &= ~SIL_BOTTOM;

    if (spr->gzt <= ds->tsilheight)
        silhouette &= ~SIL_TOP;

    if (silhouette == 1)
    {

```

```

        // bottom sil
        for (x=r1 ; x<=r2 ; x++)
            if (clipbot[x] == -2)
                clipbot[x] = ds->sprbottomclip[x];
    }
    else if (silhouette == 2)
    {
        // top sil
        for (x=r1 ; x<=r2 ; x++)
            if (cliptop[x] == -2)
                cliptop[x] = ds->sprtopclip[x];
    }
    else if (silhouette == 3)
    {
        // both
        for (x=r1 ; x<=r2 ; x++)
        {
            if (clipbot[x] == -2)
                clipbot[x] = ds->sprbottomclip[x];
            if (cliptop[x] == -2)
                cliptop[x] = ds->sprtopclip[x];
        }
    }
}

// all clipping has been performed, so draw the sprite

// check for unclipped columns
for (x = spr->x1 ; x<=spr->x2 ; x++)
{
    if (clipbot[x] == -2)
        clipbot[x] = viewheight;

    if (cliptop[x] == -2)
        cliptop[x] = -1;
}

mfloorclip = clipbot;
mceilingclip = cliptop;
R_DrawVisSprite (spr, spr->x1, spr->x2);
}

```

```

//
// R_DrawMasked
//
void R_DrawMasked (void)
{
    vissprite_t*      spr;
    drawseg_t*        ds;

    R_SortVisSprites ();

    if (vissprite_p > vissprites)
    {
        // draw all vissprites back to front
        for (spr = vsprsortedhead.next ;
             spr != &vsprsortedhead ;
             spr=spr->next)
        {
            R_DrawSprite (spr);
        }
    }
}

```

```

    }
}

// render any remaining masked mid textures
for (ds=ds_p-1 ; ds >= drawsegs ; ds--)
    if (ds->maskedtexturecol)
        R_RenderMaskedSegRange (ds, ds->x1, ds->x2);

// draw the psprites on top of everything
// but does not draw on side views
if (!viewangleoffset)
    R_DrawPlayerSprites ();
}

```

## 10.19 r\_things.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     Rendering of moving objects, sprites.
//
//-----

#ifndef __R_THINGS__
#define __R_THINGS__

#ifdef __GNUG__
#pragma interface
#endif

#define MAXVISSPRITES          128

extern vissprite_t             vissprites[MAXVISSPRITES];
extern vissprite_t*           vissprite_p;
extern vissprite_t             vsprsortedhead;

// Constant arrays used for psprite clipping
// and initializing clipping.
extern short                   negonearray[SCREENWIDTH];
extern short                   screenheightarray[SCREENWIDTH];

// vars for R_DrawMaskedColumn
extern short*                  mfloorclip;
extern short*                  mceilingclip;
extern fixed_t                 spryscale;

```

```

extern fixed_t          sprtopscreen;

extern fixed_t          pspritescale;
extern fixed_t          pspriteiscale;

void R_DrawMaskedColumn (column_t* column);

void R_SortVisSprites (void);

void R_AddSprites (sector_t* sec);
void R_AddPSprites (void);
void R_DrawSprites (void);
void R_InitSprites (char** namelist);
void R_ClearSprites (void);
void R_DrawMasked (void);

void
R_ClipVisSprite
( vissprite_t*          vis,
  int                  xl,
  int                  xh );

#endif
//-----
//
// $Log:$
//
//-----

```

## 11 Sound code

### 11.1 s\_sound.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:  none
//
//-----

static const char
rcsid[] = "$Id: s_sound.c,v 1.6 1997/02/03 22:45:12 b1 Exp $";

```



```

#include <stdio.h>
#include <stdlib.h>

#include "i_system.h"
#include "i_sound.h"
#include "sounds.h"
#include "s_sound.h"

#include "z_zone.h"
#include "m_random.h"
#include "w_wad.h"

#include "doomdef.h"
#include "p_local.h"

#include "doomstat.h"

// Purpose?
const char snd_prefixen[]
= { 'P', 'P', 'A', 'S', 'S', 'S', 'M', 'M', 'M', 'S', 'S', 'S' };

#define S_MAX_VOLUME                127

// when to clip out sounds
// Does not fit the large outdoor areas.
#define S_CLIPPING_DIST              (1200*0x10000)

// Distance tp origin when sounds should be maxed out.
// This should relate to movement clipping resolution
// (see BLOCKMAP handling).
// Originally: (200*0x10000).
#define S_CLOSE_DIST                 (160*0x10000)

#define S_ATTENUATOR                  ((S_CLIPPING_DIST-S_CLOSE_DIST)>>FRACBITS)

// Adjustable by menu.
#define NORM_VOLUME                   snd_MaxVolume

#define NORM_PITCH                    128
#define NORM_PRIORITY                  64
#define NORM_SEP                       128

#define S_PITCH_PERTURE                1
#define S_STEREO_SWING                 (96*0x10000)

// percent attenuation from front to back
#define S_IFRACVOL                     30

#define NA                             0
#define S_NUMCHANNELS                  2

// Current music/sfx card - index useless
// w/o a reference LUT in a sound module.
extern int snd_MusicDevice;
extern int snd_SfxDevice;
// Config file? Same disclaimer as above.
extern int snd_DesiredMusicDevice;
extern int snd_DesiredSfxDevice;

```

```

typedef struct

```

```

{
    // sound information (if null, channel avail.)
    sfxinfo_t*      sfxinfo;

    // origin of sound
    void*          origin;

    // handle of the sound being played
    int            handle;
} channel_t;

// the set of channels available
static channel_t*  channels;

// These are not used, but should be (menu).
// Maximum volume of a sound effect.
// Internal default is max out of 0-15.
int               snd_SfxVolume = 15;

// Maximum volume of music. Useless so far.
int               snd_MusicVolume = 15;

// whether songs are mus_paused
static boolean    mus_paused;

// music currently being played
static musicinfo_t* mus_playing=0;

// following is set
// by the defaults code in M_misc:
// number of channels available
int               numChannels;

static int        nextcleanup;

//
// Internals.
//
int
S_getChannel
( void*          origin,
  sfxinfo_t*     sfxinfo );

int
S_AdjustSoundParams
( mobj_t*        listener,
  mobj_t*        source,
  int*           vol,
  int*           sep,
  int*           pitch );

void S_StopChannel(int cnum);

//
// Initializes sound stuff, including volume
// Sets channels, SFX and music volume,

```

```

// allocates channel buffer, sets S_sfx lookup.
//
void S_Init
( int          sfxVolume,
  int          musicVolume )
{
    int          i;

    fprintf( stderr, "S_Init: default sfx volume %d\n", sfxVolume);

    // Whatever these did with DMX, these are rather dummies now.
    I_SetChannels();

    S_SetSfxVolume(sfxVolume);
    // No music with Linux - another dummy.
    S_SetMusicVolume(musicVolume);

    // Allocating the internal channels for mixing
    // (the maximum number of sounds rendered
    // simultaneously) within zone memory.
    channels =
        (channel_t *) Z_Malloc(numChannels*sizeof(channel_t), PU_STATIC, 0);

    // Free all channels for use
    for (i=0 ; i<numChannels ; i++)
        channels[i].sfxinfo = 0;

    // no sounds are playing, and they are not mus_paused
    mus_paused = 0;

    // Note that sounds have not been cached (yet).
    for (i=1 ; i<NUMSFX ; i++)
        S_sfx[i].lumpnum = S_sfx[i].usefulness = -1;
}

```

```

//
// Per level startup code.
// Kills playing sounds at start of level,
// determines music if any, changes music.
//
void S_Start(void)
{
    int cnum;
    int mnum;

    // kill all playing sounds at start of level
    // (trust me - a good idea)
    for (cnum=0 ; cnum<numChannels ; cnum++)
        if (channels[cnum].sfxinfo)
            S_StopChannel(cnum);

    // start new music for the level
    mus_paused = 0;

    if (gamemode == commercial)
        mnum = mus_runnin + gamemap - 1;
    else
    {
        int spmus[] =
        {
            // Song - Who? - Where?

```

```

    mus_e3m4,        // American      e4m1
    mus_e3m2,        // Romero        e4m2
    mus_e3m3,        // Shawn      e4m3
    mus_e1m5,        // American      e4m4
    mus_e2m7,        // Tim          e4m5
    mus_e2m4,        // Romero        e4m6
    mus_e2m6,        // J.Anderson    e4m7 CHIRON.WAD
    mus_e2m5,        // Shawn      e4m8
    mus_e1m9         // Tim          e4m9
};

if (gameepisode < 4)
    mnum = mus_e1m1 + (gameepisode-1)*9 + gamemap-1;
else
    mnum = spmus[gamemap-1];
}

// HACK FOR COMMERCIAL
// if (commercial && mnum > mus_e3m9)
//     mnum -= mus_e3m9;

S_ChangeMusic(mnum, true);

nextcleanup = 15;
}

void
S_StartSoundAtVolume
( void*          origin_p,
  int            sfx_id,
  int            volume )
{
    int          rc;
    int          sep;
    int          pitch;
    int          priority;
    sfxinfo_t*   sfx;
    int          cnum;

    mobj_t*      origin = (mobj_t *) origin_p;

    // Debug.
    /*fprintf( stderr,
               "S_StartSoundAtVolume: playing sound %d (%s)\n",
               sfx_id, S_sfx[sfx_id].name );*/

    // check for bogus sound #
    if (sfx_id < 1 || sfx_id > NUMSFX)
        I_Error("Bad sfx #: %d", sfx_id);

    sfx = &S_sfx[sfx_id];

    // Initialize sound parameters
    if (sfx->link)
    {
        pitch = sfx->pitch;
        priority = sfx->priority;
        volume += sfx->volume;
    }

```

```

    if (volume < 1)
        return;

    if (volume > snd_SfxVolume)
        volume = snd_SfxVolume;
}
else
{
    pitch = NORM_PITCH;
    priority = NORM_PRIORITY;
}

// Check to see if it is audible,
// and if not, modify the params
if (origin && origin != players[consoleplayer].mo)
{
    rc = S_AdjustSoundParams(players[consoleplayer].mo,
                             origin,
                             &volume,
                             &sep,
                             &pitch);

    if ( origin->x == players[consoleplayer].mo->x
        && origin->y == players[consoleplayer].mo->y)
    {
        sep = NORM_SEP;
    }

    if (!rc)
        return;
}
else
{
    sep = NORM_SEP;
}

// hacks to vary the sfx pitches
if (sfx_id >= sfx_sawup
    && sfx_id <= sfx_sawhit)
{
    pitch += 8 - (M_Random()&15);

    if (pitch<0)
        pitch = 0;
    else if (pitch>255)
        pitch = 255;
}
else if (sfx_id != sfx_itemup
    && sfx_id != sfx_tink)
{
    pitch += 16 - (M_Random()&31);

    if (pitch<0)
        pitch = 0;
    else if (pitch>255)
        pitch = 255;
}

// kill old sound
S_StopSound(origin);

// try to find a channel
cnum = S_getChannel(origin, sfx);

```

```

if (cnum<0)
    return;

//
// This is supposed to handle the loading/caching.
// For some odd reason, the caching is done nearly
// each time the sound is needed?
//

// get lumpnum if necessary
if (sfx->lumpnum < 0)
    sfx->lumpnum = I_GetSfxLumpNum(sfx);

#ifdef SNDSRV
    // cache data if necessary
    if (!sfx->data)
    {
        fprintf( stderr,
            "S_StartSoundAtVolume: 16bit and not pre-cached - wtf?\n");

        // DOS remains, 8bit handling
        //sfx->data = (void *) W_CacheLumpNum(sfx->lumpnum, PU_MUSIC);
        // fprintf( stderr,
        //     "S_StartSoundAtVolume: loading %d (lump %d) : 0x%x\n",
        //     sfx_id, sfx->lumpnum, (int)sfx->data );

    }
#endif

    // increase the usefulness
    if (sfx->usefulness++ < 0)
        sfx->usefulness = 1;

    // Assigns the handle to one of the channels in the
    // mix/output buffer.
    channels[cnum].handle = I_StartSound(sfx_id,
                                        /*sfx->data,*/
                                        volume,
                                        sep,
                                        pitch,
                                        priority);
}

void
S_StartSound
( void*          origin,
  int            sfx_id )
{
#ifdef SAWDEBUG
    // if (sfx_id == sfx_sawful)
    // sfx_id = sfx_itemup;
#endif

    S_StartSoundAtVolume(origin, sfx_id, snd_SfxVolume);

    // UNUSED. We had problems, had we not?
#ifdef SAWDEBUG
    {
        int i;
        int n;

        static mobj_t*    last_saw_origins[10] = {1,1,1,1,1,1,1,1,1,1};
        static int        first_saw=0;
        static int        next_saw=0;
    }
#endif

```

```

if (sfx_id == sfx_sawidl
    || sfx_id == sfx_sawful
    || sfx_id == sfx_sawhit)
{
    for (i=first_saw;i!=next_saw;i=(i+1)%10)
        if (last_saw_origins[i] != origin)
            fprintf(stderr, "old origin 0x%lx != "
                          "origin 0x%lx for sfx %d\n",
                          last_saw_origins[i],
                          origin,
                          sfx_id);

    last_saw_origins[next_saw] = origin;
    next_saw = (next_saw + 1) % 10;
    if (next_saw == first_saw)
        first_saw = (first_saw + 1) % 10;

    for (n=i=0; i<numChannels ; i++)
    {
        if (channels[i].sfxinfo == &S_sfx[sfx_sawidl]
            || channels[i].sfxinfo == &S_sfx[sfx_sawful]
            || channels[i].sfxinfo == &S_sfx[sfx_sawhit]) n++;
    }

    if (n>1)
    {
        for (i=0; i<numChannels ; i++)
        {
            if (channels[i].sfxinfo == &S_sfx[sfx_sawidl]
                || channels[i].sfxinfo == &S_sfx[sfx_sawful]
                || channels[i].sfxinfo == &S_sfx[sfx_sawhit])
            {
                fprintf(stderr,
                        "chn: sfxinfo=0x%lx, origin=0x%lx, "
                        "handle=%d\n",
                        channels[i].sfxinfo,
                        channels[i].origin,
                        channels[i].handle);
            }
        }
        fprintf(stderr, "\n");
    }
}
}
#endif
}

void S_StopSound(void *origin)
{
    int cnum;

    for (cnum=0 ; cnum<numChannels ; cnum++)
    {
        if (channels[cnum].sfxinfo && channels[cnum].origin == origin)
        {
            S_StopChannel(cnum);
            break;
        }
    }
}

```

```
}
```

```
//  
// Stop and resume music, during game PAUSE.  
//
```

```
void S_PauseSound(void)  
{  
    if (mus_playing && !mus_paused)  
    {  
        I_PauseSong(mus_playing->handle);  
        mus_paused = true;  
    }  
}
```

```
void S_ResumeSound(void)  
{  
    if (mus_playing && mus_paused)  
    {  
        I_ResumeSong(mus_playing->handle);  
        mus_paused = false;  
    }  
}
```

```
//  
// Updates music & sounds  
//
```

```
void S_UpdateSounds(void* listener_p)  
{  
    int             audible;  
    int             cnum;  
    int             volume;  
    int             sep;  
    int             pitch;  
    sfxinfo_t*      sfx;  
    channel_t*      c;  
  
    mobj_t*         listener = (mobj_t*)listener_p;
```

```
// Clean up unused data.  
// This is currently not done for 16bit (sounds cached static).  
// DOS 8bit remains.  
/*if (gametic > nextcleanup)
```

```
{  
    for (i=1 ; i<NUMSFX ; i++)  
    {  
        if (S_sfx[i].usefulness < 1  
            && S_sfx[i].usefulness > -1)  
        {  
            if (--S_sfx[i].usefulness == -1)  
            {  
                Z_ChangeTag(S_sfx[i].data, PU_CACHE);  
                S_sfx[i].data = 0;  
            }  
        }  
    }  
}
```



```

    }
    nextcleanup = gametic + 15;
}*/

for (cnum=0 ; cnum<numChannels ; cnum++)
{
    c = &channels[cnum];
    sfx = c->sfxinfo;

    if (c->sfxinfo)
    {
        if (I_SoundIsPlaying(c->handle))
        {
            // initialize parameters
            volume = snd_SfxVolume;
            pitch = NORM_PITCH;
            sep = NORM_SEP;

            if (sfx->link)
            {
                pitch = sfx->pitch;
                volume += sfx->volume;
                if (volume < 1)
                {
                    S_StopChannel(cnum);
                    continue;
                }
                else if (volume > snd_SfxVolume)
                {
                    volume = snd_SfxVolume;
                }
            }

            // check non-local sounds for distance clipping
            // or modify their params
            if (c->origin && listener_p != c->origin)
            {
                audible = S_AdjustSoundParams(listener,
                                                c->origin,
                                                &volume,
                                                &sep,
                                                &pitch);

                if (!audible)
                {
                    S_StopChannel(cnum);
                }
                else
                {
                    I_UpdateSoundParams(c->handle, volume, sep, pitch);
                }
            }
        }
        else
        {
            // if channel is allocated but sound has stopped,
            // free it
            S_StopChannel(cnum);
        }
    }
}

// kill music if it is a single-play && finished
// if (      mus_playing
//      && !I_QrySongPlaying(mus_playing->handle)
//      && !mus_paused )
// S_StopMusic();
}

```

```

void S_SetMusicVolume(int volume)
{
    if (volume < 0 || volume > 127)
    {
        I_Error("Attempt to set music volume at %d",
                volume);
    }

    I_SetMusicVolume(127);
    I_SetMusicVolume(volume);
    snd_MusicVolume = volume;
}

void S_SetSfxVolume(int volume)
{
    if (volume < 0 || volume > 127)
        I_Error("Attempt to set sfx volume at %d", volume);

    snd_SfxVolume = volume;
}

//
// Starts some music with the music id found in sounds.h.
//
void S_StartMusic(int m_id)
{
    S_ChangeMusic(m_id, false);
}

void
S_ChangeMusic
( int                musicnum,
  int                looping )
{
    musicinfo_t*      music;
    char              namebuf[9];

    if ( (musicnum <= mus_None)
        || (musicnum >= NUMMUSIC) )
    {
        I_Error("Bad music number %d", musicnum);
    }
    else
        music = &S_music[musicnum];

    if (mus_playing == music)
        return;

    // shutdown old music
    S_StopMusic();

    // get lumpnum if neccessary
    if (!music->lumpnum)
    {
        sprintf(namebuf, "d_%s", music->name);
        music->lumpnum = W_GetNumForName(namebuf);
    }

    // load & register it

```

```

music->data = (void *) W_CacheLumpNum(music->lumpnum, PU_MUSIC);
music->handle = I_RegisterSong(music->data);

// play it
I_PlaySong(music->handle, looping);

mus_playing = music;
}

void S_StopMusic(void)
{
    if (mus_playing)
    {
        if (mus_paused)
            I_ResumeSong(mus_playing->handle);

        I_StopSong(mus_playing->handle);
        I_UnRegisterSong(mus_playing->handle);
        Z_ChangeTag(mus_playing->data, PU_CACHE);

        mus_playing->data = 0;
        mus_playing = 0;
    }
}

void S_StopChannel(int cnum)
{
    int i;
    channel_t* c = &channels[cnum];

    if (c->sfxinfo)
    {
        // stop the sound playing
        if (I_SoundIsPlaying(c->handle))
        {
#ifdef SAWDEBBUG
            if (c->sfxinfo == &S_sfx[sfx_sawful])
                fprintf(stderr, "stopped\n");
#endif
            I_StopSound(c->handle);
        }

        // check to see
        // if other channels are playing the sound
        for (i=0 ; i<numChannels ; i++)
        {
            if (cnum != i
                && c->sfxinfo == channels[i].sfxinfo)
            {
                break;
            }
        }

        // degrade usefulness of sound data
        c->sfxinfo->usefulness--;

        c->sfxinfo = 0;
    }
}

```

```

//
// Changes volume, stereo-separation, and pitch variables
// from the norm of a sound effect to be played.
// If the sound is not audible, returns a 0.
// Otherwise, modifies parameters and returns 1.
//
int
S_AdjustSoundParams
( mobj_t*      listener,
  mobj_t*      source,
  int*         vol,
  int*         sep,
  int*         pitch )
{
    fixed_t     approx_dist;
    fixed_t     adx;
    fixed_t     ady;
    angle_t     angle;

    // calculate the distance to sound origin
    // and clip it if necessary
    adx = abs(listener->x - source->x);
    ady = abs(listener->y - source->y);

    // From _GG1_ p.428. Appox. euclidian distance fast.
    approx_dist = adx + ady - ((adx < ady ? adx : ady)>>1);

    if (gamemap != 8
        && approx_dist > S_CLIPPING_DIST)
    {
        return 0;
    }

    // angle of source to listener
    angle = R_PointToAngle2(listener->x,
                            listener->y,
                            source->x,
                            source->y);

    if (angle > listener->angle)
        angle = angle - listener->angle;
    else
        angle = angle + (0xffffffff - listener->angle);

    angle >>= ANGLETOFINESHIFT;

    // stereo separation
    *sep = 128 - (FixedMul(S_STEREO_SWING, finesine[angle])>>FRACBITS);

    // volume calculation
    if (approx_dist < S_CLOSE_DIST)
    {
        *vol = snd_SfxVolume;
    }
    else if (gamemap == 8)
    {
        if (approx_dist > S_CLIPPING_DIST)
            approx_dist = S_CLIPPING_DIST;

        *vol = 15+ ((snd_SfxVolume-15)
                    *((S_CLIPPING_DIST - approx_dist)>>FRACBITS))
            / S_ATTENUATOR;
    }
}

```

```

else
{
    // distance effect
    *vol = (snd_SfxVolume
            * ((S_CLIPING_DIST - approx_dist)>>FRACBITS))
        / S_ATTENUATOR;
}

return (*vol > 0);
}

//
// S_getChannel :
//   If none available, return -1. Otherwise channel #.
//
int
S_getChannel
( void*          origin,
  sfxinfo_t*     sfxinfo )
{
    // channel number to use
    int          cnum;

    channel_t*    c;

    // Find an open channel
    for (cnum=0 ; cnum<numChannels ; cnum++)
    {
        if (!channels[cnum].sfxinfo)
            break;
        else if (origin && channels[cnum].origin == origin)
        {
            S_StopChannel(cnum);
            break;
        }
    }

    // None available
    if (cnum == numChannels)
    {
        // Look for lower priority
        for (cnum=0 ; cnum<numChannels ; cnum++)
            if (channels[cnum].sfxinfo->priority >= sfxinfo->priority) break;

        if (cnum == numChannels)
        {
            // FUCK! No lower priority. Sorry, Charlie.
            return -1;
        }
        else
        {
            // Otherwise, kick out lower priority.
            S_StopChannel(cnum);
        }
    }

    c = &channels[cnum];

    // channel is decided to be cnum.
    c->sfxinfo = sfxinfo;
    c->origin = origin;

```

```

    return cnum;
}

```

## 11.2 s\_sound.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     The not so system specific sound interface.
//-----

#ifndef __S_SOUND__
#define __S_SOUND__

#ifdef __GNUG__
#pragma interface
#endif

//
// Initializes sound stuff, including volume
// Sets channels, SFX and music volume,
// allocates channel buffer, sets S_sfx lookup.
//
void
S_Init
( int          sfxVolume,
  int          musicVolume );

//
// Per level startup code.
// Kills playing sounds at start of level,
// determines music if any, changes music.
//
void S_Start(void);

//
// Start sound for thing at <origin>

```

```

// using <sound_id> from sounds.h
//
void
S_StartSound
( void*          origin,
  int            sound_id );

// Will start a sound at a given volume.
void
S_StartSoundAtVolume
( void*          origin,
  int            sound_id,
  int            volume );

// Stop sound for thing at <origin>
void S_StopSound(void* origin);

// Start music using <music_id> from sounds.h
void S_StartMusic(int music_id);

// Start music using <music_id> from sounds.h,
// and set whether looping
void
S_ChangeMusic
( int            music_id,
  int            looping );

// Stops the music fer sure.
void S_StopMusic(void);

// Stop and resume music, during game PAUSE.
void S_PauseSound(void);
void S_ResumeSound(void);

//
// Updates music & sounds
//
void S_UpdateSounds(void* listener);

void S_SetMusicVolume(int volume);
void S_SetSfxVolume(int volume);

#endif
//-----
//
// $Log:$
//
//-----

```

### 11.3 sounds.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or

```

```
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Created by a sound utility.
//     Kept as a sample, DOOM2 sounds.
//
//-----
```

```
static const char
rcsid[] = "$Id: sounds.c,v 1.3 1997/01/29 22:40:44 b1 Exp $";
```

```
#include "doomtype.h"
#include "sounds.h"
```

```
//
// Information about all the music
//
```

```
musicinfo_t S_music[] =
```

```
{
    { 0 },
    { "e1m1", 0 },
    { "e1m2", 0 },
    { "e1m3", 0 },
    { "e1m4", 0 },
    { "e1m5", 0 },
    { "e1m6", 0 },
    { "e1m7", 0 },
    { "e1m8", 0 },
    { "e1m9", 0 },
    { "e2m1", 0 },
    { "e2m2", 0 },
    { "e2m3", 0 },
    { "e2m4", 0 },
    { "e2m5", 0 },
    { "e2m6", 0 },
    { "e2m7", 0 },
    { "e2m8", 0 },
    { "e2m9", 0 },
    { "e3m1", 0 },
    { "e3m2", 0 },
    { "e3m3", 0 },
    { "e3m4", 0 },
    { "e3m5", 0 },
    { "e3m6", 0 },
    { "e3m7", 0 },
    { "e3m8", 0 },
    { "e3m9", 0 },
    { "inter", 0 },
    { "intro", 0 },
    { "bunny", 0 },
    { "victor", 0 },
    { "introa", 0 },
    { "runnin", 0 },
}
```



```

    { "stalks", 0 },
    { "countd", 0 },
    { "betwee", 0 },
    { "doom", 0 },
    { "the_da", 0 },
    { "shawn", 0 },
    { "ddtblu", 0 },
    { "in_cit", 0 },
    { "dead", 0 },
    { "stlks2", 0 },
    { "theda2", 0 },
    { "doom2", 0 },
    { "ddtbl2", 0 },
    { "runni2", 0 },
    { "dead2", 0 },
    { "stlks3", 0 },
    { "romero", 0 },
    { "shawn2", 0 },
    { "messag", 0 },
    { "count2", 0 },
    { "ddtbl3", 0 },
    { "ampie", 0 },
    { "theda3", 0 },
    { "adrian", 0 },
    { "messg2", 0 },
    { "romer2", 0 },
    { "tense", 0 },
    { "shawn3", 0 },
    { "openin", 0 },
    { "evil", 0 },
    { "ultima", 0 },
    { "read_m", 0 },
    { "dm2ttl", 0 },
    { "dm2int", 0 }
};

//
// Information about all the sfx
//

sfxinfo_t S_sfx[] =
{
    // S_sfx[0] needs to be a dummy for odd reasons.
    { "none", false, 0, 0, -1, -1, 0 },

    { "pistol", false, 64, 0, -1, -1, 0 },
    { "shotgn", false, 64, 0, -1, -1, 0 },
    { "sgcock", false, 64, 0, -1, -1, 0 },
    { "dshtgn", false, 64, 0, -1, -1, 0 },
    { "dbopn", false, 64, 0, -1, -1, 0 },
    { "dbcls", false, 64, 0, -1, -1, 0 },
    { "dbload", false, 64, 0, -1, -1, 0 },
    { "plasma", false, 64, 0, -1, -1, 0 },
    { "bfg", false, 64, 0, -1, -1, 0 },
    { "sawup", false, 64, 0, -1, -1, 0 },
    { "sawidl", false, 118, 0, -1, -1, 0 },
    { "sawful", false, 64, 0, -1, -1, 0 },
    { "sawhit", false, 64, 0, -1, -1, 0 },
    { "rlaunc", false, 64, 0, -1, -1, 0 },
    { "rxplod", false, 70, 0, -1, -1, 0 },
    { "firsht", false, 70, 0, -1, -1, 0 },
    { "firxpl", false, 70, 0, -1, -1, 0 },
    { "pstart", false, 100, 0, -1, -1, 0 },
    { "pstop", false, 100, 0, -1, -1, 0 },

```

```

{ "doropn", false, 100, 0, -1, -1, 0 },
{ "dorcls", false, 100, 0, -1, -1, 0 },
{ "stnmov", false, 119, 0, -1, -1, 0 },
{ "swtchn", false, 78, 0, -1, -1, 0 },
{ "swtchx", false, 78, 0, -1, -1, 0 },
{ "plpain", false, 96, 0, -1, -1, 0 },
{ "dmpain", false, 96, 0, -1, -1, 0 },
{ "popain", false, 96, 0, -1, -1, 0 },
{ "vipain", false, 96, 0, -1, -1, 0 },
{ "mnpain", false, 96, 0, -1, -1, 0 },
{ "pepain", false, 96, 0, -1, -1, 0 },
{ "slop", false, 78, 0, -1, -1, 0 },
{ "itemup", true, 78, 0, -1, -1, 0 },
{ "wpnup", true, 78, 0, -1, -1, 0 },
{ "oof", false, 96, 0, -1, -1, 0 },
{ "telept", false, 32, 0, -1, -1, 0 },
{ "posit1", true, 98, 0, -1, -1, 0 },
{ "posit2", true, 98, 0, -1, -1, 0 },
{ "posit3", true, 98, 0, -1, -1, 0 },
{ "bgsit1", true, 98, 0, -1, -1, 0 },
{ "bgsit2", true, 98, 0, -1, -1, 0 },
{ "sgtsit", true, 98, 0, -1, -1, 0 },
{ "cacsit", true, 98, 0, -1, -1, 0 },
{ "brssit", true, 94, 0, -1, -1, 0 },
{ "cybsit", true, 92, 0, -1, -1, 0 },
{ "spisit", true, 90, 0, -1, -1, 0 },
{ "bpsit", true, 90, 0, -1, -1, 0 },
{ "kntsit", true, 90, 0, -1, -1, 0 },
{ "vilsit", true, 90, 0, -1, -1, 0 },
{ "mansit", true, 90, 0, -1, -1, 0 },
{ "pesit", true, 90, 0, -1, -1, 0 },
{ "sklatk", false, 70, 0, -1, -1, 0 },
{ "sgtatk", false, 70, 0, -1, -1, 0 },
{ "skepch", false, 70, 0, -1, -1, 0 },
{ "vilatk", false, 70, 0, -1, -1, 0 },
{ "claw", false, 70, 0, -1, -1, 0 },
{ "skeswg", false, 70, 0, -1, -1, 0 },
{ "pldeth", false, 32, 0, -1, -1, 0 },
{ "pdiehi", false, 32, 0, -1, -1, 0 },
{ "podth1", false, 70, 0, -1, -1, 0 },
{ "podth2", false, 70, 0, -1, -1, 0 },
{ "podth3", false, 70, 0, -1, -1, 0 },
{ "bgdth1", false, 70, 0, -1, -1, 0 },
{ "bgdth2", false, 70, 0, -1, -1, 0 },
{ "sgtdth", false, 70, 0, -1, -1, 0 },
{ "cacdth", false, 70, 0, -1, -1, 0 },
{ "skldth", false, 70, 0, -1, -1, 0 },
{ "brsdth", false, 32, 0, -1, -1, 0 },
{ "cybdth", false, 32, 0, -1, -1, 0 },
{ "spidth", false, 32, 0, -1, -1, 0 },
{ "bspdth", false, 32, 0, -1, -1, 0 },
{ "vildth", false, 32, 0, -1, -1, 0 },
{ "kntdth", false, 32, 0, -1, -1, 0 },
{ "pedth", false, 32, 0, -1, -1, 0 },
{ "skedth", false, 32, 0, -1, -1, 0 },
{ "posact", true, 120, 0, -1, -1, 0 },
{ "bgact", true, 120, 0, -1, -1, 0 },
{ "dmact", true, 120, 0, -1, -1, 0 },
{ "bspact", true, 100, 0, -1, -1, 0 },
{ "bspwlk", true, 100, 0, -1, -1, 0 },
{ "vilact", true, 100, 0, -1, -1, 0 },
{ "noway", false, 78, 0, -1, -1, 0 },
{ "barexp", false, 60, 0, -1, -1, 0 },
{ "punch", false, 64, 0, -1, -1, 0 },
{ "hoof", false, 70, 0, -1, -1, 0 },

```

```

{ "metal", false, 70, 0, -1, -1, 0 },
{ "chgun", false, 64, &S_sfx[sfx_pistol], 150, 0, 0 },
{ "tink", false, 60, 0, -1, -1, 0 },
{ "bdopn", false, 100, 0, -1, -1, 0 },
{ "bdcls", false, 100, 0, -1, -1, 0 },
{ "itmbk", false, 100, 0, -1, -1, 0 },
{ "flame", false, 32, 0, -1, -1, 0 },
{ "flamst", false, 32, 0, -1, -1, 0 },
{ "getpow", false, 60, 0, -1, -1, 0 },
{ "bospit", false, 70, 0, -1, -1, 0 },
{ "boscut", false, 70, 0, -1, -1, 0 },
{ "bossit", false, 70, 0, -1, -1, 0 },
{ "bospn", false, 70, 0, -1, -1, 0 },
{ "bosdth", false, 70, 0, -1, -1, 0 },
{ "manatk", false, 70, 0, -1, -1, 0 },
{ "mandth", false, 70, 0, -1, -1, 0 },
{ "sssit", false, 70, 0, -1, -1, 0 },
{ "ssdth", false, 70, 0, -1, -1, 0 },
{ "keenpn", false, 70, 0, -1, -1, 0 },
{ "keendt", false, 70, 0, -1, -1, 0 },
{ "skeact", false, 70, 0, -1, -1, 0 },
{ "skesit", false, 70, 0, -1, -1, 0 },
{ "skeatk", false, 70, 0, -1, -1, 0 },
{ "radio", false, 60, 0, -1, -1, 0 }
};

```

## 11.4 sounds.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      Created by the sound utility written by Dave Taylor.
//      Kept as a sample, DOOM2 sounds. Frozen.
//-----

#ifndef __SOUNDS__
#define __SOUNDS__

//
// SoundFX struct.
//
typedef struct sfxinfo_struct      sfxinfo_t;

struct sfxinfo_struct
{
    // up to 6-character name
    char*      name;

```

```

// Sfx singularity (only one at a time)
int          singularity;

// Sfx priority
int          priority;

// referenced sound if a link
sfxinfo_t*   link;

// pitch if a link
int          pitch;

// volume if a link
int          volume;

// sound data
void*        data;

// this is checked every second to see if sound
// can be thrown out (if 0, then decrement, if -1,
// then throw out, if > 0, then it is in use)
int          usefulness;

// lump number of sfx
int          lumpnum;
};

```

```

//
// MusicInfo struct.
//
typedef struct
{
    // up to 6-character name
    char*      name;

    // lump number of music
    int        lumpnum;

    // music data
    void*      data;

    // music handle once registered
    int handle;
} musicinfo_t;

```

```

// the complete set of sound effects
extern sfxinfo_t    S_sfx[];

// the complete set of music
extern musicinfo_t   S_music[];

//
// Identifiers for all music in game.
//

typedef enum
{

```

mus\_None,  
mus\_e1m1,  
mus\_e1m2,  
mus\_e1m3,  
mus\_e1m4,  
mus\_e1m5,  
mus\_e1m6,  
mus\_e1m7,  
mus\_e1m8,  
mus\_e1m9,  
mus\_e2m1,  
mus\_e2m2,  
mus\_e2m3,  
mus\_e2m4,  
mus\_e2m5,  
mus\_e2m6,  
mus\_e2m7,  
mus\_e2m8,  
mus\_e2m9,  
mus\_e3m1,  
mus\_e3m2,  
mus\_e3m3,  
mus\_e3m4,  
mus\_e3m5,  
mus\_e3m6,  
mus\_e3m7,  
mus\_e3m8,  
mus\_e3m9,  
mus\_inter,  
mus\_intro,  
mus\_bunny,  
mus\_victor,  
mus\_introa,  
mus\_runnin,  
mus\_stalks,  
mus\_countd,  
mus\_betwee,  
mus\_doom,  
mus\_the\_da,  
mus\_shawn,  
mus\_ddtblu,  
mus\_in\_cit,  
mus\_dead,  
mus\_stlks2,  
mus\_theda2,  
mus\_doom2,  
mus\_ddtbl2,  
mus\_runni2,  
mus\_dead2,  
mus\_stlks3,  
mus\_romero,  
mus\_shawn2,  
mus\_messag,  
mus\_count2,  
mus\_ddtbl3,  
mus\_ampie,  
mus\_theda3,  
mus\_adrian,  
mus\_messg2,  
mus\_romer2,  
mus\_tense,  
mus\_shawn3,  
mus\_openin,  
mus\_evil,  
mus\_ultima,

```

    mus_read_m,
    mus_dm2ttl,
    mus_dm2int,
    NUMMUSIC
} musicenum_t;

//
// Identifiers for all sfx in game.
//

typedef enum
{
    sfx_None,
    sfx_pistol,
    sfx_shotgn,
    sfx_sgcock,
    sfx_dshtgn,
    sfx_dbopn,
    sfx_dbcls,
    sfx_dbload,
    sfx_plasma,
    sfx_bfg,
    sfx_sawup,
    sfx_sawidl,
    sfx_sawful,
    sfx_sawhit,
    sfx_rlaunc,
    sfx_rxplod,
    sfx_firsht,
    sfx_firxpl,
    sfx_pstart,
    sfx_pstop,
    sfx_doropn,
    sfx_dorcls,
    sfx_stnmov,
    sfx_swtnch,
    sfx_swtnchx,
    sfx_plpain,
    sfx_dmpain,
    sfx_popain,
    sfx_vipain,
    sfx_mnpain,
    sfx_pegain,
    sfx_slop,
    sfx_itemup,
    sfx_wpnup,
    sfx_oof,
    sfx_telept,
    sfx_posit1,
    sfx_posit2,
    sfx_posit3,
    sfx_bgsit1,
    sfx_bgsit2,
    sfx_sgtsit,
    sfx_cacsit,
    sfx_brssit,
    sfx_cybsit,
    sfx_spisit,
    sfx_bspsit,
    sfx_kntsit,
    sfx_vilsit,
    sfx_mansit,
    sfx_pesit,
    sfx_sklatk,

```

```

sfx_sgtatk,
sfx_skepch,
sfx_vilatk,
sfx_claw,
sfx_skeswg,
sfx_pldeth,
sfx_pdiehi,
sfx_podth1,
sfx_podth2,
sfx_podth3,
sfx_bgdth1,
sfx_bgdth2,
sfx_sgtdth,
sfx_cacdth,
sfx_skldth,
sfx_brsdth,
sfx_cybdth,
sfx_spidth,
sfx_bspdth,
sfx_vildth,
sfx_kntdth,
sfx_pedth,
sfx_skedth,
sfx_posact,
sfx_bgact,
sfx_dmact,
sfx_bspact,
sfx_bspwlk,
sfx_vilact,
sfx_noway,
sfx_barexp,
sfx_punch,
sfx_hoof,
sfx_metal,
sfx_chgun,
sfx_tink,
sfx_bdopn,
sfx_bdcls,
sfx_itmbk,
sfx_flame,
sfx_flamst,
sfx_getpow,
sfx_bospit,
sfx_boscub,
sfx_bossit,
sfx_bospn,
sfx_bosdth,
sfx_manatk,
sfx_mandth,
sfx_sssit,
sfx_ssdth,
sfx_keenpn,
sfx_keendt,
sfx_skeact,
sfx_skesit,
sfx_skeatk,
sfx_radio,
NUMSFX
} sfxenum_t;

#endif
//-----
//
// $Log:$
//

```

```
//-----
```

## 12 Status bar

### 12.1 st\_lib.c

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//     The status bar widget code.
//
//-----
```

```
static const char
rcsid[] = "$Id: st_lib.c,v 1.4 1997/02/03 16:47:56 b1 Exp $";
```

```
#include <ctype.h>
```

```
#include "doomdef.h"
```

```
#include "z_zone.h"
```

```
#include "v_video.h"
```

```
#include "m_swap.h"
```

```
#include "i_system.h"
```

```
#include "w_wad.h"
```

```
#include "st_stuff.h"
```

```
#include "st_lib.h"
```

```
#include "r_local.h"
```

```
// in AM_map.c
```

```
extern boolean          automapactive;
```

```
//
// Hack display negative frags.
// Loads and store the stminus lump.
//
patch_t*                sttminus;
```



```

void STlib_init(void)
{
    sttminus = (patch_t *) W_CacheLumpName("STTMINUS", PU_STATIC);
}

```

```

// ?
void
STlib_initNum
( st_number_t*      n,
  int               x,
  int               y,
  patch_t**         pl,
  int*              num,
  boolean*          on,
  int               width )
{
    n->x            = x;
    n->y            = y;
    n->oldnum        = 0;
    n->width         = width;
    n->num           = num;
    n->on            = on;
    n->p             = pl;
}

```

```

//
// A fairly efficient way to draw a number
// based on differences from the old number.
// Note: worth the trouble?
//
void
STlib_drawNum
( st_number_t*      n,
  boolean           refresh )
{
    int              numdigits = n->width;
    int              num = *n->num;

    int              w = SHORT(n->p[0]->width);
    int              h = SHORT(n->p[0]->height);
    int              x = n->x;

    int              neg;

    n->oldnum = *n->num;

    neg = num < 0;

    if (neg)
    {
        if (numdigits == 2 && num < -9)
            num = -9;
        else if (numdigits == 3 && num < -99)
            num = -99;

        num = -num;
    }

    // clear the area
    x = n->x - numdigits*w;

    if (n->y - ST_Y < 0)

```

```

        I_Error("drawNum: n->y - ST_Y < 0");

V_CopyRect(x, n->y - ST_Y, BG, w*numdigits, h, x, n->y, FG);

// if non-number, do not draw it
if (num == 1994)
    return;

x = n->x;

// in the special case of 0, you draw 0
if (!num)
    V_DrawPatch(x - w, n->y, FG, n->p[ 0 ]);

// draw the new number
while (num && numdigits--)
{
    x -= w;
    V_DrawPatch(x, n->y, FG, n->p[ num % 10 ]);
    num /= 10;
}

// draw a minus sign if necessary
if (neg)
    V_DrawPatch(x - 8, n->y, FG, sttminus);
}

//
void
STlib_updateNum
( st_number_t*      n,
  boolean           refresh )
{
    if (*n->on) STlib_drawNum(n, refresh);
}

//
void
STlib_initPercent
( st_percent_t*     p,
  int                x,
  int                y,
  patch_t**          pl,
  int*               num,
  boolean*           on,
  patch_t*           percent )
{
    STlib_initNum(&p->n, x, y, pl, num, on, 3);
    p->p = percent;
}

void
STlib_updatePercent
( st_percent_t*     per,
  int               refresh )
{
    if (refresh && *per->n.on)
        V_DrawPatch(per->n.x, per->n.y, FG, per->p);

    STlib_updateNum(&per->n, refresh);
}

```

```
}
```

```
void
STlib_initMultIcon
( st_multicon_t*      i,
  int                 x,
  int                 y,
  patch_t**          il,
  int*                inum,
  boolean*            on )
{
    i->x      = x;
    i->y      = y;
    i->oldinum = -1;
    i->inum    = inum;
    i->on      = on;
    i->p      = il;
}
```

```
void
STlib_updateMultIcon
( st_multicon_t*      mi,
  boolean              refresh )
{
    int                w;
    int                h;
    int                x;
    int                y;

    if (*mi->on
        && (mi->oldinum != *mi->inum || refresh)
        && (*mi->inum != -1))
    {
        if (mi->oldinum != -1)
        {
            x = mi->x - SHORT(mi->p[mi->oldinum]->leftoffset);
            y = mi->y - SHORT(mi->p[mi->oldinum]->topoffset);
            w = SHORT(mi->p[mi->oldinum]->width);
            h = SHORT(mi->p[mi->oldinum]->height);

            if (y - ST_Y < 0)
                I_Error("updateMultIcon: y - ST_Y < 0");

            V_CopyRect(x, y-ST_Y, BG, w, h, x, y, FG);
        }
        V_DrawPatch(mi->x, mi->y, FG, mi->p[*mi->inum]);
        mi->oldinum = *mi->inum;
    }
}
```

```
void
STlib_initBinIcon
( st_binicon_t*      b,
  int                 x,
  int                 y,
  patch_t*            i,
  boolean*            val,
  boolean*            on )
{
}
```

```

b->x      = x;
b->y      = y;
b->oldval = 0;
b->val     = val;
b->on      = on;
b->p      = i;
}

```

```

void
STlib_updateBinIcon
( st_binicon_t*      bi,
  boolean            refresh )
{
    int              x;
    int              y;
    int              w;
    int              h;

    if (*bi->on
        && (bi->oldval != *bi->val || refresh))
    {
        x = bi->x - SHORT(bi->p->leftoffset);
        y = bi->y - SHORT(bi->p->topoffset);
        w = SHORT(bi->p->width);
        h = SHORT(bi->p->height);

        if (y - ST_Y < 0)
            I_Error("updateBinIcon: y - ST_Y < 0");

        if (*bi->val)
            V_DrawPatch(bi->x, bi->y, FG, bi->p);
        else
            V_CopyRect(x, y-ST_Y, BG, w, h, x, y, FG);

        bi->oldval = *bi->val;
    }
}

```

## 12.2 st\_lib.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//     The status bar widget code.
//-----

```

```

#ifndef __STLIB__
#define __STLIB__

// We are referring to patches.
#include "r_defs.h"

//
// Background and foreground screen numbers
//
#define BG 4
#define FG 0

//
// Typedefs of widgets
//

// Number widget

typedef struct
{
    // upper right-hand corner
    // of the number (right-justified)
    int x;
    int y;

    // max # of digits in number
    int width;

    // last number value
    int oldnum;

    // pointer to current value
    int* num;

    // pointer to boolean stating
    // whether to update number
    boolean* on;

    // list of patches for 0-9
    patch_t** p;

    // user data
    int data;
} st_number_t;

// Percent widget ("child" of number widget,
// or, more precisely, contains a number widget.)
typedef struct
{
    // number information
    st_number_t n;

    // percent sign graphic
    patch_t* p;
} st_percent_t;

```

```

// Multiple Icon widget
typedef struct
{
    // center-justified location of icons
    int          x;
    int          y;

    // last icon number
    int          oldinum;

    // pointer to current icon
    int*         inum;

    // pointer to boolean stating
    // whether to update icon
    boolean*     on;

    // list of icons
    patch_t**    p;

    // user data
    int          data;
} st_multicon_t;


// Binary Icon widget

typedef struct
{
    // center-justified location of icon
    int          x;
    int          y;

    // last icon value
    int          oldval;

    // pointer to current icon status
    boolean*     val;

    // pointer to boolean
    // stating whether to update icon
    boolean*     on;

    patch_t*     p;          // icon
    int          data;       // user data
} st_binicon_t;


//
// Widget creation, access, and update routines
//

// Initializes widget library.
// More precisely, initialize STMINUS,
// everything else is done somewhere else.
//
void STlib_init(void);

```

```

// Number widget routines
void
STlib_initNum
( st_number_t*      n,
  int               x,
  int               y,
  patch_t**         pl,
  int*              num,
  boolean*          on,
  int               width );

void
STlib_updateNum
( st_number_t*      n,
  boolean           refresh );

// Percent widget routines
void
STlib_initPercent
( st_percent_t*     p,
  int               x,
  int               y,
  patch_t**         pl,
  int*              num,
  boolean*          on,
  patch_t*          percent );

void
STlib_updatePercent
( st_percent_t*     per,
  int               refresh );

// Multiple Icon widget routines
void
STlib_initMultIcon
( st_multicon_t*    mi,
  int               x,
  int               y,
  patch_t**         il,
  int*              inum,
  boolean*          on );

void
STlib_updateMultIcon
( st_multicon_t*    mi,
  boolean           refresh );

// Binary Icon widget routines
void
STlib_initBinIcon
( st_binicon_t*     b,
  int               x,
  int               y,
  patch_t*          i,
  boolean*          val,
  boolean*          on );

```

```

void
STlib_updateBinIcon
( st_binicon_t*      bi,
  boolean            refresh );

#endif
//-----
//
// $Log:$
//
//-----

```

### 12.3 st\_stuff.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Status bar code.
//      Does the face/direction indicator animatin.
//      Does palette indicators as well (red pain/berserk, bright pickup)
//
//-----

static const char
rcsid[] = "$Id: st_stuff.c,v 1.6 1997/02/03 22:45:13 b1 Exp $";

#include <stdio.h>

#include "i_system.h"
#include "i_video.h"
#include "z_zone.h"
#include "m_random.h"
#include "w_wad.h"

#include "doomdef.h"

#include "g_game.h"

#include "st_stuff.h"
#include "st_lib.h"
#include "r_local.h"

#include "p_local.h"
#include "p_inter.h"

#include "am_map.h"
#include "m_cheat.h"

```



```

#include "s_sound.h"

// Needs access to LFB.
#include "v_video.h"

// State.
#include "doomstat.h"

// Data.
#include "dstrings.h"
#include "sounds.h"

//
// STATUS BAR DATA
//

// Palette indices.
// For damage/bonus red-/gold-shifts
#define STARTREDPALS 1
#define STARTBONUSPALS 9
#define NUMREDPALS 8
#define NUMBONUSPALS 4
// Radiation suit, green shift.
#define RADIATIONPAL 13

// N/256*100% probability
// that the normal face state will change
#define ST_FACEPROBABILITY 96

// For Responder
#define ST_TOGGLECHAT KEY_ENTER

// Location of status bar
#define ST_X 0
#define ST_X2 104

#define ST_FX 143
#define ST_FY 169

// Should be set to patch width
// for tall numbers later on
#define ST_TALLNUMWIDTH (tallnum[0]->width)

// Number of status faces.
#define ST_NUMPAINFACES 5
#define ST_NUMSTRAIGHTFACES 3
#define ST_NUMTURNFACES 2
#define ST_NUMSPECIALFACES 3

#define ST_FACESTRIDE \
(ST_NUMSTRAIGHTFACES+ST_NUMTURNFACES+ST_NUMSPECIALFACES)

#define ST_NUMEXTRAFACES 2

#define ST_NUMFACES \
(ST_FACESTRIDE*ST_NUMPAINFACES+ST_NUMEXTRAFACES)

#define ST_TURNOFFSET (ST_NUMSTRAIGHTFACES)
#define ST_OUCHOFFSET (ST_TURNOFFSET + ST_NUMTURNFACES)
#define ST_EVILGRINOFFSET (ST_OUCHOFFSET + 1)
#define ST_RAMPAGEOFFSET (ST_EVILGRINOFFSET + 1)
#define ST_GODFACE (ST_NUMPAINFACES*ST_FACESTRIDE)
#define ST_DEADFACE (ST_GODFACE+1)

```

```

#define ST_FACESX                143
#define ST_FACESY                168

#define ST_EVILGRINCOUNT         (2*TICRATE)
#define ST_STRAIGHTFACECOUNT   (TICRATE/2)
#define ST_TURNCOUNT           (1*TICRATE)
#define ST_OUCHCOUNT           (1*TICRATE)
#define ST_RAMPAGEDELAY          (2*TICRATE)

#define ST_MUCHPAIN              20

// Location and size of statistics,
// justified according to widget type.
// Problem is, within which space? STbar? Screen?
// Note: this could be read in by a lump.
// Problem is, is the stuff rendered
// into a buffer,
// or into the frame buffer?

// AMMO number pos.
#define ST_AMMOWIDTH             3
#define ST_AMMOX                 44
#define ST_AMMOY                 171

// HEALTH number pos.
#define ST_HEALTHWIDTH          3
#define ST_HEALTHX              90
#define ST_HEALTHY              171

// Weapon pos.
#define ST_ARMSX                111
#define ST_ARMSY                172
#define ST_ARMSBGX              104
#define ST_ARMSBGY              168
#define ST_ARMSXSPACE           12
#define ST_ARMSYSPACE           10

// Frags pos.
#define ST_FRAGSX               138
#define ST_FRAGSY               171
#define ST_FRAGSWIDTH           2

// ARMOR number pos.
#define ST_ARMORWIDTH           3
#define ST_ARMORX               221
#define ST_ARMORY               171

// Key icon positions.
#define ST_KEYOWIDTH            8
#define ST_KEYOHEIGHT           5
#define ST_KEYOX                239
#define ST_KEYOY                171
#define ST_KEY1WIDTH            ST_KEYOWIDTH
#define ST_KEY1X                239
#define ST_KEY1Y                181
#define ST_KEY2WIDTH            ST_KEYOWIDTH
#define ST_KEY2X                239
#define ST_KEY2Y                191

// Ammunition counter.
#define ST_AMMOOWIDTH           3
#define ST_AMMOOHEIGHT          6
#define ST_AMMOOX               288

```

```

#define ST_AMMO0Y                173
#define ST_AMMO1WIDTH            ST_AMMO0WIDTH
#define ST_AMMO1X                288
#define ST_AMMO1Y                179
#define ST_AMMO2WIDTH            ST_AMMO0WIDTH
#define ST_AMMO2X                288
#define ST_AMMO2Y                191
#define ST_AMMO3WIDTH            ST_AMMO0WIDTH
#define ST_AMMO3X                288
#define ST_AMMO3Y                185

// Indicate maximum ammunition.
// Only needed because backpack exists.
#define ST_MAXAMMO0WIDTH        3
#define ST_MAXAMMO0HEIGHT       5
#define ST_MAXAMMO0X            314
#define ST_MAXAMMO0Y            173
#define ST_MAXAMMO1WIDTH        ST_MAXAMMO0WIDTH
#define ST_MAXAMMO1X            314
#define ST_MAXAMMO1Y            179
#define ST_MAXAMMO2WIDTH        ST_MAXAMMO0WIDTH
#define ST_MAXAMMO2X            314
#define ST_MAXAMMO2Y            191
#define ST_MAXAMMO3WIDTH        ST_MAXAMMO0WIDTH
#define ST_MAXAMMO3X            314
#define ST_MAXAMMO3Y            185

// pistol
#define ST_WEAPON0X              110
#define ST_WEAPON0Y              172

// shotgun
#define ST_WEAPON1X              122
#define ST_WEAPON1Y              172

// chain gun
#define ST_WEAPON2X              134
#define ST_WEAPON2Y              172

// missile launcher
#define ST_WEAPON3X              110
#define ST_WEAPON3Y              181

// plasma gun
#define ST_WEAPON4X              122
#define ST_WEAPON4Y              181

// bfg
#define ST_WEAPON5X              134
#define ST_WEAPON5Y              181

// WPNS title
#define ST_WPNSX                 109
#define ST_WPNSY                 191

// DETH title
#define ST_DETHX                 109
#define ST_DETHY                 191

//Incoming messages window location
//UNUSED
// #define ST_MSGTEXTX            (viewwindowx)
// #define ST_MSGTEXTY            (viewwindowy+viewheight-18)
#define ST_MSGTEXTX              0
#define ST_MSGTEXTY              0

```

```

// Dimensions given in characters.
#define ST_MSGWIDTH          52
// Or shall I say, in lines?
#define ST_MSGHEIGHT        1

#define ST_OUTTEXTX          0
#define ST_OUTTEXTY          6

// Width, in characters again.
#define ST_OUTWIDTH          52
// Height, in lines.
#define ST_OUTHEIGHT         1

#define ST_MAPWIDTH          \
    (strlen(mapnames[(gameepisode-1)*9+(gamemap-1)]))

#define ST_MAPTITLEX \
    (SCREENWIDTH - ST_MAPWIDTH * ST_CHATFONTWIDTH)

#define ST_MAPTITLEY          0
#define ST_MAPHEIGHT          1

// main player in game
static player_t*             plyr;

// ST_Start() has just been called
static boolean                st_firsttime;

// used to execute ST_Init() only once
static int                    veryfirsttime = 1;

// lump number for PLAYPAL
static int                    lu_palette;

// used for timing
static unsigned int           st_clock;

// used for making messages go away
static int                    st_msgcounter=0;

// used when in chat
static st_chatstateenum_t     st_chatstate;

// whether in automap or first-person
static st_stateenum_t         st_gamestate;

// whether left-side main status bar is active
static boolean                st_statusbaron;

// whether status bar chat is active
static boolean                st_chat;

// value of st_chat before message popped up
static boolean                st_oldchat;

// whether chat window has the cursor on
static boolean                st_cursoron;

// !deathmatch
static boolean                st_notdeathmatch;

// !deathmatch && st_statusbaron
static boolean                st_armson;

```

```

// !deathmatch
static boolean          st_fragson;

// main bar left
static patch_t*         sbar;

// 0-9, tall numbers
static patch_t*         tallnum[10];

// tall % sign
static patch_t*         tallpercent;

// 0-9, short, yellow (,different!) numbers
static patch_t*         shortnum[10];

// 3 key-cards, 3 skulls
static patch_t*         keys[NUMCARDS];

// face status patches
static patch_t*         faces[ST_NUMFACES];

// face background
static patch_t*         faceback;

// main bar right
static patch_t*         armsbg;

// weapon ownership patches
static patch_t*         arms[6][2];

// ready-weapon widget
static st_number_t      w_ready;

// in deathmatch only, summary of frags stats
static st_number_t      w_frags;

// health widget
static st_percent_t     w_health;

// arms background
static st_binicon_t     w_armsbg;

// weapon ownership widgets
static st_multicon_t    w_arms[6];

// face status widget
static st_multicon_t    w_faces;

// keycard widgets
static st_multicon_t    w_keyboxes[3];

// armor widget
static st_percent_t     w_armor;

// ammo widgets
static st_number_t      w_ammo[4];

// max ammo widgets
static st_number_t      w_maxammo[4];

// number of frags so far in deathmatch
static int              st_fragscout;

```

```

// used to use appropriately pained face
static int      st_oldhealth = -1;

// used for evil grin
static boolean   oldweaponsowned[NUMWEAPONS];

// count until face changes
static int      st_facecount = 0;

// current face index, used by w_faces
static int      st_faceindex = 0;

// holds key-type for each key box on bar
static int      keyboxes[3];

// a random number per tick
static int      st_randomnumber;

// Massive bunches of cheat shit
// to keep it from being easy to figure them out.
// Yeah, right...
unsigned char    cheat_mus_seq[] =
{
    0xb2, 0x26, 0xb6, 0xae, 0xea, 1, 0, 0, 0xff
};

unsigned char    cheat_choppers_seq[] =
{
    0xb2, 0x26, 0xe2, 0x32, 0xf6, 0x2a, 0x2a, 0xa6, 0x6a, 0xea, 0xff // id...
};

unsigned char    cheat_god_seq[] =
{
    0xb2, 0x26, 0x26, 0xaa, 0x26, 0xff // iddq
};

unsigned char    cheat_ammo_seq[] =
{
    0xb2, 0x26, 0xf2, 0x66, 0xa2, 0xff // idkfa
};

unsigned char    cheat_ammonokey_seq[] =
{
    0xb2, 0x26, 0x66, 0xa2, 0xff // idfa
};

// Smashing Pumpkins Into Samml Piles Of Putried Debris.
unsigned char    cheat_noclip_seq[] =
{
    0xb2, 0x26, 0xea, 0x2a, 0xb2, // idspispopd
    0xea, 0x2a, 0xf6, 0x2a, 0x26, 0xff
};

//
unsigned char    cheat_commercial_noclip_seq[] =
{
    0xb2, 0x26, 0xe2, 0x36, 0xb2, 0x2a, 0xff // idclip
};

```

```

unsigned char      cheat_powerup_seq[7][10] =
{
    { 0xb2, 0x26, 0x62, 0xa6, 0x32, 0xf6, 0x36, 0x26, 0x6e, 0xff },      // beholdv
    { 0xb2, 0x26, 0x62, 0xa6, 0x32, 0xf6, 0x36, 0x26, 0xea, 0xff },      // beholds
    { 0xb2, 0x26, 0x62, 0xa6, 0x32, 0xf6, 0x36, 0x26, 0xb2, 0xff },      // beholdi
    { 0xb2, 0x26, 0x62, 0xa6, 0x32, 0xf6, 0x36, 0x26, 0x6a, 0xff },      // beholddr
    { 0xb2, 0x26, 0x62, 0xa6, 0x32, 0xf6, 0x36, 0x26, 0xa2, 0xff },      // beholda
    { 0xb2, 0x26, 0x62, 0xa6, 0x32, 0xf6, 0x36, 0x26, 0x36, 0xff },      // beholdl
    { 0xb2, 0x26, 0x62, 0xa6, 0x32, 0xf6, 0x36, 0x26, 0xff }            // behold
};

unsigned char      cheat_clev_seq[] =
{
    0xb2, 0x26, 0xe2, 0x36, 0xa6, 0x6e, 1, 0, 0, 0xff      // idclev
};

// my position cheat
unsigned char      cheat_mypos_seq[] =
{
    0xb2, 0x26, 0xb6, 0xba, 0x2a, 0xf6, 0xea, 0xff      // idmypos
};

// Now what?
cheatseq_t        cheat_mus = { cheat_mus_seq, 0 };
cheatseq_t        cheat_god = { cheat_god_seq, 0 };
cheatseq_t        cheat_ammo = { cheat_ammo_seq, 0 };
cheatseq_t        cheat_ammonokey = { cheat_ammonokey_seq, 0 };
cheatseq_t        cheat_noclip = { cheat_noclip_seq, 0 };
cheatseq_t        cheat_commercial_noclip = { cheat_commercial_noclip_seq, 0 };

cheatseq_t        cheat_powerup[7] =
{
    { cheat_powerup_seq[0], 0 },
    { cheat_powerup_seq[1], 0 },
    { cheat_powerup_seq[2], 0 },
    { cheat_powerup_seq[3], 0 },
    { cheat_powerup_seq[4], 0 },
    { cheat_powerup_seq[5], 0 },
    { cheat_powerup_seq[6], 0 }
};

cheatseq_t        cheat_choppers = { cheat_choppers_seq, 0 };
cheatseq_t        cheat_clev = { cheat_clev_seq, 0 };
cheatseq_t        cheat_mypos = { cheat_mypos_seq, 0 };

//
extern char*      mapnames[];

//
// STATUS BAR CODE
//
void ST_Stop(void);

void ST_refreshBackground(void)
{
    if (st_statusbaron)
    {
        V_DrawPatch(ST_X, 0, BG, sbar);
    }
}

```

```

        if (netgame)
            V_DrawPatch(ST_FX, 0, BG, faceback);

        V_CopyRect(ST_X, 0, BG, ST_WIDTH, ST_HEIGHT, ST_X, ST_Y, FG);
    }

}

// Respond to keyboard input events,
// intercept cheats.
boolean
ST_Responder (event_t* ev)
{
    int            i;

    // Filter automap on/off.
    if (ev->type == ev_keyup
        && ((ev->data1 & 0xffff0000) == AM_MSGHEADER))
    {
        switch(ev->data1)
        {
            case AM_MSGENTERED:
                st_gamestate = AutomapState;
                st_firsttime = true;
                break;

            case AM_MSGEXITED:
                //      fprintf(stderr, "AM exited\n");
                st_gamestate = FirstPersonState;
                break;
        }
    }

    // if a user keypress...
    else if (ev->type == ev_keydown)
    {
        if (!netgame)
        {
            // b. - enabled for more debug fun.
            // if (gameskill != sk_nightmare) {

            // 'dqd' cheat for toggleable god mode
            if (cht_CheckCheat(&cheat_god, ev->data1))
            {
                plyr->cheats ^= CF_GODMODE;
                if (plyr->cheats & CF_GODMODE)
                {
                    if (plyr->mo)
                        plyr->mo->health = 100;

                    plyr->health = 100;
                    plyr->message = STSTR_DQDON;
                }
                else
                    plyr->message = STSTR_DQDOFF;
            }

            // 'fa' cheat for killer fucking arsenal
            else if (cht_CheckCheat(&cheat_ammonokey, ev->data1))
            {
                plyr->armorpoints = 200;
                plyr->armortype = 2;

                for (i=0; i<NUMWEAPONS; i++)
                    plyr->weaponowned[i] = true;
            }
        }
    }
}

```



```

    for (i=0;i<NUMAMMO;i++)
        plyr->ammo[i] = plyr->maxammo[i];

    plyr->message = STSTR_FAADDED;
}
// 'kfa' cheat for key full ammo
else if (cht_CheckCheat(&cheat_ammo, ev->data1))
{
    plyr->armorpoints = 200;
    plyr->armortype = 2;

    for (i=0;i<NUMWEAPONS;i++)
        plyr->weaponowned[i] = true;

    for (i=0;i<NUMAMMO;i++)
        plyr->ammo[i] = plyr->maxammo[i];

    for (i=0;i<NUMCARDS;i++)
        plyr->cards[i] = true;

    plyr->message = STSTR_KFAADDED;
}
// 'mus' cheat for changing music
else if (cht_CheckCheat(&cheat_mus, ev->data1))
{
    char        buf[3];
    int          musnum;

    plyr->message = STSTR_MUS;
    cht_GetParam(&cheat_mus, buf);

    if (gamemode == commercial)
    {
        musnum = mus_runnin + (buf[0]-'0')*10 + buf[1]-'0' - 1;

        if (((buf[0]-'0')*10 + buf[1]-'0') > 35)
            plyr->message = STSTR_NOMUS;
        else
            S_ChangeMusic(musnum, 1);
    }
    else
    {
        musnum = mus_e1m1 + (buf[0]-'1')*9 + (buf[1]-'1');

        if (((buf[0]-'1')*9 + buf[1]-'1') > 31)
            plyr->message = STSTR_NOMUS;
        else
            S_ChangeMusic(musnum, 1);
    }
}
// Simplified, accepting both "noclip" and "idspispopd".
// no clipping mode cheat
else if ( cht_CheckCheat(&cheat_noclip, ev->data1)
        || cht_CheckCheat(&cheat_commercial_noclip, ev->data1) )
{
    plyr->cheats ^= CF_NOCLIP;

    if (plyr->cheats & CF_NOCLIP)
        plyr->message = STSTR_NCON;
    else
        plyr->message = STSTR_NCOFF;
}
// 'behold?' power-up cheats

```

```

for (i=0;i<6;i++)
{
    if (cht_CheckCheat(&cheat_powerup[i], ev->data1))
    {
        if (!plyr->powers[i])
            P_GivePower( plyr, i);
        else if (i!=pw_strength)
            plyr->powers[i] = 1;
        else
            plyr->powers[i] = 0;

        plyr->message = STSTR_BEHOLDX;
    }
}

// 'behold' power-up menu
if (cht_CheckCheat(&cheat_powerup[6], ev->data1))
{
    plyr->message = STSTR_BEHOLD;
}

// 'choppers' invulnerability & chainsaw
else if (cht_CheckCheat(&cheat_choppers, ev->data1))
{
    plyr->weaponowned[wp_chainsaw] = true;
    plyr->powers[pw_invulnerability] = true;
    plyr->message = STSTR_CHOPPERS;
}

// 'mypos' for player position
else if (cht_CheckCheat(&cheat_mypos, ev->data1))
{
    static char          buf[ST_MSGWIDTH];
    sprintf(buf, "ang=0x%x;x,y=(0x%x,0x%x)",
            players[consoleplayer].mo->angle,
            players[consoleplayer].mo->x,
            players[consoleplayer].mo->y);
    plyr->message = buf;
}
}

// 'clev' change-level cheat
if (cht_CheckCheat(&cheat_clev, ev->data1))
{
    char          buf[3];
    int           epsd;
    int           map;

    cht_GetParam(&cheat_clev, buf);

    if (gamemode == commercial)
    {
        epsd = 0;
        map = (buf[0] - '0')*10 + buf[1] - '0';
    }
    else
    {
        epsd = buf[0] - '0';
        map = buf[1] - '0';
    }

    // Catch invalid maps.
    if (epsd < 1)
        return false;

    if (map < 1)
        return false;
}

```

```

// Ohmygod - this is not going to work.
if ((gamemode == retail)
    && ((epsd > 4) || (map > 9)))
    return false;

if ((gamemode == registered)
    && ((epsd > 3) || (map > 9)))
    return false;

if ((gamemode == shareware)
    && ((epsd > 1) || (map > 9)))
    return false;

if ((gamemode == commercial)
    && ((epsd > 1) || (map > 34)))
    return false;

// So be it.
plyr->message = STSTR_CLEV;
G_DeferedInitNew(gameskill, epsd, map);
}
}
return false;
}

int ST_calcPainOffset(void)
{
    int            health;
    static int     lastcalc;
    static int     oldhealth = -1;

    health = plyr->health > 100 ? 100 : plyr->health;

    if (health != oldhealth)
    {
        lastcalc = ST_FACESTRIDE * (((100 - health) * ST_NUMPAINFACES) / 101);
        oldhealth = health;
    }
    return lastcalc;
}

//
// This is a not-very-pretty routine which handles
// the face states and their timing.
// the precedence of expressions is:
// dead > evil grin > turned head > straight ahead
//
void ST_updateFaceWidget(void)
{
    int            i;
    angle_t        badguyangle;
    angle_t        diffang;
    static int     lastattackdown = -1;
    static int     priority = 0;
    boolean        doevilgrin;

    if (priority < 10)
    {
        // dead
        if (!plyr->health)
        {

```

```

        priority = 9;
        st_faceindex = ST_DEADFACE;
        st_facecount = 1;
    }
}

if (priority < 9)
{
    if (plyr->bonuscount)
    {
        // picking up bonus
        doevilgrin = false;

        for (i=0;i<NUMWEAPONS;i++)
        {
            if (oldweaponsowned[i] != plyr->weaponowned[i])
            {
                doevilgrin = true;
                oldweaponsowned[i] = plyr->weaponowned[i];
            }
        }
        if (doevilgrin)
        {
            // evil grin if just picked up weapon
            priority = 8;
            st_facecount = ST_EVILGRINCOUNT;
            st_faceindex = ST_calcPainOffset() + ST_EVILGRINOFFSET;
        }
    }
}

if (priority < 8)
{
    if (plyr->damagecount
        && plyr->attacker
        && plyr->attacker != plyr->mo)
    {
        // being attacked
        priority = 7;

        if (plyr->health - st_oldhealth > ST_MUCHPAIN)
        {
            st_facecount = ST_TURNCOUNT;
            st_faceindex = ST_calcPainOffset() + ST_OUCHOFFSET;
        }
        else
        {
            badguyangle = R_PointToAngle2(plyr->mo->x,
                                           plyr->mo->y,
                                           plyr->attacker->x,
                                           plyr->attacker->y);

            if (badguyangle > plyr->mo->angle)
            {
                // whether right or left
                diffang = badguyangle - plyr->mo->angle;
                i = diffang > ANG180;
            }
            else
            {
                // whether left or right
                diffang = plyr->mo->angle - badguyangle;
                i = diffang <= ANG180;
            } // confusing, aint it?
        }
    }
}

```

```

        st_facecount = ST_TURNCOUNT;
        st_faceindex = ST_calcPainOffset();

        if (diffang < ANG45)
        {
            // head-on
            st_faceindex += ST_RAMPAGEOFFSET;
        }
        else if (i)
        {
            // turn face right
            st_faceindex += ST_TURNOFFSET;
        }
        else
        {
            // turn face left
            st_faceindex += ST_TURNOFFSET+1;
        }
    }
}

if (priority < 7)
{
    // getting hurt because of your own damn stupidity
    if (plyr->damagecount)
    {
        if (plyr->health - st_oldhealth > ST_MUCHPAIN)
        {
            priority = 7;
            st_facecount = ST_TURNCOUNT;
            st_faceindex = ST_calcPainOffset() + ST_OUCHOFFSET;
        }
        else
        {
            priority = 6;
            st_facecount = ST_TURNCOUNT;
            st_faceindex = ST_calcPainOffset() + ST_RAMPAGEOFFSET;
        }
    }
}

if (priority < 6)
{
    // rapid firing
    if (plyr->attackdown)
    {
        if (lastattackdown== -1)
            lastattackdown = ST_RAMPAGEDELAY;
        else if (!--lastattackdown)
        {
            priority = 5;
            st_faceindex = ST_calcPainOffset() + ST_RAMPAGEOFFSET;
            st_facecount = 1;
            lastattackdown = 1;
        }
    }
    else
        lastattackdown = -1;
}

```

```

if (priority < 5)
{
    // invulnerability
    if ((plyr->cheats & CF_GODMODE)
        || plyr->powers[pw_invulnerability])
    {
        priority = 4;

        st_faceindex = ST_GODFACE;
        st_facecount = 1;

    }

}

// look left or look right if the facecount has timed out
if (!st_facecount)
{
    st_faceindex = ST_calcPainOffset() + (st_randomnumber % 3);
    st_facecount = ST_STRAIGHTFACECOUNT;
    priority = 0;
}

st_facecount--;

}

void ST_updateWidgets(void)
{
    static int      largeammo = 1994; // means "n/a"
    int            i;

    // must redirect the pointer if the ready weapon has changed.
    // if (w_ready.data != plyr->readyweapon)
    // {
    if (weaponinfo[plyr->readyweapon].ammo == am_noammo)
        w_ready.num = &largeammo;
    else
        w_ready.num = &plyr->ammo[weaponinfo[plyr->readyweapon].ammo];
    //{
    // static int tic=0;
    // static int dir=-1;
    // if (!(tic&15))
    //     plyr->ammo[weaponinfo[plyr->readyweapon].ammo] += dir;
    // if (plyr->ammo[weaponinfo[plyr->readyweapon].ammo] == -100)
    //     dir = 1;
    // tic++;
    // }
    w_ready.data = plyr->readyweapon;

    // if (*w_ready.on)
    //     STlib_updateNum(&w_ready, true);
    // refresh weapon change
    // }

    // update keycard multiple widgets
    for (i=0; i<3; i++)
    {
        keyboxes[i] = plyr->cards[i] ? i : -1;

        if (plyr->cards[i+3])
            keyboxes[i] = i+3;
    }
}

```

```

// refresh everything if this is him coming back to life
ST_updateFaceWidget();

// used by the w_armsbg widget
st_notdeathmatch = !deathmatch;

// used by w_arms[] widgets
st_armson = st_statusbaron && !deathmatch;

// used by w_frgs widget
st_fragson = deathmatch && st_statusbaron;
st_fragscout = 0;

for (i=0 ; i<MAXPLAYERS ; i++)
{
    if (i != consoleplayer)
        st_fragscout += plyr->frags[i];
    else
        st_fragscout -= plyr->frags[i];
}

// get rid of chat window if up because of message
if (!--st_msgcounter)
    st_chat = st_oldchat;
}

void ST_Ticker (void)
{
    st_clock++;
    st_randomnumber = M_Random();
    ST_updateWidgets();
    st_oldhealth = plyr->health;
}

static int st_palette = 0;

void ST_doPaletteStuff(void)
{
    int            palette;
    byte*          pal;
    int            cnt;
    int            bzc;

    cnt = plyr->damagecount;

    if (plyr->powers[pw_strength])
    {
        // slowly fade the berzerk out
        bzc = 12 - (plyr->powers[pw_strength]>>6);

        if (bzc > cnt)
            cnt = bzc;
    }

    if (cnt)
    {
        palette = (cnt+7)>>3;

        if (palette >= NUMREDPALS)
            palette = NUMREDPALS-1;
    }
}

```

```

        palette += STARTREDPALS;
    }

    else if (plyr->bonuscount)
    {
        palette = (plyr->bonuscount+7)>>3;

        if (palette >= NUMBONUSPALS)
            palette = NUMBONUSPALS-1;

        palette += STARTBONUSPALS;
    }

    else if ( plyr->powers[pw_ironfeet] > 4*32
              || plyr->powers[pw_ironfeet]&8)
        palette = RADIATIONPAL;
    else
        palette = 0;

    if (palette != st_palette)
    {
        st_palette = palette;
        pal = (byte *) W_CacheLumpNum (lu_palette, PU_CACHE)+palette*768;
        I_SetPalette (pal);
    }
}

void ST_drawWidgets(boolean refresh)
{
    int            i;

    // used by w_arms[] widgets
    st_armson = st_statusbaron && !deathmatch;

    // used by w_frgs widget
    st_fragson = deathmatch && st_statusbaron;

    STlib_updateNum(&w_ready, refresh);

    for (i=0;i<4;i++)
    {
        STlib_updateNum(&w_ammo[i], refresh);
        STlib_updateNum(&w_maxammo[i], refresh);
    }

    STlib_updatePercent(&w_health, refresh);
    STlib_updatePercent(&w_armor, refresh);

    STlib_updateBinIcon(&w_armsbg, refresh);

    for (i=0;i<6;i++)
        STlib_updateMultIcon(&w_arms[i], refresh);

    STlib_updateMultIcon(&w_faces, refresh);

    for (i=0;i<3;i++)
        STlib_updateMultIcon(&w_keyboxes[i], refresh);

    STlib_updateNum(&w_frgs, refresh);
}

void ST_doRefresh(void)
{

```



```

    st_firsttime = false;

    // draw status bar background to off-screen buff
    ST_refreshBackground();

    // and refresh all widgets
    ST_drawWidgets(true);
}

void ST_diffDraw(void)
{
    // update all widgets
    ST_drawWidgets(false);
}

void ST_Drawer (boolean fullscreen, boolean refresh)
{
    st_statusbaron = (!fullscreen) || automapactive;
    st_firsttime = st_firsttime || refresh;

    // Do red-/gold-shifts from damage/items
    ST_doPaletteStuff();

    // If just after ST_Start(), refresh all
    if (st_firsttime) ST_doRefresh();
    // Otherwise, update as little as possible
    else ST_diffDraw();
}

void ST_loadGraphics(void)
{
    int            i;
    int            j;
    int            facenum;

    char          namebuf[9];

    // Load the numbers, tall and short
    for (i=0;i<10;i++)
    {
        sprintf(namebuf, "STTNUM%d", i);
        tallnum[i] = (patch_t *) W_CacheLumpName(namebuf, PU_STATIC);

        sprintf(namebuf, "STYSNUM%d", i);
        shortnum[i] = (patch_t *) W_CacheLumpName(namebuf, PU_STATIC);
    }

    // Load percent key.
    //Note: why not load STMINUS here, too?
    tallpercent = (patch_t *) W_CacheLumpName("STTPRCNT", PU_STATIC);

    // key cards
    for (i=0;i<NUMCARDS;i++)
    {
        sprintf(namebuf, "STKEYS%d", i);
        keys[i] = (patch_t *) W_CacheLumpName(namebuf, PU_STATIC);
    }

    // arms background
    armsbg = (patch_t *) W_CacheLumpName("STARMS", PU_STATIC);
}

```

```

// arms ownership widgets
for (i=0;i<6;i++)
{
    sprintf(namebuf, "STGNUM%d", i+2);

    // gray #
    arms[i][0] = (patch_t *) W_CacheLumpName(namebuf, PU_STATIC);

    // yellow #
    arms[i][1] = shortnum[i+2];
}

// face backgrounds for different color players
sprintf(namebuf, "STFB%d", consoleplayer);
faceback = (patch_t *) W_CacheLumpName(namebuf, PU_STATIC);

// status bar background bits
sbar = (patch_t *) W_CacheLumpName("STBAR", PU_STATIC);

// face states
facenum = 0;
for (i=0;i<ST_NUMPAINFACES;i++)
{
    for (j=0;j<ST_NUMSTRAIGHTFACES;j++)
    {
        sprintf(namebuf, "STFST%d%d", i, j);
        faces[facenum++] = W_CacheLumpName(namebuf, PU_STATIC);
    }
    sprintf(namebuf, "STFTR%d0", i);          // turn right
    faces[facenum++] = W_CacheLumpName(namebuf, PU_STATIC);
    sprintf(namebuf, "STFTL%d0", i);          // turn left
    faces[facenum++] = W_CacheLumpName(namebuf, PU_STATIC);
    sprintf(namebuf, "STFOUCH%d", i);          // ouch!
    faces[facenum++] = W_CacheLumpName(namebuf, PU_STATIC);
    sprintf(namebuf, "STFEVL%d", i);           // evil grin ;)
    faces[facenum++] = W_CacheLumpName(namebuf, PU_STATIC);
    sprintf(namebuf, "STFKILL%d", i);          // pissed off
    faces[facenum++] = W_CacheLumpName(namebuf, PU_STATIC);
}
faces[facenum++] = W_CacheLumpName("STFGOD0", PU_STATIC);
faces[facenum++] = W_CacheLumpName("STFDEAD0", PU_STATIC);
}

void ST_loadData(void)
{
    lu_palette = W_GetNumForName ("PLAYPAL");
    ST_loadGraphics();
}

void ST_unloadGraphics(void)
{
    int i;

    // unload the numbers, tall and short
    for (i=0;i<10;i++)
    {
        Z_ChangeTag(tallnum[i], PU_CACHE);
        Z_ChangeTag(shortnum[i], PU_CACHE);
    }
    // unload tall percent
    Z_ChangeTag(tallpercent, PU_CACHE);
}

```

```

// unload arms background
Z_ChangeTag(armsbg, PU_CACHE);

// unload gray #'s
for (i=0;i<6;i++)
    Z_ChangeTag(arms[i][0], PU_CACHE);

// unload the key cards
for (i=0;i<NUMCARDS;i++)
    Z_ChangeTag(keys[i], PU_CACHE);

Z_ChangeTag(sbar, PU_CACHE);
Z_ChangeTag(faceback, PU_CACHE);

for (i=0;i<ST_NUMFACES;i++)
    Z_ChangeTag(faces[i], PU_CACHE);

// Note: nobody ain't seen no unloading
//   of stminus yet. Dude.

}

void ST_unloadData(void)
{
    ST_unloadGraphics();
}

void ST_initData(void)
{
    int            i;

    st_firsttime = true;
    plyr = &players[consoleplayer];

    st_clock = 0;
    st_chatstate = StartChatState;
    st_gamestate = FirstPersonState;

    st_statusbaron = true;
    st_oldchat = st_chat = false;
    st_cursoron = false;

    st_faceindex = 0;
    st_palette = -1;

    st_oldhealth = -1;

    for (i=0;i<NUMWEAPONS;i++)
        oldweaponsowned[i] = plyr->weaponowned[i];

    for (i=0;i<3;i++)
        keyboxes[i] = -1;

    STlib_init();
}

void ST_createWidgets(void)
{
    int i;

```

```

// ready weapon ammo
STlib_initNum(&w_ready,
              ST_AMMOX,
              ST_AMMOY,
              tallnum,
              &plyr->ammo[weaponinfo[plyr->readyweapon].ammo],
              &st_statusbaron,
              ST_AMMOWIDTH );

// the last weapon type
w_ready.data = plyr->readyweapon;

// health percentage
STlib_initPercent(&w_health,
                  ST_HEALTHX,
                  ST_HEALTHY,
                  tallnum,
                  &plyr->health,
                  &st_statusbaron,
                  tallpercent);

// arms background
STlib_initBinIcon(&w_armsbg,
                  ST_ARMSBGX,
                  ST_ARMSBGY,
                  armsbg,
                  &st_notdeathmatch,
                  &st_statusbaron);

// weapons owned
for(i=0;i<6;i++)
{
    STlib_initMultIcon(&w_arms[i],
                      ST_ARMSX+(i%3)*ST_ARMSXSPACE,
                      ST_ARMSY+(i/3)*ST_ARMSYSPACE,
                      arms[i], (int *) &plyr->weaponowned[i+1],
                      &st_armson);
}

// frags sum
STlib_initNum(&w_frags,
              ST_FRAGSX,
              ST_FRAGSY,
              tallnum,
              &st_fragscount,
              &st_fragson,
              ST_FRAGSWIDTH);

// faces
STlib_initMultIcon(&w_faces,
                  ST_FACESX,
                  ST_FACESY,
                  faces,
                  &st_faceindex,
                  &st_statusbaron);

// armor percentage - should be colored later
STlib_initPercent(&w_armor,
                  ST_ARMORX,
                  ST_ARMORY,
                  tallnum,
                  &plyr->armorpoints,
                  &st_statusbaron, tallpercent);

```

```

// keyboxes 0-2
STlib_initMultIcon(&w_keyboxes[0],
    ST_KEY0X,
    ST_KEY0Y,
    keys,
    &keyboxes[0],
    &st_statusbaron);

STlib_initMultIcon(&w_keyboxes[1],
    ST_KEY1X,
    ST_KEY1Y,
    keys,
    &keyboxes[1],
    &st_statusbaron);

STlib_initMultIcon(&w_keyboxes[2],
    ST_KEY2X,
    ST_KEY2Y,
    keys,
    &keyboxes[2],
    &st_statusbaron);

// ammo count (all four kinds)
STlib_initNum(&w_ammo[0],
    ST_AMMO0X,
    ST_AMMO0Y,
    shortnum,
    &plyr->ammo[0],
    &st_statusbaron,
    ST_AMMO0WIDTH);

STlib_initNum(&w_ammo[1],
    ST_AMMO1X,
    ST_AMMO1Y,
    shortnum,
    &plyr->ammo[1],
    &st_statusbaron,
    ST_AMMO1WIDTH);

STlib_initNum(&w_ammo[2],
    ST_AMMO2X,
    ST_AMMO2Y,
    shortnum,
    &plyr->ammo[2],
    &st_statusbaron,
    ST_AMMO2WIDTH);

STlib_initNum(&w_ammo[3],
    ST_AMMO3X,
    ST_AMMO3Y,
    shortnum,
    &plyr->ammo[3],
    &st_statusbaron,
    ST_AMMO3WIDTH);

// max ammo count (all four kinds)
STlib_initNum(&w_maxammo[0],
    ST_MAXAMMO0X,
    ST_MAXAMMO0Y,
    shortnum,
    &plyr->maxammo[0],
    &st_statusbaron,
    ST_MAXAMMO0WIDTH);

STlib_initNum(&w_maxammo[1],

```

```

        ST_MAXAMMO1X,
        ST_MAXAMMO1Y,
        shortnum,
        &plyr->maxammo[1],
        &st_statusbaron,
        ST_MAXAMMO1WIDTH);

STlib_initNum(&w_maxammo[2],
        ST_MAXAMMO2X,
        ST_MAXAMMO2Y,
        shortnum,
        &plyr->maxammo[2],
        &st_statusbaron,
        ST_MAXAMMO2WIDTH);

STlib_initNum(&w_maxammo[3],
        ST_MAXAMMO3X,
        ST_MAXAMMO3Y,
        shortnum,
        &plyr->maxammo[3],
        &st_statusbaron,
        ST_MAXAMMO3WIDTH);

}

static boolean      st_stopped = true;

void ST_Start (void)
{
    if (!st_stopped)
        ST_Stop();

    ST_initData();
    ST_createWidgets();
    st_stopped = false;
}

void ST_Stop (void)
{
    if (st_stopped)
        return;

    I_SetPalette (W_CacheLumpNum (lu_palette, PU_CACHE));

    st_stopped = true;
}

void ST_Init (void)
{
    veryfirsttime = 0;
    ST_loadData();
    screens[4] = (byte *) Z_Malloc(ST_WIDTH*ST_HEIGHT, PU_STATIC, 0);
}

```

## 12.4 st\_stuff.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.

```

```

//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// DESCRIPTION:
//      Status bar code.
//      Does the face/direction indicator animatin.
//      Does palette indicators as well (red pain/berserk, bright pickup)
//
//-----

#ifndef __STSTUFF_H__
#define __STSTUFF_H__

#include "doomtype.h"
#include "d_event.h"

// Size of statusbar.
// Now sensitive for scaling.
#define ST_HEIGHT      32*SCREEN_MUL
#define ST_WIDTH       SCREENWIDTH
#define ST_Y           (SCREENHEIGHT - ST_HEIGHT)

//
// STATUS BAR
//

// Called by main loop.
boolean ST_Responder (event_t* ev);

// Called by main loop.
void ST_Ticker (void);

// Called by main loop.
void ST_Drawer (boolean fullscreen, boolean refresh);

// Called when the console player is spawned on each level.
void ST_Start (void);

// Called by startup code.
void ST_Init (void);

// States for status bar code.
typedef enum
{
    AutomapState,
    FirstPersonState
} st_stateenum_t;

// States for the chat code.
typedef enum
{
    StartChatState,

```

```

    WaitDestState,
    GetChatState

} st_chatstateenum_t;

boolean ST_Responder(event_t* ev);

```

```

#endif
//-----
//
// $Log:$
//
//-----

```

## 13 General graphic drawing

### 13.1 v\_video.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This source is available for distribution and/or modification
// only under the terms of the DOOM Source Code License as
// published by id Software. All rights reserved.
//
// The source is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// FITNESS FOR A PARTICULAR PURPOSE. See the DOOM Source Code License
// for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Gamma correction LUT stuff.
//     Functions to draw patches (by post) directly to screen.
//     Functions to blit a block to the screen.
//
//-----

```

```

static const char
rcsid[] = "$Id: v_video.c,v 1.5 1997/02/03 22:45:13 b1 Exp $";

```

```

#include "i_system.h"
#include "r_local.h"

```

```

#include "doomdef.h"
#include "doomdata.h"

```

```

#include "m_bbox.h"
#include "m_swap.h"

```

```

#include "v_video.h"

```

```

// Each screen is [SCREENWIDTH*SCREENHEIGHT];
byte* screens[5];

```



```

int                                     dirtybox[4];

// Now where did these came from?
byte gammatable[5][256] =
{
    {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,
     17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,
     33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,
     49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,
     65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,
     81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,
     97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,
     113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,
     128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,
     144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,
     160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,
     176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,
     192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,
     208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,
     224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,
     240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255},

    {2,4,5,7,8,10,11,12,14,15,16,18,19,20,21,23,24,25,26,27,29,30,31,
     32,33,34,36,37,38,39,40,41,42,44,45,46,47,48,49,50,51,52,54,55,
     56,57,58,59,60,61,62,63,64,65,66,67,69,70,71,72,73,74,75,76,77,
     78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,
     99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,
     115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,129,
     130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,
     146,147,148,148,149,150,151,152,153,154,155,156,157,158,159,160,
     161,162,163,163,164,165,166,167,168,169,170,171,172,173,174,175,
     175,176,177,178,179,180,181,182,183,184,185,186,186,187,188,189,
     190,191,192,193,194,195,196,196,197,198,199,200,201,202,203,204,
     205,205,206,207,208,209,210,211,212,213,214,214,215,216,217,218,
     219,220,221,222,222,223,224,225,226,227,228,229,230,230,231,232,
     233,234,235,236,237,237,238,239,240,241,242,243,244,245,245,246,
     247,248,249,250,251,252,252,253,254,255},

    {4,7,9,11,13,15,17,19,21,22,24,26,27,29,30,32,33,35,36,38,39,40,42,
     43,45,46,47,48,50,51,52,54,55,56,57,59,60,61,62,63,65,66,67,68,69,
     70,72,73,74,75,76,77,78,79,80,82,83,84,85,86,87,88,89,90,91,92,93,
     94,95,96,97,98,100,101,102,103,104,105,106,107,108,109,110,111,112,
     113,114,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,
     129,130,131,132,133,133,134,135,136,137,138,139,140,141,142,143,144,
     144,145,146,147,148,149,150,151,152,153,153,154,155,156,157,158,159,
     160,160,161,162,163,164,165,166,166,167,168,169,170,171,172,172,173,
     174,175,176,177,178,178,179,180,181,182,183,183,184,185,186,187,188,
     188,189,190,191,192,193,193,194,195,196,197,197,198,199,200,201,201,
     202,203,204,205,206,206,207,208,209,210,210,211,212,213,213,214,215,
     216,217,217,218,219,220,221,221,222,223,224,224,225,226,227,228,228,
     229,230,231,231,232,233,234,235,235,236,237,238,238,239,240,241,241,
     242,243,244,244,245,246,247,247,248,249,250,251,251,252,253,254,254,
     255},

    {8,12,16,19,22,24,27,29,31,34,36,38,40,41,43,45,47,49,50,52,53,55,
     57,58,60,61,63,64,65,67,68,70,71,72,74,75,76,77,79,80,81,82,84,85,
     86,87,88,90,91,92,93,94,95,96,98,99,100,101,102,103,104,105,106,107,
     108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,
     125,126,127,128,129,130,131,132,133,134,135,135,136,137,138,139,140,
     141,142,143,143,144,145,146,147,148,149,150,150,151,152,153,154,155,
     155,156,157,158,159,160,160,161,162,163,164,165,165,166,167,168,169,
     169,170,171,172,173,173,174,175,176,176,177,178,179,180,180,181,182,

```

```

183,183,184,185,186,186,187,188,189,189,190,191,192,192,193,194,195,
195,196,197,197,198,199,200,200,201,202,202,203,204,205,205,206,207,
207,208,209,210,210,211,212,212,213,214,214,215,216,216,217,218,219,
219,220,221,221,222,223,223,224,225,225,226,227,227,228,229,229,230,
231,231,232,233,233,234,235,235,236,237,237,238,238,239,240,240,241,
242,242,243,244,244,245,246,246,247,247,248,249,249,250,251,251,252,
253,253,254,254,255},

{16,23,28,32,36,39,42,45,48,50,53,55,57,60,62,64,66,68,69,71,73,75,76,
78,80,81,83,84,86,87,89,90,92,93,94,96,97,98,100,101,102,103,105,106,
107,108,109,110,112,113,114,115,116,117,118,119,120,121,122,123,124,
125,126,128,128,129,130,131,132,133,134,135,136,137,138,139,140,141,
142,143,143,144,145,146,147,148,149,150,150,151,152,153,154,155,155,
156,157,158,159,159,160,161,162,163,163,164,165,166,166,167,168,169,
169,170,171,172,172,173,174,175,175,176,177,177,178,179,180,180,181,
182,182,183,184,184,185,186,187,187,188,189,189,190,191,191,192,193,
193,194,195,195,196,196,197,198,198,199,200,200,201,202,202,203,203,
204,205,205,206,207,207,208,208,209,210,210,211,211,212,213,213,214,
214,215,216,216,217,217,218,219,219,220,220,221,221,222,223,223,224,
224,225,225,226,227,227,228,228,229,229,230,230,231,232,232,233,233,
234,234,235,235,236,236,237,237,238,239,239,240,240,241,241,242,242,
243,243,244,244,245,245,246,246,247,247,248,248,249,249,250,250,251,
251,252,252,253,254,254,255,255}
};

```

```

int          usegamma;

//
// V_MarkRect
//
void
V_MarkRect
( int          x,
  int          y,
  int          width,
  int          height )
{
    M_AddToBox (dirtybox, x, y);
    M_AddToBox (dirtybox, x+width-1, y+height-1);
}

//
// V_CopyRect
//
void
V_CopyRect
( int          srcx,
  int          srcy,
  int          srcscrn,
  int          width,
  int          height,
  int          destx,
  int          desty,
  int          destscrn )
{
    byte*      src;
    byte*      dest;

#ifdef RANGECHECK
    if (srcx<0
        ||srcx+width >SCREENWIDTH
        || srcy<0

```

```

        || srcy+height>SCREENHEIGHT
        ||destx<0||destx+width >SCREENWIDTH
        || desty<0
        || desty+height>SCREENHEIGHT
        || (unsigned)srcscrn>4
        || (unsigned)destscrn>4)
    {
        I_Error ("Bad V_CopyRect");
    }
#endif
    V_MarkRect (destx, desty, width, height);

    src = screens[srcscrn]+SCREENWIDTH*srcy+srcx;
    dest = screens[destscrn]+SCREENWIDTH*desty+destx;

    for ( ; height>0 ; height--)
    {
        memcpy (dest, src, width);
        src += SCREENWIDTH;
        dest += SCREENWIDTH;
    }
}

//
// V_DrawPatch
// Masks a column based masked pic to the screen.
//
void
V_DrawPatch
( int          x,
  int          y,
  int          scrn,
  patch_t*     patch )
{
    int          count;
    int          col;
    column_t*    column;
    byte*        desttop;
    byte*        dest;
    byte*        source;
    int          w;

    y -= SHORT(patch->topoffset);
    x -= SHORT(patch->leftoffset);
#ifdef RANGECHECK
    if (x<0
        ||x+SHORT(patch->width) >SCREENWIDTH
        || y<0
        || y+SHORT(patch->height)>SCREENHEIGHT
        || (unsigned)scrn>4)
    {
        fprintf( stderr, "Patch at %d,%d exceeds LFB\n", x,y );
        // No I_Error abort - what is up with TNT.WAD?
        fprintf( stderr, "V_DrawPatch: bad patch (ignored)\n");
        return;
    }
#endif

    if (!scrn)
        V_MarkRect (x, y, SHORT(patch->width), SHORT(patch->height));

    col = 0;
    desttop = screens[scrn]+y*SCREENWIDTH+x;

```

```

w = SHORT(patch->width);

for ( ; col<w ; x++, col++, desttop++)
{
    column = (column_t *)((byte *)patch + LONG(patch->columnofs[col]));

    // step through the posts in a column
    while (column->topdelta != 0xff )
    {
        source = (byte *)column + 3;
        dest = desttop + column->topdelta*SCREENWIDTH;
        count = column->length;

        while (count--)
        {
            *dest = *source++;
            dest += SCREENWIDTH;
        }
        column = (column_t *)((byte *)column + column->length
                               + 4 );
    }
}

//
// V_DrawPatchFlipped
// Masks a column based masked pic to the screen.
// Flips horizontally, e.g. to mirror face.
//
void
V_DrawPatchFlipped
( int          x,
  int          y,
  int          scrn,
  patch_t*     patch )
{
    int          count;
    int          col;
    column_t*     column;
    byte*         desttop;
    byte*         dest;
    byte*         source;
    int          w;

    y -= SHORT(patch->topoffset);
    x -= SHORT(patch->leftoffset);
#ifdef RANGECHECK
    if (x<0
        || x+SHORT(patch->width) >SCREENWIDTH
        || y<0
        || y+SHORT(patch->height)>SCREENHEIGHT
        || (unsigned)scrn>4)
    {
        fprintf( stderr, "Patch origin %d,%d exceeds LFB\n", x,y );
        I_Error ("Bad V_DrawPatch in V_DrawPatchFlipped");
    }
#endif

    if (!scrn)
        V_MarkRect (x, y, SHORT(patch->width), SHORT(patch->height));

    col = 0;
    desttop = screens[scrn]+y*SCREENWIDTH+x;

```

```

w = SHORT(patch->width);

for ( ; col<w ; x++, col++, desttop++)
{
    column = (column_t *)((byte *)patch + LONG(patch->columnofs[w-1-col]));

    // step through the posts in a column
    while (column->topdelta != 0xff )
    {
        source = (byte *)column + 3;
        dest = desttop + column->topdelta*SCREENWIDTH;
        count = column->length;

        while (count--)
        {
            *dest = *source++;
            dest += SCREENWIDTH;
        }
        column = (column_t *)((byte *)column + column->length
                               + 4 );
    }
}
}

```

```

//
// V_DrawPatchDirect
// Draws directly to the screen on the pc.
//
void
V_DrawPatchDirect
( int          x,
  int          y,
  int          scrn,
  patch_t*     patch )
{
    V_DrawPatch (x,y,scrn, patch);

    /*
    int          count;
    int          col;
    column_t*    column;
    byte*        desttop;
    byte*        dest;
    byte*        source;
    int          w;

    y -= SHORT(patch->topoffset);
    x -= SHORT(patch->leftoffset);

#ifdef RANGECHECK
    if (x<0
        ||x+SHORT(patch->width) >SCREENWIDTH
        || y<0
        || y+SHORT(patch->height)>SCREENHEIGHT
        || (unsigned)scrn>4)
    {
        I_Error ("Bad V_DrawPatchDirect");
    }
#endif

    //          V_MarkRect (x, y, SHORT(patch->width), SHORT(patch->height));
    desttop = destscreen + y*SCREENWIDTH/4 + (x>>2);

```

```

w = SHORT(patch->width);
for ( col = 0 ; col<w ; col++)
{
    outp (SC_INDEX+1,1<<(x&3));
    column = (column_t *)((byte *)patch + LONG(patch->columnofs[col]));

    // step through the posts in a column

    while (column->topdelta != 0xff )
    {
        source = (byte *)column + 3;
        dest = desttop + column->topdelta*SCREENWIDTH/4;
        count = column->length;

        while (count--)
        {
            *dest = *source++;
            dest += SCREENWIDTH/4;
        }
        column = (column_t *)((byte *)column + column->length
                               + 4 );
    }
    if ( ((++x)&3) == 0 )
        desttop++;           // go to next byte, not next plane
}*/
}

```

```

//
// V_DrawBlock
// Draw a linear block of pixels into the view buffer.
//

```

```

void
V_DrawBlock
( int          x,
  int          y,
  int          scrn,
  int          width,
  int          height,
  byte*        src )
{
    byte*       dest;

#ifdef RANGECHECK
    if (x<0
        ||x+width >SCREENWIDTH
        || y<0
        || y+height>SCREENHEIGHT
        || (unsigned)scrn>4 )
    {
        I_Error ("Bad V_DrawBlock");
    }
#endif

```

```

    V_MarkRect (x, y, width, height);

    dest = screens[scrn] + y*SCREENWIDTH+x;

    while (height--)
    {
        memcpy (dest, src, width);
        src += width;
        dest += SCREENWIDTH;
    }

```

```

    }
}

//
// V_GetBlock
// Gets a linear block of pixels from the view buffer.
//
void
V_GetBlock
( int          x,
  int          y,
  int          scrn,
  int          width,
  int          height,
  byte*        dest )
{
    byte*      src;

#ifdef RANGECHECK
    if (x<0
        ||x+width >SCREENWIDTH
        || y<0
        || y+height>SCREENHEIGHT
        || (unsigned)scrn>4 )
    {
        I_Error ("Bad V_DrawBlock");
    }
#endif

    src = screens[scrn] + y*SCREENWIDTH+x;

    while (height--)
    {
        memcpy (dest, src, width);
        src += SCREENWIDTH;
        dest += width;
    }
}

//
// V_Init
//
void V_Init (void)
{
    int          i;
    byte*        base;

    // stick these in low dos memory on PCs

    base = I_AllocLow (SCREENWIDTH*SCREENHEIGHT*4);

    for (i=0 ; i<4 ; i++)
        screens[i] = base + i*SCREENWIDTH*SCREENHEIGHT;
}

```

## 13.2 v\_video.h

```

// Emacs style mode select  -*- C++ -*-
//-----
//

```

```

// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This source is available for distribution and/or modification
// only under the terms of the DOOM Source Code License as
// published by id Software. All rights reserved.
//
// The source is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// FITNESS FOR A PARTICULAR PURPOSE. See the DOOM Source Code License
// for more details.
//
// DESCRIPTION:
//     Gamma correction LUT.
//     Functions to draw patches (by post) directly to screen.
//     Functions to blit a block to the screen.
//
//-----

#ifdef __V_VIDEO__
#define __V_VIDEO__

#include "doomtype.h"

#include "doomdef.h"

// Needed because we are refering to patches.
#include "r_data.h"

//
// VIDEO
//

#define CENTERY                                (SCREENHEIGHT/2)

// Screen 0 is the screen updated by I_Update screen.
// Screen 1 is an extra buffer.

extern      byte*          screens[5];

extern int      dirtybox[4];

extern      byte          gammatable[5][256];
extern      int           usegamma;

// Allocates buffer screens, call before R_Init.
void V_Init (void);

void
V_CopyRect
( int          srcx,
  int          srcy,
  int          srcscrn,
  int          width,
  int          height,
  int          destx,
  int          desty,

```



```

    int                destscrn );

void
V_DrawPatch
( int                x,
  int                y,
  int                scrn,
  patch_t*          patch);

void
V_DrawPatchDirect
( int                x,
  int                y,
  int                scrn,
  patch_t*          patch );

// Draw a linear block of pixels into the view buffer.
void
V_DrawBlock
( int                x,
  int                y,
  int                scrn,
  int                width,
  int                height,
  byte*              src );

// Reads a linear block of pixels into the view buffer.
void
V_GetBlock
( int                x,
  int                y,
  int                scrn,
  int                width,
  int                height,
  byte*              dest );

void
V_MarkRect
( int                x,
  int                y,
  int                width,
  int                height );

#endif
//-----
//
// $Log:$
//
//-----

```

## 14 WAD file loading

### 14.1 w\_wad.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This source is available for distribution and/or modification
// only under the terms of the DOOM Source Code License as

```

```

// published by id Software. All rights reserved.
//
// The source is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// FITNESS FOR A PARTICULAR PURPOSE. See the DOOM Source Code License
// for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Handles WAD file header, directory, lump I/O.
//
//-----

static const char
rcsid[] = "$Id: w_wad.c,v 1.5 1997/02/03 16:47:57 b1 Exp $";

#ifdef NORMALUNIX
#include <ctype.h>
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <malloc.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <alloca.h>
#define O_BINARY      0
#endif

#include "doomtype.h"
#include "m_swap.h"
#include "i_system.h"
#include "z_zone.h"

#ifdef __GNUG__
#pragma implementation "w_wad.h"
#endif
#include "w_wad.h"

//
// GLOBALS
//

// Location of each lump on disk.
lumpinfo_t*      lumpinfo;
int              numlumps;

void**           lumpcache;

#define strcmpi    strcasecmp

void strupr (char* s)
{
    while (*s) { *s = toupper(*s); s++; }
}

int filelength (int handle)

```

```

{
    struct stat      fileinfo;

    if (fstat (handle,&fileinfo) == -1)
        I_Error ("Error fstating");

    return fileinfo.st_size;
}

void
ExtractFileBase
( char*      path,
  char*      dest )
{
    char*      src;
    int        length;

    src = path + strlen(path) - 1;

    // back up until a \ or the start
    while (src != path
           && *(src-1) != '\\'
           && *(src-1) != '/')
    {
        src--;
    }

    // copy up to eight characters
    memset (dest,0,8);
    length = 0;

    while (*src && *src != '.')
    {
        if (++length == 9)
            I_Error ("Filename base of %s >8 chars",path);

        *dest++ = toupper((int)*src++);
    }
}

//
// LUMP BASED ROUTINES.
//

//
// W_AddFile
// All files are optional, but at least one file must be
// found (PWAD, if all required lumps are present).
// Files with a .wad extension are wadlink files
// with multiple lumps.
// Other files are single lumps with the base filename
// for the lump name.
//
// If filename starts with a tilde, the file is handled
// specially to allow map reloads.
// But: the reload feature is a fragile hack...

int          reloadlump;
char*        reloadname;

```

```

void W_AddFile (char *filename)
{
    wadinfo_t          header;
    lumpinfo_t*        lump_p;
    unsigned           i;
    int                handle;
    int                length;
    int                startlump;
    filelump_t*        fileinfo;
    filelump_t         singleinfo;
    int                storehandle;

    // open the file and add to directory

    // handle reload indicator.
    if (filename[0] == '~')
    {
        filename++;
        reloadname = filename;
        reloadlump = numlumps;
    }

    if ( (handle = open (filename,O_RDONLY | O_BINARY)) == -1)
    {
        printf (" couldn't open %s\n",filename);
        return;
    }

    printf (" adding %s\n",filename);
    startlump = numlumps;

    if (strcmpi (filename+strlen(filename)-3 , "wad" ) )
    {
        // single lump file
        fileinfo = &singleinfo;
        singleinfo.filepos = 0;
        singleinfo.size = LONG(filelength(handle));
        ExtractFileBase (filename, singleinfo.name);
        numlumps++;
    }
    else
    {
        // WAD file
        read (handle, &header, sizeof(header));
        if (strcmp(header.identification,"IWAD",4))
        {
            // Homebrew levels?
            if (strcmp(header.identification,"PWAD",4))
            {
                I_Error ("Wad file %s doesn't have IWAD "
                        "or PWAD id\n", filename);
            }

            // ???modifiedgame = true;
        }
        header.numlumps = LONG(header.numlumps);
        header.infotableofs = LONG(header.infotableofs);
        length = header.numlumps*sizeof(filelump_t);
        fileinfo = alloca (length);
        lseek (handle, header.infotableofs, SEEK_SET);
        read (handle, fileinfo, length);
        numlumps += header.numlumps;
    }
}

```

```

// Fill in lumpinfo
lumpinfo = realloc (lumpinfo, numlumps*sizeof(lumpinfo_t));

if (!lumpinfo)
    I_Error ("Couldn't realloc lumpinfo");

lump_p = &lumpinfo[startlump];

storehandle = reloadname ? -1 : handle;

for (i=startlump ; i<numlumps ; i++,lump_p++, fileinfo++)
{
    lump_p->handle = storehandle;
    lump_p->position = LONG(fileinfo->filepos);
    lump_p->size = LONG(fileinfo->size);
    strncpy (lump_p->name, fileinfo->name, 8);
}

if (reloadname)
    close (handle);
}

```

```

//
// W_Reload
// Flushes any of the reloadable lumps in memory
// and reloads the directory.
//
void W_Reload (void)
{
    wadinfo_t          header;
    int                lumpcount;
    lumpinfo_t*        lump_p;
    unsigned           i;
    int                handle;
    int                length;
    filelump_t*        fileinfo;

    if (!reloadname)
        return;

    if ( (handle = open (reloadname,O_RDONLY | O_BINARY)) == -1)
        I_Error ("W_Reload: couldn't open %s",reloadname);

    read (handle, &header, sizeof(header));
    lumpcount = LONG(header.numlumps);
    header.infotableofs = LONG(header.infotableofs);
    length = lumpcount*sizeof(filelump_t);
    fileinfo = alloca (length);
    lseek (handle, header.infotableofs, SEEK_SET);
    read (handle, fileinfo, length);

    // Fill in lumpinfo
    lump_p = &lumpinfo[reloadlump];

    for (i=reloadlump ;
        i<reloadlump+lumpcount ;
        i++,lump_p++, fileinfo++)
    {
        if (lumpcache[i])
            Z_Free (lumpcache[i]);
    }
}

```

```

        lump_p->position = LONG(fileinfo->filepos);
        lump_p->size = LONG(fileinfo->size);
    }

    close (handle);
}

//
// W_InitMultipleFiles
// Pass a null terminated list of files to use.
// All files are optional, but at least one file
// must be found.
// Files with a .wad extension are idlink files
// with multiple lumps.
// Other files are single lumps with the base filename
// for the lump name.
// Lump names can appear multiple times.
// The name searcher looks backwards, so a later file
// does override all earlier ones.
//
void W_InitMultipleFiles (char** filenames)
{
    int                size;

    // open all the files, load headers, and count lumps
    numlumps = 0;

    // will be reallocated as lumps are added
    lumpinfo = malloc(1);

    for ( ; *filenames ; filenames++)
        W_AddFile (*filenames);

    if (!numlumps)
        I_Error ("W_InitFiles: no files found");

    // set up caching
    size = numlumps * sizeof(*lumpcache);
    lumpcache = malloc (size);

    if (!lumpcache)
        I_Error ("Couldn't allocate lumpcache");

    memset (lumpcache,0, size);
}

//
// W_InitFile
// Just initialize from a single file.
//
void W_InitFile (char* filename)
{
    char*              names[2];

    names[0] = filename;
    names[1] = NULL;
    W_InitMultipleFiles (names);
}

```

```

//
// W_NumLumps
//
int W_NumLumps (void)
{
    return numlumps;
}

//
// W_CheckNumForName
// Returns -1 if name not found.
//

int W_CheckNumForName (char* name)
{
    union {
        char      s[9];
        int       x[2];

    } name8;

    int          v1;
    int          v2;
    lumpinfo_t*  lump_p;

    // make the name into two integers for easy compares
    strncpy (name8.s,name,8);

    // in case the name was a fill 8 chars
    name8.s[8] = 0;

    // case insensitive
    strupr (name8.s);

    v1 = name8.x[0];
    v2 = name8.x[1];

    // scan backwards so patch lump files take precedence
    lump_p = lumpinfo + numlumps;

    while (lump_p-- != lumpinfo)
    {
        if ( *(int *)lump_p->name == v1
            && *(int *)&lump_p->name[4] == v2)
        {
            return lump_p - lumpinfo;
        }
    }

    // TFB. Not found.
    return -1;
}

//
// W_GetNumForName
// Calls W_CheckNumForName, but bombs out if not found.
//
int W_GetNumForName (char* name)

```

```

{
    int            i;

    i = W_CheckNumForName (name);

    if (i == -1)
        I_Error ("W_GetNumForName: %s not found!", name);

    return i;
}

//
// W_LumpLength
// Returns the buffer size needed to load the given lump.
//
int W_LumpLength (int lump)
{
    if (lump >= numlumps)
        I_Error ("W_LumpLength: %i >= numlumps",lump);

    return lumpinfo[lump].size;
}

//
// W_ReadLump
// Loads the lump into the given buffer,
// which must be >= W_LumpLength().
//
void
W_ReadLump
( int            lump,
  void*          dest )
{
    int           c;
    lumpinfo_t*   l;
    int           handle;

    if (lump >= numlumps)
        I_Error ("W_ReadLump: %i >= numlumps",lump);

    l = lumpinfo+lump;

    // ??? I_BeginRead ();

    if (l->handle == -1)
    {
        // reloadable file, so use open / read / close
        if ( (handle = open (reloadname,O_RDONLY | O_BINARY)) == -1)
            I_Error ("W_ReadLump: couldn't open %s",reloadname);
    }
    else
        handle = l->handle;

    lseek (handle, l->position, SEEK_SET);
    c = read (handle, dest, l->size);

    if (c < l->size)
        I_Error ("W_ReadLump: only read %i of %i on lump %i",
                  c,l->size,lump);

    if (l->handle == -1)
        close (handle);
}

```



```

    // ??? I_EndRead ();
}

//
// W_CacheLumpNum
//
void*
W_CacheLumpNum
( int          lump,
  int          tag )
{
    byte*      ptr;

    if ((unsigned)lump >= numlumps)
        I_Error ("W_CacheLumpNum: %i >= numlumps",lump);

    if (!lumpcache[lump])
    {
        // read the lump in

        //printf ("cache miss on lump %i\n",lump);
        ptr = Z_Malloc (W_LumpLength (lump), tag, &lumpcache[lump]);
        W_ReadLump (lump, lumpcache[lump]);
    }
    else
    {
        //printf ("cache hit on lump %i\n",lump);
        Z_ChangeTag (lumpcache[lump],tag);
    }

    return lumpcache[lump];
}

//
// W_CacheLumpName
//
void*
W_CacheLumpName
( char*        name,
  int          tag )
{
    return W_CacheLumpNum (W_GetNumForName(name), tag);
}

//
// W_Profile
//
int          info[2500][10];
int          profilecount;

void W_Profile (void)
{
    int          i;
    memblock_t*  block;
    void*        ptr;
    char         ch;
    FILE*        f;
    int          j;

```

```

char          name[9];

for (i=0 ; i<numlumps ; i++)
{
    ptr = lumpcache[i];
    if (!ptr)
    {
        ch = ' ';
        continue;
    }
    else
    {
        block = (memblock_t *) ( (byte *)ptr - sizeof(memblock_t));
        if (block->tag < PU_PURGELEVEL)
            ch = 'S';
        else
            ch = 'P';
    }
    info[i][profilecount] = ch;
}
profilecount++;

f = fopen ("waddump.txt","w");
name[8] = 0;

for (i=0 ; i<numlumps ; i++)
{
    memcpy (name,lumpinfo[i].name,8);

    for (j=0 ; j<8 ; j++)
        if (!name[j])
            break;

    for ( ; j<8 ; j++)
        name[j] = ' ';

    fprintf (f,"%s ",name);

    for (j=0 ; j<profilecount ; j++)
        fprintf (f,"    %c",info[i][j]);

    fprintf (f,"\n");
}
fclose (f);
}

```

## 14.2 w\_wad.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This source is available for distribution and/or modification
// only under the terms of the DOOM Source Code License as
// published by id Software. All rights reserved.
//
// The source is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// FITNESS FOR A PARTICULAR PURPOSE. See the DOOM Source Code License

```

```

// for more details.
//
// DESCRIPTION:
//      WAD I/O functions.
//
//-----

#ifndef __W_WAD__
#define __W_WAD__

#ifdef __GNUG__
#pragma interface
#endif

//
// TYPES
//
typedef struct
{
    // Should be "IWAD" or "PWAD".
    char            identification[4];
    int             numlumps;
    int             infotableofs;
} wadinfo_t;

typedef struct
{
    int             filepos;
    int             size;
    char            name[8];
} filelump_t;

//
// WADFILE I/O related stuff.
//
typedef struct
{
    char            name[8];
    int             handle;
    int             position;
    int             size;
} lumpinfo_t;

extern      void**      lumpcache;
extern      lumpinfo_t* lumpinfo;
extern      int         numlumps;

void  W_InitMultipleFiles (char** filenames);
void  W_Reload (void);

int    W_CheckNumForName (char* name);
int    W_GetNumForName (char* name);

int    W_LumpLength (int lump);
void   W_ReadLump (int lump, void *dest);

void*   W_CacheLumpNum (int lump, int tag);
void*   W_CacheLumpName (char* name, int tag);

```

```
#endif
//-----
//
// $Log:$
//
//-----
```

## 15 Intermission screen

### 15.1 wi\_stuff.c

```
// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This source is available for distribution and/or modification
// only under the terms of the DOOM Source Code License as
// published by id Software. All rights reserved.
//
// The source is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// FITNESS FOR A PARTICULAR PURPOSE. See the DOOM Source Code License
// for more details.
//
// $Log:$
//
// DESCRIPTION:
//      Intermission screens.
//
//-----
```

```
static const char
rcsid[] = "$Id: wi_stuff.c,v 1.7 1997/02/03 22:45:13 b1 Exp $";
```

```
#include <stdio.h>
```

```
#include "z_zone.h"
```

```
#include "m_random.h"
```

```
#include "m_swap.h"
```

```
#include "i_system.h"
```

```
#include "w_wad.h"
```

```
#include "g_game.h"
```

```
#include "r_local.h"
```

```
#include "s_sound.h"
```

```
#include "doomstat.h"
```

```
// Data.
```

```
#include "sounds.h"
```

```
// Needs access to LFB.
```

```
#include "v_video.h"
```

```

#include "wi_stuff.h"

//
// Data needed to add patches to full screen intermission pics.
// Patches are statistics messages, and animations.
// Loads of by-pixel layout and placement, offsets etc.
//

//
// Different between registered DOOM (1994) and
// Ultimate DOOM - Final edition (retail, 1995?).
// This is supposedly ignored for commercial
// release (aka DOOM II), which had 34 maps
// in one episode. So there.
#define NUMEPISODES      4
#define NUMMAPS          9

// in tics
//U #define PAUSELEN      (TICRATE*2)
//U #define SCORESTEP    100
//U #define ANIMPERIOD    32
// pixel distance from "(YOU)" to "PLAYER N"
//U #define STARDIST      10
//U #define WK 1

// GLOBAL LOCATIONS
#define WI_TITLEY        2
#define WI_SPACINGY      33

// SINGLE-PLAYER STUFF
#define SP_STATSX        50
#define SP_STATSY        50

#define SP_TIMEX         16
#define SP_TIMEY         (SCREENHEIGHT-32)

// NET GAME STUFF
#define NG_STATSY        50
#define NG_STATSX        (32 + SHORT(star->width)/2 + 32*!dofrags)

#define NG_SPACINGX      64

// DEATHMATCH STUFF
#define DM_MATRIXX       42
#define DM_MATRIXY       68

#define DM_SPACINGX      40

#define DM_TOTALSX       269

#define DM_KILLERSX      10
#define DM_KILLERSY      100
#define DM_VICTIMSX      5
#define DM_VICTIMSY      50

typedef enum
{

```

```

    ANIM_ALWAYS,
    ANIM_RANDOM,
    ANIM_LEVEL
} animenum_t;

typedef struct
{
    int            x;
    int            y;
} point_t;

//
// Animation.
// There is another anim_t used in p_spec.
//
typedef struct
{
    animenum_t      type;

    // period in tics between animations
    int             period;

    // number of animation frames
    int             nanims;

    // location of animation
    point_t         loc;

    // ALWAYS: n/a,
    // RANDOM: period deviation (<256),
    // LEVEL: level
    int             data1;

    // ALWAYS: n/a,
    // RANDOM: random base period,
    // LEVEL: n/a
    int             data2;

    // actual graphics for frames of animations
    patch_t*        p[3];

    // following must be initialized to zero before use!

    // next value of bcnt (used in conjunction with period)
    int             nexttic;

    // last drawn animation frame
    int             lastdrawn;

    // next frame number to animate
    int             ctr;

    // used by RANDOM and LEVEL when animating
    int             state;
} anim_t;

static point_t lnodes[NUMEPISODES][NUMMAPS] =
{
    // Episode 0 World Map
    {

```

```

    { 185, 164 },          // location of level 0 (CJ)
    { 148, 143 },          // location of level 1 (CJ)
    { 69, 122 },           // location of level 2 (CJ)
    { 209, 102 },          // location of level 3 (CJ)
    { 116, 89 },           // location of level 4 (CJ)
    { 166, 55 },           // location of level 5 (CJ)
    { 71, 56 },            // location of level 6 (CJ)
    { 135, 29 },           // location of level 7 (CJ)
    { 71, 24 },            // location of level 8 (CJ)
},

// Episode 1 World Map should go here
{
    { 254, 25 },           // location of level 0 (CJ)
    { 97, 50 },            // location of level 1 (CJ)
    { 188, 64 },           // location of level 2 (CJ)
    { 128, 78 },           // location of level 3 (CJ)
    { 214, 92 },           // location of level 4 (CJ)
    { 133, 130 },          // location of level 5 (CJ)
    { 208, 136 },          // location of level 6 (CJ)
    { 148, 140 },          // location of level 7 (CJ)
    { 235, 158 },          // location of level 8 (CJ)
},

// Episode 2 World Map should go here
{
    { 156, 168 },          // location of level 0 (CJ)
    { 48, 154 },           // location of level 1 (CJ)
    { 174, 95 },           // location of level 2 (CJ)
    { 265, 75 },           // location of level 3 (CJ)
    { 130, 48 },           // location of level 4 (CJ)
    { 279, 23 },           // location of level 5 (CJ)
    { 198, 48 },           // location of level 6 (CJ)
    { 140, 25 },           // location of level 7 (CJ)
    { 281, 136 },          // location of level 8 (CJ)
}

};

//
// Animation locations for episode 0 (1).
// Using patches saves a lot of space,
// as they replace 320x200 full screen frames.
//
static anim_t epsd0animinfo[] =
{
    { ANIM_ALWAYS, TICRATE/3, 3, { 224, 104 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 184, 160 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 112, 136 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 72, 112 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 88, 96 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 64, 48 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 192, 40 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 136, 16 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 80, 16 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 64, 24 } }
};

static anim_t epsd1animinfo[] =
{
    { ANIM_LEVEL, TICRATE/3, 1, { 128, 136 }, 1 },
    { ANIM_LEVEL, TICRATE/3, 1, { 128, 136 }, 2 },
    { ANIM_LEVEL, TICRATE/3, 1, { 128, 136 }, 3 },
    { ANIM_LEVEL, TICRATE/3, 1, { 128, 136 }, 4 },

```

```

    { ANIM_LEVEL, TICRATE/3, 1, { 128, 136 }, 5 },
    { ANIM_LEVEL, TICRATE/3, 1, { 128, 136 }, 6 },
    { ANIM_LEVEL, TICRATE/3, 1, { 128, 136 }, 7 },
    { ANIM_LEVEL, TICRATE/3, 3, { 192, 144 }, 8 },
    { ANIM_LEVEL, TICRATE/3, 1, { 128, 136 }, 8 }
};

static anim_t epsd2animinfo[] =
{
    { ANIM_ALWAYS, TICRATE/3, 3, { 104, 168 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 40, 136 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 160, 96 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 104, 80 } },
    { ANIM_ALWAYS, TICRATE/3, 3, { 120, 32 } },
    { ANIM_ALWAYS, TICRATE/4, 3, { 40, 0 } }
};

static int NUMANIMS[NUMEPISODES] =
{
    sizeof(epsd0animinfo)/sizeof(anim_t),
    sizeof(epsd1animinfo)/sizeof(anim_t),
    sizeof(epsd2animinfo)/sizeof(anim_t)
};

static anim_t *anims[NUMEPISODES] =
{
    epsd0animinfo,
    epsd1animinfo,
    epsd2animinfo
};

//
// GENERAL DATA
//

//
// Locally used stuff.
//
#define FB 0

// States for single-player
#define SP_KILLS 0
#define SP_ITEMS 2
#define SP_SECRET 4
#define SP_FRAGS 6
#define SP_TIME 8
#define SP_PAR ST_TIME

#define SP_PAUSE 1

// in seconds
#define SHOWNEXTLOCDELAY 4
// #define SHOWLASTLOCDELAY SHOWNEXTLOCDELAY

// used to accelerate or skip a stage
static int acceleratestage;

// wbs->pnum
static int me;

// specifies current state
static stateenum_t state;

```



```

// contains information passed into intermission
static wbstartstruct_t*      wbs;

static wbplayerstruct_t* plrs; // wbs->plyr[]

// used for general timing
static int                  cnt;

// used for timing of background animation
static int                  bcnt;

// signals to refresh everything for one frame
static int                  firstrefresh;

static int                  cnt_kills[MAXPLAYERS];
static int                  cnt_items[MAXPLAYERS];
static int                  cnt_secret[MAXPLAYERS];
static int                  cnt_time;
static int                  cnt_par;
static int                  cnt_pause;

// # of commercial levels
static int                  NUMCMAPS;

//
//      GRAPHICS
//

// background (map of levels).
static patch_t*             bg;

// You Are Here graphic
static patch_t*             yah[2];

// splat
static patch_t*             splat;

// %, : graphics
static patch_t*             percent;
static patch_t*             colon;

// 0-9 graphic
static patch_t*             num[10];

// minus sign
static patch_t*             wiminus;

// "Finished!" graphics
static patch_t*             finished;

// "Entering" graphic
static patch_t*             entering;

// "secret"
static patch_t*             sp_secret;

// "Kills", "Scrt", "Items", "Fragments"
static patch_t*             kills;
static patch_t*             secret;
static patch_t*             items;
static patch_t*             frags;

// Time sucks.

```

```

static patch_t*      time;
static patch_t*      par;
static patch_t*      sucks;

// "killers", "victims"
static patch_t*      killers;
static patch_t*      victims;

// "Total", your face, your dead face
static patch_t*      total;
static patch_t*      star;
static patch_t*      bstar;

// "red P[1..MAXPLAYERS]"
static patch_t*      p[MAXPLAYERS];

// "gray P[1..MAXPLAYERS]"
static patch_t*      bp[MAXPLAYERS];

// Name graphics of each level (centered)
static patch_t**      lnames;

//
// CODE
//

// slam background
// UNUSED static unsigned char *background=0;

void WI_slamBackground(void)
{
    memcpy(screens[0], screens[1], SCREENWIDTH * SCREENHEIGHT);
    V_MarkRect (0, 0, SCREENWIDTH, SCREENHEIGHT);
}

// The ticker is used to detect keys
// because of timing issues in netgames.
boolean WI_Responder(event_t* ev)
{
    return false;
}

// Draws "<Levelname> Finished!"
void WI_drawLF(void)
{
    int y = WI_TITLEY;

    // draw <LevelName>
    V_DrawPatch((SCREENWIDTH - SHORT(lnames[wbs->last]->width))/2,
                y, FB, lnames[wbs->last]);

    // draw "Finished!"
    y += (5*SHORT(lnames[wbs->last]->height))/4;

    V_DrawPatch((SCREENWIDTH - SHORT(finished->width))/2,
                y, FB, finished);
}

// Draws "Entering <LevelName>"
void WI_drawEL(void)
{

```

```

int y = WI_TITLEY;

// draw "Entering"
V_DrawPatch((SCREENWIDTH - SHORT(entering->width))/2,
            y, FB, entering);

// draw level
y += (5*SHORT(lnames[wbs->next]->height))/4;

V_DrawPatch((SCREENWIDTH - SHORT(lnames[wbs->next]->width))/2,
            y, FB, lnames[wbs->next]);

}

void
WI_drawOnLnode
( int          n,
  patch_t*     c[] )
{
    int          i;
    int          left;
    int          top;
    int          right;
    int          bottom;
    boolean      fits = false;

    i = 0;
    do
    {
        left = lnodes[wbs->epsd][n].x - SHORT(c[i]->leftoffset);
        top = lnodes[wbs->epsd][n].y - SHORT(c[i]->topoffset);
        right = left + SHORT(c[i]->width);
        bottom = top + SHORT(c[i]->height);

        if (left >= 0
            && right < SCREENWIDTH
            && top >= 0
            && bottom < SCREENHEIGHT)
        {
            fits = true;
        }
        else
        {
            i++;
        }
    } while (!fits && i!=2);

    if (fits && i<2)
    {
        V_DrawPatch(lnodes[wbs->epsd][n].x, lnodes[wbs->epsd][n].y,
                    FB, c[i]);
    }
    else
    {
        // DEBUG
        printf("Could not place patch on level %d", n+1);
    }
}

void WI_initAnimatedBack(void)
{
    int          i;

```

```

anim_t*      a;

if (gamemode == commercial)
    return;

if (wbs->epsd > 2)
    return;

for (i=0;i<NUMANIMS[wbs->epsd];i++)
{
    a = &anim[wbs->epsd][i];

    // init variables
    a->ctr = -1;

    // specify the next time to draw it
    if (a->type == ANIM_ALWAYS)
        a->nexttic = bcnt + 1 + (M_Random()%a->period);
    else if (a->type == ANIM_RANDOM)
        a->nexttic = bcnt + 1 + a->data2+(M_Random()%a->data1);
    else if (a->type == ANIM_LEVEL)
        a->nexttic = bcnt + 1;
}
}

void WI_updateAnimatedBack(void)
{
    int          i;
    anim_t*      a;

    if (gamemode == commercial)
        return;

    if (wbs->epsd > 2)
        return;

    for (i=0;i<NUMANIMS[wbs->epsd];i++)
    {
        a = &anim[wbs->epsd][i];

        if (bcnt == a->nexttic)
        {
            switch (a->type)
            {
                case ANIM_ALWAYS:
                    if (++a->ctr >= a->nanim) a->ctr = 0;
                    a->nexttic = bcnt + a->period;
                    break;

                case ANIM_RANDOM:
                    a->ctr++;
                    if (a->ctr == a->nanim)
                    {
                        a->ctr = -1;
                        a->nexttic = bcnt+a->data2+(M_Random()%a->data1);
                    }
                    else a->nexttic = bcnt + a->period;
                    break;

                case ANIM_LEVEL:
                    // gawd-awful hack for level anims
                    if (!(state == StatCount && i == 7)
                        && wbs->next == a->data1)
                    {

```

```

        a->ctr++;
        if (a->ctr == a->nanims) a->ctr--;
        a->nexttic = bcnt + a->period;
    }
    break;
}
}

}

}

void WI_drawAnimatedBack(void)
{
    int                i;
    anim_t*            a;

    if (commercial)
        return;

    if (wbs->epsd > 2)
        return;

    for (i=0 ; i<NUMANIMS[wbs->epsd] ; i++)
    {
        a = &anim[wbs->epsd][i];

        if (a->ctr >= 0)
            V_DrawPatch(a->loc.x, a->loc.y, FB, a->p[a->ctr]);
    }
}

//
// Draws a number.
// If digits > 0, then use that many digits minimum,
// otherwise only use as many as necessary.
// Returns new x position.
//

int
WI_drawNum
( int        x,
  int        y,
  int        n,
  int        digits )
{
    int        fontwidth = SHORT(num[0]->width);
    int        neg;
    int        temp;

    if (digits < 0)
    {
        if (!n)
        {
            // make variable-length zeros 1 digit long
            digits = 1;
        }
        else
        {
            // figure out # of digits in #
            digits = 0;
            temp = n;

```

```

        while (temp)
        {
            temp /= 10;
            digits++;
        }
    }

    neg = n < 0;
    if (neg)
        n = -n;

    // if non-number, do not draw it
    if (n == 1994)
        return 0;

    // draw the new number
    while (digits--)
    {
        x -= fontwidth;
        V_DrawPatch(x, y, FB, num[ n % 10 ]);
        n /= 10;
    }

    // draw a minus sign if necessary
    if (neg)
        V_DrawPatch(x-=8, y, FB, wiminus);

    return x;
}

void
WI_drawPercent
( int          x,
  int          y,
  int          p )
{
    if (p < 0)
        return;

    V_DrawPatch(x, y, FB, percent);
    WI_drawNum(x, y, p, -1);
}

//
// Display level completion time and par,
// or "sucks" message if overflow.
//
void
WI_drawTime
( int          x,
  int          y,
  int          t )
{
    int          div;
    int          n;

    if (t<0)
        return;

    if (t <= 61*59)

```

```

{
    div = 1;

    do
    {
        n = (t / div) % 60;
        x = WI_drawNum(x, y, n, 2) - SHORT(colon->width);
        div *= 60;

        // draw
        if (div==60 || t / div)
            V_DrawPatch(x, y, FB, colon);

    } while (t / div);
}
else
{
    // "sucks"
    V_DrawPatch(x - SHORT(sucks->width), y, FB, sucks);
}
}

```

```

void WI_End(void)
{
    void WI_unloadData(void);
    WI_unloadData();
}

```

```

void WI_initNoState(void)
{
    state = NoState;
    acceleratestage = 0;
    cnt = 10;
}

```

```

void WI_updateNoState(void) {
    WI_updateAnimatedBack();

    if (!--cnt)
    {
        WI_End();
        G_WorldDone();
    }
}

```

```

static boolean          snl_pointeron = false;

```

```

void WI_initShowNextLoc(void)
{
    state = ShowNextLoc;
    acceleratestage = 0;
    cnt = SHOWNEXTLOCDELAY * TICRATE;

    WI_initAnimatedBack();
}

```

```

void WI_updateShowNextLoc(void)
{
    WI_updateAnimatedBack();

    if (!--cnt || acceleratestage)

```

```

        WI_initNoState();
    else
        snl_pointeron = (cnt & 31) < 20;
}

void WI_drawShowNextLoc(void)
{
    int            i;
    int            last;

    WI_slamBackground();

    // draw animated background
    WI_drawAnimatedBack();

    if ( gamemode != commercial)
    {
        if (wbs->epsd > 2)
        {
            WI_drawEL();
            return;
        }

        last = (wbs->last == 8) ? wbs->next - 1 : wbs->last;

        // draw a splat on taken cities.
        for (i=0 ; i<=last ; i++)
            WI_drawOnLnode(i, &splat);

        // splat the secret level?
        if (wbs->didsecret)
            WI_drawOnLnode(8, &splat);

        // draw flashing ptr
        if (snl_pointeron)
            WI_drawOnLnode(wbs->next, yah);
    }

    // draws which level you are entering..
    if ( (gamemode != commercial)
        || wbs->next != 30)
        WI_drawEL();
}

void WI_drawNoState(void)
{
    snl_pointeron = true;
    WI_drawShowNextLoc();
}

int WI_fragSum(int playernum)
{
    int            i;
    int            frags = 0;

    for (i=0 ; i<MAXPLAYERS ; i++)
    {
        if (playeringame[i]
            && i!=playernum)
        {
            frags += plrs[playernum].frags[i];
        }
    }
}

```



```

// JDC hack - negative frags.
frags -= plrs[playernum].frags[playernum];
// UNUSED if (frags < 0)
//         frags = 0;

return frags;
}

static int          dm_state;
static int          dm_frgs[MAXPLAYERS][MAXPLAYERS];
static int          dm_totals[MAXPLAYERS];

```

```

void WI_initDeathmatchStats(void)
{
    int          i;
    int          j;

    state = StatCount;
    acceleratestage = 0;
    dm_state = 1;

    cnt_pause = TICRATE;

    for (i=0 ; i<MAXPLAYERS ; i++)
    {
        if (playeringame[i])
        {
            for (j=0 ; j<MAXPLAYERS ; j++)
                if (playeringame[j])
                    dm_frgs[i][j] = 0;

            dm_totals[i] = 0;
        }
    }

    WI_initAnimatedBack();
}

```

```

void WI_updateDeathmatchStats(void)
{
    int          i;
    int          j;

    boolean      stillticking;

    WI_updateAnimatedBack();

    if (acceleratestage && dm_state != 4)
    {
        acceleratestage = 0;

        for (i=0 ; i<MAXPLAYERS ; i++)
        {
            if (playeringame[i])
            {

```

```

        for (j=0 ; j<MAXPLAYERS ; j++)
            if (playeringame[j])
                dm_frgs[i][j] = plrs[i].frags[j];

        dm_totals[i] = WI_fragSum(i);
    }
}

S_StartSound(0, sfx_barexp);
dm_state = 4;
}

if (dm_state == 2)
{
    if (!(bcnt&3))
        S_StartSound(0, sfx_pistol);

    stillticking = false;

    for (i=0 ; i<MAXPLAYERS ; i++)
    {
        if (playeringame[i])
        {
            for (j=0 ; j<MAXPLAYERS ; j++)
            {
                if (playeringame[j]
                    && dm_frgs[i][j] != plrs[i].frags[j])
                {
                    if (plrs[i].frags[j] < 0)
                        dm_frgs[i][j]--;
                    else
                        dm_frgs[i][j]++;

                    if (dm_frgs[i][j] > 99)
                        dm_frgs[i][j] = 99;

                    if (dm_frgs[i][j] < -99)
                        dm_frgs[i][j] = -99;

                    stillticking = true;
                }
            }
            dm_totals[i] = WI_fragSum(i);

            if (dm_totals[i] > 99)
                dm_totals[i] = 99;

            if (dm_totals[i] < -99)
                dm_totals[i] = -99;
        }
    }

    if (!stillticking)
    {
        S_StartSound(0, sfx_barexp);
        dm_state++;
    }
}

else if (dm_state == 4)
{
    if (acceleratestage)
    {

```

```

        S_StartSound(0, sfx_slop);

        if ( gamemode == commercial)
            WI_initNoState();
        else
            WI_initShowNextLoc();
    }
}
else if (dm_state & 1)
{
    if (!--cnt_pause)
    {
        dm_state++;
        cnt_pause = TICRATE;
    }
}
}
}

```

```

void WI_drawDeathmatchStats(void)
{
    int            i;
    int            j;
    int            x;
    int            y;
    int            w;

    int            lh;          // line height

    lh = WI_SPACINGY;

    WI_slamBackground();

    // draw animated background
    WI_drawAnimatedBack();
    WI_drawLF();

    // draw stat titles (top line)
    V_DrawPatch(DM_TOTALSX-SHORT(total->width)/2,
                DM_MATRIXY-WI_SPACINGY+10,
                FB,
                total);

    V_DrawPatch(DM_KILLERSX, DM_KILLERSY, FB, killers);
    V_DrawPatch(DM_VICTIMSX, DM_VICTIMSY, FB, victims);

    // draw P?
    x = DM_MATRIXX + DM_SPACINGX;
    y = DM_MATRIXY;

    for (i=0 ; i<MAXPLAYERS ; i++)
    {
        if (playeringame[i])
        {
            V_DrawPatch(x-SHORT(p[i]->width)/2,
                        DM_MATRIXY - WI_SPACINGY,
                        FB,
                        p[i]);

            V_DrawPatch(DM_MATRIXX-SHORT(p[i]->width)/2,
                        y,
                        FB,
                        p[i]);
        }
    }
}

```

```

        if (i == me)
        {
            V_DrawPatch(x-SHORT(p[i]->width)/2,
                        DM_MATRIXY - WI_SPACINGY,
                        FB,
                        bstar);

            V_DrawPatch(DM_MATRIXX-SHORT(p[i]->width)/2,
                        y,
                        FB,
                        star);
        }
    }
    else
    {
        // V_DrawPatch(x-SHORT(bp[i]->width)/2,
        //   DM_MATRIXY - WI_SPACINGY, FB, bp[i]);
        // V_DrawPatch(DM_MATRIXX-SHORT(bp[i]->width)/2,
        //   y, FB, bp[i]);
    }
    x += DM_SPACINGX;
    y += WI_SPACINGY;
}

// draw stats
y = DM_MATRIXY+10;
w = SHORT(num[0]->width);

for (i=0 ; i<MAXPLAYERS ; i++)
{
    x = DM_MATRIXX + DM_SPACINGX;

    if (playeringame[i])
    {
        for (j=0 ; j<MAXPLAYERS ; j++)
        {
            if (playeringame[j])
                WI_drawNum(x+w, y, dm_frgs[i][j], 2);

            x += DM_SPACINGX;
        }
        WI_drawNum(DM_TOTALSX+w, y, dm_totals[i], 2);
    }
    y += WI_SPACINGY;
}
}

static int      cnt_frgs[MAXPLAYERS];
static int      dofrags;
static int      ng_state;

void WI_initNetgameStats(void)
{
    int i;

    state = StatCount;
    acceleratestage = 0;
    ng_state = 1;

    cnt_pause = TICRATE;

    for (i=0 ; i<MAXPLAYERS ; i++)
    {

```

```

        if (!playeringame[i])
            continue;

        cnt_kills[i] = cnt_items[i] = cnt_secret[i] = cnt_frgs[i] = 0;

        dofrags += WI_fragSum(i);
    }

    dofrags = !!dofrags;

    WI_initAnimatedBack();
}

void WI_updateNetgameStats(void)
{
    int            i;
    int            fsum;

    boolean        stillticking;

    WI_updateAnimatedBack();

    if (acceleratestage && ng_state != 10)
    {
        acceleratestage = 0;

        for (i=0 ; i<MAXPLAYERS ; i++)
        {
            if (!playeringame[i])
                continue;

            cnt_kills[i] = (plrs[i].skills * 100) / wbs->maxkills;
            cnt_items[i] = (plrs[i].sitems * 100) / wbs->maxitems;
            cnt_secret[i] = (plrs[i].ssecret * 100) / wbs->maxsecret;

            if (dofrags)
                cnt_frgs[i] = WI_fragSum(i);
        }
        S_StartSound(0, sfx_barexp);
        ng_state = 10;
    }

    if (ng_state == 2)
    {
        if (!(bcnt&3))
            S_StartSound(0, sfx_pistol);

        stillticking = false;

        for (i=0 ; i<MAXPLAYERS ; i++)
        {
            if (!playeringame[i])
                continue;

            cnt_kills[i] += 2;

            if (cnt_kills[i] >= (plrs[i].skills * 100) / wbs->maxkills)
                cnt_kills[i] = (plrs[i].skills * 100) / wbs->maxkills;
            else
                stillticking = true;
        }
    }
}

```

```

    if (!stillticking)
    {
        S_StartSound(0, sfx_barexp);
        ng_state++;
    }
}
else if (ng_state == 4)
{
    if (!(bcnt&3))
        S_StartSound(0, sfx_pistol);

    stillticking = false;

    for (i=0 ; i<MAXPLAYERS ; i++)
    {
        if (!playeringame[i])
            continue;

        cnt_items[i] += 2;
        if (cnt_items[i] >= (plrs[i].sitems * 100) / wbs->maxitems)
            cnt_items[i] = (plrs[i].sitems * 100) / wbs->maxitems;
        else
            stillticking = true;
    }
    if (!stillticking)
    {
        S_StartSound(0, sfx_barexp);
        ng_state++;
    }
}
else if (ng_state == 6)
{
    if (!(bcnt&3))
        S_StartSound(0, sfx_pistol);

    stillticking = false;

    for (i=0 ; i<MAXPLAYERS ; i++)
    {
        if (!playeringame[i])
            continue;

        cnt_secret[i] += 2;

        if (cnt_secret[i] >= (plrs[i].ssecret * 100) / wbs->maxsecret)
            cnt_secret[i] = (plrs[i].ssecret * 100) / wbs->maxsecret;
        else
            stillticking = true;
    }

    if (!stillticking)
    {
        S_StartSound(0, sfx_barexp);
        ng_state += 1 + 2*!dofrags;
    }
}
else if (ng_state == 8)
{
    if (!(bcnt&3))
        S_StartSound(0, sfx_pistol);

    stillticking = false;

    for (i=0 ; i<MAXPLAYERS ; i++)
    {

```

```

        if (!playeringame[i])
            continue;

        cnt_f frags[i] += 1;

        if (cnt_f frags[i] >= (fsum = WI_fragSum(i)))
            cnt_f frags[i] = fsum;
        else
            stillticking = true;
    }

    if (!stillticking)
    {
        S_StartSound(0, sfx_pldeth);
        ng_state++;
    }
}
else if (ng_state == 10)
{
    if (acceleratestage)
    {
        S_StartSound(0, sfx_sgcock);
        if ( gamemode == commercial )
            WI_initNoState();
        else
            WI_initShowNextLoc();
    }
}
else if (ng_state & 1)
{
    if (!--cnt_pause)
    {
        ng_state++;
        cnt_pause = TICRATE;
    }
}
}

void WI_drawNetgameStats(void)
{
    int i;
    int x;
    int y;
    int pwidth = SHORT(percent->width);

    WI_slamBackground();

    // draw animated background
    WI_drawAnimatedBack();

    WI_drawLF();

    // draw stat titles (top line)
    V_DrawPatch(NG_STATSX+NG_SPACINGX-SHORT(kills->width),
        NG_STATSY, FB, kills);

    V_DrawPatch(NG_STATSX+2*NG_SPACINGX-SHORT(items->width),
        NG_STATSY, FB, items);

    V_DrawPatch(NG_STATSX+3*NG_SPACINGX-SHORT(secret->width),
        NG_STATSY, FB, secret);

    if (dofrags)

```

```

        V_DrawPatch(NG_STATSX+4*NG_SPACINGX-SHORT( frags->width),
                    NG_STATSY, FB, frags);

// draw stats
y = NG_STATSY + SHORT(kills->height);

for (i=0 ; i<MAXPLAYERS ; i++)
{
    if (!playeringame[i])
        continue;

    x = NG_STATSX;
    V_DrawPatch(x-SHORT(p[i]->width), y, FB, p[i]);

    if (i == me)
        V_DrawPatch(x-SHORT(p[i]->width), y, FB, star);

    x += NG_SPACINGX;
    WI_drawPercent(x-pwidth, y+10, cnt_kills[i]);          x += NG_SPACINGX;
    WI_drawPercent(x-pwidth, y+10, cnt_items[i]);          x += NG_SPACINGX;
    WI_drawPercent(x-pwidth, y+10, cnt_secret[i]);          x += NG_SPACINGX;

    if (dofrags)
        WI_drawNum(x, y+10, cnt_fragments[i], -1);

    y += WI_SPACINGY;
}
}

static int      sp_state;

void WI_initStats(void)
{
    state = StatCount;
    acceleratestage = 0;
    sp_state = 1;
    cnt_kills[0] = cnt_items[0] = cnt_secret[0] = -1;
    cnt_time = cnt_par = -1;
    cnt_pause = TICRATE;

    WI_initAnimatedBack();
}

void WI_updateStats(void)
{
    WI_updateAnimatedBack();

    if (acceleratestage && sp_state != 10)
    {
        acceleratestage = 0;
        cnt_kills[0] = (plrs[me].skills * 100) / wbs->maxkills;
        cnt_items[0] = (plrs[me].sitems * 100) / wbs->maxitems;
        cnt_secret[0] = (plrs[me].ssecret * 100) / wbs->maxsecret;
        cnt_time = plrs[me].stime / TICRATE;
        cnt_par = wbs->partime / TICRATE;
        S_StartSound(0, sfx_barexp);
        sp_state = 10;
    }

    if (sp_state == 2)
    {
        cnt_kills[0] += 2;
    }
}

```



```

    if (!(bcnt&3))
        S_StartSound(0, sfx_pistol);

    if (cnt_kills[0] >= (plrs[me].skills * 100) / wbs->maxkills)
    {
        cnt_kills[0] = (plrs[me].skills * 100) / wbs->maxkills;
        S_StartSound(0, sfx_barexp);
        sp_state++;
    }
}
else if (sp_state == 4)
{
    cnt_items[0] += 2;

    if (!(bcnt&3))
        S_StartSound(0, sfx_pistol);

    if (cnt_items[0] >= (plrs[me].sitems * 100) / wbs->maxitems)
    {
        cnt_items[0] = (plrs[me].sitems * 100) / wbs->maxitems;
        S_StartSound(0, sfx_barexp);
        sp_state++;
    }
}
else if (sp_state == 6)
{
    cnt_secret[0] += 2;

    if (!(bcnt&3))
        S_StartSound(0, sfx_pistol);

    if (cnt_secret[0] >= (plrs[me].ssecret * 100) / wbs->maxsecret)
    {
        cnt_secret[0] = (plrs[me].ssecret * 100) / wbs->maxsecret;
        S_StartSound(0, sfx_barexp);
        sp_state++;
    }
}

else if (sp_state == 8)
{
    if (!(bcnt&3))
        S_StartSound(0, sfx_pistol);

    cnt_time += 3;

    if (cnt_time >= plrs[me].stime / TICRATE)
        cnt_time = plrs[me].stime / TICRATE;

    cnt_par += 3;

    if (cnt_par >= wbs->partime / TICRATE)
    {
        cnt_par = wbs->partime / TICRATE;

        if (cnt_time >= plrs[me].stime / TICRATE)
        {
            S_StartSound(0, sfx_barexp);
            sp_state++;
        }
    }
}
else if (sp_state == 10)
{
    if (acceleratestage)

```

```

    {
        S_StartSound(0, sfx_sgcock);

        if (gamemode == commercial)
            WI_initNoState();
        else
            WI_initShowNextLoc();
    }
}
else if (sp_state & 1)
{
    if (!--cnt_pause)
    {
        sp_state++;
        cnt_pause = TICRATE;
    }
}
}

void WI_drawStats(void)
{
    // line height
    int lh;

    lh = (3*SHORT(num[0]->height))/2;

    WI_slamBackground();

    // draw animated background
    WI_drawAnimatedBack();

    WI_drawLF();

    V_DrawPatch(SP_STATSX, SP_STATSY, FB, kills);
    WI_drawPercent(SCREENWIDTH - SP_STATSX, SP_STATSY, cnt_kills[0]);

    V_DrawPatch(SP_STATSX, SP_STATSY+lh, FB, items);
    WI_drawPercent(SCREENWIDTH - SP_STATSX, SP_STATSY+lh, cnt_items[0]);

    V_DrawPatch(SP_STATSX, SP_STATSY+2*lh, FB, sp_secret);
    WI_drawPercent(SCREENWIDTH - SP_STATSX, SP_STATSY+2*lh, cnt_secret[0]);

    V_DrawPatch(SP_TIMEX, SP_TIMEY, FB, time);
    WI_drawTime(SCREENWIDTH/2 - SP_TIMEX, SP_TIMEY, cnt_time);

    if (wbs->epsd < 3)
    {
        V_DrawPatch(SCREENWIDTH/2 + SP_TIMEX, SP_TIMEY, FB, par);
        WI_drawTime(SCREENWIDTH - SP_TIMEX, SP_TIMEY, cnt_par);
    }
}

void WI_checkForAccelerate(void)
{
    int i;
    player_t *player;

    // check for button presses to skip delays
    for (i=0, player = players ; i<MAXPLAYERS ; i++, player++)
    {
        if (playeringame[i])
        {
            if (player->cmd.buttons & BT_ATTACK)

```

```

        {
            if (!player->attackdown)
                acceleratestage = 1;
            player->attackdown = true;
        }
        else
            player->attackdown = false;
        if (player->cmd.buttons & BT_USE)
        {
            if (!player->usedown)
                acceleratestage = 1;
            player->usedown = true;
        }
        else
            player->usedown = false;
    }
}
}

```

```

// Updates stuff each tick
void WI_Ticker(void)
{
    // counter for general background animation
    bcnt++;

    if (bcnt == 1)
    {
        // intermission music
        if (gamemode == commercial)
            S_ChangeMusic(mus_dm2int, true);
        else
            S_ChangeMusic(mus_inter, true);
    }

    WI_checkForAccelerate();

    switch (state)
    {
        case StatCount:
            if (deathmatch) WI_updateDeathmatchStats();
            else if (netgame) WI_updateNetgameStats();
            else WI_updateStats();
            break;

        case ShowNextLoc:
            WI_updateShowNextLoc();
            break;

        case NoState:
            WI_updateNoState();
            break;
    }
}

```

```

void WI_loadData(void)
{
    int          i;
    int          j;
    char         name[9];
    anim_t*      a;

    if (gamemode == commercial)

```

```

    strcpy(name, "INTERPIC");
else
    sprintf(name, "WIMAP%d", wbs->epsd);

if ( gamemode == retail )
{
    if (wbs->epsd == 3)
        strcpy(name, "INTERPIC");
}

// background
bg = W_CacheLumpName(name, PU_CACHE);
V_DrawPatch(0, 0, 1, bg);

// UNUSED unsigned char *pic = screens[1];
// if (gamemode == commercial)
// {
//     darken the background image
//     while (pic != screens[1] + SCREENHEIGHT*SCREENWIDTH)
//     {
//         *pic = colormaps[256*25 + *pic];
//         pic++;
//     }
// }

if (gamemode == commercial)
{
    NUMCMAPS = 32;
    lnames = (patch_t **) Z_Malloc(sizeof(patch_t*) * NUMCMAPS,
                                   PU_STATIC, 0);

    for (i=0 ; i<NUMCMAPS ; i++)
    {
        sprintf(name, "CWILV%2.2d", i);
        lnames[i] = W_CacheLumpName(name, PU_STATIC);
    }
}
else
{
    lnames = (patch_t **) Z_Malloc(sizeof(patch_t*) * NUMMAPS,
                                   PU_STATIC, 0);

    for (i=0 ; i<NUMMAPS ; i++)
    {
        sprintf(name, "WILV%d%d", wbs->epsd, i);
        lnames[i] = W_CacheLumpName(name, PU_STATIC);
    }

    // you are here
    yah[0] = W_CacheLumpName("WIURH0", PU_STATIC);

    // you are here (alt.)
    yah[1] = W_CacheLumpName("WIURH1", PU_STATIC);

    // splat
    splat = W_CacheLumpName("WISPLAT", PU_STATIC);

    if (wbs->epsd < 3)
    {
        for (j=0; j<NUMANIMS[wbs->epsd]; j++)
        {
            a = &anim[wbs->epsd][j];
            for (i=0; i<a->nanims; i++)
            {
                // MONDO HACK!
                if (wbs->epsd != 1 || j != 8)

```

```

        {
            // animations
            sprintf(name, "WIA%d%.2d%.2d", wbs->epsd, j, i);
            a->p[i] = W_CacheLumpName(name, PU_STATIC);
        }
        else
        {
            // HACK ALERT!
            a->p[i] = anims[1][4].p[i];
        }
    }
}

// More hacks on minus sign.
wiminus = W_CacheLumpName("WIMINUS", PU_STATIC);

for (i=0;i<10;i++)
{
    // numbers 0-9
    sprintf(name, "WINUM%d", i);
    num[i] = W_CacheLumpName(name, PU_STATIC);
}

// percent sign
percent = W_CacheLumpName("WIPCNT", PU_STATIC);

// "finished"
finished = W_CacheLumpName("WIF", PU_STATIC);

// "entering"
entering = W_CacheLumpName("WIENTER", PU_STATIC);

// "kills"
kills = W_CacheLumpName("WIOSTK", PU_STATIC);

// "scrt"
secret = W_CacheLumpName("WIOSTS", PU_STATIC);

// "secret"
sp_secret = W_CacheLumpName("WISCRT2", PU_STATIC);

// Yuck.
if (french)
{
    // "items"
    if (netgame && !deathmatch)
        items = W_CacheLumpName("WIOBJ", PU_STATIC);
    else
        items = W_CacheLumpName("WIOSTI", PU_STATIC);
} else
    items = W_CacheLumpName("WIOSTI", PU_STATIC);

// "frgs"
frags = W_CacheLumpName("WIFRGS", PU_STATIC);

// ":"
colon = W_CacheLumpName("WICOLON", PU_STATIC);

// "time"
time = W_CacheLumpName("WITIME", PU_STATIC);

// "sucks"
sucks = W_CacheLumpName("WISUCKS", PU_STATIC);

```

```

// "par"
par = W_CacheLumpName("WIPAR", PU_STATIC);

// "killers" (vertical)
killers = W_CacheLumpName("WIKILRS", PU_STATIC);

// "victims" (horiz)
victims = W_CacheLumpName("WIVCTMS", PU_STATIC);

// "total"
total = W_CacheLumpName("WIMSTT", PU_STATIC);

// your face
star = W_CacheLumpName("STFST01", PU_STATIC);

// dead face
bstar = W_CacheLumpName("STFDEAD0", PU_STATIC);

for (i=0 ; i<MAXPLAYERS ; i++)
{
    // "1,2,3,4"
    sprintf(name, "STPB%d", i);
    p[i] = W_CacheLumpName(name, PU_STATIC);

    // "1,2,3,4"
    sprintf(name, "WIBP%d", i+1);
    bp[i] = W_CacheLumpName(name, PU_STATIC);
}
}

void WI_unloadData(void)
{
    int i;
    int j;

    Z_ChangeTag(wiminus, PU_CACHE);

    for (i=0 ; i<10 ; i++)
        Z_ChangeTag(num[i], PU_CACHE);

    if (gamemode == commercial)
    {
        for (i=0 ; i<NUMCMAPS ; i++)
            Z_ChangeTag(lnames[i], PU_CACHE);
    }
    else
    {
        Z_ChangeTag(yah[0], PU_CACHE);
        Z_ChangeTag(yah[1], PU_CACHE);

        Z_ChangeTag(splat, PU_CACHE);

        for (i=0 ; i<NUMMAPS ; i++)
            Z_ChangeTag(lnames[i], PU_CACHE);

        if (wbs->epsd < 3)
        {
            for (j=0; j<NUMANIMS[wbs->epsd]; j++)
            {
                if (wbs->epsd != 1 || j != 8)
                    for (i=0; i<anim[wbs->epsd][j].nanims; i++)
                        Z_ChangeTag(anim[wbs->epsd][j].p[i], PU_CACHE);
            }
        }
    }
}

```

```

    }
}

Z_Free(lnames);

Z_ChangeTag(percent, PU_CACHE);
Z_ChangeTag(colon, PU_CACHE);
Z_ChangeTag(finished, PU_CACHE);
Z_ChangeTag(entering, PU_CACHE);
Z_ChangeTag(kills, PU_CACHE);
Z_ChangeTag(secret, PU_CACHE);
Z_ChangeTag(sp_secret, PU_CACHE);
Z_ChangeTag(items, PU_CACHE);
Z_ChangeTag( frags, PU_CACHE);
Z_ChangeTag(time, PU_CACHE);
Z_ChangeTag(sucks, PU_CACHE);
Z_ChangeTag(par, PU_CACHE);

Z_ChangeTag(victims, PU_CACHE);
Z_ChangeTag(killers, PU_CACHE);
Z_ChangeTag(total, PU_CACHE);
// Z_ChangeTag(star, PU_CACHE);
// Z_ChangeTag(bstar, PU_CACHE);

for (i=0 ; i<MAXPLAYERS ; i++)
    Z_ChangeTag(p[i], PU_CACHE);

for (i=0 ; i<MAXPLAYERS ; i++)
    Z_ChangeTag(bp[i], PU_CACHE);
}

void WI_Drawer (void)
{
    switch (state)
    {
        case StatCount:
            if (deathmatch)
                WI_drawDeathmatchStats();
            else if (netgame)
                WI_drawNetgameStats();
            else
                WI_drawStats();
            break;

        case ShowNextLoc:
            WI_drawShowNextLoc();
            break;

        case NoState:
            WI_drawNoState();
            break;
    }
}

void WI_initVariables(wbstartstruct_t* wbstartstruct)
{
    wbs = wbstartstruct;

#ifdef RANGECHECKING
    if (gamemode != commercial)
    {
        if ( gamemode == retail )
            RNGCHECK(wbs->epsd, 0, 3);
    }
}

```

```

        else
            RNGCHECK(wbs->epsd, 0, 2);
    }
    else
    {
        RNGCHECK(wbs->last, 0, 8);
        RNGCHECK(wbs->next, 0, 8);
    }
    RNGCHECK(wbs->pnum, 0, MAXPLAYERS);
    RNGCHECK(wbs->pnum, 0, MAXPLAYERS);
#endif

    acceleratestage = 0;
    cnt = bcnt = 0;
    firstrefresh = 1;
    me = wbs->pnum;
    plrs = wbs->plyr;

    if (!wbs->maxkills)
        wbs->maxkills = 1;

    if (!wbs->maxitems)
        wbs->maxitems = 1;

    if (!wbs->maxsecret)
        wbs->maxsecret = 1;

    if ( gamemode != retail )
        if (wbs->epsd > 2)
            wbs->epsd -= 3;
}

void WI_Start(wbstartstruct_t* wbstartstruct)
{
    WI_initVariables(wbstartstruct);
    WI_loadData();

    if (deathmatch)
        WI_initDeathmatchStats();
    else if (netgame)
        WI_initNetgameStats();
    else
        WI_initStats();
}

```

## 15.2 wi\_stuff.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This source is available for distribution and/or modification
// only under the terms of the DOOM Source Code License as
// published by id Software. All rights reserved.
//
// The source is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// FITNESS FOR A PARTICULAR PURPOSE. See the DOOM Source Code License
// for more details.
//
// DESCRIPTION:

```



```

// Intermission.
//
//-----

#ifndef __WI_STUFF__
#define __WI_STUFF__

//#include "v_video.h"

#include "doomdef.h"

// States for the intermission

typedef enum
{
    NoState = -1,
    StatCount,
    ShowNextLoc
} stateenum_t;

// Called by main loop, animate the intermission.
void WI_Ticker (void);

// Called by main loop,
// draws the intermission directly into the screen buffer.
void WI_Drawer (void);

// Setup for an intermission screen.
void WI_Start(wbstartstruct_t*          wbstartstruct);

#endif
//-----
//
// $Log:$
//
//-----

```

## 16 Zone memory allocation system

### 16.1 z\_zone.c

```

// Emacs style mode select  -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This source is available for distribution and/or modification
// only under the terms of the DOOM Source Code License as
// published by id Software. All rights reserved.
//
// The source is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// FITNESS FOR A PARTICULAR PURPOSE. See the DOOM Source Code License
// for more details.
//
// $Log:$
//
// DESCRIPTION:
//     Zone Memory Allocation. Neat.
//
//-----

```

```

static const char
rcsid[] = "$Id: z_zone.c,v 1.4 1997/02/03 16:47:58 b1 Exp $";

#include "z_zone.h"
#include "i_system.h"
#include "doomdef.h"

//
// ZONE MEMORY ALLOCATION
//
// There is never any space between memblocks,
// and there will never be two contiguous free memblocks.
// The rover can be left pointing at a non-empty block.
//
// It is of no value to free a cachable block,
// because it will get overwritten automatically if needed.
//

#define ZONEID          0x1d4a11

typedef struct
{
    // total bytes malloced, including header
    int          size;

    // start / end cap for linked list
    memblock_t    blocklist;

    memblock_t*    rover;
} memzone_t;

memzone_t*        mainzone;

//
// Z_ClearZone
//
void Z_ClearZone (memzone_t* zone)
{
    memblock_t*    block;

    // set the entire zone to one free block
    zone->blocklist.next =
        zone->blocklist.prev =
        block = (memblock_t *) ( (byte *)zone + sizeof(memzone_t) );

    zone->blocklist.user = (void *)zone;
    zone->blocklist.tag = PU_STATIC;
    zone->rover = block;

    block->prev = block->next = &zone->blocklist;

    // NULL indicates a free block.
    block->user = NULL;

    block->size = zone->size - sizeof(memzone_t);
}

```

```

//
// Z_Init
//
void Z_Init (void)
{
    memblock_t*      block;
    int              size;

    mainzone = (memzone_t *)I_ZoneBase (&size);
    mainzone->size = size;

    // set the entire zone to one free block
    mainzone->blocklist.next =
        mainzone->blocklist.prev =
            block = (memblock_t *) ( (byte *)mainzone + sizeof(memzone_t) );

    mainzone->blocklist.user = (void *)mainzone;
    mainzone->blocklist.tag = PU_STATIC;
    mainzone->rover = block;

    block->prev = block->next = &mainzone->blocklist;

    // NULL indicates a free block.
    block->user = NULL;

    block->size = mainzone->size - sizeof(memzone_t);
}

```

```

//
// Z_Free
//
void Z_Free (void* ptr)
{
    memblock_t*      block;
    memblock_t*      other;

    block = (memblock_t *) ( (byte *)ptr - sizeof(memblock_t));

    if (block->id != ZONEID)
        I_Error ("Z_Free: freed a pointer without ZONEID");

    if (block->user > (void **)0x100)
    {
        // smaller values are not pointers
        // Note: OS-dependend?

        // clear the user's mark
        *block->user = 0;
    }

    // mark as free
    block->user = NULL;
    block->tag = 0;
    block->id = 0;

    other = block->prev;

    if (!other->user)
    {
        // merge with previous free block
        other->size += block->size;
        other->next = block->next;
        other->next->prev = other;
    }
}

```

```

        if (block == mainzone->rover)
            mainzone->rover = other;

        block = other;
    }

    other = block->next;
    if (!other->user)
    {
        // merge the next free block onto the end
        block->size += other->size;
        block->next = other->next;
        block->next->prev = block;

        if (other == mainzone->rover)
            mainzone->rover = block;
    }
}

//
// Z_Malloc
// You can pass a NULL user if the tag is < PU_PURGELEVEL.
//
#define MINFRAGMENT          64

void*
Z_Malloc
( int          size,
  int          tag,
  void*        user )
{
    int          extra;
    memblock_t*  start;
    memblock_t*  rover;
    memblock_t*  newblock;
    memblock_t*  base;

    size = (size + 3) & ~3;

    // scan through the block list,
    // looking for the first free block
    // of sufficient size,
    // throwing out any purgable blocks along the way.

    // account for size of block header
    size += sizeof(memblock_t);

    // if there is a free block behind the rover,
    // back up over them
    base = mainzone->rover;

    if (!base->prev->user)
        base = base->prev;

    rover = base;
    start = base->prev;

    do
    {
        if (rover == start)
        {

```

```

        // scanned all the way around the list
        I_Error ("Z_Malloc: failed on allocation of %i bytes", size);
    }

    if (rover->user)
    {
        if (rover->tag < PU_PURGELEVEL)
        {
            // hit a block that can't be purged,
            // so move base past it
            base = rover = rover->next;
        }
        else
        {
            // free the rover block (adding the size to base)

            // the rover can be the base block
            base = base->prev;
            Z_Free ((byte *)rover+sizeof(memblock_t));
            base = base->next;
            rover = base->next;
        }
    }
    else
        rover = rover->next;
} while (base->user || base->size < size);

// found a block big enough
extra = base->size - size;

if (extra > MINFRAGMENT)
{
    // there will be a free fragment after the allocated block
    newblock = (memblock_t *) ((byte *)base + size );
    newblock->size = extra;

    // NULL indicates free block.
    newblock->user = NULL;
    newblock->tag = 0;
    newblock->prev = base;
    newblock->next = base->next;
    newblock->next->prev = newblock;

    base->next = newblock;
    base->size = size;
}

if (user)
{
    // mark as an in use block
    base->user = user;
    *(void **)user = (void *) ((byte *)base + sizeof(memblock_t));
}
else
{
    if (tag >= PU_PURGELEVEL)
        I_Error ("Z_Malloc: an owner is required for purgable blocks");

    // mark as in use, but unowned
    base->user = (void *)2;
}
base->tag = tag;

// next allocation will start looking here

```

```

    mainzone->rover = base->next;

    base->id = ZONEID;

    return (void *) ((byte *)base + sizeof(memblock_t));
}

```

```

//
// Z_FreeTags
//
void
Z_FreeTags
( int          lowtag,
  int          hightag )
{
    memblock_t*    block;
    memblock_t*    next;

    for (block = mainzone->blocklist.next ;
         block != &mainzone->blocklist ;
         block = next)
    {
        // get link before freeing
        next = block->next;

        // free block?
        if (!block->user)
            continue;

        if (block->tag >= lowtag && block->tag <= hightag)
            Z_Free ( (byte *)block+sizeof(memblock_t));
    }
}

```

```

//
// Z_DumpHeap
// Note: TFileDumpHeap( stdout ) ?
//
void
Z_DumpHeap
( int          lowtag,
  int          hightag )
{
    memblock_t*    block;

    printf ("zone size: %i  location: %p\n",
            mainzone->size,mainzone);

    printf ("tag range: %i to %i\n",
            lowtag, hightag);

    for (block = mainzone->blocklist.next ; ; block = block->next)
    {
        if (block->tag >= lowtag && block->tag <= hightag)
            printf ("block:%p  size:%7i  user:%p  tag:%3i\n",
                    block, block->size, block->user, block->tag);

        if (block->next == &mainzone->blocklist)
        {
            // all blocks have been hit
            break;
        }
    }
}

```

```

    }

    if ( (byte *)block + block->size != (byte *)block->next)
        printf ("ERROR: block size does not touch the next block\n");

    if ( block->next->prev != block)
        printf ("ERROR: next block doesn't have proper back link\n");

    if (!block->user && !block->next->user)
        printf ("ERROR: two consecutive free blocks\n");
}
}

//
// Z_FileDumpHeap
//
void Z_FileDumpHeap (FILE* f)
{
    memblock_t*      block;

    fprintf (f,"zone size: %i  location: %p\n",mainzone->size,mainzone);

    for (block = mainzone->blocklist.next ; ; block = block->next)
    {
        fprintf (f,"block:%p  size:%7i  user:%p  tag:%3i\n",
            block, block->size, block->user, block->tag);

        if (block->next == &mainzone->blocklist)
        {
            // all blocks have been hit
            break;
        }

        if ( (byte *)block + block->size != (byte *)block->next)
            fprintf (f,"ERROR: block size does not touch the next block\n");

        if ( block->next->prev != block)
            fprintf (f,"ERROR: next block doesn't have proper back link\n");

        if (!block->user && !block->next->user)
            fprintf (f,"ERROR: two consecutive free blocks\n");
    }
}

//
// Z_CheckHeap
//
void Z_CheckHeap (void)
{
    memblock_t*      block;

    for (block = mainzone->blocklist.next ; ; block = block->next)
    {
        if (block->next == &mainzone->blocklist)
        {
            // all blocks have been hit
            break;
        }

        if ( (byte *)block + block->size != (byte *)block->next)
            I_Error ("Z_CheckHeap: block size does not touch the next block\n");
    }
}

```

```

    if ( block->next->prev != block)
        I_Error ("Z_CheckHeap: next block doesn't have proper back link\n");

    if (!block->user && !block->next->user)
        I_Error ("Z_CheckHeap: two consecutive free blocks\n");
}
}

```

```

//
// Z_ChangeTag
//
void
Z_ChangeTag2
( void*          ptr,
  int            tag )
{
    memblock_t*   block;

    block = (memblock_t *) ( (byte *)ptr - sizeof(memblock_t));

    if (block->id != ZONEID)
        I_Error ("Z_ChangeTag: freed a pointer without ZONEID");

    if (tag >= PU_PURGELEVEL && (unsigned)block->user < 0x100)
        I_Error ("Z_ChangeTag: an owner is required for purgable blocks");

    block->tag = tag;
}

```

```

//
// Z_FreeMemory
//
int Z_FreeMemory (void)
{
    memblock_t*   block;
    int           free;

    free = 0;

    for (block = mainzone->blocklist.next ;
         block != &mainzone->blocklist;
         block = block->next)
    {
        if (!block->user || block->tag >= PU_PURGELEVEL)
            free += block->size;
    }
    return free;
}

```

## 16.2 z\_zone.h

```

// Emacs style mode select   -*- C++ -*-
//-----
//
// $Id:$
//
// Copyright (C) 1993-1996 by id Software, Inc.
//
// This source is available for distribution and/or modification

```



```

// only under the terms of the DOOM Source Code License as
// published by id Software. All rights reserved.
//
// The source is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// FITNESS FOR A PARTICULAR PURPOSE. See the DOOM Source Code License
// for more details.
//
// DESCRIPTION:
//     Zone Memory Allocation, perhaps NeXT ObjectiveC inspired.
//     Remark: this was the only stuff that, according
//     to John Carmack, might have been useful for
//     Quake.
//
//-----

#ifdef __Z_ZONE__
#define __Z_ZONE__

#include <stdio.h>

//
// ZONE MEMORY
// PU - purge tags.
// Tags < 100 are not overwritten until freed.
#define PU_STATIC          1          // static entire execution time
#define PU_SOUND           2          // static while playing
#define PU_MUSIC           3          // static while playing
#define PU_DAVE            4          // anything else Dave wants static
#define PU_LEVEL           50         // static until level exited
#define PU_LEVSPEC         51         // a special thinker in a level
// Tags >= 100 are purgable whenever needed.
#define PU_PURGELEVEL      100
#define PU_CACHE           101

void      Z_Init (void);
void*     Z_Malloc (int size, int tag, void *ptr);
void      Z_Free (void *ptr);
void      Z_FreeTags (int lowtag, int hightag);
void      Z_DumpHeap (int lowtag, int hightag);
void      Z_FileDumpHeap (FILE *f);
void      Z_CheckHeap (void);
void      Z_ChangeTag2 (void *ptr, int tag);
int       Z_FreeMemory (void);

typedef struct memblock_s
{
    int          size;          // including the header and possibly tiny fragments
    void**       user;          // NULL if a free block
    int          tag;           // purgelevel
    int          id;            // should be ZONEID
    struct memblock_s* next;
    struct memblock_s* prev;
} memblock_t;

//
// This is used to get the local FILE:LINE info from CPP
// prior to really call the function in question.
//
#define Z_ChangeTag(p,t) \
{ \

```

```

        if (( (memblock_t *) ( (byte *) (p) - sizeof(memblock_t))) ->id!=0x1d4a11) \
            I_Error("Z_CT at "__FILE__":%i",__LINE__); \
            Z_ChangeTag2(p,t); \
};

#endif
//-----
//
// $Log:$
//
//-----

```