Quake Progs 1.06 Source -- 36 .qc files in Total -- Available at http://wiki.quakesrc.org/files/progs106.zip
Also useful: http://www.quakeone.com/navigator May 17, 2006

Browse progs 1.06 online at

http://www.inside3d.com/browse.php

progs.src

../progs.dat

defs.qc
subs.qc
fight.qc
ai.qc
combat.qc
items.qc
weapons.qc
world.qc
client.qc
player.qc
monsters.qc
doors.qc
buttons.qc
triggers.qc
plats.qc
misc.qc

ogre.qc
demon.qc
shambler.qc
knight.qc
soldier.qc
wizard.qc
dog.qc
zombie.qc
boss.qc

tarbaby.qc // registered
hknight.qc // registered
fish.qc // registered
shalrath.qc // registered
enforcer.qc // registered
oldone.qc // registered

This document is in the following order: Non-monster QC files alphabetically, followed monsters qc files, alphabetically.

AI.QC

```
void() movetarget_f;
void() t_movetarget;
void() knight_walk1;
void() knight_bow6;
void() knight_bow1;
void(entity etemp, entity stemp, entity stemp, float dmg) T_Damage;
/*

.enemy
Will be world if not currently angry at anyone.

.movetarget
The next path spot to walk toward.  If .enemy, ignore .movetarget.
When an enemy is killed, the monster will try to return to it's path.

.huntt_ime
Set to time + something when the player is in sight, but movement straight for
him is blocked.  This causes the monster to use wall following code for
movement direction instead of sighting on the player.

.ideal_yaw
A yaw angle of the intended direction, which will be turned towards at up
to 45 deg / state.  If the enemy is in view and hunt_time is not active,
this will be the exact line towards the enemy.

.pausetime
A monster will leave it's stand state and head towards it's .movetarget when
time > .pausetime.

walkmove(angle, speed) primitive is all or nothing
*/


//
// globals
//
float      current_yaw;

//
// when a monster becomes angry at a player, that monster will be used
// as the sight target the next frame so that monsters near that one
// will wake up even if they wouldn't have noticed the player
//
entity     sight_entity;
float      sight_entity_time;

float(float v) anglemod =
{
        while (v >= 360)
                v = v - 360;
        while (v < 0)
                v = v + 360;
        return v;
};

/*
==============================================================================

MOVETARGET CODE
```

The angle of the movetarget effects standing and bowing direction, but has no effect on movement, which allways heads to the next target.

targetname
must be present. The name of this movetarget.

target
the next spot to move to. If not present, stop here for good.

pausetime
The number of seconds to spend standing or bowing for path_stand or path_bow

==========
*/

void() movetarget_f =
{
	if (!self.targetname)
		objerror ("monster_movetarget: no targetname");

	self.solid = SOLID_TRIGGER;
	self.touch = t_movetarget;
	setsize (self, '-8 -8 -8', '8 8 8');

};

/*QUAKED path_corner (0.5 0.3 0) (-8 -8 -8) (8 8 8)
Monsters will continue walking towards the next target corner.
*/
void() path_corner =
{
	movetarget_f ();
};

/*
=============
t_movetarget

Something has bumped into a movetarget. If it is a monster
moving towards it, change the next destination and continue.
==============
*/
void() t_movetarget =
{
local entity	temp;

	if (other.movetarget != self)
		return;

	if (other.enemy)
		return;		// fighting, not following a path

	temp = self;
	self = other;
	other = temp;

	if (self.classname == "monster_ogre")
		sound (self, CHAN_VOICE, "ogre/ogdrag.wav", 1, ATTN_IDLE);// play chainsaw drag sound

//dprint ("t_movetarget\n");

```
	self.goalentity = self.movetarget = find (world, targetname, other.target);
	self.ideal_yaw = vectoyaw(self.goalentity.origin - self.origin);
	if (!self.movetarget)
	{
		self.pausetime = time + 999999;
		self.th_stand ();
		return;
	}
};



//============================================================================

/*
=============
range

returns the range catagorization of an entity reletive to self
0	melee range, will become hostile even if back is turned
1	visibility and infront, or visibility and show hostile
2	infront and show hostile
3	only triggered by damage
=============
*/
float(entity targ) range =
{
local vector	spot1, spot2;
local float		r;
	spot1 = self.origin + self.view_ofs;
	spot2 = targ.origin + targ.view_ofs;

	r = vlen (spot1 - spot2);
	if (r < 120)
		return RANGE_MELEE;
	if (r < 500)
		return RANGE_NEAR;
	if (r < 1000)
		return RANGE_MID;
	return RANGE_FAR;
};

/*
=============
visible

returns 1 if the entity is visible to self, even if not infront ()
=============
*/
float (entity targ) visible =
{
	local vector	spot1, spot2;

	spot1 = self.origin + self.view_ofs;
	spot2 = targ.origin + targ.view_ofs;
	traceline (spot1, spot2, TRUE, self);	// see through other monsters

	if (trace_inopen && trace_inwater)
		return FALSE;			// sight line crossed contents

	if (trace_fraction == 1)
		return TRUE;
```

```
	return FALSE;
};


/*
=============
infront

returns 1 if the entity is in front (in sight) of self
=============
*/
float(entity targ) infront =
{
	local vector	vec;
	local float		dot;

	makevectors (self.angles);
	vec = normalize (targ.origin - self.origin);
	dot = vec * v_forward;

	if ( dot > 0.3)
	{
		return TRUE;
	}
	return FALSE;
};


//============================================================================

/*
===========
ChangeYaw

Turns towards self.ideal_yaw at self.yaw_speed
Sets the global variable current_yaw
Called every 0.1 sec by monsters
============
*/
/*

void() ChangeYaw =
{
	local float		ideal, move;

//current_yaw = self.ideal_yaw;
// mod down the current angle
	current_yaw = anglemod( self.angles_y );
	ideal = self.ideal_yaw;

	if (current_yaw == ideal)
		return;

	move = ideal - current_yaw;
	if (ideal > current_yaw)
	{
		if (move > 180)
			move = move - 360;
	}
	else
	{
		if (move < -180)
```

```
                move = move + 360;
        }

        if (move > 0)
        {
                if (move > self.yaw_speed)
                        move = self.yaw_speed;
        }
        else
        {
                if (move < 0-self.yaw_speed )
                        move = 0-self.yaw_speed;
        }

        current_yaw = anglemod (current_yaw + move);

        self.angles_y = current_yaw;
};

*/


//============================================================================

void() HuntTarget =
{
        self.goalentity = self.enemy;
        self.think = self.th_run;
        self.ideal_yaw = vectoyaw(self.enemy.origin - self.origin);
        self.nextthink = time + 0.1;
        SUB_AttackFinished (1);             // wait a while before first attack
};

void() SightSound =
{
local float         rsnd;

        if (self.classname == "monster_ogre")
                sound (self, CHAN_VOICE, "ogre/ogwake.wav", 1, ATTN_NORM);
        else if (self.classname == "monster_knight")
                sound (self, CHAN_VOICE, "knight/ksight.wav", 1, ATTN_NORM);
        else if (self.classname == "monster_shambler")
                sound (self, CHAN_VOICE, "shambler/ssight.wav", 1, ATTN_NORM);
        else if (self.classname == "monster_demon1")
                sound (self, CHAN_VOICE, "demon/sight2.wav", 1, ATTN_NORM);
        else if (self.classname == "monster_wizard")
                sound (self, CHAN_VOICE, "wizard/wsight.wav", 1, ATTN_NORM);
        else if (self.classname == "monster_zombie")
                sound (self, CHAN_VOICE, "zombie/z_idle.wav", 1, ATTN_NORM);
        else if (self.classname == "monster_dog")
                sound (self, CHAN_VOICE, "dog/dsight.wav", 1, ATTN_NORM);
        else if (self.classname == "monster_hell_knight")
                sound (self, CHAN_VOICE, "hknight/sight1.wav", 1, ATTN_NORM);
        else if (self.classname == "monster_tarbaby")
                sound (self, CHAN_VOICE, "blob/sight1.wav", 1, ATTN_NORM);
        else if (self.classname == "monster_vomit")
                sound (self, CHAN_VOICE, "vomitus/v_sight1.wav", 1, ATTN_NORM);
        else if (self.classname == "monster_enforcer")
        {
                rsnd = rint(random() * 3);
                if (rsnd == 1)
                        sound (self, CHAN_VOICE, "enforcer/sight1.wav", 1, ATTN_NORM);
```

```
		else if (rsnd == 2)
			sound (self, CHAN_VOICE, "enforcer/sight2.wav", 1, ATTN_NORM);
		else if (rsnd == 0)
			sound (self, CHAN_VOICE, "enforcer/sight3.wav", 1, ATTN_NORM);
		else
			sound (self, CHAN_VOICE, "enforcer/sight4.wav", 1, ATTN_NORM);
	}
	else if (self.classname == "monster_army")
		sound (self, CHAN_VOICE, "soldier/sight1.wav", 1, ATTN_NORM);
	else if (self.classname == "monster_shalrath")
		sound (self, CHAN_VOICE, "shalrath/sight.wav", 1, ATTN_NORM);
};

void() FoundTarget =
{
	if (self.enemy.classname == "player")
	{	// let other monsters see this monster for a while
		sight_entity = self;
		sight_entity_time = time;
	}

	self.show_hostile = time + 1;		// wake up other monsters

	SightSound ();
	HuntTarget ();
};

/*
============
FindTarget

Self is currently not attacking anything, so try to find a target

Returns TRUE if an enemy was sighted

When a player fires a missile, the point of impact becomes a fakeplayer so
that monsters that see the impact will respond as if they had seen the
player.

To avoid spending too much time, only a single client (or fakeclient) is
checked each frame.  This means multi player games will have slightly
slower noticing monsters.
============
*/
float() FindTarget =
{
	local entity	client;
	local float		r;

// if the first spawnflag bit is set, the monster will only wake up on
// really seeing the player, not another monster getting angry

// spawnflags & 3 is a big hack, because zombie crucified used the first
// spawn flag prior to the ambush flag, and I forgot about it, so the second
// spawn flag works as well
	if (sight_entity_time >= time - 0.1 && !(self.spawnflags & 3) )
	{
		client = sight_entity;
		if (client.enemy == self.enemy)
			return;
	}
	else
```

```
	{
		client = checkclient ();
		if (!client)
			return FALSE;	// current check entity isn't in PVS
	}

	if (client == self.enemy)
		return FALSE;

	if (client.flags & FL_NOTARGET)
		return FALSE;
	if (client.items & IT_INVISIBILITY)
		return FALSE;

	r = range (client);
	if (r == RANGE_FAR)
		return FALSE;

	if (!visible (client))
		return FALSE;

	if (r == RANGE_NEAR)
	{
		if (client.show_hostile < time && !infront (client))
			return FALSE;
	}
	else if (r == RANGE_MID)
	{
		if ( /* client.show_hostile < time || */ !infront (client))
			return FALSE;
	}

//
// got one
//
	self.enemy = client;
	if (self.enemy.classname != "player")
	{
		self.enemy = self.enemy.enemy;
		if (self.enemy.classname != "player")
		{
			self.enemy = world;
			return FALSE;
		}
	}

	FoundTarget ();

	return TRUE;
};


//============================================================================

void(float dist) ai_forward =
{
	walkmove (self.angles_y, dist);
};

void(float dist) ai_back =
{
	walkmove ( (self.angles_y+180), dist);
```

```
};


/*
=============
ai_pain

stagger back a bit
=============
*/
void(float dist) ai_pain =
{
	ai_back (dist);
/*
	local float	away;

	away = anglemod (vectoyaw (self.origin - self.enemy.origin)
	+ 180*(random()- 0.5) );

	walkmove (away, dist);
*/
};

/*
=============
ai_painforward

stagger back a bit
=============
*/
void(float dist) ai_painforward =
{
	walkmove (self.ideal_yaw, dist);
};

/*
=============
ai_walk

The monster is walking it's beat
=============
*/
void(float dist) ai_walk =
{
	local vector		mtemp;

	movedist = dist;

	if (self.classname == "monster_dragon")
	{
		movetogoal (dist);
		return;
	}
	// check for noticing a player
	if (FindTarget ())
		return;

	movetogoal (dist);
};


/*
```

```
=============
ai_stand

The monster is staying in one place for a while, with slight angle turns
=============
*/
void() ai_stand =
{
	if (FindTarget ())
		return;

	if (time > self.pausetime)
	{
		self.th_walk ();
		return;
	}

// change angle slightly

};

/*
=============
ai_turn

don't move, but turn towards ideal_yaw
=============
*/
void() ai_turn =
{
	if (FindTarget ())
		return;

	ChangeYaw ();
};

//============================================================================

/*
=============
ChooseTurn
=============
*/
void(vector dest3) ChooseTurn =
{
	local vector	dir, newdir;

	dir = self.origin - dest3;

	newdir_x = trace_plane_normal_y;
	newdir_y = 0 - trace_plane_normal_x;
	newdir_z = 0;

	if (dir * newdir > 0)
	{
		dir_x = 0 - trace_plane_normal_y;
		dir_y = trace_plane_normal_x;
	}
	else
	{
		dir_x = trace_plane_normal_y;
		dir_y = 0 - trace_plane_normal_x;
```

```
	}

	dir_z = 0;
	self.ideal_yaw = vectoyaw(dir);
};

/*
============
FacingIdeal

============
*/
float() FacingIdeal =
{
	local	float	delta;

	delta = anglemod(self.angles_y - self.ideal_yaw);
	if (delta > 45 && delta < 315)
		return FALSE;
	return TRUE;
};


//============================================================================

float()	WizardCheckAttack;
float()	DogCheckAttack;

float() CheckAnyAttack =
{
	if (!enemy_vis)
		return;
	if (self.classname == "monster_army")
		return SoldierCheckAttack ();
	if (self.classname == "monster_ogre")
		return OgreCheckAttack ();
	if (self.classname == "monster_shambler")
		return ShamCheckAttack ();
	if (self.classname == "monster_demon1")
		return DemonCheckAttack ();
	if (self.classname == "monster_dog")
		return DogCheckAttack ();
	if (self.classname == "monster_wizard")
		return WizardCheckAttack ();
	return CheckAttack ();
};


/*
=============
ai_run_melee

Turn and close until within an angle to launch a melee attack
=============
*/
void() ai_run_melee =
{
	self.ideal_yaw = enemy_yaw;
	ChangeYaw ();

	if (FacingIdeal())
	{
```

```
                self.th_melee ();
                self.attack_state = AS_STRAIGHT;
        }
};


/*
=============
ai_run_missile

Turn in place until within an angle to launch a missile attack
=============
*/
void() ai_run_missile =
{
        self.ideal_yaw = enemy_yaw;
        ChangeYaw ();
        if (FacingIdeal())
        {
                self.th_missile ();
                self.attack_state = AS_STRAIGHT;
        }
};


/*
=============
ai_run_slide

Strafe sideways, but stay at aproximately the same range
=============
*/
void() ai_run_slide =
{
        local float          ofs;

        self.ideal_yaw = enemy_yaw;
        ChangeYaw ();
        if (self.lefty)
                ofs = 90;
        else
                ofs = -90;

        if (walkmove (self.ideal_yaw + ofs, movedist))
                return;

        self.lefty = 1 - self.lefty;

        walkmove (self.ideal_yaw - ofs, movedist);
};


/*
=============
ai_run

The monster has an enemy it is trying to kill
=============
*/
void(float dist) ai_run =
{
        local     vector   delta;
```

```
	local	float	axis;
	local	float	direct, ang_rint, ang_floor, ang_ceil;

	movedist = dist;
// see if the enemy is dead
	if (self.enemy.health <= 0)
	{
		self.enemy = world;
	// FIXME: look all around for other targets
		if (self.oldenemy.health > 0)
		{
			self.enemy = self.oldenemy;
			HuntTarget ();
		}
		else
		{
			if (self.movetarget)
				self.th_walk ();
			else
				self.th_stand ();
			return;
		}
	}

	self.show_hostile = time + 1;		// wake up other monsters

// check knowledge of enemy
	enemy_vis = visible(self.enemy);
	if (enemy_vis)
		self.search_time = time + 5;

// look for other coop players
	if (coop && self.search_time < time)
	{
		if (FindTarget ())
			return;
	}

	enemy_infront = infront(self.enemy);
	enemy_range = range(self.enemy);
	enemy_yaw = vectoyaw(self.enemy.origin - self.origin);

	if (self.attack_state == AS_MISSILE)
	{
//dprint ("ai_run_missile\n");
		ai_run_missile ();
		return;
	}
	if (self.attack_state == AS_MELEE)
	{
//dprint ("ai_run_melee\n");
		ai_run_melee ();
		return;
	}

	if (CheckAnyAttack ())
		return;					// beginning an attack

	if (self.attack_state == AS_SLIDING)
	{
		ai_run_slide ();
		return;
```

```
        }

// head straight in
        movetogoal (dist);                  // done in C code...
};
```

AMTEST.QC

```
/*~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>
~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~*/

void() test_teleport_touch;
void() tele_done;

/*QUAKED test_teleport (0 .5 .8) ?
Teleporter testing
*/
void() test_teleport =
{
	precache_model ("sprites/s_aball.spr");
	setsize (self, self.mins, self.maxs);
	self.touch = test_teleport_touch;
	self.solid = 1;

	if (!self.target)
		objerror ("no target\n");
};

void() test_teleport_touch =
{
local entity oldself;
	other.movetype = MOVETYPE_TOSS;
//	other.solid = SOLID_NOT;
	other.dest = '256 -128 -128';
	oldself = self;
	self = other;
//	SUB_CalcMove (self.dest, 200, tele_done);
	self.velocity = '1000 0 0 ';
	self = oldself;
};

void() tele_done =
{
	self.movetype = MOVETYPE_WALK;
	self.solid = SOLID_SLIDEBOX;
};

/*~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>
~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~*/

void() test_goaway;
void() test_spawn;

/*QUAKED test_fodder (0 .5 .8) ?
beating guy
*/
void() test_fodder =
{
	self.nextthink = time + 3;
	self.think = test_spawn;
};

void() test_spawn =
{
local entity	body;
	makevectors (self.angles);

	body = spawn();
```

```
	setmodel (body, "progs/soldier.mdl");
	setorigin (body, self.origin);
	body.classname = "player";
	body.health = 1000;
	body.frags = 0;
	body.takedamage = DAMAGE_AIM;
	body.solid = SOLID_SLIDEBOX;
	body.movetype = MOVETYPE_WALK;
	body.show_hostile = 0;
	body.weapon = 1;
	body.velocity = v_forward * 200;

	body.nextthink = time + 5;
	body.think = test_goaway;

self.nextthink = time + 3;
self.think = test_spawn;

};

void() test_goaway =
{
	remove (self);
};
```

BUTTONS.QC

```
// button and multiple button

void() button_wait;
void() button_return;

void() button_wait =
{
	self.state = STATE_TOP;
	self.nextthink = self.ltime + self.wait;
	self.think = button_return;
	activator = self.enemy;
	SUB_UseTargets();
	self.frame = 1;			// use alternate textures
};

void() button_done =
{
	self.state = STATE_BOTTOM;
};

void() button_return =
{
	self.state = STATE_DOWN;
	SUB_CalcMove (self.pos1, self.speed, button_done);
	self.frame = 0;			// use normal textures
	if (self.health)
		self.takedamage = DAMAGE_YES;	// can be shot again
};


void() button_blocked =
{	// do nothing, just don't ome all the way back out
};


void() button_fire =
{
	if (self.state == STATE_UP || self.state == STATE_TOP)
		return;

	sound (self, CHAN_VOICE, self.noise, 1, ATTN_NORM);

	self.state = STATE_UP;
	SUB_CalcMove (self.pos2, self.speed, button_wait);
};


void() button_use =
{
	self.enemy = activator;
	button_fire ();
};

void() button_touch =
{
	if (other.classname != "player")
		return;
	self.enemy = other;
	button_fire ();
};
```

```
void() button_killed =
{
	self.enemy = damage_attacker;
	self.health = self.max_health;
	self.takedamage = DAMAGE_NO;	// wil be reset upon return
	button_fire ();
};


/*QUAKED func_button (0 .5 .8) ?
When a button is touched, it moves some distance in the direction of it's angle, triggers all of it's targets, waits some time, then returns to it's original position where it can be triggered again.

"angle"		determines the opening direction
"target" all entities with a matching targetname will be used
"speed"		override the default 40 speed
"wait"		override the default 1 second wait (-1 = never return)
"lip"		override the default 4 pixel lip remaining at end of move
"health" if set, the button must be killed instead of touched
"sounds"
0) steam metal
1) wooden clunk
2) metallic click
3) in-out
*/
void() func_button =
{
local float			gtemp, ftemp;

	if (self.sounds == 0)
	{
		precache_sound ("buttons/airbut1.wav");
		self.noise = "buttons/airbut1.wav";
	}
	if (self.sounds == 1)
	{
		precache_sound ("buttons/switch21.wav");
		self.noise = "buttons/switch21.wav";
	}
	if (self.sounds == 2)
	{
		precache_sound ("buttons/switch02.wav");
		self.noise = "buttons/switch02.wav";
	}
	if (self.sounds == 3)
	{
		precache_sound ("buttons/switch04.wav");
		self.noise = "buttons/switch04.wav";
	}

	SetMovedir ();

	self.movetype = MOVETYPE_PUSH;
	self.solid = SOLID_BSP;
	setmodel (self, self.model);

	self.blocked = button_blocked;
	self.use = button_use;

	if (self.health)
	{
```

```
		self.max_health = self.health;
		self.th_die = button_killed;
		self.takedamage = DAMAGE_YES;
	}
	else
		self.touch = button_touch;

	if (!self.speed)
		self.speed = 40;
	if (!self.wait)
		self.wait = 1;
	if (!self.lip)
		self.lip = 4;

	self.state = STATE_BOTTOM;

	self.pos1 = self.origin;
	self.pos2 = self.pos1 + self.movedir*(fabs(self.movedir*self.size) - self.lip);
};
```

CLIENT.QC

```
// prototypes
void () W_WeaponFrame;
void() W_SetCurrentAmmo;
void() player_pain;
void() player_stand1;
void (vector org) spawn_tfog;
void (vector org, entity death_owner) spawn_tdeath;

float	modelindex_eyes, modelindex_player;

/*
==============================================================================

                                LEVEL CHANGING / INTERMISSION

==============================================================================
*/

float	intermission_running;
float	intermission_exittime;

/*QUAKED info_intermission (1 0.5 0.5) (-16 -16 -16) (16 16 16)
This is the camera point for the intermission.
Use mangle instead of angle, so you can set pitch or roll as well as yaw.  'pitch roll yaw'
*/
void() info_intermission =
{
};



void() SetChangeParms =
{
	if (self.health <= 0)
	{
		SetNewParms ();
		return;
	}

// remove items
	self.items = self.items - (self.items &
	(IT_KEY1 | IT_KEY2 | IT_INVISIBILITY | IT_INVULNERABILITY | IT_SUIT | IT_QUAD) );

// cap super health
	if (self.health > 100)
		self.health = 100;
	if (self.health < 50)
		self.health = 50;
	parm1 = self.items;
	parm2 = self.health;
	parm3 = self.armorvalue;
	if (self.ammo_shells < 25)
		parm4 = 25;
	else
		parm4 = self.ammo_shells;
	parm5 = self.ammo_nails;
	parm6 = self.ammo_rockets;
	parm7 = self.ammo_cells;
	parm8 = self.weapon;
```

```
	parm9 = self.armortype * 100;
};

void() SetNewParms =
{
	parm1 = IT_SHOTGUN | IT_AXE;
	parm2 = 100;
	parm3 = 0;
	parm4 = 25;
	parm5 = 0;
	parm6 = 0;
	parm7 = 0;
	parm8 = 1;
	parm9 = 0;
};

void() DecodeLevelParms =
{
	if (serverflags)
	{
		if (world.model == "maps/start.bsp")
			SetNewParms ();			// take away all stuff on starting new episode
	}

	self.items = parm1;
	self.health = parm2;
	self.armorvalue = parm3;
	self.ammo_shells = parm4;
	self.ammo_nails = parm5;
	self.ammo_rockets = parm6;
	self.ammo_cells = parm7;
	self.weapon = parm8;
	self.armortype = parm9 * 0.01;
};

/*
============
FindIntermission

Returns the entity to view from
============
*/
entity() FindIntermission =
{
	local	entity spot;
	local	float cyc;

// look for info_intermission first
	spot = find (world, classname, "info_intermission");
	if (spot)
	{	// pick a random one
		cyc = random() * 4;
		while (cyc > 1)
		{
			spot = find (spot, classname, "info_intermission");
			if (!spot)
				spot = find (spot, classname, "info_intermission");
			cyc = cyc - 1;
		}
		return spot;
	}
```

```
// then look for the start position
	spot = find (world, classname, "info_player_start");
	if (spot)
		return spot;

// testinfo_player_start is only found in regioned levels
	spot = find (world, classname, "testplayerstart");
	if (spot)
		return spot;

	objerror ("FindIntermission: no spot");
};


string nextmap;
void() GotoNextMap =
{
	if (cvar("samelevel"))	// if samelevel is set, stay on same level
		changelevel (mapname);
	else
		changelevel (nextmap);
};


void() ExitIntermission =
{
// skip any text in deathmatch
	if (deathmatch)
	{
		GotoNextMap ();
		return;
	}

	intermission_exittime = time + 1;
	intermission_running = intermission_running + 1;

//
// run some text if at the end of an episode
//
	if (intermission_running == 2)
	{
		if (world.model == "maps/e1m7.bsp")
		{
			WriteByte (MSG_ALL, SVC_CDTRACK);
			WriteByte (MSG_ALL, 2);
			WriteByte (MSG_ALL, 3);
			if (!cvar("registered"))
			{
				WriteByte (MSG_ALL, SVC_FINALE);
				WriteString (MSG_ALL, "As the corpse of the monstrous entity\nChthon sinks back into the lava whence\nit rose, you grip the Rune of Earth\nMagic tightly. Now that you have\nconquered the Dimension of the Doomed,\nrealm of Earth Magic, you are ready to\ncomplete your task in the other three\nhaunted lands of Quake. Or are you? If\nyou don't register Quake, you'll never\nknow what awaits you in the Realm of\nBlack Magic, the Netherworld, and the\nElder World!");
			}
			else
			{
				WriteByte (MSG_ALL, SVC_FINALE);
				WriteString (MSG_ALL, "As the corpse of the monstrous entity\nChthon sinks back into the lava whence\nit rose, you grip the Rune of Earth\nMagic tightly. Now that you have\nconquered the Dimension of the Doomed,\nrealm of Earth Magic, you are ready to\ncomplete your task. A Rune of magic\npower lies at the end of each haunted\nland of Quake. Go forth, seek the\ntotality of the four Runes!");
```

```
			}
			return;
		}
		else if (world.model == "maps/e2m6.bsp")
		{
			WriteByte (MSG_ALL, SVC_CDTRACK);
			WriteByte (MSG_ALL, 2);
			WriteByte (MSG_ALL, 3);

			WriteByte (MSG_ALL, SVC_FINALE);
			WriteString (MSG_ALL, "The Rune of Black Magic throbs evilly in\nyour hand and whispers dark thoughts\ninto your brain. You learn the inmost\nlore of the Hell-Mother; Shub-Niggurath!\nYou now know that she is behind all the\nterrible plotting which has led to so\nmuch death and horror. But she is not\ninviolate! Armed with this Rune, you\nrealize that once all four Runes are\ncombined, the gate to Shub-Niggurath's\nPit will open, and you can face the\nWitch-Goddess herself in her frightful\notherworld cathedral.");
			return;
		}
		else if (world.model == "maps/e3m6.bsp")
		{
			WriteByte (MSG_ALL, SVC_CDTRACK);
			WriteByte (MSG_ALL, 2);
			WriteByte (MSG_ALL, 3);

			WriteByte (MSG_ALL, SVC_FINALE);
			WriteString (MSG_ALL, "The charred viscera of diabolic horrors\nbubble viscously as you seize the Rune\nof Hell Magic. Its heat scorches your\nhand, and its terrible secrets blight\nyour mind. Gathering the shreds of your\ncourage, you shake the devil's shackles\nfrom your soul, and become ever more\nhard and determined to destroy the\nhideous creatures whose mere existence\nthreatens the souls and psyches of all\nthe population of Earth.");
			return;
		}
		else if (world.model == "maps/e4m7.bsp")
		{
			WriteByte (MSG_ALL, SVC_CDTRACK);
			WriteByte (MSG_ALL, 2);
			WriteByte (MSG_ALL, 3);

			WriteByte (MSG_ALL, SVC_FINALE);
			WriteString (MSG_ALL, "Despite the awful might of the Elder\nWorld, you have achieved the Rune of\nElder Magic, capstone of all types of\narcane wisdom. Beyond good and evil,\nbeyond life and death, the Rune\npulsates, heavy with import. Patient and\npotent, the Elder Being Shub-Niggurath\nweaves her dire plans to clear off all\nlife from the Earth, and bring her own\nfoul offspring to our world! For all the\ndwellers in these nightmare dimensions\nare her descendants! Once all Runes of\nmagic power are united, the energy\nbehind them will blast open the Gateway\nto Shub-Niggurath, and you can travel\nthere to foil the Hell-Mother's plots\nin person.");
			return;
		}

		GotoNextMap();
	}

	if (intermission_running == 3)
	{
		if (!cvar("registered"))
		{	// shareware episode has been completed, go to sell screen
			WriteByte (MSG_ALL, SVC_SELLSCREEN);
			return;
		}

		if ( (serverflags&15) == 15)
		{
			WriteByte (MSG_ALL, SVC_FINALE);
			WriteString (MSG_ALL, "Now, you have all four Runes. You sense\ntremendous invisible forces moving to\nunseal ancient barriers. Shub-Niggurath\nhad hoped to use the Runes Herself to\nclear off the Earth, but now
```

```
instead,\nyou will use them to enter her home and\nconfront her as an avatar of avenging\nEarth-life. If you defeat her,
you will\nbe remembered forever as the savior of\nthe planet. If she conquers, it will be\nas if you had never been born.");
                                return;
                        }

                }

                GotoNextMap();
};

/*
============
IntermissionThink

When the player presses attack or jump, change to the next level
============
*/
void() IntermissionThink =
{
        if (time < intermission_exittime)
                return;

        if (!self.button0 && !self.button1 && !self.button2)
                return;

        ExitIntermission ();
};

void() execute_changelevel =
{
        local entity        pos;

        intermission_running = 1;

// enforce a wait time before allowing changelevel
        if (deathmatch)
                intermission_exittime = time + 5;
        else
                intermission_exittime = time + 2;

        WriteByte (MSG_ALL, SVC_CDTRACK);
        WriteByte (MSG_ALL, 3);
        WriteByte (MSG_ALL, 3);

        pos = FindIntermission ();

        other = find (world, classname, "player");
        while (other != world)
        {
                other.view_ofs = '0 0 0';
                other.angles = other.v_angle = pos.mangle;
                other.fixangle = TRUE;              // turn this way immediately
                other.nextthink = time + 0.5;
                other.takedamage = DAMAGE_NO;
                other.solid = SOLID_NOT;
                other.movetype = MOVETYPE_NONE;
                other.modelindex = 0;
                setorigin (other, pos.origin);
                other = find (other, classname, "player");
        }

        WriteByte (MSG_ALL, SVC_INTERMISSION);
```

```
};


void() changelevel_touch =
{
	local entity	pos;

	if (other.classname != "player")
		return;

	if ((cvar("noexit") == 1) || ((cvar("noexit") == 2) && (mapname != "start")))
	{
		T_Damage (other, self, self, 50000);
		return;
	}

	if (coop || deathmatch)
	{
		bprint (other.netname);
		bprint (" exited the level\n");
	}

	nextmap = self.map;

	SUB_UseTargets ();

	if ( (self.spawnflags & 1) && (deathmatch == 0) )
	{	// NO_INTERMISSION
		GotoNextMap();
		return;
	}

	self.touch = SUB_Null;

// we can't move people right now, because touch functions are called
// in the middle of C movement code, so set a think time to do it
	self.think = execute_changelevel;
	self.nextthink = time + 0.1;
};

/*QUAKED trigger_changelevel (0.5 0.5 0.5) ? NO_INTERMISSION
When the player touches this, he gets sent to the map listed in the "map" variable.  Unless the NO_INTERMISSION flag is set, the view will go to the info_intermission spot and display stats.
*/
void() trigger_changelevel =
{
	if (!self.map)
		objerror ("chagnelevel trigger doesn't have map");

	InitTrigger ();
	self.touch = changelevel_touch;
};


/*
==============================================================================

PLAYER GAME EDGE FUNCTIONS

==============================================================================
*/
```

```
void() set_suicide_frame;

// called by ClientKill and DeadThink
void() respawn =
{
	if (coop)
	{
		// make a copy of the dead body for appearances sake
		CopyToBodyQue (self);
		// get the spawn parms as they were at level start
		setspawnparms (self);
		// respawn
		PutClientInServer ();
	}
	else if (deathmatch)
	{
		// make a copy of the dead body for appearances sake
		CopyToBodyQue (self);
		// set default spawn parms
		SetNewParms ();
		// respawn
		PutClientInServer ();
	}
	else
	{	// restart the entire server
		localcmd ("restart\n");
	}
};


/*
============
ClientKill

Player entered the suicide command
============
*/
void() ClientKill =
{
	bprint (self.netname);
	bprint (" suicides\n");
	set_suicide_frame ();
	self.modelindex = modelindex_player;
	self.frags = self.frags - 2;		// extra penalty
	respawn ();
};

float(vector v) CheckSpawnPoint =
{
	return FALSE;
};

/*
============
SelectSpawnPoint

Returns the entity to spawn at
============
*/
entity() SelectSpawnPoint =
{
	local	entity spot;
```

```
	local	entity thing;
	local	float  pcount;

// testinfo_player_start is only found in regioned levels
	spot = find (world, classname, "testplayerstart");
	if (spot)
		return spot;

// choose a info_player_deathmatch point
	if (coop)
	{
		lastspawn = find(lastspawn, classname, "info_player_coop");
		if (lastspawn == world)
			lastspawn = find (lastspawn, classname, "info_player_start");
		if (lastspawn != world)
			return lastspawn;
	}
	else if (deathmatch)
	{
		spot = lastspawn;
		while (1)
		{
			spot = find(spot, classname, "info_player_deathmatch");
			if (spot != world)
			{
				if (spot == lastspawn)
					return lastspawn;
				pcount = 0;
				thing = findradius(spot.origin, 32);
				while(thing)
				{
					if (thing.classname == "player")
						pcount = pcount + 1;
					thing = thing.chain;
				}
				if (pcount == 0)
				{
					lastspawn = spot;
					return spot;
				}
			}
		}
	}

	if (serverflags)
	{	// return with a rune to start
		spot = find (world, classname, "info_player_start2");
		if (spot)
			return spot;
	}

	spot = find (world, classname, "info_player_start");
	if (!spot)
		error ("PutClientInServer: no info_player_start on level");

	return spot;
};

/*
===========
PutClientInServer
```

```
called each time a player is spawned
============
*/
void() DecodeLevelParms;
void() PlayerDie;


void() PutClientInServer =
{
	local	entity spot;

	spot = SelectSpawnPoint ();

	self.classname = "player";
	self.health = 100;
	self.takedamage = DAMAGE_AIM;
	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_WALK;
	self.show_hostile = 0;
	self.max_health = 100;
	self.flags = FL_CLIENT;
	self.air_finished = time + 12;
	self.dmg = 2;  		// initial water damage
	self.super_damage_finished = 0;
	self.radsuit_finished = 0;
	self.invisible_finished = 0;
	self.invincible_finished = 0;
	self.effects = 0;
	self.invincible_time = 0;

	DecodeLevelParms ();

	W_SetCurrentAmmo ();

	self.attack_finished = time;
	self.th_pain = player_pain;
	self.th_die = PlayerDie;

	self.deadflag = DEAD_NO;
// paustime is set by teleporters to keep the player from moving a while
	self.pausetime = 0;

//	spot = SelectSpawnPoint ();

	self.origin = spot.origin + '0 0 1';
	self.angles = spot.angles;
	self.fixangle = TRUE;  		// turn this way immediately

// oh, this is a hack!
	setmodel (self, "progs/eyes.mdl");
	modelindex_eyes = self.modelindex;

	setmodel (self, "progs/player.mdl");
	modelindex_player = self.modelindex;

	setsize (self, VEC_HULL_MIN, VEC_HULL_MAX);

	self.view_ofs = '0 0 22';

	player_stand1 ();

	if (deathmatch || coop)
```

```
	{
		makevectors(self.angles);
		spawn_tfog (self.origin + v_forward*20);
	}

	spawn_tdeath (self.origin, self);
};


/*
==============================================================================

QUAKED FUNCTIONS

==============================================================================
*/


/*QUAKED info_player_start (1 0 0) (-16 -16 -24) (16 16 24)
The normal starting point for a level.
*/
void() info_player_start =
{
};


/*QUAKED info_player_start2 (1 0 0) (-16 -16 -24) (16 16 24)
Only used on start map for the return point from an episode.
*/
void() info_player_start2 =
{
};


/*
saved out by quaked in region mode
*/
void() testplayerstart =
{
};

/*QUAKED info_player_deathmatch (1 0 1) (-16 -16 -24) (16 16 24)
potential spawning position for deathmatch games
*/
void() info_player_deathmatch =
{
};

/*QUAKED info_player_coop (1 0 1) (-16 -16 -24) (16 16 24)
potential spawning position for coop games
*/
void() info_player_coop =
{
};

/*
==============================================================================

RULES

==============================================================================
*/
```

```
/*
go to the next level for deathmatch
only called if a time or frag limit has expired
*/
void() NextLevel =
{
	local entity o;

	if (mapname == "start")
	{
		if (!cvar("registered"))
		{
			mapname = "e1m1";
		}
		else if (!(serverflags & 1))
		{
			mapname = "e1m1";
			serverflags = serverflags | 1;
		}
		else if (!(serverflags & 2))
		{
			mapname = "e2m1";
			serverflags = serverflags | 2;
		}
		else if (!(serverflags & 4))
		{
			mapname = "e3m1";
			serverflags = serverflags | 4;
		}
		else if (!(serverflags & 8))
		{
			mapname = "e4m1";
			serverflags = serverflags - 7;
		}

		o = spawn();
		o.map = mapname;
	}
	else
	{
		// find a trigger changelevel
		o = find(world, classname, "trigger_changelevel");

		// go back to start if no trigger_changelevel
		if (!o)
		{
			mapname = "start";
			o = spawn();
			o.map = mapname;
		}
	}

	nextmap = o.map;
	gameover = TRUE;

	if (o.nextthink < time)
	{
		o.think = execute_changelevel;
		o.nextthink = time + 0.1;
	}
};
```

```
/*
============
CheckRules

Exit deathmatch games upon conditions
============
*/
void() CheckRules =
{
	local	float		timelimit;
	local	float		fraglimit;

	if (gameover)	// someone else quit the game already
		return;

	timelimit = cvar("timelimit") * 60;
	fraglimit = cvar("fraglimit");

	if (timelimit && time >= timelimit)
	{
		NextLevel ();
		return;
	}

	if (fraglimit && self.frags >= fraglimit)
	{
		NextLevel ();
		return;
	}
};

//============================================================================

void() PlayerDeathThink =
{
	local entity	old_self;
	local float		forward;

	if ((self.flags & FL_ONGROUND))
	{
		forward = vlen (self.velocity);
		forward = forward - 20;
		if (forward <= 0)
			self.velocity = '0 0 0';
		else
			self.velocity = forward * normalize(self.velocity);
	}

// wait for all buttons released
	if (self.deadflag == DEAD_DEAD)
	{
		if (self.button2 || self.button1 || self.button0)
			return;
		self.deadflag = DEAD_RESPAWNABLE;
		return;
	}

// wait for any button down
	if (!self.button2 && !self.button1 && !self.button0)
		return;
```

```
	self.button0 = 0;
	self.button1 = 0;
	self.button2 = 0;
	respawn();
};


void() PlayerJump =
{
	local vector start, end;

	if (self.flags & FL_WATERJUMP)
		return;

	if (self.waterlevel >= 2)
	{
		if (self.watertype == CONTENT_WATER)
			self.velocity_z = 100;
		else if (self.watertype == CONTENT_SLIME)
			self.velocity_z = 80;
		else
			self.velocity_z = 50;

// play swiming sound
		if (self.swim_flag < time)
		{
			self.swim_flag = time + 1;
			if (random() < 0.5)
				sound (self, CHAN_BODY, "misc/water1.wav", 1, ATTN_NORM);
			else
				sound (self, CHAN_BODY, "misc/water2.wav", 1, ATTN_NORM);
		}

		return;
	}

	if (!(self.flags & FL_ONGROUND))
		return;

	if ( !(self.flags & FL_JUMPRELEASED) )
		return;		// don't pogo stick

	self.flags = self.flags - (self.flags & FL_JUMPRELEASED);

	self.flags = self.flags - FL_ONGROUND;// don't stairwalk

	self.button2 = 0;
// player jumping sound
	sound (self, CHAN_BODY, "player/plyrjmp8.wav", 1, ATTN_NORM);
	self.velocity_z = self.velocity_z + 270;
};


/*
===========
WaterMove

============
*/
.float	dmgtime;

void() WaterMove =
```

```
{
//dprint (ftos(self.waterlevel));
	if (self.movetype == MOVETYPE_NOCLIP)
		return;
	if (self.health < 0)
		return;

	if (self.waterlevel != 3)
	{
		if (self.air_finished < time)
			sound (self, CHAN_VOICE, "player/gasp2.wav", 1, ATTN_NORM);
		else if (self.air_finished < time + 9)
			sound (self, CHAN_VOICE, "player/gasp1.wav", 1, ATTN_NORM);
		self.air_finished = time + 12;
		self.dmg = 2;
	}
	else if (self.air_finished < time)
	{	// drown!
		if (self.pain_finished < time)
		{
			self.dmg = self.dmg + 2;
			if (self.dmg > 15)
				self.dmg = 10;
			T_Damage (self, world, world, self.dmg);
			self.pain_finished = time + 1;
		}
	}

	if (!self.waterlevel)
	{
		if (self.flags & FL_INWATER)
		{
			// play leave water sound
			sound (self, CHAN_BODY, "misc/outwater.wav", 1, ATTN_NORM);
			self.flags = self.flags - FL_INWATER;
		}
		return;
	}

	if (self.watertype == CONTENT_LAVA)
	{	// do damage
		if (self.dmgtime < time)
		{
			if (self.radsuit_finished > time)
				self.dmgtime = time + 1;
			else
				self.dmgtime = time + 0.2;

			T_Damage (self, world, world, 10*self.waterlevel);
		}
	}
	else if (self.watertype == CONTENT_SLIME)
	{	// do damage
		if (self.dmgtime < time && self.radsuit_finished < time)
		{
			self.dmgtime = time + 1;
			T_Damage (self, world, world, 4*self.waterlevel);
		}
	}

	if ( !(self.flags & FL_INWATER) )
	{
```

```
// player enter water sound

                if (self.watertype == CONTENT_LAVA)
                        sound (self, CHAN_BODY, "player/inlava.wav", 1, ATTN_NORM);
                if (self.watertype == CONTENT_WATER)
                        sound (self, CHAN_BODY, "player/inh2o.wav", 1, ATTN_NORM);
                if (self.watertype == CONTENT_SLIME)
                        sound (self, CHAN_BODY, "player/slimbrn2.wav", 1, ATTN_NORM);

                self.flags = self.flags + FL_INWATER;
                self.dmgtime = 0;
        }

        if (! (self.flags & FL_WATERJUMP) )
                self.velocity = self.velocity - 0.8*self.waterlevel*frametime*self.velocity;
};

void() CheckWaterJump =
{
        local vector start, end;

// check for a jump-out-of-water
        makevectors (self.angles);
        start = self.origin;
        start_z = start_z + 8;
        v_forward_z = 0;
        normalize(v_forward);
        end = start + v_forward*24;
        traceline (start, end, TRUE, self);
        if (trace_fraction < 1)
        {       // solid at waist
                start_z = start_z + self.maxs_z - 8;
                end = start + v_forward*24;
                self.movedir = trace_plane_normal * -50;
                traceline (start, end, TRUE, self);
                if (trace_fraction == 1)
                {       // open at eye level
                        self.flags = self.flags | FL_WATERJUMP;
                        self.velocity_z = 225;
                        self.flags = self.flags - (self.flags & FL_JUMPRELEASED);
                        self.teleport_time = time + 2;      // safety net
                        return;
                }
        }
};


/*
================
PlayerPreThink

Called every frame before physics are run
================
*/
void() PlayerPreThink =
{
        local   float   mspeed, aspeed;
        local   float   r;

        if (intermission_running)
        {
```

```
		IntermissionThink ();	// otherwise a button could be missed between
		return;					// the think tics
	}

	if (self.view_ofs == '0 0 0')
		return;		// intermission or finale

	makevectors (self.v_angle);		// is this still used

	CheckRules ();
	WaterMove ();

	if (self.waterlevel == 2)
		CheckWaterJump ();

	if (self.deadflag >= DEAD_DEAD)
	{
		PlayerDeathThink ();
		return;
	}

	if (self.deadflag == DEAD_DYING)
		return;	// dying, so do nothing

	if (self.button2)
	{
		PlayerJump ();
	}
	else
		self.flags = self.flags | FL_JUMPRELEASED;

// teleporters can force a non-moving pause time
	if (time < self.pausetime)
		self.velocity = '0 0 0';

	if(time > self.attack_finished && self.currentammo == 0 && self.weapon != IT_AXE)
	{
		self.weapon = W_BestWeapon ();
		W_SetCurrentAmmo ();
	}
};

/*
================
CheckPowerups

Check for turning off powerups
================
*/
void() CheckPowerups =
{
	if (self.health <= 0)
		return;

// invisibility
	if (self.invisible_finished)
	{
// sound and screen flash when items starts to run out
		if (self.invisible_sound < time)
		{
			sound (self, CHAN_AUTO, "items/inv3.wav", 0.5, ATTN_IDLE);
			self.invisible_sound = time + ((random() * 3) + 1);
```

```
		}


		if (self.invisible_finished < time + 3)
		{
			if (self.invisible_time == 1)
			{
				sprint (self, "Ring of Shadows magic is fading\n");
				stuffcmd (self, "bf\n");
				sound (self, CHAN_AUTO, "items/inv2.wav", 1, ATTN_NORM);
				self.invisible_time = time + 1;
			}

			if (self.invisible_time < time)
			{
				self.invisible_time = time + 1;
				stuffcmd (self, "bf\n");
			}
		}

		if (self.invisible_finished < time)
		{	// just stopped
			self.items = self.items - IT_INVISIBILITY;
			self.invisible_finished = 0;
			self.invisible_time = 0;
		}

	// use the eyes
		self.frame = 0;
		self.modelindex = modelindex_eyes;
	}
	else
		self.modelindex = modelindex_player;	// don't use eyes

// invincibility
	if (self.invincible_finished)
	{
// sound and screen flash when items starts to run out
		if (self.invincible_finished < time + 3)
		{
			if (self.invincible_time == 1)
			{
				sprint (self, "Protection is almost burned out\n");
				stuffcmd (self, "bf\n");
				sound (self, CHAN_AUTO, "items/protect2.wav", 1, ATTN_NORM);
				self.invincible_time = time + 1;
			}

			if (self.invincible_time < time)
			{
				self.invincible_time = time + 1;
				stuffcmd (self, "bf\n");
			}
		}

		if (self.invincible_finished < time)
		{	// just stopped
			self.items = self.items - IT_INVULNERABILITY;
			self.invincible_time = 0;
			self.invincible_finished = 0;
		}
		if (self.invincible_finished > time)
```

```
                self.effects = self.effects | EF_DIMLIGHT;
        else
                self.effects = self.effects - (self.effects & EF_DIMLIGHT);
    }

// super damage
    if (self.super_damage_finished)
    {

// sound and screen flash when items starts to run out

        if (self.super_damage_finished < time + 3)
        {
            if (self.super_time == 1)
            {
                sprint (self, "Quad Damage is wearing off\n");
                stuffcmd (self, "bf\n");
                sound (self, CHAN_AUTO, "items/damage2.wav", 1, ATTN_NORM);
                self.super_time = time + 1;
            }

            if (self.super_time < time)
            {
                self.super_time = time + 1;
                stuffcmd (self, "bf\n");
            }
        }

        if (self.super_damage_finished < time)
        {   // just stopped
            self.items = self.items - IT_QUAD;
            self.super_damage_finished = 0;
            self.super_time = 0;
        }
        if (self.super_damage_finished > time)
            self.effects = self.effects | EF_DIMLIGHT;
        else
            self.effects = self.effects - (self.effects & EF_DIMLIGHT);
    }

// suit
    if (self.radsuit_finished)
    {
        self.air_finished = time + 12;                  // don't drown

// sound and screen flash when items starts to run out
        if (self.radsuit_finished < time + 3)
        {
            if (self.rad_time == 1)
            {
                sprint (self, "Air supply in Biosuit expiring\n");
                stuffcmd (self, "bf\n");
                sound (self, CHAN_AUTO, "items/suit2.wav", 1, ATTN_NORM);
                self.rad_time = time + 1;
            }

            if (self.rad_time < time)
            {
                self.rad_time = time + 1;
                stuffcmd (self, "bf\n");
            }
        }
```

```
		if (self.radsuit_finished < time)
		{		// just stopped
			self.items = self.items - IT_SUIT;
			self.rad_time = 0;
			self.radsuit_finished = 0;
		}
	}

};


/*
================
PlayerPostThink

Called every frame after physics are run
================
*/
void() PlayerPostThink =
{
	local	float	mspeed, aspeed;
	local	float	r;

	if (self.view_ofs == '0 0 0')
		return;		// intermission or finale
	if (self.deadflag)
		return;

// do weapon stuff

	W_WeaponFrame ();

// check to see if player landed and play landing sound
	if ((self.jump_flag < -300) && (self.flags & FL_ONGROUND) && (self.health > 0))
	{
		if (self.watertype == CONTENT_WATER)
			sound (self, CHAN_BODY, "player/h2ojump.wav", 1, ATTN_NORM);
		else if (self.jump_flag < -650)
		{
			T_Damage (self, world, world, 5);
			sound (self, CHAN_VOICE, "player/land2.wav", 1, ATTN_NORM);
			self.deathtype = "falling";
		}
		else
			sound (self, CHAN_VOICE, "player/land.wav", 1, ATTN_NORM);

		self.jump_flag = 0;
	}

	if (!(self.flags & FL_ONGROUND))
		self.jump_flag = self.velocity_z;

	CheckPowerups ();
};


/*
===========
ClientConnect

called when a player connects to a server
```

```
============
*/
void() ClientConnect =
{
	bprint (self.netname);
	bprint (" entered the game\n");

// a client connecting during an intermission can cause problems
	if (intermission_running)
		ExitIntermission ();
};


/*
============
ClientDisconnect

called when a player disconnects from a server
============
*/
void() ClientDisconnect =
{
	if (gameover)
		return;
	// if the level end trigger has been activated, just return
	// since they aren't *really* leaving

	// let everyone else know
	bprint (self.netname);
	bprint (" left the game with ");
	bprint (ftos(self.frags));
	bprint (" frags\n");
	sound (self, CHAN_BODY, "player/tornoff2.wav", 1, ATTN_NONE);
	set_suicide_frame ();
};

/*
============
ClientObituary

called when a player dies
============
*/
void(entity targ, entity attacker) ClientObituary =
{
	local	float rnum;
	local	string deathstring, deathstring2;
	rnum = random();

	if (targ.classname == "player")
	{
		if (attacker.classname == "teledeath")
		{
			bprint (targ.netname);
			bprint (" was telefragged by ");
			bprint (attacker.owner.netname);
			bprint ("\n");

			attacker.owner.frags = attacker.owner.frags + 1;
			return;
		}
```

```
if (attacker.classname == "teledeath2")
{
	bprint ("Satan's power deflects ");
	bprint (targ.netname);
	bprint ("'s telefrag\n");

	targ.frags = targ.frags - 1;
	return;
}

if (attacker.classname == "player")
{
	if (targ == attacker)
	{
		// killed self
		attacker.frags = attacker.frags - 1;
		bprint (targ.netname);

		if (targ.weapon == 64 && targ.waterlevel > 1)
		{
			bprint (" discharges into the water.\n");
			return;
		}
		if (targ.weapon == IT_GRENADE_LAUNCHER)
			bprint (" tries to put the pin back in\n");
		else
			bprint (" becomes bored with life\n");
		return;
	}
	else if ( (teamplay == 2) && (targ.team > 0)&&(targ.team == attacker.team) )
	{
		if (rnum < 0.25)
			deathstring = " mows down a teammate\n";
		else if (rnum < 0.50)
			deathstring = " checks his glasses\n";
		else if (rnum < 0.75)
			deathstring = " gets a frag for the other team\n";
		else
			deathstring = " loses another friend\n";
		bprint (attacker.netname);
		bprint (deathstring);
		attacker.frags = attacker.frags - 1;
		return;
	}
	else
	{
		attacker.frags = attacker.frags + 1;

		rnum = attacker.weapon;
		if (rnum == IT_AXE)
		{
			deathstring = " was ax-murdered by ";
			deathstring2 = "\n";
		}
		if (rnum == IT_SHOTGUN)
		{
			deathstring = " chewed on ";
			deathstring2 = "'s boomstick\n";
		}
		if (rnum == IT_SUPER_SHOTGUN)
		{
			deathstring = " ate 2 loads of ";
```

```
                deathstring2 = "'s buckshot\n";
        }
        if (rnum == IT_NAILGUN)
        {
                deathstring = " was nailed by ";
                deathstring2 = "\n";
        }
        if (rnum == IT_SUPER_NAILGUN)
        {
                deathstring = " was punctured by ";
                deathstring2 = "\n";
        }
        if (rnum == IT_GRENADE_LAUNCHER)
        {
                deathstring = " eats ";
                deathstring2 = "'s pineapple\n";
                if (targ.health < -40)
                {
                        deathstring = " was gibbed by ";
                        deathstring2 = "'s grenade\n";
                }
        }
        if (rnum == IT_ROCKET_LAUNCHER)
        {
                deathstring = " rides ";
                deathstring2 = "'s rocket\n";
                if (targ.health < -40)
                {
                        deathstring = " was gibbed by ";
                        deathstring2 = "'s rocket\n" ;
                }
        }
        if (rnum == IT_LIGHTNING)
        {
                deathstring = " accepts ";
                if (attacker.waterlevel > 1)
                        deathstring2 = "'s discharge\n";
                else
                        deathstring2 = "'s shaft\n";
        }
        bprint (targ.netname);
        bprint (deathstring);
        bprint (attacker.netname);
        bprint (deathstring2);
    }
    return;
}
else
{
    targ.frags = targ.frags - 1;
    bprint (targ.netname);

    // killed by a montser?
    if (attacker.flags & FL_MONSTER)
    {
        if (attacker.classname == "monster_army")
                bprint (" was shot by a Grunt\n");
        if (attacker.classname == "monster_demon1")
                bprint (" was eviscerated by a Fiend\n");
        if (attacker.classname == "monster_dog")
                bprint (" was mauled by a Rottweiler\n");
        if (attacker.classname == "monster_dragon")
```

```
                bprint (" was fried by a Dragon\n");
        if (attacker.classname == "monster_enforcer")
                bprint (" was blasted by an Enforcer\n");
        if (attacker.classname == "monster_fish")
                bprint (" was fed to the Rotfish\n");
        if (attacker.classname == "monster_hell_knight")
                bprint (" was slain by a Death Knight\n");
        if (attacker.classname == "monster_knight")
                bprint (" was slashed by a Knight\n");
        if (attacker.classname == "monster_ogre")
                bprint (" was destroyed by an Ogre\n");
        if (attacker.classname == "monster_oldone")
                bprint (" became one with Shub-Niggurath\n");
        if (attacker.classname == "monster_shalrath")
                bprint (" was exploded by a Vore\n");
        if (attacker.classname == "monster_shambler")
                bprint (" was smashed by a Shambler\n");
        if (attacker.classname == "monster_tarbaby")
                bprint (" was slimed by a Spawn\n");
        if (attacker.classname == "monster_vomit")
                bprint (" was vomited on by a Vomitus\n");
        if (attacker.classname == "monster_wizard")
                bprint (" was scragged by a Scrag\n");
        if (attacker.classname == "monster_zombie")
                bprint (" joins the Zombies\n");

        return;
}

// tricks and traps
if (attacker.classname == "explo_box")
{
        bprint (" blew up\n");
        return;
}
if (attacker.solid == SOLID_BSP && attacker != world)
{
        bprint (" was squished\n");
        return;
}
if (attacker.classname == "trap_shooter" || attacker.classname == "trap_spikeshooter")
{
        bprint (" was spiked\n");
        return;
}
if (attacker.classname == "fireball")
{
        bprint (" ate a lavaball\n");
        return;
}
if (attacker.classname == "trigger_changelevel")
{
        bprint (" tried to leave\n");
        return;
}

// in-water deaths
rnum = targ.watertype;
if (rnum == -3)
{
        if (random() < 0.5)
                bprint (" sleeps with the fishes\n");
```

```
                        else
                                bprint (" sucks it down\n");
                        return;
                }
                else if (rnum == -4)
                {
                        if (random() < 0.5)
                                bprint (" gulped a load of slime\n");
                        else
                                bprint (" can't exist on slime alone\n");
                        return;
                }
                else if (rnum == -5)
                {
                        if (targ.health < -15)
                        {
                                bprint (" burst into flames\n");
                                return;
                        }
                        if (random() < 0.5)
                                bprint (" turned into hot slag\n");
                        else
                                bprint (" visits the Volcano God\n");
                        return;
                }

                // fell to their death?
                if (targ.deathtype == "falling")
                {
                        targ.deathtype = "";
                        bprint (" fell to his death\n");
                        return;
                }

                // hell if I know; he's just dead!!!
                bprint (" died\n");
        }
    }
};
```

COMBAT.QC

```
void() T_MissileTouch;
void() info_player_start;
void(entity targ, entity attacker) ClientObituary;

void() monster_death_use;

//============================================================================

/*
============
CanDamage

Returns true if the inflictor can directly damage the target.  Used for
explosions and melee attacks.
============
*/
float(entity targ, entity inflictor) CanDamage =
{
// bmodels need special checking because their origin is 0,0,0
	if (targ.movetype == MOVETYPE_PUSH)
	{
		traceline(inflictor.origin, 0.5 * (targ.absmin + targ.absmax), TRUE, self);
		if (trace_fraction == 1)
			return TRUE;
		if (trace_ent == targ)
			return TRUE;
		return FALSE;
	}

	traceline(inflictor.origin, targ.origin, TRUE, self);
	if (trace_fraction == 1)
		return TRUE;
	traceline(inflictor.origin, targ.origin + '15 15 0', TRUE, self);
	if (trace_fraction == 1)
		return TRUE;
	traceline(inflictor.origin, targ.origin + '-15 -15 0', TRUE, self);
	if (trace_fraction == 1)
		return TRUE;
	traceline(inflictor.origin, targ.origin + '-15 15 0', TRUE, self);
	if (trace_fraction == 1)
		return TRUE;
	traceline(inflictor.origin, targ.origin + '15 -15 0', TRUE, self);
	if (trace_fraction == 1)
		return TRUE;

	return FALSE;
};


/*
============
Killed
============
*/
void(entity targ, entity attacker) Killed =
{
	local entity oself;

	oself = self;
```

```
	self = targ;

	if (self.health < -99)
		self.health = -99;			// don't let sbar look bad if a player

	if (self.movetype == MOVETYPE_PUSH || self.movetype == MOVETYPE_NONE)
	{	// doors, triggers, etc
		self.th_die ();
		self = oself;
		return;
	}

	self.enemy = attacker;

// bump the monster counter
	if (self.flags & FL_MONSTER)
	{
		killed_monsters = killed_monsters + 1;
		WriteByte (MSG_ALL, SVC_KILLEDMONSTER);
	}

	ClientObituary(self, attacker);

	self.takedamage = DAMAGE_NO;
	self.touch = SUB_Null;

	monster_death_use();
	self.th_die ();

	self = oself;
};


/*
============
T_Damage

The damage is coming from inflictor, but get mad at attacker
This should be the only function that ever reduces health.
============
*/
void(entity targ, entity inflictor, entity attacker, float damage) T_Damage=
{
	local	vector	dir;
	local	entity	oldself;
	local	float	save;
	local	float	take;

	if (!targ.takedamage)
		return;

// used by buttons and triggers to set activator for target firing
	damage_attacker = attacker;

// check for quad damage powerup on the attacker
	if (attacker.super_damage_finished > time)
		damage = damage * 4;

// save damage based on the target's armor level

	save = ceil(targ.armortype*damage);
	if (save >= targ.armorvalue)
```

```
	{
		save = targ.armorvalue;
		targ.armortype = 0;	// lost all armor
		targ.items = targ.items - (targ.items & (IT_ARMOR1 | IT_ARMOR2 | IT_ARMOR3));
	}

	targ.armorvalue = targ.armorvalue - save;
	take = ceil(damage-save);

// add to the damage total for clients, which will be sent as a single
// message at the end of the frame
// FIXME: remove after combining shotgun blasts?
	if (targ.flags & FL_CLIENT)
	{
		targ.dmg_take = targ.dmg_take + take;
		targ.dmg_save = targ.dmg_save + save;
		targ.dmg_inflictor = inflictor;
	}

// figure momentum add
	if ( (inflictor != world) && (targ.movetype == MOVETYPE_WALK) )
	{
		dir = targ.origin - (inflictor.absmin + inflictor.absmax) * 0.5;
		dir = normalize(dir);
		targ.velocity = targ.velocity + dir*damage*8;
	}

// check for godmode or invincibility
	if (targ.flags & FL_GODMODE)
		return;
	if (targ.invincible_finished >= time)
	{
		if (self.invincible_sound < time)
		{
			sound (targ, CHAN_ITEM, "items/protect3.wav", 1, ATTN_NORM);
			self.invincible_sound = time + 2;
		}
		return;
	}

// team play damage avoidance
	if ( (teamplay == 1) && (targ.team > 0)&&(targ.team == attacker.team) )
		return;

// do the damage
	targ.health = targ.health - take;

	if (targ.health <= 0)
	{
		Killed (targ, attacker);
		return;
	}

// react to the damage
	oldself = self;
	self = targ;

	if ( (self.flags & FL_MONSTER) && attacker != world)
	{
	// get mad unless of the same class (except for soldiers)
		if (self != attacker && attacker != self.enemy)
		{
```

```
                        if ( (self.classname != attacker.classname)
                        || (self.classname == "monster_army" ) )
                        {
                                if (self.enemy.classname == "player")
                                        self.oldenemy = self.enemy;
                                self.enemy = attacker;
                                FoundTarget ();
                        }
                }
        }

        if (self.th_pain)
        {
                self.th_pain (attacker, take);
        // nightmare mode monsters don't go into pain frames often
                if (skill == 3)
                        self.pain_finished = time + 5;
        }

        self = oldself;
};

/*
============
T_RadiusDamage
============
*/
void(entity inflictor, entity attacker, float damage, entity ignore) T_RadiusDamage =
{
        local   float   points;
        local   entity  head;
        local   vector  org;

        head = findradius(inflictor.origin, damage+40);

        while (head)
        {
                if (head != ignore)
                {
                        if (head.takedamage)
                        {
                                org = head.origin + (head.mins + head.maxs)*0.5;
                                points = 0.5*vlen (inflictor.origin - org);
                                if (points < 0)
                                        points = 0;
                                points = damage - points;
                                if (head == attacker)
                                        points = points * 0.5;
                                if (points > 0)
                                {
                                        if (CanDamage (head, inflictor))
                                        {       // shambler takes half damage from all explosions
                                                if (head.classname == "monster_shambler")

                                                        T_Damage (head, inflictor, attacker, points*0.5);
                                                else
                                                        T_Damage (head, inflictor, attacker, points);
                                        }
                                }
                        }
                }
                head = head.chain;
```

```
	}
};

/*
============
T_BeamDamage
============
*/
void(entity attacker, float damage) T_BeamDamage =
{
	local	float	points;
	local	entity	head;

	head = findradius(attacker.origin, damage+40);

	while (head)
	{
		if (head.takedamage)
		{
			points = 0.5*vlen (attacker.origin - head.origin);
			if (points < 0)
				points = 0;
			points = damage - points;
			if (head == attacker)
				points = points * 0.5;
			if (points > 0)
			{
				if (CanDamage (head, attacker))
				{
					if (head.classname == "monster_shambler")

						T_Damage (head, attacker, attacker, points*0.5);
					else
						T_Damage (head, attacker, attacker, points);
				}
			}
		}
		head = head.chain;
	}
};
```

DEFS.QC

```
/*
==============================================================================

                        SOURCE FOR GLOBALVARS_T C STRUCTURE

==============================================================================
*/

//
// system globals
//
entity          self;
entity          other;
entity          world;
float           time;
float           frametime;

float           force_retouch;          // force all entities to touch triggers
                                                                // next frame.  this is needed because
                                                                // non-moving things don't normally scan
                                                                // for triggers, and when a trigger is
                                                                // created (like a teleport trigger), it
                                                                // needs to catch everything.
                                                                // decremented each frame, so set to 2
                                                                // to guarantee everything is touched
string          mapname;

float           deathmatch;
float           coop;
float           teamplay;

float           serverflags;            // propagated from level to level, used to
                                                                // keep track of completed episodes

float           total_secrets;
float           total_monsters;

float           found_secrets;          // number of secrets found
float           killed_monsters;        // number of monsters killed


// spawnparms are used to encode information about clients across server
// level changes
float           parm1, parm2, parm3, parm4, parm5, parm6, parm7, parm8, parm9, parm10, parm11, parm12, parm13,
parm14, parm15, parm16;

//
// global variables set by built in functions
//
vector          v_forward, v_up, v_right;       // set by makevectors()

// set by traceline / tracebox
float           trace_allsolid;
float           trace_startsolid;
float           trace_fraction;
vector          trace_endpos;
vector          trace_plane_normal;
float           trace_plane_dist;
entity          trace_ent;
```

```
float           trace_inopen;
float           trace_inwater;

entity          msg_entity;                     // destination of single entity writes

//
// required prog functions
//
void()          main;                                   // only for testing

void()          StartFrame;

void()          PlayerPreThink;
void()          PlayerPostThink;

void()          ClientKill;
void()          ClientConnect;
void()          PutClientInServer;      // call after setting the parm1... parms
void()          ClientDisconnect;

void()          SetNewParms;            // called when a client first connects to
                                                        // a server. sets parms so they can be
                                                        // saved off for restarts

void()          SetChangeParms;                 // call to set parms for self so they can
                                                        // be saved for a level transition


//============================================
void            end_sys_globals;        // flag for structure dumping
//============================================

/*
================================================================================

                        SOURCE FOR ENTVARS_T C STRUCTURE

================================================================================
*/

//
// system fields (*** = do not set in prog code, maintained by C code)
//
.float          modelindex;             // *** model index in the precached list
.vector         absmin, absmax;         // *** origin + mins / maxs

.float          ltime;                  // local time for entity
.float          movetype;
.float          solid;

.vector         origin;                 // ***
.vector         oldorigin;              // ***
.vector         velocity;
.vector         angles;
.vector         avelocity;

.vector         punchangle;             // temp angle adjust from damage or recoil

.string         classname;              // spawn function
.string         model;
.float          frame;
.float          skin;
```

```
.float          effects;

.vector         mins, maxs;             // bounding box extents reletive to origin
.vector         size;                   // maxs - mins

.void()         touch;
.void()         use;
.void()         think;
.void()         blocked;                // for doors or plats, called when can't push other

.float          nextthink;
.entity         groundentity;

// stats
.float          health;
.float          frags;
.float          weapon;                         // one of the IT_SHOTGUN, etc flags
.string         weaponmodel;
.float          weaponframe;
.float          currentammo;
.float          ammo_shells, ammo_nails, ammo_rockets, ammo_cells;

.float          items;                  // bit flags

.float          takedamage;
.entity         chain;
.float          deadflag;

.vector         view_ofs;                       // add to origin to get eye point


.float          button0;                // fire
.float          button1;                // use
.float          button2;                // jump

.float          impulse;                // weapon changes

.float          fixangle;
.vector         v_angle;                // view / targeting angle for players
.float          idealpitch;             // calculated pitch angle for lookup up slopes


.string         netname;

.entity    enemy;

.float          flags;

.float          colormap;
.float          team;

.float          max_health;             // players maximum health is stored here

.float          teleport_time;    // don't back up

.float          armortype;              // save this fraction of incoming damage
.float          armorvalue;

.float          waterlevel;             // 0 = not in, 1 = feet, 2 = wast, 3 = eyes
.float          watertype;              // a contents value

.float          ideal_yaw;
```

```
.float          yaw_speed;

.entity         aiment;

.entity   goalentity;           // a movetarget or an enemy

.float          spawnflags;

.string         target;
.string         targetname;

// damage is accumulated through a frame. and sent as one single
// message, so the super shotgun doesn't generate huge messages
.float          dmg_take;
.float          dmg_save;
.entity         dmg_inflictor;

.entity         owner;          // who launched a missile
.vector         movedir;        // mostly for doors, but also used for waterjump

.string         message;                // trigger messages

.float          sounds;         // either a cd track number or sound number

.string         noise, noise1, noise2, noise3;    // contains names of wavs to play

//================================================
void            end_sys_fields;                 // flag for structure dumping
//================================================

/*
==============================================================================

                        VARS NOT REFERENCED BY C CODE

==============================================================================
*/


//
// constants
//

float   FALSE                           = 0;
float   TRUE                            = 1;

// edict.flags
float   FL_FLY                          = 1;
float   FL_SWIM                                 = 2;
float   FL_CLIENT                       = 8;    // set for all client edicts
float   FL_INWATER                      = 16;   // for enter / leave water splash
float   FL_MONSTER                      = 32;
float   FL_GODMODE                      = 64;   // player cheat
float   FL_NOTARGET                             = 128;  // player cheat
float   FL_ITEM                                 = 256;  // extra wide size for bonus items
float   FL_ONGROUND                             = 512;  // standing on something
float   FL_PARTIALGROUND        = 1024; // not all corners are valid
float   FL_WATERJUMP                    = 2048; // player jumping out of water
float   FL_JUMPRELEASED                 = 4096; // for jump debouncing

// edict.movetype values
float   MOVETYPE_NONE                   = 0;    // never moves
```

```
//float    MOVETYPE_ANGLENOCLIP    = 1;
//float    MOVETYPE_ANGLECLIP              = 2;
float    MOVETYPE_WALK                   = 3;    // players only
float    MOVETYPE_STEP                   = 4;    // discrete, not real time unless fall
float    MOVETYPE_FLY                    = 5;
float    MOVETYPE_TOSS                   = 6;    // gravity
float    MOVETYPE_PUSH                   = 7;    // no clip to world, push and crush
float    MOVETYPE_NOCLIP                 = 8;
float    MOVETYPE_FLYMISSILE             = 9;    // fly with extra size against monsters
float    MOVETYPE_BOUNCE                 = 10;
float    MOVETYPE_BOUNCEMISSILE = 11;    // bounce with extra size

// edict.solid values
float    SOLID_NOT                       = 0;    // no interaction with other objects
float    SOLID_TRIGGER                   = 1;    // touch on edge, but not blocking
float    SOLID_BBOX                      = 2;    // touch on edge, block
float    SOLID_SLIDEBOX                  = 3;    // touch on edge, but not an onground
float    SOLID_BSP                       = 4;    // bsp clip, touch on edge, block

// range values
float    RANGE_MELEE                             = 0;
float    RANGE_NEAR                      = 1;
float    RANGE_MID                       = 2;
float    RANGE_FAR                       = 3;

// deadflag values

float    DEAD_NO                                 = 0;
float    DEAD_DYING                      = 1;
float    DEAD_DEAD                       = 2;
float    DEAD_RESPAWNABLE                = 3;

// takedamage values

float    DAMAGE_NO                       = 0;
float    DAMAGE_YES                      = 1;
float    DAMAGE_AIM                      = 2;

// items
float    IT_AXE                          = 4096;
float    IT_SHOTGUN                      = 1;
float    IT_SUPER_SHOTGUN        = 2;
float    IT_NAILGUN                      = 4;
float    IT_SUPER_NAILGUN        = 8;
float    IT_GRENADE_LAUNCHER             = 16;
float    IT_ROCKET_LAUNCHER              = 32;
float    IT_LIGHTNING            = 64;
float    IT_EXTRA_WEAPON                 = 128;

float    IT_SHELLS                       = 256;
float    IT_NAILS                        = 512;
float    IT_ROCKETS                      = 1024;
float    IT_CELLS                        = 2048;

float    IT_ARMOR1                       = 8192;
float    IT_ARMOR2                       = 16384;
float    IT_ARMOR3                       = 32768;
float    IT_SUPERHEALTH                  = 65536;

float    IT_KEY1                                 = 131072;
float    IT_KEY2                                 = 262144;
```

```
float	IT_INVISIBILITY			= 524288;
float	IT_INVULNERABILITY		= 1048576;
float	IT_SUIT					= 2097152;
float	IT_QUAD					= 4194304;

// point content values

float	CONTENT_EMPTY			= -1;
float	CONTENT_SOLID			= -2;
float	CONTENT_WATER			= -3;
float	CONTENT_SLIME			= -4;
float	CONTENT_LAVA			= -5;
float	CONTENT_SKY				= -6;

float	STATE_TOP		= 0;
float	STATE_BOTTOM	= 1;
float	STATE_UP		= 2;
float	STATE_DOWN		= 3;

vector	VEC_ORIGIN = '0 0 0';
vector	VEC_HULL_MIN = '-16 -16 -24';
vector	VEC_HULL_MAX = '16 16 32';

vector	VEC_HULL2_MIN = '-32 -32 -24';
vector	VEC_HULL2_MAX = '32 32 64';

// protocol bytes
float	SVC_TEMPENTITY		= 23;
float	SVC_KILLEDMONSTER	= 27;
float	SVC_FOUNDSECRET		= 28;
float	SVC_INTERMISSION	= 30;
float	SVC_FINALE			= 31;
float	SVC_CDTRACK			= 32;
float	SVC_SELLSCREEN		= 33;


float	TE_SPIKE		= 0;
float	TE_SUPERSPIKE	= 1;
float	TE_GUNSHOT		= 2;
float	TE_EXPLOSION	= 3;
float	TE_TAREXPLOSION	= 4;
float	TE_LIGHTNING1	= 5;
float	TE_LIGHTNING2	= 6;
float	TE_WIZSPIKE		= 7;
float	TE_KNIGHTSPIKE	= 8;
float	TE_LIGHTNING3	= 9;
float	TE_LAVASPLASH	= 10;
float	TE_TELEPORT		= 11;

// sound channels
// channel 0 never willingly overrides
// other channels (1-7) allways override a playing sound on that channel
float	CHAN_AUTO		= 0;
float	CHAN_WEAPON		= 1;
float	CHAN_VOICE		= 2;
float	CHAN_ITEM		= 3;
float	CHAN_BODY		= 4;

float	ATTN_NONE		= 0;
float	ATTN_NORM		= 1;
float	ATTN_IDLE		= 2;
float	ATTN_STATIC		= 3;
```

```
// update types

float	UPDATE_GENERAL	= 0;
float	UPDATE_STATIC	= 1;
float	UPDATE_BINARY	= 2;
float	UPDATE_TEMP		= 3;

// entity effects

float	EF_BRIGHTFIELD	= 1;
float	EF_MUZZLEFLASH	= 2;
float	EF_BRIGHTLIGHT	= 4;
float	EF_DIMLIGHT  = 8;


// messages
float	MSG_BROADCAST	= 0;		// unreliable to all
float	MSG_ONE			= 1;		// reliable to one (msg_entity)
float	MSG_ALL			= 2;		// reliable to all
float	MSG_INIT		= 3;		// write to the init string

//============================================

//
// globals
//
float	movedist;
float	gameover;		// set when a rule exits

string	string_null;	// null string, nothing should be held here
float	empty_float;

entity	newmis;			// launch_spike sets this after spawning it

entity	activator;		// the entity that activated a trigger or brush

entity	damage_attacker;	// set by T_Damage
float	framecount;

float		skill;

//============================================

//
// world fields (FIXME: make globals)
//
.string		wad;
.string	map;
.float		worldtype;	// 0=medieval 1=metal 2=base

//============================================

.string		killtarget;

//
// quakeed fields
//
.float		light_lev;		// not used by game, but parsed by light util
.float		style;
```

```
//
// monster ai
//
.void()             th_stand;
.void()             th_walk;
.void()             th_run;
.void()             th_missile;
.void()             th_melee;
.void(entity attacker, float damage)                th_pain;
.void()             th_die;

.entity             oldenemy;               // mad at this player before taking damage

.float              speed;

.float      lefty;

.float      search_time;
.float      attack_state;

float       AS_STRAIGHT         = 1;
float       AS_SLIDING          = 2;
float       AS_MELEE            = 3;
float       AS_MISSILE          = 4;

//
// player only fields
//
.float              walkframe;

.float              attack_finished;
.float              pain_finished;

.float              invincible_finished;
.float              invisible_finished;
.float              super_damage_finished;
.float              radsuit_finished;

.float              invincible_time, invincible_sound;
.float              invisible_time, invisible_sound;
.float              super_time, super_sound;
.float              rad_time;
.float              fly_sound;

.float              axhitme;

.float              show_hostile;   // set to time+0.2 whenever a client fires a
                                                        // weapon or takes damage.  Used to alert
                                                        // monsters that otherwise would let the player go
.float              jump_flag;              // player jump flag
.float              swim_flag;              // player swimming sound flag
.float              air_finished;       // when time > air_finished, start drowning
.float              bubble_count;   // keeps track of the number of bubbles
.string             deathtype;              // keeps track of how the player died

//
// object stuff
//
.string             mdl;
.vector             mangle;                 // angle at start

.vector             oldorigin;              // only used by secret door
```

```
.float          t_length, t_width;


//
// doors, etc
//
.vector         dest, dest1, dest2;
.float          wait;                   // time from firing to restarting
.float          delay;                  // time from activation to firing
.entity         trigger_field;      // door's trigger entity
.string         noise4;

//
// monsters
//
.float          pausetime;
.entity    movetarget;


//
// doors
//
.float          aflag;
.float          dmg;                    // damage done by door when hit

//
// misc
//
.float          cnt;                    // misc flag

//
// subs
//
.void()         think1;
.vector         finaldest, finalangle;

//
// triggers
//
.float          count;                  // for counting triggers


//
// plats / doors / buttons
//
.float          lip;
.float          state;
.vector         pos1, pos2;             // top and bottom positions
.float          height;

//
// sounds
//
.float          waitmin, waitmax;
.float          distance;
.float          volume;



//============================================================================
```

```
//
// builtin functions
//

void(vector ang)makevectors             = #1;           // sets v_forward, etc globals
void(entity e, vector o) setorigin  = #2;
void(entity e, string m) setmodel = #3;             // set movetype and solid first
void(entity e, vector min, vector max) setsize = #4;
// #5 was removed
void() break                                            = #6;
float() random                                          = #7;           // returns 0 - 1
void(entity e, float chan, string samp, float vol, float atten) sound = #8;
vector(vector v) normalize                      = #9;
void(string e) error                                    = #10;
void(string e) objerror                         = #11;
float(vector v) vlen                                    = #12;
float(vector v) vectoyaw                = #13;
entity() spawn                                          = #14;
void(entity e) remove                           = #15;

// sets trace_* globals
// nomonsters can be:
// An entity will also be ignored for testing if forent == test,
// forent->owner == test, or test->owner == forent
// a forent of world is ignored
void(vector v1, vector v2, float nomonsters, entity forent) traceline = #16;

entity() checkclient                            = #17;  // returns a client to look for
entity(entity start, .string fld, string match) find = #18;
string(string s) precache_sound         = #19;
string(string s) precache_model         = #20;
void(entity client, string s)stuffcmd = #21;
entity(vector org, float rad) findradius = #22;
void(string s) bprint                           = #23;
void(entity client, string s) sprint = #24;
void(string s) dprint                           = #25;
string(float f) ftos                            = #26;
string(vector v) vtos                           = #27;
void() coredump                                         = #28;          // prints all edicts
void() traceon                                  = #29;          // turns statment trace on
void() traceoff                                 = #30;
void(entity e) eprint                           = #31;          // prints an entire edict
float(float yaw, float dist) walkmove   = #32;  // returns TRUE or FALSE
// #33 was removed
float(float yaw, float dist) droptofloor= #34;          // TRUE if landed on floor
void(float style, string value) lightstyle = #35;
float(float v) rint                             = #36;          // round to nearest int
float(float v) floor                            = #37;          // largest integer <= v
float(float v) ceil                             = #38;          // smallest integer >= v
// #39 was removed
float(entity e) checkbottom                     = #40;          // true if self is on ground
float(vector v) pointcontents           = #41;          // returns a CONTENT_*
// #42 was removed
float(float f) fabs = #43;
vector(entity e, float speed) aim = #44;            // returns the shooting vector
float(string s) cvar = #45;                                     // return cvar.value
void(string s) localcmd = #46;                          // put string into local que
entity(entity e) nextent = #47;                         // for looping through all ents
void(vector o, vector d, float color, float count) particle = #48;// start a particle effect
void() ChangeYaw = #49;                                         // turn towards self.ideal_yaw
```

```
                                                                    // at self.yaw_speed
// #50 was removed
vector(vector v) vectoangles                = #51;

//
// direct client message generation
//
void(float to, float f) WriteByte           = #52;
void(float to, float f) WriteChar           = #53;
void(float to, float f) WriteShort          = #54;
void(float to, float f) WriteLong           = #55;
void(float to, float f) WriteCoord          = #56;
void(float to, float f) WriteAngle          = #57;
void(float to, string s) WriteString        = #58;
void(float to, entity s) WriteEntity= #59;

//
// broadcast client message generation
//

// void(float f) bWriteByte                 = #59;
// void(float f) bWriteChar                 = #60;
// void(float f) bWriteShort                = #61;
// void(float f) bWriteLong                 = #62;
// void(float f) bWriteCoord                = #63;
// void(float f) bWriteAngle                = #64;
// void(string s) bWriteString      = #65;
// void(entity e) bWriteEntity = #66;

void(float step) movetogoal                         = #67;

string(string s) precache_file              = #68;  // no effect except for -copy
void(entity e) makestatic           = #69;
void(string s) changelevel = #70;

//#71 was removed

void(string var, string val) cvar_set = #72;        // sets cvar.value

void(entity client, string s) centerprint = #73;        // sprint, but in middle

void(vector pos, string samp, float vol, float atten) ambientsound = #74;

string(string s) precache_model2            = #75;          // registered version only
string(string s) precache_sound2            = #76;          // registered version only
string(string s) precache_file2             = #77;          // registered version only

void(entity e) setspawnparms                = #78;          // set parm1... to the
                                                                    // values at level start
                                                                    // for coop respawn

//============================================================================

//
// subs.qc
//
void(vector tdest, float tspeed, void() func) SUB_CalcMove;
void(entity ent, vector tdest, float tspeed, void() func) SUB_CalcMoveEnt;
void(vector destangle, float tspeed, void() func) SUB_CalcAngleMove;
void()  SUB_CalcMoveDone;
void() SUB_CalcAngleMoveDone;
void() SUB_Null;
```

```
void() SUB_UseTargets;
void() SUB_Remove;

//
//          combat.qc
//
void(entity targ, entity inflictor, entity attacker, float damage) T_Damage;


float (entity e, float healamount, float ignore) T_Heal; // health function

float(entity targ, entity inflictor) CanDamage;
```

DOORS.QC

```
float DOOR_START_OPEN = 1;
float DOOR_DONT_LINK = 4;
float DOOR_GOLD_KEY = 8;
float DOOR_SILVER_KEY = 16;
float DOOR_TOGGLE = 32;

/*

Doors are similar to buttons, but can spawn a fat trigger field around them
to open without a touch, and they link together to form simultanious
double/quad doors.

Door.owner is the master door.  If there is only one door, it points to itself.
If multiple doors, all will point to a single one.

Door.enemy chains from the master door through all doors linked in the chain.

*/

/*
============================================================================

THINK FUNCTIONS

============================================================================
*/

void() door_go_down;
void() door_go_up;

void() door_blocked =
{
	T_Damage (other, self, self, self.dmg);

// if a door has a negative wait, it would never come back if blocked,
// so let it just squash the object to death real fast
	if (self.wait >= 0)
	{
		if (self.state == STATE_DOWN)
			door_go_up ();
		else
			door_go_down ();
	}
};


void() door_hit_top =
{
	sound (self, CHAN_VOICE, self.noise1, 1, ATTN_NORM);
	self.state = STATE_TOP;
	if (self.spawnflags & DOOR_TOGGLE)
		return;		// don't come down automatically
	self.think = door_go_down;
	self.nextthink = self.ltime + self.wait;
};

void() door_hit_bottom =
{
	sound (self, CHAN_VOICE, self.noise1, 1, ATTN_NORM);
	self.state = STATE_BOTTOM;
```

```
};

void() door_go_down =
{
	sound (self, CHAN_VOICE, self.noise2, 1, ATTN_NORM);
	if (self.max_health)
	{
		self.takedamage = DAMAGE_YES;
		self.health = self.max_health;
	}

	self.state = STATE_DOWN;
	SUB_CalcMove (self.pos1, self.speed, door_hit_bottom);
};

void() door_go_up =
{
	if (self.state == STATE_UP)
		return;		// allready going up

	if (self.state == STATE_TOP)
	{	// reset top wait time
		self.nextthink = self.ltime + self.wait;
		return;
	}

	sound (self, CHAN_VOICE, self.noise2, 1, ATTN_NORM);
	self.state = STATE_UP;
	SUB_CalcMove (self.pos2, self.speed, door_hit_top);

	SUB_UseTargets();
};


/*
==============================================================================

ACTIVATION FUNCTIONS

==============================================================================
*/

void() door_fire =
{
	local entity	oself;
	local entity	starte;

	if (self.owner != self)
		objerror ("door_fire: self.owner != self");

// play use key sound

	if (self.items)
		sound (self, CHAN_VOICE, self.noise4, 1, ATTN_NORM);

	self.message = string_null;			// no more message
	oself = self;

	if (self.spawnflags & DOOR_TOGGLE)
	{
		if (self.state == STATE_UP || self.state == STATE_TOP)
		{
```

```
                        starte = self;
                        do
                        {
                                door_go_down ();
                                self = self.enemy;
                        } while ( (self != starte) && (self != world) );
                        self = oself;
                        return;
                }
        }

// trigger all paired doors
        starte = self;
        do
        {
                door_go_up ();
                self = self.enemy;
        } while ( (self != starte) && (self != world) );
        self = oself;
};


void() door_use =
{
        local entity oself;

        self.message = "";                      // door message are for touch only
        self.owner.message = "";
        self.enemy.message = "";
        oself = self;
        self = self.owner;
        door_fire ();
        self = oself;
};


void() door_trigger_touch =
{
        if (other.health <= 0)
                return;

        if (time < self.attack_finished)
                return;
        self.attack_finished = time + 1;

        activator = other;

        self = self.owner;
        door_use ();
};


void() door_killed =
{
        local entity oself;

        oself = self;
        self = self.owner;
        self.health = self.max_health;
        self.takedamage = DAMAGE_NO;        // wil be reset upon return
        door_use ();
        self = oself;
```

```
};


/*
================
door_touch

Prints messages and opens key doors
================
*/
void() door_touch =
{
	if (other.classname != "player")
		return;
	if (self.owner.attack_finished > time)
		return;

	self.owner.attack_finished = time + 2;

	if (self.owner.message != "")
	{
		centerprint (other, self.owner.message);
		sound (other, CHAN_VOICE, "misc/talk.wav", 1, ATTN_NORM);
	}

// key door stuff
	if (!self.items)
		return;

// FIXME: blink key on player's status bar
	if ( (self.items & other.items) != self.items )
	{
		if (self.owner.items == IT_KEY1)
		{
			if (world.worldtype == 2)
			{
				centerprint (other, "You need the silver keycard");
				sound (self, CHAN_VOICE, self.noise3, 1, ATTN_NORM);
			}
			else if (world.worldtype == 1)
			{
				centerprint (other, "You need the silver runekey");
				sound (self, CHAN_VOICE, self.noise3, 1, ATTN_NORM);
			}
			else if (world.worldtype == 0)
			{
				centerprint (other, "You need the silver key");
				sound (self, CHAN_VOICE, self.noise3, 1, ATTN_NORM);
			}
		}
		else
		{
			if (world.worldtype == 2)
			{
				centerprint (other, "You need the gold keycard");
				sound (self, CHAN_VOICE, self.noise3, 1, ATTN_NORM);
			}
			else if (world.worldtype == 1)
			{
				centerprint (other, "You need the gold runekey");
				sound (self, CHAN_VOICE, self.noise3, 1, ATTN_NORM);
			}
```

```
                        else if (world.worldtype == 0)
                        {
                                centerprint (other, "You need the gold key");
                                sound (self, CHAN_VOICE, self.noise3, 1, ATTN_NORM);
                        }
                }
                return;
        }

        other.items = other.items - self.items;
        self.touch = SUB_Null;
        if (self.enemy)
                self.enemy.touch = SUB_Null;    // get paired door
        door_use ();
};

/*
==============================================================================

SPAWNING FUNCTIONS

==============================================================================
*/


entity(vector fmins, vector fmaxs) spawn_field =
{
        local entity        trigger;
        local     vector    t1, t2;

        trigger = spawn();
        trigger.movetype = MOVETYPE_NONE;
        trigger.solid = SOLID_TRIGGER;
        trigger.owner = self;
        trigger.touch = door_trigger_touch;

        t1 = fmins;
        t2 = fmaxs;
        setsize (trigger, t1 - '60 60 8', t2 + '60 60 8');
        return (trigger);
};


float (entity e1, entity e2) EntitiesTouching =
{
        if (e1.mins_x > e2.maxs_x)
                return FALSE;
        if (e1.mins_y > e2.maxs_y)
                return FALSE;
        if (e1.mins_z > e2.maxs_z)
                return FALSE;
        if (e1.maxs_x < e2.mins_x)
                return FALSE;
        if (e1.maxs_y < e2.mins_y)
                return FALSE;
        if (e1.maxs_z < e2.mins_z)
                return FALSE;
        return TRUE;
};


/*
```

```
=============
LinkDoors


=============
*/
void() LinkDoors =
{
	local entity	t, starte;
	local vector	cmins, cmaxs;

	if (self.enemy)
		return;		// already linked by another door
	if (self.spawnflags & 4)
	{
		self.owner = self.enemy = self;
		return;		// don't want to link this door
	}

	cmins = self.mins;
	cmaxs = self.maxs;

	starte = self;
	t = self;

	do
	{
		self.owner = starte;				// master door

		if (self.health)
			starte.health = self.health;
		if (self.targetname)
			starte.targetname = self.targetname;
		if (self.message != "")
			starte.message = self.message;

		t = find (t, classname, self.classname);
		if (!t)
		{
			self.enemy = starte;			// make the chain a loop

		// shootable, fired, or key doors just needed the owner/enemy links,
		// they don't spawn a field

			self = self.owner;

			if (self.health)
				return;
			if (self.targetname)
				return;
			if (self.items)
				return;

			self.owner.trigger_field = spawn_field(cmins, cmaxs);

			return;
		}

		if (EntitiesTouching(self,t))
		{
			if (t.enemy)
				objerror ("cross connected doors");
```

```
                        self.enemy = t;
                        self = t;

                        if (t.mins_x < cmins_x)
                                cmins_x = t.mins_x;
                        if (t.mins_y < cmins_y)
                                cmins_y = t.mins_y;
                        if (t.mins_z < cmins_z)
                                cmins_z = t.mins_z;
                        if (t.maxs_x > cmaxs_x)
                                cmaxs_x = t.maxs_x;
                        if (t.maxs_y > cmaxs_y)
                                cmaxs_y = t.maxs_y;
                        if (t.maxs_z > cmaxs_z)
                                cmaxs_z = t.maxs_z;
                }
        } while (1 );

};


/*QUAKED func_door (0 .5 .8) ? START_OPEN x DOOR_DONT_LINK GOLD_KEY SILVER_KEY TOGGLE
if two doors touch, they are assumed to be connected and operate as a unit.

TOGGLE causes the door to wait in both the start and end states for a trigger event.

START_OPEN causes the door to move to its destination when spawned, and operate in reverse.  It is used to
temporarily or permanently close off an area when triggered (not usefull for touch or takedamage doors).

Key doors are allways wait -1.

"message"       is printed when the door is touched if it is a trigger door and it hasn't been fired yet
"angle"         determines the opening direction
"targetname" if set, no touch field will be spawned and a remote button or trigger field activates the door.
"health" if set, door must be shot open
"speed"         movement speed (100 default)
"wait"          wait before returning (3 default, -1 = never return)
"lip"           lip remaining at end of move (8 default)
"dmg"           damage to inflict when blocked (2 default)
"sounds"
0)      no sound
1)      stone
2)      base
3)      stone chain
4)      screechy metal
*/

void() func_door =

{

        if (world.worldtype == 0)
        {
                precache_sound ("doors/medtry.wav");
                precache_sound ("doors/meduse.wav");
                self.noise3 = "doors/medtry.wav";
                self.noise4 = "doors/meduse.wav";
        }
        else if (world.worldtype == 1)
        {
                precache_sound ("doors/runetry.wav");
```

```
		precache_sound ("doors/runeuse.wav");
		self.noise3 = "doors/runetry.wav";
		self.noise4 = "doors/runeuse.wav";
}
else if (world.worldtype == 2)
{
		precache_sound ("doors/basetry.wav");
		precache_sound ("doors/baseuse.wav");
		self.noise3 = "doors/basetry.wav";
		self.noise4 = "doors/baseuse.wav";
}
else
{
		dprint ("no worldtype set!\n");
}
if (self.sounds == 0)
{
		precache_sound ("misc/null.wav");
		precache_sound ("misc/null.wav");
		self.noise1 = "misc/null.wav";
		self.noise2 = "misc/null.wav";
}
if (self.sounds == 1)
{
		precache_sound ("doors/drclos4.wav");
		precache_sound ("doors/doormv1.wav");
		self.noise1 = "doors/drclos4.wav";
		self.noise2 = "doors/doormv1.wav";
}
if (self.sounds == 2)
{
		precache_sound ("doors/hydro1.wav");
		precache_sound ("doors/hydro2.wav");
		self.noise2 = "doors/hydro1.wav";
		self.noise1 = "doors/hydro2.wav";
}
if (self.sounds == 3)
{
		precache_sound ("doors/stndr1.wav");
		precache_sound ("doors/stndr2.wav");
		self.noise2 = "doors/stndr1.wav";
		self.noise1 = "doors/stndr2.wav";
}
if (self.sounds == 4)
{
		precache_sound ("doors/ddoor1.wav");
		precache_sound ("doors/ddoor2.wav");
		self.noise1 = "doors/ddoor2.wav";
		self.noise2 = "doors/ddoor1.wav";
}


SetMovedir ();

self.max_health = self.health;
self.solid = SOLID_BSP;
self.movetype = MOVETYPE_PUSH;
setorigin (self, self.origin);
setmodel (self, self.model);
self.classname = "door";

self.blocked = door_blocked;
```

```
	self.use = door_use;

	if (self.spawnflags & DOOR_SILVER_KEY)
		self.items = IT_KEY1;
	if (self.spawnflags & DOOR_GOLD_KEY)
		self.items = IT_KEY2;

	if (!self.speed)
		self.speed = 100;
	if (!self.wait)
		self.wait = 3;
	if (!self.lip)
		self.lip = 8;
	if (!self.dmg)
		self.dmg = 2;

	self.pos1 = self.origin;
	self.pos2 = self.pos1 + self.movedir*(fabs(self.movedir*self.size) - self.lip);

// DOOR_START_OPEN is to allow an entity to be lighted in the closed position
// but spawn in the open position
	if (self.spawnflags & DOOR_START_OPEN)
	{
		setorigin (self, self.pos2);
		self.pos2 = self.pos1;
		self.pos1 = self.origin;
	}

	self.state = STATE_BOTTOM;

	if (self.health)
	{
		self.takedamage = DAMAGE_YES;
		self.th_die = door_killed;
	}

	if (self.items)
		self.wait = -1;

	self.touch = door_touch;

// LinkDoors can't be done until all of the doors have been spawned, so
// the sizes can be detected properly.
	self.think = LinkDoors;
	self.nextthink = self.ltime + 0.1;
};

/*
================================================================================

SECRET DOORS

================================================================================
*/

void() fd_secret_move1;
void() fd_secret_move2;
void() fd_secret_move3;
void() fd_secret_move4;
void() fd_secret_move5;
void() fd_secret_move6;
void() fd_secret_done;
```

```
float SECRET_OPEN_ONCE = 1;                     // stays open
float SECRET_1ST_LEFT = 2;              // 1st move is left of arrow
float SECRET_1ST_DOWN = 4;              // 1st move is down from arrow
float SECRET_NO_SHOOT = 8;              // only opened by trigger
float SECRET_YES_SHOOT = 16;            // shootable even if targeted


void () fd_secret_use =
{
        local float temp;

        self.health = 10000;

        // exit if still moving around...
        if (self.origin != self.oldorigin)
                return;

        self.message = string_null;                     // no more message

        SUB_UseTargets();                                       // fire all targets / killtargets

        if (!(self.spawnflags & SECRET_NO_SHOOT))
        {
                self.th_pain = SUB_Null;
                self.takedamage = DAMAGE_NO;
        }
        self.velocity = '0 0 0';

        // Make a sound, wait a little...

        sound(self, CHAN_VOICE, self.noise1, 1, ATTN_NORM);
        self.nextthink = self.ltime + 0.1;

        temp = 1 - (self.spawnflags & SECRET_1ST_LEFT);       // 1 or -1
        makevectors(self.mangle);

        if (!self.t_width)
        {
                if (self.spawnflags & SECRET_1ST_DOWN)
                        self. t_width = fabs(v_up * self.size);
                else
                        self. t_width = fabs(v_right * self.size);
        }

        if (!self.t_length)
                self. t_length = fabs(v_forward * self.size);

        if (self.spawnflags & SECRET_1ST_DOWN)
                self.dest1 = self.origin - v_up * self.t_width;
        else
                self.dest1 = self.origin + v_right * (self.t_width * temp);

        self.dest2 = self.dest1 + v_forward * self.t_length;
        SUB_CalcMove(self.dest1, self.speed, fd_secret_move1);
        sound(self, CHAN_VOICE, self.noise2, 1, ATTN_NORM);
};

// Wait after first movement...
void () fd_secret_move1 =
{
        self.nextthink = self.ltime + 1.0;
```

```
	self.think = fd_secret_move2;
	sound(self, CHAN_VOICE, self.noise3, 1, ATTN_NORM);
};

// Start moving sideways w/sound...
void () fd_secret_move2 =
{
	sound(self, CHAN_VOICE, self.noise2, 1, ATTN_NORM);
	SUB_CalcMove(self.dest2, self.speed, fd_secret_move3);
};

// Wait here until time to go back...
void () fd_secret_move3 =
{
	sound(self, CHAN_VOICE, self.noise3, 1, ATTN_NORM);
	if (!(self.spawnflags & SECRET_OPEN_ONCE))
	{
		self.nextthink = self.ltime + self.wait;
		self.think = fd_secret_move4;
	}
};

// Move backward...
void () fd_secret_move4 =
{
	sound(self, CHAN_VOICE, self.noise2, 1, ATTN_NORM);
	SUB_CalcMove(self.dest1, self.speed, fd_secret_move5);
};

// Wait 1 second...
void () fd_secret_move5 =
{
	self.nextthink = self.ltime + 1.0;
	self.think = fd_secret_move6;
	sound(self, CHAN_VOICE, self.noise3, 1, ATTN_NORM);
};

void () fd_secret_move6 =
{
	sound(self, CHAN_VOICE, self.noise2, 1, ATTN_NORM);
	SUB_CalcMove(self.oldorigin, self.speed, fd_secret_done);
};

void () fd_secret_done =
{
	if (!self.targetname || self.spawnflags&SECRET_YES_SHOOT)
	{
		self.health = 10000;
		self.takedamage = DAMAGE_YES;
		self.th_pain = fd_secret_use;
	}
	sound(self, CHAN_VOICE, self.noise3, 1, ATTN_NORM);
};

void () secret_blocked =
{
	if (time < self.attack_finished)
		return;
	self.attack_finished = time + 0.5;
	T_Damage (other, self, self, self.dmg);
};
```

```
/*
================
secret_touch

Prints messages
================
*/
void() secret_touch =
{
	if (other.classname != "player")
		return;
	if (self.attack_finished > time)
		return;

	self.attack_finished = time + 2;

	if (self.message)
	{
		centerprint (other, self.message);
		sound (other, CHAN_BODY, "misc/talk.wav", 1, ATTN_NORM);
	}
};


/*QUAKED func_door_secret (0 .5 .8) ? open_once 1st_left 1st_down no_shoot always_shoot
Basic secret door. Slides back, then to the side. Angle determines direction.
wait  = # of seconds before coming back
1st_left = 1st move is left of arrow
1st_down = 1st move is down from arrow
always_shoot = even if targeted, keep shootable
t_width = override WIDTH to move back (or height if going down)
t_length = override LENGTH to move sideways
"dmg"		damage to inflict when blocked (2 default)

If a secret door has a targetname, it will only be opened by it's botton or trigger, not by damage.
"sounds"
1) medieval
2) metal
3) base
*/

void () func_door_secret =
{
	if (self.sounds == 0)
		self.sounds = 3;
	if (self.sounds == 1)
	{
		precache_sound ("doors/latch2.wav");
		precache_sound ("doors/winch2.wav");
		precache_sound ("doors/drclos4.wav");
		self.noise1 = "doors/latch2.wav";
		self.noise2 = "doors/winch2.wav";
		self.noise3 = "doors/drclos4.wav";
	}
	if (self.sounds == 2)
	{
		precache_sound ("doors/airdoor1.wav");
		precache_sound ("doors/airdoor2.wav");
		self.noise2 = "doors/airdoor1.wav";
		self.noise1 = "doors/airdoor2.wav";
		self.noise3 = "doors/airdoor2.wav";
	}
```

```
	if (self.sounds == 3)
	{
		precache_sound ("doors/basesec1.wav");
		precache_sound ("doors/basesec2.wav");
		self.noise2 = "doors/basesec1.wav";
		self.noise1 = "doors/basesec2.wav";
		self.noise3 = "doors/basesec2.wav";
	}

	if (!self.dmg)
		self.dmg = 2;

	// Magic formula...
	self.mangle = self.angles;
	self.angles = '0 0 0';
	self.solid = SOLID_BSP;
	self.movetype = MOVETYPE_PUSH;
	self.classname = "door";
	setmodel (self, self.model);
	setorigin (self, self.origin);

	self.touch = secret_touch;
	self.blocked = secret_blocked;
	self.speed = 50;
	self.use = fd_secret_use;
	if ( !self.targetname || self.spawnflags&SECRET_YES_SHOOT)
	{
		self.health = 10000;
		self.takedamage = DAMAGE_YES;
		self.th_pain = fd_secret_use;
		self.th_die = fd_secret_use;
	}
	self.oldorigin = self.origin;
	if (!self.wait)
		self.wait = 5;			// 5 seconds before closing
};
```

FIGHT.QC

```
/*

A monster is in fight mode if it thinks it can effectively attack its
enemy.

When it decides it can't attack, it goes into hunt mode.

*/

float(float v) anglemod;

void() knight_atk1;
void() knight_runatk1;
void() ogre_smash1;
void() ogre_swing1;

void() sham_smash1;
void() sham_swingr1;
void() sham_swingl1;

float()    DemonCheckAttack;
void(float side)   Demon_Melee;

void(vector dest) ChooseTurn;

void() ai_face;


float      enemy_vis, enemy_infront, enemy_range;
float      enemy_yaw;


void() knight_attack =
{
        local float                     len;

// decide if now is a good swing time
        len = vlen(self.enemy.origin+self.enemy.view_ofs - (self.origin+self.view_ofs));

        if (len<80)
                knight_atk1 ();
        else
                knight_runatk1 ();
};

//============================================================================

/*
===========
CheckAttack

The player is in view, so decide to move or launch an attack
Returns FALSE if movement should continue
============
*/
float() CheckAttack =
{
        local vector       spot1, spot2;
        local entity       targ;
```

```
	local float		chance;

	targ = self.enemy;

// see if any entities are in the way of the shot
	spot1 = self.origin + self.view_ofs;
	spot2 = targ.origin + targ.view_ofs;

	traceline (spot1, spot2, FALSE, self);

	if (trace_ent != targ)
		return FALSE;		// don't have a clear shot

	if (trace_inopen && trace_inwater)
		return FALSE;			// sight line crossed contents

	if (enemy_range == RANGE_MELEE)
	{	// melee attack
		if (self.th_melee)
		{
			if (self.classname == "monster_knight")
				knight_attack ();
			else
				self.th_melee ();
			return TRUE;
		}
	}

// missile attack
	if (!self.th_missile)
		return FALSE;

	if (time < self.attack_finished)
		return FALSE;

	if (enemy_range == RANGE_FAR)
		return FALSE;

	if (enemy_range == RANGE_MELEE)
	{
		chance = 0.9;
		self.attack_finished = 0;
	}
	else if (enemy_range == RANGE_NEAR)
	{
		if (self.th_melee)
			chance = 0.2;
		else
			chance = 0.4;
	}
	else if (enemy_range == RANGE_MID)
	{
		if (self.th_melee)
			chance = 0.05;
		else
			chance = 0.1;
	}
	else
		chance = 0;

	if (random () < chance)
	{
```

```
                self.th_missile ();
                SUB_AttackFinished (2*random());
                return TRUE;
        }

        return FALSE;
};


/*
=============
ai_face

Stay facing the enemy
=============
*/
void() ai_face =
{
        self.ideal_yaw = vectoyaw(self.enemy.origin - self.origin);
        ChangeYaw ();
};

/*
=============
ai_charge

The monster is in a melee attack, so get as close as possible to .enemy
=============
*/
float (entity targ) visible;
float(entity targ) infront;
float(entity targ) range;

void(float d) ai_charge =
{
        ai_face ();
        movetogoal (d);             // done in C code...
};

void() ai_charge_side =
{
        local   vector  dtemp;
        local   float   heading;

// aim to the left of the enemy for a flyby

        self.ideal_yaw = vectoyaw(self.enemy.origin - self.origin);
        ChangeYaw ();

        makevectors (self.angles);
        dtemp = self.enemy.origin - 30*v_right;
        heading = vectoyaw(dtemp - self.origin);

        walkmove(heading, 20);
};


/*
=============
ai_melee

=============
```

```
*/
void() ai_melee =
{
	local vector	delta;
	local float	ldmg;

	if (!self.enemy)
		return;		// removed before stroke

	delta = self.enemy.origin - self.origin;

	if (vlen(delta) > 60)
		return;

	ldmg = (random() + random() + random()) * 3;
	T_Damage (self.enemy, self, self, ldmg);
};


void() ai_melee_side =
{
	local vector	delta;
	local float	ldmg;

	if (!self.enemy)
		return;		// removed before stroke

	ai_charge_side();

	delta = self.enemy.origin - self.origin;

	if (vlen(delta) > 60)
		return;
	if (!CanDamage (self.enemy, self))
		return;
	ldmg = (random() + random() + random()) * 3;
	T_Damage (self.enemy, self, self, ldmg);
};


//============================================================================

/*
===========
SoldierCheckAttack

The player is in view, so decide to move or launch an attack
Returns FALSE if movement should continue
============
*/
float() SoldierCheckAttack =
{
	local vector	spot1, spot2;
	local entity	targ;
	local float		chance;

	targ = self.enemy;

// see if any entities are in the way of the shot
	spot1 = self.origin + self.view_ofs;
	spot2 = targ.origin + targ.view_ofs;
```

```
	traceline (spot1, spot2, FALSE, self);

	if (trace_inopen && trace_inwater)
		return FALSE;			// sight line crossed contents

	if (trace_ent != targ)
		return FALSE;	// don't have a clear shot


// missile attack
	if (time < self.attack_finished)
		return FALSE;

	if (enemy_range == RANGE_FAR)
		return FALSE;

	if (enemy_range == RANGE_MELEE)
		chance = 0.9;
	else if (enemy_range == RANGE_NEAR)
		chance = 0.4;
	else if (enemy_range == RANGE_MID)
		chance = 0.05;
	else
		chance = 0;

	if (random () < chance)
	{
		self.th_missile ();
		SUB_AttackFinished (1 + random());
		if (random() < 0.3)
			self.lefty = !self.lefty;

		return TRUE;
	}

	return FALSE;
};
//============================================================================

/*
============
ShamCheckAttack

The player is in view, so decide to move or launch an attack
Returns FALSE if movement should continue
============
*/
float() ShamCheckAttack =
{
	local vector	spot1, spot2;
	local entity	targ;
	local float		chance;
	local float		enemy_yaw;

	if (enemy_range == RANGE_MELEE)
	{
		if (CanDamage (self.enemy, self))
		{
			self.attack_state = AS_MELEE;
			return TRUE;
		}
	}
```

```
	if (time < self.attack_finished)
		return FALSE;

	if (!enemy_vis)
		return FALSE;

	targ = self.enemy;

// see if any entities are in the way of the shot
	spot1 = self.origin + self.view_ofs;
	spot2 = targ.origin + targ.view_ofs;

	if (vlen(spot1 - spot2) > 600)
		return FALSE;

	traceline (spot1, spot2, FALSE, self);

	if (trace_inopen && trace_inwater)
		return FALSE;			// sight line crossed contents

	if (trace_ent != targ)
	{
		return FALSE;	// don't have a clear shot
	}

// missile attack
	if (enemy_range == RANGE_FAR)
		return FALSE;

	self.attack_state = AS_MISSILE;
	SUB_AttackFinished (2 + 2*random());
	return TRUE;
};

//=============================================================================

/*
===========
OgreCheckAttack

The player is in view, so decide to move or launch an attack
Returns FALSE if movement should continue
============
*/
float() OgreCheckAttack =
{
	local vector	spot1, spot2;
	local entity	targ;
	local float		chance;

	if (enemy_range == RANGE_MELEE)
	{
		if (CanDamage (self.enemy, self))
		{
			self.attack_state = AS_MELEE;
			return TRUE;
		}
	}

	if (time < self.attack_finished)
		return FALSE;
```

```
	if (!enemy_vis)
		return FALSE;

	targ = self.enemy;

// see if any entities are in the way of the shot
	spot1 = self.origin + self.view_ofs;
	spot2 = targ.origin + targ.view_ofs;

	traceline (spot1, spot2, FALSE, self);

	if (trace_inopen && trace_inwater)
		return FALSE;			// sight line crossed contents

	if (trace_ent != targ)
	{
		return FALSE;	// don't have a clear shot
	}

// missile attack
	if (time < self.attack_finished)
		return FALSE;

	if (enemy_range == RANGE_FAR)
		return FALSE;

	else if (enemy_range == RANGE_NEAR)
		chance = 0.10;
	else if (enemy_range == RANGE_MID)
		chance = 0.05;
	else
		chance = 0;

	self.attack_state = AS_MISSILE;
	SUB_AttackFinished (1 + 2*random());
	return TRUE;
};
```

FLAG.QC

```
/*QUAKED item_deathball (.3 .3 1) (0 0 0) (32 32 32)
*/
void() deathball_touch;

void() item_deathball =
{
	self.touch = deathball_touch;
};
```

ITEMS.QC

```
void() W_SetCurrentAmmo;
/* ALL LIGHTS SHOULD BE 0 1 0 IN COLOR ALL OTHER ITEMS SHOULD
BE .8 .3 .4 IN COLOR */


void() SUB_regen =
{
	self.model = self.mdl;		// restore original model
	self.solid = SOLID_TRIGGER;	// allow it to be touched again
	sound (self, CHAN_VOICE, "items/itembk2.wav", 1, ATTN_NORM);	// play respawn sound
	setorigin (self, self.origin);
};



/*QUAKED noclass (0 0 0) (-8 -8 -8) (8 8 8)
prints a warning message when spawned
*/
void() noclass =
{
	dprint ("noclass spawned at");
	dprint (vtos(self.origin));
	dprint ("\n");
	remove (self);
};



/*
============
PlaceItem

plants the object on the floor
============
*/
void() PlaceItem =
{
	local float	oldz;

	self.mdl = self.model;		// so it can be restored on respawn
	self.flags = FL_ITEM;		// make extra wide
	self.solid = SOLID_TRIGGER;
	self.movetype = MOVETYPE_TOSS;
	self.velocity = '0 0 0';
	self.origin_z = self.origin_z + 6;
	oldz = self.origin_z;
	if (!droptofloor())
	{
		dprint ("Bonus item fell out of level at ");
		dprint (vtos(self.origin));
		dprint ("\n");
		remove(self);
		return;
	}
};

/*
============
StartItem
```

```
Sets the clipping size and plants the object on the floor
============
*/
void() StartItem =
{
	self.nextthink = time + 0.2;		// items start after other solids
	self.think = PlaceItem;
};

/*
=============================================================================

HEALTH BOX

=============================================================================
*/
//
// T_Heal: add health to an entity, limiting health to max_health
// "ignore" will ignore max_health limit
//
float (entity e, float healamount, float ignore) T_Heal =
{
	if (e.health <= 0)
		return 0;
	if ((!ignore) && (e.health >= other.max_health))
		return 0;
	healamount = ceil(healamount);

	e.health = e.health + healamount;
	if ((!ignore) && (e.health >= other.max_health))
		e.health = other.max_health;

	if (e.health > 250)
		e.health = 250;
	return 1;
};

/*QUAKED item_health (.3 .3 1) (0 0 0) (32 32 32) rotten megahealth
Health box. Normally gives 25 points.
Rotten box heals 5-10 points,
megahealth will add 100 health, then
rot you down to your maximum health limit,
one point per second.
*/

float	H_ROTTEN = 1;
float	H_MEGA = 2;
.float	healamount, healtype;
void() health_touch;
void() item_megahealth_rot;

void() item_health =
{
	self.touch = health_touch;

	if (self.spawnflags & H_ROTTEN)
	{
		precache_model("maps/b_bh10.bsp");

		precache_sound("items/r_item1.wav");
		setmodel(self, "maps/b_bh10.bsp");
		self.noise = "items/r_item1.wav";
```

```
		self.healamount = 15;
		self.healtype = 0;
	}
	else
	if (self.spawnflags & H_MEGA)
	{
		precache_model("maps/b_bh100.bsp");
		precache_sound("items/r_item2.wav");
		setmodel(self, "maps/b_bh100.bsp");
		self.noise = "items/r_item2.wav";
		self.healamount = 100;
		self.healtype = 2;
	}
	else
	{
		precache_model("maps/b_bh25.bsp");
		precache_sound("items/health1.wav");
		setmodel(self, "maps/b_bh25.bsp");
		self.noise = "items/health1.wav";
		self.healamount = 25;
		self.healtype = 1;
	}
	setsize (self, '0 0 0', '32 32 56');
	StartItem ();
};


void() health_touch =
{
	local	float amount;
	local	string	s;

	if (other.classname != "player")
		return;

	if (self.healtype == 2) // Megahealth?  Ignore max_health...
	{
		if (other.health >= 250)
			return;
		if (!T_Heal(other, self.healamount, 1))
			return;
	}
	else
	{
		if (!T_Heal(other, self.healamount, 0))
			return;
	}

	sprint(other, "You receive ");
	s = ftos(self.healamount);
	sprint(other, s);
	sprint(other, " health\n");

// health touch sound
	sound(other, CHAN_ITEM, self.noise, 1, ATTN_NORM);

	stuffcmd (other, "bf\n");

	self.model = string_null;
	self.solid = SOLID_NOT;

	// Megahealth = rot down the player's super health
```

```
	if (self.healtype == 2)
	{
		other.items = other.items | IT_SUPERHEALTH;
		self.nextthink = time + 5;
		self.think = item_megahealth_rot;
		self.owner = other;
	}
	else
	{
		if (deathmatch != 2)		// deathmatch 2 is the silly old rules
		{
			if (deathmatch)
				self.nextthink = time + 20;
			self.think = SUB_regen;
		}
	}

	activator = other;
	SUB_UseTargets();					// fire all targets / killtargets
};

void() item_megahealth_rot =
{
	other = self.owner;

	if (other.health > other.max_health)
	{
		other.health = other.health - 1;
		self.nextthink = time + 1;
		return;
	}

// it is possible for a player to die and respawn between rots, so don't
// just blindly subtract the flag off
	other.items = other.items - (other.items & IT_SUPERHEALTH);

	if (deathmatch == 1)	// deathmatch 2 is silly old rules
	{
		self.nextthink = time + 20;
		self.think = SUB_regen;
	}
};

/*
==============================================================================

ARMOR

==============================================================================
*/

void() armor_touch;

void() armor_touch =
{
	local	float	type, value, bit;

	if (other.health <= 0)
		return;
	if (other.classname != "player")
		return;
```

```
	if (self.classname == "item_armor1")
	{
		type = 0.3;
		value = 100;
		bit = IT_ARMOR1;
	}
	if (self.classname == "item_armor2")
	{
		type = 0.6;
		value = 150;
		bit = IT_ARMOR2;
	}
	if (self.classname == "item_armorInv")
	{
		type = 0.8;
		value = 200;
		bit = IT_ARMOR3;
	}
	if (other.armortype*other.armorvalue >= type*value)
		return;

	other.armortype = type;
	other.armorvalue = value;
	other.items = other.items - (other.items & (IT_ARMOR1 | IT_ARMOR2 | IT_ARMOR3)) + bit;

	self.solid = SOLID_NOT;
	self.model = string_null;
	if (deathmatch == 1)
		self.nextthink = time + 20;
	self.think = SUB_regen;

	sprint(other, "You got armor\n");
// armor touch sound
	sound(other, CHAN_ITEM, "items/armor1.wav", 1, ATTN_NORM);
	stuffcmd (other, "bf\n");

	activator = other;
	SUB_UseTargets();				// fire all targets / killtargets
};


/*QUAKED item_armor1 (0 .5 .8) (-16 -16 0) (16 16 32)
*/

void() item_armor1 =
{
	self.touch = armor_touch;
	precache_model ("progs/armor.mdl");
	setmodel (self, "progs/armor.mdl");
	self.skin = 0;
	setsize (self, '-16 -16 0', '16 16 56');
	StartItem ();
};

/*QUAKED item_armor2 (0 .5 .8) (-16 -16 0) (16 16 32)
*/

void() item_armor2 =
{
	self.touch = armor_touch;
	precache_model ("progs/armor.mdl");
	setmodel (self, "progs/armor.mdl");
```

```
	self.skin = 1;
	setsize (self, '-16 -16 0', '16 16 56');
	StartItem ();
};

/*QUAKED item_armorInv (0 .5 .8) (-16 -16 0) (16 16 32)
*/

void() item_armorInv =
{
	self.touch = armor_touch;
	precache_model ("progs/armor.mdl");
	setmodel (self, "progs/armor.mdl");
	self.skin = 2;
	setsize (self, '-16 -16 0', '16 16 56');
	StartItem ();
};

/*
=============================================================================

WEAPONS

=============================================================================
*/

void() bound_other_ammo =
{
	if (other.ammo_shells > 100)
		other.ammo_shells = 100;
	if (other.ammo_nails > 200)
		other.ammo_nails = 200;
	if (other.ammo_rockets > 100)
		other.ammo_rockets = 100;
	if (other.ammo_cells > 100)
		other.ammo_cells = 100;
};


float(float w) RankForWeapon =
{
	if (w == IT_LIGHTNING)
		return 1;
	if (w == IT_ROCKET_LAUNCHER)
		return 2;
	if (w == IT_SUPER_NAILGUN)
		return 3;
	if (w == IT_GRENADE_LAUNCHER)
		return 4;
	if (w == IT_SUPER_SHOTGUN)
		return 5;
	if (w == IT_NAILGUN)
		return 6;
	return 7;
};

/*
=============
Deathmatch_Weapon

Deathmatch weapon change rules for picking up a weapon
```

```
.float          ammo_shells, ammo_nails, ammo_rockets, ammo_cells;
=============
*/
void(float old, float new) Deathmatch_Weapon =
{
        local float or, nr;

// change self.weapon if desired
        or = RankForWeapon (self.weapon);
        nr = RankForWeapon (new);
        if ( nr < or )
                self.weapon = new;
};

/*
=============
weapon_touch
=============
*/
float() W_BestWeapon;

void() weapon_touch =
{
        local   float   hadammo, best, new, old;
        local   entity  stemp;
        local   float   leave;

        if (!(other.flags & FL_CLIENT))
                return;

// if the player was using his best weapon, change up to the new one if better
        stemp = self;
        self = other;
        best = W_BestWeapon();
        self = stemp;

        if (deathmatch == 2 || coop)
                leave = 1;
        else
                leave = 0;

        if (self.classname == "weapon_nailgun")
        {
                if (leave && (other.items & IT_NAILGUN) )
                        return;
                hadammo = other.ammo_nails;
                new = IT_NAILGUN;
                other.ammo_nails = other.ammo_nails + 30;
        }
        else if (self.classname == "weapon_supernailgun")
        {
                if (leave && (other.items & IT_SUPER_NAILGUN) )
                        return;
                hadammo = other.ammo_rockets;
                new = IT_SUPER_NAILGUN;
                other.ammo_nails = other.ammo_nails + 30;
        }
        else if (self.classname == "weapon_supershotgun")
        {
                if (leave && (other.items & IT_SUPER_SHOTGUN) )
                        return;
                hadammo = other.ammo_rockets;
```

```
		new = IT_SUPER_SHOTGUN;
		other.ammo_shells = other.ammo_shells + 5;
	}
	else if (self.classname == "weapon_rocketlauncher")
	{
		if (leave && (other.items & IT_ROCKET_LAUNCHER) )
			return;
		hadammo = other.ammo_rockets;
		new = IT_ROCKET_LAUNCHER;
		other.ammo_rockets = other.ammo_rockets + 5;
	}
	else if (self.classname == "weapon_grenadelauncher")
	{
		if (leave && (other.items & IT_GRENADE_LAUNCHER) )
			return;
		hadammo = other.ammo_rockets;
		new = IT_GRENADE_LAUNCHER;
		other.ammo_rockets = other.ammo_rockets + 5;
	}
	else if (self.classname == "weapon_lightning")
	{
		if (leave && (other.items & IT_LIGHTNING) )
			return;
		hadammo = other.ammo_rockets;
		new = IT_LIGHTNING;
		other.ammo_cells = other.ammo_cells + 15;
	}
	else
		objerror ("weapon_touch: unknown classname");

	sprint (other, "You got the ");
	sprint (other, self.netname);
	sprint (other, "\n");
// weapon touch sound
	sound (other, CHAN_ITEM, "weapons/pkup.wav", 1, ATTN_NORM);
	stuffcmd (other, "bf\n");

	bound_other_ammo ();

// change to the weapon
	old = other.items;
	other.items = other.items | new;

	stemp = self;
	self = other;

	if (!deathmatch)
		self.weapon = new;
	else
		Deathmatch_Weapon (old, new);

	W_SetCurrentAmmo();

	self = stemp;

	if (leave)
		return;

// remove it in single player, or setup for respawning in deathmatch
	self.model = string_null;
	self.solid = SOLID_NOT;
	if (deathmatch == 1)
```

```
                self.nextthink = time + 30;
        self.think = SUB_regen;

        activator = other;
        SUB_UseTargets();                               // fire all targets / killtargets
};


/*QUAKED weapon_supershotgun (0 .5 .8) (-16 -16 0) (16 16 32)
*/

void() weapon_supershotgun =
{
        precache_model ("progs/g_shot.mdl");
        setmodel (self, "progs/g_shot.mdl");
        self.weapon = IT_SUPER_SHOTGUN;
        self.netname = "Double-barrelled Shotgun";
        self.touch = weapon_touch;
        setsize (self, '-16 -16 0', '16 16 56');
        StartItem ();
};

/*QUAKED weapon_nailgun (0 .5 .8) (-16 -16 0) (16 16 32)
*/

void() weapon_nailgun =
{
        precache_model ("progs/g_nail.mdl");
        setmodel (self, "progs/g_nail.mdl");
        self.weapon = IT_NAILGUN;
        self.netname = "nailgun";
        self.touch = weapon_touch;
        setsize (self, '-16 -16 0', '16 16 56');
        StartItem ();
};

/*QUAKED weapon_supernailgun (0 .5 .8) (-16 -16 0) (16 16 32)
*/

void() weapon_supernailgun =
{
        precache_model ("progs/g_nail2.mdl");
        setmodel (self, "progs/g_nail2.mdl");
        self.weapon = IT_SUPER_NAILGUN;
        self.netname = "Super Nailgun";
        self.touch = weapon_touch;
        setsize (self, '-16 -16 0', '16 16 56');
        StartItem ();
};

/*QUAKED weapon_grenadelauncher (0 .5 .8) (-16 -16 0) (16 16 32)
*/

void() weapon_grenadelauncher =
{
        precache_model ("progs/g_rock.mdl");
        setmodel (self, "progs/g_rock.mdl");
        self.weapon = 3;
        self.netname = "Grenade Launcher";
        self.touch = weapon_touch;
        setsize (self, '-16 -16 0', '16 16 56');
        StartItem ();
```

```
};

/*QUAKED weapon_rocketlauncher (0 .5 .8) (-16 -16 0) (16 16 32)
*/

void() weapon_rocketlauncher =
{
	precache_model ("progs/g_rock2.mdl");
	setmodel (self, "progs/g_rock2.mdl");
	self.weapon = 3;
	self.netname = "Rocket Launcher";
	self.touch = weapon_touch;
	setsize (self, '-16 -16 0', '16 16 56');
	StartItem ();
};


/*QUAKED weapon_lightning (0 .5 .8) (-16 -16 0) (16 16 32)
*/

void() weapon_lightning =
{
	precache_model ("progs/g_light.mdl");
	setmodel (self, "progs/g_light.mdl");
	self.weapon = 3;
	self.netname = "Thunderbolt";
	self.touch = weapon_touch;
	setsize (self, '-16 -16 0', '16 16 56');
	StartItem ();
};


/*
==============================================================================

AMMO

==============================================================================
*/

void() ammo_touch =
{
local entity	stemp;
local float		best;

	if (other.classname != "player")
		return;
	if (other.health <= 0)
		return;

// if the player was using his best weapon, change up to the new one if better
	stemp = self;
	self = other;
	best = W_BestWeapon();
	self = stemp;


// shotgun
	if (self.weapon == 1)
	{
		if (other.ammo_shells >= 100)
			return;
```

```
                other.ammo_shells = other.ammo_shells + self.aflag;
        }

// spikes
        if (self.weapon == 2)
        {
                if (other.ammo_nails >= 200)
                        return;
                other.ammo_nails = other.ammo_nails + self.aflag;
        }

//      rockets
        if (self.weapon == 3)
        {
                if (other.ammo_rockets >= 100)
                        return;
                other.ammo_rockets = other.ammo_rockets + self.aflag;
        }

//      cells
        if (self.weapon == 4)
        {
                if (other.ammo_cells >= 100)
                        return;
                other.ammo_cells = other.ammo_cells + self.aflag;
        }

        bound_other_ammo ();

        sprint (other, "You got the ");
        sprint (other, self.netname);
        sprint (other, "\n");
// ammo touch sound
        sound (other, CHAN_ITEM, "weapons/lock4.wav", 1, ATTN_NORM);
        stuffcmd (other, "bf\n");

// change to a better weapon if appropriate

        if ( other.weapon == best )
        {
                stemp = self;
                self = other;
                self.weapon = W_BestWeapon();
                W_SetCurrentAmmo ();
                self = stemp;
        }

// if changed current ammo, update it
        stemp = self;
        self = other;
        W_SetCurrentAmmo();
        self = stemp;

// remove it in single player, or setup for respawning in deathmatch
        self.model = string_null;
        self.solid = SOLID_NOT;
        if (deathmatch == 1)
                self.nextthink = time + 30;
        self.think = SUB_regen;

        activator = other;
        SUB_UseTargets();                               // fire all targets / killtargets
```

```
};



float WEAPON_BIG2 = 1;

/*QUAKED item_shells (0 .5 .8) (0 0 0) (32 32 32) big
*/

void() item_shells =
{
	self.touch = ammo_touch;

	if (self.spawnflags & WEAPON_BIG2)
	{
		precache_model ("maps/b_shell1.bsp");
		setmodel (self, "maps/b_shell1.bsp");
		self.aflag = 40;
	}
	else
	{
		precache_model ("maps/b_shell0.bsp");
		setmodel (self, "maps/b_shell0.bsp");
		self.aflag = 20;
	}
	self.weapon = 1;
	self.netname = "shells";
	setsize (self, '0 0 0', '32 32 56');
	StartItem ();
};

/*QUAKED item_spikes (0 .5 .8) (0 0 0) (32 32 32) big
*/

void() item_spikes =
{
	self.touch = ammo_touch;

	if (self.spawnflags & WEAPON_BIG2)
	{
		precache_model ("maps/b_nail1.bsp");
		setmodel (self, "maps/b_nail1.bsp");
		self.aflag = 50;
	}
	else
	{
		precache_model ("maps/b_nail0.bsp");
		setmodel (self, "maps/b_nail0.bsp");
		self.aflag = 25;
	}
	self.weapon = 2;
	self.netname = "nails";
	setsize (self, '0 0 0', '32 32 56');
	StartItem ();
};

/*QUAKED item_rockets (0 .5 .8) (0 0 0) (32 32 32) big
*/

void() item_rockets =
{
```

```
	self.touch = ammo_touch;

	if (self.spawnflags & WEAPON_BIG2)
	{
		precache_model ("maps/b_rock1.bsp");
		setmodel (self, "maps/b_rock1.bsp");
		self.aflag = 10;
	}
	else
	{
		precache_model ("maps/b_rock0.bsp");
		setmodel (self, "maps/b_rock0.bsp");
		self.aflag = 5;
	}
	self.weapon = 3;
	self.netname = "rockets";
	setsize (self, '0 0 0', '32 32 56');
	StartItem ();
};


/*QUAKED item_cells (0 .5 .8) (0 0 0) (32 32 32) big
*/

void() item_cells =
{
	self.touch = ammo_touch;

	if (self.spawnflags & WEAPON_BIG2)
	{
		precache_model ("maps/b_batt1.bsp");
		setmodel (self, "maps/b_batt1.bsp");
		self.aflag = 12;
	}
	else
	{
		precache_model ("maps/b_batt0.bsp");
		setmodel (self, "maps/b_batt0.bsp");
		self.aflag = 6;
	}
	self.weapon = 4;
	self.netname = "cells";
	setsize (self, '0 0 0', '32 32 56');
	StartItem ();
};


/*QUAKED item_weapon (0 .5 .8) (0 0 0) (32 32 32) shotgun rocket spikes big
DO NOT USE THIS!!!! IT WILL BE REMOVED!
*/

float WEAPON_SHOTGUN = 1;
float WEAPON_ROCKET = 2;
float WEAPON_SPIKES = 4;
float WEAPON_BIG = 8;
void() item_weapon =
{
	self.touch = ammo_touch;

	if (self.spawnflags & WEAPON_SHOTGUN)
	{
		if (self.spawnflags & WEAPON_BIG)
```

```
		{
			precache_model ("maps/b_shell1.bsp");
			setmodel (self, "maps/b_shell1.bsp");
			self.aflag = 40;
		}
		else
		{
			precache_model ("maps/b_shell0.bsp");
			setmodel (self, "maps/b_shell0.bsp");
			self.aflag = 20;
		}
		self.weapon = 1;
		self.netname = "shells";
	}

	if (self.spawnflags & WEAPON_SPIKES)
	{
		if (self.spawnflags & WEAPON_BIG)
		{
			precache_model ("maps/b_nail1.bsp");
			setmodel (self, "maps/b_nail1.bsp");
			self.aflag = 40;
		}
		else
		{
			precache_model ("maps/b_nail0.bsp");
			setmodel (self, "maps/b_nail0.bsp");
			self.aflag = 20;
		}
		self.weapon = 2;
		self.netname = "spikes";
	}

	if (self.spawnflags & WEAPON_ROCKET)
	{
		if (self.spawnflags & WEAPON_BIG)
		{
			precache_model ("maps/b_rock1.bsp");
			setmodel (self, "maps/b_rock1.bsp");
			self.aflag = 10;
		}
		else
		{
			precache_model ("maps/b_rock0.bsp");
			setmodel (self, "maps/b_rock0.bsp");
			self.aflag = 5;
		}
		self.weapon = 3;
		self.netname = "rockets";
	}

	setsize (self, '0 0 0', '32 32 56');
	StartItem ();
};


/*
=============================================================================

KEYS

=============================================================================
```

```
*/

void() key_touch =
{
local entity        stemp;
local float                 best;

        if (other.classname != "player")
                return;
        if (other.health <= 0)
                return;
        if (other.items & self.items)
                return;

        sprint (other, "You got the ");
        sprint (other, self.netname);
        sprint (other,"\n");

        sound (other, CHAN_ITEM, self.noise, 1, ATTN_NORM);
        stuffcmd (other, "bf\n");
        other.items = other.items | self.items;

        if (!coop)
        {
                self.solid = SOLID_NOT;
                self.model = string_null;
        }

        activator = other;
        SUB_UseTargets();                               // fire all targets / killtargets
};


void() key_setsounds =
{
        if (world.worldtype == 0)
        {
                precache_sound ("misc/medkey.wav");
                self.noise = "misc/medkey.wav";
        }
        if (world.worldtype == 1)
        {
                precache_sound ("misc/runekey.wav");
                self.noise = "misc/runekey.wav";
        }
        if (world.worldtype == 2)
        {
                precache_sound2 ("misc/basekey.wav");
                self.noise = "misc/basekey.wav";
        }
};

/*QUAKED item_key1 (0 .5 .8) (-16 -16 -24) (16 16 32)
SILVER key
In order for keys to work
you MUST set your maps
worldtype to one of the
following:
0: medieval
1: metal
2: base
*/
```

```
void() item_key1 =
{
	if (world.worldtype == 0)
	{
		precache_model ("progs/w_s_key.mdl");
		setmodel (self, "progs/w_s_key.mdl");
		self.netname = "silver key";
	}
	else if (world.worldtype == 1)
	{
		precache_model ("progs/m_s_key.mdl");
		setmodel (self, "progs/m_s_key.mdl");
		self.netname = "silver runekey";
	}
	else if (world.worldtype == 2)
	{
		precache_model2 ("progs/b_s_key.mdl");
		setmodel (self, "progs/b_s_key.mdl");
		self.netname = "silver keycard";
	}
	key_setsounds();
	self.touch = key_touch;
	self.items = IT_KEY1;
	setsize (self, '-16 -16 -24', '16 16 32');
	StartItem ();
};

/*QUAKED item_key2 (0 .5 .8) (-16 -16 -24) (16 16 32)
GOLD key
In order for keys to work
you MUST set your maps
worldtype to one of the
following:
0: medieval
1: metal
2: base
*/

void() item_key2 =
{
	if (world.worldtype == 0)
	{
		precache_model ("progs/w_g_key.mdl");
		setmodel (self, "progs/w_g_key.mdl");
		self.netname = "gold key";
	}
	if (world.worldtype == 1)
	{
		precache_model ("progs/m_g_key.mdl");
		setmodel (self, "progs/m_g_key.mdl");
		self.netname = "gold runekey";
	}
	if (world.worldtype == 2)
	{
		precache_model2 ("progs/b_g_key.mdl");
		setmodel (self, "progs/b_g_key.mdl");
		self.netname = "gold keycard";
	}
	key_setsounds();
	self.touch = key_touch;
	self.items = IT_KEY2;
```

```
	setsize (self, '-16 -16 -24', '16 16 32');
	StartItem ();
};



/*
==============================================================================

END OF LEVEL RUNES

==============================================================================
*/

void() sigil_touch =
{
local entity	stemp;
local float		best;

	if (other.classname != "player")
		return;
	if (other.health <= 0)
		return;

	centerprint (other, "You got the rune!");

	sound (other, CHAN_ITEM, self.noise, 1, ATTN_NORM);
	stuffcmd (other, "bf\n");
	self.solid = SOLID_NOT;
	self.model = string_null;
	serverflags = serverflags | (self.spawnflags & 15);
	self.classname = "";			// so rune doors won't find it

	activator = other;
	SUB_UseTargets();				// fire all targets / killtargets
};


/*QUAKED item_sigil (0 .5 .8) (-16 -16 -24) (16 16 32) E1 E2 E3 E4
End of level sigil, pick up to end episode and return to jrstart.
*/

void() item_sigil =
{
	if (!self.spawnflags)
		objerror ("no spawnflags");

	precache_sound ("misc/runekey.wav");
	self.noise = "misc/runekey.wav";

	if (self.spawnflags & 1)
	{
		precache_model ("progs/end1.mdl");
		setmodel (self, "progs/end1.mdl");
	}
	if (self.spawnflags & 2)
	{
		precache_model2 ("progs/end2.mdl");
		setmodel (self, "progs/end2.mdl");
	}
	if (self.spawnflags & 4)
	{
```

```
		precache_model2 ("progs/end3.mdl");
		setmodel (self, "progs/end3.mdl");
	}
	if (self.spawnflags & 8)
	{
		precache_model2 ("progs/end4.mdl");
		setmodel (self, "progs/end4.mdl");
	}

	self.touch = sigil_touch;
	setsize (self, '-16 -16 -24', '16 16 32');
	StartItem ();
};

/*
==============================================================================

POWERUPS

==============================================================================
*/

void() powerup_touch;


void() powerup_touch =
{
local entity	stemp;
local float		best;

	if (other.classname != "player")
		return;
	if (other.health <= 0)
		return;

	sprint (other, "You got the ");
	sprint (other, self.netname);
	sprint (other,"\n");

	if (deathmatch)
	{
		self.mdl = self.model;

		if ((self.classname == "item_artifact_invulnerability") ||
			(self.classname == "item_artifact_invisibility"))
			self.nextthink = time + 60*5;
		else
			self.nextthink = time + 60;

		self.think = SUB_regen;
	}

	sound (other, CHAN_VOICE, self.noise, 1, ATTN_NORM);
	stuffcmd (other, "bf\n");
	self.solid = SOLID_NOT;
	other.items = other.items | self.items;
	self.model = string_null;

// do the apropriate action
	if (self.classname == "item_artifact_envirosuit")
	{
		other.rad_time = 1;
```

```
		other.radsuit_finished = time + 30;
	}

	if (self.classname == "item_artifact_invulnerability")
	{
		other.invincible_time = 1;
		other.invincible_finished = time + 30;
	}

	if (self.classname == "item_artifact_invisibility")
	{
		other.invisible_time = 1;
		other.invisible_finished = time + 30;
	}

	if (self.classname == "item_artifact_super_damage")
	{
		other.super_time = 1;
		other.super_damage_finished = time + 30;
	}

	activator = other;
	SUB_UseTargets();				// fire all targets / killtargets
};



/*QUAKED item_artifact_invulnerability (0 .5 .8) (-16 -16 -24) (16 16 32)
Player is invulnerable for 30 seconds
*/
void() item_artifact_invulnerability =
{
	self.touch = powerup_touch;

	precache_model ("progs/invulner.mdl");
	precache_sound ("items/protect.wav");
	precache_sound ("items/protect2.wav");
	precache_sound ("items/protect3.wav");
	self.noise = "items/protect.wav";
	setmodel (self, "progs/invulner.mdl");
	self.netname = "Pentagram of Protection";
	self.items = IT_INVULNERABILITY;
	setsize (self, '-16 -16 -24', '16 16 32');
	StartItem ();
};

/*QUAKED item_artifact_envirosuit (0 .5 .8) (-16 -16 -24) (16 16 32)
Player takes no damage from water or slime for 30 seconds
*/
void() item_artifact_envirosuit =
{
	self.touch = powerup_touch;

	precache_model ("progs/suit.mdl");
	precache_sound ("items/suit.wav");
	precache_sound ("items/suit2.wav");
	self.noise = "items/suit.wav";
	setmodel (self, "progs/suit.mdl");
	self.netname = "Biosuit";
	self.items = IT_SUIT;
	setsize (self, '-16 -16 -24', '16 16 32');
	StartItem ();
```

```
};


/*QUAKED item_artifact_invisibility (0 .5 .8) (-16 -16 -24) (16 16 32)
Player is invisible for 30 seconds
*/
void() item_artifact_invisibility =
{
	self.touch = powerup_touch;

	precache_model ("progs/invisibl.mdl");
	precache_sound ("items/inv1.wav");
	precache_sound ("items/inv2.wav");
	precache_sound ("items/inv3.wav");
	self.noise = "items/inv1.wav";
	setmodel (self, "progs/invisibl.mdl");
	self.netname = "Ring of Shadows";
	self.items = IT_INVISIBILITY;
	setsize (self, '-16 -16 -24', '16 16 32');
	StartItem ();
};


/*QUAKED item_artifact_super_damage (0 .5 .8) (-16 -16 -24) (16 16 32)
The next attack from the player will do 4x damage
*/
void() item_artifact_super_damage =
{
	self.touch = powerup_touch;

	precache_model ("progs/quaddama.mdl");
	precache_sound ("items/damage.wav");
	precache_sound ("items/damage2.wav");
	precache_sound ("items/damage3.wav");
	self.noise = "items/damage.wav";
	setmodel (self, "progs/quaddama.mdl");
	self.netname = "Quad Damage";
	self.items = IT_QUAD;
	setsize (self, '-16 -16 -24', '16 16 32');
	StartItem ();
};



/*
==============================================================================

PLAYER BACKPACKS

==============================================================================
*/

void() BackpackTouch =
{
	local string	s;
	local	float	best, old, new;
	local		entity	stemp;
	local	float	acount;

	if (other.classname != "player")
		return;
	if (other.health <= 0)
```

```
		return;

	acount = 0;
	sprint (other, "You get ");

	if (self.items)
		if ((other.items & self.items) == 0)
		{
			acount = 1;
			sprint (other, "the ");
			sprint (other, self.netname);
		}

// if the player was using his best weapon, change up to the new one if better
	stemp = self;
	self = other;
	best = W_BestWeapon();
	self = stemp;

// change weapons
	other.ammo_shells = other.ammo_shells + self.ammo_shells;
	other.ammo_nails = other.ammo_nails + self.ammo_nails;
	other.ammo_rockets = other.ammo_rockets + self.ammo_rockets;
	other.ammo_cells = other.ammo_cells + self.ammo_cells;

	new = self.items;
	if (!new)
		new = other.weapon;
	old = other.items;
	other.items = other.items | new;

	bound_other_ammo ();

	if (self.ammo_shells)
	{
		if (acount)
			sprint(other, ", ");
		acount = 1;
		s = ftos(self.ammo_shells);
		sprint (other, s);
		sprint (other, " shells");
	}
	if (self.ammo_nails)
	{
		if (acount)
			sprint(other, ", ");
		acount = 1;
		s = ftos(self.ammo_nails);
		sprint (other, s);
		sprint (other, " nails");
	}
	if (self.ammo_rockets)
	{
		if (acount)
			sprint(other, ", ");
		acount = 1;
		s = ftos(self.ammo_rockets);
		sprint (other, s);
		sprint (other, " rockets");
	}
	if (self.ammo_cells)
	{
```

```
                if (acount)
                        sprint(other, ", ");
                acount = 1;
                s = ftos(self.ammo_cells);
                sprint (other, s);
                sprint (other, " cells");
        }

        sprint (other, "\n");
// backpack touch sound
        sound (other, CHAN_ITEM, "weapons/lock4.wav", 1, ATTN_NORM);
        stuffcmd (other, "bf\n");

// remove the backpack, change self to the player
        remove(self);
        self = other;

// change to the weapon
        if (!deathmatch)
                self.weapon = new;
        else
                Deathmatch_Weapon (old, new);

        W_SetCurrentAmmo ();
};

/*
===============
DropBackpack
===============
*/
void() DropBackpack =
{
        local entity        item;

        if (!(self.ammo_shells + self.ammo_nails + self.ammo_rockets + self.ammo_cells))
                return;  // nothing in it

        item = spawn();
        item.origin = self.origin - '0 0 24';

        item.items = self.weapon;
        if (item.items == IT_AXE)
                item.netname = "Axe";
        else if (item.items == IT_SHOTGUN)
                item.netname = "Shotgun";
        else if (item.items == IT_SUPER_SHOTGUN)
                item.netname = "Double-barrelled Shotgun";
        else if (item.items == IT_NAILGUN)
                item.netname = "Nailgun";
        else if (item.items == IT_SUPER_NAILGUN)
                item.netname = "Super Nailgun";
        else if (item.items == IT_GRENADE_LAUNCHER)
                item.netname = "Grenade Launcher";
        else if (item.items == IT_ROCKET_LAUNCHER)
                item.netname = "Rocket Launcher";
        else if (item.items == IT_LIGHTNING)
                item.netname = "Thunderbolt";
        else
                item.netname = "";

        item.ammo_shells = self.ammo_shells;
```

```
        item.ammo_nails = self.ammo_nails;
        item.ammo_rockets = self.ammo_rockets;
        item.ammo_cells = self.ammo_cells;

        item.velocity_z = 300;
        item.velocity_x = -100 + (random() * 200);
        item.velocity_y = -100 + (random() * 200);

        item.flags = FL_ITEM;
        item.solid = SOLID_TRIGGER;
        item.movetype = MOVETYPE_TOSS;
        setmodel (item, "progs/backpack.mdl");
        setsize (item, '-16 -16 0', '16 16 56');
        item.touch = BackpackTouch;

        item.nextthink = time + 120;        // remove after 2 minutes
        item.think = SUB_Remove;
};
```

JCTEST.QC

```
void() jctrig =
{
dprint ("here\n\n");
	lightstyle(0, "az");
};

/*QUAKED trigger_jctest (.5 .5 .5) ?
*/
void() trigger_jctest =
{
	setsize (self, self.mins, self.maxs);
	self.solid = SOLID_EDGE;
	self.touch = jctrig;
};
```

MISC.QC

```
/*QUAKED info_null (0 0.5 0) (-4 -4 -4) (4 4 4)
Used as a positional target for spotlights, etc.
*/
void() info_null =
{
	remove(self);
};

/*QUAKED info_notnull (0 0.5 0) (-4 -4 -4) (4 4 4)
Used as a positional target for lightning.
*/
void() info_notnull =
{
};

//============================================================================

float START_OFF = 1;

void() light_use =
{
	if (self.spawnflags & START_OFF)
	{
		lightstyle(self.style, "m");
		self.spawnflags = self.spawnflags - START_OFF;
	}
	else
	{
		lightstyle(self.style, "a");
		self.spawnflags = self.spawnflags + START_OFF;
	}
};

/*QUAKED light (0 1 0) (-8 -8 -8) (8 8 8) START_OFF
Non-displayed light.
Default light value is 300
Default style is 0
If targeted, it will toggle between on or off.
*/
void() light =
{
	if (!self.targetname)
	{	// inert light
		remove(self);
		return;
	}

	if (self.style >= 32)
	{
		self.use = light_use;
		if (self.spawnflags & START_OFF)
			lightstyle(self.style, "a");
		else
			lightstyle(self.style, "m");
	}
};

/*QUAKED light_fluoro (0 1 0) (-8 -8 -8) (8 8 8) START_OFF
Non-displayed light.
```

```
Default light value is 300
Default style is 0
If targeted, it will toggle between on or off.
Makes steady fluorescent humming sound
*/
void() light_fluoro =
{
	if (self.style >= 32)
	{
		self.use = light_use;
		if (self.spawnflags & START_OFF)
			lightstyle(self.style, "a");
		else
			lightstyle(self.style, "m");
	}

	precache_sound ("ambience/fl_hum1.wav");
	ambientsound (self.origin, "ambience/fl_hum1.wav", 0.5, ATTN_STATIC);
};

/*QUAKED light_fluorospark (0 1 0) (-8 -8 -8) (8 8 8)
Non-displayed light.
Default light value is 300
Default style is 10
Makes sparking, broken fluorescent sound
*/
void() light_fluorospark =
{
	if (!self.style)
		self.style = 10;

	precache_sound ("ambience/buzz1.wav");
	ambientsound (self.origin, "ambience/buzz1.wav", 0.5, ATTN_STATIC);
};

/*QUAKED light_globe (0 1 0) (-8 -8 -8) (8 8 8)
Sphere globe light.
Default light value is 300
Default style is 0
*/
void() light_globe =
{
	precache_model ("progs/s_light.spr");
	setmodel (self, "progs/s_light.spr");
	makestatic (self);
};

void() FireAmbient =
{
	precache_sound ("ambience/fire1.wav");
// attenuate fast
	ambientsound (self.origin, "ambience/fire1.wav", 0.5, ATTN_STATIC);
};

/*QUAKED light_torch_small_walltorch (0 .5 0) (-10 -10 -20) (10 10 20)
Short wall torch
Default light value is 200
Default style is 0
*/
void() light_torch_small_walltorch =
{
	precache_model ("progs/flame.mdl");
```

```
	setmodel (self, "progs/flame.mdl");
	FireAmbient ();
	makestatic (self);
};

/*QUAKED light_flame_large_yellow (0 1 0) (-10 -10 -12) (12 12 18)
Large yellow flame ball
*/
void() light_flame_large_yellow =
{
	precache_model ("progs/flame2.mdl");
	setmodel (self, "progs/flame2.mdl");
	self.frame = 1;
	FireAmbient ();
	makestatic (self);
};

/*QUAKED light_flame_small_yellow (0 1 0) (-8 -8 -8) (8 8 8) START_OFF
Small yellow flame ball
*/
void() light_flame_small_yellow =
{
	precache_model ("progs/flame2.mdl");
	setmodel (self, "progs/flame2.mdl");
	FireAmbient ();
	makestatic (self);
};

/*QUAKED light_flame_small_white (0 1 0) (-10 -10 -40) (10 10 40) START_OFF
Small white flame ball
*/
void() light_flame_small_white =
{
	precache_model ("progs/flame2.mdl");
	setmodel (self, "progs/flame2.mdl");
	FireAmbient ();
	makestatic (self);
};

//===========================================================================


/*QUAKED misc_fireball (0 .5 .8) (-8 -8 -8) (8 8 8)
Lava Balls
*/

void() fire_fly;
void() fire_touch;
void() misc_fireball =
{

	precache_model ("progs/lavaball.mdl");
	self.classname = "fireball";
	self.nextthink = time + (random() * 5);
	self.think = fire_fly;
	if (!self.speed)
		self.speed == 1000;
};

void() fire_fly =
{
local entity	fireball;
```

```
	fireball = spawn();
	fireball.solid = SOLID_TRIGGER;
	fireball.movetype = MOVETYPE_TOSS;
	fireball.velocity = '0 0 1000';
	fireball.velocity_x = (random() * 100) - 50;
	fireball.velocity_y = (random() * 100) - 50;
	fireball.velocity_z = self.speed + (random() * 200);
	fireball.classname = "fireball";
	setmodel (fireball, "progs/lavaball.mdl");
	setsize (fireball, '0 0 0', '0 0 0');
	setorigin (fireball, self.origin);
	fireball.nextthink = time + 5;
	fireball.think = SUB_Remove;
	fireball.touch = fire_touch;

	self.nextthink = time + (random() * 5) + 3;
	self.think = fire_fly;
};


void() fire_touch =
{
	T_Damage (other, self, self, 20);
	remove(self);
};

//============================================================================


void() barrel_explode =
{
	self.takedamage = DAMAGE_NO;
	self.classname = "explo_box";
	// did say self.owner
	T_RadiusDamage (self, self, 160, world);
	sound (self, CHAN_VOICE, "weapons/r_exp3.wav", 1, ATTN_NORM);
	particle (self.origin, '0 0 0', 75, 255);

	self.origin_z = self.origin_z + 32;
	BecomeExplosion ();
};



/*QUAKED misc_explobox (0 .5 .8) (0 0 0) (32 32 64)
TESTING THING
*/

void() misc_explobox =
{
	local float	oldz;

	self.solid = SOLID_BBOX;
	self.movetype = MOVETYPE_NONE;
	precache_model ("maps/b_explob.bsp");
	setmodel (self, "maps/b_explob.bsp");
	precache_sound ("weapons/r_exp3.wav");
	self.health = 20;
	self.th_die = barrel_explode;
	self.takedamage = DAMAGE_AIM;
```

```
	self.origin_z = self.origin_z + 2;
	oldz = self.origin_z;
	droptofloor();
	if (oldz - self.origin_z > 250)
	{
		dprint ("item fell out of level at ");
		dprint (vtos(self.origin));
		dprint ("\n");
		remove(self);
	}
};



/*QUAKED misc_explobox2 (0 .5 .8) (0 0 0) (32 32 64)
Smaller exploding box, REGISTERED ONLY
*/

void() misc_explobox2 =
{
	local float	oldz;

	self.solid = SOLID_BBOX;
	self.movetype = MOVETYPE_NONE;
	precache_model2 ("maps/b_exbox2.bsp");
	setmodel (self, "maps/b_exbox2.bsp");
	precache_sound ("weapons/r_exp3.wav");
	self.health = 20;
	self.th_die = barrel_explode;
	self.takedamage = DAMAGE_AIM;

	self.origin_z = self.origin_z + 2;
	oldz = self.origin_z;
	droptofloor();
	if (oldz - self.origin_z > 250)
	{
		dprint ("item fell out of level at ");
		dprint (vtos(self.origin));
		dprint ("\n");
		remove(self);
	}
};

//============================================================================

float SPAWNFLAG_SUPERSPIKE	= 1;
float SPAWNFLAG_LASER = 2;

void(vector org, vector vec) LaunchLaser;

void() spikeshooter_use =
{
	if (self.spawnflags & SPAWNFLAG_LASER)
	{
		sound (self, CHAN_VOICE, "enforcer/enfire.wav", 1, ATTN_NORM);
		LaunchLaser (self.origin, self.movedir);
	}
	else
	{
		sound (self, CHAN_VOICE, "weapons/spike2.wav", 1, ATTN_NORM);
		launch_spike (self.origin, self.movedir);
```

```
		newmis.velocity = self.movedir * 500;
		if (self.spawnflags & SPAWNFLAG_SUPERSPIKE)
			newmis.touch = superspike_touch;
	}
};

void() shooter_think =
{
	spikeshooter_use ();
	self.nextthink = time + self.wait;
	newmis.velocity = self.movedir * 500;
};


/*QUAKED trap_spikeshooter (0 .5 .8) (-8 -8 -8) (8 8 8) superspike laser
When triggered, fires a spike in the direction set in QuakeEd.
Laser is only for REGISTERED.
*/

void() trap_spikeshooter =
{
	SetMovedir ();
	self.use = spikeshooter_use;
	if (self.spawnflags & SPAWNFLAG_LASER)
	{
		precache_model2 ("progs/laser.mdl");

		precache_sound2 ("enforcer/enfire.wav");
		precache_sound2 ("enforcer/enfstop.wav");
	}
	else
		precache_sound ("weapons/spike2.wav");
};


/*QUAKED trap_shooter (0 .5 .8) (-8 -8 -8) (8 8 8) superspike laser
Continuously fires spikes.
"wait" time between spike (1.0 default)
"nextthink" delay before firing first spike, so multiple shooters can be stagered.
*/
void() trap_shooter =
{
	trap_spikeshooter ();

	if (self.wait == 0)
		self.wait = 1;
	self.nextthink = self.nextthink + self.wait + self.ltime;
	self.think = shooter_think;
};



/*
==============================================================================


==============================================================================
*/


void() make_bubbles;
void() bubble_remove;
```

```
void() bubble_bob;

/*QUAKED air_bubbles (0 .5 .8) (-8 -8 -8) (8 8 8)

testing air bubbles
*/

void() air_bubbles =

{
	if (deathmatch)
	{
		remove (self);
		return;
	}
	precache_model ("progs/s_bubble.spr");
	self.nextthink = time + 1;
	self.think = make_bubbles;
};

void() make_bubbles =
{
local entity	bubble;

	bubble = spawn();
	setmodel (bubble, "progs/s_bubble.spr");
	setorigin (bubble, self.origin);
	bubble.movetype = MOVETYPE_NOCLIP;
	bubble.solid = SOLID_NOT;
	bubble.velocity = '0 0 15';
	bubble.nextthink = time + 0.5;
	bubble.think = bubble_bob;
	bubble.touch = bubble_remove;
	bubble.classname = "bubble";
	bubble.frame = 0;
	bubble.cnt = 0;
	setsize (bubble, '-8 -8 -8', '8 8 8');
	self.nextthink = time + random() + 0.5;
	self.think = make_bubbles;
};

void() bubble_split =
{
local entity	bubble;
	bubble = spawn();
	setmodel (bubble, "progs/s_bubble.spr");
	setorigin (bubble, self.origin);
	bubble.movetype = MOVETYPE_NOCLIP;
	bubble.solid = SOLID_NOT;
	bubble.velocity = self.velocity;
	bubble.nextthink = time + 0.5;
	bubble.think = bubble_bob;
	bubble.touch = bubble_remove;
	bubble.classname = "bubble";
	bubble.frame = 1;
	bubble.cnt = 10;
	setsize (bubble, '-8 -8 -8', '8 8 8');
	self.frame = 1;
	self.cnt = 10;
	if (self.waterlevel != 3)
		remove (self);
};
```

```
void() bubble_remove =
{
	if (other.classname == self.classname)
	{
//			dprint ("bump");
			return;
	}
	remove(self);
};

void() bubble_bob =
{
local float			rnd1, rnd2, rnd3;
local vector	vtmp1, modi;

	self.cnt = self.cnt + 1;
	if (self.cnt == 4)
		bubble_split();
	if (self.cnt == 20)
		remove(self);

	rnd1 = self.velocity_x + (-10 + (random() * 20));
	rnd2 = self.velocity_y + (-10 + (random() * 20));
	rnd3 = self.velocity_z + 10 + random() * 10;

	if (rnd1 > 10)
		rnd1 = 5;
	if (rnd1 < -10)
		rnd1 = -5;

	if (rnd2 > 10)
		rnd2 = 5;
	if (rnd2 < -10)
		rnd2 = -5;

	if (rnd3 < 10)
		rnd3 = 15;
	if (rnd3 > 30)
		rnd3 = 25;

	self.velocity_x = rnd1;
	self.velocity_y = rnd2;
	self.velocity_z = rnd3;

	self.nextthink = time + 0.5;
	self.think = bubble_bob;
};

/*~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>
~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~<~>~*/

/*QUAKED viewthing (0 .5 .8) (-8 -8 -8) (8 8 8)

Just for the debugging level.  Don't use
*/

void() viewthing =

{
	self.movetype = MOVETYPE_NONE;
	self.solid = SOLID_NOT;
```

```
	precache_model ("progs/player.mdl");
	setmodel (self, "progs/player.mdl");
};


/*
==============================================================================

SIMPLE BMODELS

==============================================================================
*/

void() func_wall_use =
{	// change to alternate textures
	self.frame = 1 - self.frame;
};

/*QUAKED func_wall (0 .5 .8) ?
This is just a solid wall if not inhibitted
*/
void() func_wall =
{
	self.angles = '0 0 0';
	self.movetype = MOVETYPE_PUSH;	// so it doesn't get pushed by anything
	self.solid = SOLID_BSP;
	self.use = func_wall_use;
	setmodel (self, self.model);
};


/*QUAKED func_illusionary (0 .5 .8) ?
A simple entity that looks solid but lets you walk through it.
*/
void() func_illusionary =

{
	self.angles = '0 0 0';
	self.movetype = MOVETYPE_NONE;
	self.solid = SOLID_NOT;
	setmodel (self, self.model);
	makestatic ();
};

/*QUAKED func_episodegate (0 .5 .8) ? E1 E2 E3 E4
This bmodel will appear if the episode has allready been completed, so players can't reenter it.
*/
void() func_episodegate =

{
	if (!(serverflags & self.spawnflags))
		return;			// can still enter episode

	self.angles = '0 0 0';
	self.movetype = MOVETYPE_PUSH;	// so it doesn't get pushed by anything
	self.solid = SOLID_BSP;
	self.use = func_wall_use;
	setmodel (self, self.model);
};

/*QUAKED func_bossgate (0 .5 .8) ?
This bmodel appears unless players have all of the episode sigils.
```

```
*/
void() func_bossgate =

{
	if ( (serverflags & 15) == 15)
		return;		// all episodes completed
	self.angles = '0 0 0';
	self.movetype = MOVETYPE_PUSH;	// so it doesn't get pushed by anything
	self.solid = SOLID_BSP;
	self.use = func_wall_use;
	setmodel (self, self.model);
};

//============================================================================
/*QUAKED ambient_suck_wind (0.3 0.1 0.6) (-10 -10 -8) (10 10 8)
*/
void() ambient_suck_wind =
{
	precache_sound ("ambience/suck1.wav");
	ambientsound (self.origin, "ambience/suck1.wav", 1, ATTN_STATIC);
};

/*QUAKED ambient_drone (0.3 0.1 0.6) (-10 -10 -8) (10 10 8)
*/
void() ambient_drone =
{
	precache_sound ("ambience/drone6.wav");
	ambientsound (self.origin, "ambience/drone6.wav", 0.5, ATTN_STATIC);
};

/*QUAKED ambient_flouro_buzz (0.3 0.1 0.6) (-10 -10 -8) (10 10 8)
*/
void() ambient_flouro_buzz =
{
	precache_sound ("ambience/buzz1.wav");
	ambientsound (self.origin, "ambience/buzz1.wav", 1, ATTN_STATIC);
};
/*QUAKED ambient_drip (0.3 0.1 0.6) (-10 -10 -8) (10 10 8)
*/
void() ambient_drip =
{
	precache_sound ("ambience/drip1.wav");
	ambientsound (self.origin, "ambience/drip1.wav", 0.5, ATTN_STATIC);
};
/*QUAKED ambient_comp_hum (0.3 0.1 0.6) (-10 -10 -8) (10 10 8)
*/
void() ambient_comp_hum =
{
	precache_sound ("ambience/comp1.wav");
	ambientsound (self.origin, "ambience/comp1.wav", 1, ATTN_STATIC);
};
/*QUAKED ambient_thunder (0.3 0.1 0.6) (-10 -10 -8) (10 10 8)
*/
void() ambient_thunder =
{
	precache_sound ("ambience/thunder1.wav");
	ambientsound (self.origin, "ambience/thunder1.wav", 0.5, ATTN_STATIC);
};
/*QUAKED ambient_light_buzz (0.3 0.1 0.6) (-10 -10 -8) (10 10 8)
*/
void() ambient_light_buzz =
{
```

```
	precache_sound ("ambience/fl_hum1.wav");
	ambientsound (self.origin, "ambience/fl_hum1.wav", 0.5, ATTN_STATIC);
};
/*QUAKED ambient_swamp1 (0.3 0.1 0.6) (-10 -10 -8) (10 10 8)
*/
void() ambient_swamp1 =
{
	precache_sound ("ambience/swamp1.wav");
	ambientsound (self.origin, "ambience/swamp1.wav", 0.5, ATTN_STATIC);
};
/*QUAKED ambient_swamp2 (0.3 0.1 0.6) (-10 -10 -8) (10 10 8)
*/
void() ambient_swamp2 =
{
	precache_sound ("ambience/swamp2.wav");
	ambientsound (self.origin, "ambience/swamp2.wav", 0.5, ATTN_STATIC);
};

//==============================================================================

void() noise_think =
{
	self.nextthink = time + 0.5;
	sound (self, 1, "enforcer/enfire.wav", 1, ATTN_NORM);
	sound (self, 2, "enforcer/enfstop.wav", 1, ATTN_NORM);
	sound (self, 3, "enforcer/sight1.wav", 1, ATTN_NORM);
	sound (self, 4, "enforcer/sight2.wav", 1, ATTN_NORM);
	sound (self, 5, "enforcer/sight3.wav", 1, ATTN_NORM);
	sound (self, 6, "enforcer/sight4.wav", 1, ATTN_NORM);
	sound (self, 7, "enforcer/pain1.wav", 1, ATTN_NORM);
};

/*QUAKED misc_noisemaker (1 0.5 0) (-10 -10 -10) (10 10 10)

For optimzation testing, starts a lot of sounds.
*/

void() misc_noisemaker =

{
	precache_sound2 ("enforcer/enfire.wav");
	precache_sound2 ("enforcer/enfstop.wav");
	precache_sound2 ("enforcer/sight1.wav");
	precache_sound2 ("enforcer/sight2.wav");
	precache_sound2 ("enforcer/sight3.wav");
	precache_sound2 ("enforcer/sight4.wav");
	precache_sound2 ("enforcer/pain1.wav");
	precache_sound2 ("enforcer/pain2.wav");
	precache_sound2 ("enforcer/death1.wav");
	precache_sound2 ("enforcer/idle1.wav");

	self.nextthink = time + 0.1 + random();
	self.think = noise_think;
};
```

MODELS.QC

```
/*
==============================================================================

WORLD WEAPONS

==============================================================================
*/

$modelname g_shot
$cd id1/models/g_shot
$origin 0 0 -24
$flags 8            // client side rotate
$base base
$skin skin
$frame shot1


$modelname g_nail
$cd id1/models/g_nail
$flags 8            // client side rotate
$origin 0 0 -24
$base base
$skin skin
$frame shot1


$modelname g_nail2
$cd id1/models/g_nail2
$flags 8            // client side rotate
$origin 0 0 -24
$base base
$skin skin
$frame shot2


$modelname g_rock
$cd id1/models/g_rock
$flags 8            // client side rotate
$origin 0 0 -24
$base base
$skin skin
$frame shot1


$modelname g_rock2
$cd id1/models/g_rock2
$flags 8            // client side rotate
$origin 0 0 -24
$base base
$skin skin
$frame shot1

$modelname g_light
$cd id1/models/g_light
$flags 8            // client side rotate
$origin 0 0 -24
$base base
$skin skin
$frame shot1
```

```
/*
==============================================================================

VIEW WEAPONS

==============================================================================
*/

$modelname v_axe
$cd id1/models/v_axe
$origin 0 5 54
$base base
$skin skin
$frame frame1 frame2 frame3 frame4 frame5 frame6 frame7 frame8 frame9


$modelname v_shot
$cd id1/models/v_shot
$origin 0 0 54
$base base
$skin skin
$frame shot1 shot2 shot3 shot4 shot5 shot6 shot7


$modelname v_shot2
$cd id1/models/v_shot2
$origin 0 0 56
$base base
$skin skin
$frame shot1 shot2 shot3 shot4 shot5 shot6 shot7


$modelname v_rock2
$cd id1/models/v_rock2
$origin 0 0 54
$base base
$skin skin
$frame shot1 shot2 shot3 shot4 shot5 shot6 shot6


$modelname v_rock
$cd id1/models/v_rock
$origin 0 0 54
$base base
$skin skin
$frame shot1 shot2 shot3 shot4 shot5 shot6 shot7


$modelname v_nail2
$cd id1/models/v_nail2
$origin 0 0 54
$base base
$skin skin
$frame shot1 shot2 shot3 shot4 shot5 shot6 shot7 shot8 shot9


$modelname v_nail
$cd id1/models/v_nail
$origin 0 0 54
$base base
$skin skin
```

```
$frame shot1 shot2 shot3 shot4 shot5 shot6 shot7 shot8 shot9

$modelname v_light
$cd id1/models/v_light
$origin 0 0 54
$base base
$skin skin
$frame shot1 shot2 shot3 shot4 shot5


/*
==============================================================================

ITEMS

==============================================================================
*/

$modelname w_g_key
$cd id1/models/w_g_key
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1

$modelname w_s_key
$cd id1/models/w_s_key
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1

$modelname m_g_key
$cd id1/models/m_g_key
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1

$modelname m_s_key
$cd id1/models/m_s_key
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1

$modelname b_g_key
$cd id1/models/b_g_key
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1

$modelname b_s_key
$cd id1/models/b_s_key
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1


$modelname quaddama
```

```
$cd id1/models/quaddama
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1

$modelname invisibl
$cd id1/models/invisibl
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1

$modelname invulner
$flags 8            // client side rotate
$cd id1/models/invulner
$base base
$skin skin
$frame frame1

//modelname jetpack
//cd id1/models/jetpack
//flags 8           // client side rotate
//base base
//skin skin
//frame frame1

$modelname cube
$cd id1/models/cube
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1

$modelname suit
$cd id1/models/suit
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1

$modelname boots
$cd id1/models/boots
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1

$modelname end1
$cd id1/models/end1
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1

$modelname end2
$cd id1/models/end2
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1
```

```
$modelname end3
$cd id1/models/end3
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1

$modelname end4
$cd id1/models/end4
$flags 8            // client side rotate
$base base
$skin skin
$frame frame1


/*
==============================================================================

GIBS

==============================================================================
*/

$modelname gib1
$cd id1/models/gib1
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1


// torso
$modelname gib2
$cd id1/models/gib2
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname gib3
$cd id1/models/gib3
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1


// heads

$modelname h_player
$cd id1/models/h_player
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname h_dog
$cd id1/models/h_dog
```

```
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname h_mega
$cd id1/models/h_mega
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname h_guard
$cd id1/models/h_guard
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname h_wizard
$cd id1/models/h_wizard
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname h_knight
$cd id1/models/h_knight
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname h_hellkn
$cd id1/models/h_hellkn
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname h_zombie
$cd id1/models/h_zombie
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname h_shams
$cd id1/models/h_shams
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1
```

```
$modelname h_shal
$cd id1/models/h_shal
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname h_ogre
$cd id1/models/h_ogre
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname h_demon
$cd id1/models/h_demon
$flags 4            // EF_GIB
$origin 0 0 0
$base base
$skin skin
$frame frame1

/*
==============================================================================

MISC

==============================================================================
*/

$modelname armor
$cd id1/models/armor
$flags 8            // client side rotate
$origin 0 0 -8
$base base
$skin skin
$skin skin2
$skin skin3
$frame armor

$modelname s_light                  // shambler lightning ready
$cd id1/models/s_light
$origin 0 0 24
$base base
$skin skin
$frame  frame1 frame2 frame3

$modelname bolt3                    // lightning towar bolts
$cd id1/models/bolt2
$origin 0 0 0
$base base
$scale 4
$skin skin
$frame  light

$modelname bolt2
$cd id1/models/bolt2
$origin 0 0 0
$base base
$skin skin
```

```
$frame  light

$modelname bolt
$cd id1/models/bolt
$origin 0 0 0
$base light
$skin light
$frame  light

$modelname laser
$cd id1/models/laser
$base base
$skin skin
$scale 2
$frame frame1

$modelname flame                // with torch
$cd id1/models/flame
$origin 0 0 12
$base base
$skin skin
$framegroupstart
$frame flame1   0.1
$frame flame2   0.1
$frame flame3   0.1
$frame flame4   0.1
$frame flame5   0.1
$frame flame6   0.1
$framegroupend

$modelname flame2               // standing flame, no torch
$cd id1/models/flame2
$origin 0 0 12
$base base
$skin skin
$framegroupstart
$frame flame1   0.1
$frame flame2   0.1
$frame flame3   0.1
$frame flame4   0.1
$frame flame5   0.1
$frame flame6   0.1
$framegroupend
$framegroupstart
$frame flameb1
$frame flameb2
$frame flameb3
$frame flameb4
$frame flameb5
$frame flameb6
$frame flameb7
$frame flameb8
$frame flameb9
$frame flameb10
$frame flameb11
$framegroupend

$modelname zom_gib
$cd id1/models/zom_gib
$flags 32                   // EF_ZOMGIB
$base base
$skin skin
```

```
$frame frame1

$modelname eyes
$cd id1/models/eyes
$origin 0 0 -24
$base base
$skin skin
$frame frame1

$modelname spike
$cd id1/models/spike
$origin 0 0 0
$base spike
$skin skin
$frame spike

$modelname s_spike
$cd id1/models/s_spike
$origin 0 0 0
$base spike
$skin skin
$frame spike

$modelname v_spike
$cd id1/models/v_spike
$flags 128                  // EF_TRACER3
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname w_spike
$cd id1/models/w_spike
$flags 16                   // EF_TRACER
$origin 0 0 0
$base base
$skin skin
$framegroupstart
$frame frame1   0.1
$frame frame2   0.1
$frame frame3   0.1
$frame frame4   0.1
$framegroupend

$modelname k_spike
$cd id1/models/k_spike
$flags 64                   // EF_TRACER2
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname backpack
$cd id1/models/backpack
$flags 8            // EF_ROTATE
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname grenade
$cd id1/models/grenade2
```

```
$flags 2            // EF_GRENADE
$origin 0 0 0
$base base
$skin skin
$frame grenade

$modelname missile
$cd id1/models/missile
$flags   1                          // EF_ROCKET
$origin 0 0 0
$base base
$skin skin
$frame missile

$modelname lavaball
$cd id1/models/lavaball
$flags   1                          // EF_ROCKET
$origin 0 0 0
$base base
$skin skin
$frame frame1

$modelname teleport
$cd id1/models/teleport
$origin 0 0 24
$base base
$skin skin
$frame frame1
```

MONSTERS.QC

```
/* ALL MONSTERS SHOULD BE 1 0 0 IN COLOR */

// name =[framenum,      nexttime, nextthink] {code}
// expands to:
// name ()
// {
//                      self.frame=framenum;
//                      self.nextthink = time + nexttime;
//                      self.think = nextthink
//                      <code>
// };


/*
================
monster_use

Using a monster makes it angry at the current activator
================
*/
void() monster_use =
{
        if (self.enemy)
                return;
        if (self.health <= 0)
                return;
        if (activator.items & IT_INVISIBILITY)
                return;
        if (activator.flags & FL_NOTARGET)
                return;
        if (activator.classname != "player")
                return;

// delay reaction so if the monster is teleported, its sound is still
// heard
        self.enemy = activator;
        self.nextthink = time + 0.1;
        self.think = FoundTarget;
};

/*
================
monster_death_use

When a mosnter dies, it fires all of its targets with the current
enemy as activator.
================
*/
void() monster_death_use =
{
        local entity        ent, otemp, stemp;

// fall to ground
        if (self.flags & FL_FLY)
                self.flags = self.flags - FL_FLY;
        if (self.flags & FL_SWIM)
                self.flags = self.flags - FL_SWIM;

        if (!self.target)
                return;
```

```
	activator = self.enemy;
	SUB_UseTargets ();
};


//============================================================================

void() walkmonster_start_go =
{
local string	stemp;
local entity	etemp;

	self.origin_z = self.origin_z + 1;	// raise off floor a bit
	droptofloor();

	if (!walkmove(0,0))
	{
		dprint ("walkmonster in wall at: ");
		dprint (vtos(self.origin));
		dprint ("\n");
	}

	self.takedamage = DAMAGE_AIM;

	self.ideal_yaw = self.angles * '0 1 0';
	if (!self.yaw_speed)
		self.yaw_speed = 20;
	self.view_ofs = '0 0 25';
	self.use = monster_use;

	self.flags = self.flags | FL_MONSTER;

	if (self.target)
	{
		self.goalentity = self.movetarget = find(world, targetname, self.target);
		self.ideal_yaw = vectoyaw(self.goalentity.origin - self.origin);
		if (!self.movetarget)
		{
			dprint ("Monster can't find target at ");
			dprint (vtos(self.origin));
			dprint ("\n");
		}
// this used to be an objerror
		if (self.movetarget.classname == "path_corner")
			self.th_walk ();
		else
			self.pausetime = 99999999;
			self.th_stand ();
	}
	else
	{
		self.pausetime = 99999999;
		self.th_stand ();
	}

// spread think times so they don't all happen at same time
	self.nextthink = self.nextthink + random()*0.5;
};


void() walkmonster_start =
```

```
{
// delay drop to floor to make sure all doors have been spawned
// spread think times so they don't all happen at same time
	self.nextthink = self.nextthink + random()*0.5;
	self.think = walkmonster_start_go;
	total_monsters = total_monsters + 1;
};



void() flymonster_start_go =
{
	self.takedamage = DAMAGE_AIM;

	self.ideal_yaw = self.angles * '0 1 0';
	if (!self.yaw_speed)
		self.yaw_speed = 10;
	self.view_ofs = '0 0 25';
	self.use = monster_use;

	self.flags = self.flags | FL_FLY;
	self.flags = self.flags | FL_MONSTER;

	if (!walkmove(0,0))
	{
		dprint ("flymonster in wall at: ");
		dprint (vtos(self.origin));
		dprint ("\n");
	}

	if (self.target)
	{
		self.goalentity = self.movetarget = find(world, targetname, self.target);
		if (!self.movetarget)
		{
			dprint ("Monster can't find target at ");
			dprint (vtos(self.origin));
			dprint ("\n");
		}
// this used to be an objerror
		if (self.movetarget.classname == "path_corner")
			self.th_walk ();
		else
			self.pausetime = 99999999;
			self.th_stand ();
	}
	else
	{
		self.pausetime = 99999999;
		self.th_stand ();
	}
};

void() flymonster_start =
{
// spread think times so they don't all happen at same time
	self.nextthink = self.nextthink + random()*0.5;
	self.think = flymonster_start_go;
	total_monsters = total_monsters + 1;
};
```

```
void() swimmonster_start_go =
{
	if (deathmatch)
	{
		remove(self);
		return;
	}

	self.takedamage = DAMAGE_AIM;
	total_monsters = total_monsters + 1;

	self.ideal_yaw = self.angles * '0 1 0';
	if (!self.yaw_speed)
		self.yaw_speed = 10;
	self.view_ofs = '0 0 10';
	self.use = monster_use;

	self.flags = self.flags | FL_SWIM;
	self.flags = self.flags | FL_MONSTER;

	if (self.target)
	{
		self.goalentity = self.movetarget = find(world, targetname, self.target);
		if (!self.movetarget)
		{
			dprint ("Monster can't find target at ");
			dprint (vtos(self.origin));
			dprint ("\n");
		}
// this used to be an objerror
		self.ideal_yaw = vectoyaw(self.goalentity.origin - self.origin);
		self.th_walk ();
	}
	else
	{
		self.pausetime = 99999999;
		self.th_stand ();
	}

// spread think times so they don't all happen at same time
	self.nextthink = self.nextthink + random()*0.5;
};

void() swimmonster_start =
{
// spread think times so they don't all happen at same time
	self.nextthink = self.nextthink + random()*0.5;
	self.think = swimmonster_start_go;
	total_monsters = total_monsters + 1;
};
```

```
void() plat_center_touch;
void() plat_outside_touch;
void() plat_trigger_use;
void() plat_go_up;
void() plat_go_down;
void() plat_crush;
float PLAT_LOW_TRIGGER = 1;

void() plat_spawn_inside_trigger =
{
	local entity	trigger;
	local vector	tmin, tmax;

//
// middle trigger
//
	trigger = spawn();
	trigger.touch = plat_center_touch;
	trigger.movetype = MOVETYPE_NONE;
	trigger.solid = SOLID_TRIGGER;
	trigger.enemy = self;

	tmin = self.mins + '25 25 0';
	tmax = self.maxs - '25 25 -8';
	tmin_z = tmax_z - (self.pos1_z - self.pos2_z + 8);
	if (self.spawnflags & PLAT_LOW_TRIGGER)
		tmax_z = tmin_z + 8;

	if (self.size_x <= 50)
	{
		tmin_x = (self.mins_x + self.maxs_x) / 2;
		tmax_x = tmin_x + 1;
	}
	if (self.size_y <= 50)
	{
		tmin_y = (self.mins_y + self.maxs_y) / 2;
		tmax_y = tmin_y + 1;
	}

	setsize (trigger, tmin, tmax);
};

void() plat_hit_top =
{
	sound (self, CHAN_VOICE, self.noise1, 1, ATTN_NORM);
	self.state = STATE_TOP;
	self.think = plat_go_down;
	self.nextthink = self.ltime + 3;
};

void() plat_hit_bottom =
{
	sound (self, CHAN_VOICE, self.noise1, 1, ATTN_NORM);
	self.state = STATE_BOTTOM;
};

void() plat_go_down =
{
```

```
	sound (self, CHAN_VOICE, self.noise, 1, ATTN_NORM);
	self.state = STATE_DOWN;
	SUB_CalcMove (self.pos2, self.speed, plat_hit_bottom);
};

void() plat_go_up =
{
	sound (self, CHAN_VOICE, self.noise, 1, ATTN_NORM);
	self.state = STATE_UP;
	SUB_CalcMove (self.pos1, self.speed, plat_hit_top);
};

void() plat_center_touch =
{
	if (other.classname != "player")
		return;

	if (other.health <= 0)
		return;

	self = self.enemy;
	if (self.state == STATE_BOTTOM)
		plat_go_up ();
	else if (self.state == STATE_TOP)
		self.nextthink = self.ltime + 1;	// delay going down
};

void() plat_outside_touch =
{
	if (other.classname != "player")
		return;

	if (other.health <= 0)
		return;

//dprint ("plat_outside_touch\n");
	self = self.enemy;
	if (self.state == STATE_TOP)
		plat_go_down ();
};

void() plat_trigger_use =
{
	if (self.think)
		return;		// allready activated
	plat_go_down();
};


void() plat_crush =
{
//dprint ("plat_crush\n");

	T_Damage (other, self, self, 1);

	if (self.state == STATE_UP)
		plat_go_down ();
	else if (self.state == STATE_DOWN)
		plat_go_up ();
	else
		objerror ("plat_crush: bad self.state\n");
};
```

```
void() plat_use =
{
	self.use = SUB_Null;
	if (self.state != STATE_UP)
		objerror ("plat_use: not in up state");
	plat_go_down();
};


/*QUAKED func_plat (0 .5 .8) ? PLAT_LOW_TRIGGER
speed	default 150

Plats are always drawn in the extended position, so they will light correctly.

If the plat is the target of another trigger or button, it will start out disabled in the extended position until it is trigger, when it will lower and become a normal plat.

If the "height" key is set, that will determine the amount the plat moves, instead of being implicitly determined by the model's height.
Set "sounds" to one of the following:
1) base fast
2) chain slow
*/


void() func_plat =

{
local entity t;

	if (!self.t_length)
		self.t_length = 80;
	if (!self.t_width)
		self.t_width = 10;

	if (self.sounds == 0)
		self.sounds = 2;
// FIX THIS TO LOAD A GENERIC PLAT SOUND

	if (self.sounds == 1)
	{
		precache_sound ("plats/plat1.wav");
		precache_sound ("plats/plat2.wav");
		self.noise = "plats/plat1.wav";
		self.noise1 = "plats/plat2.wav";
	}

	if (self.sounds == 2)
	{
		precache_sound ("plats/medplat1.wav");
		precache_sound ("plats/medplat2.wav");
		self.noise = "plats/medplat1.wav";
		self.noise1 = "plats/medplat2.wav";
	}


	self.mangle = self.angles;
	self.angles = '0 0 0';

	self.classname = "plat";
	self.solid = SOLID_BSP;
```

```
	self.movetype = MOVETYPE_PUSH;
	setorigin (self, self.origin);
	setmodel (self, self.model);
	setsize (self, self.mins , self.maxs);

	self.blocked = plat_crush;
	if (!self.speed)
		self.speed = 150;

// pos1 is the top position, pos2 is the bottom
	self.pos1 = self.origin;
	self.pos2 = self.origin;
	if (self.height)
		self.pos2_z = self.origin_z - self.height;
	else
		self.pos2_z = self.origin_z - self.size_z + 8;

	self.use = plat_trigger_use;

	plat_spawn_inside_trigger ();	// the "start moving" trigger

	if (self.targetname)
	{
		self.state = STATE_UP;
		self.use = plat_use;
	}
	else
	{
		setorigin (self, self.pos2);
		self.state = STATE_BOTTOM;
	}
};

//============================================================================

void() train_next;
void() func_train_find;

void() train_blocked =
{
	if (time < self.attack_finished)
		return;
	self.attack_finished = time + 0.5;
	T_Damage (other, self, self, self.dmg);
};
void() train_use =
{
	if (self.think != func_train_find)
		return;		// already activated
	train_next();
};

void() train_wait =
{
	if (self.wait)
	{
		self.nextthink = self.ltime + self.wait;
		sound (self, CHAN_VOICE, self.noise, 1, ATTN_NORM);
	}
	else
		self.nextthink = self.ltime + 0.1;
```

```
	self.think = train_next;
};

void() train_next =
{
	local entity	targ;

	targ = find (world, targetname, self.target);
	self.target = targ.target;
	if (!self.target)
		objerror ("train_next: no next target");
	if (targ.wait)
		self.wait = targ.wait;
	else
		self.wait = 0;
	sound (self, CHAN_VOICE, self.noise1, 1, ATTN_NORM);
	SUB_CalcMove (targ.origin - self.mins, self.speed, train_wait);
};

void() func_train_find =

{
	local entity	targ;

	targ = find (world, targetname, self.target);
	self.target = targ.target;
	setorigin (self, targ.origin - self.mins);
	if (!self.targetname)
	{	// not triggered, so start immediately
		self.nextthink = self.ltime + 0.1;
		self.think = train_next;
	}
};

/*QUAKED func_train (0 .5 .8) ?
Trains are moving platforms that players can ride.
The targets origin specifies the min point of the train at each corner.
The train spawns at the first target it is pointing at.
If the train is the target of a button or trigger, it will not begin moving until activated.
speed   default 100
dmg		default  2
sounds
1) ratchet metal

*/
void() func_train =
{
	if (!self.speed)
		self.speed = 100;
	if (!self.target)
		objerror ("func_train without a target");
	if (!self.dmg)
		self.dmg = 2;

	if (self.sounds == 0)
	{
		self.noise = ("misc/null.wav");
		precache_sound ("misc/null.wav");
		self.noise1 = ("misc/null.wav");
		precache_sound ("misc/null.wav");
	}
```

```
	if (self.sounds == 1)
	{
		self.noise = ("plats/train2.wav");
		precache_sound ("plats/train2.wav");
		self.noise1 = ("plats/train1.wav");
		precache_sound ("plats/train1.wav");
	}

	self.cnt = 1;
	self.solid = SOLID_BSP;
	self.movetype = MOVETYPE_PUSH;
	self.blocked = train_blocked;
	self.use = train_use;
	self.classname = "train";

	setmodel (self, self.model);
	setsize (self, self.mins , self.maxs);
	setorigin (self, self.origin);

// start trains on the second frame, to make sure their targets have had
// a chance to spawn
	self.nextthink = self.ltime + 0.1;
	self.think = func_train_find;
};

/*QUAKED misc_teleporttrain (0 .5 .8) (-8 -8 -8) (8 8 8)
This is used for the final bos
*/
void() misc_teleporttrain =
{
	if (!self.speed)
		self.speed = 100;
	if (!self.target)
		objerror ("func_train without a target");

	self.cnt = 1;
	self.solid = SOLID_NOT;
	self.movetype = MOVETYPE_PUSH;
	self.blocked = train_blocked;
	self.use = train_use;
	self.avelocity = '100 200 300';

	self.noise = ("misc/null.wav");
	precache_sound ("misc/null.wav");
	self.noise1 = ("misc/null.wav");
	precache_sound ("misc/null.wav");

	precache_model2 ("progs/teleport.mdl");
	setmodel (self, "progs/teleport.mdl");
	setsize (self, self.mins , self.maxs);
	setorigin (self, self.origin);

// start trains on the second frame, to make sure their targets have had
// a chance to spawn
	self.nextthink = self.ltime + 0.1;
	self.think = func_train_find;
};
```

PLAYER.QC

```
void() bubble_bob;

/*
==============================================================================

PLAYER

==============================================================================
*/

$cd id1/models/player_4
$origin 0 -6 24
$base base
$skin skin

//
// running
//
$frame axrun1 axrun2 axrun3 axrun4 axrun5 axrun6

$frame rockrun1 rockrun2 rockrun3 rockrun4 rockrun5 rockrun6

//
// standing
//
$frame stand1 stand2 stand3 stand4 stand5

$frame axstnd1 axstnd2 axstnd3 axstnd4 axstnd5 axstnd6
$frame axstnd7 axstnd8 axstnd9 axstnd10 axstnd11 axstnd12


//
// pain
//
$frame axpain1 axpain2 axpain3 axpain4 axpain5 axpain6

$frame pain1 pain2 pain3 pain4 pain5 pain6


//
// death
//

$frame axdeth1 axdeth2 axdeth3 axdeth4 axdeth5 axdeth6
$frame axdeth7 axdeth8 axdeth9

$frame deatha1 deatha2 deatha3 deatha4 deatha5 deatha6 deatha7 deatha8
$frame deatha9 deatha10 deatha11

$frame deathb1 deathb2 deathb3 deathb4 deathb5 deathb6 deathb7 deathb8
$frame deathb9

$frame deathc1 deathc2 deathc3 deathc4 deathc5 deathc6 deathc7 deathc8
$frame deathc9 deathc10 deathc11 deathc12 deathc13 deathc14 deathc15

$frame deathd1 deathd2 deathd3 deathd4 deathd5 deathd6 deathd7
$frame deathd8 deathd9

$frame deathe1 deathe2 deathe3 deathe4 deathe5 deathe6 deathe7
```

```
$frame deathe8 deathe9

//
// attacks
//
$frame nailatt1 nailatt2

$frame light1 light2

$frame rockatt1 rockatt2 rockatt3 rockatt4 rockatt5 rockatt6

$frame shotatt1 shotatt2 shotatt3 shotatt4 shotatt5 shotatt6

$frame axatt1 axatt2 axatt3 axatt4 axatt5 axatt6

$frame axattb1 axattb2 axattb3 axattb4 axattb5 axattb6

$frame axattc1 axattc2 axattc3 axattc4 axattc5 axattc6

$frame axattd1 axattd2 axattd3 axattd4 axattd5 axattd6


/*
==============================================================================
PLAYER
==============================================================================
*/

void() player_run;

void()  player_stand1 =[        $axstnd1,       player_stand1   ]
{
        self.weaponframe=0;
        if (self.velocity_x || self.velocity_y)
        {
                self.walkframe=0;
                player_run();
                return;
        }

        if (self.weapon == IT_AXE)
        {
                if (self.walkframe >= 12)
                        self.walkframe = 0;
                self.frame = $axstnd1 + self.walkframe;
        }
        else
        {
                if (self.walkframe >= 5)
                        self.walkframe = 0;
                self.frame = $stand1 + self.walkframe;
        }
        self.walkframe = self.walkframe + 1;
};

void()  player_run =[   $rockrun1,      player_run      ]
{
        self.weaponframe=0;
        if (!self.velocity_x && !self.velocity_y)
        {
                self.walkframe=0;
                player_stand1();
```

```
		return;
	}

	if (self.weapon == IT_AXE)
	{
		if (self.walkframe == 6)
			self.walkframe = 0;
		self.frame = $axrun1 + self.walkframe;
	}
	else
	{
		if (self.walkframe == 6)
			self.walkframe = 0;
		self.frame = self.frame + self.walkframe;
	}
	self.walkframe = self.walkframe + 1;
};


void()	player_shot1 =	[$shotatt1, player_shot2 ] {self.weaponframe=1;
self.effects = self.effects | EF_MUZZLEFLASH;};
void()	player_shot2 =	[$shotatt2, player_shot3 ] {self.weaponframe=2;};
void()	player_shot3 =	[$shotatt3, player_shot4 ] {self.weaponframe=3;};
void()	player_shot4 =	[$shotatt4, player_shot5 ] {self.weaponframe=4;};
void()	player_shot5 =	[$shotatt5, player_shot6 ] {self.weaponframe=5;};
void()	player_shot6 =	[$shotatt6, player_run	 ] {self.weaponframe=6;};

void()	player_axe1 =	[$axatt1, player_axe2	 ] {self.weaponframe=1;};
void()	player_axe2 =	[$axatt2, player_axe3	 ] {self.weaponframe=2;};
void()	player_axe3 =	[$axatt3, player_axe4	 ] {self.weaponframe=3;W_FireAxe();};
void()	player_axe4 =	[$axatt4, player_run	 ] {self.weaponframe=4;};

void()	player_axeb1 =	[$axattb1, player_axeb2 ] {self.weaponframe=5;};
void()	player_axeb2 =	[$axattb2, player_axeb3 ] {self.weaponframe=6;};
void()	player_axeb3 =	[$axattb3, player_axeb4 ] {self.weaponframe=7;W_FireAxe();};
void()	player_axeb4 =	[$axattb4, player_run	 ] {self.weaponframe=8;};

void()	player_axec1 =	[$axattc1, player_axec2 ] {self.weaponframe=1;};
void()	player_axec2 =	[$axattc2, player_axec3 ] {self.weaponframe=2;};
void()	player_axec3 =	[$axattc3, player_axec4 ] {self.weaponframe=3;W_FireAxe();};
void()	player_axec4 =	[$axattc4, player_run	 ] {self.weaponframe=4;};

void()	player_axed1 =	[$axattd1, player_axed2 ] {self.weaponframe=5;};
void()	player_axed2 =	[$axattd2, player_axed3 ] {self.weaponframe=6;};
void()	player_axed3 =	[$axattd3, player_axed4 ] {self.weaponframe=7;W_FireAxe();};
void()	player_axed4 =	[$axattd4, player_run	 ] {self.weaponframe=8;};


//============================================================================

void() player_nail1	=[$nailatt1, player_nail2  ]
{
	self.effects = self.effects | EF_MUZZLEFLASH;

	if (!self.button0)
		{player_run ();return;}
	self.weaponframe = self.weaponframe + 1;
	if (self.weaponframe == 9)
		self.weaponframe = 1;
	SuperDamageSound();
	W_FireSpikes (4);
	self.attack_finished = time + 0.2;
```

```
};
void() player_nail2   =[$nailatt2, player_nail1  ]
{
	self.effects = self.effects | EF_MUZZLEFLASH;

	if (!self.button0)
		{player_run ();return;}
	self.weaponframe = self.weaponframe + 1;
	if (self.weaponframe == 9)
		self.weaponframe = 1;
	SuperDamageSound();
	W_FireSpikes (-4);
	self.attack_finished = time + 0.2;
};

//============================================================================

void() player_light1   =[$light1, player_light2  ]
{
	self.effects = self.effects | EF_MUZZLEFLASH;

	if (!self.button0)
		{player_run ();return;}
	self.weaponframe = self.weaponframe + 1;
	if (self.weaponframe == 5)
		self.weaponframe = 1;
	SuperDamageSound();
	W_FireLightning();
	self.attack_finished = time + 0.2;
};
void() player_light2   =[$light2, player_light1  ]
{
	self.effects = self.effects | EF_MUZZLEFLASH;

	if (!self.button0)
		{player_run ();return;}
	self.weaponframe = self.weaponframe + 1;
	if (self.weaponframe == 5)
		self.weaponframe = 1;
	SuperDamageSound();
	W_FireLightning();
	self.attack_finished = time + 0.2;
};

//============================================================================


void() player_rocket1   =[$rockatt1, player_rocket2  ] {self.weaponframe=1;
self.effects = self.effects | EF_MUZZLEFLASH;};
void() player_rocket2   =[$rockatt2, player_rocket3  ] {self.weaponframe=2;};
void() player_rocket3   =[$rockatt3, player_rocket4  ] {self.weaponframe=3;};
void() player_rocket4   =[$rockatt4, player_rocket5  ] {self.weaponframe=4;};
void() player_rocket5   =[$rockatt5, player_rocket6  ] {self.weaponframe=5;};
void() player_rocket6   =[$rockatt6, player_run  ] {self.weaponframe=6;};
void(float num_bubbles) DeathBubbles;

void() PainSound =
{
local float                     rs;

	if (self.health < 0)
		return;
```

```
	if (damage_attacker.classname == "teledeath")
	{
		sound (self, CHAN_VOICE, "player/teledth1.wav", 1, ATTN_NONE);
		return;
	}

// water pain sounds
	if (self.watertype == CONTENT_WATER && self.waterlevel == 3)
	{
		DeathBubbles(1);
		if (random() > 0.5)
			sound (self, CHAN_VOICE, "player/drown1.wav", 1, ATTN_NORM);
		else
			sound (self, CHAN_VOICE, "player/drown2.wav", 1, ATTN_NORM);
		return;
	}

// slime pain sounds
	if (self.watertype == CONTENT_SLIME)
	{
// FIX ME	put in some steam here
		if (random() > 0.5)
			sound (self, CHAN_VOICE, "player/lburn1.wav", 1, ATTN_NORM);
		else
			sound (self, CHAN_VOICE, "player/lburn2.wav", 1, ATTN_NORM);
		return;
	}

	if (self.watertype == CONTENT_LAVA)
	{
		if (random() > 0.5)
			sound (self, CHAN_VOICE, "player/lburn1.wav", 1, ATTN_NORM);
		else
			sound (self, CHAN_VOICE, "player/lburn2.wav", 1, ATTN_NORM);
		return;
	}

	if (self.pain_finished > time)
	{
		self.axhitme = 0;
		return;
	}
	self.pain_finished = time + 0.5;

// don't make multiple pain sounds right after each other

// ax pain sound
	if (self.axhitme == 1)
	{
		self.axhitme = 0;
		sound (self, CHAN_VOICE, "player/axhit1.wav", 1, ATTN_NORM);
		return;
	}


	rs = rint((random() * 5) + 1);

	self.noise = "";
	if (rs == 1)
		self.noise = "player/pain1.wav";
	else if (rs == 2)
```

```
                self.noise = "player/pain2.wav";
        else if (rs == 3)
                self.noise = "player/pain3.wav";
        else if (rs == 4)
                self.noise = "player/pain4.wav";
        else if (rs == 5)
                self.noise = "player/pain5.wav";
        else
                self.noise = "player/pain6.wav";

        sound (self, CHAN_VOICE, self.noise, 1, ATTN_NORM);
        return;
};

void()  player_pain1 =  [       $pain1, player_pain2    ] {PainSound();self.weaponframe=0;};
void()  player_pain2 =  [       $pain2, player_pain3    ] {};
void()  player_pain3 =  [       $pain3, player_pain4    ] {};
void()  player_pain4 =  [       $pain4, player_pain5    ] {};
void()  player_pain5 =  [       $pain5, player_pain6    ] {};
void()  player_pain6 =  [       $pain6, player_run      ] {};

void()  player_axpain1 =        [       $axpain1,       player_axpain2  ] {PainSound();self.weaponframe=0;};
void()  player_axpain2 =        [       $axpain2,       player_axpain3  ] {};
void()  player_axpain3 =        [       $axpain3,       player_axpain4  ] {};
void()  player_axpain4 =        [       $axpain4,       player_axpain5  ] {};
void()  player_axpain5 =        [       $axpain5,       player_axpain6  ] {};
void()  player_axpain6 =        [       $axpain6,       player_run      ] {};

void() player_pain =
{
        if (self.weaponframe)
                return;

        if (self.invisible_finished > time)
                return;         // eyes don't have pain frames

        if (self.weapon == IT_AXE)
                player_axpain1 ();
        else
                player_pain1 ();
};

void() player_diea1;
void() player_dieb1;
void() player_diec1;
void() player_died1;
void() player_diee1;
void() player_die_ax1;

void() DeathBubblesSpawn =
{
local entity    bubble;
        if (self.owner.waterlevel != 3)
                return;
        bubble = spawn();
        setmodel (bubble, "progs/s_bubble.spr");
        setorigin (bubble, self.owner.origin + '0 0 24');
        bubble.movetype = MOVETYPE_NOCLIP;
        bubble.solid = SOLID_NOT;
        bubble.velocity = '0 0 15';
        bubble.nextthink = time + 0.5;
        bubble.think = bubble_bob;
```

```
	bubble.classname = "bubble";
	bubble.frame = 0;
	bubble.cnt = 0;
	setsize (bubble, '-8 -8 -8', '8 8 8');
	self.nextthink = time + 0.1;
	self.think = DeathBubblesSpawn;
	self.air_finished = self.air_finished + 1;
	if (self.air_finished >= self.bubble_count)
		remove(self);
};

void(float num_bubbles) DeathBubbles =
{
local entity	bubble_spawner;

	bubble_spawner = spawn();
	setorigin (bubble_spawner, self.origin);
	bubble_spawner.movetype = MOVETYPE_NONE;
	bubble_spawner.solid = SOLID_NOT;
	bubble_spawner.nextthink = time + 0.1;
	bubble_spawner.think = DeathBubblesSpawn;
	bubble_spawner.air_finished = 0;
	bubble_spawner.owner = self;
	bubble_spawner.bubble_count = num_bubbles;
	return;
};


void() DeathSound =
{
local float			rs;

	// water death sounds
	if (self.waterlevel == 3)
	{
		DeathBubbles(20);
		sound (self, CHAN_VOICE, "player/h2odeath.wav", 1, ATTN_NONE);
		return;
	}

	rs = rint ((random() * 4) + 1);
	if (rs == 1)
		self.noise = "player/death1.wav";
	if (rs == 2)
		self.noise = "player/death2.wav";
	if (rs == 3)
		self.noise = "player/death3.wav";
	if (rs == 4)
		self.noise = "player/death4.wav";
	if (rs == 5)
		self.noise = "player/death5.wav";

	sound (self, CHAN_VOICE, self.noise, 1, ATTN_NONE);
	return;
};


void() PlayerDead =
{
	self.nextthink = -1;
// allow respawn after a certain time
	self.deadflag = DEAD_DEAD;
```

```
};

vector(float dm) VelocityForDamage =
{
	local vector v;

	v_x = 100 * crandom();
	v_y = 100 * crandom();
	v_z = 200 + 100 * random();

	if (dm > -50)
	{
//			dprint ("level 1\n");
			v = v * 0.7;
	}
	else if (dm > -200)
	{
//			dprint ("level 3\n");
			v = v * 2;
	}
	else
			v = v * 10;

	return v;
};

void(string gibname, float dm) ThrowGib =
{
	local	entity new;

	new = spawn();
	new.origin = self.origin;
	setmodel (new, gibname);
	setsize (new, '0 0 0', '0 0 0');
	new.velocity = VelocityForDamage (dm);
	new.movetype = MOVETYPE_BOUNCE;
	new.solid = SOLID_NOT;
	new.avelocity_x = random()*600;
	new.avelocity_y = random()*600;
	new.avelocity_z = random()*600;
	new.think = SUB_Remove;
	new.ltime = time;
	new.nextthink = time + 10 + random()*10;
	new.frame = 0;
	new.flags = 0;
};

void(string gibname, float dm) ThrowHead =
{
	setmodel (self, gibname);
	self.frame = 0;
	self.nextthink = -1;
	self.movetype = MOVETYPE_BOUNCE;
	self.takedamage = DAMAGE_NO;
	self.solid = SOLID_NOT;
	self.view_ofs = '0 0 8';
	setsize (self, '-16 -16 0', '16 16 56');
	self.velocity = VelocityForDamage (dm);
	self.origin_z = self.origin_z - 24;
	self.flags = self.flags - (self.flags & FL_ONGROUND);
	self.avelocity = crandom() * '0 600 0';
};
```

```
void() GibPlayer =
{
	ThrowHead ("progs/h_player.mdl", self.health);
	ThrowGib ("progs/gib1.mdl", self.health);
	ThrowGib ("progs/gib2.mdl", self.health);
	ThrowGib ("progs/gib3.mdl", self.health);

	self.deadflag = DEAD_DEAD;

	if (damage_attacker.classname == "teledeath")
	{
		sound (self, CHAN_VOICE, "player/teledth1.wav", 1, ATTN_NONE);
		return;
	}

	if (damage_attacker.classname == "teledeath2")
	{
		sound (self, CHAN_VOICE, "player/teledth1.wav", 1, ATTN_NONE);
		return;
	}

	if (random() < 0.5)
		sound (self, CHAN_VOICE, "player/gib.wav", 1, ATTN_NONE);
	else
		sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NONE);
};

void() PlayerDie =
{
	local	float	i;

	self.items = self.items - (self.items & IT_INVISIBILITY);
	self.invisible_finished = 0;		// don't die as eyes
	self.invincible_finished = 0;
	self.super_damage_finished = 0;
	self.radsuit_finished = 0;
	self.modelindex = modelindex_player;	// don't use eyes

	if (deathmatch || coop)
		DropBackpack();

	self.weaponmodel="";
	self.view_ofs = '0 0 -8';
	self.deadflag = DEAD_DYING;
	self.solid = SOLID_NOT;
	self.flags = self.flags - (self.flags & FL_ONGROUND);
	self.movetype = MOVETYPE_TOSS;
	if (self.velocity_z < 10)
		self.velocity_z = self.velocity_z + random()*300;

	if (self.health < -40)
	{
		GibPlayer ();
		return;
	}

	DeathSound();

	self.angles_x = 0;
	self.angles_z = 0;
```

```
	if (self.weapon == IT_AXE)
	{
		player_die_ax1 ();
		return;
	}

	i = cvar("temp1");
	if (!i)
		i = 1 + floor(random()*6);

	if (i == 1)
		player_diea1();
	else if (i == 2)
		player_dieb1();
	else if (i == 3)
		player_diec1();
	else if (i == 4)
		player_died1();
	else
		player_diee1();

};

void() set_suicide_frame =
{	// used by klill command and diconnect command
	if (self.model != "progs/player.mdl")
		return;	// allready gibbed
	self.frame = $deatha11;
	self.solid = SOLID_NOT;
	self.movetype = MOVETYPE_TOSS;
	self.deadflag = DEAD_DEAD;
	self.nextthink = -1;
};


void()	player_diea1	=	[	$deatha1,	player_diea2	] {};
void()	player_diea2	=	[	$deatha2,	player_diea3	] {};
void()	player_diea3	=	[	$deatha3,	player_diea4	] {};
void()	player_diea4	=	[	$deatha4,	player_diea5	] {};
void()	player_diea5	=	[	$deatha5,	player_diea6	] {};
void()	player_diea6	=	[	$deatha6,	player_diea7	] {};
void()	player_diea7	=	[	$deatha7,	player_diea8	] {};
void()	player_diea8	=	[	$deatha8,	player_diea9	] {};
void()	player_diea9	=	[	$deatha9,	player_diea10	] {};
void()	player_diea10	=	[	$deatha10,	player_diea11	] {};
void()	player_diea11	=	[	$deatha11,	player_diea11 ] {PlayerDead();};

void()	player_dieb1	=	[	$deathb1,	player_dieb2	] {};
void()	player_dieb2	=	[	$deathb2,	player_dieb3	] {};
void()	player_dieb3	=	[	$deathb3,	player_dieb4	] {};
void()	player_dieb4	=	[	$deathb4,	player_dieb5	] {};
void()	player_dieb5	=	[	$deathb5,	player_dieb6	] {};
void()	player_dieb6	=	[	$deathb6,	player_dieb7	] {};
void()	player_dieb7	=	[	$deathb7,	player_dieb8	] {};
void()	player_dieb8	=	[	$deathb8,	player_dieb9	] {};
void()	player_dieb9	=	[	$deathb9,	player_dieb9	] {PlayerDead();};

void()	player_diec1	=	[	$deathc1,	player_diec2	] {};
void()	player_diec2	=	[	$deathc2,	player_diec3	] {};
void()	player_diec3	=	[	$deathc3,	player_diec4	] {};
void()	player_diec4	=	[	$deathc4,	player_diec5	] {};
```

```
void()	player_diec5	=	[	$deathc5,	player_diec6	] {};
void()	player_diec6	=	[	$deathc6,	player_diec7	] {};
void()	player_diec7	=	[	$deathc7,	player_diec8	] {};
void()	player_diec8	=	[	$deathc8,	player_diec9	] {};
void()	player_diec9	=	[	$deathc9,	player_diec10	] {};
void()	player_diec10	=	[	$deathc10,	player_diec11	] {};
void()	player_diec11	=	[	$deathc11,	player_diec12	] {};
void()	player_diec12	=	[	$deathc12,	player_diec13	] {};
void()	player_diec13	=	[	$deathc13,	player_diec14	] {};
void()	player_diec14	=	[	$deathc14,	player_diec15	] {};
void()	player_diec15	=	[	$deathc15,	player_diec15 ] {PlayerDead();};

void()	player_died1	=	[	$deathd1,	player_died2	] {};
void()	player_died2	=	[	$deathd2,	player_died3	] {};
void()	player_died3	=	[	$deathd3,	player_died4	] {};
void()	player_died4	=	[	$deathd4,	player_died5	] {};
void()	player_died5	=	[	$deathd5,	player_died6	] {};
void()	player_died6	=	[	$deathd6,	player_died7	] {};
void()	player_died7	=	[	$deathd7,	player_died8	] {};
void()	player_died8	=	[	$deathd8,	player_died9	] {};
void()	player_died9	=	[	$deathd9,	player_died9	] {PlayerDead();};

void()	player_diee1	=	[	$deathe1,	player_diee2	] {};
void()	player_diee2	=	[	$deathe2,	player_diee3	] {};
void()	player_diee3	=	[	$deathe3,	player_diee4	] {};
void()	player_diee4	=	[	$deathe4,	player_diee5	] {};
void()	player_diee5	=	[	$deathe5,	player_diee6	] {};
void()	player_diee6	=	[	$deathe6,	player_diee7	] {};
void()	player_diee7	=	[	$deathe7,	player_diee8	] {};
void()	player_diee8	=	[	$deathe8,	player_diee9	] {};
void()	player_diee9	=	[	$deathe9,	player_diee9	] {PlayerDead();};

void()	player_die_ax1	=	[	$axdeth1,	player_die_ax2	] {};
void()	player_die_ax2	=	[	$axdeth2,	player_die_ax3	] {};
void()	player_die_ax3	=	[	$axdeth3,	player_die_ax4	] {};
void()	player_die_ax4	=	[	$axdeth4,	player_die_ax5	] {};
void()	player_die_ax5	=	[	$axdeth5,	player_die_ax6	] {};
void()	player_die_ax6	=	[	$axdeth6,	player_die_ax7	] {};
void()	player_die_ax7	=	[	$axdeth7,	player_die_ax8	] {};
void()	player_die_ax8	=	[	$axdeth8,	player_die_ax9	] {};
void()	player_die_ax9	=	[	$axdeth9,	player_die_ax9	] {PlayerDead();};
```

SPRITES.QC

```
void() bubble_bob;

/*
==============================================================================

PLAYER

==============================================================================
*/

$cd id1/models/player_4
$origin 0 -6 24
$base base
$skin skin

//
// running
//
$frame axrun1 axrun2 axrun3 axrun4 axrun5 axrun6

$frame rockrun1 rockrun2 rockrun3 rockrun4 rockrun5 rockrun6

//
// standing
//
$frame stand1 stand2 stand3 stand4 stand5

$frame axstnd1 axstnd2 axstnd3 axstnd4 axstnd5 axstnd6
$frame axstnd7 axstnd8 axstnd9 axstnd10 axstnd11 axstnd12


//
// pain
//
$frame axpain1 axpain2 axpain3 axpain4 axpain5 axpain6

$frame pain1 pain2 pain3 pain4 pain5 pain6


//
// death
//

$frame axdeth1 axdeth2 axdeth3 axdeth4 axdeth5 axdeth6
$frame axdeth7 axdeth8 axdeth9

$frame deatha1 deatha2 deatha3 deatha4 deatha5 deatha6 deatha7 deatha8
$frame deatha9 deatha10 deatha11

$frame deathb1 deathb2 deathb3 deathb4 deathb5 deathb6 deathb7 deathb8
$frame deathb9

$frame deathc1 deathc2 deathc3 deathc4 deathc5 deathc6 deathc7 deathc8
$frame deathc9 deathc10 deathc11 deathc12 deathc13 deathc14 deathc15

$frame deathd1 deathd2 deathd3 deathd4 deathd5 deathd6 deathd7
$frame deathd8 deathd9

$frame deathe1 deathe2 deathe3 deathe4 deathe5 deathe6 deathe7
```

```
$frame deathe8 deathe9

//
// attacks
//
$frame nailatt1 nailatt2

$frame light1 light2

$frame rockatt1 rockatt2 rockatt3 rockatt4 rockatt5 rockatt6

$frame shotatt1 shotatt2 shotatt3 shotatt4 shotatt5 shotatt6

$frame axatt1 axatt2 axatt3 axatt4 axatt5 axatt6

$frame axattb1 axattb2 axattb3 axattb4 axattb5 axattb6

$frame axattc1 axattc2 axattc3 axattc4 axattc5 axattc6

$frame axattd1 axattd2 axattd3 axattd4 axattd5 axattd6


/*
==============================================================================
PLAYER
==============================================================================
*/

void() player_run;

void()  player_stand1 =[        $axstnd1,       player_stand1   ]
{
        self.weaponframe=0;
        if (self.velocity_x || self.velocity_y)
        {
                self.walkframe=0;
                player_run();
                return;
        }

        if (self.weapon == IT_AXE)
        {
                if (self.walkframe >= 12)
                        self.walkframe = 0;
                self.frame = $axstnd1 + self.walkframe;
        }
        else
        {
                if (self.walkframe >= 5)
                        self.walkframe = 0;
                self.frame = $stand1 + self.walkframe;
        }
        self.walkframe = self.walkframe + 1;
};

void()  player_run =[   $rockrun1,      player_run      ]
{
        self.weaponframe=0;
        if (!self.velocity_x && !self.velocity_y)
        {
                self.walkframe=0;
                player_stand1();
```

```
		return;
	}

	if (self.weapon == IT_AXE)
	{
		if (self.walkframe == 6)
			self.walkframe = 0;
		self.frame = $axrun1 + self.walkframe;
	}
	else
	{
		if (self.walkframe == 6)
			self.walkframe = 0;
		self.frame = self.frame + self.walkframe;
	}
	self.walkframe = self.walkframe + 1;
};


void()	player_shot1 =	[$shotatt1, player_shot2 ] {self.weaponframe=1;
self.effects = self.effects | EF_MUZZLEFLASH;};
void()	player_shot2 =	[$shotatt2, player_shot3 ] {self.weaponframe=2;};
void()	player_shot3 =	[$shotatt3, player_shot4 ] {self.weaponframe=3;};
void()	player_shot4 =	[$shotatt4, player_shot5 ] {self.weaponframe=4;};
void()	player_shot5 =	[$shotatt5, player_shot6 ] {self.weaponframe=5;};
void()	player_shot6 =	[$shotatt6, player_run	 ] {self.weaponframe=6;};

void()	player_axe1 =	[$axatt1, player_axe2	 ] {self.weaponframe=1;};
void()	player_axe2 =	[$axatt2, player_axe3	 ] {self.weaponframe=2;};
void()	player_axe3 =	[$axatt3, player_axe4	 ] {self.weaponframe=3;W_FireAxe();};
void()	player_axe4 =	[$axatt4, player_run	 ] {self.weaponframe=4;};

void()	player_axeb1 =	[$axattb1, player_axeb2 ] {self.weaponframe=5;};
void()	player_axeb2 =	[$axattb2, player_axeb3 ] {self.weaponframe=6;};
void()	player_axeb3 =	[$axattb3, player_axeb4 ] {self.weaponframe=7;W_FireAxe();};
void()	player_axeb4 =	[$axattb4, player_run	 ] {self.weaponframe=8;};

void()	player_axec1 =	[$axattc1, player_axec2 ] {self.weaponframe=1;};
void()	player_axec2 =	[$axattc2, player_axec3 ] {self.weaponframe=2;};
void()	player_axec3 =	[$axattc3, player_axec4 ] {self.weaponframe=3;W_FireAxe();};
void()	player_axec4 =	[$axattc4, player_run	 ] {self.weaponframe=4;};

void()	player_axed1 =	[$axattd1, player_axed2 ] {self.weaponframe=5;};
void()	player_axed2 =	[$axattd2, player_axed3 ] {self.weaponframe=6;};
void()	player_axed3 =	[$axattd3, player_axed4 ] {self.weaponframe=7;W_FireAxe();};
void()	player_axed4 =	[$axattd4, player_run	 ] {self.weaponframe=8;};


//============================================================================

void() player_nail1	=[$nailatt1, player_nail2  ]
{
	self.effects = self.effects | EF_MUZZLEFLASH;

	if (!self.button0)
		{player_run ();return;}
	self.weaponframe = self.weaponframe + 1;
	if (self.weaponframe == 9)
		self.weaponframe = 1;
	SuperDamageSound();
	W_FireSpikes (4);
	self.attack_finished = time + 0.2;
```

```
};
void() player_nail2   =[$nailatt2, player_nail1  ]
{
	self.effects = self.effects | EF_MUZZLEFLASH;

	if (!self.button0)
		{player_run ();return;}
	self.weaponframe = self.weaponframe + 1;
	if (self.weaponframe == 9)
		self.weaponframe = 1;
	SuperDamageSound();
	W_FireSpikes (-4);
	self.attack_finished = time + 0.2;
};

//============================================================================

void() player_light1   =[$light1, player_light2  ]
{
	self.effects = self.effects | EF_MUZZLEFLASH;

	if (!self.button0)
		{player_run ();return;}
	self.weaponframe = self.weaponframe + 1;
	if (self.weaponframe == 5)
		self.weaponframe = 1;
	SuperDamageSound();
	W_FireLightning();
	self.attack_finished = time + 0.2;
};
void() player_light2   =[$light2, player_light1  ]
{
	self.effects = self.effects | EF_MUZZLEFLASH;

	if (!self.button0)
		{player_run ();return;}
	self.weaponframe = self.weaponframe + 1;
	if (self.weaponframe == 5)
		self.weaponframe = 1;
	SuperDamageSound();
	W_FireLightning();
	self.attack_finished = time + 0.2;
};

//============================================================================


void() player_rocket1   =[$rockatt1, player_rocket2  ] {self.weaponframe=1;
self.effects = self.effects | EF_MUZZLEFLASH;};
void() player_rocket2   =[$rockatt2, player_rocket3  ] {self.weaponframe=2;};
void() player_rocket3   =[$rockatt3, player_rocket4  ] {self.weaponframe=3;};
void() player_rocket4   =[$rockatt4, player_rocket5  ] {self.weaponframe=4;};
void() player_rocket5   =[$rockatt5, player_rocket6  ] {self.weaponframe=5;};
void() player_rocket6   =[$rockatt6, player_run  ] {self.weaponframe=6;};
void(float num_bubbles) DeathBubbles;

void() PainSound =
{
local float                     rs;

	if (self.health < 0)
		return;
```

```
	if (damage_attacker.classname == "teledeath")
	{
		sound (self, CHAN_VOICE, "player/teledth1.wav", 1, ATTN_NONE);
		return;
	}

// water pain sounds
	if (self.watertype == CONTENT_WATER && self.waterlevel == 3)
	{
		DeathBubbles(1);
		if (random() > 0.5)
			sound (self, CHAN_VOICE, "player/drown1.wav", 1, ATTN_NORM);
		else
			sound (self, CHAN_VOICE, "player/drown2.wav", 1, ATTN_NORM);
		return;
	}

// slime pain sounds
	if (self.watertype == CONTENT_SLIME)
	{
// FIX ME	put in some steam here
		if (random() > 0.5)
			sound (self, CHAN_VOICE, "player/lburn1.wav", 1, ATTN_NORM);
		else
			sound (self, CHAN_VOICE, "player/lburn2.wav", 1, ATTN_NORM);
		return;
	}

	if (self.watertype == CONTENT_LAVA)
	{
		if (random() > 0.5)
			sound (self, CHAN_VOICE, "player/lburn1.wav", 1, ATTN_NORM);
		else
			sound (self, CHAN_VOICE, "player/lburn2.wav", 1, ATTN_NORM);
		return;
	}

	if (self.pain_finished > time)
	{
		self.axhitme = 0;
		return;
	}
	self.pain_finished = time + 0.5;

// don't make multiple pain sounds right after each other

// ax pain sound
	if (self.axhitme == 1)
	{
		self.axhitme = 0;
		sound (self, CHAN_VOICE, "player/axhit1.wav", 1, ATTN_NORM);
		return;
	}


	rs = rint((random() * 5) + 1);

	self.noise = "";
	if (rs == 1)
		self.noise = "player/pain1.wav";
	else if (rs == 2)
```

```
                self.noise = "player/pain2.wav";
        else if (rs == 3)
                self.noise = "player/pain3.wav";
        else if (rs == 4)
                self.noise = "player/pain4.wav";
        else if (rs == 5)
                self.noise = "player/pain5.wav";
        else
                self.noise = "player/pain6.wav";

        sound (self, CHAN_VOICE, self.noise, 1, ATTN_NORM);
        return;
};

void()  player_pain1 =  [       $pain1, player_pain2    ] {PainSound();self.weaponframe=0;};
void()  player_pain2 =  [       $pain2, player_pain3    ] {};
void()  player_pain3 =  [       $pain3, player_pain4    ] {};
void()  player_pain4 =  [       $pain4, player_pain5    ] {};
void()  player_pain5 =  [       $pain5, player_pain6    ] {};
void()  player_pain6 =  [       $pain6, player_run      ] {};

void()  player_axpain1 =        [       $axpain1,       player_axpain2  ] {PainSound();self.weaponframe=0;};
void()  player_axpain2 =        [       $axpain2,       player_axpain3  ] {};
void()  player_axpain3 =        [       $axpain3,       player_axpain4  ] {};
void()  player_axpain4 =        [       $axpain4,       player_axpain5  ] {};
void()  player_axpain5 =        [       $axpain5,       player_axpain6  ] {};
void()  player_axpain6 =        [       $axpain6,       player_run      ] {};

void() player_pain =
{
        if (self.weaponframe)
                return;

        if (self.invisible_finished > time)
                return;         // eyes don't have pain frames

        if (self.weapon == IT_AXE)
                player_axpain1 ();
        else
                player_pain1 ();
};

void() player_diea1;
void() player_dieb1;
void() player_diec1;
void() player_died1;
void() player_diee1;
void() player_die_ax1;

void() DeathBubblesSpawn =
{
local entity    bubble;
        if (self.owner.waterlevel != 3)
                return;
        bubble = spawn();
        setmodel (bubble, "progs/s_bubble.spr");
        setorigin (bubble, self.owner.origin + '0 0 24');
        bubble.movetype = MOVETYPE_NOCLIP;
        bubble.solid = SOLID_NOT;
        bubble.velocity = '0 0 15';
        bubble.nextthink = time + 0.5;
        bubble.think = bubble_bob;
```

```
	bubble.classname = "bubble";
	bubble.frame = 0;
	bubble.cnt = 0;
	setsize (bubble, '-8 -8 -8', '8 8 8');
	self.nextthink = time + 0.1;
	self.think = DeathBubblesSpawn;
	self.air_finished = self.air_finished + 1;
	if (self.air_finished >= self.bubble_count)
		remove(self);
};

void(float num_bubbles) DeathBubbles =
{
local entity	bubble_spawner;

	bubble_spawner = spawn();
	setorigin (bubble_spawner, self.origin);
	bubble_spawner.movetype = MOVETYPE_NONE;
	bubble_spawner.solid = SOLID_NOT;
	bubble_spawner.nextthink = time + 0.1;
	bubble_spawner.think = DeathBubblesSpawn;
	bubble_spawner.air_finished = 0;
	bubble_spawner.owner = self;
	bubble_spawner.bubble_count = num_bubbles;
	return;
};


void() DeathSound =
{
local float			rs;

	// water death sounds
	if (self.waterlevel == 3)
	{
		DeathBubbles(20);
		sound (self, CHAN_VOICE, "player/h2odeath.wav", 1, ATTN_NONE);
		return;
	}

	rs = rint ((random() * 4) + 1);
	if (rs == 1)
		self.noise = "player/death1.wav";
	if (rs == 2)
		self.noise = "player/death2.wav";
	if (rs == 3)
		self.noise = "player/death3.wav";
	if (rs == 4)
		self.noise = "player/death4.wav";
	if (rs == 5)
		self.noise = "player/death5.wav";

	sound (self, CHAN_VOICE, self.noise, 1, ATTN_NONE);
	return;
};


void() PlayerDead =
{
	self.nextthink = -1;
// allow respawn after a certain time
	self.deadflag = DEAD_DEAD;
```

```
};

vector(float dm) VelocityForDamage =
{
	local vector v;

	v_x = 100 * crandom();
	v_y = 100 * crandom();
	v_z = 200 + 100 * random();

	if (dm > -50)
	{
//			dprint ("level 1\n");
			v = v * 0.7;
	}
	else if (dm > -200)
	{
//			dprint ("level 3\n");
			v = v * 2;
	}
	else
			v = v * 10;

	return v;
};

void(string gibname, float dm) ThrowGib =
{
	local	entity new;

	new = spawn();
	new.origin = self.origin;
	setmodel (new, gibname);
	setsize (new, '0 0 0', '0 0 0');
	new.velocity = VelocityForDamage (dm);
	new.movetype = MOVETYPE_BOUNCE;
	new.solid = SOLID_NOT;
	new.avelocity_x = random()*600;
	new.avelocity_y = random()*600;
	new.avelocity_z = random()*600;
	new.think = SUB_Remove;
	new.ltime = time;
	new.nextthink = time + 10 + random()*10;
	new.frame = 0;
	new.flags = 0;
};

void(string gibname, float dm) ThrowHead =
{
	setmodel (self, gibname);
	self.frame = 0;
	self.nextthink = -1;
	self.movetype = MOVETYPE_BOUNCE;
	self.takedamage = DAMAGE_NO;
	self.solid = SOLID_NOT;
	self.view_ofs = '0 0 8';
	setsize (self, '-16 -16 0', '16 16 56');
	self.velocity = VelocityForDamage (dm);
	self.origin_z = self.origin_z - 24;
	self.flags = self.flags - (self.flags & FL_ONGROUND);
	self.avelocity = crandom() * '0 600 0';
};
```

```
void() GibPlayer =
{
	ThrowHead ("progs/h_player.mdl", self.health);
	ThrowGib ("progs/gib1.mdl", self.health);
	ThrowGib ("progs/gib2.mdl", self.health);
	ThrowGib ("progs/gib3.mdl", self.health);

	self.deadflag = DEAD_DEAD;

	if (damage_attacker.classname == "teledeath")
	{
		sound (self, CHAN_VOICE, "player/teledth1.wav", 1, ATTN_NONE);
		return;
	}

	if (damage_attacker.classname == "teledeath2")
	{
		sound (self, CHAN_VOICE, "player/teledth1.wav", 1, ATTN_NONE);
		return;
	}

	if (random() < 0.5)
		sound (self, CHAN_VOICE, "player/gib.wav", 1, ATTN_NONE);
	else
		sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NONE);
};

void() PlayerDie =
{
	local	float	i;

	self.items = self.items - (self.items & IT_INVISIBILITY);
	self.invisible_finished = 0;		// don't die as eyes
	self.invincible_finished = 0;
	self.super_damage_finished = 0;
	self.radsuit_finished = 0;
	self.modelindex = modelindex_player;	// don't use eyes

	if (deathmatch || coop)
		DropBackpack();

	self.weaponmodel="";
	self.view_ofs = '0 0 -8';
	self.deadflag = DEAD_DYING;
	self.solid = SOLID_NOT;
	self.flags = self.flags - (self.flags & FL_ONGROUND);
	self.movetype = MOVETYPE_TOSS;
	if (self.velocity_z < 10)
		self.velocity_z = self.velocity_z + random()*300;

	if (self.health < -40)
	{
		GibPlayer ();
		return;
	}

	DeathSound();

	self.angles_x = 0;
	self.angles_z = 0;
```

```
	if (self.weapon == IT_AXE)
	{
		player_die_ax1 ();
		return;
	}

	i = cvar("temp1");
	if (!i)
		i = 1 + floor(random()*6);

	if (i == 1)
		player_diea1();
	else if (i == 2)
		player_dieb1();
	else if (i == 3)
		player_diec1();
	else if (i == 4)
		player_died1();
	else
		player_diee1();

};

void() set_suicide_frame =
{	// used by klill command and diconnect command
	if (self.model != "progs/player.mdl")
		return;	// allready gibbed
	self.frame = $deatha11;
	self.solid = SOLID_NOT;
	self.movetype = MOVETYPE_TOSS;
	self.deadflag = DEAD_DEAD;
	self.nextthink = -1;
};


void()	player_diea1	=	[	$deatha1,	player_diea2	] {};
void()	player_diea2	=	[	$deatha2,	player_diea3	] {};
void()	player_diea3	=	[	$deatha3,	player_diea4	] {};
void()	player_diea4	=	[	$deatha4,	player_diea5	] {};
void()	player_diea5	=	[	$deatha5,	player_diea6	] {};
void()	player_diea6	=	[	$deatha6,	player_diea7	] {};
void()	player_diea7	=	[	$deatha7,	player_diea8	] {};
void()	player_diea8	=	[	$deatha8,	player_diea9	] {};
void()	player_diea9	=	[	$deatha9,	player_diea10	] {};
void()	player_diea10	=	[	$deatha10,	player_diea11	] {};
void()	player_diea11	=	[	$deatha11,	player_diea11 ] {PlayerDead();};

void()	player_dieb1	=	[	$deathb1,	player_dieb2	] {};
void()	player_dieb2	=	[	$deathb2,	player_dieb3	] {};
void()	player_dieb3	=	[	$deathb3,	player_dieb4	] {};
void()	player_dieb4	=	[	$deathb4,	player_dieb5	] {};
void()	player_dieb5	=	[	$deathb5,	player_dieb6	] {};
void()	player_dieb6	=	[	$deathb6,	player_dieb7	] {};
void()	player_dieb7	=	[	$deathb7,	player_dieb8	] {};
void()	player_dieb8	=	[	$deathb8,	player_dieb9	] {};
void()	player_dieb9	=	[	$deathb9,	player_dieb9	] {PlayerDead();};

void()	player_diec1	=	[	$deathc1,	player_diec2	] {};
void()	player_diec2	=	[	$deathc2,	player_diec3	] {};
void()	player_diec3	=	[	$deathc3,	player_diec4	] {};
void()	player_diec4	=	[	$deathc4,	player_diec5	] {};
```

```
void()	player_diec5	=	[	$deathc5,	player_diec6	] {};
void()	player_diec6	=	[	$deathc6,	player_diec7	] {};
void()	player_diec7	=	[	$deathc7,	player_diec8	] {};
void()	player_diec8	=	[	$deathc8,	player_diec9	] {};
void()	player_diec9	=	[	$deathc9,	player_diec10	] {};
void()	player_diec10	=	[	$deathc10,	player_diec11	] {};
void()	player_diec11	=	[	$deathc11,	player_diec12	] {};
void()	player_diec12	=	[	$deathc12,	player_diec13	] {};
void()	player_diec13	=	[	$deathc13,	player_diec14	] {};
void()	player_diec14	=	[	$deathc14,	player_diec15	] {};
void()	player_diec15	=	[	$deathc15,	player_diec15 ] {PlayerDead();};

void()	player_died1	=	[	$deathd1,	player_died2	] {};
void()	player_died2	=	[	$deathd2,	player_died3	] {};
void()	player_died3	=	[	$deathd3,	player_died4	] {};
void()	player_died4	=	[	$deathd4,	player_died5	] {};
void()	player_died5	=	[	$deathd5,	player_died6	] {};
void()	player_died6	=	[	$deathd6,	player_died7	] {};
void()	player_died7	=	[	$deathd7,	player_died8	] {};
void()	player_died8	=	[	$deathd8,	player_died9	] {};
void()	player_died9	=	[	$deathd9,	player_died9	] {PlayerDead();};

void()	player_diee1	=	[	$deathe1,	player_diee2	] {};
void()	player_diee2	=	[	$deathe2,	player_diee3	] {};
void()	player_diee3	=	[	$deathe3,	player_diee4	] {};
void()	player_diee4	=	[	$deathe4,	player_diee5	] {};
void()	player_diee5	=	[	$deathe5,	player_diee6	] {};
void()	player_diee6	=	[	$deathe6,	player_diee7	] {};
void()	player_diee7	=	[	$deathe7,	player_diee8	] {};
void()	player_diee8	=	[	$deathe8,	player_diee9	] {};
void()	player_diee9	=	[	$deathe9,	player_diee9	] {PlayerDead();};

void()	player_die_ax1	=	[	$axdeth1,	player_die_ax2	] {};
void()	player_die_ax2	=	[	$axdeth2,	player_die_ax3	] {};
void()	player_die_ax3	=	[	$axdeth3,	player_die_ax4	] {};
void()	player_die_ax4	=	[	$axdeth4,	player_die_ax5	] {};
void()	player_die_ax5	=	[	$axdeth5,	player_die_ax6	] {};
void()	player_die_ax6	=	[	$axdeth6,	player_die_ax7	] {};
void()	player_die_ax7	=	[	$axdeth7,	player_die_ax8	] {};
void()	player_die_ax8	=	[	$axdeth8,	player_die_ax9	] {};
void()	player_die_ax9	=	[	$axdeth9,	player_die_ax9	] {PlayerDead();};
```

SUBS.QC

```
void() SUB_Null = {};

void() SUB_Remove = {remove(self);};


/*
QuakeEd only writes a single float for angles (bad idea), so up and down are
just constant angles.
*/
vector() SetMovedir =
{
	if (self.angles == '0 -1 0')
		self.movedir = '0 0 1';
	else if (self.angles == '0 -2 0')
		self.movedir = '0 0 -1';
	else
	{
		makevectors (self.angles);
		self.movedir = v_forward;
	}

	self.angles = '0 0 0';
};

/*
================
InitTrigger
================
*/
void() InitTrigger =
{
// trigger angles are used for one-way touches.  An angle of 0 is assumed
// to mean no restrictions, so use a yaw of 360 instead.
	if (self.angles != '0 0 0')
		SetMovedir ();
	self.solid = SOLID_TRIGGER;
	setmodel (self, self.model);	// set size and link into world
	self.movetype = MOVETYPE_NONE;
	self.modelindex = 0;
	self.model = "";
};

/*
=============
SUB_CalcMove

calculate self.velocity and self.nextthink to reach dest from
self.origin traveling at speed
===============
*/
void(entity ent, vector tdest, float tspeed, void() func) SUB_CalcMoveEnt =
{
local entity	stemp;
	stemp = self;
	self = ent;

	SUB_CalcMove (tdest, tspeed, func);
	self = stemp;
```

```
};

void(vector tdest, float tspeed, void() func) SUB_CalcMove =
{
local vector	vdestdelta;
local float		len, traveltime;

	if (!tspeed)
		objerror("No speed is defined!");

	self.think1 = func;
	self.finaldest = tdest;
	self.think = SUB_CalcMoveDone;

	if (tdest == self.origin)
	{
		self.velocity = '0 0 0';
		self.nextthink = self.ltime + 0.1;
		return;
	}

// set destdelta to the vector needed to move
	vdestdelta = tdest - self.origin;

// calculate length of vector
	len = vlen (vdestdelta);

// divide by speed to get time to reach dest
	traveltime = len / tspeed;

	if (traveltime < 0.1)
	{
		self.velocity = '0 0 0';
		self.nextthink = self.ltime + 0.1;
		return;
	}

// set nextthink to trigger a think when dest is reached
	self.nextthink = self.ltime + traveltime;

// scale the destdelta vector by the time spent traveling to get velocity
	self.velocity = vdestdelta * (1/traveltime); // qcc won't take vec/float
};

/*
============
After moving, set origin to exact final destination
============
*/
void()  SUB_CalcMoveDone =
{
	setorigin(self, self.finaldest);
	self.velocity = '0 0 0';
	self.nextthink = -1;
	if (self.think1)
		self.think1();
};


/*
=============
SUB_CalcAngleMove
```

```
calculate self.avelocity and self.nextthink to reach destangle from
self.angles rotating

The calling function should make sure self.think is valid
===============
*/
void(entity ent, vector destangle, float tspeed, void() func) SUB_CalcAngleMoveEnt =
{
local entity                    stemp;
        stemp = self;
        self = ent;
        SUB_CalcAngleMove (destangle, tspeed, func);
        self = stemp;
};

void(vector destangle, float tspeed, void() func) SUB_CalcAngleMove =
{
local vector    destdelta;
local float                     len, traveltime;

        if (!tspeed)
                objerror("No speed is defined!");

// set destdelta to the vector needed to move
        destdelta = destangle - self.angles;

// calculate length of vector
        len = vlen (destdelta);

// divide by speed to get time to reach dest
        traveltime = len / tspeed;

// set nextthink to trigger a think when dest is reached
        self.nextthink = self.ltime + traveltime;

// scale the destdelta vector by the time spent traveling to get velocity
        self.avelocity = destdelta * (1 / traveltime);

        self.think1 = func;
        self.finalangle = destangle;
        self.think = SUB_CalcAngleMoveDone;
};

/*
============
After rotating, set angle to exact final angle
============
*/
void() SUB_CalcAngleMoveDone =
{
        self.angles = self.finalangle;
        self.avelocity = '0 0 0';
        self.nextthink = -1;
        if (self.think1)
                self.think1();
};


//============================================================================

void() DelayThink =
```

```
{
	activator = self.enemy;
	SUB_UseTargets ();
	remove(self);
};

/*
==============================
SUB_UseTargets

the global "activator" should be set to the entity that initiated the firing.

If self.delay is set, a DelayedUse entity will be created that will actually
do the SUB_UseTargets after that many seconds have passed.

Centerprints any self.message to the activator.

Removes all entities with a targetname that match self.killtarget,
and removes them, so some events can remove other triggers.

Search for (string)targetname in all entities that
match (string)self.target and call their .use function

==============================
*/
void() SUB_UseTargets =
{
	local entity t, stemp, otemp, act;

//
// check for a delay
//
	if (self.delay)
	{
	// create a temp object to fire at a later time
		t = spawn();
		t.classname = "DelayedUse";
		t.nextthink = time + self.delay;
		t.think = DelayThink;
		t.enemy = activator;
		t.message = self.message;
		t.killtarget = self.killtarget;
		t.target = self.target;
		return;
	}


//
// print the message
//
	if (activator.classname == "player" && self.message != "")
	{
		centerprint (activator, self.message);
		if (!self.noise)
			sound (activator, CHAN_VOICE, "misc/talk.wav", 1, ATTN_NORM);
	}

//
// kill the killtagets
//
	if (self.killtarget)
	{
```

```
		t = world;
		do
		{
			t = find (t, targetname, self.killtarget);
			if (!t)
				return;
			remove (t);
		} while ( 1 );
	}

//
// fire targets
//
	if (self.target)
	{
		act = activator;
		t = world;
		do
		{
			t = find (t, targetname, self.target);
			if (!t)
			{
				return;
			}
			stemp = self;
			otemp = other;
			self = t;
			other = stemp;
			if (self.use != SUB_Null)
			{
				if (self.use)
					self.use ();
			}
			self = stemp;
			other = otemp;
			activator = act;
		} while ( 1 );
	}


};


/*

in nightmare mode, all attack_finished times become 0
some monsters refire twice automatically

*/

void(float normal) SUB_AttackFinished =
{
	self.cnt = 0;			// refire count for nightmare
	if (skill != 3)
		self.attack_finished = time + normal;
};

float (entity targ) visible;

void (void() thinkst) SUB_CheckRefire =
{
	if (skill != 3)
```

```
                return;
        if (self.cnt == 1)
                return;
        if (!visible (self.enemy))
                return;
        self.cnt = 1;
        self.think = thinkst;
};
```

TRIGGERS.QC

```
entity stemp, otemp, s, old;


void() trigger_reactivate =
{
	self.solid = SOLID_TRIGGER;
};

//============================================================================

float	SPAWNFLAG_NOMESSAGE = 1;
float	SPAWNFLAG_NOTOUCH = 1;

// the wait time has passed, so set back up for another activation
void() multi_wait =
{
	if (self.max_health)
	{
		self.health = self.max_health;
		self.takedamage = DAMAGE_YES;
		self.solid = SOLID_BBOX;
	}
};


// the trigger was just touched/killed/used
// self.enemy should be set to the activator so it can be held through a delay
// so wait for the delay time before firing
void() multi_trigger =
{
	if (self.nextthink > time)
	{
		return;		// allready been triggered
	}

	if (self.classname == "trigger_secret")
	{
		if (self.enemy.classname != "player")
			return;
		found_secrets = found_secrets + 1;
		WriteByte (MSG_ALL, SVC_FOUNDSECRET);
	}

	if (self.noise)
		sound (self, CHAN_VOICE, self.noise, 1, ATTN_NORM);

// don't trigger again until reset
	self.takedamage = DAMAGE_NO;

	activator = self.enemy;

	SUB_UseTargets();

	if (self.wait > 0)
	{
		self.think = multi_wait;
		self.nextthink = time + self.wait;
	}
	else
```

```
	{		// we can't just remove (self) here, because this is a touch function
			// called wheil C code is looping through area links...
			self.touch = SUB_Null;
			self.nextthink = time + 0.1;
			self.think = SUB_Remove;
	}
};

void() multi_killed =
{
	self.enemy = damage_attacker;
	multi_trigger();
};

void() multi_use =
{
	self.enemy = activator;
	multi_trigger();
};

void() multi_touch =
{
	if (other.classname != "player")
		return;

// if the trigger has an angles field, check player's facing direction
	if (self.movedir != '0 0 0')
	{
		makevectors (other.angles);
		if (v_forward * self.movedir < 0)
			return;		// not facing the right way
	}

	self.enemy = other;
	multi_trigger ();
};

/*QUAKED trigger_multiple (.5 .5 .5) ? notouch
Variable sized repeatable trigger.  Must be targeted at one or more entities.  If "health" is set, the trigger must be killed to activate each time.
If "delay" is set, the trigger waits some time after activating before firing.
"wait" : Seconds between triggerings. (.2 default)
If notouch is set, the trigger is only fired by other entities, not by touching.
NOTOUCH has been obsoleted by trigger_relay!
sounds
1)	secret
2)	beep beep
3)	large switch
4)
set "message" to text string
*/
void() trigger_multiple =
{
	if (self.sounds == 1)
	{
		precache_sound ("misc/secret.wav");
		self.noise = "misc/secret.wav";
	}
	else if (self.sounds == 2)
	{
		precache_sound ("misc/talk.wav");
		self.noise = "misc/talk.wav";
```

```
	}
	else if (self.sounds == 3)
	{
		precache_sound ("misc/trigger1.wav");
		self.noise = "misc/trigger1.wav";
	}

	if (!self.wait)
		self.wait = 0.2;
	self.use = multi_use;

	InitTrigger ();

	if (self.health)
	{
		if (self.spawnflags & SPAWNFLAG_NOTOUCH)
			objerror ("health and notouch don't make sense\n");
		self.max_health = self.health;
		self.th_die = multi_killed;
		self.takedamage = DAMAGE_YES;
		self.solid = SOLID_BBOX;
		setorigin (self, self.origin);		// make sure it links into the world
	}
	else
	{
		if ( !(self.spawnflags & SPAWNFLAG_NOTOUCH) )
		{
			self.touch = multi_touch;
		}
	}
};


/*QUAKED trigger_once (.5 .5 .5) ? notouch
Variable sized trigger. Triggers once, then removes itself.  You must set the key "target" to the name of another object in the level that has a matching
"targetname".  If "health" is set, the trigger must be killed to activate.
If notouch is set, the trigger is only fired by other entities, not by touching.
if "killtarget" is set, any objects that have a matching "target" will be removed when the trigger is fired.
if "angle" is set, the trigger will only fire when someone is facing the direction of the angle.  Use "360" for an angle of 0.
sounds
1)	secret
2)	beep beep
3)	large switch
4)
set "message" to text string
*/
void() trigger_once =
{
	self.wait = -1;
	trigger_multiple();
};

//============================================================================

/*QUAKED trigger_relay (.5 .5 .5) (-8 -8 -8) (8 8 8)
This fixed size trigger cannot be touched, it can only be fired by other events.  It can contain killtargets, targets, delays, and messages.
*/
void() trigger_relay =
{
	self.use = SUB_UseTargets;
```

```
};


//==========================================================================

/*QUAKED trigger_secret (.5 .5 .5) ?
secret counter trigger
sounds
1)	secret
2)	beep beep
3)
4)
set "message" to text string
*/
void() trigger_secret =
{
	total_secrets = total_secrets + 1;
	self.wait = -1;
	if (!self.message)
		self.message = "You found a secret area!";
	if (!self.sounds)
		self.sounds = 1;

	if (self.sounds == 1)
	{
		precache_sound ("misc/secret.wav");
		self.noise = "misc/secret.wav";
	}
	else if (self.sounds == 2)
	{
		precache_sound ("misc/talk.wav");
		self.noise = "misc/talk.wav";
	}

	trigger_multiple ();
};

//==========================================================================


void() counter_use =
{
	local string junk;

	self.count = self.count - 1;
	if (self.count < 0)
		return;

	if (self.count != 0)
	{
		if (activator.classname == "player"
		&& (self.spawnflags & SPAWNFLAG_NOMESSAGE) == 0)
		{
			if (self.count >= 4)
				centerprint (activator, "There are more to go...");
			else if (self.count == 3)
				centerprint (activator, "Only 3 more to go...");
			else if (self.count == 2)
				centerprint (activator, "Only 2 more to go...");
			else
				centerprint (activator, "Only 1 more to go...");
		}
```

```
                return;
        }

        if (activator.classname == "player"
        && (self.spawnflags & SPAWNFLAG_NOMESSAGE) == 0)
                centerprint(activator, "Sequence completed!");
        self.enemy = activator;
        multi_trigger ();
};

/*QUAKED trigger_counter (.5 .5 .5) ? nomessage
Acts as an intermediary for an action that takes multiple inputs.

If nomessage is not set, t will print "1 more.. " etc when triggered and "sequence complete" when finished.

After the counter has been triggered "count" times (default 2), it will fire all of it's targets and remove itself.
*/
void() trigger_counter =
{
        self.wait = -1;
        if (!self.count)
                self.count = 2;

        self.use = counter_use;
};


/*
==============================================================================

TELEPORT TRIGGERS

==============================================================================
*/

float   PLAYER_ONLY             = 1;
float   SILENT = 2;

void() play_teleport =
{
        local   float v;
        local   string tmpstr;

        v = random() * 5;
        if (v < 1)
                tmpstr = "misc/r_tele1.wav";
        else if (v < 2)
                tmpstr = "misc/r_tele2.wav";
        else if (v < 3)
                tmpstr = "misc/r_tele3.wav";
        else if (v < 4)
                tmpstr = "misc/r_tele4.wav";
        else
                tmpstr = "misc/r_tele5.wav";

        sound (self, CHAN_VOICE, tmpstr, 1, ATTN_NORM);
        remove (self);
};

void(vector org) spawn_tfog =
{
        s = spawn ();
```

```
	s.origin = org;
	s.nextthink = time + 0.2;
	s.think = play_teleport;

	WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
	WriteByte (MSG_BROADCAST, TE_TELEPORT);
	WriteCoord (MSG_BROADCAST, org_x);
	WriteCoord (MSG_BROADCAST, org_y);
	WriteCoord (MSG_BROADCAST, org_z);
};


void() tdeath_touch =
{
	if (other == self.owner)
		return;

// frag anyone who teleports in on top of an invincible player
	if (other.classname == "player")
	{
		if (other.invincible_finished > time)
			self.classname = "teledeath2";
		if (self.owner.classname != "player")
		{	// other monsters explode themselves
			T_Damage (self.owner, self, self, 50000);
			return;
		}

	}

	if (other.health)
	{
		T_Damage (other, self, self, 50000);
	}
};


void(vector org, entity death_owner) spawn_tdeath =
{
local entity	death;

	death = spawn();
	death.classname = "teledeath";
	death.movetype = MOVETYPE_NONE;
	death.solid = SOLID_TRIGGER;
	death.angles = '0 0 0';
	setsize (death, death_owner.mins - '1 1 1', death_owner.maxs + '1 1 1');
	setorigin (death, org);
	death.touch = tdeath_touch;
	death.nextthink = time + 0.2;
	death.think = SUB_Remove;
	death.owner = death_owner;

	force_retouch = 2;		// make sure even still objects get hit
};

void() teleport_touch =
{
local entity	t;
local vector	org;

	if (self.targetname)
```

```
	{
		if (self.nextthink < time)
		{
			return;		// not fired yet
		}
	}

	if (self.spawnflags & PLAYER_ONLY)
	{
		if (other.classname != "player")
			return;
	}

// only teleport living creatures
	if (other.health <= 0 || other.solid != SOLID_SLIDEBOX)
		return;

	SUB_UseTargets ();

// put a tfog where the player was
	spawn_tfog (other.origin);

	t = find (world, targetname, self.target);
	if (!t)
		objerror ("couldn't find target");

// spawn a tfog flash in front of the destination
	makevectors (t.mangle);
	org = t.origin + 32 * v_forward;

	spawn_tfog (org);
	spawn_tdeath(t.origin, other);

// move the player and lock him down for a little while
	if (!other.health)
	{
		other.origin = t.origin;
		other.velocity = (v_forward * other.velocity_x) + (v_forward * other.velocity_y);
		return;
	}

	setorigin (other, t.origin);
	other.angles = t.mangle;
	if (other.classname == "player")
	{
		other.fixangle = 1;			// turn this way immediately
		other.teleport_time = time + 0.7;
		if (other.flags & FL_ONGROUND)
			other.flags = other.flags - FL_ONGROUND;
		other.velocity = v_forward * 300;
	}
	other.flags = other.flags - other.flags & FL_ONGROUND;
};

/*QUAKED info_teleport_destination (.5 .5 .5) (-8 -8 -8) (8 8 32)
This is the destination marker for a teleporter.  It should have a "targetname" field with the same value as a teleporter's
"target" field.
*/
void() info_teleport_destination =
{
// this does nothing, just serves as a target spot
	self.mangle = self.angles;
```

```
	self.angles = '0 0 0';
	self.model = "";
	self.origin = self.origin + '0 0 27';
	if (!self.targetname)
		objerror ("no targetname");
};

void() teleport_use =
{
	self.nextthink = time + 0.2;
	force_retouch = 2;			// make sure even still objects get hit
	self.think = SUB_Null;
};

/*QUAKED trigger_teleport (.5 .5 .5) ? PLAYER_ONLY SILENT
Any object touching this will be transported to the corresponding info_teleport_destination entity. You must set the "target" field, and create an object with a "targetname" field that matches.

If the trigger_teleport has a targetname, it will only teleport entities when it has been fired.
*/
void() trigger_teleport =
{
	local vector o;

	InitTrigger ();
	self.touch = teleport_touch;
	// find the destination
	if (!self.target)
		objerror ("no target");
	self.use = teleport_use;

	if (!(self.spawnflags & SILENT))
	{
		precache_sound ("ambience/hum1.wav");
		o = (self.mins + self.maxs)*0.5;
		ambientsound (o, "ambience/hum1.wav",0.5 , ATTN_STATIC);
	}
};

/*
=============================================================================

trigger_setskill

=============================================================================
*/

void() trigger_skill_touch =
{
	if (other.classname != "player")
		return;

	cvar_set ("skill", self.message);
};

/*QUAKED trigger_setskill (.5 .5 .5) ?
sets skill level to the value of "message".
Only used on start map.
*/
void() trigger_setskill =
{
	InitTrigger ();
```

```
	self.touch = trigger_skill_touch;
};


/*
==============================================================================

ONLY REGISTERED TRIGGERS

==============================================================================
*/

void() trigger_onlyregistered_touch =
{
	if (other.classname != "player")
		return;
	if (self.attack_finished > time)
		return;

	self.attack_finished = time + 2;
	if (cvar("registered"))
	{
		self.message = "";
		SUB_UseTargets ();
		remove (self);
	}
	else
	{
		if (self.message != "")
		{
			centerprint (other, self.message);
			sound (other, CHAN_BODY, "misc/talk.wav", 1, ATTN_NORM);
		}
	}
};

/*QUAKED trigger_onlyregistered (.5 .5 .5) ?
Only fires if playing the registered version, otherwise prints the message
*/
void() trigger_onlyregistered =
{
	precache_sound ("misc/talk.wav");
	InitTrigger ();
	self.touch = trigger_onlyregistered_touch;
};

//==============================================================================

void() hurt_on =
{
	self.solid = SOLID_TRIGGER;
	self.nextthink = -1;
};

void() hurt_touch =
{
	if (other.takedamage)
	{
		self.solid = SOLID_NOT;
		T_Damage (other, self, self, self.dmg);
		self.think = hurt_on;
		self.nextthink = time + 1;
```

```
	}

	return;
};

/*QUAKED trigger_hurt (.5 .5 .5) ?
Any object touching this will be hurt
set dmg to damage amount
defalt dmg = 5
*/
void() trigger_hurt =
{
	InitTrigger ();
	self.touch = hurt_touch;
	if (!self.dmg)
		self.dmg = 5;
};

//============================================================================

float PUSH_ONCE = 1;

void() trigger_push_touch =
{
	if (other.classname == "grenade")
		other.velocity = self.speed * self.movedir * 10;
	else if (other.health > 0)
	{
		other.velocity = self.speed * self.movedir * 10;
		if (other.classname == "player")
		{
			if (other.fly_sound < time)
			{
				other.fly_sound = time + 1.5;
				sound (other, CHAN_AUTO, "ambience/windfly.wav", 1, ATTN_NORM);
			}
		}
	}
	if (self.spawnflags & PUSH_ONCE)
		remove(self);
};


/*QUAKED trigger_push (.5 .5 .5) ? PUSH_ONCE
Pushes the player
*/
void() trigger_push =
{
	InitTrigger ();
	precache_sound ("ambience/windfly.wav");
	self.touch = trigger_push_touch;
	if (!self.speed)
		self.speed = 1000;
};

//============================================================================

void() trigger_monsterjump_touch =
{
	if ( other.flags & (FL_MONSTER | FL_FLY | FL_SWIM) != FL_MONSTER )
		return;
```

```
// set XY even if not on ground, so the jump will clear lips
	other.velocity_x = self.movedir_x * self.speed;
	other.velocity_y = self.movedir_y * self.speed;

	if ( !(other.flags & FL_ONGROUND) )
		return;

	other.flags = other.flags - FL_ONGROUND;

	other.velocity_z = self.height;
};

/*QUAKED trigger_monsterjump (.5 .5 .5) ?
Walking monsters that touch this will jump in the direction of the trigger's angle
"speed" default to 200, the speed thrown forward
"height" default to 200, the speed thrown upwards
*/
void() trigger_monsterjump =
{
	if (!self.speed)
		self.speed = 200;
	if (!self.height)
		self.height = 200;
	if (self.angles == '0 0 0')
		self.angles = '0 360 0';
	InitTrigger ();
	self.touch = trigger_monsterjump_touch;
};
```

WEAPONS.QC

```
/*
*/
void (entity targ, entity inflictor, entity attacker, float damage) T_Damage;
void () player_run;
void(entity bomb, entity attacker, float rad, entity ignore) T_RadiusDamage;
void(vector org, vector vel, float damage) SpawnBlood;
void() SuperDamageSound;


// called by worldspawn
void() W_Precache =
{
	precache_sound ("weapons/r_exp3.wav");		// new rocket explosion
	precache_sound ("weapons/rocket1i.wav");	// spike gun
	precache_sound ("weapons/sgun1.wav");
	precache_sound ("weapons/guncock.wav");	// player shotgun
	precache_sound ("weapons/ric1.wav");	// ricochet (used in c code)
	precache_sound ("weapons/ric2.wav");	// ricochet (used in c code)
	precache_sound ("weapons/ric3.wav");	// ricochet (used in c code)
	precache_sound ("weapons/spike2.wav");		// super spikes
	precache_sound ("weapons/tink1.wav"); // spikes tink (used in c code)
	precache_sound ("weapons/grenade.wav");	// grenade launcher
	precache_sound ("weapons/bounce.wav");			// grenade bounce
	precache_sound ("weapons/shotgn2.wav");	// super shotgun
};

float() crandom =
{
	return 2*(random() - 0.5);
};

/*
================
W_FireAxe
================
*/
void() W_FireAxe =
{
	local	vector	source;
	local	vector	org;

	makevectors (self.v_angle);
	source = self.origin + '0 0 16';
	traceline (source, source + v_forward*64, FALSE, self);
	if (trace_fraction == 1.0)
		return;

	org = trace_endpos - v_forward*4;

	if (trace_ent.takedamage)
	{
		trace_ent.axhitme = 1;
		SpawnBlood (org, '0 0 0', 20);
		T_Damage (trace_ent, self, self, 20);
	}
	else
	{	// hit wall
		sound (self, CHAN_WEAPON, "player/axhit2.wav", 1, ATTN_NORM);
		WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
		WriteByte (MSG_BROADCAST, TE_GUNSHOT);
```

```
                WriteCoord (MSG_BROADCAST, org_x);
                WriteCoord (MSG_BROADCAST, org_y);
                WriteCoord (MSG_BROADCAST, org_z);
        }
};


//============================================================================


vector() wall_velocity =
{
        local vector       vel;

        vel = normalize (self.velocity);
        vel = normalize(vel + v_up*(random()- 0.5) + v_right*(random()- 0.5));
        vel = vel + 2*trace_plane_normal;
        vel = vel * 200;

        return vel;
};


/*
================
SpawnMeatSpray
================
*/
void(vector org, vector vel) SpawnMeatSpray =
{
        local     entity missile, mpuff;
        local     vector   org;

        missile = spawn ();
        missile.owner = self;
        missile.movetype = MOVETYPE_BOUNCE;
        missile.solid = SOLID_NOT;

        makevectors (self.angles);

        missile.velocity = vel;
        missile.velocity_z = missile.velocity_z + 250 + 50*random();

        missile.avelocity = '3000 1000 2000';

// set missile duration
        missile.nextthink = time + 1;
        missile.think = SUB_Remove;

        setmodel (missile, "progs/zom_gib.mdl");
        setsize (missile, '0 0 0', '0 0 0');
        setorigin (missile, org);
};

/*
================
SpawnBlood
================
*/
void(vector org, vector vel, float damage) SpawnBlood =
{
        particle (org, vel*0.1, 73, damage*2);
```

```
};

/*
================
spawn_touchblood
================
*/
void(float damage) spawn_touchblood =
{
	local vector	vel;

	vel = wall_velocity () * 0.2;
	SpawnBlood (self.origin + vel*0.01, vel, damage);
};


/*
================
SpawnChunk
================
*/
void(vector org, vector vel) SpawnChunk =
{
	particle (org, vel*0.02, 0, 10);
};

/*
==============================================================================

MULTI-DAMAGE

Collects multiple small damages into a single damage

==============================================================================
*/

entity	multi_ent;
float	multi_damage;

void() ClearMultiDamage =
{
	multi_ent = world;
	multi_damage = 0;
};

void() ApplyMultiDamage =
{
	if (!multi_ent)
		return;
	T_Damage (multi_ent, self, self, multi_damage);
};

void(entity hit, float damage) AddMultiDamage =
{
	if (!hit)
		return;

	if (hit != multi_ent)
	{
		ApplyMultiDamage ();
		multi_damage = damage;
		multi_ent = hit;
```

```
	}
	else
		multi_damage = multi_damage + damage;
};

/*
==============================================================================

BULLETS

==============================================================================
*/

/*
================
TraceAttack
================
*/
void(float damage, vector dir) TraceAttack =
{
	local	vector	vel, org;

	vel = normalize(dir + v_up*crandom() + v_right*crandom());
	vel = vel + 2*trace_plane_normal;
	vel = vel * 200;

	org = trace_endpos - dir*4;

	if (trace_ent.takedamage)
	{
		SpawnBlood (org, vel*0.2, damage);
		AddMultiDamage (trace_ent, damage);
	}
	else
	{
		WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
		WriteByte (MSG_BROADCAST, TE_GUNSHOT);
		WriteCoord (MSG_BROADCAST, org_x);
		WriteCoord (MSG_BROADCAST, org_y);
		WriteCoord (MSG_BROADCAST, org_z);
	}
};

/*
================
FireBullets

Used by shotgun, super shotgun, and enemy soldier firing
Go to the trouble of combining multiple pellets into a single damage call.
================
*/
void(float shotcount, vector dir, vector spread) FireBullets =
{
	local	vector direction;
	local	vector	src;

	makevectors(self.v_angle);

	src = self.origin + v_forward*10;
	src_z = self.absmin_z + self.size_z * 0.7;

	ClearMultiDamage ();
```

```
	while (shotcount > 0)
	{
		direction = dir + crandom()*spread_x*v_right + crandom()*spread_y*v_up;

		traceline (src, src + direction*2048, FALSE, self);
		if (trace_fraction != 1.0)
			TraceAttack (4, direction);

		shotcount = shotcount - 1;
	}
	ApplyMultiDamage ();
};

/*
================
W_FireShotgun
================
*/
void() W_FireShotgun =
{
	local vector dir;

	sound (self, CHAN_WEAPON, "weapons/guncock.wav", 1, ATTN_NORM);

	self.punchangle_x = -2;

	self.currentammo = self.ammo_shells = self.ammo_shells - 1;
	dir = aim (self, 100000);
	FireBullets (6, dir, '0.04 0.04 0');
};


/*
================
W_FireSuperShotgun
================
*/
void() W_FireSuperShotgun =
{
	local vector dir;

	if (self.currentammo == 1)
	{
		W_FireShotgun ();
		return;
	}

	sound (self ,CHAN_WEAPON, "weapons/shotgn2.wav", 1, ATTN_NORM);

	self.punchangle_x = -4;

	self.currentammo = self.ammo_shells = self.ammo_shells - 2;
	dir = aim (self, 100000);
	FireBullets (14, dir, '0.14 0.08 0');
};


/*
==============================================================================

ROCKETS
```

```
==============================================================================
*/

void()	s_explode1	=	[0,		s_explode2] {};
void()	s_explode2	=	[1,		s_explode3] {};
void()	s_explode3	=	[2,		s_explode4] {};
void()	s_explode4	=	[3,		s_explode5] {};
void()	s_explode5	=	[4,		s_explode6] {};
void()	s_explode6	=	[5,		SUB_Remove] {};

void() BecomeExplosion =
{
	self.movetype = MOVETYPE_NONE;
	self.velocity = '0 0 0';
	self.touch = SUB_Null;
	setmodel (self, "progs/s_explod.spr");
	self.solid = SOLID_NOT;
	s_explode1 ();
};

void() T_MissileTouch =
{
	local float	damg;

	if (other == self.owner)
		return;		// don't explode on owner

	if (pointcontents(self.origin) == CONTENT_SKY)
	{
		remove(self);
		return;
	}

	damg = 100 + random()*20;

	if (other.health)
	{
		if (other.classname == "monster_shambler")
			damg = damg * 0.5;	// mostly immune
		T_Damage (other, self, self.owner, damg );
	}

	// don't do radius damage to the other, because all the damage
	// was done in the impact
	T_RadiusDamage (self, self.owner, 120, other);

//	sound (self, CHAN_WEAPON, "weapons/r_exp3.wav", 1, ATTN_NORM);
	self.origin = self.origin - 8*normalize(self.velocity);

	WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
	WriteByte (MSG_BROADCAST, TE_EXPLOSION);
	WriteCoord (MSG_BROADCAST, self.origin_x);
	WriteCoord (MSG_BROADCAST, self.origin_y);
	WriteCoord (MSG_BROADCAST, self.origin_z);

	BecomeExplosion ();
};



/*
===============
```

```
W_FireRocket
================
*/
void() W_FireRocket =
{
	local	entity missile, mpuff;

	self.currentammo = self.ammo_rockets = self.ammo_rockets - 1;

	sound (self, CHAN_WEAPON, "weapons/sgun1.wav", 1, ATTN_NORM);

	self.punchangle_x = -2;

	missile = spawn ();
	missile.owner = self;
	missile.movetype = MOVETYPE_FLYMISSILE;
	missile.solid = SOLID_BBOX;
	missile.classname = "missile";

// set missile speed

	makevectors (self.v_angle);
	missile.velocity = aim(self, 1000);
	missile.velocity = missile.velocity * 1000;
	missile.angles = vectoangles(missile.velocity);

	missile.touch = T_MissileTouch;

// set missile duration
	missile.nextthink = time + 5;
	missile.think = SUB_Remove;

	setmodel (missile, "progs/missile.mdl");
	setsize (missile, '0 0 0', '0 0 0');
	setorigin (missile, self.origin + v_forward*8 + '0 0 16');
};

/*
==============================================================================

LIGHTNING

==============================================================================
*/

/*
=================
LightningDamage
=================
*/
void(vector p1, vector p2, entity from, float damage) LightningDamage =
{
	local entity		e1, e2;
	local vector		f;

	f = p2 - p1;
	normalize (f);
	f_x = 0 - f_y;
	f_y = f_x;
	f_z = 0;
	f = f*16;
```

```
	e1 = e2 = world;

	traceline (p1, p2, FALSE, self);
	if (trace_ent.takedamage)
	{
		particle (trace_endpos, '0 0 100', 225, damage*4);
		T_Damage (trace_ent, from, from, damage);
		if (self.classname == "player")
		{
			if (other.classname == "player")
				trace_ent.velocity_z = trace_ent.velocity_z + 400;
		}
	}
	e1 = trace_ent;

	traceline (p1 + f, p2 + f, FALSE, self);
	if (trace_ent != e1 && trace_ent.takedamage)
	{
		particle (trace_endpos, '0 0 100', 225, damage*4);
		T_Damage (trace_ent, from, from, damage);
	}
	e2 = trace_ent;

	traceline (p1 - f, p2 - f, FALSE, self);
	if (trace_ent != e1 && trace_ent != e2 && trace_ent.takedamage)
	{
		particle (trace_endpos, '0 0 100', 225, damage*4);
		T_Damage (trace_ent, from, from, damage);
	}
};


void() W_FireLightning =
{
	local	vector		org;
	local	float		cells;

	if (self.ammo_cells < 1)
	{
		self.weapon = W_BestWeapon ();
		W_SetCurrentAmmo ();
		return;
	}

// explode if under water
	if (self.waterlevel > 1)
	{
		cells = self.ammo_cells;
		self.ammo_cells = 0;
		W_SetCurrentAmmo ();
		T_RadiusDamage (self, self, 35*cells, world);
		return;
	}

	if (self.t_width < time)
	{
		sound (self, CHAN_WEAPON, "weapons/lhit.wav", 1, ATTN_NORM);
		self.t_width = time + 0.6;
	}
	self.punchangle_x = -2;

	self.currentammo = self.ammo_cells = self.ammo_cells - 1;
```

```
	org = self.origin + '0 0 16';

	traceline (org, org + v_forward*600, TRUE, self);

	WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
	WriteByte (MSG_BROADCAST, TE_LIGHTNING2);
	WriteEntity (MSG_BROADCAST, self);
	WriteCoord (MSG_BROADCAST, org_x);
	WriteCoord (MSG_BROADCAST, org_y);
	WriteCoord (MSG_BROADCAST, org_z);
	WriteCoord (MSG_BROADCAST, trace_endpos_x);
	WriteCoord (MSG_BROADCAST, trace_endpos_y);
	WriteCoord (MSG_BROADCAST, trace_endpos_z);

	LightningDamage (self.origin, trace_endpos + v_forward*4, self, 30);
};


//============================================================================


void() GrenadeExplode =
{
	T_RadiusDamage (self, self.owner, 120, world);

	WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
	WriteByte (MSG_BROADCAST, TE_EXPLOSION);
	WriteCoord (MSG_BROADCAST, self.origin_x);
	WriteCoord (MSG_BROADCAST, self.origin_y);
	WriteCoord (MSG_BROADCAST, self.origin_z);

	BecomeExplosion ();
};

void() GrenadeTouch =
{
	if (other == self.owner)
		return;		// don't explode on owner
	if (other.takedamage == DAMAGE_AIM)
	{
		GrenadeExplode();
		return;
	}
	sound (self, CHAN_WEAPON, "weapons/bounce.wav", 1, ATTN_NORM);	// bounce sound
	if (self.velocity == '0 0 0')
		self.avelocity = '0 0 0';
};

/*
================
W_FireGrenade
================
*/
void() W_FireGrenade =
{
	local	entity missile, mpuff;

	self.currentammo = self.ammo_rockets = self.ammo_rockets - 1;

	sound (self, CHAN_WEAPON, "weapons/grenade.wav", 1, ATTN_NORM);
```

```
	self.punchangle_x = -2;

	missile = spawn ();
	missile.owner = self;
	missile.movetype = MOVETYPE_BOUNCE;
	missile.solid = SOLID_BBOX;
	missile.classname = "grenade";

// set missile speed

	makevectors (self.v_angle);

	if (self.v_angle_x)
		missile.velocity = v_forward*600 + v_up * 200 + crandom()*v_right*10 + crandom()*v_up*10;
	else
	{
		missile.velocity = aim(self, 10000);
		missile.velocity = missile.velocity * 600;
		missile.velocity_z = 200;
	}

	missile.avelocity = '300 300 300';

	missile.angles = vectoangles(missile.velocity);

	missile.touch = GrenadeTouch;

// set missile duration
	missile.nextthink = time + 2.5;
	missile.think = GrenadeExplode;

	setmodel (missile, "progs/grenade.mdl");
	setsize (missile, '0 0 0', '0 0 0');
	setorigin (missile, self.origin);
};


//============================================================================

void() spike_touch;
void() superspike_touch;


/*
===============
launch_spike

Used for both the player and the ogre
===============
*/
void(vector org, vector dir) launch_spike =
{
	newmis = spawn ();
	newmis.owner = self;
	newmis.movetype = MOVETYPE_FLYMISSILE;
	newmis.solid = SOLID_BBOX;

	newmis.angles = vectoangles(dir);

	newmis.touch = spike_touch;
	newmis.classname = "spike";
	newmis.think = SUB_Remove;
```

```
	newmis.nextthink = time + 6;
	setmodel (newmis, "progs/spike.mdl");
	setsize (newmis, VEC_ORIGIN, VEC_ORIGIN);
	setorigin (newmis, org);

	newmis.velocity = dir * 1000;
};

void() W_FireSuperSpikes =
{
	local vector	dir;
	local entity	old;

	sound (self, CHAN_WEAPON, "weapons/spike2.wav", 1, ATTN_NORM);
	self.attack_finished = time + 0.2;
	self.currentammo = self.ammo_nails = self.ammo_nails - 2;
	dir = aim (self, 1000);
	launch_spike (self.origin + '0 0 16', dir);
	newmis.touch = superspike_touch;
	setmodel (newmis, "progs/s_spike.mdl");
	setsize (newmis, VEC_ORIGIN, VEC_ORIGIN);
	self.punchangle_x = -2;
};

void(float ox) W_FireSpikes =
{
	local vector	dir;
	local entity	old;

	makevectors (self.v_angle);

	if (self.ammo_nails >= 2 && self.weapon == IT_SUPER_NAILGUN)
	{
		W_FireSuperSpikes ();
		return;
	}

	if (self.ammo_nails < 1)
	{
		self.weapon = W_BestWeapon ();
		W_SetCurrentAmmo ();
		return;
	}

	sound (self, CHAN_WEAPON, "weapons/rocket1i.wav", 1, ATTN_NORM);
	self.attack_finished = time + 0.2;
	self.currentammo = self.ammo_nails = self.ammo_nails - 1;
	dir = aim (self, 1000);
	launch_spike (self.origin + '0 0 16' + v_right*ox, dir);

	self.punchangle_x = -2;
};



.float hit_z;
void() spike_touch =
{
local float rand;
	if (other == self.owner)
		return;
```

```
	if (other.solid == SOLID_TRIGGER)
		return;	// trigger field, do nothing

	if (pointcontents(self.origin) == CONTENT_SKY)
	{
		remove(self);
		return;
	}

// hit something that bleeds
	if (other.takedamage)
	{
		spawn_touchblood (9);
		T_Damage (other, self, self.owner, 9);
	}
	else
	{
		WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);

		if (self.classname == "wizspike")
			WriteByte (MSG_BROADCAST, TE_WIZSPIKE);
		else if (self.classname == "knightspike")
			WriteByte (MSG_BROADCAST, TE_KNIGHTSPIKE);
		else
			WriteByte (MSG_BROADCAST, TE_SPIKE);
		WriteCoord (MSG_BROADCAST, self.origin_x);
		WriteCoord (MSG_BROADCAST, self.origin_y);
		WriteCoord (MSG_BROADCAST, self.origin_z);
	}

	remove(self);

};

void() superspike_touch =
{
local float rand;
	if (other == self.owner)
		return;

	if (other.solid == SOLID_TRIGGER)
		return;	// trigger field, do nothing

	if (pointcontents(self.origin) == CONTENT_SKY)
	{
		remove(self);
		return;
	}

// hit something that bleeds
	if (other.takedamage)
	{
		spawn_touchblood (18);
		T_Damage (other, self, self.owner, 18);
	}
	else
	{
		WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
		WriteByte (MSG_BROADCAST, TE_SUPERSPIKE);
		WriteCoord (MSG_BROADCAST, self.origin_x);
		WriteCoord (MSG_BROADCAST, self.origin_y);
		WriteCoord (MSG_BROADCAST, self.origin_z);
```

```
	}

	remove(self);

};


/*
==============================================================================

PLAYER WEAPON USE

==============================================================================
*/

void() W_SetCurrentAmmo =
{
	player_run ();			// get out of any weapon firing states

	self.items = self.items - ( self.items & (IT_SHELLS | IT_NAILS | IT_ROCKETS | IT_CELLS) );

	if (self.weapon == IT_AXE)
	{
		self.currentammo = 0;
		self.weaponmodel = "progs/v_axe.mdl";
		self.weaponframe = 0;
	}
	else if (self.weapon == IT_SHOTGUN)
	{
		self.currentammo = self.ammo_shells;
		self.weaponmodel = "progs/v_shot.mdl";
		self.weaponframe = 0;
		self.items = self.items | IT_SHELLS;
	}
	else if (self.weapon == IT_SUPER_SHOTGUN)
	{
		self.currentammo = self.ammo_shells;
		self.weaponmodel = "progs/v_shot2.mdl";
		self.weaponframe = 0;
		self.items = self.items | IT_SHELLS;
	}
	else if (self.weapon == IT_NAILGUN)
	{
		self.currentammo = self.ammo_nails;
		self.weaponmodel = "progs/v_nail.mdl";
		self.weaponframe = 0;
		self.items = self.items | IT_NAILS;
	}
	else if (self.weapon == IT_SUPER_NAILGUN)
	{
		self.currentammo = self.ammo_nails;
		self.weaponmodel = "progs/v_nail2.mdl";
		self.weaponframe = 0;
		self.items = self.items | IT_NAILS;
	}
	else if (self.weapon == IT_GRENADE_LAUNCHER)
	{
		self.currentammo = self.ammo_rockets;
		self.weaponmodel = "progs/v_rock.mdl";
		self.weaponframe = 0;
		self.items = self.items | IT_ROCKETS;
	}
```

```
	else if (self.weapon == IT_ROCKET_LAUNCHER)
	{
		self.currentammo = self.ammo_rockets;
		self.weaponmodel = "progs/v_rock2.mdl";
		self.weaponframe = 0;
		self.items = self.items | IT_ROCKETS;
	}
	else if (self.weapon == IT_LIGHTNING)
	{
		self.currentammo = self.ammo_cells;
		self.weaponmodel = "progs/v_light.mdl";
		self.weaponframe = 0;
		self.items = self.items | IT_CELLS;
	}
	else
	{
		self.currentammo = 0;
		self.weaponmodel = "";
		self.weaponframe = 0;
	}
};

float() W_BestWeapon =
{
	local	float	it;

	it = self.items;

	if (self.waterlevel <= 1 && self.ammo_cells >= 1 && (it & IT_LIGHTNING) )
			return IT_LIGHTNING;
	if(self.ammo_nails >= 2 && (it & IT_SUPER_NAILGUN) )
		return IT_SUPER_NAILGUN;
	if(self.ammo_shells >= 2 && (it & IT_SUPER_SHOTGUN) )
		return IT_SUPER_SHOTGUN;
	if(self.ammo_nails >= 1 && (it & IT_NAILGUN) )
		return IT_NAILGUN;
	if(self.ammo_shells >= 1 && (it & IT_SHOTGUN) )
		return IT_SHOTGUN;
	return IT_AXE;
};

float() W_CheckNoAmmo =
{
	if (self.currentammo > 0)
		return TRUE;

	if (self.weapon == IT_AXE)
		return TRUE;

	self.weapon = W_BestWeapon ();

	W_SetCurrentAmmo ();

// drop the weapon down
	return FALSE;
};

/*
============
W_Attack

An attack impulse can be triggered now
```

```
============
*/
void()	player_axe1;
void()	player_axeb1;
void()	player_axec1;
void()	player_axed1;
void()	player_shot1;
void()	player_nail1;
void()	player_light1;
void()	player_rocket1;

void() W_Attack =
{
	local	float	r;

	if (!W_CheckNoAmmo ())
		return;

	makevectors	(self.v_angle);			// calculate forward angle for velocity
	self.show_hostile = time + 1;	// wake monsters up

	if (self.weapon == IT_AXE)
	{
		sound (self, CHAN_WEAPON, "weapons/ax1.wav", 1, ATTN_NORM);
		r = random();
		if (r < 0.25)
			player_axe1 ();
		else if (r<0.5)
			player_axeb1 ();
		else if (r<0.75)
			player_axec1 ();
		else
			player_axed1 ();
		self.attack_finished = time + 0.5;
	}
	else if (self.weapon == IT_SHOTGUN)
	{
		player_shot1 ();
		W_FireShotgun ();
		self.attack_finished = time + 0.5;
	}
	else if (self.weapon == IT_SUPER_SHOTGUN)
	{
		player_shot1 ();
		W_FireSuperShotgun ();
		self.attack_finished = time + 0.7;
	}
	else if (self.weapon == IT_NAILGUN)
	{
		player_nail1 ();
	}
	else if (self.weapon == IT_SUPER_NAILGUN)
	{
		player_nail1 ();
	}
	else if (self.weapon == IT_GRENADE_LAUNCHER)
	{
		player_rocket1();
		W_FireGrenade();
		self.attack_finished = time + 0.6;
	}
	else if (self.weapon == IT_ROCKET_LAUNCHER)
```

```
	{
		player_rocket1();
		W_FireRocket();
		self.attack_finished = time + 0.8;
	}
	else if (self.weapon == IT_LIGHTNING)
	{
		player_light1();
		self.attack_finished = time + 0.1;
		sound (self, CHAN_AUTO, "weapons/lstart.wav", 1, ATTN_NORM);
	}
};

/*
============
W_ChangeWeapon

============
*/
void() W_ChangeWeapon =
{
	local	float	it, am, fl;

	it = self.items;
	am = 0;

	if (self.impulse == 1)
	{
		fl = IT_AXE;
	}
	else if (self.impulse == 2)
	{
		fl = IT_SHOTGUN;
		if (self.ammo_shells < 1)
			am = 1;
	}
	else if (self.impulse == 3)
	{
		fl = IT_SUPER_SHOTGUN;
		if (self.ammo_shells < 2)
			am = 1;
	}
	else if (self.impulse == 4)
	{
		fl = IT_NAILGUN;
		if (self.ammo_nails < 1)
			am = 1;
	}
	else if (self.impulse == 5)
	{
		fl = IT_SUPER_NAILGUN;
		if (self.ammo_nails < 2)
			am = 1;
	}
	else if (self.impulse == 6)
	{
		fl = IT_GRENADE_LAUNCHER;
		if (self.ammo_rockets < 1)
			am = 1;
	}
	else if (self.impulse == 7)
	{
```

```
                fl = IT_ROCKET_LAUNCHER;
                if (self.ammo_rockets < 1)
                        am = 1;
        }
        else if (self.impulse == 8)
        {
                fl = IT_LIGHTNING;
                if (self.ammo_cells < 1)
                        am = 1;
        }

        self.impulse = 0;

        if (!(self.items & fl))
        {       // don't have the weapon or the ammo
                sprint (self, "no weapon.\n");
                return;
        }

        if (am)
        {       // don't have the ammo
                sprint (self, "not enough ammo.\n");
                return;
        }

//
// set weapon, set ammo
//
        self.weapon = fl;
        W_SetCurrentAmmo ();
};

/*
============
CheatCommand
============
*/
void() CheatCommand =
{
        if (deathmatch || coop)
                return;

        self.ammo_rockets = 100;
        self.ammo_nails = 200;
        self.ammo_shells = 100;
        self.items = self.items |
                IT_AXE |
                IT_SHOTGUN |
                IT_SUPER_SHOTGUN |
                IT_NAILGUN |
                IT_SUPER_NAILGUN |
                IT_GRENADE_LAUNCHER |
                IT_ROCKET_LAUNCHER |
                IT_KEY1 | IT_KEY2;

        self.ammo_cells = 200;
        self.items = self.items | IT_LIGHTNING;

        self.weapon = IT_ROCKET_LAUNCHER;
        self.impulse = 0;
        W_SetCurrentAmmo ();
};
```

```
/*
============
CycleWeaponCommand

Go to the next weapon with ammo
============
*/
void() CycleWeaponCommand =
{
	local	float	it, am;

	it = self.items;
	self.impulse = 0;

	while (1)
	{
		am = 0;

		if (self.weapon == IT_LIGHTNING)
		{
			self.weapon = IT_AXE;
		}
		else if (self.weapon == IT_AXE)
		{
			self.weapon = IT_SHOTGUN;
			if (self.ammo_shells < 1)
				am = 1;
		}
		else if (self.weapon == IT_SHOTGUN)
		{
			self.weapon = IT_SUPER_SHOTGUN;
			if (self.ammo_shells < 2)
				am = 1;
		}
		else if (self.weapon == IT_SUPER_SHOTGUN)
		{
			self.weapon = IT_NAILGUN;
			if (self.ammo_nails < 1)
				am = 1;
		}
		else if (self.weapon == IT_NAILGUN)
		{
			self.weapon = IT_SUPER_NAILGUN;
			if (self.ammo_nails < 2)
				am = 1;
		}
		else if (self.weapon == IT_SUPER_NAILGUN)
		{
			self.weapon = IT_GRENADE_LAUNCHER;
			if (self.ammo_rockets < 1)
				am = 1;
		}
		else if (self.weapon == IT_GRENADE_LAUNCHER)
		{
			self.weapon = IT_ROCKET_LAUNCHER;
			if (self.ammo_rockets < 1)
				am = 1;
		}
		else if (self.weapon == IT_ROCKET_LAUNCHER)
		{
			self.weapon = IT_LIGHTNING;
```

```
                        if (self.ammo_cells < 1)
                                am = 1;
                }

                if ( (it & self.weapon) && am == 0)
                {
                        W_SetCurrentAmmo ();
                        return;
                }
        }

};

/*
============
CycleWeaponReverseCommand

Go to the prev weapon with ammo
============
*/
void() CycleWeaponReverseCommand =
{
        local   float   it, am;

        it = self.items;
        self.impulse = 0;

        while (1)
        {
                am = 0;

                if (self.weapon == IT_LIGHTNING)
                {
                        self.weapon = IT_ROCKET_LAUNCHER;
                        if (self.ammo_rockets < 1)
                                am = 1;
                }
                else if (self.weapon == IT_ROCKET_LAUNCHER)
                {
                        self.weapon = IT_GRENADE_LAUNCHER;
                        if (self.ammo_rockets < 1)
                                am = 1;
                }
                else if (self.weapon == IT_GRENADE_LAUNCHER)
                {
                        self.weapon = IT_SUPER_NAILGUN;
                        if (self.ammo_nails < 2)
                                am = 1;
                }
                else if (self.weapon == IT_SUPER_NAILGUN)
                {
                        self.weapon = IT_NAILGUN;
                        if (self.ammo_nails < 1)
                                am = 1;
                }
                else if (self.weapon == IT_NAILGUN)
                {
                        self.weapon = IT_SUPER_SHOTGUN;
                        if (self.ammo_shells < 2)
                                am = 1;
                }
                else if (self.weapon == IT_SUPER_SHOTGUN)
```

```
                {
                        self.weapon = IT_SHOTGUN;
                        if (self.ammo_shells < 1)
                                am = 1;
                }
                else if (self.weapon == IT_SHOTGUN)
                {
                        self.weapon = IT_AXE;
                }
                else if (self.weapon == IT_AXE)
                {
                        self.weapon = IT_LIGHTNING;
                        if (self.ammo_cells < 1)
                                am = 1;
                }

                if ( (it & self.weapon) && am == 0)
                {
                        W_SetCurrentAmmo ();
                        return;
                }
        }

};

/*
============
ServerflagsCommand

Just for development
============
*/
void() ServerflagsCommand =
{
        serverflags = serverflags * 2 + 1;
};

void() QuadCheat =
{
        if (deathmatch || coop)
                return;
        self.super_time = 1;
        self.super_damage_finished = time + 30;
        self.items = self.items | IT_QUAD;
        dprint ("quad cheat\n");
};

/*
============
ImpulseCommands

============
*/
void() ImpulseCommands =
{
        if (self.impulse >= 1 && self.impulse <= 8)
                W_ChangeWeapon ();

        if (self.impulse == 9)
                CheatCommand ();
        if (self.impulse == 10)
                CycleWeaponCommand ();
```

```
	if (self.impulse == 11)
		ServerflagsCommand ();
	if (self.impulse == 12)
		CycleWeaponReverseCommand ();

	if (self.impulse == 255)
		QuadCheat ();

	self.impulse = 0;
};

/*
============
W_WeaponFrame

Called every frame so impulse events can be handled as well as possible
============
*/
void() W_WeaponFrame =
{
	if (time < self.attack_finished)
		return;

	ImpulseCommands ();

// check for attack
	if (self.button0)
	{
		SuperDamageSound ();
		W_Attack ();
	}
};

/*
========
SuperDamageSound

Plays sound if needed
========
*/
void() SuperDamageSound =
{
	if (self.super_damage_finished > time)
	{
		if (self.super_sound < time)
		{
			self.super_sound = time + 1;
			sound (self, CHAN_BODY, "items/damage3.wav", 1, ATTN_NORM);
		}
	}
	return;
};
```

WORLD.QC

```
void() InitBodyQue;


void() main =
{
	dprint ("main function\n");

// these are just commands the the prog compiler to copy these files

	precache_file ("progs.dat");
	precache_file ("gfx.wad");
	precache_file ("quake.rc");
	precache_file ("default.cfg");

	precache_file ("end1.bin");
	precache_file2 ("end2.bin");

	precache_file ("demo1.dem");
	precache_file ("demo2.dem");
	precache_file ("demo3.dem");

//
// these are all of the lumps from the cached.ls files
//
	precache_file ("gfx/palette.lmp");
	precache_file ("gfx/colormap.lmp");

	precache_file2 ("gfx/pop.lmp");

	precache_file ("gfx/complete.lmp");
	precache_file ("gfx/inter.lmp");

	precache_file ("gfx/ranking.lmp");
	precache_file ("gfx/vidmodes.lmp");
	precache_file ("gfx/finale.lmp");
	precache_file ("gfx/conback.lmp");
	precache_file ("gfx/qplaque.lmp");

	precache_file ("gfx/menudot1.lmp");
	precache_file ("gfx/menudot2.lmp");
	precache_file ("gfx/menudot3.lmp");
	precache_file ("gfx/menudot4.lmp");
	precache_file ("gfx/menudot5.lmp");
	precache_file ("gfx/menudot6.lmp");

	precache_file ("gfx/menuplyr.lmp");
	precache_file ("gfx/bigbox.lmp");
	precache_file ("gfx/dim_modm.lmp");
	precache_file ("gfx/dim_drct.lmp");
	precache_file ("gfx/dim_ipx.lmp");
	precache_file ("gfx/dim_tcp.lmp");
	precache_file ("gfx/dim_mult.lmp");
	precache_file ("gfx/mainmenu.lmp");

	precache_file ("gfx/box_tl.lmp");
	precache_file ("gfx/box_tm.lmp");
	precache_file ("gfx/box_tr.lmp");

	precache_file ("gfx/box_ml.lmp");
```

```
	precache_file ("gfx/box_mm.lmp");
	precache_file ("gfx/box_mm2.lmp");
	precache_file ("gfx/box_mr.lmp");

	precache_file ("gfx/box_bl.lmp");
	precache_file ("gfx/box_bm.lmp");
	precache_file ("gfx/box_br.lmp");

	precache_file ("gfx/sp_menu.lmp");
	precache_file ("gfx/ttl_sgl.lmp");
	precache_file ("gfx/ttl_main.lmp");
	precache_file ("gfx/ttl_cstm.lmp");

	precache_file ("gfx/mp_menu.lmp");

	precache_file ("gfx/netmen1.lmp");
	precache_file ("gfx/netmen2.lmp");
	precache_file ("gfx/netmen3.lmp");
	precache_file ("gfx/netmen4.lmp");
	precache_file ("gfx/netmen5.lmp");

	precache_file ("gfx/sell.lmp");

	precache_file ("gfx/help0.lmp");
	precache_file ("gfx/help1.lmp");
	precache_file ("gfx/help2.lmp");
	precache_file ("gfx/help3.lmp");
	precache_file ("gfx/help4.lmp");
	precache_file ("gfx/help5.lmp");

	precache_file ("gfx/pause.lmp");
	precache_file ("gfx/loading.lmp");

	precache_file ("gfx/p_option.lmp");
	precache_file ("gfx/p_load.lmp");
	precache_file ("gfx/p_save.lmp");
	precache_file ("gfx/p_multi.lmp");

// sounds loaded by C code
	precache_sound ("misc/menu1.wav");
	precache_sound ("misc/menu2.wav");
	precache_sound ("misc/menu3.wav");

	precache_sound ("ambience/water1.wav");
	precache_sound ("ambience/wind2.wav");

// shareware
	precache_file ("maps/start.bsp");

	precache_file ("maps/e1m1.bsp");
	precache_file ("maps/e1m2.bsp");
	precache_file ("maps/e1m3.bsp");
	precache_file ("maps/e1m4.bsp");
	precache_file ("maps/e1m5.bsp");
	precache_file ("maps/e1m6.bsp");
	precache_file ("maps/e1m7.bsp");
	precache_file ("maps/e1m8.bsp");

// registered
	precache_file2 ("gfx/pop.lmp");

	precache_file2 ("maps/e2m1.bsp");
```

```
	precache_file2 ("maps/e2m2.bsp");
	precache_file2 ("maps/e2m3.bsp");
	precache_file2 ("maps/e2m4.bsp");
	precache_file2 ("maps/e2m5.bsp");
	precache_file2 ("maps/e2m6.bsp");
	precache_file2 ("maps/e2m7.bsp");

	precache_file2 ("maps/e3m1.bsp");
	precache_file2 ("maps/e3m2.bsp");
	precache_file2 ("maps/e3m3.bsp");
	precache_file2 ("maps/e3m4.bsp");
	precache_file2 ("maps/e3m5.bsp");
	precache_file2 ("maps/e3m6.bsp");
	precache_file2 ("maps/e3m7.bsp");

	precache_file2 ("maps/e4m1.bsp");
	precache_file2 ("maps/e4m2.bsp");
	precache_file2 ("maps/e4m3.bsp");
	precache_file2 ("maps/e4m4.bsp");
	precache_file2 ("maps/e4m5.bsp");
	precache_file2 ("maps/e4m6.bsp");
	precache_file2 ("maps/e4m7.bsp");
	precache_file2 ("maps/e4m8.bsp");

	precache_file2 ("maps/end.bsp");

	precache_file2 ("maps/dm1.bsp");
	precache_file2 ("maps/dm2.bsp");
	precache_file2 ("maps/dm3.bsp");
	precache_file2 ("maps/dm4.bsp");
	precache_file2 ("maps/dm5.bsp");
	precache_file2 ("maps/dm6.bsp");
};


entity	lastspawn;

//=======================
/*QUAKED worldspawn (0 0 0) ?
Only used for the world entity.
Set message to the level name.
Set sounds to the cd track to play.

World Types:
0: medieval
1: metal
2: base
*/
//=======================
void() worldspawn =
{
	lastspawn = world;
	InitBodyQue ();

// custom map attributes
	if (self.model == "maps/e1m8.bsp")
		cvar_set ("sv_gravity", "100");
	else
		cvar_set ("sv_gravity", "800");

// the area based ambient sounds MUST be the first precache_sounds
```

```
// player precaches
	W_Precache ();			// get weapon precaches

// sounds used from C physics code
	precache_sound ("demon/dland2.wav");		// landing thud
	precache_sound ("misc/h2ohit1.wav");		// landing splash

// setup precaches allways needed
	precache_sound ("items/itembk2.wav");		// item respawn sound
	precache_sound ("player/plyrjmp8.wav");		// player jump
	precache_sound ("player/land.wav");		// player landing
	precache_sound ("player/land2.wav");		// player hurt landing
	precache_sound ("player/drown1.wav");		// drowning pain
	precache_sound ("player/drown2.wav");		// drowning pain
	precache_sound ("player/gasp1.wav");		// gasping for air
	precache_sound ("player/gasp2.wav");		// taking breath
	precache_sound ("player/h2odeath.wav");		// drowning death

	precache_sound ("misc/talk.wav");		// talk
	precache_sound ("player/teledth1.wav");		// telefrag
	precache_sound ("misc/r_tele1.wav");		// teleport sounds
	precache_sound ("misc/r_tele2.wav");
	precache_sound ("misc/r_tele3.wav");
	precache_sound ("misc/r_tele4.wav");
	precache_sound ("misc/r_tele5.wav");
	precache_sound ("weapons/lock4.wav");		// ammo pick up
	precache_sound ("weapons/pkup.wav");		// weapon up
	precache_sound ("items/armor1.wav");		// armor up
	precache_sound ("weapons/lhit.wav");		//lightning
	precache_sound ("weapons/lstart.wav");		//lightning start
	precache_sound ("items/damage3.wav");

	precache_sound ("misc/power.wav");		//lightning for boss

// player gib sounds
	precache_sound ("player/gib.wav");		// player gib sound
	precache_sound ("player/udeath.wav");		// player gib sound
	precache_sound ("player/tornoff2.wav");		// gib sound

// player pain sounds

	precache_sound ("player/pain1.wav");
	precache_sound ("player/pain2.wav");
	precache_sound ("player/pain3.wav");
	precache_sound ("player/pain4.wav");
	precache_sound ("player/pain5.wav");
	precache_sound ("player/pain6.wav");

// player death sounds
	precache_sound ("player/death1.wav");
	precache_sound ("player/death2.wav");
	precache_sound ("player/death3.wav");
	precache_sound ("player/death4.wav");
	precache_sound ("player/death5.wav");

// ax sounds
	precache_sound ("weapons/ax1.wav");		// ax swoosh
	precache_sound ("player/axhit1.wav");		// ax hit meat
	precache_sound ("player/axhit2.wav");		// ax hit world

	precache_sound ("player/h2ojump.wav");		// player jumping into water
	precache_sound ("player/slimbrn2.wav");		// player enter slime
```

```
	precache_sound ("player/inh2o.wav");		// player enter water
	precache_sound ("player/inlava.wav");		// player enter lava
	precache_sound ("misc/outwater.wav");		// leaving water sound

	precache_sound ("player/lburn1.wav");		// lava burn
	precache_sound ("player/lburn2.wav");		// lava burn

	precache_sound ("misc/water1.wav");			// swimming
	precache_sound ("misc/water2.wav");			// swimming

	precache_model ("progs/player.mdl");
	precache_model ("progs/eyes.mdl");
	precache_model ("progs/h_player.mdl");
	precache_model ("progs/gib1.mdl");
	precache_model ("progs/gib2.mdl");
	precache_model ("progs/gib3.mdl");

	precache_model ("progs/s_bubble.spr"); // drowning bubbles
	precache_model ("progs/s_explod.spr"); // sprite explosion

	precache_model ("progs/v_axe.mdl");
	precache_model ("progs/v_shot.mdl");
	precache_model ("progs/v_nail.mdl");
	precache_model ("progs/v_rock.mdl");
	precache_model ("progs/v_shot2.mdl");
	precache_model ("progs/v_nail2.mdl");
	precache_model ("progs/v_rock2.mdl");

	precache_model ("progs/bolt.mdl");		// for lightning gun
	precache_model ("progs/bolt2.mdl");		// for lightning gun
	precache_model ("progs/bolt3.mdl");		// for boss shock
	precache_model ("progs/lavaball.mdl");	// for testing

	precache_model ("progs/missile.mdl");
	precache_model ("progs/grenade.mdl");
	precache_model ("progs/spike.mdl");
	precache_model ("progs/s_spike.mdl");

	precache_model ("progs/backpack.mdl");

	precache_model ("progs/zom_gib.mdl");

	precache_model ("progs/v_light.mdl");


//
// Setup light animation tables. 'a' is total darkness, 'z' is maxbright.
//

	// 0 normal
	lightstyle(0, "m");

	// 1 FLICKER (first variety)
	lightstyle(1, "mmnmmommommnonmmonqnmmo");

	// 2 SLOW STRONG PULSE
	lightstyle(2, "abcdefghijklmnopqrstuvwxyzyxwvutsrqponmlkjihgfedcba");

	// 3 CANDLE (first variety)
	lightstyle(3, "mmmmmaaaaammmmmaaaaaabcdefgabcdefg");

	// 4 FAST STROBE
```

```
	lightstyle(4, "mamamamamama");

	// 5 GENTLE PULSE 1
	lightstyle(5,"jklmnopqrstuvwxyzyxwvutsrqponmlkj");

	// 6 FLICKER (second variety)
	lightstyle(6, "nmonqnmomnmomomno");

	// 7 CANDLE (second variety)
	lightstyle(7, "mmmaaaabcdefgmmmmaaaammmaamm");

	// 8 CANDLE (third variety)
	lightstyle(8, "mmmaaammmaaammmabcdefaaaammmmabcdefmmmaaaa");

	// 9 SLOW STROBE (fourth variety)
	lightstyle(9, "aaaaaaaazzzzzzzz");

	// 10 FLUORESCENT FLICKER
	lightstyle(10, "mmamammmmammamamaaamammma");

	// 11 SLOW PULSE NOT FADE TO BLACK
	lightstyle(11, "abcdefghijklmnopqrrqponmlkjihgfedcba");

	// styles 32-62 are assigned by the light program for switchable lights

	// 63 testing
	lightstyle(63, "a");
};

void() StartFrame =
{
	teamplay = cvar("teamplay");
	skill = cvar("skill");
	framecount = framecount + 1;
};

/*
==============================================================================

BODY QUE

==============================================================================
*/

entity	bodyque_head;

void() bodyque =
{	// just here so spawn functions don't complain after the world
	// creates bodyques
};

void() InitBodyQue =
{
	local entity	e;

	bodyque_head = spawn();
	bodyque_head.classname = "bodyque";
	bodyque_head.owner = spawn();
	bodyque_head.owner.classname = "bodyque";
	bodyque_head.owner.owner = spawn();
	bodyque_head.owner.owner.classname = "bodyque";
	bodyque_head.owner.owner.owner = spawn();
```

```
	bodyque_head.owner.owner.owner.classname = "bodyque";
	bodyque_head.owner.owner.owner.owner = bodyque_head;
};


// make a body que entry for the given ent so the ent can be
// respawned elsewhere
void(entity ent) CopyToBodyQue =
{
	bodyque_head.angles = ent.angles;
	bodyque_head.model = ent.model;
	bodyque_head.modelindex = ent.modelindex;
	bodyque_head.frame = ent.frame;
	bodyque_head.colormap = ent.colormap;
	bodyque_head.movetype = ent.movetype;
	bodyque_head.velocity = ent.velocity;
	bodyque_head.flags = 0;
	setorigin (bodyque_head, ent.origin);
	setsize (bodyque_head, ent.mins, ent.maxs);
	bodyque_head = bodyque_head.owner;
};
```

BOSS.QC (CHTHON)

```
/*
==============================================================================

BOSS-ONE

==============================================================================
*/
$cd id1/models/boss1
$origin 0 0 -15
$base base
$skin skin
$scale 5

$frame rise1 rise2 rise3 rise4 rise5 rise6 rise7 rise8 rise9 rise10
$frame rise11 rise12 rise13 rise14 rise15 rise16 rise17

$frame walk1 walk2 walk3 walk4 walk5 walk6 walk7 walk8
$frame walk9 walk10 walk11 walk12 walk13 walk14 walk15
$frame walk16 walk17 walk18 walk19 walk20 walk21 walk22
$frame walk23 walk24 walk25 walk26 walk27 walk28 walk29 walk30 walk31

$frame death1 death2 death3 death4 death5 death6 death7 death8 death9

$frame attack1 attack2 attack3 attack4 attack5 attack6 attack7 attack8
$frame attack9 attack10 attack11 attack12 attack13 attack14 attack15
$frame attack16 attack17 attack18 attack19 attack20 attack21 attack22
$frame attack23

$frame shocka1 shocka2 shocka3 shocka4 shocka5 shocka6 shocka7 shocka8
$frame shocka9 shocka10

$frame shockb1 shockb2 shockb3 shockb4 shockb5 shockb6

$frame shockc1 shockc2 shockc3 shockc4 shockc5 shockc6 shockc7 shockc8
$frame shockc9 shockc10


void(vector p) boss_missile;

void() boss_face =
{

// go for another player if multi player
        if (self.enemy.health <= 0 || random() < 0.02)
        {
                self.enemy = find(self.enemy, classname, "player");
                if (!self.enemy)
                        self.enemy = find(self.enemy, classname, "player");
        }
        ai_face();
};

void() boss_rise1       =[      $rise1, boss_rise2 ] {
sound (self, CHAN_WEAPON, "boss1/out1.wav", 1, ATTN_NORM);
};
void() boss_rise2       =[      $rise2, boss_rise3 ] {
sound (self, CHAN_VOICE, "boss1/sight1.wav", 1, ATTN_NORM);
};
void() boss_rise3       =[      $rise3, boss_rise4 ] {};
void() boss_rise4       =[      $rise4, boss_rise5 ] {};
```

```
void() boss_rise5       =[      $rise5, boss_rise6 ] {};
void() boss_rise6       =[      $rise6, boss_rise7 ] {};
void() boss_rise7       =[      $rise7, boss_rise8 ] {};
void() boss_rise8       =[      $rise8, boss_rise9 ] {};
void() boss_rise9       =[      $rise9, boss_rise10 ] {};
void() boss_rise10      =[      $rise10, boss_rise11 ] {};
void() boss_rise11      =[      $rise11, boss_rise12 ] {};
void() boss_rise12      =[      $rise12, boss_rise13 ] {};
void() boss_rise13      =[      $rise13, boss_rise14 ] {};
void() boss_rise14      =[      $rise14, boss_rise15 ] {};
void() boss_rise15      =[      $rise15, boss_rise16 ] {};
void() boss_rise16      =[      $rise16, boss_rise17 ] {};
void() boss_rise17      =[      $rise17, boss_missile1 ] {};

void() boss_idle1       =[      $walk1, boss_idle2 ]
{
// look for other players
};
void() boss_idle2       =[      $walk2, boss_idle3 ] {boss_face();};
void() boss_idle3       =[      $walk3, boss_idle4 ] {boss_face();};
void() boss_idle4       =[      $walk4, boss_idle5 ] {boss_face();};
void() boss_idle5       =[      $walk5, boss_idle6 ] {boss_face();};
void() boss_idle6       =[      $walk6, boss_idle7 ] {boss_face();};
void() boss_idle7       =[      $walk7, boss_idle8 ] {boss_face();};
void() boss_idle8       =[      $walk8, boss_idle9 ] {boss_face();};
void() boss_idle9       =[      $walk9, boss_idle10 ] {boss_face();};
void() boss_idle10      =[      $walk10, boss_idle11 ] {boss_face();};
void() boss_idle11      =[      $walk11, boss_idle12 ] {boss_face();};
void() boss_idle12      =[      $walk12, boss_idle13 ] {boss_face();};
void() boss_idle13      =[      $walk13, boss_idle14 ] {boss_face();};
void() boss_idle14      =[      $walk14, boss_idle15 ] {boss_face();};
void() boss_idle15      =[      $walk15, boss_idle16 ] {boss_face();};
void() boss_idle16      =[      $walk16, boss_idle17 ] {boss_face();};
void() boss_idle17      =[      $walk17, boss_idle18 ] {boss_face();};
void() boss_idle18      =[      $walk18, boss_idle19 ] {boss_face();};
void() boss_idle19      =[      $walk19, boss_idle20 ] {boss_face();};
void() boss_idle20      =[      $walk20, boss_idle21 ] {boss_face();};
void() boss_idle21      =[      $walk21, boss_idle22 ] {boss_face();};
void() boss_idle22      =[      $walk22, boss_idle23 ] {boss_face();};
void() boss_idle23      =[      $walk23, boss_idle24 ] {boss_face();};
void() boss_idle24      =[      $walk24, boss_idle25 ] {boss_face();};
void() boss_idle25      =[      $walk25, boss_idle26 ] {boss_face();};
void() boss_idle26      =[      $walk26, boss_idle27 ] {boss_face();};
void() boss_idle27      =[      $walk27, boss_idle28 ] {boss_face();};
void() boss_idle28      =[      $walk28, boss_idle29 ] {boss_face();};
void() boss_idle29      =[      $walk29, boss_idle30 ] {boss_face();};
void() boss_idle30      =[      $walk30, boss_idle31 ] {boss_face();};
void() boss_idle31      =[      $walk31, boss_idle1 ] {boss_face();};

void() boss_missile1    =[      $attack1, boss_missile2 ] {boss_face();};
void() boss_missile2    =[      $attack2, boss_missile3 ] {boss_face();};
void() boss_missile3    =[      $attack3, boss_missile4 ] {boss_face();};
void() boss_missile4    =[      $attack4, boss_missile5 ] {boss_face();};
void() boss_missile5    =[      $attack5, boss_missile6 ] {boss_face();};
void() boss_missile6    =[      $attack6, boss_missile7 ] {boss_face();};
void() boss_missile7    =[      $attack7, boss_missile8 ] {boss_face();};
void() boss_missile8    =[      $attack8, boss_missile9 ] {boss_face();};
void() boss_missile9    =[      $attack9, boss_missile10 ] {boss_missile('100 100 200');};
void() boss_missile10   =[      $attack10, boss_missile11 ] {boss_face();};
void() boss_missile11   =[      $attack11, boss_missile12 ] {boss_face();};
void() boss_missile12   =[      $attack12, boss_missile13 ] {boss_face();};
void() boss_missile13   =[      $attack13, boss_missile14 ] {boss_face();};
```

```
void() boss_missile14	=[	$attack14, boss_missile15 ] {boss_face();};
void() boss_missile15	=[	$attack15, boss_missile16 ] {boss_face();};
void() boss_missile16	=[	$attack16, boss_missile17 ] {boss_face();};
void() boss_missile17	=[	$attack17, boss_missile18 ] {boss_face();};
void() boss_missile18	=[	$attack18, boss_missile19 ] {boss_face();};
void() boss_missile19	=[	$attack19, boss_missile20 ] {boss_face();};
void() boss_missile20	=[	$attack20, boss_missile21 ] {boss_missile('100 -100 200');};
void() boss_missile21	=[	$attack21, boss_missile22 ] {boss_face();};
void() boss_missile22	=[	$attack22, boss_missile23 ] {boss_face();};
void() boss_missile23	=[	$attack23, boss_missile1 ] {boss_face();};

void() boss_shocka1 =[	$shocka1, boss_shocka2 ] {};
void() boss_shocka2 =[	$shocka2, boss_shocka3 ] {};
void() boss_shocka3 =[	$shocka3, boss_shocka4 ] {};
void() boss_shocka4 =[	$shocka4, boss_shocka5 ] {};
void() boss_shocka5 =[	$shocka5, boss_shocka6 ] {};
void() boss_shocka6 =[	$shocka6, boss_shocka7 ] {};
void() boss_shocka7 =[	$shocka7, boss_shocka8 ] {};
void() boss_shocka8 =[	$shocka8, boss_shocka9 ] {};
void() boss_shocka9 =[	$shocka9, boss_shocka10 ] {};
void() boss_shocka10 =[		$shocka10, boss_missile1 ] {};

void() boss_shockb1 =[	$shockb1, boss_shockb2 ] {};
void() boss_shockb2 =[	$shockb2, boss_shockb3 ] {};
void() boss_shockb3 =[	$shockb3, boss_shockb4 ] {};
void() boss_shockb4 =[	$shockb4, boss_shockb5 ] {};
void() boss_shockb5 =[	$shockb5, boss_shockb6 ] {};
void() boss_shockb6 =[	$shockb6, boss_shockb7 ] {};
void() boss_shockb7 =[	$shockb1, boss_shockb8 ] {};
void() boss_shockb8 =[	$shockb2, boss_shockb9 ] {};
void() boss_shockb9 =[	$shockb3, boss_shockb10 ] {};
void() boss_shockb10 =[		$shockb4, boss_missile1 ] {};

void() boss_shockc1 =[	$shockc1, boss_shockc2 ] {};
void() boss_shockc2 =[	$shockc2, boss_shockc3 ] {};
void() boss_shockc3 =[	$shockc3, boss_shockc4 ] {};
void() boss_shockc4 =[	$shockc4, boss_shockc5 ] {};
void() boss_shockc5 =[	$shockc5, boss_shockc6 ] {};
void() boss_shockc6 =[	$shockc6, boss_shockc7 ] {};
void() boss_shockc7 =[	$shockc7, boss_shockc8 ] {};
void() boss_shockc8 =[	$shockc8, boss_shockc9 ] {};
void() boss_shockc9 =[	$shockc9, boss_shockc10 ] {};
void() boss_shockc10 =[		$shockc10, boss_death1 ] {};

void() boss_death1 = [$death1, boss_death2] {
sound (self, CHAN_VOICE, "boss1/death.wav", 1, ATTN_NORM);
};
void() boss_death2 = [$death2, boss_death3] {};
void() boss_death3 = [$death3, boss_death4] {};
void() boss_death4 = [$death4, boss_death5] {};
void() boss_death5 = [$death5, boss_death6] {};
void() boss_death6 = [$death6, boss_death7] {};
void() boss_death7 = [$death7, boss_death8] {};
void() boss_death8 = [$death8, boss_death9] {};
void() boss_death9 = [$death9, boss_death10]
{
	sound (self, CHAN_BODY, "boss1/out1.wav", 1, ATTN_NORM);
	WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
	WriteByte (MSG_BROADCAST, TE_LAVASPLASH);
	WriteCoord (MSG_BROADCAST, self.origin_x);
	WriteCoord (MSG_BROADCAST, self.origin_y);
	WriteCoord (MSG_BROADCAST, self.origin_z);
```

```
};

void() boss_death10 = [$death9, boss_death10]
{
	killed_monsters = killed_monsters + 1;
	WriteByte (MSG_ALL, SVC_KILLEDMONSTER);		// FIXME: reliable broadcast
	SUB_UseTargets ();
	remove (self);
};

void(vector p) boss_missile =
{
	local	vector	offang;
	local	vector	org, vec, d;
	local	float	t;

	offang = vectoangles (self.enemy.origin - self.origin);
	makevectors (offang);

	org = self.origin + p_x*v_forward + p_y*v_right + p_z*'0 0 1';

// lead the player on hard mode
	if (skill > 1)
	{
		t = vlen(self.enemy.origin - org) / 300;
		vec = self.enemy.velocity;
		vec_z = 0;
		d = self.enemy.origin + t * vec;
	}
	else
	{
		d = self.enemy.origin;
	}

	vec = normalize (d - org);

	launch_spike (org, vec);
	setmodel (newmis, "progs/lavaball.mdl");
	newmis.avelocity = '200 100 300';
	setsize (newmis, VEC_ORIGIN, VEC_ORIGIN);
	newmis.velocity = vec*300;
	newmis.touch = T_MissileTouch; // rocket explosion
	sound (self, CHAN_WEAPON, "boss1/throw.wav", 1, ATTN_NORM);

// check for dead enemy
	if (self.enemy.health <= 0)
		boss_idle1 ();
};


void() boss_awake =
{
	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;
	self.takedamage = DAMAGE_NO;

	setmodel (self, "progs/boss.mdl");
	setsize (self, '-128 -128 -24', '128 128 256');

	if (skill == 0)
		self.health = 1;
	else
```

```
		self.health = 3;

	self.enemy = activator;

	WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
	WriteByte (MSG_BROADCAST, TE_LAVASPLASH);
	WriteCoord (MSG_BROADCAST, self.origin_x);
	WriteCoord (MSG_BROADCAST, self.origin_y);
	WriteCoord (MSG_BROADCAST, self.origin_z);

	self.yaw_speed = 20;
	boss_rise1 ();
};


/*QUAKED monster_boss (1 0 0) (-128 -128 -24) (128 128 256)
*/
void() monster_boss =
{
	if (deathmatch)
	{
		remove(self);
		return;
	}
	precache_model ("progs/boss.mdl");
	precache_model ("progs/lavaball.mdl");

	precache_sound ("weapons/rocket1i.wav");
	precache_sound ("boss1/out1.wav");
	precache_sound ("boss1/sight1.wav");
	precache_sound ("misc/power.wav");
	precache_sound ("boss1/throw.wav");
	precache_sound ("boss1/pain.wav");
	precache_sound ("boss1/death.wav");

	total_monsters = total_monsters + 1;

	self.use = boss_awake;
};

//============================================================================

entity	le1, le2;
float	lightning_end;

void() lightning_fire =
{
	local vector p1, p2;

	if (time >= lightning_end)
	{	// done here, put the terminals back up
		self = le1;
		door_go_down ();
		self = le2;
		door_go_down ();
		return;
	}

	p1 = (le1.mins + le1.maxs) * 0.5;
	p1_z = le1.absmin_z - 16;

	p2 = (le2.mins + le2.maxs) * 0.5;
```

```
	p2_z = le2.absmin_z - 16;

	// compensate for length of bolt
	p2 = p2 - normalize(p2-p1)*100;

	self.nextthink = time + 0.1;
	self.think = lightning_fire;

	WriteByte (MSG_ALL, SVC_TEMPENTITY);
	WriteByte (MSG_ALL, TE_LIGHTNING3);
	WriteEntity (MSG_ALL, world);
	WriteCoord (MSG_ALL, p1_x);
	WriteCoord (MSG_ALL, p1_y);
	WriteCoord (MSG_ALL, p1_z);
	WriteCoord (MSG_ALL, p2_x);
	WriteCoord (MSG_ALL, p2_y);
	WriteCoord (MSG_ALL, p2_z);
};

void() lightning_use =
{
	if (lightning_end >= time + 1)
		return;

	le1 = find( world, target, "lightning");
	le2 = find( le1, target, "lightning");
	if (!le1 || !le2)
	{
		dprint ("missing lightning targets\n");
		return;
	}

	if (
	 (le1.state != STATE_TOP && le1.state != STATE_BOTTOM)
	|| (le2.state != STATE_TOP && le2.state != STATE_BOTTOM)
	|| (le1.state != le2.state) )
	{
//		dprint ("not aligned\n");
		return;
	}

// don't let the electrodes go back up until the bolt is done
	le1.nextthink = -1;
	le2.nextthink = -1;
	lightning_end = time + 1;

	sound (self, CHAN_VOICE, "misc/power.wav", 1, ATTN_NORM);
	lightning_fire ();

// advance the boss pain if down
	self = find (world, classname, "monster_boss");
	if (!self)
		return;
	self.enemy = activator;
	if (le1.state == STATE_TOP && self.health > 0)
	{
		sound (self, CHAN_VOICE, "boss1/pain.wav", 1, ATTN_NORM);
		self.health = self.health - 1;
		if (self.health >= 2)
			boss_shocka1();
		else if (self.health == 1)
			boss_shockb1();
```

```
                else if (self.health == 0)
                        boss_shockc1();
        }
};


/*QUAKED event_lightning (0 1 1) (-16 -16 -16) (16 16 16)
Just for boss level.
*/
void() event_lightning =
{
        self.use = lightning_use;
};
```

DEMON.QC (FIEND)

```
/*
==============================================================================

DEMON

==============================================================================
*/

$cd id1/models/demon3
$scale   0.8
$origin 0 0 24
$base base
$skin base

$frame stand1 stand2 stand3 stand4 stand5 stand6 stand7 stand8 stand9
$frame stand10 stand11 stand12 stand13

$frame walk1 walk2 walk3 walk4 walk5 walk6 walk7 walk8

$frame run1 run2 run3 run4 run5 run6

$frame leap1 leap2 leap3 leap4 leap5 leap6 leap7 leap8 leap9 leap10
$frame leap11 leap12

$frame pain1 pain2 pain3 pain4 pain5 pain6

$frame death1 death2 death3 death4 death5 death6 death7 death8 death9

$frame attacka1 attacka2 attacka3 attacka4 attacka5 attacka6 attacka7 attacka8
$frame attacka9 attacka10 attacka11 attacka12 attacka13 attacka14 attacka15

//==========================================================================

void()    Demon_JumpTouch;

void()    demon1_stand1=[       $stand1,          demon1_stand2] {ai_stand();};
void()    demon1_stand2=[       $stand2,          demon1_stand3] {ai_stand();};
void()    demon1_stand3=[       $stand3,          demon1_stand4] {ai_stand();};
void()    demon1_stand4=[       $stand4,          demon1_stand5] {ai_stand();};
void()    demon1_stand5=[       $stand5,          demon1_stand6] {ai_stand();};
void()    demon1_stand6=[       $stand6,          demon1_stand7] {ai_stand();};
void()    demon1_stand7=[       $stand7,          demon1_stand8] {ai_stand();};
void()    demon1_stand8=[       $stand8,          demon1_stand9] {ai_stand();};
void()    demon1_stand9=[       $stand9,          demon1_stand10          ] {ai_stand();};
void()    demon1_stand10        =[        $stand10,         demon1_stand11          ] {ai_stand();};
void()    demon1_stand11        =[        $stand11,         demon1_stand12          ] {ai_stand();};
void()    demon1_stand12        =[        $stand12,         demon1_stand13          ] {ai_stand();};
void()    demon1_stand13        =[        $stand13,         demon1_stand1] {ai_stand();};

void()    demon1_walk1  =[       $walk1,           demon1_walk2  ] {
if (random() < 0.2)
    sound (self, CHAN_VOICE, "demon/idle1.wav", 1, ATTN_IDLE);
ai_walk(8);
};
void()    demon1_walk2  =[       $walk2,           demon1_walk3  ] {ai_walk(6);};
void()    demon1_walk3  =[       $walk3,           demon1_walk4  ] {ai_walk(6);};
void()    demon1_walk4  =[       $walk4,           demon1_walk5  ] {ai_walk(7);};
void()    demon1_walk5  =[       $walk5,           demon1_walk6  ] {ai_walk(4);};
void()    demon1_walk6  =[       $walk6,           demon1_walk7  ] {ai_walk(6);};
void()    demon1_walk7  =[       $walk7,           demon1_walk8  ] {ai_walk(10);};
```

```
void()	demon1_walk8 =[		$walk8,			demon1_walk1	] {ai_walk(10);};

void()	demon1_run1	=[		$run1,			demon1_run2	] {
if (random() < 0.2)
   sound (self, CHAN_VOICE, "demon/idle1.wav", 1, ATTN_IDLE);
ai_run(20);};
void()	demon1_run2	=[		$run2,			demon1_run3	] {ai_run(15);};
void()	demon1_run3	=[		$run3,			demon1_run4	] {ai_run(36);};
void()	demon1_run4	=[		$run4,			demon1_run5	] {ai_run(20);};
void()	demon1_run5	=[		$run5,			demon1_run6	] {ai_run(15);};
void()	demon1_run6	=[		$run6,			demon1_run1	] {ai_run(36);};

void()	demon1_jump1 =[		$leap1,			demon1_jump2 ] {ai_face();};
void()	demon1_jump2 =[		$leap2,			demon1_jump3 ] {ai_face();};
void()	demon1_jump3 =[		$leap3,			demon1_jump4 ] {ai_face();};
void()	demon1_jump4 =[		$leap4,			demon1_jump5 ]
{
	ai_face();

	self.touch = Demon_JumpTouch;
	makevectors (self.angles);
	self.origin_z = self.origin_z + 1;
	self.velocity = v_forward * 600 + '0 0 250';
	if (self.flags & FL_ONGROUND)
		self.flags = self.flags - FL_ONGROUND;
};
void()	demon1_jump5 =[		$leap5,			demon1_jump6 ] {};
void()	demon1_jump6 =[		$leap6,			demon1_jump7 ] {};
void()	demon1_jump7 =[		$leap7,			demon1_jump8 ] {};
void()	demon1_jump8 =[		$leap8,			demon1_jump9 ] {};
void()	demon1_jump9 =[		$leap9,			demon1_jump10		] {};
void()	demon1_jump10		=[	$leap10,		demon1_jump1 ] {
self.nextthink = time + 3;
// if three seconds pass, assume demon is stuck and jump again
};

void()	demon1_jump11		=[	$leap11,		demon1_jump12		] {};
void()	demon1_jump12		=[	$leap12,		demon1_run1	] {};


void()	demon1_atta1	=[		$attacka1,			demon1_atta2	] {ai_charge(4);};
void()	demon1_atta2	=[		$attacka2,			demon1_atta3	] {ai_charge(0);};
void()	demon1_atta3	=[		$attacka3,			demon1_atta4	] {ai_charge(0);};
void()	demon1_atta4	=[		$attacka4,			demon1_atta5	] {ai_charge(1);};
void()	demon1_atta5	=[		$attacka5,			demon1_atta6	] {ai_charge(2); Demon_Melee(200);};
void()	demon1_atta6	=[		$attacka6,			demon1_atta7	] {ai_charge(1);};
void()	demon1_atta7	=[		$attacka7,			demon1_atta8	] {ai_charge(6);};
void()	demon1_atta8	=[		$attacka8,			demon1_atta9	] {ai_charge(8);};
void()	demon1_atta9	=[		$attacka9,			demon1_atta10] {ai_charge(4);};
void()	demon1_atta10 =[		$attacka10,			demon1_atta11] {ai_charge(2);};
void()	demon1_atta11 =[		$attacka11,			demon1_atta12] {Demon_Melee(-200);};
void()	demon1_atta12 =[		$attacka12,			demon1_atta13] {ai_charge(5);};
void()	demon1_atta13 =[		$attacka13,			demon1_atta14] {ai_charge(8);};
void()	demon1_atta14 =[		$attacka14,			demon1_atta15] {ai_charge(4);};
void()	demon1_atta15 =[		$attacka15,			demon1_run1] {ai_charge(4);};

void()	demon1_pain1	=[		$pain1,			demon1_pain2	] {};
void()	demon1_pain2	=[		$pain2,			demon1_pain3	] {};
void()	demon1_pain3	=[		$pain3,			demon1_pain4	] {};
void()	demon1_pain4	=[		$pain4,			demon1_pain5	] {};
void()	demon1_pain5	=[		$pain5,			demon1_pain6	] {};
void()	demon1_pain6	=[		$pain6,			demon1_run1	] {};
```

```
void(entity attacker, float damage)		demon1_pain =
{
	if (self.touch == Demon_JumpTouch)
		return;

	if (self.pain_finished > time)
		return;

	self.pain_finished = time + 1;
   sound (self, CHAN_VOICE, "demon/dpain1.wav", 1, ATTN_NORM);

	if (random()*200 > damage)
		return;		// didn't flinch

	demon1_pain1 ();
};

void()	demon1_die1		=[	$death1,		demon1_die2	] {
sound (self, CHAN_VOICE, "demon/ddeath.wav", 1, ATTN_NORM);};
void()	demon1_die2		=[	$death2,		demon1_die3	] {};
void()	demon1_die3		=[	$death3,		demon1_die4	] {};
void()	demon1_die4		=[	$death4,		demon1_die5	] {};
void()	demon1_die5		=[	$death5,		demon1_die6	] {};
void()	demon1_die6		=[	$death6,		demon1_die7	]
{self.solid = SOLID_NOT;};
void()	demon1_die7		=[	$death7,		demon1_die8	] {};
void()	demon1_die8		=[	$death8,		demon1_die9	] {};
void()	demon1_die9		=[	$death9,		demon1_die9 ] {};

void() demon_die =
{
// check for gib
	if (self.health < -80)
	{
		sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NORM);
		ThrowHead ("progs/h_demon.mdl", self.health);
		ThrowGib ("progs/gib1.mdl", self.health);
		ThrowGib ("progs/gib1.mdl", self.health);
		ThrowGib ("progs/gib1.mdl", self.health);
		return;
	}

// regular death
	demon1_die1 ();
};


void() Demon_MeleeAttack =
{
	demon1_atta1 ();
};


/*QUAKED monster_demon1 (1 0 0) (-32 -32 -24) (32 32 64) Ambush

*/
void() monster_demon1 =
{
	if (deathmatch)
	{
		remove(self);
```

```
		return;
	}
	precache_model ("progs/demon.mdl");
	precache_model ("progs/h_demon.mdl");

	precache_sound ("demon/ddeath.wav");
	precache_sound ("demon/dhit2.wav");
	precache_sound ("demon/djump.wav");
	precache_sound ("demon/dpain1.wav");
	precache_sound ("demon/idle1.wav");
	precache_sound ("demon/sight2.wav");

	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;

	setmodel (self, "progs/demon.mdl");

	setsize (self, VEC_HULL2_MIN, VEC_HULL2_MAX);
	self.health = 300;

	self.th_stand = demon1_stand1;
	self.th_walk = demon1_walk1;
	self.th_run = demon1_run1;
	self.th_die = demon_die;
	self.th_melee = Demon_MeleeAttack;		// one of two attacks
	self.th_missile = demon1_jump1;				// jump attack
	self.th_pain = demon1_pain;

	walkmonster_start();
};


/*
==============================================================================

DEMON

==============================================================================
*/

/*
==============
CheckDemonMelee

Returns TRUE if a melee attack would hit right now
==============
*/
float()	CheckDemonMelee =
{
	if (enemy_range == RANGE_MELEE)
	{	// FIXME: check canreach
		self.attack_state = AS_MELEE;
		return TRUE;
	}
	return FALSE;
};

/*
==============
CheckDemonJump

==============
```

```
*/
float()	CheckDemonJump =
{
	local	vector	dist;
	local	float	d;

	if (self.origin_z + self.mins_z > self.enemy.origin_z + self.enemy.mins_z
	+ 0.75 * self.enemy.size_z)
		return FALSE;

	if (self.origin_z + self.maxs_z < self.enemy.origin_z + self.enemy.mins_z
	+ 0.25 * self.enemy.size_z)
		return FALSE;

	dist = self.enemy.origin - self.origin;
	dist_z = 0;

	d = vlen(dist);

	if (d < 100)
		return FALSE;

	if (d > 200)
	{
		if (random() < 0.9)
			return FALSE;
	}

	return TRUE;
};

float()	DemonCheckAttack =
{
	local	vector	vec;

// if close enough for slashing, go for it
	if (CheckDemonMelee ())
	{
		self.attack_state = AS_MELEE;
		return TRUE;
	}

	if (CheckDemonJump ())
	{
		self.attack_state = AS_MISSILE;
      sound (self, CHAN_VOICE, "demon/djump.wav", 1, ATTN_NORM);
		return TRUE;
	}

	return FALSE;
};


//============================================================================

void(float side)	Demon_Melee =
{
	local	float	ldmg;
	local vector	delta;

	ai_face ();
	walkmove (self.ideal_yaw, 12);	// allow a little closing
```

```
	delta = self.enemy.origin - self.origin;

	if (vlen(delta) > 100)
		return;
	if (!CanDamage (self.enemy, self))
		return;

   sound (self, CHAN_WEAPON, "demon/dhit2.wav", 1, ATTN_NORM);
	ldmg = 10 + 5*random();
	T_Damage (self.enemy, self, self, ldmg);

	makevectors (self.angles);
	SpawnMeatSpray (self.origin + v_forward*16, side * v_right);
};


void()	Demon_JumpTouch =
{
	local	float	ldmg;

	if (self.health <= 0)
		return;

	if (other.takedamage)
	{
		if ( vlen(self.velocity) > 400 )
		{
			ldmg = 40 + 10*random();
			T_Damage (other, self, self, ldmg);
		}
	}

	if (!checkbottom(self))
	{
		if (self.flags & FL_ONGROUND)
		{		// jump randomly to not get hung up
//dprint ("popjump\n");
	self.touch = SUB_Null;
	self.think = demon1_jump1;
	self.nextthink = time + 0.1;

//				self.velocity_x = (random() - 0.5) * 600;
//				self.velocity_y = (random() - 0.5) * 600;
//				self.velocity_z = 200;
//				self.flags = self.flags - FL_ONGROUND;
		}
		return;  // not on ground yet
	}

	self.touch = SUB_Null;
	self.think = demon1_jump11;
	self.nextthink = time + 0.1;
};
```

DOG.QC

```
/*
==============================================================================

DOG

==============================================================================
*/
$cd id1/models/dog
$origin 0 0 24
$base base
$skin skin

$frame attack1 attack2 attack3 attack4 attack5 attack6 attack7 attack8

$frame death1 death2 death3 death4 death5 death6 death7 death8 death9

$frame deathb1 deathb2 deathb3 deathb4 deathb5 deathb6 deathb7 deathb8
$frame deathb9

$frame pain1 pain2 pain3 pain4 pain5 pain6

$frame painb1 painb2 painb3 painb4 painb5 painb6 painb7 painb8 painb9 painb10
$frame painb11 painb12 painb13 painb14 painb15 painb16

$frame run1 run2 run3 run4 run5 run6 run7 run8 run9 run10 run11 run12

$frame leap1 leap2 leap3 leap4 leap5 leap6 leap7 leap8 leap9

$frame stand1 stand2 stand3 stand4 stand5 stand6 stand7 stand8 stand9

$frame walk1 walk2 walk3 walk4 walk5 walk6 walk7 walk8


void() dog_leap1;
void() dog_run1;

/*
================
dog_bite

================
*/
void() dog_bite =
{
local vector       delta;
local float        ldmg;

        if (!self.enemy)
                return;

        ai_charge(10);

        if (!CanDamage (self.enemy, self))
                return;

        delta = self.enemy.origin - self.origin;

        if (vlen(delta) > 100)
                return;
```

```
	ldmg = (random() + random() + random()) * 8;
	T_Damage (self.enemy, self, self, ldmg);
};

void()	Dog_JumpTouch =
{
	local	float	ldmg;

	if (self.health <= 0)
		return;

	if (other.takedamage)
	{
		if ( vlen(self.velocity) > 300 )
		{
			ldmg = 10 + 10*random();
			T_Damage (other, self, self, ldmg);
		}
	}

	if (!checkbottom(self))
	{
		if (self.flags & FL_ONGROUND)
		{	// jump randomly to not get hung up
//dprint ("popjump\n");
	self.touch = SUB_Null;
	self.think = dog_leap1;
	self.nextthink = time + 0.1;

//				self.velocity_x = (random() - 0.5) * 600;
//				self.velocity_y = (random() - 0.5) * 600;
//				self.velocity_z = 200;
//				self.flags = self.flags - FL_ONGROUND;
		}
		return;	// not on ground yet
	}

	self.touch = SUB_Null;
	self.think = dog_run1;
	self.nextthink = time + 0.1;
};


void() dog_stand1	=[	$stand1,	dog_stand2	] {ai_stand();};
void() dog_stand2	=[	$stand2,	dog_stand3	] {ai_stand();};
void() dog_stand3	=[	$stand3,	dog_stand4	] {ai_stand();};
void() dog_stand4	=[	$stand4,	dog_stand5	] {ai_stand();};
void() dog_stand5	=[	$stand5,	dog_stand6	] {ai_stand();};
void() dog_stand6	=[	$stand6,	dog_stand7	] {ai_stand();};
void() dog_stand7	=[	$stand7,	dog_stand8	] {ai_stand();};
void() dog_stand8	=[	$stand8,	dog_stand9	] {ai_stand();};
void() dog_stand9	=[	$stand9,	dog_stand1	] {ai_stand();};

void() dog_walk1	=[	$walk1 ,	dog_walk2	] {
if (random() < 0.2)
	sound (self, CHAN_VOICE, "dog/idle.wav", 1, ATTN_IDLE);
ai_walk(8);};
void() dog_walk2	=[	$walk2 ,	dog_walk3	] {ai_walk(8);};
void() dog_walk3	=[	$walk3 ,	dog_walk4	] {ai_walk(8);};
void() dog_walk4	=[	$walk4 ,	dog_walk5	] {ai_walk(8);};
void() dog_walk5	=[	$walk5 ,	dog_walk6	] {ai_walk(8);};
void() dog_walk6	=[	$walk6 ,	dog_walk7	] {ai_walk(8);};
```

```
void() dog_walk7        =[      $walk7 ,        dog_walk8       ] {ai_walk(8);};
void() dog_walk8        =[      $walk8 ,        dog_walk1       ] {ai_walk(8);};

void() dog_run1         =[      $run1  , dog_run2       ] {
if (random() < 0.2)
        sound (self, CHAN_VOICE, "dog/idle.wav", 1, ATTN_IDLE);
ai_run(16);};
void() dog_run2         =[      $run2  , dog_run3       ] {ai_run(32);};
void() dog_run3         =[      $run3  , dog_run4       ] {ai_run(32);};
void() dog_run4         =[      $run4  , dog_run5       ] {ai_run(20);};
void() dog_run5         =[      $run5  , dog_run6       ] {ai_run(64);};
void() dog_run6         =[      $run6  , dog_run7       ] {ai_run(32);};
void() dog_run7         =[      $run7  , dog_run8       ] {ai_run(16);};
void() dog_run8         =[      $run8  , dog_run9       ] {ai_run(32);};
void() dog_run9         =[      $run9  , dog_run10      ] {ai_run(32);};
void() dog_run10        =[      $run10  ,       dog_run11       ] {ai_run(20);};
void() dog_run11        =[      $run11  ,       dog_run12       ] {ai_run(64);};
void() dog_run12        =[      $run12  ,       dog_run1        ] {ai_run(32);};

void() dog_atta1        =[      $attack1,       dog_atta2       ] {ai_charge(10);};
void() dog_atta2        =[      $attack2,       dog_atta3       ] {ai_charge(10);};
void() dog_atta3        =[      $attack3,       dog_atta4       ] {ai_charge(10);};
void() dog_atta4        =[      $attack4,       dog_atta5       ] {
sound (self, CHAN_VOICE, "dog/dattack1.wav", 1, ATTN_NORM);
dog_bite();};
void() dog_atta5        =[      $attack5,       dog_atta6       ] {ai_charge(10);};
void() dog_atta6        =[      $attack6,       dog_atta7       ] {ai_charge(10);};
void() dog_atta7        =[      $attack7,       dog_atta8       ] {ai_charge(10);};
void() dog_atta8        =[      $attack8,       dog_run1        ] {ai_charge(10);};

void() dog_leap1        =[      $leap1,         dog_leap2       ] {ai_face();};
void() dog_leap2        =[      $leap2,         dog_leap3       ]
{
        ai_face();

        self.touch = Dog_JumpTouch;
        makevectors (self.angles);
        self.origin_z = self.origin_z + 1;
        self.velocity = v_forward * 300 + '0 0 200';
        if (self.flags & FL_ONGROUND)
                self.flags = self.flags - FL_ONGROUND;
};

void() dog_leap3        =[      $leap3,         dog_leap4       ] {};
void() dog_leap4        =[      $leap4,         dog_leap5       ] {};
void() dog_leap5        =[      $leap5,         dog_leap6       ] {};
void() dog_leap6        =[      $leap6,         dog_leap7       ] {};
void() dog_leap7        =[      $leap7,         dog_leap8       ] {};
void() dog_leap8        =[      $leap8,         dog_leap9       ] {};
void() dog_leap9        =[      $leap9,         dog_leap9       ] {};

void() dog_pain1        =[      $pain1 ,dog_pain2       ] {};
void() dog_pain2        =[      $pain2 ,dog_pain3       ] {};
void() dog_pain3        =[      $pain3 ,dog_pain4       ] {};
void() dog_pain4        =[      $pain4 ,dog_pain5       ] {};
void() dog_pain5        =[      $pain5 ,dog_pain6       ] {};
void() dog_pain6        =[      $pain6 ,dog_run1        ] {};

void() dog_painb1       =[      $painb1 ,       dog_painb2      ] {};
void() dog_painb2       =[      $painb2 ,       dog_painb3      ] {};
void() dog_painb3       =[      $painb3 ,       dog_painb4      ] {ai_pain(4);};
void() dog_painb4       =[      $painb4 ,       dog_painb5      ] {ai_pain(12);};
```

```
void() dog_painb5       =[      $painb5 ,       dog_painb6      ] {ai_pain(12);};
void() dog_painb6       =[      $painb6 ,       dog_painb7      ] {ai_pain(2);};
void() dog_painb7       =[      $painb7 ,       dog_painb8      ] {};
void() dog_painb8       =[      $painb8 ,       dog_painb9      ] {ai_pain(4);};
void() dog_painb9       =[      $painb9 ,       dog_painb10     ] {};
void() dog_painb10      =[      $painb10 ,      dog_painb11     ] {ai_pain(10);};
void() dog_painb11      =[      $painb11 ,      dog_painb12     ] {};
void() dog_painb12      =[      $painb12 ,      dog_painb13     ] {};
void() dog_painb13      =[      $painb13 ,      dog_painb14     ] {};
void() dog_painb14      =[      $painb14 ,      dog_painb15     ] {};
void() dog_painb15      =[      $painb15 ,      dog_painb16     ] {};
void() dog_painb16      =[      $painb16 ,      dog_run1        ] {};

void() dog_pain =
{
        sound (self, CHAN_VOICE, "dog/dpain1.wav", 1, ATTN_NORM);

        if (random() > 0.5)
                dog_pain1 ();
        else
                dog_painb1 ();
};

void() dog_die1         =[      $death1,        dog_die2        ] {};
void() dog_die2         =[      $death2,        dog_die3        ] {};
void() dog_die3         =[      $death3,        dog_die4        ] {};
void() dog_die4         =[      $death4,        dog_die5        ] {};
void() dog_die5         =[      $death5,        dog_die6        ] {};
void() dog_die6         =[      $death6,        dog_die7        ] {};
void() dog_die7         =[      $death7,        dog_die8        ] {};
void() dog_die8         =[      $death8,        dog_die9        ] {};
void() dog_die9         =[      $death9,        dog_die9        ] {};

void() dog_dieb1                =[      $deathb1,       dog_dieb2       ] {};
void() dog_dieb2                =[      $deathb2,       dog_dieb3       ] {};
void() dog_dieb3                =[      $deathb3,       dog_dieb4       ] {};
void() dog_dieb4                =[      $deathb4,       dog_dieb5       ] {};
void() dog_dieb5                =[      $deathb5,       dog_dieb6       ] {};
void() dog_dieb6                =[      $deathb6,       dog_dieb7       ] {};
void() dog_dieb7                =[      $deathb7,       dog_dieb8       ] {};
void() dog_dieb8                =[      $deathb8,       dog_dieb9       ] {};
void() dog_dieb9                =[      $deathb9,       dog_dieb9       ] {};


void() dog_die =
{
// check for gib
        if (self.health < -35)
        {
                sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NORM);
                ThrowGib ("progs/gib3.mdl", self.health);
                ThrowGib ("progs/gib3.mdl", self.health);
                ThrowGib ("progs/gib3.mdl", self.health);
                ThrowHead ("progs/h_dog.mdl", self.health);
                return;
        }

// regular death
        sound (self, CHAN_VOICE, "dog/ddeath.wav", 1, ATTN_NORM);
        self.solid = SOLID_NOT;

        if (random() > 0.5)
```

```
                dog_die1 ();
        else
                dog_dieb1 ();
};

//============================================================================

/*
==============
CheckDogMelee

Returns TRUE if a melee attack would hit right now
==============
*/
float()    CheckDogMelee =
{
        if (enemy_range == RANGE_MELEE)
        {       // FIXME: check canreach
                self.attack_state = AS_MELEE;
                return TRUE;
        }
        return FALSE;
};

/*
==============
CheckDogJump

==============
*/
float()    CheckDogJump =
{
        local    vector   dist;
        local    float    d;

        if (self.origin_z + self.mins_z > self.enemy.origin_z + self.enemy.mins_z
        + 0.75 * self.enemy.size_z)
                return FALSE;

        if (self.origin_z + self.maxs_z < self.enemy.origin_z + self.enemy.mins_z
        + 0.25 * self.enemy.size_z)
                return FALSE;

        dist = self.enemy.origin - self.origin;
        dist_z = 0;

        d = vlen(dist);

        if (d < 80)
                return FALSE;

        if (d > 150)
                return FALSE;

        return TRUE;
};

float()    DogCheckAttack =
{
        local    vector   vec;

// if close enough for slashing, go for it
```

```
	if (CheckDogMelee ())
	{
		self.attack_state = AS_MELEE;
		return TRUE;
	}

	if (CheckDogJump ())
	{
		self.attack_state = AS_MISSILE;
		return TRUE;
	}

	return FALSE;
};


//=============================================================================

/*QUAKED monster_dog (1 0 0) (-32 -32 -24) (32 32 40) Ambush

*/
void() monster_dog =
{
	if (deathmatch)
	{
		remove(self);
		return;
	}
	precache_model ("progs/h_dog.mdl");
	precache_model ("progs/dog.mdl");

	precache_sound ("dog/dattack1.wav");
	precache_sound ("dog/ddeath.wav");
	precache_sound ("dog/dpain1.wav");
	precache_sound ("dog/dsight.wav");
	precache_sound ("dog/idle.wav");

	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;

	setmodel (self, "progs/dog.mdl");

	setsize (self, '-32 -32 -24', '32 32 40');
	self.health = 25;

	self.th_stand = dog_stand1;
	self.th_walk = dog_walk1;
	self.th_run = dog_run1;
	self.th_pain = dog_pain;
	self.th_die = dog_die;
	self.th_melee = dog_atta1;
	self.th_missile = dog_leap1;

	walkmonster_start();
};
```

ENFORCER.QC

```
/*
==========================================================================

SOLDIER / PLAYER

==========================================================================
*/

$cd id1/models/enforcer
$origin 0 -6 24
$base base
$skin skin

$frame stand1 stand2 stand3 stand4 stand5 stand6 stand7

$frame walk1 walk2 walk3 walk4 walk5 walk6 walk7 walk8 walk9 walk10
$frame walk11 walk12 walk13 walk14 walk15 walk16

$frame run1 run2 run3 run4 run5 run6 run7 run8

$frame attack1 attack2 attack3 attack4 attack5 attack6
$frame attack7 attack8 attack9 attack10

$frame death1 death2 death3 death4 death5 death6 death7 death8
$frame death9 death10 death11 death12 death13 death14

$frame fdeath1 fdeath2 fdeath3 fdeath4 fdeath5 fdeath6 fdeath7 fdeath8
$frame fdeath9 fdeath10 fdeath11

$frame paina1 paina2 paina3 paina4

$frame painb1 painb2 painb3 painb4 painb5

$frame painc1 painc2 painc3 painc4 painc5 painc6 painc7 painc8

$frame paind1 paind2 paind3 paind4 paind5 paind6 paind7 paind8
$frame paind9 paind10 paind11 paind12 paind13 paind14 paind15 paind16
$frame paind17 paind18 paind19


void() Laser_Touch =
{
	local vector org;

	if (other == self.owner)
		return;		// don't explode on owner

	if (pointcontents(self.origin) == CONTENT_SKY)
	{
		remove(self);
		return;
	}

	sound (self, CHAN_WEAPON, "enforcer/enfstop.wav", 1, ATTN_STATIC);
	org = self.origin - 8*normalize(self.velocity);

	if (other.health)
	{
		SpawnBlood (org, self.velocity*0.2, 15);
		T_Damage (other, self, self.owner, 15);
```

```
	}
	else
	{
		WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
		WriteByte (MSG_BROADCAST, TE_GUNSHOT);
		WriteCoord (MSG_BROADCAST, org_x);
		WriteCoord (MSG_BROADCAST, org_y);
		WriteCoord (MSG_BROADCAST, org_z);
	}

	remove(self);
};

void(vector org, vector vec) LaunchLaser =
{
	local	vector	vec;

	if (self.classname == "monster_enforcer")
		sound (self, CHAN_WEAPON, "enforcer/enfire.wav", 1, ATTN_NORM);

	vec = normalize(vec);

	newmis = spawn();
	newmis.owner = self;
	newmis.movetype = MOVETYPE_FLY;
	newmis.solid = SOLID_BBOX;
	newmis.effects = EF_DIMLIGHT;

	setmodel (newmis, "progs/laser.mdl");
	setsize (newmis, '0 0 0', '0 0 0');

	setorigin (newmis, org);

	newmis.velocity = vec * 600;
	newmis.angles = vectoangles(newmis.velocity);

	newmis.nextthink = time + 5;
	newmis.think = SUB_Remove;
	newmis.touch = Laser_Touch;
};



void() enforcer_fire =
{
	local vector org;

	self.effects = self.effects | EF_MUZZLEFLASH;
	makevectors (self.angles);

	org = self.origin + v_forward * 30 + v_right * 8.5 + '0 0 16';

	LaunchLaser(org, self.enemy.origin - self.origin);
};

//============================================================================

void()	enf_stand1	=[	$stand1,	enf_stand2	] {ai_stand();};
void()	enf_stand2	=[	$stand2,	enf_stand3	] {ai_stand();};
void()	enf_stand3	=[	$stand3,	enf_stand4	] {ai_stand();};
void()	enf_stand4	=[	$stand4,	enf_stand5	] {ai_stand();};
void()	enf_stand5	=[	$stand5,	enf_stand6	] {ai_stand();};
```

```
void()	enf_stand6	=[	$stand6,	enf_stand7	] {ai_stand();};
void()	enf_stand7	=[	$stand7,	enf_stand1	] {ai_stand();};

void()	enf_walk1	=[	$walk1 ,	enf_walk2	] {
if (random() < 0.2)
	sound (self, CHAN_VOICE, "enforcer/idle1.wav", 1, ATTN_IDLE);
ai_walk(2);};
void()	enf_walk2	=[	$walk2 ,	enf_walk3	] {ai_walk(4);};
void()	enf_walk3	=[	$walk3 ,	enf_walk4	] {ai_walk(4);};
void()	enf_walk4	=[	$walk4 ,	enf_walk5	] {ai_walk(3);};
void()	enf_walk5	=[	$walk5 ,	enf_walk6	] {ai_walk(1);};
void()	enf_walk6	=[	$walk6 ,	enf_walk7	] {ai_walk(2);};
void()	enf_walk7	=[	$walk7 ,	enf_walk8	] {ai_walk(2);};
void()	enf_walk8	=[	$walk8 ,	enf_walk9	] {ai_walk(1);};
void()	enf_walk9	=[	$walk9 ,	enf_walk10	] {ai_walk(2);};
void()	enf_walk10	=[	$walk10,	enf_walk11	] {ai_walk(4);};
void()	enf_walk11	=[	$walk11,	enf_walk12	] {ai_walk(4);};
void()	enf_walk12	=[	$walk12,	enf_walk13	] {ai_walk(1);};
void()	enf_walk13	=[	$walk13,	enf_walk14	] {ai_walk(2);};
void()	enf_walk14	=[	$walk14,	enf_walk15	] {ai_walk(3);};
void()	enf_walk15	=[	$walk15,	enf_walk16	] {ai_walk(4);};
void()	enf_walk16	=[	$walk16,	enf_walk1	] {ai_walk(2);};

void()	enf_run1	=[	$run1  , enf_run2	] {
if (random() < 0.2)
	sound (self, CHAN_VOICE, "enforcer/idle1.wav", 1, ATTN_IDLE);
ai_run(18);};
void()	enf_run2	=[	$run2  , enf_run3	] {ai_run(14);};
void()	enf_run3	=[	$run3  , enf_run4	] {ai_run(7);};
void()	enf_run4	=[	$run4  , enf_run5	] {ai_run(12);};
void()	enf_run5	=[	$run5  , enf_run6	] {ai_run(14);};
void()	enf_run6	=[	$run6  , enf_run7	] {ai_run(14);};
void()	enf_run7	=[	$run7  , enf_run8	] {ai_run(7);};
void()	enf_run8	=[	$run8  , enf_run1	] {ai_run(11);};

void()	enf_atk1	=[	$attack1,	enf_atk2	] {ai_face();};
void()	enf_atk2	=[	$attack2,	enf_atk3	] {ai_face();};
void()	enf_atk3	=[	$attack3,	enf_atk4	] {ai_face();};
void()	enf_atk4	=[	$attack4,	enf_atk5	] {ai_face();};
void()	enf_atk5	=[	$attack5,	enf_atk6	] {ai_face();};
void()	enf_atk6	=[	$attack6,	enf_atk7	] {enforcer_fire();};
void()	enf_atk7	=[	$attack7,	enf_atk8	] {ai_face();};
void()	enf_atk8	=[	$attack8,	enf_atk9	] {ai_face();};
void()	enf_atk9	=[	$attack5,	enf_atk10	] {ai_face();};
void()	enf_atk10	=[	$attack6,	enf_atk11	] {enforcer_fire();};
void()	enf_atk11	=[	$attack7,	enf_atk12	] {ai_face();};
void()	enf_atk12	=[	$attack8,	enf_atk13	] {ai_face();};
void()	enf_atk13	=[	$attack9,	enf_atk14	] {ai_face();};
void()	enf_atk14	=[	$attack10,	enf_run1	] {ai_face();
SUB_CheckRefire (enf_atk1);
};

void()	enf_paina1	=[	$paina1,	enf_paina2	] {};
void()	enf_paina2	=[	$paina2,	enf_paina3	] {};
void()	enf_paina3	=[	$paina3,	enf_paina4	] {};
void()	enf_paina4	=[	$paina4,	enf_run1	] {};

void()	enf_painb1	=[	$painb1,	enf_painb2	] {};
void()	enf_painb2	=[	$painb2,	enf_painb3	] {};
void()	enf_painb3	=[	$painb3,	enf_painb4	] {};
void()	enf_painb4	=[	$painb4,	enf_painb5	] {};
void()	enf_painb5	=[	$painb5,	enf_run1	] {};
```

```
void()	enf_painc1	=[	$painc1,	enf_painc2	] {};
void()	enf_painc2	=[	$painc2,	enf_painc3	] {};
void()	enf_painc3	=[	$painc3,	enf_painc4	] {};
void()	enf_painc4	=[	$painc4,	enf_painc5	] {};
void()	enf_painc5	=[	$painc5,	enf_painc6	] {};
void()	enf_painc6	=[	$painc6,	enf_painc7	] {};
void()	enf_painc7	=[	$painc7,	enf_painc8	] {};
void()	enf_painc8	=[	$painc8,	enf_run1	] {};

void()	enf_paind1	=[	$paind1,	enf_paind2	] {};
void()	enf_paind2	=[	$paind2,	enf_paind3	] {};
void()	enf_paind3	=[	$paind3,	enf_paind4	] {};
void()	enf_paind4	=[	$paind4,	enf_paind5	] {ai_painforward(2);};
void()	enf_paind5	=[	$paind5,	enf_paind6	] {ai_painforward(1);};
void()	enf_paind6	=[	$paind6,	enf_paind7	] {};
void()	enf_paind7	=[	$paind7,	enf_paind8	] {};
void()	enf_paind8	=[	$paind8,	enf_paind9	] {};
void()	enf_paind9	=[	$paind9,	enf_paind10	] {};
void()	enf_paind10	=[	$paind10,	enf_paind11	] {};
void()	enf_paind11	=[	$paind11,	enf_paind12	] {ai_painforward(1);};
void()	enf_paind12	=[	$paind12,	enf_paind13	] {ai_painforward(1);};
void()	enf_paind13	=[	$paind13,	enf_paind14	] {ai_painforward(1);};
void()	enf_paind14	=[	$paind14,	enf_paind15	] {};
void()	enf_paind15	=[	$paind15,	enf_paind16	] {};
void()	enf_paind16	=[	$paind16,	enf_paind17	] {ai_pain(1);};
void()	enf_paind17	=[	$paind17,	enf_paind18	] {ai_pain(1);};
void()	enf_paind18	=[	$paind18,	enf_paind19	] {};
void()	enf_paind19	=[	$paind19,	enf_run1	] {};

void(entity attacker, float damage)		enf_pain =
{
	local float r;

	r = random ();
	if (self.pain_finished > time)
		return;


	if (r < 0.5)
		sound (self, CHAN_VOICE, "enforcer/pain1.wav", 1, ATTN_NORM);
	else
		sound (self, CHAN_VOICE, "enforcer/pain2.wav", 1, ATTN_NORM);

	if (r < 0.2)
	{
		self.pain_finished = time + 1;
		enf_paina1 ();
	}
	else if (r < 0.4)
	{
		self.pain_finished = time + 1;
		enf_painb1 ();
	}
	else if (r < 0.7)
	{
		self.pain_finished = time + 1;
		enf_painc1 ();
	}
	else
	{
		self.pain_finished = time + 2;
```

```
		enf_paind1 ();
	}
};

//============================================================================




void()	enf_die1		=[	$death1,		enf_die2		] {};
void()	enf_die2		=[	$death2,		enf_die3		] {};
void()	enf_die3		=[	$death3,		enf_die4		]
{self.solid = SOLID_NOT;self.ammo_cells = 5;DropBackpack();};
void()	enf_die4		=[	$death4,		enf_die5		] {ai_forward(14);};
void()	enf_die5		=[	$death5,		enf_die6		] {ai_forward(2);};
void()	enf_die6		=[	$death6,		enf_die7		] {};
void()	enf_die7		=[	$death7,		enf_die8		] {};
void()	enf_die8		=[	$death8,		enf_die9		] {};
void()	enf_die9		=[	$death9,		enf_die10		] {ai_forward(3);};
void()	enf_die10		=[	$death10,		enf_die11		] {ai_forward(5);};
void()	enf_die11		=[	$death11,		enf_die12		] {ai_forward(5);};
void()	enf_die12		=[	$death12,		enf_die13		] {ai_forward(5);};
void()	enf_die13		=[	$death13,		enf_die14		] {};
void()	enf_die14		=[	$death14,		enf_die14		] {};

void()	enf_fdie1		=[	$fdeath1,		enf_fdie2		] {

};
void()	enf_fdie2		=[	$fdeath2,		enf_fdie3		] {};
void()	enf_fdie3		=[	$fdeath3,		enf_fdie4		]
{self.solid = SOLID_NOT;self.ammo_cells = 5;DropBackpack();};
void()	enf_fdie4		=[	$fdeath4,		enf_fdie5		] {};
void()	enf_fdie5		=[	$fdeath5,		enf_fdie6		] {};
void()	enf_fdie6		=[	$fdeath6,		enf_fdie7		] {};
void()	enf_fdie7		=[	$fdeath7,		enf_fdie8		] {};
void()	enf_fdie8		=[	$fdeath8,		enf_fdie9		] {};
void()	enf_fdie9		=[	$fdeath9,		enf_fdie10		] {};
void()	enf_fdie10		=[	$fdeath10,		enf_fdie11		] {};
void()	enf_fdie11		=[	$fdeath11,		enf_fdie11		] {};


void() enf_die =
{
// check for gib
	if (self.health < -35)
	{
		sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NORM);
		ThrowHead ("progs/h_mega.mdl", self.health);
		ThrowGib ("progs/gib1.mdl", self.health);
		ThrowGib ("progs/gib2.mdl", self.health);
		ThrowGib ("progs/gib3.mdl", self.health);
		return;
	}

// regular death
	sound (self, CHAN_VOICE, "enforcer/death1.wav", 1, ATTN_NORM);
	if (random() > 0.5)
		enf_die1 ();
	else
		enf_fdie1 ();
};
```

```
/*QUAKED monster_enforcer (1 0 0) (-16 -16 -24) (16 16 40) Ambush

*/
void() monster_enforcer =
{
	if (deathmatch)
	{
		remove(self);
		return;
	}
	precache_model2 ("progs/enforcer.mdl");
	precache_model2 ("progs/h_mega.mdl");
	precache_model2 ("progs/laser.mdl");

	precache_sound2 ("enforcer/death1.wav");
	precache_sound2 ("enforcer/enfire.wav");
	precache_sound2 ("enforcer/enfstop.wav");
	precache_sound2 ("enforcer/idle1.wav");
	precache_sound2 ("enforcer/pain1.wav");
	precache_sound2 ("enforcer/pain2.wav");
	precache_sound2 ("enforcer/sight1.wav");
	precache_sound2 ("enforcer/sight2.wav");
	precache_sound2 ("enforcer/sight3.wav");
	precache_sound2 ("enforcer/sight4.wav");

	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;

	setmodel (self, "progs/enforcer.mdl");

	setsize (self, '-16 -16 -24', '16 16 40');
	self.health = 80;

	self.th_stand = enf_stand1;
	self.th_walk = enf_walk1;
	self.th_run = enf_run1;
	self.th_pain = enf_pain;
	self.th_die = enf_die;
	self.th_missile = enf_atk1;

	walkmonster_start();
};
```

FISH.QC

```
$cd id1/models/fish
$origin 0 0 24
$base base
$skin skin

$frame attack1 attack2 attack3 attack4 attack5 attack6
$frame attack7 attack8 attack9 attack10 attack11 attack12 attack13
$frame attack14 attack15 attack16 attack17 attack18

$frame death1 death2 death3 death4 death5 death6 death7
$frame death8 death9 death10 death11 death12 death13 death14 death15
$frame death16 death17 death18 death19 death20 death21

$frame swim1 swim2 swim3 swim4 swim5 swim6 swim7 swim8
$frame swim9 swim10 swim11 swim12 swim13 swim14 swim15 swim16 swim17
$frame swim18

$frame pain1 pain2 pain3 pain4 pain5 pain6 pain7 pain8
$frame pain9

void() swimmonster_start;

void() f_stand1  =[      $swim1, f_stand2 ] {ai_stand();};
void() f_stand2  =[      $swim2, f_stand3 ] {ai_stand();};
void() f_stand3  =[      $swim3, f_stand4 ] {ai_stand();};
void() f_stand4  =[      $swim4, f_stand5 ] {ai_stand();};
void() f_stand5  =[      $swim5, f_stand6 ] {ai_stand();};
void() f_stand6  =[      $swim6, f_stand7 ] {ai_stand();};
void() f_stand7  =[      $swim7, f_stand8 ] {ai_stand();};
void() f_stand8  =[      $swim8, f_stand9 ] {ai_stand();};
void() f_stand9  =[      $swim9, f_stand10  ] {ai_stand();};
void() f_stand10 =[      $swim10, f_stand11 ] {ai_stand();};
void() f_stand11 =[      $swim11, f_stand12 ] {ai_stand();};
void() f_stand12 =[      $swim12, f_stand13 ] {ai_stand();};
void() f_stand13 =[      $swim13, f_stand14 ] {ai_stand();};
void() f_stand14 =[      $swim14, f_stand15 ] {ai_stand();};
void() f_stand15 =[      $swim15, f_stand16 ] {ai_stand();};
void() f_stand16 =[      $swim16, f_stand17 ] {ai_stand();};
void() f_stand17 =[      $swim17, f_stand18 ] {ai_stand();};
void() f_stand18 =[      $swim18, f_stand1 ] {ai_stand();};

void() f_walk1  =[      $swim1, f_walk2 ] {ai_walk(8);};
void() f_walk2  =[      $swim2, f_walk3 ] {ai_walk(8);};
void() f_walk3  =[      $swim3, f_walk4 ] {ai_walk(8);};
void() f_walk4  =[      $swim4, f_walk5 ] {ai_walk(8);};
void() f_walk5  =[      $swim5, f_walk6 ] {ai_walk(8);};
void() f_walk6  =[      $swim6, f_walk7 ] {ai_walk(8);};
void() f_walk7  =[      $swim7, f_walk8 ] {ai_walk(8);};
void() f_walk8  =[      $swim8, f_walk9 ] {ai_walk(8);};
void() f_walk9  =[      $swim9, f_walk10  ] {ai_walk(8);};
void() f_walk10 =[      $swim10, f_walk11 ] {ai_walk(8);};
void() f_walk11 =[      $swim11, f_walk12 ] {ai_walk(8);};
void() f_walk12 =[      $swim12, f_walk13 ] {ai_walk(8);};
void() f_walk13 =[      $swim13, f_walk14 ] {ai_walk(8);};
void() f_walk14 =[      $swim14, f_walk15 ] {ai_walk(8);};
void() f_walk15 =[      $swim15, f_walk16 ] {ai_walk(8);};
void() f_walk16 =[      $swim16, f_walk17 ] {ai_walk(8);};
void() f_walk17 =[      $swim17, f_walk18 ] {ai_walk(8);};
void() f_walk18 =[      $swim18, f_walk1 ] {ai_walk(8);};
```

```
void() f_run1  =[      $swim1, f_run2 ] {ai_run(12);
        if (random() < 0.5)
                sound (self, CHAN_VOICE, "fish/idle.wav", 1, ATTN_NORM);
};
void() f_run2  =[      $swim3, f_run3 ] {ai_run(12);};
void() f_run3  =[      $swim5, f_run4 ] {ai_run(12);};
void() f_run4  =[      $swim7, f_run5 ] {ai_run(12);};
void() f_run5  =[      $swim9, f_run6 ] {ai_run(12);};
void() f_run6  =[      $swim11, f_run7 ] {ai_run(12);};
void() f_run7  =[      $swim13, f_run8 ] {ai_run(12);};
void() f_run8  =[      $swim15, f_run9 ] {ai_run(12);};
void() f_run9  =[      $swim17, f_run1 ] {ai_run(12);};

void() fish_melee =
{
        local vector       delta;
        local float        ldmg;

        if (!self.enemy)
                return;             // removed before stroke

        delta = self.enemy.origin - self.origin;

        if (vlen(delta) > 60)
                return;

        sound (self, CHAN_VOICE, "fish/bite.wav", 1, ATTN_NORM);
        ldmg = (random() + random()) * 3;
        T_Damage (self.enemy, self, self, ldmg);
};

void() f_attack1       =[      $attack1,       f_attack2 ] {ai_charge(10);};
void() f_attack2       =[      $attack2,       f_attack3 ] {ai_charge(10);};
void() f_attack3       =[      $attack3,       f_attack4 ] {fish_melee();};
void() f_attack4       =[      $attack4,       f_attack5 ] {ai_charge(10);};
void() f_attack5       =[      $attack5,       f_attack6 ] {ai_charge(10);};
void() f_attack6       =[      $attack6,       f_attack7 ] {ai_charge(10);};
void() f_attack7       =[      $attack7,       f_attack8 ] {ai_charge(10);};
void() f_attack8       =[      $attack8,       f_attack9 ] {ai_charge(10);};
void() f_attack9       =[      $attack9,       f_attack10] {fish_melee();};
void() f_attack10      =[      $attack10,      f_attack11] {ai_charge(10);};
void() f_attack11      =[      $attack11,      f_attack12] {ai_charge(10);};
void() f_attack12      =[      $attack12,      f_attack13] {ai_charge(10);};
void() f_attack13      =[      $attack13,      f_attack14] {ai_charge(10);};
void() f_attack14      =[      $attack14,      f_attack15] {ai_charge(10);};
void() f_attack15      =[      $attack15,      f_attack16] {fish_melee();};
void() f_attack16      =[      $attack16,      f_attack17] {ai_charge(10);};
void() f_attack17      =[      $attack17,      f_attack18] {ai_charge(10);};
void() f_attack18      =[      $attack18,      f_run1    ] {ai_charge(10);};

void() f_death1 =[      $death1,        f_death2        ] {
sound (self, CHAN_VOICE, "fish/death.wav", 1, ATTN_NORM);
};
void() f_death2 =[      $death2,        f_death3        ] {};
void() f_death3 =[      $death3,        f_death4        ] {};
void() f_death4 =[      $death4,        f_death5        ] {};
void() f_death5 =[      $death5,        f_death6        ] {};
void() f_death6 =[      $death6,        f_death7        ] {};
void() f_death7 =[      $death7,        f_death8        ] {};
void() f_death8 =[      $death8,        f_death9        ] {};
void() f_death9 =[      $death9,        f_death10       ] {};
void() f_death10 =[     $death10,       f_death11       ] {};
```

```
void() f_death11 =[     $death11,       f_death12       ] {};
void() f_death12 =[     $death12,       f_death13       ] {};
void() f_death13 =[     $death13,       f_death14       ] {};
void() f_death14 =[     $death14,       f_death15       ] {};
void() f_death15 =[     $death15,       f_death16       ] {};
void() f_death16 =[     $death16,       f_death17       ] {};
void() f_death17 =[     $death17,       f_death18       ] {};
void() f_death18 =[     $death18,       f_death19       ] {};
void() f_death19 =[     $death19,       f_death20       ] {};
void() f_death20 =[     $death20,       f_death21       ] {};
void() f_death21 =[     $death21,       f_death21       ] {self.solid = SOLID_NOT;};

void() f_pain1  =[      $pain1, f_pain2 ] {};
void() f_pain2  =[      $pain2, f_pain3 ] {ai_pain(6);};
void() f_pain3  =[      $pain3, f_pain4 ] {ai_pain(6);};
void() f_pain4  =[      $pain4, f_pain5 ] {ai_pain(6);};
void() f_pain5  =[      $pain5, f_pain6 ] {ai_pain(6);};
void() f_pain6  =[      $pain6, f_pain7 ] {ai_pain(6);};
void() f_pain7  =[      $pain7, f_pain8 ] {ai_pain(6);};
void() f_pain8  =[      $pain8, f_pain9 ] {ai_pain(6);};
void() f_pain9  =[      $pain9, f_run1 ] {ai_pain(6);};

void(entity attacker, float damage)          fish_pain =
{

// fish allways do pain frames
        f_pain1 ();
};

/*QUAKED monster_fish (1 0 0) (-16 -16 -24) (16 16 24) Ambush
*/
void() monster_fish =
{
        if (deathmatch)
        {
                remove(self);
                return;
        }
        precache_model2 ("progs/fish.mdl");

        precache_sound2 ("fish/death.wav");
        precache_sound2 ("fish/bite.wav");
        precache_sound2 ("fish/idle.wav");

        self.solid = SOLID_SLIDEBOX;
        self.movetype = MOVETYPE_STEP;

        setmodel (self, "progs/fish.mdl");

        setsize (self, '-16 -16 -24', '16 16 24');
        self.health = 25;

        self.th_stand = f_stand1;
        self.th_walk = f_walk1;
        self.th_run = f_run1;
        self.th_die = f_death1;
        self.th_pain = fish_pain;
        self.th_melee = f_attack1;

        swimmonster_start ();
};
```

HKNIGHT.QC

```
/*
==============================================================================

KNIGHT

==============================================================================
*/

$cd id1/models/knight2
$origin 0 0 24
$base base
$skin skin

$frame stand1 stand2 stand3 stand4 stand5 stand6 stand7 stand8 stand9

$frame walk1 walk2 walk3 walk4 walk5 walk6 walk7 walk8 walk9
$frame walk10 walk11 walk12 walk13 walk14 walk15 walk16 walk17
$frame walk18 walk19 walk20

$frame run1 run2 run3 run4 run5 run6 run7 run8

$frame pain1 pain2 pain3 pain4 pain5

$frame death1 death2 death3 death4 death5 death6 death7 death8
$frame death9 death10 death11 death12

$frame deathb1 deathb2 deathb3 deathb4 deathb5 deathb6 deathb7 deathb8
$frame deathb9

$frame char_a1 char_a2 char_a3 char_a4 char_a5 char_a6 char_a7 char_a8
$frame char_a9 char_a10 char_a11 char_a12 char_a13 char_a14 char_a15 char_a16

$frame magica1 magica2 magica3 magica4 magica5 magica6 magica7 magica8
$frame magica9 magica10 magica11 magica12 magica13 magica14

$frame magicb1 magicb2 magicb3 magicb4 magicb5 magicb6 magicb7 magicb8
$frame magicb9 magicb10 magicb11 magicb12 magicb13

$frame char_b1 char_b2 char_b3 char_b4 char_b5 char_b6

$frame slice1 slice2 slice3 slice4 slice5 slice6 slice7 slice8 slice9 slice10

$frame smash1 smash2 smash3 smash4 smash5 smash6 smash7 smash8 smash9 smash10
$frame smash11

$frame w_attack1 w_attack2 w_attack3 w_attack4 w_attack5 w_attack6 w_attack7
$frame w_attack8 w_attack9 w_attack10 w_attack11 w_attack12 w_attack13 w_attack14
$frame w_attack15 w_attack16 w_attack17 w_attack18 w_attack19 w_attack20
$frame w_attack21 w_attack22

$frame magicc1 magicc2 magicc3 magicc4 magicc5 magicc6 magicc7 magicc8
$frame magicc9 magicc10 magicc11


void() hknight_char_a1;
void() hknight_run1;
void() hk_idle_sound;

void(float offset) hknight_shot =
{
```

```
	local	vector	offang;
	local	vector	org, vec;

	offang = vectoangles (self.enemy.origin - self.origin);
	offang_y = offang_y + offset * 6;

	makevectors (offang);

	org = self.origin + self.mins + self.size*0.5 + v_forward * 20;

// set missile speed
	vec = normalize (v_forward);
	vec_z = 0 - vec_z + (random() - 0.5)*0.1;

	launch_spike (org, vec);
	newmis.classname = "knightspike";
	setmodel (newmis, "progs/k_spike.mdl");
	setsize (newmis, VEC_ORIGIN, VEC_ORIGIN);
	newmis.velocity = vec*300;
	sound (self, CHAN_WEAPON, "hknight/attack1.wav", 1, ATTN_NORM);
};

void() CheckForCharge =
{
// check for mad charge
if (!enemy_vis)
	return;
if (time < self.attack_finished)
	return;
if ( fabs(self.origin_z - self.enemy.origin_z) > 20)
	return;		// too much height change
if ( vlen (self.origin - self.enemy.origin) < 80)
	return;		// use regular attack

// charge
	SUB_AttackFinished (2);
	hknight_char_a1 ();

};

void() CheckContinueCharge =
{
	if (time > self.attack_finished)
	{
		SUB_AttackFinished (3);
		hknight_run1 ();
		return;		// done charging
	}
	if (random() > 0.5)
		sound (self, CHAN_WEAPON, "knight/sword2.wav", 1, ATTN_NORM);
	else
		sound (self, CHAN_WEAPON, "knight/sword1.wav", 1, ATTN_NORM);
};

//=============================================================================

void()	hknight_stand1 =[	$stand1,	hknight_stand2 ] {ai_stand();};
void()	hknight_stand2 =[	$stand2,	hknight_stand3 ] {ai_stand();};
void()	hknight_stand3 =[	$stand3,	hknight_stand4 ] {ai_stand();};
void()	hknight_stand4 =[	$stand4,	hknight_stand5 ] {ai_stand();};
void()	hknight_stand5 =[	$stand5,	hknight_stand6 ] {ai_stand();};
void()	hknight_stand6 =[	$stand6,	hknight_stand7 ] {ai_stand();};
```

```
void()	hknight_stand7 =[	$stand7,	hknight_stand8 ] {ai_stand();};
void()	hknight_stand8 =[	$stand8,	hknight_stand9 ] {ai_stand();};
void()	hknight_stand9 =[	$stand9,	hknight_stand1 ] {ai_stand();};

//============================================================================

void()	hknight_walk1	=[	$walk1,		hknight_walk2	] {
hk_idle_sound();
ai_walk(2);};
void()	hknight_walk2	=[	$walk2,		hknight_walk3	] {ai_walk(5);};
void()	hknight_walk3	=[	$walk3,		hknight_walk4	] {ai_walk(5);};
void()	hknight_walk4	=[	$walk4,		hknight_walk5	] {ai_walk(4);};
void()	hknight_walk5	=[	$walk5,		hknight_walk6	] {ai_walk(4);};
void()	hknight_walk6	=[	$walk6,		hknight_walk7	] {ai_walk(2);};
void()	hknight_walk7	=[	$walk7,		hknight_walk8	] {ai_walk(2);};
void()	hknight_walk8	=[	$walk8,		hknight_walk9	] {ai_walk(3);};
void()	hknight_walk9	=[	$walk9,		hknight_walk10 ] {ai_walk(3);};
void()	hknight_walk10 =[	$walk10,	hknight_walk11 ] {ai_walk(4);};
void()	hknight_walk11 =[	$walk11,	hknight_walk12 ] {ai_walk(3);};
void()	hknight_walk12 =[	$walk12,	hknight_walk13 ] {ai_walk(4);};
void()	hknight_walk13 =[	$walk13,	hknight_walk14 ] {ai_walk(6);};
void()	hknight_walk14 =[	$walk14,	hknight_walk15 ] {ai_walk(2);};
void()	hknight_walk15 =[	$walk15,	hknight_walk16 ] {ai_walk(2);};
void()	hknight_walk16 =[	$walk16,	hknight_walk17 ] {ai_walk(4);};
void()	hknight_walk17 =[	$walk17,	hknight_walk18 ] {ai_walk(3);};
void()	hknight_walk18 =[	$walk18,	hknight_walk19 ] {ai_walk(3);};
void()	hknight_walk19 =[	$walk19,	hknight_walk20 ] {ai_walk(3);};
void()	hknight_walk20 =[	$walk20,	hknight_walk1	] {ai_walk(2);};

//============================================================================

void()	hknight_run1	=[	$run1,		hknight_run2	] {
hk_idle_sound();
ai_run (20); CheckForCharge (); };
void()	hknight_run2	=[	$run2,		hknight_run3	] {ai_run(25);};
void()	hknight_run3	=[	$run3,		hknight_run4	] {ai_run(18);};
void()	hknight_run4	=[	$run4,		hknight_run5	] {ai_run(16);};
void()	hknight_run5	=[	$run5,		hknight_run6	] {ai_run(14);};
void()	hknight_run6	=[	$run6,		hknight_run7	] {ai_run(25);};
void()	hknight_run7	=[	$run7,		hknight_run8	] {ai_run(21);};
void()	hknight_run8	=[	$run8,		hknight_run1	] {ai_run(13);};

//============================================================================

void()	hknight_pain1	=[	$pain1,		hknight_pain2	] {sound (self, CHAN_VOICE, "hknight/pain1.wav", 1,
ATTN_NORM);};
void()	hknight_pain2	=[	$pain2,		hknight_pain3	] {};
void()	hknight_pain3	=[	$pain3,		hknight_pain4	] {};
void()	hknight_pain4	=[	$pain4,		hknight_pain5	] {};
void()	hknight_pain5	=[	$pain5,		hknight_run1	] {};

//============================================================================

void()	hknight_die1	=[	$death1,	hknight_die2	] {ai_forward(10);};
void()	hknight_die2	=[	$death2,	hknight_die3	] {ai_forward(8);};
void()	hknight_die3	=[	$death3,	hknight_die4	]
{self.solid = SOLID_NOT; ai_forward(7);};
void()	hknight_die4	=[	$death4,	hknight_die5	] {};
void()	hknight_die5	=[	$death5,	hknight_die6	] {};
void()	hknight_die6	=[	$death6,	hknight_die7	] {};
void()	hknight_die7	=[	$death7,	hknight_die8	] {};
void()	hknight_die8	=[	$death8,	hknight_die9	] {ai_forward(10);};
```

```
void()	hknight_die9	=[	$death9,	hknight_die10	] {ai_forward(11);};
void()	hknight_die10	=[	$death10,	hknight_die11	] {};
void()	hknight_die11	=[	$death11,	hknight_die12	] {};
void()	hknight_die12	=[	$death12,	hknight_die12	] {};

void()	hknight_dieb1	=[	$deathb1,	hknight_dieb2	] {};
void()	hknight_dieb2	=[	$deathb2,	hknight_dieb3	] {};
void()	hknight_dieb3	=[	$deathb3,	hknight_dieb4	]
{self.solid = SOLID_NOT;};
void()	hknight_dieb4	=[	$deathb4,	hknight_dieb5	] {};
void()	hknight_dieb5	=[	$deathb5,	hknight_dieb6	] {};
void()	hknight_dieb6	=[	$deathb6,	hknight_dieb7	] {};
void()	hknight_dieb7	=[	$deathb7,	hknight_dieb8	] {};
void()	hknight_dieb8	=[	$deathb8,	hknight_dieb9	] {};
void()	hknight_dieb9	=[	$deathb9,	hknight_dieb9	] {};

void() hknight_die =
{
// check for gib
	if (self.health < -40)
	{
		sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NORM);
		ThrowHead ("progs/h_hellkn.mdl", self.health);
		ThrowGib ("progs/gib1.mdl", self.health);
		ThrowGib ("progs/gib2.mdl", self.health);
		ThrowGib ("progs/gib3.mdl", self.health);
		return;
	}

// regular death
	sound (self, CHAN_VOICE, "hknight/death1.wav", 1, ATTN_NORM);
	if (random() > 0.5)
		hknight_die1 ();
	else
		hknight_dieb1 ();
};


//============================================================================

void()	hknight_magica1 =[	$magica1,	hknight_magica2		] {ai_face();};
void()	hknight_magica2 =[	$magica2,	hknight_magica3		] {ai_face();};
void()	hknight_magica3 =[	$magica3,	hknight_magica4		] {ai_face();};
void()	hknight_magica4 =[	$magica4,	hknight_magica5		] {ai_face();};
void()	hknight_magica5 =[	$magica5,	hknight_magica6		] {ai_face();};
void()	hknight_magica6 =[	$magica6,	hknight_magica7		] {ai_face();};
void()	hknight_magica7 =[	$magica7,	hknight_magica8		] {hknight_shot(-2);};
void()	hknight_magica8 =[	$magica8,	hknight_magica9		] {hknight_shot(-1);};
void()	hknight_magica9 =[	$magica9,	hknight_magica10] {hknight_shot(0);};
void()	hknight_magica10 =[	$magica10,	hknight_magica11] {hknight_shot(1);};
void()	hknight_magica11 =[	$magica11,	hknight_magica12] {hknight_shot(2);};
void()	hknight_magica12 =[	$magica12,	hknight_magica13] {hknight_shot(3);};
void()	hknight_magica13 =[	$magica13,	hknight_magica14] {ai_face();};
void()	hknight_magica14 =[	$magica14,	hknight_run1	] {ai_face();};

//============================================================================

void()	hknight_magicb1 =[	$magicb1,	hknight_magicb2		] {ai_face();};
void()	hknight_magicb2 =[	$magicb2,	hknight_magicb3		] {ai_face();};
void()	hknight_magicb3 =[	$magicb3,	hknight_magicb4		] {ai_face();};
void()	hknight_magicb4 =[	$magicb4,	hknight_magicb5		] {ai_face();};
void()	hknight_magicb5 =[	$magicb5,	hknight_magicb6		] {ai_face();};
```

```
void()	hknight_magicb6 =[	$magicb6,	hknight_magicb7		] {ai_face();};
void()	hknight_magicb7 =[	$magicb7,	hknight_magicb8		] {hknight_shot(-2);};
void()	hknight_magicb8 =[	$magicb8,	hknight_magicb9		] {hknight_shot(-1);};
void()	hknight_magicb9 =[	$magicb9,	hknight_magicb10] {hknight_shot(0);};
void()	hknight_magicb10 =[	$magicb10,	hknight_magicb11] {hknight_shot(1);};
void()	hknight_magicb11 =[	$magicb11,	hknight_magicb12] {hknight_shot(2);};
void()	hknight_magicb12 =[	$magicb12,	hknight_magicb13] {hknight_shot(3);};
void()	hknight_magicb13 =[	$magicb13,	hknight_run1] {ai_face();};

//==========================================================================

void()	hknight_magicc1 =[	$magicc1,	hknight_magicc2		] {ai_face();};
void()	hknight_magicc2 =[	$magicc2,	hknight_magicc3		] {ai_face();};
void()	hknight_magicc3 =[	$magicc3,	hknight_magicc4		] {ai_face();};
void()	hknight_magicc4 =[	$magicc4,	hknight_magicc5		] {ai_face();};
void()	hknight_magicc5 =[	$magicc5,	hknight_magicc6		] {ai_face();};
void()	hknight_magicc6 =[	$magicc6,	hknight_magicc7		] {hknight_shot(-2);};
void()	hknight_magicc7 =[	$magicc7,	hknight_magicc8		] {hknight_shot(-1);};
void()	hknight_magicc8 =[	$magicc8,	hknight_magicc9		] {hknight_shot(0);};
void()	hknight_magicc9 =[	$magicc9,	hknight_magicc10] {hknight_shot(1);};
void()	hknight_magicc10 =[	$magicc10,	hknight_magicc11] {hknight_shot(2);};
void()	hknight_magicc11 =[	$magicc11,	hknight_run1] {hknight_shot(3);};

//==========================================================================

void()	hknight_char_a1		=[	$char_a1,	hknight_char_a2		] {ai_charge(20);};
void()	hknight_char_a2		=[	$char_a2,	hknight_char_a3		] {ai_charge(25);};
void()	hknight_char_a3		=[	$char_a3,	hknight_char_a4		] {ai_charge(18);};
void()	hknight_char_a4		=[	$char_a4,	hknight_char_a5		] {ai_charge(16);};
void()	hknight_char_a5		=[	$char_a5,	hknight_char_a6		] {ai_charge(14);};
void()	hknight_char_a6		=[	$char_a6,	hknight_char_a7		] {ai_charge(20); ai_melee();};
void()	hknight_char_a7		=[	$char_a7,	hknight_char_a8		] {ai_charge(21); ai_melee();};
void()	hknight_char_a8		=[	$char_a8,	hknight_char_a9		] {ai_charge(13); ai_melee();};
void()	hknight_char_a9		=[	$char_a9,	hknight_char_a10	] {ai_charge(20); ai_melee();};
void()	hknight_char_a10=[	$char_a10,	hknight_char_a11	] {ai_charge(20); ai_melee();};
void()	hknight_char_a11=[	$char_a11,	hknight_char_a12	] {ai_charge(18); ai_melee();};
void()	hknight_char_a12=[	$char_a12,	hknight_char_a13	] {ai_charge(16);};
void()	hknight_char_a13=[	$char_a13,	hknight_char_a14	] {ai_charge(14);};
void()	hknight_char_a14=[	$char_a14,	hknight_char_a15	] {ai_charge(25);};
void()	hknight_char_a15=[	$char_a15,	hknight_char_a16	] {ai_charge(21);};
void()	hknight_char_a16=[	$char_a16,	hknight_run1	] {ai_charge(13);};

//==========================================================================

void()	hknight_char_b1		=[	$char_b1,	hknight_char_b2		]
{CheckContinueCharge (); ai_charge(23); ai_melee();};
void()	hknight_char_b2		=[	$char_b2,	hknight_char_b3		] {ai_charge(17); ai_melee();};
void()	hknight_char_b3		=[	$char_b3,	hknight_char_b4		] {ai_charge(12); ai_melee();};
void()	hknight_char_b4		=[	$char_b4,	hknight_char_b5		] {ai_charge(22); ai_melee();};
void()	hknight_char_b5		=[	$char_b5,	hknight_char_b6		] {ai_charge(18); ai_melee();};
void()	hknight_char_b6		=[	$char_b6,	hknight_char_b1		] {ai_charge(8); ai_melee();};

//==========================================================================

void()	hknight_slice1	=[	$slice1, hknight_slice2	] {ai_charge(9);};
void()	hknight_slice2	=[	$slice2, hknight_slice3	] {ai_charge(6);};
void()	hknight_slice3	=[	$slice3, hknight_slice4	] {ai_charge(13);};
void()	hknight_slice4	=[	$slice4, hknight_slice5	] {ai_charge(4);};
void()	hknight_slice5	=[	$slice5, hknight_slice6	] {ai_charge(7); ai_melee();};
void()	hknight_slice6	=[	$slice6, hknight_slice7	] {ai_charge(15); ai_melee();};
void()	hknight_slice7	=[	$slice7, hknight_slice8	] {ai_charge(8); ai_melee();};
void()	hknight_slice8	=[	$slice8, hknight_slice9	] {ai_charge(2); ai_melee();};
```

```
void()	hknight_slice9	=[	$slice9, hknight_slice10 ] {ai_melee();};
void()	hknight_slice10 =[	$slice10,	hknight_run1	] {ai_charge(3);};

//============================================================================

void()	hknight_smash1		=[	$smash1,	hknight_smash2		] {ai_charge(1);};
void()	hknight_smash2		=[	$smash2,	hknight_smash3		] {ai_charge(13);};
void()	hknight_smash3		=[	$smash3,	hknight_smash4		] {ai_charge(9);};
void()	hknight_smash4		=[	$smash4,	hknight_smash5		] {ai_charge(11);};
void()	hknight_smash5		=[	$smash5,	hknight_smash6		] {ai_charge(10); ai_melee();};
void()	hknight_smash6		=[	$smash6,	hknight_smash7		] {ai_charge(7); ai_melee();};
void()	hknight_smash7		=[	$smash7,	hknight_smash8		] {ai_charge(12); ai_melee();};
void()	hknight_smash8		=[	$smash8,	hknight_smash9		] {ai_charge(2); ai_melee();};
void()	hknight_smash9		=[	$smash9,	hknight_smash10		] {ai_charge(3); ai_melee();};
void()	hknight_smash10		=[	$smash10,	hknight_smash11		] {ai_charge(0);};
void()	hknight_smash11		=[	$smash11,	hknight_run1	] {ai_charge(0);};

//============================================================================

void()	hknight_watk1	=[	$w_attack1,	hknight_watk2	] {ai_charge(2);};
void()	hknight_watk2	=[	$w_attack2,	hknight_watk3	] {ai_charge(0);};
void()	hknight_watk3	=[	$w_attack3,	hknight_watk4	] {ai_charge(0);};
void()	hknight_watk4	=[	$w_attack4,	hknight_watk5	] {ai_melee();};
void()	hknight_watk5	=[	$w_attack5,	hknight_watk6	] {ai_melee();};
void()	hknight_watk6	=[	$w_attack6,	hknight_watk7	] {ai_melee();};
void()	hknight_watk7	=[	$w_attack7,	hknight_watk8	] {ai_charge(1);};
void()	hknight_watk8	=[	$w_attack8,	hknight_watk9	] {ai_charge(4);};
void()	hknight_watk9	=[	$w_attack9,	hknight_watk10 ] {ai_charge(5);};
void()	hknight_watk10 =[	$w_attack10,	hknight_watk11 ] {ai_charge(3); ai_melee();};
void()	hknight_watk11 =[	$w_attack11,	hknight_watk12 ] {ai_charge(2); ai_melee();};
void()	hknight_watk12 =[	$w_attack12,	hknight_watk13 ] {ai_charge(2); ai_melee();};
void()	hknight_watk13 =[	$w_attack13,	hknight_watk14 ] {ai_charge(0);};
void()	hknight_watk14 =[	$w_attack14,	hknight_watk15 ] {ai_charge(0);};
void()	hknight_watk15 =[	$w_attack15,	hknight_watk16 ] {ai_charge(0);};
void()	hknight_watk16 =[	$w_attack16,	hknight_watk17 ] {ai_charge(1);};
void()	hknight_watk17 =[	$w_attack17,	hknight_watk18 ] {ai_charge(1); ai_melee();};
void()	hknight_watk18 =[	$w_attack18,	hknight_watk19 ] {ai_charge(3); ai_melee();};
void()	hknight_watk19 =[	$w_attack19,	hknight_watk20 ] {ai_charge(4); ai_melee();};
void()	hknight_watk20 =[	$w_attack20,	hknight_watk21 ] {ai_charge(6);};
void()	hknight_watk21 =[	$w_attack21,	hknight_watk22 ] {ai_charge(7);};
void()	hknight_watk22 =[	$w_attack22,	hknight_run1	] {ai_charge(3);};

//============================================================================

void() hk_idle_sound =
{
	if (random() < 0.2)
		sound (self, CHAN_VOICE, "hknight/idle.wav", 1, ATTN_NORM);
};

void(entity attacker, float damage)	hknight_pain =
{
	if (self.pain_finished > time)
		return;

	sound (self, CHAN_VOICE, "hknight/pain1.wav", 1, ATTN_NORM);

	if (time - self.pain_finished > 5)
	{	// allways go into pain frame if it has been a while
		hknight_pain1 ();
		self.pain_finished = time + 1;
		return;
```

```
	}

	if ((random()*30 > damage) )
		return;		// didn't flinch

	self.pain_finished = time + 1;
	hknight_pain1 ();
};

float	hknight_type;

void() hknight_melee =
{
	hknight_type = hknight_type + 1;

	sound (self, CHAN_WEAPON, "hknight/slash1.wav", 1, ATTN_NORM);
	if (hknight_type == 1)
		hknight_slice1 ();
	else if (hknight_type == 2)
		hknight_smash1 ();
	else if (hknight_type == 3)
	{
		hknight_watk1 ();
		hknight_type = 0;
	}
};

/*QUAKED monster_hell_knight (1 0 0) (-16 -16 -24) (16 16 40) Ambush
*/
void() monster_hell_knight =
{
	if (deathmatch)
	{
		remove(self);
		return;
	}
	precache_model2 ("progs/hknight.mdl");
	precache_model2 ("progs/k_spike.mdl");
	precache_model2 ("progs/h_hellkn.mdl");


	precache_sound2 ("hknight/attack1.wav");
	precache_sound2 ("hknight/death1.wav");
	precache_sound2 ("hknight/pain1.wav");
	precache_sound2 ("hknight/sight1.wav");
	precache_sound ("hknight/hit.wav");			// used by C code, so don't sound2
	precache_sound2 ("hknight/slash1.wav");
	precache_sound2 ("hknight/idle.wav");
	precache_sound2 ("hknight/grunt.wav");

	precache_sound ("knight/sword1.wav");
	precache_sound ("knight/sword2.wav");

	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;

	setmodel (self, "progs/hknight.mdl");

	setsize (self, '-16 -16 -24', '16 16 40');
	self.health = 250;

	self.th_stand = hknight_stand1;
```

```
	self.th_walk = hknight_walk1;
	self.th_run = hknight_run1;
	self.th_melee = hknight_melee;
	self.th_missile = hknight_magicc1;
	self.th_pain = hknight_pain;
	self.th_die = hknight_die;

	walkmonster_start ();
};
```

KNIGHT.QC

```
/*
==========================================================================

KNIGHT

==========================================================================
*/

$cd id1/models/knight
$origin 0 0 24
$base base
$skin badass3

$frame stand1 stand2 stand3 stand4 stand5 stand6 stand7 stand8 stand9

$frame runb1 runb2 runb3 runb4 runb5 runb6 runb7 runb8

//frame runc1 runc2 runc3 runc4 runc5 runc6

$frame runattack1 runattack2 runattack3 runattack4 runattack5
$frame runattack6 runattack7 runattack8 runattack9 runattack10
$frame runattack11

$frame pain1 pain2 pain3

$frame painb1 painb2 painb3 painb4 painb5 painb6 painb7 painb8 painb9
$frame painb10 painb11

//frame attack1 attack2 attack3 attack4 attack5 attack6 attack7
//frame attack8 attack9 attack10 attack11

$frame attackb1 attackb1 attackb2 attackb3 attackb4 attackb5
$frame attackb6 attackb7 attackb8 attackb9 attackb10

$frame walk1 walk2 walk3 walk4 walk5 walk6 walk7 walk8 walk9
$frame walk10 walk11 walk12 walk13 walk14

$frame kneel1 kneel2 kneel3 kneel4 kneel5

$frame standing2 standing3 standing4 standing5

$frame death1 death2 death3 death4 death5 death6 death7 death8
$frame death9 death10

$frame deathb1 deathb2 deathb3 deathb4 deathb5 deathb6 deathb7 deathb8
$frame deathb9 deathb10 deathb11

void()  knight_stand1   =[       $stand1,        knight_stand2   ] {ai_stand();};
void()  knight_stand2   =[       $stand2,        knight_stand3   ] {ai_stand();};
void()  knight_stand3   =[       $stand3,        knight_stand4   ] {ai_stand();};
void()  knight_stand4   =[       $stand4,        knight_stand5   ] {ai_stand();};
void()  knight_stand5   =[       $stand5,        knight_stand6   ] {ai_stand();};
void()  knight_stand6   =[       $stand6,        knight_stand7   ] {ai_stand();};
void()  knight_stand7   =[       $stand7,        knight_stand8   ] {ai_stand();};
void()  knight_stand8   =[       $stand8,        knight_stand9   ] {ai_stand();};
void()  knight_stand9   =[       $stand9,        knight_stand1   ] {ai_stand();};

void()  knight_walk1    =[       $walk1,         knight_walk2    ] {
if (random() < 0.2)
        sound (self, CHAN_VOICE, "knight/idle.wav", 1,  ATTN_IDLE);
```

```
ai_walk(3);};
void()  knight_walk2   =[       $walk2,         knight_walk3    ] {ai_walk(2);};
void()  knight_walk3   =[       $walk3,         knight_walk4    ] {ai_walk(3);};
void()  knight_walk4   =[       $walk4,         knight_walk5    ] {ai_walk(4);};
void()  knight_walk5   =[       $walk5,         knight_walk6    ] {ai_walk(3);};
void()  knight_walk6   =[       $walk6,         knight_walk7    ] {ai_walk(3);};
void()  knight_walk7   =[       $walk7,         knight_walk8    ] {ai_walk(3);};
void()  knight_walk8   =[       $walk8,         knight_walk9    ] {ai_walk(4);};
void()  knight_walk9   =[       $walk9,         knight_walk10   ] {ai_walk(3);};
void()  knight_walk10  =[       $walk10,        knight_walk11   ] {ai_walk(3);};
void()  knight_walk11  =[       $walk11,        knight_walk12   ] {ai_walk(2);};
void()  knight_walk12  =[       $walk12,        knight_walk13   ] {ai_walk(3);};
void()  knight_walk13  =[       $walk13,        knight_walk14   ] {ai_walk(4);};
void()  knight_walk14  =[       $walk14,        knight_walk1    ] {ai_walk(3);};


void()  knight_run1    =[       $runb1,         knight_run2     ] {
if (random() < 0.2)
        sound (self, CHAN_VOICE, "knight/idle.wav", 1,  ATTN_IDLE);
ai_run(16);};
void()  knight_run2    =[       $runb2,         knight_run3     ] {ai_run(20);};
void()  knight_run3    =[       $runb3,         knight_run4     ] {ai_run(13);};
void()  knight_run4    =[       $runb4,         knight_run5     ] {ai_run(7);};
void()  knight_run5    =[       $runb5,         knight_run6     ] {ai_run(16);};
void()  knight_run6    =[       $runb6,         knight_run7     ] {ai_run(20);};
void()  knight_run7    =[       $runb7,         knight_run8     ] {ai_run(14);};
void()  knight_run8    =[       $runb8,         knight_run1     ] {ai_run(6);};


void()  knight_runatk1 =[       $runattack1,            knight_runatk2  ]
{
if (random() > 0.5)
        sound (self, CHAN_WEAPON, "knight/sword2.wav", 1, ATTN_NORM);
else
        sound (self, CHAN_WEAPON, "knight/sword1.wav", 1, ATTN_NORM);
ai_charge(20);
};
void()  knight_runatk2 =[       $runattack2,    knight_runatk3  ] {ai_charge_side();};
void()  knight_runatk3 =[       $runattack3,    knight_runatk4  ] {ai_charge_side();};
void()  knight_runatk4 =[       $runattack4,    knight_runatk5  ] {ai_charge_side();};
void()  knight_runatk5 =[       $runattack5,    knight_runatk6  ] {ai_melee_side();};
void()  knight_runatk6 =[       $runattack6,    knight_runatk7  ] {ai_melee_side();};
void()  knight_runatk7 =[       $runattack7,    knight_runatk8  ] {ai_melee_side();};
void()  knight_runatk8 =[       $runattack8,    knight_runatk9  ] {ai_melee_side();};
void()  knight_runatk9 =[       $runattack9,    knight_runatk10] {ai_melee_side();};
void()  knight_runatk10=[       $runattack10,   knight_runatk11] {ai_charge_side();};
void()  knight_runatk11=[       $runattack11,   knight_run1     ] {ai_charge(10);};

void()  knight_atk1    =[       $attackb1,              knight_atk2     ]
{
sound (self, CHAN_WEAPON, "knight/sword1.wav", 1, ATTN_NORM);
ai_charge(0);};
void()  knight_atk2    =[       $attackb2,              knight_atk3     ] {ai_charge(7);};
void()  knight_atk3    =[       $attackb3,              knight_atk4     ] {ai_charge(4);};
void()  knight_atk4    =[       $attackb4,              knight_atk5     ] {ai_charge(0);};
void()  knight_atk5    =[       $attackb5,              knight_atk6     ] {ai_charge(3);};
void()  knight_atk6    =[       $attackb6,              knight_atk7     ] {ai_charge(4); ai_melee();};
void()  knight_atk7    =[       $attackb7,              knight_atk8     ] {ai_charge(1); ai_melee();};
void()  knight_atk8    =[       $attackb8,              knight_atk9     ] {ai_charge(3);
ai_melee();};
void()  knight_atk9    =[       $attackb9,              knight_atk10] {ai_charge(1);};
void()  knight_atk10=[  $attackb10,             knight_run1     ] {ai_charge(5);};
```

```
//void()  knight_atk9     =[      $attack9,               knight_atk10    ] {};
//void()  knight_atk10    =[      $attack10,              knight_atk11    ] {};
//void()  knight_atk11    =[      $attack11,              knight_run1     ] {};

//============================================================================

void()  knight_pain1    =[      $pain1,         knight_pain2    ] {};
void()  knight_pain2    =[      $pain2,         knight_pain3    ] {};
void()  knight_pain3    =[      $pain3,         knight_run1     ] {};

void()  knight_painb1   =[      $painb1,        knight_painb2   ] {ai_painforward(0);};
void()  knight_painb2   =[      $painb2,        knight_painb3   ] {ai_painforward(3);};
void()  knight_painb3   =[      $painb3,        knight_painb4   ] {};
void()  knight_painb4   =[      $painb4,        knight_painb5   ] {};
void()  knight_painb5   =[      $painb5,        knight_painb6   ] {ai_painforward(2);};
void()  knight_painb6   =[      $painb6,        knight_painb7   ] {ai_painforward(4);};
void()  knight_painb7   =[      $painb7,        knight_painb8   ] {ai_painforward(2);};
void()  knight_painb8   =[      $painb8,        knight_painb9   ] {ai_painforward(5);};
void()  knight_painb9   =[      $painb9,        knight_painb10  ] {ai_painforward(5);};
void()  knight_painb10  =[      $painb10,       knight_painb11  ] {ai_painforward(0);};
void()  knight_painb11  =[      $painb11,       knight_run1     ] {};

void(entity attacker, float damage)     knight_pain =
{
        local float r;

        if (self.pain_finished > time)
                return;

        r = random();

        sound (self, CHAN_VOICE, "knight/khurt.wav", 1, ATTN_NORM);
        if (r < 0.85)
        {
                knight_pain1 ();
                self.pain_finished = time + 1;
        }
        else
        {
                knight_painb1 ();
                self.pain_finished = time + 1;
        }

};

//============================================================================

void()  knight_bow1     =[      $kneel1,                knight_bow2     ] {ai_turn();};
void()  knight_bow2     =[      $kneel2,                knight_bow3     ] {ai_turn();};
void()  knight_bow3     =[      $kneel3,                knight_bow4     ] {ai_turn();};
void()  knight_bow4     =[      $kneel4,                knight_bow5     ] {ai_turn();};

void()  knight_bow5     =[      $kneel5,                knight_bow5     ] {ai_turn();};

void()  knight_bow6     =[      $kneel4,                knight_bow7     ] {ai_turn();};
void()  knight_bow7     =[      $kneel3,                knight_bow8     ] {ai_turn();};
void()  knight_bow8     =[      $kneel2,                knight_bow9     ] {ai_turn();};
void()  knight_bow9     =[      $kneel1,                knight_bow10    ] {ai_turn();};
void()  knight_bow10    =[      $walk1,         knight_walk1    ] {ai_turn();};
```

```
void()	knight_die1	=[	$death1,		knight_die2	] {};
void()	knight_die2	=[	$death2,		knight_die3	] {};
void()	knight_die3	=[	$death3,		knight_die4	]
{self.solid = SOLID_NOT;};
void()	knight_die4	=[	$death4,		knight_die5	] {};
void()	knight_die5	=[	$death5,		knight_die6	] {};
void()	knight_die6	=[	$death6,		knight_die7	] {};
void()	knight_die7	=[	$death7,		knight_die8	] {};
void()	knight_die8	=[	$death8,		knight_die9	] {};
void()	knight_die9	=[	$death9,		knight_die10] {};
void()	knight_die10=[	$death10,		knight_die10] {};


void()	knight_dieb1	=[	$deathb1,		knight_dieb2	] {};
void()	knight_dieb2	=[	$deathb2,		knight_dieb3	] {};
void()	knight_dieb3	=[	$deathb3,		knight_dieb4	]
{self.solid = SOLID_NOT;};
void()	knight_dieb4	=[	$deathb4,		knight_dieb5	] {};
void()	knight_dieb5	=[	$deathb5,		knight_dieb6	] {};
void()	knight_dieb6	=[	$deathb6,		knight_dieb7	] {};
void()	knight_dieb7	=[	$deathb7,		knight_dieb8	] {};
void()	knight_dieb8	=[	$deathb8,		knight_dieb9	] {};
void()	knight_dieb9	=[	$deathb9,		knight_dieb10] {};
void()	knight_dieb10 = [	$deathb10,		knight_dieb11] {};
void()	knight_dieb11 = [	$deathb11,		knight_dieb11] {};


void() knight_die =
{
// check for gib
	if (self.health < -40)
	{
		sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NORM);
		ThrowHead ("progs/h_knight.mdl", self.health);
		ThrowGib ("progs/gib1.mdl", self.health);
		ThrowGib ("progs/gib2.mdl", self.health);
		ThrowGib ("progs/gib3.mdl", self.health);
		return;
	}

// regular death
	sound (self, CHAN_VOICE, "knight/kdeath.wav", 1, ATTN_NORM);
	if (random() < 0.5)
		knight_die1 ();
	else
		knight_dieb1 ();
};


/*QUAKED monster_knight (1 0 0) (-16 -16 -24) (16 16 40) Ambush
*/
void() monster_knight =
{
	if (deathmatch)
	{
		remove(self);
		return;
	}
	precache_model ("progs/knight.mdl");
	precache_model ("progs/h_knight.mdl");
```

```
	precache_sound ("knight/kdeath.wav");
	precache_sound ("knight/khurt.wav");
	precache_sound ("knight/ksight.wav");
	precache_sound ("knight/sword1.wav");
	precache_sound ("knight/sword2.wav");
	precache_sound ("knight/idle.wav");

	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;

	setmodel (self, "progs/knight.mdl");

	setsize (self, '-16 -16 -24', '16 16 40');
	self.health = 75;

	self.th_stand = knight_stand1;
	self.th_walk = knight_walk1;
	self.th_run = knight_run1;
	self.th_melee = knight_atk1;
	self.th_pain = knight_pain;
	self.th_die = knight_die;

	walkmonster_start ();
};
```

OGRE.QC

```
/*
==========================================================================

OGRE

==========================================================================
*/

$cd id1/models/ogre_c
$origin 0 0 24
$base base
$skin base

$frame  stand1 stand2 stand3 stand4 stand5 stand6 stand7 stand8 stand9

$frame walk1 walk2 walk3 walk4 walk5 walk6 walk7
$frame walk8 walk9 walk10 walk11 walk12 walk13 walk14 walk15 walk16

$frame run1 run2 run3 run4 run5 run6 run7 run8

$frame swing1 swing2 swing3 swing4 swing5 swing6 swing7
$frame swing8 swing9 swing10 swing11 swing12 swing13 swing14

$frame smash1 smash2 smash3 smash4 smash5 smash6 smash7
$frame smash8 smash9 smash10 smash11 smash12 smash13 smash14

$frame shoot1 shoot2 shoot3 shoot4 shoot5 shoot6

$frame pain1 pain2 pain3 pain4 pain5

$frame painb1 painb2 painb3

$frame painc1 painc2 painc3 painc4 painc5 painc6

$frame paind1 paind2 paind3 paind4 paind5 paind6 paind7 paind8 paind9 paind10
$frame paind11 paind12 paind13 paind14 paind15 paind16

$frame paine1 paine2 paine3 paine4 paine5 paine6 paine7 paine8 paine9 paine10
$frame paine11 paine12 paine13 paine14 paine15

$frame death1 death2 death3 death4 death5 death6
$frame death7 death8 death9 death10 death11 death12
$frame death13 death14

$frame bdeath1 bdeath2 bdeath3 bdeath4 bdeath5 bdeath6
$frame bdeath7 bdeath8 bdeath9 bdeath10

$frame pull1 pull2 pull3 pull4 pull5 pull6 pull7 pull8 pull9 pull10 pull11

//==========================================================================


void() OgreGrenadeExplode =
{
        T_RadiusDamage (self, self.owner, 40, world);
        sound (self, CHAN_VOICE, "weapons/r_exp3.wav", 1, ATTN_NORM);

        WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
        WriteByte (MSG_BROADCAST, TE_EXPLOSION);
        WriteCoord (MSG_BROADCAST, self.origin_x);
```

```
	WriteCoord (MSG_BROADCAST, self.origin_y);
	WriteCoord (MSG_BROADCAST, self.origin_z);

	self.velocity = '0 0 0';
	self.touch = SUB_Null;
	setmodel (self, "progs/s_explod.spr");
	self.solid = SOLID_NOT;
	s_explode1 ();
};

void() OgreGrenadeTouch =
{
	if (other == self.owner)
		return;		// don't explode on owner
	if (other.takedamage == DAMAGE_AIM)
	{
		OgreGrenadeExplode();
		return;
	}
	sound (self, CHAN_VOICE, "weapons/bounce.wav", 1, ATTN_NORM);	// bounce sound
	if (self.velocity == '0 0 0')
		self.avelocity = '0 0 0';
};

/*
================
OgreFireGrenade
================
*/
void() OgreFireGrenade =
{
	local	entity missile, mpuff;

	self.effects = self.effects | EF_MUZZLEFLASH;

	sound (self, CHAN_WEAPON, "weapons/grenade.wav", 1, ATTN_NORM);

	missile = spawn ();
	missile.owner = self;
	missile.movetype = MOVETYPE_BOUNCE;
	missile.solid = SOLID_BBOX;

// set missile speed

	makevectors (self.angles);

	missile.velocity = normalize(self.enemy.origin - self.origin);
	missile.velocity = missile.velocity * 600;
	missile.velocity_z = 200;

	missile.avelocity = '300 300 300';

	missile.angles = vectoangles(missile.velocity);

	missile.touch = OgreGrenadeTouch;

// set missile duration
	missile.nextthink = time + 2.5;
	missile.think = OgreGrenadeExplode;

	setmodel (missile, "progs/grenade.mdl");
	setsize (missile, '0 0 0', '0 0 0');
```

```
	setorigin (missile, self.origin);
};


//============================================================================

/*
================
chainsaw

FIXME
================
*/
void(float side) chainsaw =
{
local vector	delta;
local float	ldmg;

	if (!self.enemy)
		return;
	if (!CanDamage (self.enemy, self))
		return;

	ai_charge(10);

	delta = self.enemy.origin - self.origin;

	if (vlen(delta) > 100)
		return;

	ldmg = (random() + random() + random()) * 4;
	T_Damage (self.enemy, self, self, ldmg);

	if (side)
	{
		makevectors (self.angles);
		if (side == 1)
			SpawnMeatSpray (self.origin + v_forward*16, crandom() * 100 * v_right);
		else
			SpawnMeatSpray (self.origin + v_forward*16, side * v_right);
	}
};


void() ogre_stand1	=[	$stand1,	ogre_stand2	] {ai_stand();};
void() ogre_stand2	=[	$stand2,	ogre_stand3	] {ai_stand();};
void() ogre_stand3	=[	$stand3,	ogre_stand4	] {ai_stand();};
void() ogre_stand4	=[	$stand4,	ogre_stand5	] {ai_stand();};
void() ogre_stand5	=[	$stand5,	ogre_stand6	] {
if (random() < 0.2)
	sound (self, CHAN_VOICE, "ogre/ogidle.wav", 1, ATTN_IDLE);
ai_stand();
};
void() ogre_stand6	=[	$stand6,	ogre_stand7	] {ai_stand();};
void() ogre_stand7	=[	$stand7,	ogre_stand8	] {ai_stand();};
void() ogre_stand8	=[	$stand8,	ogre_stand9	] {ai_stand();};
void() ogre_stand9	=[	$stand9,	ogre_stand1	] {ai_stand();};

void() ogre_walk1	=[	$walk1,		ogre_walk2	] {ai_walk(3);};
void() ogre_walk2	=[	$walk2,		ogre_walk3	] {ai_walk(2);};
void() ogre_walk3	=[	$walk3,		ogre_walk4	] {
ai_walk(2);
```

```
if (random() < 0.2)
        sound (self, CHAN_VOICE, "ogre/ogidle.wav", 1, ATTN_IDLE);
};
void() ogre_walk4       =[      $walk4,         ogre_walk5      ] {ai_walk(2);};
void() ogre_walk5       =[      $walk5,         ogre_walk6      ] {ai_walk(2);};
void() ogre_walk6       =[      $walk6,         ogre_walk7      ] {
ai_walk(5);
if (random() < 0.1)
        sound (self, CHAN_VOICE, "ogre/ogdrag.wav", 1, ATTN_IDLE);
};
void() ogre_walk7       =[      $walk7,         ogre_walk8      ] {ai_walk(3);};
void() ogre_walk8       =[      $walk8,         ogre_walk9      ] {ai_walk(2);};
void() ogre_walk9       =[      $walk9,         ogre_walk10     ] {ai_walk(3);};
void() ogre_walk10      =[      $walk10,        ogre_walk11     ] {ai_walk(1);};
void() ogre_walk11      =[      $walk11,        ogre_walk12     ] {ai_walk(2);};
void() ogre_walk12      =[      $walk12,        ogre_walk13     ] {ai_walk(3);};
void() ogre_walk13      =[      $walk13,        ogre_walk14     ] {ai_walk(3);};
void() ogre_walk14      =[      $walk14,        ogre_walk15     ] {ai_walk(3);};
void() ogre_walk15      =[      $walk15,        ogre_walk16     ] {ai_walk(3);};
void() ogre_walk16      =[      $walk16,        ogre_walk1      ] {ai_walk(4);};

void() ogre_run1        =[      $run1,          ogre_run2       ] {ai_run(9);
if (random() < 0.2)
        sound (self, CHAN_VOICE, "ogre/ogidle2.wav", 1, ATTN_IDLE);
};
void() ogre_run2        =[      $run2,          ogre_run3       ] {ai_run(12);};
void() ogre_run3        =[      $run3,          ogre_run4       ] {ai_run(8);};
void() ogre_run4        =[      $run4,          ogre_run5       ] {ai_run(22);};
void() ogre_run5        =[      $run5,          ogre_run6       ] {ai_run(16);};
void() ogre_run6        =[      $run6,          ogre_run7       ] {ai_run(4);};
void() ogre_run7        =[      $run7,          ogre_run8       ] {ai_run(13);};
void() ogre_run8        =[      $run8,          ogre_run1       ] {ai_run(24);};

void() ogre_swing1      =[      $swing1,                ogre_swing2     ] {ai_charge(11);
sound (self, CHAN_WEAPON, "ogre/ogsawatk.wav", 1, ATTN_NORM);
};
void() ogre_swing2      =[      $swing2,                ogre_swing3     ] {ai_charge(1);};
void() ogre_swing3      =[      $swing3,                ogre_swing4     ] {ai_charge(4);};
void() ogre_swing4      =[      $swing4,                ogre_swing5     ] {ai_charge(13);};
void() ogre_swing5      =[      $swing5,                ogre_swing6     ] {ai_charge(9); chainsaw(0);self.angles_y =
self.angles_y + random()*25;};
void() ogre_swing6      =[      $swing6,                ogre_swing7     ] {chainsaw(200);self.angles_y = self.angles_y +
random()* 25;};
void() ogre_swing7      =[      $swing7,                ogre_swing8     ] {chainsaw(0);self.angles_y = self.angles_y +
random()* 25;};
void() ogre_swing8      =[      $swing8,                ogre_swing9     ] {chainsaw(0);self.angles_y = self.angles_y +
random()* 25;};
void() ogre_swing9      =[      $swing9,                ogre_swing10 ] {chainsaw(0);self.angles_y = self.angles_y +
random()* 25;};
void() ogre_swing10     =[      $swing10,               ogre_swing11 ] {chainsaw(-200);self.angles_y = self.angles_y +
random()* 25;};
void() ogre_swing11     =[      $swing11,               ogre_swing12 ] {chainsaw(0);self.angles_y = self.angles_y +
random()* 25;};
void() ogre_swing12     =[      $swing12,               ogre_swing13 ] {ai_charge(3);};
void() ogre_swing13     =[      $swing13,               ogre_swing14 ] {ai_charge(8);};
void() ogre_swing14     =[      $swing14,               ogre_run1       ] {ai_charge(9);};

void() ogre_smash1      =[      $smash1,                ogre_smash2     ] {ai_charge(6);
sound (self, CHAN_WEAPON, "ogre/ogsawatk.wav", 1, ATTN_NORM);
};
void() ogre_smash2      =[      $smash2,                ogre_smash3     ] {ai_charge(0);};
void() ogre_smash3      =[      $smash3,                ogre_smash4     ] {ai_charge(0);};
```

```
void() ogre_smash4      =[      $smash4,                ogre_smash5     ] {ai_charge(1);};
void() ogre_smash5      =[      $smash5,                ogre_smash6     ] {ai_charge(4);};
void() ogre_smash6      =[      $smash6,                ogre_smash7     ] {ai_charge(4); chainsaw(0);};
void() ogre_smash7      =[      $smash7,                ogre_smash8     ] {ai_charge(4); chainsaw(0);};
void() ogre_smash8      =[      $smash8,                ogre_smash9     ] {ai_charge(10); chainsaw(0);};
void() ogre_smash9      =[      $smash9,                ogre_smash10 ] {ai_charge(13); chainsaw(0);};
void() ogre_smash10     =[      $smash10,               ogre_smash11 ] {chainsaw(1);};
void() ogre_smash11     =[      $smash11,               ogre_smash12 ] {ai_charge(2); chainsaw(0);
self.nextthink = self.nextthink + random()*0.2;};   // slight variation
void() ogre_smash12     =[      $smash12,               ogre_smash13 ] {ai_charge();};
void() ogre_smash13     =[      $smash13,               ogre_smash14 ] {ai_charge(4);};
void() ogre_smash14     =[      $smash14,               ogre_run1       ] {ai_charge(12);};

void() ogre_nail1       =[      $shoot1,                ogre_nail2      ] {ai_face();};
void() ogre_nail2       =[      $shoot2,                ogre_nail3      ] {ai_face();};
void() ogre_nail3       =[      $shoot2,                ogre_nail4      ] {ai_face();};
void() ogre_nail4       =[      $shoot3,                ogre_nail5      ] {ai_face();OgreFireGrenade();};
void() ogre_nail5       =[      $shoot4,                ogre_nail6      ] {ai_face();};
void() ogre_nail6       =[      $shoot5,                ogre_nail7      ] {ai_face();};
void() ogre_nail7       =[      $shoot6,                ogre_run1       ] {ai_face();};

void()  ogre_pain1      =[      $pain1,         ogre_pain2      ] {};
void()  ogre_pain2      =[      $pain2,         ogre_pain3      ] {};
void()  ogre_pain3      =[      $pain3,         ogre_pain4      ] {};
void()  ogre_pain4      =[      $pain4,         ogre_pain5      ] {};
void()  ogre_pain5      =[      $pain5,         ogre_run1       ] {};


void()  ogre_painb1     =[      $painb1,        ogre_painb2     ] {};
void()  ogre_painb2     =[      $painb2,        ogre_painb3     ] {};
void()  ogre_painb3     =[      $painb3,        ogre_run1       ] {};


void()  ogre_painc1     =[      $painc1,        ogre_painc2     ] {};
void()  ogre_painc2     =[      $painc2,        ogre_painc3     ] {};
void()  ogre_painc3     =[      $painc3,        ogre_painc4     ] {};
void()  ogre_painc4     =[      $painc4,        ogre_painc5     ] {};
void()  ogre_painc5     =[      $painc5,        ogre_painc6     ] {};
void()  ogre_painc6     =[      $painc6,        ogre_run1       ] {};


void()  ogre_paind1     =[      $paind1,        ogre_paind2     ] {};
void()  ogre_paind2     =[      $paind2,        ogre_paind3     ] {ai_pain(10);};
void()  ogre_paind3     =[      $paind3,        ogre_paind4     ] {ai_pain(9);};
void()  ogre_paind4     =[      $paind4,        ogre_paind5     ] {ai_pain(4);};
void()  ogre_paind5     =[      $paind5,        ogre_paind6     ] {};
void()  ogre_paind6     =[      $paind6,        ogre_paind7     ] {};
void()  ogre_paind7     =[      $paind7,        ogre_paind8     ] {};
void()  ogre_paind8     =[      $paind8,        ogre_paind9     ] {};
void()  ogre_paind9     =[      $paind9,        ogre_paind10    ] {};
void()  ogre_paind10=[ $paind10,        ogre_paind11    ] {};
void()  ogre_paind11=[ $paind11,        ogre_paind12    ] {};
void()  ogre_paind12=[ $paind12,        ogre_paind13    ] {};
void()  ogre_paind13=[ $paind13,        ogre_paind14    ] {};
void()  ogre_paind14=[ $paind14,        ogre_paind15    ] {};
void()  ogre_paind15=[ $paind15,        ogre_paind16    ] {};
void()  ogre_paind16=[ $paind16,        ogre_run1       ] {};

void()  ogre_paine1     =[      $paine1,        ogre_paine2     ] {};
void()  ogre_paine2     =[      $paine2,        ogre_paine3     ] {ai_pain(10);};
void()  ogre_paine3     =[      $paine3,        ogre_paine4     ] {ai_pain(9);};
void()  ogre_paine4     =[      $paine4,        ogre_paine5     ] {ai_pain(4);};
```

```
void()	ogre_paine5	=[	$paine5,	ogre_paine6	] {};
void()	ogre_paine6	=[	$paine6,	ogre_paine7	] {};
void()	ogre_paine7	=[	$paine7,	ogre_paine8	] {};
void()	ogre_paine8	=[	$paine8,	ogre_paine9	] {};
void()	ogre_paine9	=[	$paine9,	ogre_paine10	] {};
void()	ogre_paine10=[ $paine10,	ogre_paine11	] {};
void()	ogre_paine11=[ $paine11,	ogre_paine12	] {};
void()	ogre_paine12=[ $paine12,	ogre_paine13	] {};
void()	ogre_paine13=[ $paine13,	ogre_paine14	] {};
void()	ogre_paine14=[ $paine14,	ogre_paine15	] {};
void()	ogre_paine15=[ $paine15,	ogre_run1	] {};


void(entity attacker, float damage)	ogre_pain =
{
	local float	r;

// don't make multiple pain sounds right after each other
	if (self.pain_finished > time)
		return;

	sound (self, CHAN_VOICE, "ogre/ogpain1.wav", 1, ATTN_NORM);

	r = random();

	if (r < 0.25)
	{
		ogre_pain1 ();
		self.pain_finished = time + 1;
	}
	else if (r < 0.5)
	{
		ogre_painb1 ();
		self.pain_finished = time + 1;
	}
	else if (r < 0.75)
	{
		ogre_painc1 ();
		self.pain_finished = time + 1;
	}
	else if (r < 0.88)
	{
		ogre_paind1 ();
		self.pain_finished = time + 2;
	}
	else
	{
		ogre_paine1 ();
		self.pain_finished = time + 2;
	}
};

void()	ogre_die1	=[	$death1,	ogre_die2	] {};
void()	ogre_die2	=[	$death2,	ogre_die3	] {};
void()	ogre_die3	=[	$death3,	ogre_die4	]
{self.solid = SOLID_NOT;
self.ammo_rockets = 2;DropBackpack();};
void()	ogre_die4	=[	$death4,	ogre_die5	] {};
void()	ogre_die5	=[	$death5,	ogre_die6	] {};
void()	ogre_die6	=[	$death6,	ogre_die7	] {};
void()	ogre_die7	=[	$death7,	ogre_die8	] {};
void()	ogre_die8	=[	$death8,	ogre_die9	] {};
```

```
void()	ogre_die9	=[	$death9,	ogre_die10	] {};
void()	ogre_die10	=[	$death10,	ogre_die11	] {};
void()	ogre_die11	=[	$death11,	ogre_die12	] {};
void()	ogre_die12	=[	$death12,	ogre_die13	] {};
void()	ogre_die13	=[	$death13,	ogre_die14	] {};
void()	ogre_die14	=[	$death14,	ogre_die14	] {};

void()	ogre_bdie1	=[	$bdeath1,	ogre_bdie2	] {};
void()	ogre_bdie2	=[	$bdeath2,	ogre_bdie3	] {ai_forward(5);};
void()	ogre_bdie3	=[	$bdeath3,	ogre_bdie4	]
{self.solid = SOLID_NOT;
self.ammo_rockets = 2;DropBackpack();};
void()	ogre_bdie4	=[	$bdeath4,	ogre_bdie5	] {ai_forward(1);};
void()	ogre_bdie5	=[	$bdeath5,	ogre_bdie6	] {ai_forward(3);};
void()	ogre_bdie6	=[	$bdeath6,	ogre_bdie7	] {ai_forward(7);};
void()	ogre_bdie7	=[	$bdeath7,	ogre_bdie8	] {ai_forward(25);};
void()	ogre_bdie8	=[	$bdeath8,	ogre_bdie9	] {};
void()	ogre_bdie9	=[	$bdeath9,	ogre_bdie10	] {};
void()	ogre_bdie10	=[	$bdeath10,	ogre_bdie10	] {};

void() ogre_die =
{
// check for gib
	if (self.health < -80)
	{
		sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NORM);
		ThrowHead ("progs/h_ogre.mdl", self.health);
		ThrowGib ("progs/gib3.mdl", self.health);
		ThrowGib ("progs/gib3.mdl", self.health);
		ThrowGib ("progs/gib3.mdl", self.health);
		return;
	}

	sound (self, CHAN_VOICE, "ogre/ogdth.wav", 1, ATTN_NORM);

	if (random() < 0.5)
		ogre_die1 ();
	else
		ogre_bdie1 ();
};

void() ogre_melee =
{
	if (random() > 0.5)
		ogre_smash1 ();
	else
		ogre_swing1 ();
};


/*QUAKED monster_ogre (1 0 0) (-32 -32 -24) (32 32 64) Ambush

*/
void() monster_ogre =
{
	if (deathmatch)
	{
		remove(self);
		return;
	}
	precache_model ("progs/ogre.mdl");
	precache_model ("progs/h_ogre.mdl");
```

```
	precache_model ("progs/grenade.mdl");

	precache_sound ("ogre/ogdrag.wav");
	precache_sound ("ogre/ogdth.wav");
	precache_sound ("ogre/ogidle.wav");
	precache_sound ("ogre/ogidle2.wav");
	precache_sound ("ogre/ogpain1.wav");
	precache_sound ("ogre/ogsawatk.wav");
	precache_sound ("ogre/ogwake.wav");

	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;

	setmodel (self, "progs/ogre.mdl");

	setsize (self, VEC_HULL2_MIN, VEC_HULL2_MAX);
	self.health = 200;

	self.th_stand = ogre_stand1;
	self.th_walk = ogre_walk1;
	self.th_run = ogre_run1;
	self.th_die = ogre_die;
	self.th_melee = ogre_melee;
	self.th_missile = ogre_nail1;
	self.th_pain = ogre_pain;

	walkmonster_start();
};

void() monster_ogre_marksman =
{
	monster_ogre ();
};
```

OLDONE.QC (SHUB NIGGURATH)

```
/*
==============================================================================

OLD ONE

==============================================================================
*/
$cd id1/models/old_one
$origin 0 0 24
$base base
$skin skin
$scale 1

void() finale_1;
void() finale_2;
void() finale_3;
void() finale_4;


entity     shub;

$frame old1 old2 old3 old4 old5 old6 old7 old8 old9
$frame old10 old11 old12 old13 old14 old15 old16 old17 old18 old19
$frame old20 old21 old22 old23 old24 old25 old26 old27 old28 old29
$frame old30 old31 old32 old33 old34 old35 old36 old37 old38 old39
$frame old40 old41 old42 old43 old44 old45 old46

$frame shake1 shake2 shake3 shake4 shake5 shake6 shake7 shake8
$frame shake9 shake10 shake11 shake12 shake12 shake13 shake14
$frame shake15 shake16 shake17 shake18 shake19 shake20

//void() old_stand     =[      $old1,       old_stand    ] {};

void() old_idle1        =[      $old1,  old_idle2       ] {};
void() old_idle2        =[      $old2,  old_idle3       ] {};
void() old_idle3        =[      $old3,  old_idle4       ] {};
void() old_idle4        =[      $old4,  old_idle5       ] {};
void() old_idle5        =[      $old5,  old_idle6       ] {};
void() old_idle6        =[      $old6,  old_idle7       ] {};
void() old_idle7        =[      $old7,  old_idle8       ] {};
void() old_idle8        =[      $old8,  old_idle9       ] {};
void() old_idle9        =[      $old9,  old_idle10      ] {};
void() old_idle10       =[      $old10, old_idle11      ] {};
void() old_idle11       =[      $old11, old_idle12      ] {};
void() old_idle12       =[      $old12, old_idle13      ] {};
void() old_idle13       =[      $old13, old_idle14      ] {};
void() old_idle14       =[      $old14, old_idle15      ] {};
void() old_idle15       =[      $old15, old_idle16      ] {};
void() old_idle16       =[      $old16, old_idle17      ] {};
void() old_idle17       =[      $old17, old_idle18      ] {};
void() old_idle18       =[      $old18, old_idle19      ] {};
void() old_idle19       =[      $old19, old_idle20      ] {};
void() old_idle20       =[      $old20, old_idle21      ] {};
void() old_idle21       =[      $old21, old_idle22      ] {};
void() old_idle22       =[      $old22, old_idle23      ] {};
void() old_idle23       =[      $old23, old_idle24      ] {};
void() old_idle24       =[      $old24, old_idle25      ] {};
void() old_idle25       =[      $old25, old_idle26      ] {};
void() old_idle26       =[      $old26, old_idle27      ] {};
void() old_idle27       =[      $old27, old_idle28      ] {};
```

```
void() old_idle28	=[	$old28, old_idle29	] {};
void() old_idle29	=[	$old29, old_idle30	] {};
void() old_idle30	=[	$old30, old_idle31	] {};
void() old_idle31	=[	$old31, old_idle32	] {};
void() old_idle32	=[	$old32, old_idle33	] {};
void() old_idle33	=[	$old33, old_idle34	] {};
void() old_idle34	=[	$old34, old_idle35	] {};
void() old_idle35	=[	$old35, old_idle36	] {};
void() old_idle36	=[	$old36, old_idle37	] {};
void() old_idle37	=[	$old37, old_idle38	] {};
void() old_idle38	=[	$old38, old_idle39	] {};
void() old_idle39	=[	$old39, old_idle40	] {};
void() old_idle40	=[	$old40, old_idle41	] {};
void() old_idle41	=[	$old41, old_idle42	] {};
void() old_idle42	=[	$old42, old_idle43	] {};
void() old_idle43	=[	$old43, old_idle44	] {};
void() old_idle44	=[	$old44, old_idle45	] {};
void() old_idle45	=[	$old45, old_idle46	] {};
void() old_idle46	=[	$old46, old_idle1	] {};


void() old_thrash1	=[ $shake1,  old_thrash2	] {lightstyle(0, "m");};
void() old_thrash2	=[	$shake2,  old_thrash3	] {lightstyle(0, "k");};
void() old_thrash3	=[	$shake3,  old_thrash4	] {lightstyle(0, "k");};
void() old_thrash4	=[	$shake4,  old_thrash5	] {lightstyle(0, "i");};
void() old_thrash5	=[	$shake5,  old_thrash6	] {lightstyle(0, "g");};
void() old_thrash6	=[	$shake6,  old_thrash7	] {lightstyle(0, "e");};
void() old_thrash7	=[	$shake7,  old_thrash8	] {lightstyle(0, "c");};
void() old_thrash8	=[	$shake8,  old_thrash9	] {lightstyle(0, "a");};
void() old_thrash9	=[	$shake9,  old_thrash10	] {lightstyle(0, "c");};
void() old_thrash10	=[	$shake10, old_thrash11	] {lightstyle(0, "e");};
void() old_thrash11	=[	$shake11, old_thrash12	] {lightstyle(0, "g");};
void() old_thrash12	=[	$shake12, old_thrash13	] {lightstyle(0, "i");};
void() old_thrash13	=[	$shake13, old_thrash14	] {lightstyle(0, "k");};
void() old_thrash14	=[	$shake14, old_thrash15	] {lightstyle(0, "m");};
void() old_thrash15	=[	$shake15, old_thrash16	] {lightstyle(0, "m");
self.cnt = self.cnt + 1;
if (self.cnt != 3)
	self.think = old_thrash1;
};
void() old_thrash16	=[	$shake16, old_thrash17	] {lightstyle(0, "g");};
void() old_thrash17	=[	$shake17, old_thrash18	] {lightstyle(0, "c");};
void() old_thrash18	=[	$shake18, old_thrash19	] {lightstyle(0, "b");};
void() old_thrash19	=[	$shake19, old_thrash20	] {lightstyle(0, "a");};
void() old_thrash20	=[	$shake20, old_thrash20	] {finale_4();};

//==========================================================================

void() finale_1 =
{
	local entity	pos, pl;
	local entity	timer;

	intermission_exittime = time + 10000000;	// never allow exit
	intermission_running = 1;

	// find the intermission spot
	pos = find (world, classname, "info_intermission");
	if (!pos)
		error ("no info_intermission");
	pl = find (world, classname, "misc_teleporttrain");
	if (!pl)
```

```
		error ("no teleporttrain");
	remove (pl);

	WriteByte (MSG_ALL, SVC_FINALE);
	WriteString (MSG_ALL, "");

	pl = find (world, classname, "player");
	while (pl != world)
	{
		pl.view_ofs = '0 0 0';
		pl.angles = other.v_angle = pos.mangle;
		pl.fixangle = TRUE;			// turn this way immediately
		pl.map = self.map;
		pl.nextthink = time + 0.5;
		pl.takedamage = DAMAGE_NO;
		pl.solid = SOLID_NOT;
		pl.movetype = MOVETYPE_NONE;
		pl.modelindex = 0;
		setorigin (pl, pos.origin);
		pl = find (pl, classname, "player");
	}

	// make fake versions of all players as standins, and move the real
	// players to the intermission spot

	// wait for 1 second
	timer = spawn();
	timer.nextthink = time + 1;
	timer.think = finale_2;
};

void() finale_2 =
{
	local vector	o;

	// start a teleport splash inside shub

	o = shub.origin - '0 100 0';
	WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
	WriteByte (MSG_BROADCAST, TE_TELEPORT);
	WriteCoord (MSG_BROADCAST, o_x);
	WriteCoord (MSG_BROADCAST, o_y);
	WriteCoord (MSG_BROADCAST, o_z);

	sound (shub, CHAN_VOICE, "misc/r_tele1.wav", 1, ATTN_NORM);

	self.nextthink = time + 2;
	self.think = finale_3;
};

void() finale_3 =
{
	// start shub thrashing wildly
	shub.think = old_thrash1;
	sound (shub, CHAN_VOICE, "boss2/death.wav", 1, ATTN_NORM);
	lightstyle(0, "abcdefghijklmlkjihgfedcb");
};

void() finale_4 =
{
	// throw tons of meat chunks
	local	vector	oldo;
```

```
	local	float	x, y, z;
	local	float	r;
	local entity	n;

	sound (self, CHAN_VOICE, "boss2/pop2.wav", 1, ATTN_NORM);

	oldo = self.origin;

	z = 16;
	while (z <= 144)
	{
		x = -64;
		while (x <= 64)
		{
			y = -64;
			while (y <= 64)
			{
				self.origin_x = oldo_x + x;
				self.origin_y = oldo_y + y;
				self.origin_z = oldo_z + z;

				r = random();
				if (r < 0.3)
					ThrowGib ("progs/gib1.mdl", -999);
				else if (r < 0.6)
					ThrowGib ("progs/gib2.mdl", -999);
				else
					ThrowGib ("progs/gib3.mdl", -999);
				y = y + 32;
			}
			x = x + 32;
		}
		z = z + 96;
	}
	// start the end text
	WriteByte (MSG_ALL, SVC_FINALE);
	WriteString (MSG_ALL, "Congratulations and well done! You have\nbeaten the hideous Shub-Niggurath,
and\nher hundreds of ugly changelings and\nmonsters. You have proven that your\nskill and your cunning are greater
than\nall the powers of Quake. You are the\nmaster now. Id Software salutes you.");

// put a player model down
	n = spawn();
	setmodel (n, "progs/player.mdl");
	oldo = oldo - '32 264 0';
	setorigin (n, oldo);
	n.angles = '0 290 0';
	n.frame = 1;

	remove (self);

// switch cd track
	WriteByte (MSG_ALL, SVC_CDTRACK);
	WriteByte (MSG_ALL, 3);
	WriteByte (MSG_ALL, 3);
	lightstyle(0, "m");
};

//==========================================================================

void () nopain =
{
	self.health = 40000;
```

```
};

//==========================================================================


/*QUAKED monster_oldone (1 0 0) (-16 -16 -24) (16 16 32)
*/
void() monster_oldone =
{
	if (deathmatch)
	{
		remove(self);
		return;
	}

	precache_model2 ("progs/oldone.mdl");

	precache_sound2 ("boss2/death.wav");
	precache_sound2 ("boss2/idle.wav");
	precache_sound2 ("boss2/sight.wav");
	precache_sound2 ("boss2/pop2.wav");

	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;

	setmodel (self, "progs/oldone.mdl");
	setsize (self, '-160 -128 -24', '160 128 256');

	self.health = 40000;			// kill by telefrag
	self.think = old_idle1;
	self.nextthink = time + 0.1;
	self.takedamage = DAMAGE_YES;
	self.th_pain = nopain;
	self.th_die = finale_1;
	shub = self;

	total_monsters = total_monsters + 1;
};
```

SHALRATH.QC (VORE)

```
/*
==============================================================================

SHAL-RATH

==============================================================================
*/
$cd id1/models/shalrath
$origin 0 0 24
$base base
$skin skin
$scale 0.7

$frame attack1 attack2 attack3 attack4 attack5 attack6 attack7 attack8
$frame attack9 attack10 attack11

$frame pain1 pain2 pain3 pain4 pain5

$frame death1 death2 death3 death4 death5 death6 death7

$frame  walk1 walk2 walk3 walk4 walk5 walk6 walk7 walk8 walk9 walk10
$frame  walk11 walk12

void() shalrath_pain;
void() ShalMissile;
void() shal_stand     =[      $walk1,       shal_stand    ] {ai_stand();};

void() shal_walk1     =[      $walk2,       shal_walk2    ] {
if (random() < 0.2)
          sound (self, CHAN_VOICE, "shalrath/idle.wav", 1, ATTN_IDLE);
ai_walk(6);};
void() shal_walk2     =[      $walk3,       shal_walk3    ] {ai_walk(4);};
void() shal_walk3     =[      $walk4,       shal_walk4    ] {ai_walk(0);};
void() shal_walk4     =[      $walk5,       shal_walk5    ] {ai_walk(0);};
void() shal_walk5     =[      $walk6,       shal_walk6    ] {ai_walk(0);};
void() shal_walk6     =[      $walk7,       shal_walk7    ] {ai_walk(0);};
void() shal_walk7     =[      $walk8,       shal_walk8    ] {ai_walk(5);};
void() shal_walk8     =[      $walk9,       shal_walk9    ] {ai_walk(6);};
void() shal_walk9     =[      $walk10,      shal_walk10    ] {ai_walk(5);};
void() shal_walk10    =[      $walk11,      shal_walk11    ] {ai_walk(0);};
void() shal_walk11    =[      $walk12,      shal_walk12    ] {ai_walk(4);};
void() shal_walk12    =[      $walk1,       shal_walk1    ] {ai_walk(5);};

void() shal_run1      =[      $walk2,       shal_run2     ] {
if (random() < 0.2)
          sound (self, CHAN_VOICE, "shalrath/idle.wav", 1, ATTN_IDLE);
ai_run(6);};
void() shal_run2      =[      $walk3,       shal_run3     ] {ai_run(4);};
void() shal_run3      =[      $walk4,       shal_run4     ] {ai_run(0);};
void() shal_run4      =[      $walk5,       shal_run5     ] {ai_run(0);};
void() shal_run5      =[      $walk6,       shal_run6     ] {ai_run(0);};
void() shal_run6      =[      $walk7,       shal_run7     ] {ai_run(0);};
void() shal_run7      =[      $walk8,       shal_run8     ] {ai_run(5);};
void() shal_run8      =[      $walk9,       shal_run9     ] {ai_run(6);};
void() shal_run9      =[      $walk10,      shal_run10     ] {ai_run(5);};
void() shal_run10     =[      $walk11,      shal_run11     ] {ai_run(0);};
void() shal_run11     =[      $walk12,      shal_run12     ] {ai_run(4);};
void() shal_run12     =[      $walk1,       shal_run1     ] {ai_run(5);};

void() shal_attack1     =[      $attack1,       shal_attack2    ] {
```

```
sound (self, CHAN_VOICE, "shalrath/attack.wav", 1, ATTN_NORM);
ai_face();
};
void() shal_attack2     =[      $attack2,       shal_attack3    ] {ai_face();};
void() shal_attack3     =[      $attack3,       shal_attack4    ] {ai_face();};
void() shal_attack4     =[      $attack4,       shal_attack5    ] {ai_face();};
void() shal_attack5     =[      $attack5,       shal_attack6    ] {ai_face();};
void() shal_attack6     =[      $attack6,       shal_attack7    ] {ai_face();};
void() shal_attack7     =[      $attack7,       shal_attack8    ] {ai_face();};
void() shal_attack8     =[      $attack8,       shal_attack9    ] {ai_face();};
void() shal_attack9     =[      $attack9,       shal_attack10   ] {ShalMissile();};
void() shal_attack10    =[      $attack10,      shal_attack11   ] {ai_face();};
void() shal_attack11    =[      $attack11,      shal_run1   ] {};

void() shal_pain1       =[      $pain1, shal_pain2      ] {};
void() shal_pain2       =[      $pain2, shal_pain3      ] {};
void() shal_pain3       =[      $pain3, shal_pain4      ] {};
void() shal_pain4       =[      $pain4, shal_pain5      ] {};
void() shal_pain5       =[      $pain5, shal_run1       ] {};

void() shal_death1      =[      $death1,        shal_death2     ] {};
void() shal_death2      =[      $death2,        shal_death3     ] {};
void() shal_death3      =[      $death3,        shal_death4     ] {};
void() shal_death4      =[      $death4,        shal_death5     ] {};
void() shal_death5      =[      $death5,        shal_death6     ] {};
void() shal_death6      =[      $death6,        shal_death7     ] {};
void() shal_death7      =[      $death7,        shal_death7     ] {};


void() shalrath_pain =
{
        if (self.pain_finished > time)
                return;

        sound (self, CHAN_VOICE, "shalrath/pain.wav", 1, ATTN_NORM);
        shal_pain1();
        self.pain_finished = time + 3;
};

void() shalrath_die =
{
// check for gib
        if (self.health < -90)
        {
                sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NORM);
                ThrowHead ("progs/h_shal.mdl", self.health);
                ThrowGib ("progs/gib1.mdl", self.health);
                ThrowGib ("progs/gib2.mdl", self.health);
                ThrowGib ("progs/gib3.mdl", self.health);
                return;
        }

        sound (self, CHAN_VOICE, "shalrath/death.wav", 1, ATTN_NORM);
        shal_death1();
        self.solid = SOLID_NOT;
        // insert death sounds here
};

/*
================
ShalMissile
================
```

```
*/
void() ShalMissileTouch;
void() ShalHome;
void() ShalMissile =
{
	local	entity	missile;
	local	vector	dir;
	local	float	dist, flytime;

	dir = normalize((self.enemy.origin + '0 0 10') - self.origin);
	dist = vlen (self.enemy.origin - self.origin);
	flytime = dist * 0.002;
	if (flytime < 0.1)
		flytime = 0.1;

	self.effects = self.effects | EF_MUZZLEFLASH;
	sound (self, CHAN_WEAPON, "shalrath/attack2.wav", 1, ATTN_NORM);

	missile = spawn ();
	missile.owner = self;

	missile.solid = SOLID_BBOX;
	missile.movetype = MOVETYPE_FLYMISSILE;
	setmodel (missile, "progs/v_spike.mdl");

	setsize (missile, '0 0 0', '0 0 0');

	missile.origin = self.origin + '0 0 10';
	missile.velocity = dir * 400;
	missile.avelocity = '300 300 300';
	missile.nextthink = flytime + time;
	missile.think = ShalHome;
	missile.enemy = self.enemy;
	missile.touch = ShalMissileTouch;
};

void() ShalHome =
{
	local vector	dir, vtemp;
	vtemp = self.enemy.origin + '0 0 10';
	if (self.enemy.health < 1)
	{
		remove(self);
		return;
	}
	dir = normalize(vtemp - self.origin);
	if (skill == 3)
		self.velocity = dir * 350;
	else
		self.velocity = dir * 250;
	self.nextthink = time + 0.2;
	self.think = ShalHome;
};

void() ShalMissileTouch =
{
	if (other == self.owner)
		return;		// don't explode on owner

	if (other.classname == "monster_zombie")
		T_Damage (other, self, self, 110);
	T_RadiusDamage (self, self.owner, 40, world);
```

```
	sound (self, CHAN_WEAPON, "weapons/r_exp3.wav", 1, ATTN_NORM);

	WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
	WriteByte (MSG_BROADCAST, TE_EXPLOSION);
	WriteCoord (MSG_BROADCAST, self.origin_x);
	WriteCoord (MSG_BROADCAST, self.origin_y);
	WriteCoord (MSG_BROADCAST, self.origin_z);

	self.velocity = '0 0 0';
	self.touch = SUB_Null;
	setmodel (self, "progs/s_explod.spr");
	self.solid = SOLID_NOT;
	s_explode1 ();
};

//============================================================================

/*QUAKED monster_shalrath (1 0 0) (-32 -32 -24) (32 32 48) Ambush
*/
void() monster_shalrath =
{
	if (deathmatch)
	{
		remove(self);
		return;
	}
	precache_model2 ("progs/shalrath.mdl");
	precache_model2 ("progs/h_shal.mdl");
	precache_model2 ("progs/v_spike.mdl");

	precache_sound2 ("shalrath/attack.wav");
	precache_sound2 ("shalrath/attack2.wav");
	precache_sound2 ("shalrath/death.wav");
	precache_sound2 ("shalrath/idle.wav");
	precache_sound2 ("shalrath/pain.wav");
	precache_sound2 ("shalrath/sight.wav");

	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;

	setmodel (self, "progs/shalrath.mdl");
	setsize (self, VEC_HULL2_MIN, VEC_HULL2_MAX);
	self.health = 400;

	self.th_stand = shal_stand;
	self.th_walk = shal_walk1;
	self.th_run = shal_run1;
	self.th_die = shalrath_die;
	self.th_pain = shalrath_pain;
	self.th_missile = shal_attack1;

	self.think = walkmonster_start;
	self.nextthink = time + 0.1 + random ()*0.1;

};
```

SHAMBLER.QC

```
/*
==========================================================================

SHAMBLER

==========================================================================
*/

$cd id1/models/shams
$origin 0 0 24
$base base
$skin base

$frame stand1 stand2 stand3 stand4 stand5 stand6 stand7 stand8 stand9
$frame stand10 stand11 stand12 stand13 stand14 stand15 stand16 stand17

$frame walk1 walk2 walk3 walk4 walk5 walk6 walk7
$frame walk8 walk9 walk10 walk11 walk12

$frame  run1 run2 run3 run4 run5 run6

$frame smash1 smash2 smash3 smash4 smash5 smash6 smash7
$frame smash8 smash9 smash10 smash11 smash12

$frame swingr1 swingr2 swingr3 swingr4 swingr5
$frame swingr6 swingr7 swingr8 swingr9

$frame swingl1 swingl2 swingl3 swingl4 swingl5
$frame swingl6 swingl7 swingl8 swingl9

$frame magic1 magic2 magic3 magic4 magic5
$frame magic6 magic7 magic8 magic9 magic10 magic11 magic12

$frame pain1 pain2 pain3 pain4 pain5 pain6

$frame death1 death2 death3 death4 death5 death6
$frame death7 death8 death9 death10 death11

void() sham_stand1      =[      $stand1,        sham_stand2     ] {ai_stand();};
void() sham_stand2      =[      $stand2,        sham_stand3     ] {ai_stand();};
void() sham_stand3      =[      $stand3,        sham_stand4     ] {ai_stand();};
void() sham_stand4      =[      $stand4,        sham_stand5     ] {ai_stand();};
void() sham_stand5      =[      $stand5,        sham_stand6     ] {ai_stand();};
void() sham_stand6      =[      $stand6,        sham_stand7     ] {ai_stand();};
void() sham_stand7      =[      $stand7,        sham_stand8     ] {ai_stand();};
void() sham_stand8      =[      $stand8,        sham_stand9     ] {ai_stand();};
void() sham_stand9      =[      $stand9,        sham_stand10] {ai_stand();};
void() sham_stand10     =[      $stand10,       sham_stand11] {ai_stand();};
void() sham_stand11     =[      $stand11,       sham_stand12] {ai_stand();};
void() sham_stand12     =[      $stand12,       sham_stand13] {ai_stand();};
void() sham_stand13     =[      $stand13,       sham_stand14] {ai_stand();};
void() sham_stand14     =[      $stand14,       sham_stand15] {ai_stand();};
void() sham_stand15     =[      $stand15,       sham_stand16] {ai_stand();};
void() sham_stand16     =[      $stand16,       sham_stand17] {ai_stand();};
void() sham_stand17     =[      $stand17,       sham_stand1     ] {ai_stand();};

void() sham_walk1               =[      $walk1,         sham_walk2 ] {ai_walk(10);};
void() sham_walk2       =[      $walk2,         sham_walk3 ] {ai_walk(9);};
void() sham_walk3       =[      $walk3,         sham_walk4 ] {ai_walk(9);};
void() sham_walk4       =[      $walk4,         sham_walk5 ] {ai_walk(5);};
```

```
void() sham_walk5	=[	$walk5,		sham_walk6 ] {ai_walk(6);};
void() sham_walk6	=[	$walk6,		sham_walk7 ] {ai_walk(12);};
void() sham_walk7	=[	$walk7,		sham_walk8 ] {ai_walk(8);};
void() sham_walk8	=[	$walk8,		sham_walk9 ] {ai_walk(3);};
void() sham_walk9	=[	$walk9,		sham_walk10] {ai_walk(13);};
void() sham_walk10	=[	$walk10,	sham_walk11] {ai_walk(9);};
void() sham_walk11	=[	$walk11,	sham_walk12] {ai_walk(7);};
void() sham_walk12	=[	$walk12,	sham_walk1 ] {ai_walk(7);
if (random() > 0.8)
	sound (self, CHAN_VOICE, "shambler/sidle.wav", 1, ATTN_IDLE);};

void() sham_run1	=[	$run1,		sham_run2	] {ai_run(20);};
void() sham_run2	=[	$run2,		sham_run3	] {ai_run(24);};
void() sham_run3	=[	$run3,		sham_run4	] {ai_run(20);};
void() sham_run4	=[	$run4,		sham_run5	] {ai_run(20);};
void() sham_run5	=[	$run5,		sham_run6	] {ai_run(24);};
void() sham_run6	=[	$run6,		sham_run1	] {ai_run(20);
if (random() > 0.8)
	sound (self, CHAN_VOICE, "shambler/sidle.wav", 1, ATTN_IDLE);
};

void() sham_smash1	=[	$smash1,	sham_smash2	] {
sound (self, CHAN_VOICE, "shambler/melee1.wav", 1, ATTN_NORM);
ai_charge(2);};
void() sham_smash2	=[	$smash2,	sham_smash3	] {ai_charge(6);};
void() sham_smash3	=[	$smash3,	sham_smash4	] {ai_charge(6);};
void() sham_smash4	=[	$smash4,	sham_smash5	] {ai_charge(5);};
void() sham_smash5	=[	$smash5,	sham_smash6	] {ai_charge(4);};
void() sham_smash6	=[	$smash6,	sham_smash7	] {ai_charge(1);};
void() sham_smash7	=[	$smash7,	sham_smash8	] {ai_charge(0);};
void() sham_smash8	=[	$smash8,	sham_smash9	] {ai_charge(0);};
void() sham_smash9	=[	$smash9,	sham_smash10	] {ai_charge(0);};
void() sham_smash10	=[	$smash10,	sham_smash11	] {
local vector	delta;
local float	ldmg;

	if (!self.enemy)
		return;
	ai_charge(0);

	delta = self.enemy.origin - self.origin;

	if (vlen(delta) > 100)
		return;
	if (!CanDamage (self.enemy, self))
		return;

	ldmg = (random() + random() + random()) * 40;
	T_Damage (self.enemy, self, self, ldmg);
	sound (self, CHAN_VOICE, "shambler/smack.wav", 1, ATTN_NORM);

	SpawnMeatSpray (self.origin + v_forward*16, crandom() * 100 * v_right);
	SpawnMeatSpray (self.origin + v_forward*16, crandom() * 100 * v_right);
};
void() sham_smash11	=[	$smash11,	sham_smash12	] {ai_charge(5);};
void() sham_smash12	=[	$smash12,	sham_run1	] {ai_charge(4);};

void() sham_swingr1;

void(float side) ShamClaw =
{
local vector	delta;
```

```
local float		ldmg;

	if (!self.enemy)
		return;
	ai_charge(10);

	delta = self.enemy.origin - self.origin;

	if (vlen(delta) > 100)
		return;

	ldmg = (random() + random() + random()) * 20;
	T_Damage (self.enemy, self, self, ldmg);
	sound (self, CHAN_VOICE, "shambler/smack.wav", 1, ATTN_NORM);

	if (side)
	{
		makevectors (self.angles);
		SpawnMeatSpray (self.origin + v_forward*16, side * v_right);
	}
};

void() sham_swingl1	=[	$swingl1,	sham_swingl2	] {
sound (self, CHAN_VOICE, "shambler/melee2.wav", 1, ATTN_NORM);
ai_charge(5);};
void() sham_swingl2 =[	$swingl2,	sham_swingl3	] {ai_charge(3);};
void() sham_swingl3 =[	$swingl3,	sham_swingl4	] {ai_charge(7);};
void() sham_swingl4 =[	$swingl4,	sham_swingl5	] {ai_charge(3);};
void() sham_swingl5 =[	$swingl5,	sham_swingl6	] {ai_charge(7);};
void() sham_swingl6 =[	$swingl6,	sham_swingl7	] {ai_charge(9);};
void() sham_swingl7 =[	$swingl7,	sham_swingl8	] {ai_charge(5); ShamClaw(250);};
void() sham_swingl8 =[	$swingl8,	sham_swingl9	] {ai_charge(4);};
void() sham_swingl9 =[	$swingl9,	sham_run1	] {
ai_charge(8);
if (random()<0.5)
	self.think = sham_swingr1;
};

void() sham_swingr1	=[	$swingr1,	sham_swingr2	] {
sound (self, CHAN_VOICE, "shambler/melee1.wav", 1, ATTN_NORM);
ai_charge(1);};
void() sham_swingr2	=[	$swingr2,	sham_swingr3	] {ai_charge(8);};
void() sham_swingr3 =[	$swingr3,	sham_swingr4	] {ai_charge(14);};
void() sham_swingr4 =[	$swingr4,	sham_swingr5	] {ai_charge(7);};
void() sham_swingr5 =[	$swingr5,	sham_swingr6	] {ai_charge(3);};
void() sham_swingr6 =[	$swingr6,	sham_swingr7	] {ai_charge(6);};
void() sham_swingr7 =[	$swingr7,	sham_swingr8	] {ai_charge(6); ShamClaw(-250);};
void() sham_swingr8 =[	$swingr8,	sham_swingr9	] {ai_charge(3);};
void() sham_swingr9 =[	$swingr9,	sham_run1	] {ai_charge(1);
ai_charge(10);
if (random()<0.5)
	self.think = sham_swingl1;
};

void() sham_melee =
{
	local float chance;

	chance = random();
	if (chance > 0.6 || self.health == 600)
		sham_smash1 ();
	else if (chance > 0.3)
```

```
		sham_swingr1 ();
	else
		sham_swingl1 ();
};


//============================================================================

void() CastLightning =
{
	local	vector	org, dir;

	self.effects = self.effects | EF_MUZZLEFLASH;

	ai_face ();

	org = self.origin + '0 0 40';

	dir = self.enemy.origin + '0 0 16' - org;
	dir = normalize (dir);

	traceline (org, self.origin + dir*600, TRUE, self);

	WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
	WriteByte (MSG_BROADCAST, TE_LIGHTNING1);
	WriteEntity (MSG_BROADCAST, self);
	WriteCoord (MSG_BROADCAST, org_x);
	WriteCoord (MSG_BROADCAST, org_y);
	WriteCoord (MSG_BROADCAST, org_z);
	WriteCoord (MSG_BROADCAST, trace_endpos_x);
	WriteCoord (MSG_BROADCAST, trace_endpos_y);
	WriteCoord (MSG_BROADCAST, trace_endpos_z);

	LightningDamage (org, trace_endpos, self, 10);
};

void() sham_magic1	=[	$magic1,	sham_magic2	] {ai_face();
	sound (self, CHAN_WEAPON, "shambler/sattck1.wav", 1, ATTN_NORM);
};
void() sham_magic2	=[	$magic2,	sham_magic3	] {ai_face();};
void() sham_magic3	=[	$magic3,	sham_magic4	] {ai_face();self.nextthink = self.nextthink + 0.2;
local entity o;

self.effects = self.effects | EF_MUZZLEFLASH;
ai_face();
self.owner = spawn();
o = self.owner;
setmodel (o, "progs/s_light.mdl");
setorigin (o, self.origin);
o.angles = self.angles;
o.nextthink = time + 0.7;
o.think = SUB_Remove;
};
void() sham_magic4	=[	$magic4,	sham_magic5	]
{
self.effects = self.effects | EF_MUZZLEFLASH;
self.owner.frame = 1;
};
void() sham_magic5	=[	$magic5,	sham_magic6	]
{
self.effects = self.effects | EF_MUZZLEFLASH;
self.owner.frame = 2;
```

```
};
void() sham_magic6     =[      $magic6,       sham_magic9    ]
{
remove (self.owner);
CastLightning();
sound (self, CHAN_WEAPON, "shambler/sboom.wav", 1, ATTN_NORM);
};
void() sham_magic9     =[      $magic9,       sham_magic10   ]
{CastLightning();};
void() sham_magic10    =[      $magic10,      sham_magic11   ]
{CastLightning();};
void() sham_magic11    =[      $magic11,      sham_magic12   ]
{
if (skill == 3)
        CastLightning();
};
void() sham_magic12    =[      $magic12,      sham_run1      ] {};



void() sham_pain1      =[      $pain1, sham_pain2      ] {};
void() sham_pain2      =[      $pain2, sham_pain3      ] {};
void() sham_pain3      =[      $pain3, sham_pain4      ] {};
void() sham_pain4      =[      $pain4, sham_pain5      ] {};
void() sham_pain5      =[      $pain5, sham_pain6      ] {};
void() sham_pain6      =[      $pain6, sham_run1      ] {};

void(entity attacker, float damage)          sham_pain =
{
        sound (self, CHAN_VOICE, "shambler/shurt2.wav", 1, ATTN_NORM);

        if (self.health <= 0)
                return;                 // allready dying, don't go into pain frame

        if (random()*400 > damage)
                return;                 // didn't flinch

        if (self.pain_finished > time)
                return;
        self.pain_finished = time + 2;

        sham_pain1 ();
};


//==========================================================================

void() sham_death1     =[      $death1,       sham_death2     ] {};
void() sham_death2     =[      $death2,       sham_death3     ] {};
void() sham_death3     =[      $death3,       sham_death4     ] {self.solid = SOLID_NOT;};
void() sham_death4     =[      $death4,       sham_death5     ] {};
void() sham_death5     =[      $death5,       sham_death6     ] {};
void() sham_death6     =[      $death6,       sham_death7     ] {};
void() sham_death7     =[      $death7,       sham_death8     ] {};
void() sham_death8     =[      $death8,       sham_death9     ] {};
void() sham_death9     =[      $death9,       sham_death10    ] {};
void() sham_death10    =[      $death10,      sham_death11    ] {};
void() sham_death11    =[      $death11,      sham_death11    ] {};

void() sham_die =
{
// check for gib
```

```
	if (self.health < -60)
	{
		sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NORM);
		ThrowHead ("progs/h_shams.mdl", self.health);
		ThrowGib ("progs/gib1.mdl", self.health);
		ThrowGib ("progs/gib2.mdl", self.health);
		ThrowGib ("progs/gib3.mdl", self.health);
		return;
	}

// regular death
	sound (self, CHAN_VOICE, "shambler/sdeath.wav", 1, ATTN_NORM);
	sham_death1 ();
};

//============================================================================


/*QUAKED monster_shambler (1 0 0) (-32 -32 -24) (32 32 64) Ambush
*/
void() monster_shambler =
{
	if (deathmatch)
	{
		remove(self);
		return;
	}
	precache_model ("progs/shambler.mdl");
	precache_model ("progs/s_light.mdl");
	precache_model ("progs/h_shams.mdl");
	precache_model ("progs/bolt.mdl");

	precache_sound ("shambler/sattck1.wav");
	precache_sound ("shambler/sboom.wav");
	precache_sound ("shambler/sdeath.wav");
	precache_sound ("shambler/shurt2.wav");
	precache_sound ("shambler/sidle.wav");
	precache_sound ("shambler/ssight.wav");
	precache_sound ("shambler/melee1.wav");
	precache_sound ("shambler/melee2.wav");
	precache_sound ("shambler/smack.wav");

	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;
	setmodel (self, "progs/shambler.mdl");

	setsize (self, VEC_HULL2_MIN, VEC_HULL2_MAX);
	self.health = 600;

	self.th_stand = sham_stand1;
	self.th_walk = sham_walk1;
	self.th_run = sham_run1;
	self.th_die = sham_die;
	self.th_melee = sham_melee;
	self.th_missile = sham_magic1;
	self.th_pain = sham_pain;

	walkmonster_start();
};
```

SOLDIER.QC

```
/*
==============================================================================

SOLDIER / PLAYER

==============================================================================
*/

$cd id1/models/soldier3
$origin 0 -6 24
$base base
$skin skin

$frame stand1 stand2 stand3 stand4 stand5 stand6 stand7 stand8

$frame death1 death2 death3 death4 death5 death6 death7 death8
$frame death9 death10

$frame deathc1 deathc2 deathc3 deathc4 deathc5 deathc6 deathc7 deathc8
$frame deathc9 deathc10 deathc11

$frame load1 load2 load3 load4 load5 load6 load7 load8 load9 load10 load11

$frame pain1 pain2 pain3 pain4 pain5 pain6

$frame painb1 painb2 painb3 painb4 painb5 painb6 painb7 painb8 painb9 painb10
$frame painb11 painb12 painb13 painb14

$frame painc1 painc2 painc3 painc4 painc5 painc6 painc7 painc8 painc9 painc10
$frame painc11 painc12 painc13

$frame run1 run2 run3 run4 run5 run6 run7 run8

$frame shoot1 shoot2 shoot3 shoot4 shoot5 shoot6 shoot7 shoot8 shoot9

$frame prowl_1 prowl_2 prowl_3 prowl_4 prowl_5 prowl_6 prowl_7 prowl_8
$frame prowl_9 prowl_10 prowl_11 prowl_12 prowl_13 prowl_14 prowl_15 prowl_16
$frame prowl_17 prowl_18 prowl_19 prowl_20 prowl_21 prowl_22 prowl_23 prowl_24

/*
==============================================================================
SOLDIER CODE
==============================================================================
*/

void() army_fire;

void()	army_stand1	=[	$stand1,	army_stand2	] {ai_stand();};
void()	army_stand2	=[	$stand2,	army_stand3	] {ai_stand();};
void()	army_stand3	=[	$stand3,	army_stand4	] {ai_stand();};
void()	army_stand4	=[	$stand4,	army_stand5	] {ai_stand();};
void()	army_stand5	=[	$stand5,	army_stand6	] {ai_stand();};
void()	army_stand6	=[	$stand6,	army_stand7	] {ai_stand();};
void()	army_stand7	=[	$stand7,	army_stand8	] {ai_stand();};
void()	army_stand8	=[	$stand8,	army_stand1	] {ai_stand();};

void()	army_walk1	=[	$prowl_1,	army_walk2	] {
if (random() < 0.2)
	sound (self, CHAN_VOICE, "soldier/idle.wav", 1, ATTN_IDLE);
ai_walk(1);};
```

```
void()	army_walk2	=[	$prowl_2,	army_walk3	] {ai_walk(1);};
void()	army_walk3	=[	$prowl_3,	army_walk4	] {ai_walk(1);};
void()	army_walk4	=[	$prowl_4,	army_walk5	] {ai_walk(1);};
void()	army_walk5	=[	$prowl_5,	army_walk6	] {ai_walk(2);};
void()	army_walk6	=[	$prowl_6,	army_walk7	] {ai_walk(3);};
void()	army_walk7	=[	$prowl_7,	army_walk8	] {ai_walk(4);};
void()	army_walk8	=[	$prowl_8,	army_walk9	] {ai_walk(4);};
void()	army_walk9	=[	$prowl_9,	army_walk10	] {ai_walk(2);};
void()	army_walk10	=[	$prowl_10,	army_walk11	] {ai_walk(2);};
void()	army_walk11	=[	$prowl_11,	army_walk12	] {ai_walk(2);};
void()	army_walk12	=[	$prowl_12,	army_walk13	] {ai_walk(1);};
void()	army_walk13	=[	$prowl_13,	army_walk14	] {ai_walk(0);};
void()	army_walk14	=[	$prowl_14,	army_walk15	] {ai_walk(1);};
void()	army_walk15	=[	$prowl_15,	army_walk16	] {ai_walk(1);};
void()	army_walk16	=[	$prowl_16,	army_walk17	] {ai_walk(1);};
void()	army_walk17	=[	$prowl_17,	army_walk18	] {ai_walk(3);};
void()	army_walk18	=[	$prowl_18,	army_walk19	] {ai_walk(3);};
void()	army_walk19	=[	$prowl_19,	army_walk20	] {ai_walk(3);};
void()	army_walk20	=[	$prowl_20,	army_walk21	] {ai_walk(3);};
void()	army_walk21	=[	$prowl_21,	army_walk22	] {ai_walk(2);};
void()	army_walk22	=[	$prowl_22,	army_walk23	] {ai_walk(1);};
void()	army_walk23	=[	$prowl_23,	army_walk24	] {ai_walk(1);};
void()	army_walk24	=[	$prowl_24,	army_walk1	] {ai_walk(1);};

void()	army_run1	=[	$run1,	army_run2	] {
if (random() < 0.2)
	sound (self, CHAN_VOICE, "soldier/idle.wav", 1, ATTN_IDLE);
ai_run(11);};
void()	army_run2	=[	$run2,	army_run3	] {ai_run(15);};
void()	army_run3	=[	$run3,	army_run4	] {ai_run(10);};
void()	army_run4	=[	$run4,	army_run5	] {ai_run(10);};
void()	army_run5	=[	$run5,	army_run6	] {ai_run(8);};
void()	army_run6	=[	$run6,	army_run7	] {ai_run(15);};
void()	army_run7	=[	$run7,	army_run8	] {ai_run(10);};
void()	army_run8	=[	$run8,	army_run1	] {ai_run(8);};

void()	army_atk1	=[	$shoot1,	army_atk2	] {ai_face();};
void()	army_atk2	=[	$shoot2,	army_atk3	] {ai_face();};
void()	army_atk3	=[	$shoot3,	army_atk4	] {ai_face();};
void()	army_atk4	=[	$shoot4,	army_atk5	] {ai_face();};
void()	army_atk5	=[	$shoot5,	army_atk6	] {ai_face();army_fire();
self.effects = self.effects | EF_MUZZLEFLASH;};
void()	army_atk6	=[	$shoot6,	army_atk7	] {ai_face();};
void()	army_atk7	=[	$shoot7,	army_atk8	] {ai_face();SUB_CheckRefire (army_atk1);};
void()	army_atk8	=[	$shoot8,	army_atk9	] {ai_face();};
void()	army_atk9	=[	$shoot9,	army_run1	] {ai_face();};


void()	army_pain1	=[	$pain1,	army_pain2	] {};
void()	army_pain2	=[	$pain2,	army_pain3	] {};
void()	army_pain3	=[	$pain3,	army_pain4	] {};
void()	army_pain4	=[	$pain4,	army_pain5	] {};
void()	army_pain5	=[	$pain5,	army_pain6	] {};
void()	army_pain6	=[	$pain6,	army_run1	] {ai_pain(1);};

void()	army_painb1	=[	$painb1,	army_painb2	] {};
void()	army_painb2	=[	$painb2,	army_painb3	] {ai_painforward(13);};
void()	army_painb3	=[	$painb3,	army_painb4	] {ai_painforward(9);};
void()	army_painb4	=[	$painb4,	army_painb5	] {};
void()	army_painb5	=[	$painb5,	army_painb6	] {};
void()	army_painb6	=[	$painb6,	army_painb7	] {};
void()	army_painb7	=[	$painb7,	army_painb8	] {};
```

```
void()	army_painb8	=[	$painb8,		army_painb9	] {};
void()	army_painb9	=[	$painb9,		army_painb10] {};
void()	army_painb10=[$painb10,		army_painb11] {};
void()	army_painb11=[$painb11,		army_painb12] {};
void()	army_painb12=[$painb12,		army_painb13] {ai_pain(2);};
void()	army_painb13=[$painb13,		army_painb14] {};
void()	army_painb14=[$painb14,		army_run1	] {};

void()	army_painc1	=[	$painc1,		army_painc2	] {};
void()	army_painc2	=[	$painc2,		army_painc3	] {ai_pain(1);};
void()	army_painc3	=[	$painc3,		army_painc4	] {};
void()	army_painc4	=[	$painc4,		army_painc5	] {};
void()	army_painc5	=[	$painc5,		army_painc6	] {ai_painforward(1);};
void()	army_painc6	=[	$painc6,		army_painc7	] {ai_painforward(1);};
void()	army_painc7	=[	$painc7,		army_painc8	] {};
void()	army_painc8	=[	$painc8,		army_painc9	] {ai_pain(1);};
void()	army_painc9	=[	$painc9,		army_painc10] {ai_painforward(4);};
void()	army_painc10=[$painc10,		army_painc11] {ai_painforward(3);};
void()	army_painc11=[$painc11,		army_painc12] {ai_painforward(6);};
void()	army_painc12=[$painc12,		army_painc13] {ai_painforward(8);};
void()	army_painc13=[$painc13,		army_run1] {};

void(entity attacker, float damage)		army_pain =
{
	local float r;

	if (self.pain_finished > time)
		return;

	r = random();

	if (r < 0.2)
	{
		self.pain_finished = time + 0.6;
		army_pain1 ();
		sound (self, CHAN_VOICE, "soldier/pain1.wav", 1, ATTN_NORM);
	}
	else if (r < 0.6)
	{
		self.pain_finished = time + 1.1;
		army_painb1 ();
		sound (self, CHAN_VOICE, "soldier/pain2.wav", 1, ATTN_NORM);
	}
	else
	{
		self.pain_finished = time + 1.1;
		army_painc1 ();
		sound (self, CHAN_VOICE, "soldier/pain2.wav", 1, ATTN_NORM);
	}
};


void() army_fire =
{
	local	vector	dir;
	local	entity	en;

	ai_face();

	sound (self, CHAN_WEAPON, "soldier/sattck1.wav", 1, ATTN_NORM);

// fire somewhat behind the player, so a dodging player is harder to hit
```

```
	en = self.enemy;

	dir = en.origin - en.velocity*0.2;
	dir = normalize (dir - self.origin);

	FireBullets (4, dir, '0.1 0.1 0');
};



void()	army_die1	=[	$death1,	army_die2	] {};
void()	army_die2	=[	$death2,	army_die3	] {};
void()	army_die3	=[	$death3,	army_die4	]
{self.solid = SOLID_NOT;self.ammo_shells = 5;DropBackpack();};
void()	army_die4	=[	$death4,	army_die5	] {};
void()	army_die5	=[	$death5,	army_die6	] {};
void()	army_die6	=[	$death6,	army_die7	] {};
void()	army_die7	=[	$death7,	army_die8	] {};
void()	army_die8	=[	$death8,	army_die9	] {};
void()	army_die9	=[	$death9,	army_die10	] {};
void()	army_die10	=[	$death10,	army_die10	] {};

void()	army_cdie1	=[	$deathc1,	army_cdie2	] {};
void()	army_cdie2	=[	$deathc2,	army_cdie3	] {ai_back(5);};
void()	army_cdie3	=[	$deathc3,	army_cdie4	]
{self.solid = SOLID_NOT;self.ammo_shells = 5;DropBackpack();ai_back(4);};
void()	army_cdie4	=[	$deathc4,	army_cdie5	] {ai_back(13);};
void()	army_cdie5	=[	$deathc5,	army_cdie6	] {ai_back(3);};
void()	army_cdie6	=[	$deathc6,	army_cdie7	] {ai_back(4);};
void()	army_cdie7	=[	$deathc7,	army_cdie8	] {};
void()	army_cdie8	=[	$deathc8,	army_cdie9	] {};
void()	army_cdie9	=[	$deathc9,	army_cdie10	] {};
void()	army_cdie10	=[	$deathc10,	army_cdie11	] {};
void()	army_cdie11	=[	$deathc11,	army_cdie11	] {};


void() army_die =
{
// check for gib
	if (self.health < -35)
	{
		sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NORM);
		ThrowHead ("progs/h_guard.mdl", self.health);
		ThrowGib ("progs/gib1.mdl", self.health);
		ThrowGib ("progs/gib2.mdl", self.health);
		ThrowGib ("progs/gib3.mdl", self.health);
		return;
	}

// regular death
	sound (self, CHAN_VOICE, "soldier/death1.wav", 1, ATTN_NORM);
	if (random() < 0.5)
		army_die1 ();
	else
		army_cdie1 ();
};


/*QUAKED monster_army (1 0 0) (-16 -16 -24) (16 16 40) Ambush
*/
void() monster_army =
{
```

```
	if (deathmatch)
	{
		remove(self);
		return;
	}
	precache_model ("progs/soldier.mdl");
	precache_model ("progs/h_guard.mdl");
	precache_model ("progs/gib1.mdl");
	precache_model ("progs/gib2.mdl");
	precache_model ("progs/gib3.mdl");

	precache_sound ("soldier/death1.wav");
	precache_sound ("soldier/idle.wav");
	precache_sound ("soldier/pain1.wav");
	precache_sound ("soldier/pain2.wav");
	precache_sound ("soldier/sattck1.wav");
	precache_sound ("soldier/sight1.wav");

	precache_sound ("player/udeath.wav");		// gib death


	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;

	setmodel (self, "progs/soldier.mdl");

	setsize (self, '-16 -16 -24', '16 16 40');
	self.health = 30;

	self.th_stand = army_stand1;
	self.th_walk = army_walk1;
	self.th_run = army_run1;
	self.th_missile = army_atk1;
	self.th_pain = army_pain;
	self.th_die = army_die;

	walkmonster_start ();
};
```

TARBABY.QC (SPAWN)

```
/*
==============================================================================

BLOB

==============================================================================
*/

$cd id1/models/tarbaby
$origin 0 0 24
$base base

$skin skin

$frame walk1 walk2 walk3 walk4  walk5 walk6 walk7 walk8 walk9 walk10
$frame walk11 walk12 walk13 walk14 walk15 walk16 walk17 walk18 walk19
$frame walk20 walk21 walk22 walk23 walk24 walk25

$frame run1 run2 run3 run4 run5 run6  run7 run8 run9 run10 run11 run12 run13
$frame run14 run15 run16 run17 run18 run19 run20 run21 run22 run23
$frame run24 run25

$frame jump1 jump2 jump3 jump4 jump5 jump6

$frame fly1 fly2 fly3 fly4

$frame exp

void()  tbaby_stand1    =[      $walk1,         tbaby_stand1    ] {ai_stand();};

void()  tbaby_hang1     =[      $walk1,         tbaby_hang1     ] {ai_stand();};

void()  tbaby_walk1     =[      $walk1,         tbaby_walk2     ] {ai_turn();};
void()  tbaby_walk2     =[      $walk2,         tbaby_walk3     ] {ai_turn();};
void()  tbaby_walk3     =[      $walk3,         tbaby_walk4     ] {ai_turn();};
void()  tbaby_walk4     =[      $walk4,         tbaby_walk5     ] {ai_turn();};
void()  tbaby_walk5     =[      $walk5,         tbaby_walk6     ] {ai_turn();};
void()  tbaby_walk6     =[      $walk6,         tbaby_walk7     ] {ai_turn();};
void()  tbaby_walk7     =[      $walk7,         tbaby_walk8     ] {ai_turn();};
void()  tbaby_walk8     =[      $walk8,         tbaby_walk9     ] {ai_turn();};
void()  tbaby_walk9     =[      $walk9,         tbaby_walk10    ] {ai_turn();};
void()  tbaby_walk10    =[      $walk10,                tbaby_walk11    ] {ai_turn();};
void()  tbaby_walk11    =[      $walk11,                tbaby_walk12    ] {ai_walk(2);};
void()  tbaby_walk12    =[      $walk12,                tbaby_walk13    ] {ai_walk(2);};
void()  tbaby_walk13    =[      $walk13,                tbaby_walk14    ] {ai_walk(2);};
void()  tbaby_walk14    =[      $walk14,                tbaby_walk15    ] {ai_walk(2);};
void()  tbaby_walk15    =[      $walk15,                tbaby_walk16    ] {ai_walk(2);};
void()  tbaby_walk16    =[      $walk16,                tbaby_walk17    ] {ai_walk(2);};
void()  tbaby_walk17    =[      $walk17,                tbaby_walk18    ] {ai_walk(2);};
void()  tbaby_walk18    =[      $walk18,                tbaby_walk19    ] {ai_walk(2);};
void()  tbaby_walk19    =[      $walk19,                tbaby_walk20    ] {ai_walk(2);};
void()  tbaby_walk20    =[      $walk20,                tbaby_walk21    ] {ai_walk(2);};
void()  tbaby_walk21    =[      $walk21,                tbaby_walk22    ] {ai_walk(2);};
void()  tbaby_walk22    =[      $walk22,                tbaby_walk23    ] {ai_walk(2);};
void()  tbaby_walk23    =[      $walk23,                tbaby_walk24    ] {ai_walk(2);};
void()  tbaby_walk24    =[      $walk24,                tbaby_walk25    ] {ai_walk(2);};
void()  tbaby_walk25    =[      $walk25,                tbaby_walk1     ] {ai_walk(2);};

void()  tbaby_run1      =[      $run1,          tbaby_run2      ] {ai_face();};
void()  tbaby_run2      =[      $run2,          tbaby_run3      ] {ai_face();};
```

```
void()	tbaby_run3	=[	$run3,		tbaby_run4	] {ai_face();};
void()	tbaby_run4	=[	$run4,		tbaby_run5	] {ai_face();};
void()	tbaby_run5	=[	$run5,		tbaby_run6	] {ai_face();};
void()	tbaby_run6	=[	$run6,		tbaby_run7	] {ai_face();};
void()	tbaby_run7	=[	$run7,		tbaby_run8	] {ai_face();};
void()	tbaby_run8	=[	$run8,		tbaby_run9	] {ai_face();};
void()	tbaby_run9	=[	$run9,		tbaby_run10	] {ai_face();};
void()	tbaby_run10	=[	$run10,		tbaby_run11	] {ai_face();};
void()	tbaby_run11	=[	$run11,		tbaby_run12	] {ai_run(2);};
void()	tbaby_run12	=[	$run12,		tbaby_run13	] {ai_run(2);};
void()	tbaby_run13	=[	$run13,		tbaby_run14	] {ai_run(2);};
void()	tbaby_run14	=[	$run14,		tbaby_run15	] {ai_run(2);};
void()	tbaby_run15	=[	$run15,		tbaby_run16	] {ai_run(2);};
void()	tbaby_run16	=[	$run16,		tbaby_run17	] {ai_run(2);};
void()	tbaby_run17	=[	$run17,		tbaby_run18	] {ai_run(2);};
void()	tbaby_run18	=[	$run18,		tbaby_run19	] {ai_run(2);};
void()	tbaby_run19	=[	$run19,		tbaby_run20	] {ai_run(2);};
void()	tbaby_run20	=[	$run20,		tbaby_run21	] {ai_run(2);};
void()	tbaby_run21	=[	$run21,		tbaby_run22	] {ai_run(2);};
void()	tbaby_run22	=[	$run22,		tbaby_run23	] {ai_run(2);};
void()	tbaby_run23	=[	$run23,		tbaby_run24	] {ai_run(2);};
void()	tbaby_run24	=[	$run24,		tbaby_run25	] {ai_run(2);};
void()	tbaby_run25	=[	$run25,		tbaby_run1	] {ai_run(2);};


//==========================================================================


void()	tbaby_jump1;

void()	Tar_JumpTouch =
{
	local	float	ldmg;

	if (other.takedamage && other.classname != self.classname)
	{
		if ( vlen(self.velocity) > 400 )
		{
			ldmg = 10 + 10*random();
			T_Damage (other, self, self, ldmg);
			sound (self, CHAN_WEAPON, "blob/hit1.wav", 1, ATTN_NORM);
		}
	}
	else
		sound (self, CHAN_WEAPON, "blob/land1.wav", 1, ATTN_NORM);


	if (!checkbottom(self))
	{
		if (self.flags & FL_ONGROUND)
		{	// jump randomly to not get hung up
//dprint ("popjump\n");
	self.touch = SUB_Null;
	self.think = tbaby_run1;
	self.movetype = MOVETYPE_STEP;
	self.nextthink = time + 0.1;

//				self.velocity_x = (random() - 0.5) * 600;
//				self.velocity_y = (random() - 0.5) * 600;
//				self.velocity_z = 200;
//				self.flags = self.flags - FL_ONGROUND;
		}
```

```
		return;  // not on ground yet
	}

	self.touch = SUB_Null;
	self.think = tbaby_jump1;
	self.nextthink = time + 0.1;
};

void()	tbaby_jump5;

void()	tbaby_fly1		=[	$fly1,	tbaby_fly2	] {};
void()	tbaby_fly2		=[	$fly2,	tbaby_fly3	] {};
void()	tbaby_fly3		=[	$fly3,	tbaby_fly4	] {};
void()	tbaby_fly4		=[	$fly4,	tbaby_fly1	] {
self.cnt = self.cnt + 1;
if (self.cnt == 4)
{
//dprint ("spawn hop\n");
tbaby_jump5 ();
}
};

void()	tbaby_jump1		=[	$jump1,tbaby_jump2		] {ai_face();};
void()	tbaby_jump2		=[	$jump2,tbaby_jump3		] {ai_face();};
void()	tbaby_jump3		=[	$jump3,tbaby_jump4		] {ai_face();};
void()	tbaby_jump4		=[	$jump4,tbaby_jump5		] {ai_face();};
void()	tbaby_jump5		=[	$jump5,tbaby_jump6		]
{
	self.movetype = MOVETYPE_BOUNCE;
	self.touch = Tar_JumpTouch;
	makevectors (self.angles);
	self.origin_z = self.origin_z + 1;
	self.velocity = v_forward * 600 + '0 0 200';
	self.velocity_z = self.velocity_z + random()*150;
	if (self.flags & FL_ONGROUND)
		self.flags = self.flags - FL_ONGROUND;
	self.cnt = 0;
};
void()	tbaby_jump6	=[	$jump6,tbaby_fly1	] {};



//============================================================================

void()	tbaby_die1	=[	$exp,		tbaby_die2	] {
self.takedamage = DAMAGE_NO;
};
void()	tbaby_die2	=[	$exp,		tbaby_run1	]
{
	T_RadiusDamage (self, self, 120, world);

	sound (self, CHAN_VOICE, "blob/death1.wav", 1, ATTN_NORM);
	self.origin = self.origin - 8*normalize(self.velocity);

	WriteByte (MSG_BROADCAST, SVC_TEMPENTITY);
	WriteByte (MSG_BROADCAST, TE_TAREXPLOSION);
	WriteCoord (MSG_BROADCAST, self.origin_x);
	WriteCoord (MSG_BROADCAST, self.origin_y);
	WriteCoord (MSG_BROADCAST, self.origin_z);

	BecomeExplosion ();
};
```

```
//==========================================================================


/*QUAKED monster_tarbaby (1 0 0) (-16 -16 -24) (16 16 24) Ambush
*/
void() monster_tarbaby =
{
	if (deathmatch)
	{
		remove(self);
		return;
	}
	precache_model2 ("progs/tarbaby.mdl");

	precache_sound2 ("blob/death1.wav");
	precache_sound2 ("blob/hit1.wav");
	precache_sound2 ("blob/land1.wav");
	precache_sound2 ("blob/sight1.wav");

	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;

	setmodel (self, "progs/tarbaby.mdl");

	setsize (self, '-16 -16 -24', '16 16 40');
	self.health = 80;

	self.th_stand = tbaby_stand1;
	self.th_walk = tbaby_walk1;
	self.th_run = tbaby_run1;
	self.th_missile = tbaby_jump1;
	self.th_melee = tbaby_jump1;
	self.th_die = tbaby_die1;

	walkmonster_start ();
};
```

WIZARD.QC (SCRAG)

```
/*
==============================================================================

WIZARD

==============================================================================
*/

$cd id1/models/a_wizard
$origin 0 0 24
$base wizbase
$skin wizbase

$frame hover1 hover2 hover3 hover4 hover5 hover6 hover7 hover8
$frame hover9 hover10 hover11 hover12 hover13 hover14 hover15

$frame fly1 fly2 fly3 fly4 fly5 fly6 fly7 fly8 fly9 fly10
$frame fly11 fly12 fly13 fly14

$frame magatt1 magatt2 magatt3 magatt4 magatt5 magatt6 magatt7
$frame magatt8 magatt9 magatt10 magatt11 magatt12 magatt13

$frame pain1 pain2 pain3 pain4

$frame death1 death2 death3 death4 death5 death6 death7 death8

/*
==============================================================================

WIZARD

If the player moves behind cover before the missile is launched, launch it
at the last visible spot with no velocity leading, in hopes that the player
will duck back out and catch it.
==============================================================================
*/

/*
==============
LaunchMissile

Sets the given entities velocity and angles so that it will hit self.enemy
if self.enemy maintains it's current velocity
0.1 is moderately accurate, 0.0 is totally accurate
==============
*/
void(entity missile, float mspeed, float accuracy) LaunchMissile =
{
        local   vector  vec, move;
        local   float   fly;

        makevectors (self.angles);

// set missile speed
        vec = self.enemy.origin + self.enemy.mins + self.enemy.size * 0.7 - missile.origin;

// calc aproximate time for missile to reach vec
        fly = vlen (vec) / mspeed;

// get the entities xy velocity
```

```
	move = self.enemy.velocity;
	move_z = 0;

// project the target forward in time
	vec = vec + move * fly;

	vec = normalize(vec);
	vec = vec + accuracy*v_up*(random()- 0.5) + accuracy*v_right*(random()- 0.5);

	missile.velocity = vec * mspeed;

	missile.angles = '0 0 0';
	missile.angles_y = vectoyaw(missile.velocity);

// set missile duration
	missile.nextthink = time + 5;
	missile.think = SUB_Remove;
};


void() wiz_run1;
void() wiz_side1;

/*
=================
WizardCheckAttack
=================
*/
float()	WizardCheckAttack =
{
	local vector	spot1, spot2;
	local entity	targ;
	local float		chance;

	if (time < self.attack_finished)
		return FALSE;
	if (!enemy_vis)
		return FALSE;

	if (enemy_range == RANGE_FAR)
	{
		if (self.attack_state != AS_STRAIGHT)
		{
			self.attack_state = AS_STRAIGHT;
			wiz_run1 ();
		}
		return FALSE;
	}

	targ = self.enemy;

// see if any entities are in the way of the shot
	spot1 = self.origin + self.view_ofs;
	spot2 = targ.origin + targ.view_ofs;

	traceline (spot1, spot2, FALSE, self);

	if (trace_ent != targ)
	{	// don't have a clear shot, so move to a side
		if (self.attack_state != AS_STRAIGHT)
		{
			self.attack_state = AS_STRAIGHT;
```

```
                wiz_run1 ();
        }
        return FALSE;
    }

    if (enemy_range == RANGE_MELEE)
        chance = 0.9;
    else if (enemy_range == RANGE_NEAR)
        chance = 0.6;
    else if (enemy_range == RANGE_MID)
        chance = 0.2;
    else
        chance = 0;

    if (random () < chance)
    {
        self.attack_state = AS_MISSILE;
        return TRUE;
    }

    if (enemy_range == RANGE_MID)
    {
        if (self.attack_state != AS_STRAIGHT)
        {
            self.attack_state = AS_STRAIGHT;
            wiz_run1 ();
        }
    }
    else
    {
        if (self.attack_state != AS_SLIDING)
        {
            self.attack_state = AS_SLIDING;
            wiz_side1 ();
        }
    }

    return FALSE;
};

/*
=================
WizardAttackFinished
=================
*/
float()    WizardAttackFinished =
{
    if (enemy_range >= RANGE_MID || !enemy_vis)
    {
        self.attack_state = AS_STRAIGHT;
        self.think = wiz_run1;
    }
    else
    {
        self.attack_state = AS_SLIDING;
        self.think = wiz_side1;
    }
};

/*
==============================================================================
```

FAST ATTACKS

```
==============================================================================
*/

void() Wiz_FastFire =
{
	local vector		vec;
	local vector		dst;

	if (self.owner.health > 0)
	{
		self.owner.effects = self.owner.effects | EF_MUZZLEFLASH;

		makevectors (self.enemy.angles);
		dst = self.enemy.origin - 13*self.movedir;

		vec = normalize(dst - self.origin);
		sound (self, CHAN_WEAPON, "wizard/wattack.wav", 1, ATTN_NORM);
		launch_spike (self.origin, vec);
		newmis.velocity = vec*600;
		newmis.owner = self.owner;
		newmis.classname = "wizspike";
		setmodel (newmis, "progs/w_spike.mdl");
		setsize (newmis, VEC_ORIGIN, VEC_ORIGIN);
	}

	remove (self);
};


void() Wiz_StartFast =
{
	local	entity	missile;

	sound (self, CHAN_WEAPON, "wizard/wattack.wav", 1, ATTN_NORM);
	self.v_angle = self.angles;
	makevectors (self.angles);

	missile = spawn ();
	missile.owner = self;
	missile.nextthink = time + 0.6;
	setsize (missile, '0 0 0', '0 0 0');
	setorigin (missile, self.origin + '0 0 30' + v_forward*14 + v_right*14);
	missile.enemy = self.enemy;
	missile.nextthink = time + 0.8;
	missile.think = Wiz_FastFire;
	missile.movedir = v_right;

	missile = spawn ();
	missile.owner = self;
	missile.nextthink = time + 1;
	setsize (missile, '0 0 0', '0 0 0');
	setorigin (missile, self.origin + '0 0 30' + v_forward*14 + v_right* -14);
	missile.enemy = self.enemy;
	missile.nextthink = time + 0.3;
	missile.think = Wiz_FastFire;
	missile.movedir = VEC_ORIGIN - v_right;
};
```

```
void() Wiz_idlesound =
{
local float wr;
	wr = random() * 5;

	if (self.waitmin < time)
	{
		self.waitmin = time + 2;
		if (wr > 4.5)
			sound (self, CHAN_VOICE, "wizard/widle1.wav", 1,  ATTN_IDLE);
		if (wr < 1.5)
			sound (self, CHAN_VOICE, "wizard/widle2.wav", 1, ATTN_IDLE);
	}
	return;
};

void()	wiz_stand1	=[	$hover1,		wiz_stand2	] {ai_stand();};
void()	wiz_stand2	=[	$hover2,		wiz_stand3	] {ai_stand();};
void()	wiz_stand3	=[	$hover3,		wiz_stand4	] {ai_stand();};
void()	wiz_stand4	=[	$hover4,		wiz_stand5	] {ai_stand();};
void()	wiz_stand5	=[	$hover5,		wiz_stand6	] {ai_stand();};
void()	wiz_stand6	=[	$hover6,		wiz_stand7	] {ai_stand();};
void()	wiz_stand7	=[	$hover7,		wiz_stand8	] {ai_stand();};
void()	wiz_stand8	=[	$hover8,		wiz_stand1	] {ai_stand();};

void()	wiz_walk1	=[	$hover1,		wiz_walk2	] {ai_walk(8);
Wiz_idlesound();};
void()	wiz_walk2	=[	$hover2,		wiz_walk3	] {ai_walk(8);};
void()	wiz_walk3	=[	$hover3,		wiz_walk4	] {ai_walk(8);};
void()	wiz_walk4	=[	$hover4,		wiz_walk5	] {ai_walk(8);};
void()	wiz_walk5	=[	$hover5,		wiz_walk6	] {ai_walk(8);};
void()	wiz_walk6	=[	$hover6,		wiz_walk7	] {ai_walk(8);};
void()	wiz_walk7	=[	$hover7,		wiz_walk8	] {ai_walk(8);};
void()	wiz_walk8	=[	$hover8,		wiz_walk1	] {ai_walk(8);};

void()	wiz_side1	=[	$hover1,		wiz_side2	] {ai_run(8);
Wiz_idlesound();};
void()	wiz_side2	=[	$hover2,		wiz_side3	] {ai_run(8);};
void()	wiz_side3	=[	$hover3,		wiz_side4	] {ai_run(8);};
void()	wiz_side4	=[	$hover4,		wiz_side5	] {ai_run(8);};
void()	wiz_side5	=[	$hover5,		wiz_side6	] {ai_run(8);};
void()	wiz_side6	=[	$hover6,		wiz_side7	] {ai_run(8);};
void()	wiz_side7	=[	$hover7,		wiz_side8	] {ai_run(8);};
void()	wiz_side8	=[	$hover8,		wiz_side1	] {ai_run(8);};

void()	wiz_run1	=[	$fly1,		wiz_run2	] {ai_run(16);
Wiz_idlesound();
};
void()	wiz_run2	=[	$fly2,		wiz_run3	] {ai_run(16);};
void()	wiz_run3	=[	$fly3,		wiz_run4	] {ai_run(16);};
void()	wiz_run4	=[	$fly4,		wiz_run5	] {ai_run(16);};
void()	wiz_run5	=[	$fly5,		wiz_run6	] {ai_run(16);};
void()	wiz_run6	=[	$fly6,		wiz_run7	] {ai_run(16);};
void()	wiz_run7	=[	$fly7,		wiz_run8	] {ai_run(16);};
void()	wiz_run8	=[	$fly8,		wiz_run9	] {ai_run(16);};
void()	wiz_run9	=[	$fly9,		wiz_run10	] {ai_run(16);};
void()	wiz_run10	=[	$fly10,		wiz_run11	] {ai_run(16);};
void()	wiz_run11	=[	$fly11,		wiz_run12	] {ai_run(16);};
void()	wiz_run12	=[	$fly12,		wiz_run13	] {ai_run(16);};
void()	wiz_run13	=[	$fly13,		wiz_run14	] {ai_run(16);};
void()	wiz_run14	=[	$fly14,		wiz_run1	] {ai_run(16);};
```

```
void()	wiz_fast1	=[	$magatt1,		wiz_fast2	] {ai_face();Wiz_StartFast();};
void()	wiz_fast2	=[	$magatt2,		wiz_fast3	] {ai_face();};
void()	wiz_fast3	=[	$magatt3,		wiz_fast4	] {ai_face();};
void()	wiz_fast4	=[	$magatt4,		wiz_fast5	] {ai_face();};
void()	wiz_fast5	=[	$magatt5,		wiz_fast6	] {ai_face();};
void()	wiz_fast6	=[	$magatt6,		wiz_fast7	] {ai_face();};
void()	wiz_fast7	=[	$magatt5,		wiz_fast8	] {ai_face();};
void()	wiz_fast8	=[	$magatt4,		wiz_fast9	] {ai_face();};
void()	wiz_fast9	=[	$magatt3,		wiz_fast10	] {ai_face();};
void()	wiz_fast10	=[	$magatt2,		wiz_run1	]
{ai_face();SUB_AttackFinished(2);WizardAttackFinished ();};

void()	wiz_pain1	=[	$pain1,		wiz_pain2	] {};
void()	wiz_pain2	=[	$pain2,		wiz_pain3	] {};
void()	wiz_pain3	=[	$pain3,		wiz_pain4	] {};
void()	wiz_pain4	=[	$pain4,		wiz_run1	] {};

void()	wiz_death1	=[	$death1,		wiz_death2	] {

self.velocity_x = -200 + 400*random();
self.velocity_y = -200 + 400*random();
self.velocity_z = 100 + 100*random();
self.flags = self.flags - (self.flags & FL_ONGROUND);
sound (self, CHAN_VOICE, "wizard/wdeath.wav", 1, ATTN_NORM);
};
void()	wiz_death2	=[	$death2,		wiz_death3	] {};
void()	wiz_death3	=[	$death3,		wiz_death4	]{self.solid = SOLID_NOT;};
void()	wiz_death4	=[	$death4,		wiz_death5	] {};
void()	wiz_death5	=[	$death5,		wiz_death6	] {};
void()	wiz_death6	=[	$death6,		wiz_death7	] {};
void()	wiz_death7	=[	$death7,		wiz_death8	] {};
void()	wiz_death8	=[	$death8,		wiz_death8	] {};

void() wiz_die =
{
// check for gib
	if (self.health < -40)
	{
		sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NORM);
		ThrowHead ("progs/h_wizard.mdl", self.health);
		ThrowGib ("progs/gib2.mdl", self.health);
		ThrowGib ("progs/gib2.mdl", self.health);
		ThrowGib ("progs/gib2.mdl", self.health);
		return;
	}

	wiz_death1 ();
};


void(entity attacker, float damage) Wiz_Pain =
{
	sound (self, CHAN_VOICE, "wizard/wpain.wav", 1, ATTN_NORM);
	if (random()*70 > damage)
		return;		// didn't flinch

	wiz_pain1 ();
};


void() Wiz_Missile =
{
```

```
	wiz_fast1();
};

/*QUAKED monster_wizard (1 0 0) (-16 -16 -24) (16 16 40) Ambush
*/
void() monster_wizard =
{
	if (deathmatch)
	{
		remove(self);
		return;
	}
	precache_model ("progs/wizard.mdl");
	precache_model ("progs/h_wizard.mdl");
	precache_model ("progs/w_spike.mdl");

	precache_sound ("wizard/hit.wav");		// used by c code
	precache_sound ("wizard/wattack.wav");
	precache_sound ("wizard/wdeath.wav");
	precache_sound ("wizard/widle1.wav");
	precache_sound ("wizard/widle2.wav");
	precache_sound ("wizard/wpain.wav");
	precache_sound ("wizard/wsight.wav");

	self.solid = SOLID_SLIDEBOX;
	self.movetype = MOVETYPE_STEP;

	setmodel (self, "progs/wizard.mdl");

	setsize (self, '-16 -16 -24', '16 16 40');
	self.health = 80;

	self.th_stand = wiz_stand1;
	self.th_walk = wiz_walk1;
	self.th_run = wiz_run1;
	self.th_missile = Wiz_Missile;
	self.th_pain = Wiz_Pain;
	self.th_die = wiz_die;

	flymonster_start ();
};
```