MULTIPLAYER MAP CREATION

BY RAVEN SOFTWARE

1.0 Introduction

The purpose of this document is to address common issues when making maps for SoFII multiplayer, or MP. This document assumes that you are familiar with both shaders and Quake-engine map editing. For more detailed info on shaders please refer to “Quake III Arena Shader Manual.”

2.0 Basic Multiplayer Design Issues

2.1 Setting up to create a SOF2 map

The tools used to create MP maps are SOF2Radiant with the sof2mp.qe4 project file. Once the editor is set up, load the sof2mp project file and you are ready to go. In case you are not familiar with Radiant, the .QE4 file defines the specifics for a project, including entities and textures paths. There is a separate sof2.qe4 file for Single Player.

Once SOF2Radiant is installed and loaded with the .QE4 file (**File – Load Project**) change the preferences (**P**) to your liking.

2.2 Making a gametype work

To make a map capable of supporting a particular game type, three things must be included: spawn entities, mission items and gametype triggers.

2.2.1 Player Spawns

There are two types of player spawns: **info_player_deathmatch** and **gametype_player**. The latter entity is a unique SoFII entity that is absolutely required for elimination, infiltration and any custom created game type. Although key/values are not required for spawn points, you can set values to determine how a particular spawn point is used.

The **angles** value determines the starting facing of a player upon spawning in. The value is in y,z,x format, or pitch, yaw, roll. Commonly, only yaw is set on player spawns, so values such as 0 135 0 or 0 227 0 are fine. **Angles** is valid for both types of spawn points.

The **gametype** value determines what game a particular spawn point is valid in. A map can be designed for more than one game type, and it is common to want different spawn areas for those games. Valid values for **gametype** are:

dm (for deathmatch)
tdm (for team deathmatch)
ctf (for capture the flag)
elim (for elimination)
inf (for infiltration)

Multiple games can also use the same spawn point. If you wanted a spawn point to be used for both team deathmatch and elimination, **gametype** would have a value of 'tdm elim'.

Note that if a **gametype_player** is not given a **gametype** value, it will be used by all game modes. In essence, not listing the gametypes for a player spawn is the same as **gametype** 'dm tdm ctf elim inf' Also a **gametype_player** entity with **gametype** 'dm' is equivalent to an **info_player_deathmatch**.

(Example 1)
In Radiant, open the map file called 'examples'. In this map you will see the two above-mentioned player spawn entities near the number 1.

Select one of the spawn point entities, and then press (**N**) to bring up the Entity Console. Notice that these spawn points have the **angles** field set with the previously mentioned values of pitch yaw roll. Again, the middle number is the only number needed a majority of the time, and represents the direction that the player will face when they spawn in.

To add the **angles** key/value to your own spawn points you can either type both the key and value in or you can press (**CTRL+Left-Click**) on the angle buttons at the bottom of the Entity Console.

Important note: You must hold down (**CTRL**) when you use the angle buttons, otherwise the key/value put on the entity will be **angle** instead of **angles**. **Angle** is not a valid value in SoFII and will be ignored.

2.2.2 Mission Items

If a map is going to support mission objectives such as those found in Capture The Flag (ctf) or Infiltration (inf), that map needs mission items. A mission item in Radiant is called **gametype_item**. A **gametype_item** needs two values to work: a **targetname** and a **gametype**.

The **targetname** of the **gametype_item** should be whatever is required from the .gametype file. .Gametype files reside in the mp.pk3 or base/mp/scripts if the pak file is extracted and can be read in any text editor. For example, in the 'inf.gametype' file it states the targetname for the briefcase needs to be 'briefcase'. If the **gametype_item** doesn't have the necessary targetname, the game mode won't work. An example where to find the targetname in a .gametype file follows. The .gametype file is explained more in depth in another document. Blue text is not in the file:

```
items
{
        briefcase                       the targetname of the mission item
        {
                displayname             "Briefcase"
                team                    "blue"
                enabled                 0

                icon                    "gfx/menus/hud/briefcase_small"
                model                   "models/pick_ups/briefcase_silver.md3"
                boltmodel               "models/characters/bolt_ons/briefcase_back.glm"
        }
```

}

The **gametype** value for the **gametype_item** works the same way that it does in the **gametype_player**, expect that there should only be one value. The value for the mission item's **gametype** should be whatever the item is meant for.

Additonal Note: The **gametype** key/value can be put on ANY entity to tell the game when to render that entity. For example, the helicopter in The Shop has the **gametype** key/value of 'elim inf', so that it only shows up for the elimination and infiltration game modes. Of course, this will only work on entities, not world architecture.

(Example 2)
Open the 'examples' map and look at how the **gametype_item** is setup near the number 2. Select the **gametype_item** and look at the key/values in the Entity Console (**N**).

To check your **gametype_item** in the game, type in 'sv_pure 0' at the console. This will allow you to load a custom map. Next, type 'map' to load the map you just created. For example 'map examples' will load the above map if it compiled. Again, this document assumes you are familiar with basic map creation.

Important note: At this point you will notice that the briefcase is not there! Don't worry; this happens when you load a map in the wrong gametype. The default gametype is 'dm'. To see the briefcase, change the gametype with 'g_gametype' in the console and then reload the map. If using this 'examples' map, changing the gametype from dm to inf will allow you to see the briefcase.

2.2.3 gametype_trigger

The **gametype_trigger** is necessary so the game knows when a mission item has been captured. The **gametype_trigger** needs the **targetname** and **gametype** key/value to work correctly. To get the correct **targetname**, you must again look in the appropriate .gametype file.

In the 'ctf.gametype' file for example, you find the targetnames of "red_capture_point" and "blue_capture_point". A trigger is activated when the require mission item (like a flag) touches the trigger. Where the trigger is actually placed depends on the gametype. For a CTF game, the trigger should be placed around the flag. So, the 'blue_capture_point' trigger will go around the blue flag. For Infiltration however, the trigger will be placed where the briefcase needs to be brought to, not around the briefcase.

In order for the trigger to only be activated in the gametype it was intended for, it needs a **gametype** value. So for the above example, the **gametype_trigger** would need **gametype** with a value of 'ctf'.

(Example 3)
Open the 'examples' map and look at how the **gametype_trigger** is setup near the number 3. Select the **gametype_trigger** and look at the key/values in the Entity Console (**N**).

Again, both the **gametype_trigger** and the **gametype_item** near the number 3 will not be present unless g_gametype is set to 'ctf'.

2.3 Optional entities

2.3.1 Effects

There are two different ways to call effects, **target_effect** and **fx_play_effect**.

The **target_effect** entity is used when you want an effect to fire every time the entity is activated. A **target_effect** is activated via a trigger and thus, needs a **targetname**. Two other necessary key/values are **effect** and **delay**. The value of the **effect** key determines what effect is played. For example, to have a red flare shoot off on capturing a red CTF flag, there is an effect named 'flare_red.efx'. Effects are stored in base/effects. The **delay** value determines the amount of time in milliseconds that pass before the effect is played.

(Example 4)
Open the 'examples' map. Select the target_effect near the number 4. Look at the entity in the Entity Console (**N**) to see the above-mentioned keys/values. The effect for the red flag will have **effect** value of 'flare_red.efx' and **targetname** value of 'redflare'. Also note that normally you would want to put the flare at the bottom of the flag, but for the purposes of this document the entities were moved away to make it easier to find them. Effects play from the origin of the entity.

An **fx_play_effect** entity can be use to play an effect continuously. For example, a flame in a barrel is just a looping effect. For the entity, the necessary values are **effect** and **count**. A **count** of –1 will cause the effect to loop forever. Optional values are **wait**, **target**, and **random**. **Wait** determines the time between triggerings. For a looping effect, the **wait** should be very small. **Target** allows for aiming certain effects like steam and smoke. The **fx_play_effect** should target an **info_notnull**. **Random** allows for variance in the triggering.

Important note: Effects are expensive to render and system resource intensive, so use them sparingly.

(Example 5)
Open the 'examples' map and select the **fx_play_effect** hidden inside the barrel near number 5. Look at the entity in the Entity Console (**N**) to see the above-mentioned keys/values. If you are having trouble selecting the entity because other entities are in the way, use the (**H**) key on selected brushes to hide them. Then use (**SHIFT+H**) to make all hidden objects reappear. Another method for selecting brushes or entities that are blocked is to use (**ALT+SHIFT+Left-Click**). This method will cycle through all brushes and entities under your mouse pointer.

2.3.2 Breakable glass

Breakable glass in multiplayer differs from that in single player, but is easy to implement. Simply create a brush and turn it into **func_glass**. There are two ways to turn a brush into an entity. One is to (**right-click**) while the brush is selected, then choose the

desired entity. The other way is to bring up the entity console and (**double-click**) on the desired entity.

Although you can use any shader for **func_glass** you generally should use a glass shader. The reason is that the effect created when the func_glass is broken is a glass sound with glass shards.

2.4 Missing entities

Only glass is breakable in SOF2 MP. Most breakable objects were never enabled for MP due to networking and performance reasons. Therefore, there are no **func_breakable_brush** entities, as there are in single player.

In comparing the project files between the single player and multiplayer game, many other entities simply do not exist in multiplayer. The reasons are varied, but the main reason is that networking and performance in multiplayer meet different criteria.

3.0 Other Design Criteria

3.1 Why do bullets and grenades sometimes not penetrate railings and other objects?

As with other Q3 based games, shaders are a crucial part of making maps. Shaders dictate how a brush appears and interacts with the world. For example, a shader can make a brush solid or not, or transparent or not. There are a lot of things that can be done by using the right shaders and there are some tricks a mapper should be aware of.

For example, even if a brush has a railing shader on it that says it is non-solid, that doesn't mean projectiles will pass through it. You have to make sure that all faces have shaders that are non-solid. For the faces that can't be seen put the shader 'tools/_nodraw' on them.

Important note: Never mark a shader as 'noimpact' without understanding that any projectile that makes contact with that face will disappear!

(Example 6a)
Open the 'examples' map and look at 6a. This railing cannot be shot through. This railing is a brush, but instead of using 'nodraw', the 'caulk' shader was used. When you shoot this railing you will notice two things: bullets are stopped and grenades disappear.

The reason the bullets are stopped is because the solid property of the caulk shader overrides the non-solid properties of the railing shader.
The reason grenades disappear is because the shader definition has 'surfaceparm noimpact' in it and this shouldn't be used on any shader, except skies.

(Example 6b)
This railing is also a brush but with 'nodraw' on all faces except one. This gives the illusion that the railing is there, but still allows projectile to go through.

(Example 6c)
While the frame of the railing is made of regular brushes, the part that actually has the railing texture is a patch mesh. Generally it is best to use patch meshes when you can since they use LOD (Level Of Detail). LOD objects render cheaper (with less detail) the further away you are. If you do use a patch mesh there are two important things you have to keep in mind.

A. Patch meshes only have collision detection on one side. Even if a shader is marked solid, only the outward faces will have collision. In other words bullets, grenades and even players can travel through the *inside* faces of a curve. When using patch meshes, it is best to put clip brushes around the patch mesh.

   Important note: A clip brush is a brush with a clip shader on it. A clip shader turns the brush invisible and non-solid to everything except players. There are also shot brushes, which act the same as clip brushes, but block only projectiles.

B. If the curve is going to be viewed on both the inside and outside, the shader must be two-sided. The reason for this was explained above, because only the outside faces are drawn. To make a shader two-sided, include the line 'cull disable' in the shader definition.

(Example 6d)
This canvas brush will block grenades, but not bullets. The reason for this is the surfaceparm 'slime'.

Important note: When the slime parameter is used the brush will turn non-solid, so make sure to have a clip brush around the brush so that players can't walk through it.

3.2 How to create a map with terrain

See the sof2_terrain.doc for details.

3.3 Valid model formats for maps

The two models types that can be used in MP maps are **misc_model** and **model_static**. Both of these models require .md3 format.

3.3.1 Misc_model

Misc_models become part of the world architecture and have advantages and drawbacks. The advantages are that they are not entities; so do not add to network traffic. The disadvantages are that they add to the BSP file size and they require separate shaders.

Important note: Not all models have shaders for misc_models. If a shader does not exist for a **misc_model**, you need to do three things:

A. Copy the original shaders for the model. If you cannot figure out what shaders a model uses, open the model in md3View and do the following:

   Go to 'View' in the menu bar
   Inside the view menu option drag down until you reach 'View Load-Models Info'

B. Add '_bsp' to the end of the new shader name. For example if we had a model called lamp.md3 and if this model uses the following shaders 'lamp_base' and 'lamp_shade'. The new misc_model shader would need to be 'lamp_base_bsp and 'lamp_shade_bsp'.

C. Make sure that in the shader definition that the 'rgbGen' is set to 'vertex'

   Important note: If the model is only used in the RMG then set 'rgbGen' to 'lightingDiffuse'. If the model is used in the RMG and regular MP maps then, yet another special shader needs to be created. Look to the RMG document for details.

3.3.2 Model_static

Model_statics are entities and count towards the entity limit. It is best to not use these models if possible. If you do use **model_static** and the model is missing a shader, then create a shader for it like you would normally for a model. The process is basically the same as for other shaders.

3.3.3 Models do not have physics

The only models that have physics in SoFII are ConfusEd models and they are only used in SP. This means that bullets, grenades, players or anything dynamic will pass through the model. To make a model appear to have bounding, use clip brushes where necessary. Also, using shot clip brushes will allow for the illusion of interaction with the model. When adding shot clips, make them approximately the same size as the model so that the shots appear to be hitting the model, not the air.

(Example 7)
The 'examples' map contains a **misc_model** and **model_static** near the number 7. The model on the left is the **misc_model** and the one on the right is the **model_static**. Both have a clip brush and shot clip brush, although in this case the models didn't have to have a clip brush.

3.4 Sounds

Place a **target_speaker** entity in the world. Either the **noise** or **soundSet** key/values are valid. The **noise** value needs to be the path to an actual sound file while the **soundSet** is defined in the sounds.txt file located in base/sound.

3.5 Dynamic Lights

A dynamic light is a light that changes while in the game. This can be as simple as turning a light on or off, or having a light that flashes.

In order to turn a light on or off it simply needs to be targeted and therefore requires a **targetname**.

If the **style** key is used, the light will flicker. The frequency of the flicker depends on what value is given. The values 1 through 12 are already defined with some basic styles.

3.6 Map Location

It is useful to know where a team is when they speak. This can be done via a **target_location** in the map. The way this works is that when a player speaks, the message of the closest **target_location** will be added. The message that is displayed is the value of the **message** key on the **target_location**.

(Example 8)
The 'examples' map contains a **target_location** near the number 8. It has a **message** of 'Test Room'. Thus, if a player uses a voice command near this **target_location**, 'Test Room' will be added to his message.

3.7 Music

To add music to a map, add the **music** value to the worldspawn.

Important note: The worldspawn is the classname for any brush that is not an entity or a **func_group**.

(Example)
Open the 'examples' map and select any wall. Press (**N**) to bring up the Entity Console and look at the value for **music**. It has a path of music/x.

3.8 Valid teams for a map

Because of the diversity in the look of maps, the skins allowed on a particular map can be defined. Note this is only for team games; plain deathmatch can have any skins. To define what the teams on a map look like use the **redteam** and **blueteam** key/values in the worldspawn. The available values for teams are:

"colombian"
"hospital"
"thug"
"marine"
"snow"
"terrorist"
"suits"

4.0 Getting your map in the game

4.1 To have a map show up in the menus

A .ARENA file must be created for a map to show up in the menus. For example, if a map is named ‘foo’, there needs to be a ‘foo.arena’ file created. This file declares which gametypes are valid for the map, which helps the menu system know where to display the map.

4.2 The syntax of .ARENA

As stated above, a .ARENA is required by the menu system to display the map. The syntax is as follows. Blue text is not in the file.

```
{
map             "mp_shop"               the name of the .bsp file
longname        "The Shop Under Siege"  the title of the map (for loading screens, menu, etc.)
type            "dm tdm elim inf"       what gametypes are valid for this map
}
```