TUTORIAL 5.1 – Scripting a light

Now that we know how to creates models, levels, materials and particle effects lets try and learn how to make some interesting things happen in a level using some scripts.

For this tutorial you can use any level you want as long as you know it's working and has a functioning script file. If you created the level in tutorial 2 you can use that and be sure to have a good foundation for this tutorial.

So lets start with having a simple empty room with a single light in the center, rename the light to ”light”. Make sure you have a _start_ location in one corner for the player to start at. Then create a new cube(0.5m*0.5m*0.5m) in another corner of the room, rename the node to ”_area_script_lighttrigger”.

Now import the referens of your wooden box and place it as you like as long as it's inside the room. If you used the tutorial naming conventions the referens should be named ”_ref_mystuff_woodbox_woodbox1”, if its not make sure you have the reference named “_ref_FILENAME_woodbox1”.

What we will do in this first tutorial is to make it so that the light turns red when you push the box into the corner with the “lighttrigger” area. And when you drag the box out of the corner the light will go back to white.

Export your level and save it as “mystuff_scriptlevel.dae” in your “mystuff” directory.

Now copy the “tut_scriptbase.hps” file from “tutorials/tutorial5” directory to your “mystuff“ directory, rename it “mystuff_scriptlevel.hps”.

Before we continue and add our scripts lets make sure we have everything we need and that it's named correctly. Check the following.

1. mystuff_scriptlevel.dae
    - One light named “light”
    - One wood box entity named “woodbox1”
    - One area named “lighttrigger”
2. mystuff_scriptlevel.hps

Test and load the level, make sure you start at the _start location, that there is a light and that you see the wood box in the room.

Now let's open the script file “mystuff_scriptlevel.hps”. As you can see I have written some explanations in it for you. When commenting in a script you use // and /* */. If that was news to you, you really should read some basic C/C++ syntax tutorials first, you do not need to know much but the first couple of chapters in an introduction to C/C++ book is very useful.

The first thing we are going to do is to create the “callback” for when the box enters the “lighttrigger” area. To do this add the following to “OnStart” making it look like this, minus the comment already in the file:

```
void OnStart()
```

```
{
    //---AREA CALLBACKS----
    AddEntityCollideCallback("Enter","ENTITY","AREA_NAME","CALLBACK_NAME");
}
```

AddEntityCollideCallback, this creats a callback for when an entity collides with an area. You specify excatly what in the following strings. ”Enter” this can also be ”During” and ”Leave”, for now it should be ”Enter”. ”ENTITY” this says what entity it is, in our case ”woodbox1” since it's when the wood box enters the area we want something to happen. If we, for example, write ”Player” here the player will be the trigger for the area instead. ”AREA_NAME” this we change to ”lighttrigger”, because that's the name of our area to check for collision with an entity. ”CALLBACK_NAME” This will be the name of the actual callback, the funktion that will contain our code for turning the lights red. Let's name it ”TurnLightsRed”.

What will happen is that when the game loads the level it will create a callback that will activate when ”woodbox1” enters the ”lighttrigger” area. We now need to create the callback that will contain the actual events that will occur. Enter the following in the ”CALLBACK FUNCTIONS” area:

```
//////////////////////
/*CALLBACK FUNCTIONS*/

void CALLBACK_NAME(string asParent, string asChild)
{
    FadeLight3D("LIGHT_NAME", R,G,B,A, Radius,Time);
}
```

CALLBACK_NAME now this should be named ”TurnLightsRed” and in the FadeLight3D we should change LIGHT_NAME to ”light”.

Following there are 6 settings, first the RGB is used to set the color, A is alpha and used to control the amount of specular, Radius is the size of the light and Time is how long it should take to fade.

Change it to

```
FadeLight3D("light", 1,,,1, 6,3);
```

This will give a red light, with full specular, a radius of 6 and it will take 3 seconds for the light to fade from white to these settings. Save the script file and lets test it ingame. First you need to edit your ”settings.cfg” file in the redist folder. It should load the Map file ”mystuff_scriptlevel.dae” and use the start location you have entered, if you used the tutorial level it should be “location1”.

Start penumbra, move the box from it's start postion to the corner where the “triggerlight” area is. When you do this the lights should fade to red.

Now exit Penumbra and lets continue on and make it so that the lights fade back ones you pull the box out of the area again.

Copy and paste the AddEntityCollideCallback so that you get this in your “OnStart” area:

```
void OnStart()
{
    //---AREA CALLBACKS----
AddEntityCollideCallback("Enter","woodbox1","lighttrigger","TurnLightsRed");
AddEntityCollideCallback("Enter","woodbox1","lighttrigger","TurnLightsRed");
}
```

Change the second AddEntityCollideCallback so that it's “Leave” instead of “Enter” and “TurnLightsBack” instead of “TurnLightsRed”. You now have another callback called TurnLightsBack that will be triggered ones the wood box leaves the script area.

To create the callback make a copy of the other callback so that you get this:

```
//////////////////////
/*CALLBACK FUNCTIONS*/

void TurnLightsRed(string asParent, string asChild)
{
    FadeLight3D("light", 1,,,1, 6,3);
}

void TurnLightsRed(string asParent, string asChild)
{
    FadeLight3D("light", 1,,,1, 6,3);
}
```

The second one should be called ”TurnLightsBack” and the light should still be called ”light” because it's the same light we want to fade back to white. Using these settings:

```
FadeLight3D("light", 1,1,1,1,8,1.5);
```

you will get a total white light with full specular, a radius of 8 and a fade time of 1.5 seconds. You can tweak this a little if you want another radius or slightly different color.

Save the file, launch Penumbra and test the level. Can you make the lights fade back and forth by moving the box in and out of the area?

TUTORIAL 5.2 – Scripting several things

Lets move on and try and make this level into a puzzle. What we are going to do is to make it so that the ”gravity” is reversed for the wood box, making the box float up to the roof. We will also add a door with a key pad and the key pad will only work if the box is in the ”lighttrigger” area in the corner. Since the box is up in the roof it does not reach down into the corner therefor we need to make something that will make the ”gravity” right again.

To turn of the reversed ”gravity” we will use a simple switch that when pulled down will turn the gravity back to normal. This will allow theplayer to position the box in the area.

Finally to open the door you will need a code for the key pad. The player will get the code from an audio file that starts to play as soon as he combines two key parts found in the room.

Yes this is not logical at all, but hey it's a tutorial.

Let's start with the gravity! In the script file, at the bottom you have the ”void OnUpdate()” area, where you can add things that should be continousely running. We will use this area to add a force to our woodbox, making it float in the air.

The force command is this:

```
AddBodyForce("woodbox1", "world", , 500, );
```

”woodbox1” this is the name of the entity as you know, ”world” this is if the XYZ is based on the worlds coordinates or with ”local” the objects cooridnates. The difference is that if you set ”local”, for example, Y can be down instead of up if you turn the object upside down. Since we always want Y to be up we use ”world”. The last ”, 0, 500, 0” are the values of force applied to XYZ. Since we want the object to float we add force to the Y, in this case 500. Your ”OnUpdate” should look like this now:

```
void OnUpdate()
{
        AddBodyForce("woodbox1", "world", , 500, );
}
```

You can save the script and start the level, the box should float up to the roof as the level starts. Make sure you have not placed the box below the light! Because if you have, the level will turn dark when the box blocks the light.

Now we need to prepare to be able to control when and how this gravity can be turned of. To do this we will create a variable that: as long as it's 0 the Force will be in effect, but if the variable is anything else the Force will not be applied anymore.

Firts we create the variable in the section called ”OnStart”, it's here we added the callbacks in the first tutorial. Below the callbacks add:

```
    //---LOCAL VARIABLES----
    CreateLocalVar("float", );
```

This creates a local variable, you can also create Global variables if you want to use a variable over several different levels. The local variable we create is called ”float” and is assigned the value of 0.

To create the loop that checks this value to be true and therfor apply the force we go down to ”OnUpdate” again and we add a little ”if” statement for our ”AddBodyForce”.

```
    if (GetLocalVar("float") == )
    {
        AddBodyForce("woodbox1", "world", , 500, );
    }
    else
    {
        AddBodyForce("woodbox1", "world", , , );
    }
```

What you say here is that if the local variable named ”float” is 0 you will apply a force to Y. If it's anything else that is not 0, it will stop applying that force to Y.

If you save now and run the level everything will work just as before since you at the start create the local variable with the value of 0 it will be true and the force will be applied.

Now lets add a switch to the level that when pulled down will turn off the force. First go to your 3D editor and import the refernce file ”tut_ref_stick.dae” located in ”tutorials/tutorial_5”. Place the refernce in mid-air close to a wall, but not in a wall. As you can see I have allready named the referens correctly ”_ref_tut_stick_switch”. There is an entity file included with the tutorial, that this referense points to, called ”tut_stick.ent” that uses the ”tut_stick.dae” model. I have named this model ”switch”, this is the name that we will use in the script for the entity.

Before we continue make sure that you either 1. Add ”tutorials/tutorial_5” to your resources or that you 2. copy the ”tut_stick” files to your mystuff directory. Otherwise the game can't load them! When you have checked this, export the level overwriting your old one. If you now start Penumbra you should see the stick in the level.

The stick we have created has a joint in the model. I call the joint ”joint1”, it's this joint that we will use to know when the stick is pulled down to turn of the force.

To get the event we want we create another callback. Go up to ”OnStart” and below your other entries add:

```
    //---JOINT CALLBACKS----
    SetJointCallback("switch_joint1", "OnMin", "TurnGravityOn");
```

You have now created a joint callback for the entity called ”switch” and it's joint called ”joint1”. When it reaches the minimum value it will activate the callback called ”TurnGravityOn”. Other value you can have is ”OnMax” if you want something to activate on the joint when it reaches it's maximum value. The are other types of scripts that can be used to get exact values from a joint, but this one is the most simple and effective to use.

We now need to create the actual callback. Go up to ”CALLBACK FUNCTIONS” and below your light callbacks add this:

```
void TurnGravityOn(string asJoint)
{
    SetLocalVar("float", 1);
}
```

If you start Penumbra now the box will fall to the ground when you pull the switch down. Why? Becuase this callback does one simple thing, it changes the variable we created to have a value of 1 instead of 0. This makes the if statement in the ”OnUpdate” section false since the value is not 0 anymore. Since we say everything but zero should result in no force applied to Y that is what happens.

Let's prepare the door with the panel. Import the model ”tut_door.dae” to the level, notice that it's in a group so that you move the whole group. What is this you ask? It's a static mesh, a model grouped with it's colliders. The door is not going to be an entity, it's a static model that we only want in the game for the graphics. The actual scripts will be done by using an area.

Lets go ahead and create that area, create a box and resize it so that it covers the door and is slightly larger. Name the node ”_area_script_doorarea”. Export the level overwriting your old one. Make sure you move the 6 tut_door_x.jpg and material files to your ”mystuff” directory if you have not added the tutorial_5 folder to your resources. Otherwise the game will crash on launch because it can't find the material files and their corresponding textures.

Back to the script file, we wanted the door only to be active when the woodbox is in the "lighttrigger" area. By default nothing will happen if the player interacts with the door, or more specific the area that is around the door called "doorarea". To make something happen we need to create a new callback, BUT this time we will not create it in "OnLoad" rather we will go up to the "CALLBACK FUNCTIONS" and in the "TurnLightsRed" callback we add the following:

```
    AddEntityCallback("PlayerInteract","doorarea,"ActivateDoor");
```

This will make the "TurnLightsRed" callback look like this:

```
void TurnLightsRed(string asParent, string asChild)
{
    FadeLight3D("light", 1,,,1, 6,3);

    AddEntityCallback("PlayerInteract","doorarea","ActivateDoor");
}
```

What we have done is that a a new callback will be created when the wood box is in the "lighttrigger" area. This callback will be activated when the player interacts with the "doorarea" and it will be called "ActivateDoor".

Now we create the "ActivateDoor" callback by adding the following below the other callbacks:

```
void ActivateDoor(string asEntity)
{
    StartNumericalPanel("panel",1,2,3,4,10.0f,"PanelCode");
}
```

This callback will bring upp a code panel graphic on the screen. This panel has a code of "1234" and the callback to create different events depending on if the code is right or wrong is called "PanelCode". The "10.0f" is currently not used, originally it was planned to have doors "hackable" and this value is the time it takes to hack a door.

If you save and play the game now not much will happen, we need to create the "PanelCode" callback to tell the game what will happend depending on if the code is right or wrong.

Create a new callback below like this:

```
void PanelCode(string asName, bool abCodeWasCorrect)
{
    if(abCodeWasCorrect)
    {
        PlayGuiSound("door_code_correct", 0.8);
    }
    else
    {
        PlayGuiSound("door_code_incorrect", 0.8);
    }
}
```

This will play different sounds depeding on if the code is right or wrong. Start Penumbra and test it, enter a false code and then test to enter the right ”1234” code. Do you get the sounds?

Let's leave it like that for now. We should add the callback that removes the functionallity of the door when the wood box is not in the ”triggerlight” area anymore. In the callback ”TurnLightsBack” add:

```
    RemoveEntityCallback("PlayerInteract","doorarea");
```

This means that when the box leaves the ”lighttrigger” area it will also remove the callback we created when it enter the ”lighttrigger” area. We now have a ”loop” that will create and remove the callback for the door everytime the box moves in and out of the area. You can save and take it for a test run if you wish.

Now we need to add the two key parts to the level, make the script for combining them and by doing so playing an audio file that will tell the player the code for the door. Finally we will add some more stuff to the code for the door so that the game exits to the start menu when the code is correct.

First I have created the key mode, entity and references for you. They are big, ugly and odd but work. The referens is not the exakt model and things like that, this is simply to show that when doing tests you can quickly create stuff and not worry about being to exact. For what we want to do we only want some objects and it's their names and what we do with them in the script that's important, not the actual object.

Start by importing the references ”tut_ref_key1.dae” and ”tut_ref_key2.dae” to your level. Place them somewhere in the room so that they are not in the way for any of the other objects.

As you can see I have named the two objects ”keypart1” and ”keypart2”. If you export the level and overwrite your old one, you can start the level and pick the keys up, they will be in your inventory and look like the ones in Penumbra.

Going back to the script file, lets add a combination callback to create an event when the player combines the two parts in the inventory.

In OnStart add:

```
    //---COMBINE CALLBACKS----
    AddCombineCallback("keypart1","keypart2", "KeyCombine");
```

This creates a callback when the two entities keypart1 and keypart2 are combined. The callback is named ”KeyCombine”.

Adding the actual callback is done below the others in the CALLBACK FUNCTIONS area.

```
void KeyCombine(string asItem1, string asItem2, int alSlot)
{
    SetInventoryActive(false);

    RemoveItem("keypart1");
    RemoveItem("keypart2");

    PlayGuiSound("tut_voice1", 1);
}
```

What we do in this script is that , when you have done the combination of the keyparts the game will automatically close the Inventory screen and remove the two keyparts from the inventory. It will then play the sound file called ”tut_voice1” at full volume(the 1).

You can find this sound file in the ”tutorials/tutorial_5” directory, it's ”tut_voice1.ogg”. The PlayGuiSound is a useful function to use, it plays a sound directly for the player without any extra work. To play ambient sounds and other sounds in a level you will need to use other functions, these you will have to look in the content creation document for as well as to examine at Penumbra levels.

Make sure you either add the ”tutorial_5” directory to the resource.cfg file, or copy the file to your ”mystuff” directory.

The last thing to do is adding the exit to main menu command for when you enter the right code for the door.

The command for this is ResetGame(); and you this to the PanelCode callback like this:

```
void PanelCode(string asName, bool abCodeWasCorrect)
{
    if(abCodeWasCorrect)
    {
        PlayGuiSound("door_code_correct", 0.8);

        ResetGame();
    }
    else
    {
        PlayGuiSound("door_code_incorrect", 0.8);
    }
}
```

There, now you have a level with some different basic scripts creating an actual working puzzle for a player to enjoy. A good idea might be to look over the code, comment it and strcture it to make as much sense as possible so that you can go back and look at it later and know what's going on.

A bonus fix you also can do is to change the cursor for the door. Currently it displays a hand, this is the default icon, but for a door we have the door icon. To make the game diplay this icon add

```
SetAreaCustomIcon("doorarea", "DoorLink");
```

to the ”OnStart” area.

To keep learning how to script and what you can do with the engine, examine the levels and script files for penumbra. Use the script and content creation documents and participate in the discussions on the forums to learn more.