Lab 1 – Update the Creeps

Due Friday at 5PM of the week this lab comes out

The goals here are to expose you to programmatically instantiating scenes, sound, materials, and creating more events (Signals) yourself.

Introduction

This lab will (thankfully) be shorter than the last one! However, I'll start giving less guidance here. Refer to the prior lab if you need.

This lab, we will explore a few more fundamentals you should know, like how to add sounds to a game in Godot, how to do some of the things we did last lab in the Editor, but this time in code.

For this lab, you should first finish Lab 0. No solution will be provided – it is necessary to master these basics before moving on. If you had problems, this is a good time to fix them up and learn more about things you struggled with.

Sound

Sound makes a game a lot more lively and engaging. Sound itself has a huge amount of design space to explore, but unfortunately is out of scope for this class. I've seen entire research journals dedicated to the effects of sound on people's emotions, thoughts, and perceptions! If you want to learn more about getting into sound, here's a good start https://www.gamedeveloper.com/audio/sound-design-for-video-games-a-primer

Back to us though, for the basics, we'll want to be able to make sounds occur. For that, we'll need some sound files. Download Lab1water.mp3 from the Extra section of D2L. You can then drop it into your project file, or drag and drop it into the FileSystem window in Godot.

Godot uses AudioStreamPlayer ndoes to play audio easily. Add An AudioStreamPlayer3D node to your player in the Player scene (not main scene) and name it "CatchSound". Click on the node and check the Inspector tab. There are a lot of options here! We'll talk a bit more about sound later in the class. For now, drag the Lab1Water.mp3 from the FileSystem window to the "Stream" menu item in the inspector. It should set that mp3 as the audio to be played. Change the "volume" (currently at 0 decibels) to something reasonable – mine was at 7db. 7db is actually quite low (a whisper is around 30db) but this gets processed at other stages. Just start low, and bring the volume up until its comfortable – be careful as loud sounds can damage ears and sound equipment!

In the _handleCollisions() function we made last lab, add $CatchSound.play() before the emit():

```
# If the collider is with an enemy
        if collision.get_collider().is_in_group("enemy"):
                var enemy = collision.get_collider()
                $CatchSound.play()
                enemy.catch()
```

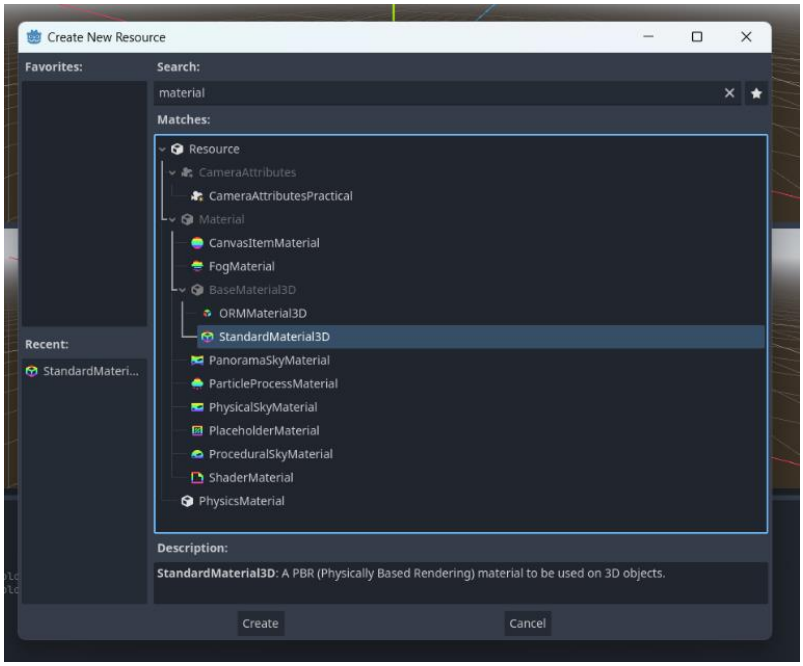
If you named your audio stream something other than “CatchSound” you should replace “$CatchSound” with “$<yourStreamNameHere”. The “$” operator allows you to access a node by name, a bit like JQuery if you’ve every used that. It’s useful for nodes you will know the name of beforehand.

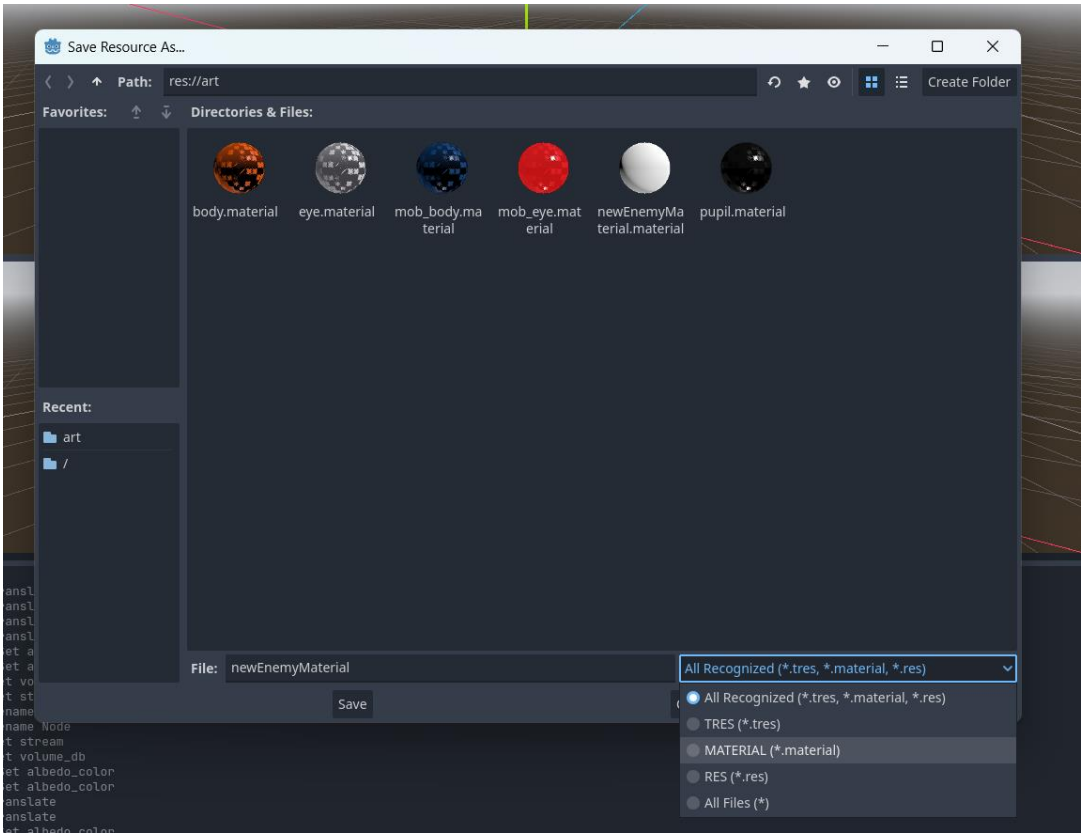
Test your game. It should now play a sound when the collision happens!

Materials

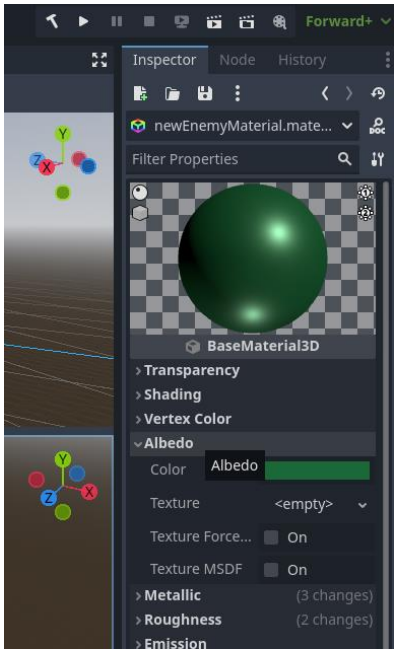
Materials tell the engine the colour, texture, and other visual properties of an object, such as how it interacts with light (reflectiveness, transparency, etc). While we won’t go into a lot of details on this subject, most game editors allow you to set and change some basic material properties of 3D objects. Materials can also be created or imported from external resources.

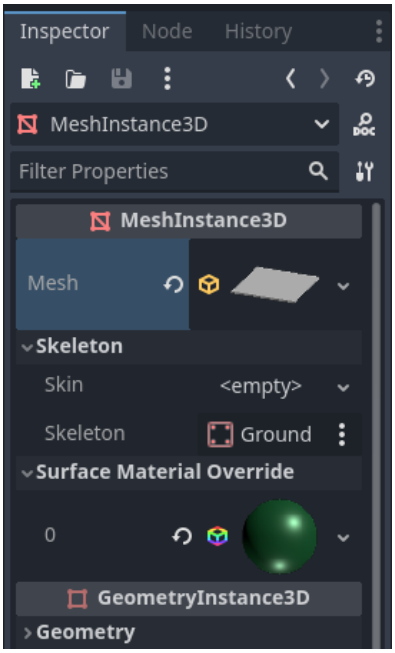
Let’s create our own material. In the FileSystem window, right click the Art subfolder and select “create new” and select “resource” from the pop out menu. Search for Material and select “StandardMaterial3D”, create it, and name it something like “newEnemyMaterial” and to the right, select ‘Material” – then save.





Now, in Godot, click on the material and check the inspector tab. There are a lot of things we can play with here, much like sound! We'll not cover a lot about this in this class, but you can look up 3D modelling, graphics, and materials for more information. You can look around and play with the settings (many only show their details once they are enabled), but for now, click Albedo, and Color. Select a color you want the floor to be. I picked a nice Green, but it's up to you.





The material is applied to the visible part of our object – the MeshInstance3D node of our Ground. Click on the MeshInstance. On the inspector tab, note in the MesInstance3D properties the Surface Material Override property, and drag your material from the FileSystem to that property. The ground should now be that colour!

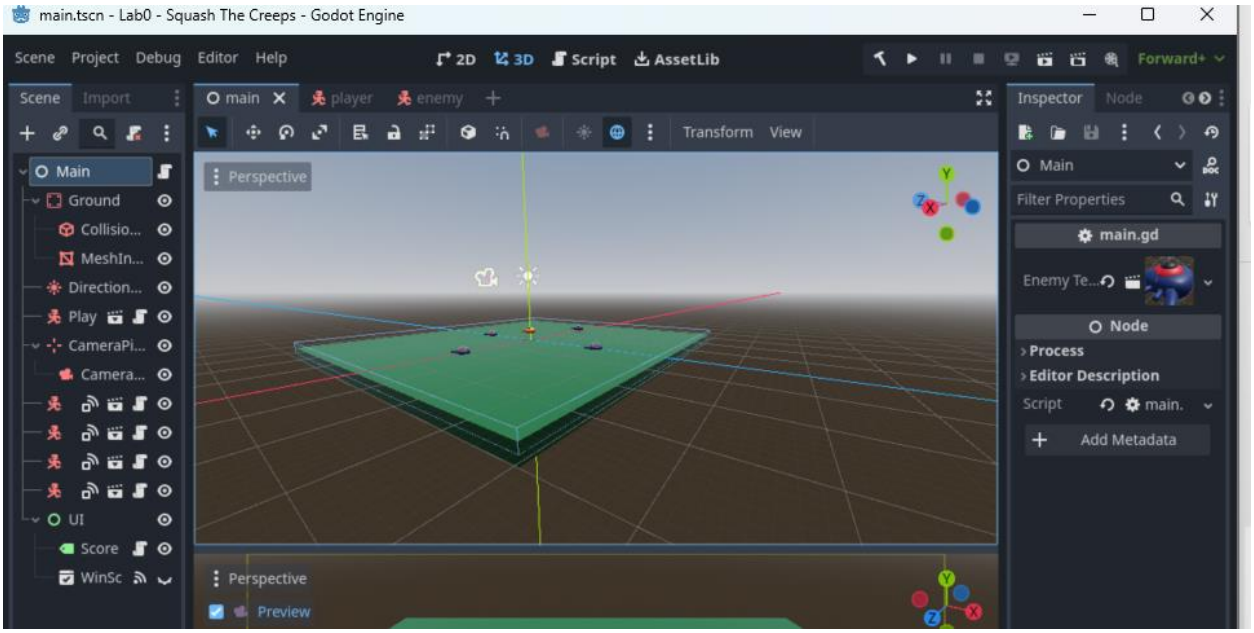
Spawning Creeps

In Lab 0 we manually created creeps in the inspector. However, it's a bit boring to only have the 4 creeps in the game, and is annoying to add them by hand. This lab, we will instantiate those child scenes ourselves in our program so we can spawn in new creeps dynamically!

First, as this will be something controlled by the environment, let's attach a new script to the Main node of the main scene. Then, add in a variable:

```
@export var EnemyTemplate : PackedScene
```

The : operator after a variable allows you to tell Godot what type something is. This is useful for a number of reasons, including autocomplete in the script editor, and for assigning specific typed nodes in the visual editor. Save the script, and go to your main scene. You should see the EnemyTemplate variable in the Inspector tab – locate your enemy.tscn (or mob.tscn, whatever your enemy scene was called), and drag it onto the EnemyTemplate property which should then look like this:



Let's spawn the enemies in on a timer. Add a new child node to Main called SpawnTimer. In the editor, change the Wait Time to the desired spawn rate – I set mine to 5 seconds, but h[illegible] fun! Then select Autostart to make sure the timer starts as soon as the game does. The timer will now emit a Signal at the rate you specified (in my case, 5s). Check the Signals in the Node tab, and note the timeout() event Signal now available. Double click it to create a function to handle this event, and let's make it in the main script we created earlier. This will create _on_spawn_time_timeout() in main.gd.

In this script, we'll want to create a creep, and give it a position randomly on the ground. First, let's create a copy of an enemy:

```
var enemy = EnemyTemplate.instantiate()
```

Now, it's not in the game yet – we have to add it to our Scene Tree, but before we do that, we should completely finish setting it up (or strange things can happen…). Let's give it a position. To do this, we'll need to know how big the ground is, and to generate a random number. We can get the ground's size from it's MeshInstance properties. We will then use randf(), which generates a random float between 0 and 1 inclusive, and use that with our ground size to set a position:

```
#set position of enemy to randomly on the ground
var groundSize = $Ground/MeshInstance3D.mesh.size
enemy.position.x = (randf() * groundSize.x) - groundSize.x/2
enemy.position.z = (randf() * groundSize.z) - groundSize.z/2
```

Note the new notation here. We saw "$" to select a node by name. We can select a sub node with the "/" operator. So $Ground/MeshInstance3D looks for a Node called Ground in the Scene Tree, and then a subnode called MeshInstance3D. We then grab properties as you may expect. Note the properties are just the ones we can see in the Inspector tab, but lower case. If you're unsure what properties a Node has, consult the Godot doc online, or by right clicking a Node or function and selecting "lookup symbol" for code or "open documentation" for nodes.

Then, we set the position. Any guesses how this works? Why do we subtract groundSize/2? Think about it yourself or discuss it with a labmate. If you can't figure it out, ask the lab instructor or TA.

Finally, add the Creep to the scene:

```
add_child(enemy) #adds as a child of ourselves
```

Try it out! It works mostly fine buttttttt….

Note that score doesn't go up when we catch the programmatically spawned Creeps. Take a moment and think about what we've done in the editor after instantiating our enemies in Lab0…

Link the Signal to the function_on_enemy_caught function! We can do that programmatically by calling connect() on a Signal from a specific Node, and providing a function to run when that signal is called. So, for our enemy, that looks like:

```
enemy.caught.connect($UI/ScoreLabel._on_enemy_caught)
```

Now let's try it! You main want to adjust your MAX_SCORE now that there are more creeps.

Losing a Game

For this one, I'll be leaving it up to you to leverage what you have learned so far. Let's make the game have actual challenge by having to collect a certain number of creeps within a time limit. Let's say you have to collect 5 creeps with a challenging time, like 15 or 20s. Display the timer counting down on screen. When it hits 0, display a game over and restart message. Good luck!

Useful tip: you can only $node your child nodes. If you want a non-child, use the whole path:

```
$/root/Main/SpawnTimer.stop()
```

For a final challenge (OPTIONAL), try moving your creeps until they go off the cliff, then delete them! Look to your player movement for inspiration on this.