Hiding in Shadows

by Anders [Wolf] Jenbo (NoBody) with Loki. Modified by CRxTRDude.
Original site: http://collective.valve-erc.com/index.php?doc=1092932859-33776100

Tutorial type: Intermediate - A/F, FGD

In this article I will explain how to make a monster disregard the player if they're within a certain level of light, and how you can define this level of light from within the map.

Reading the Map Properties

First we need to get the data from the map properties and store it in a variable. Open dlls\world.cpp and add the bold code just before the CWorld :: KeyValue, around line line 674.

```
int g_darklevel;

//
// Just to ignore the "wad" field.
//
void CWorld :: KeyValue( KeyValueData *pkvd )
```

Then, just before the last else, add the bold code:

```
else if ( FStrEq(pkvd->szKeyName, "defaultteam") )
{
  if ( atoi(pkvd->szValue) )
  {
    pev->spawnflags |= SF_WORLD_FORCETEAM;
  }
  pkvd->fHandled = TRUE;
}
else if ( FStrEq(pkvd->szKeyName, "darklevel") )
{
  g_darklevel = atoi(pkvd->szValue);
  pkvd->fHandled = TRUE;
}
else CBaseEntity::KeyValue( pkvd );}
```

Now we will need to declare the variable as a global, as we will need to read it from another part of the DLL for the next part. We could make our own .h, but it's easier to just use cbase.h as it's already included where we need the variable. Open dlls\cbase.h and add the bold code in the very beginning:

```
extern int g_darklevel;
```

The Actual Check

Open dlls\player.cpp, and as the first thing in the CBasePlayer::Classify, around line 1603, add the bold code:

```
//
// ID's player as such.
//
int CBasePlayer::Classify ( void )
{
if (Illumination() <= g_darklevel)
return CLASS_NONE;

return CLASS_PLAYER;
}
```

This just compares the variable to the current illumination. If this checks out, it returns CLASS_NONE which monsters ignore; if the illumination is higher, it returns CLASS_PLAYER and monsters will attack. The illumination is a function that is already in the player.cpp. It simply takes the brightness of the light map under the player and adds a virtual muzzle flash light value to it (if the player fires a gun).

The brightness of the muzzle flashes can be changed accordingly in dlls\weapons.h (Note that this doesn't change the actual muzzle flash). Also note that any values above 255 will simply add to the time of the effect.

```
#define BRIGHT_GUN_FLASH 512
#define NORMAL_GUN_FLASH 256
#define DIM_GUN_FLASH 128
```

Muzzle flash definitions for various weapons:

```
mp5grenade      BRIGHT_GUN_FLASH
python          BRIGHT_GUN_FLASH
rpg             BRIGHT_GUN_FLASH
mp5             NORMAL_GUN_FLASH

shotgun         NORMAL_GUN_FLASH
glock           NORMAL_GUN_FLASH

hornetgun       DIM_GUN_FLASH
```

The Gauss and Egon should also have an effect on the illumination, correct? Note that we can add any of the 3 muzzle flash definitions to the weapons, but for this article we will be using BRIGHT_GUN_FLASH. Now for the Egon, open dlls\egon.cpp and go to line 289. Add the bold code

after case FIRE_WIDE: but outside the #ifndef CLIENT_DLL, as the light check happens on the server side.

```
case FIRE_WIDE:

m_pPlayer->m_iWeaponFlash = BRIGHT_GUN_FLASH;

#ifndef CLIENT_DLL
```

Then for the Gauss, open dlls\gauss.cpp and add the muzzle flash next to the fire animation, like so.

```
// player "shoot" animation
m_pPlayer->SetAnimation( PLAYER_ATTACK1 );

m_pPlayer->m_iWeaponFlash = BRIGHT_GUN_FLASH;

}
```

Say, for example, we also need it to illuminate when the Gauss overcharges. Here we want the player to glow for the same amount of time that the effect lasts, so we use a fixed value for the muzzle flash. Add the bold code where that happens.

```
if ( m_pPlayer->m_flStartCharge < gpGlobals->time - 10 )
{
// Player charged up too long. Zap him.

m_pPlayer->m_iWeaponFlash = 768;
```

However, the illumination doesn't include dynamic lights like the flashlight, so we will have to add this ourselves. Back in dlls\player.cpp find CBasePlayer :: Illumination, which should be around line 4130. Add the bold code:

```
int CBasePlayer :: Illumination( void )
{
int iIllum = CBaseEntity::Illumination( );

if (!FlashlightIsOn() == false)
iIllum += 200;

iIllum += m_iWeaponFlash;
if (iIllum > 255)
return 255;

return iIllum;
```

You could also add other conditions (such as crouching) in this way.

Developer Tools

Assuming you will want to debug maps that use this feature, you would probably want to add a developer command to test illumination levels around the map. To do this, we will have to add a CVAR. This is basically done the same way we added the global variable. Open dlls\game.cpp and add the bold code right after the #include's:

```
#include "extdll.h"
#include "eiface.h"
#include "util.h"
#include "game.h"

cvar_t devlight = { "dev_light", "0" };
```

Then move down to the GameDLLInit and add the bold code

```
void GameDLLInit( void )
{
// Register cvars here:

CVAR_REGISTER (&devlight);
```

Now open dlls\game.h and right after extern void GameDLLInit() add the bold code.

```
extern void GameDLLInit( void );

extern cvar_t devlight;

extern cvar_t displaysoundlist;
```

Then open player.cpp and find PreThink, around line line 1769. Add the bold code to the beginning of the function:

```
void CBasePlayer::PreThink(void)
{

if (devlight.value != 0)
ALERT ( at_console, "You are standing in %d light!\n",Illumination() );

int buttonsChanged = (m_afButtonLast ^ pev->button); // These buttons have
changed this frame
```

I use PreThink because it updates every frame, so we will get the light value real-time. To activate the light tester in-game, you have to be in developer mode, which you can activate by bringing down the console and type dev_light 1 [1].

The FGD

Finally, we need to edit the FGD to include this new option. Open a Half-Life FGD in Notepad, and search for worldspawn, then add the bold code in between the other options. Make sure the proper game configuration and FGD are selected when loading Valve Hammer Editor.

```
@SolidClass = worldspawn : "World entity"
[
message(string) : "Map Description / Title"
skyname(string) : "environment map (cl_skyname)"
darklevel(integer) : "Hide shadow 0-255 (0=black)" : 15"
sounds(integer) : "CD track to play" : 1
```

Then, to set the needed light level in a map within Valve Hammer Editor, go to Map Properties and edit the value in the Hide shadow field. («Map Properties» essentially accesses the worldspawn entity, in case you haven't noticed.)

Lastly, I would like to thank Teh_Freak for helping me with some of the functions.

Loki's tutorial - CLASS_SHADOW

In response to mathews post:

«good tut but i would adapt it to use the cbasemonster a.i for allowing me to controll what monsters can't see you»

... What I thought was instead of changing the player class to CLASS_NONE, you could create a class that affects different monsters to see you. for example:

in cbase.h create a new line something like this...

#define CLASS_PLAYER_BIOWEAPON 12 *hornets and snarks ... #define CLASS_ALIEN_BIOWEAPON 13* hornets and snarks ... #define CLASS_SHADOWS 14 *Only certain monsters can see you #define CLASS_BARNACLE 99 Now open up monsters.cpp and find the monster relationship table, I think it's called IRelationship, inside you'll find all the classes and how they react to one another... Underneath you will need to add another line something like this... <code cpp> /*SHADOWS*/ { R_NO, R_NO, R_NO, R_NO... etc </code> Now you need to create another column in the table to show how other monsters react to you. Because I believe that some of the alien creatures have something akin to night vision, mine goes something like this.. <code cpp>* SHAD R_NO R_DL R_AL R_NO R_HT R_NO R_NO R_NO R_DL R_FR R_AL R_DL R_DL </code>

Then, where you're told to put CLASS_NONE at the top of the tutorial, just put CLASS_SHADOWS

Hope that helps! :)

Issues:

- There is a bug in which this code is not initialized pre-game and needs restarting the game in console to work. I'll still need to consult with people to deal with this.