Godot 4 for Beginners

By Daniel Buckley
Godot Game Developer

Table of Contents

How to Create a Game with Godot 4 – Beginner's Tutorial

Want to make games, but are tired of everything being about Unity and Unreal Engine?

Godot is a great game engine for both 2D and 3D games – making it a multi-purpose engine suitable for many kinds of projects. It's also open source and has a strong community backing the project, so there are constant updates and new features being released. With the release of Godot 4, it's also better than ever to explore game development with this fantastic tool.

In this tutorial, we're going to be learning how to create a very simple coin collection game in Godot 4. We'll discover not only some of the core tools offered by Godot, but how to use GDScript to render our various mechanics.

Without wasting any more time, let's dive in!

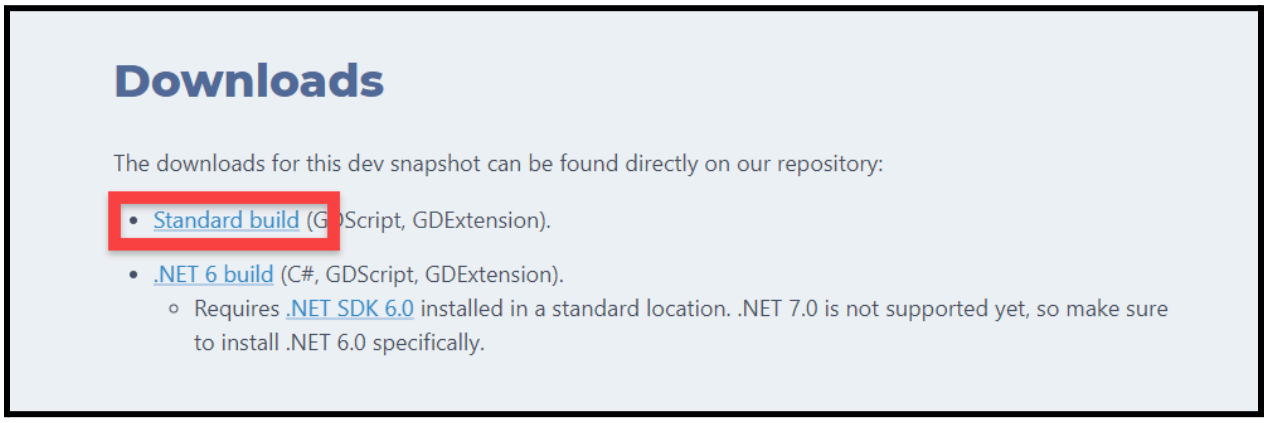
Project Files

You can download the complete project we'll be making in this tutorial here.

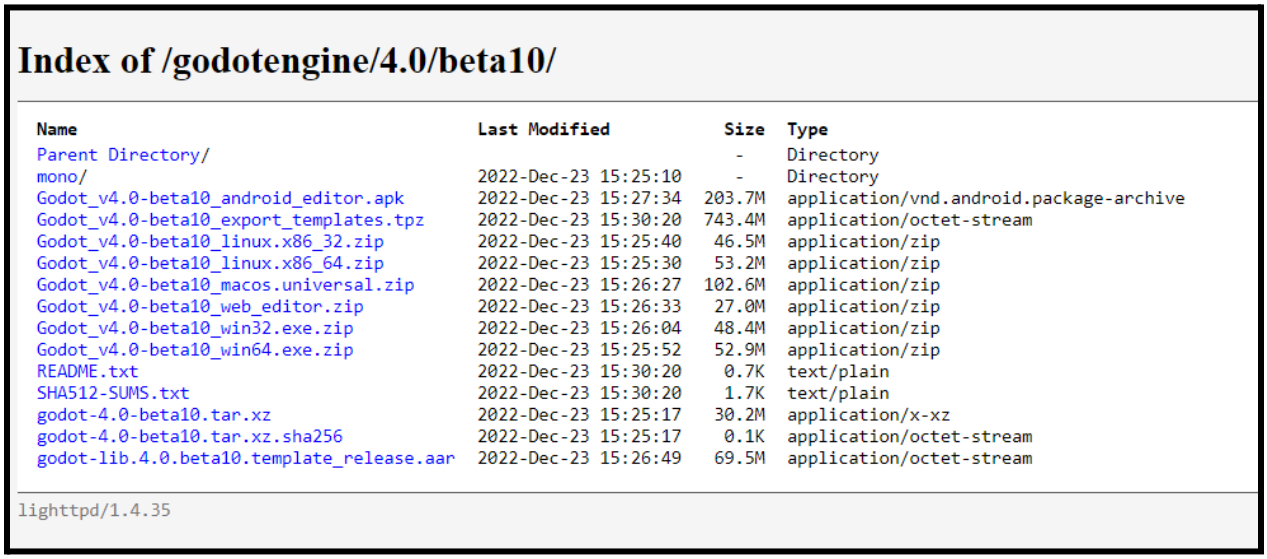
While we'll include it later below, you may also wish to download the character sprite and coin sprite that we'll be using as well.

Installing Godot

Right now, Godot 4 is in beta, so let's go to the Beta Download Page and click Standard Build.



Then, we want to click on the download that matches our operating system to begin the download.

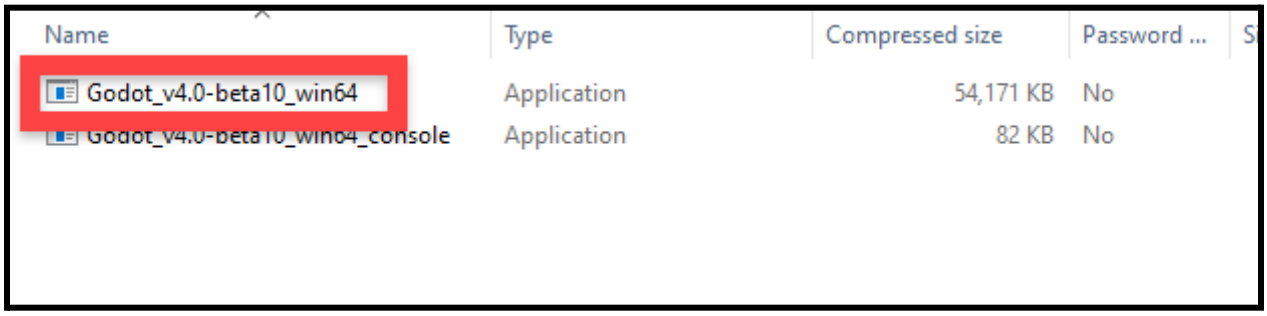


Index of /godotengine/4.0/beta10/

Name	Last Modified	Size	Type
Parent Directory/		-	Directory
mono/	2022-Dec-23 15:25:10	-	Directory
Godot_v4.0-beta10_android_editor.apk	2022-Dec-23 15:27:34	203.7M	application/vnd.android.package-archive
Godot_v4.0-beta10_export_templates.tpz	2022-Dec-23 15:30:20	743.4M	application/octet-stream
Godot_v4.0-beta10_linux.x86_32.zip	2022-Dec-23 15:25:40	46.5M	application/zip
Godot_v4.0-beta10_linux.x86_64.zip	2022-Dec-23 15:25:30	53.2M	application/zip
Godot_v4.0-beta10_macos.universal.zip	2022-Dec-23 15:26:27	102.6M	application/zip
Godot_v4.0-beta10_web_editor.zip	2022-Dec-23 15:26:33	27.0M	application/zip
Godot_v4.0-beta10_win32.exe.zip	2022-Dec-23 15:26:04	48.4M	application/zip
Godot_v4.0-beta10_win64.exe.zip	2022-Dec-23 15:25:52	52.9M	application/zip
README.txt	2022-Dec-23 15:30:20	0.7K	text/plain
SHA512-SUMS.txt	2022-Dec-23 15:30:20	1.7K	text/plain
godot-4.0-beta10.tar.xz	2022-Dec-23 15:25:17	30.2M	application/x-xz
godot-4.0-beta10.tar.xz.sha256	2022-Dec-23 15:25:17	0.1K	application/octet-stream
godot-lib.4.0.beta10.template_release.aar	2022-Dec-23 15:26:49	69.5M	application/octet-stream

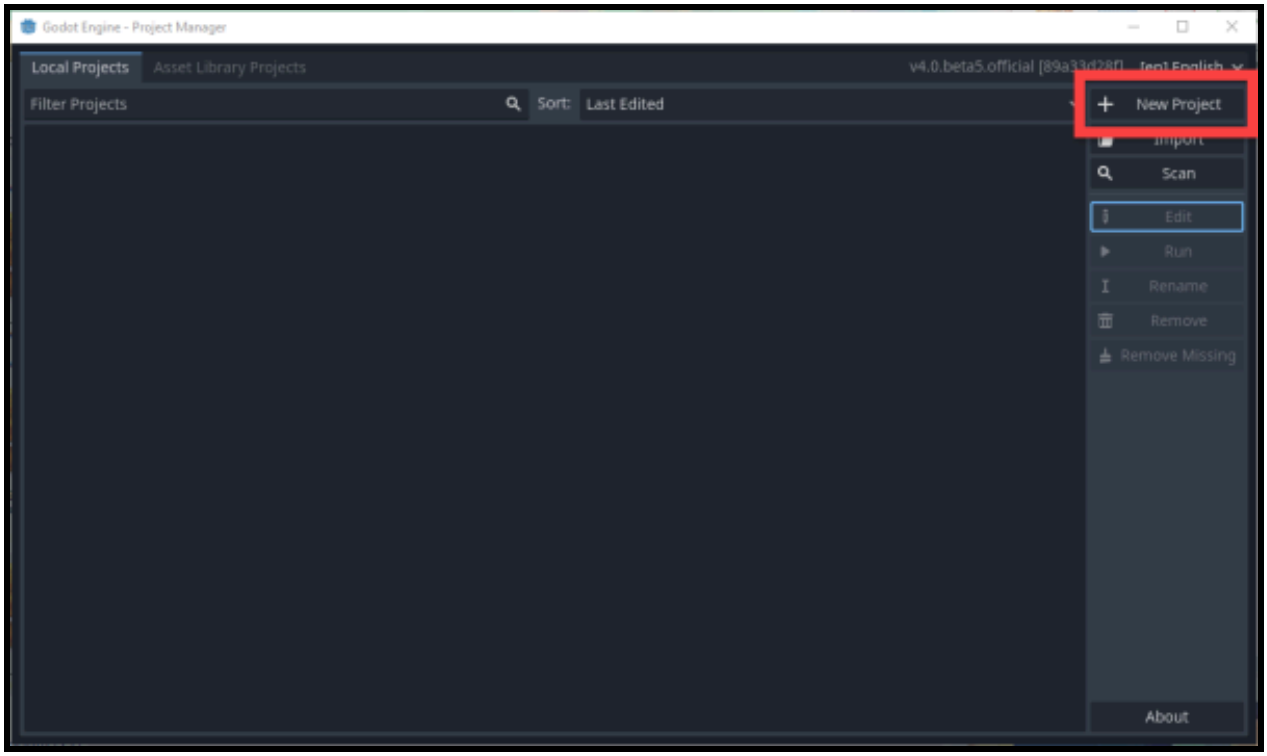
lighttpd/1.4.35

Once that the download is complete, you should have a ZIP file which you can extract the contents of. The application file is the game engine, so we can double click on that to open it up.



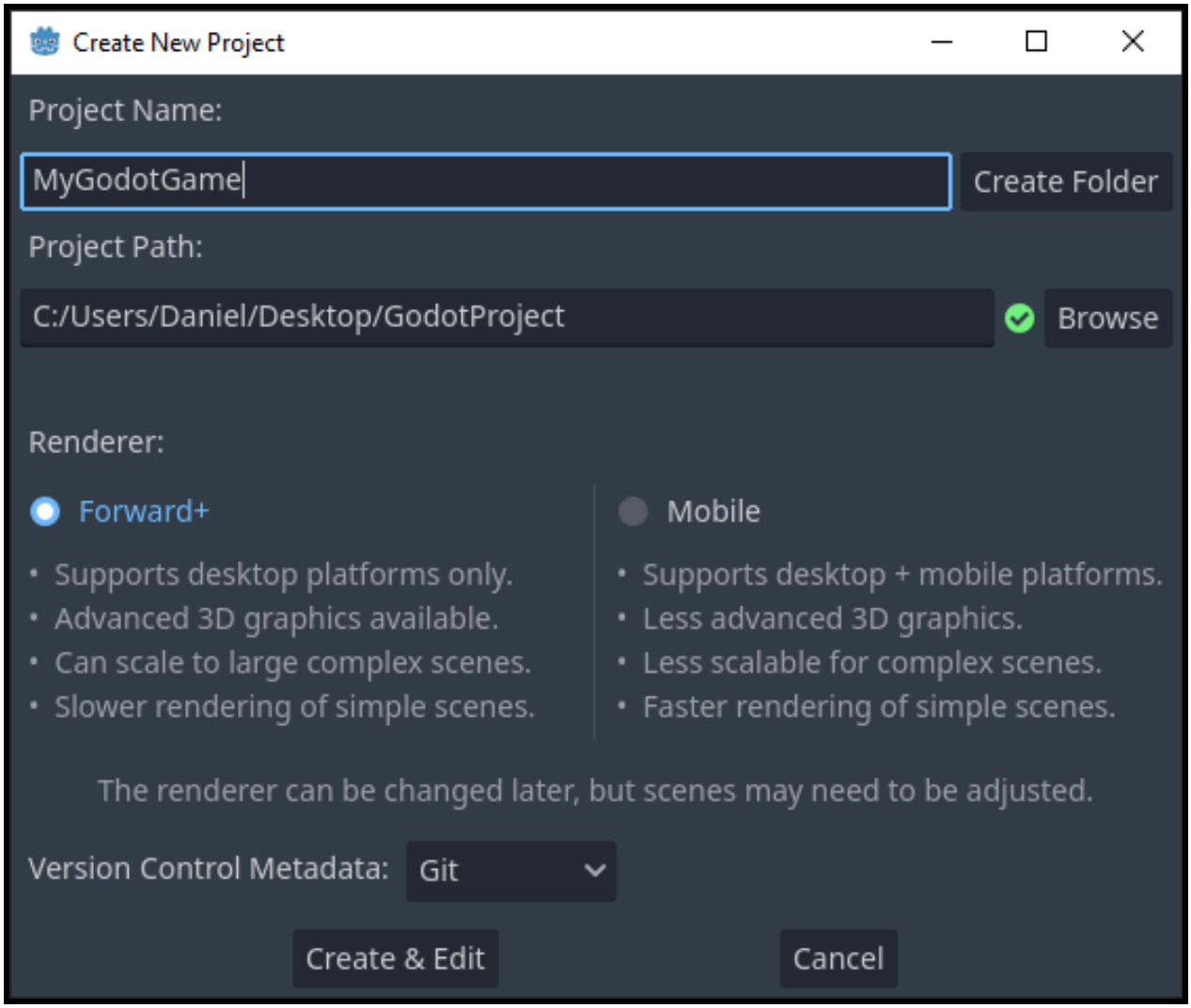
Creating a Project

You should now have the *Project Manager* window open. This is where all of your Godot projects will be listed. For now, we have none, so let's create one. Click on the **New Project button**.



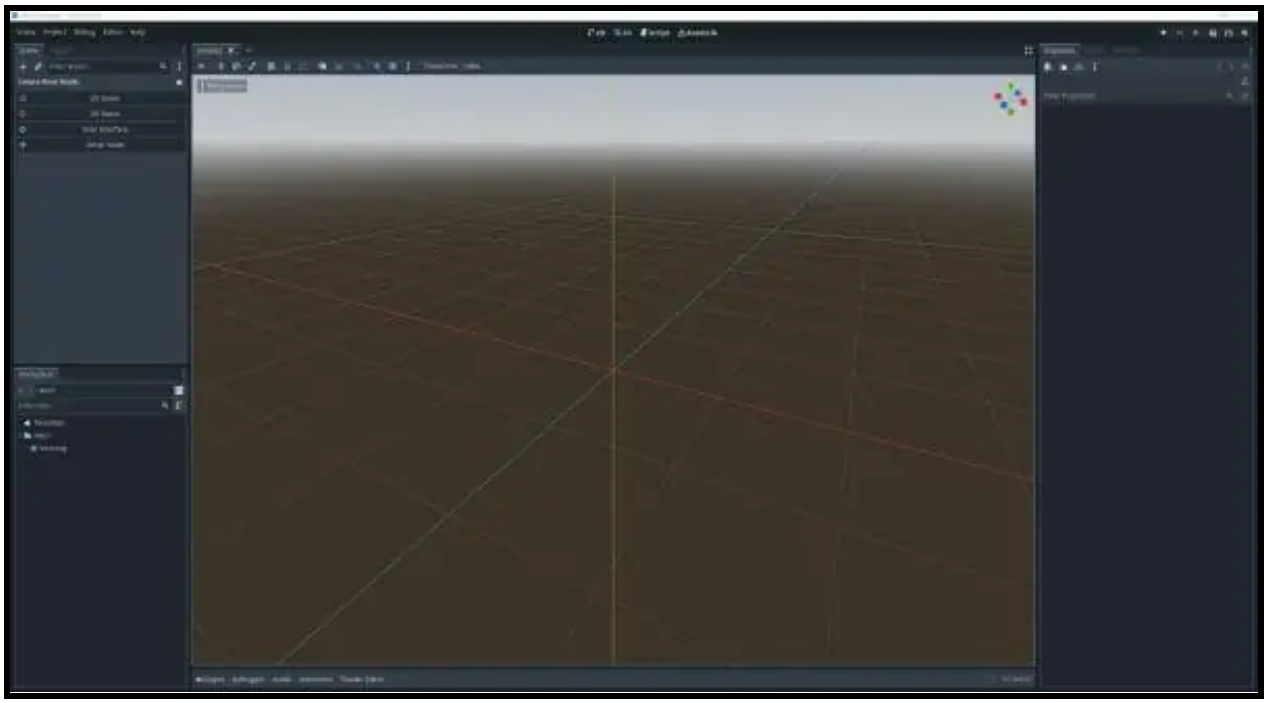
This will open up another window where we need to setup a few things.

1. Choose a *Project Name*.
2. Choose a *Project Path*. This will be where the project files are located on your computer.
3. Then, click **Create & Edit**.

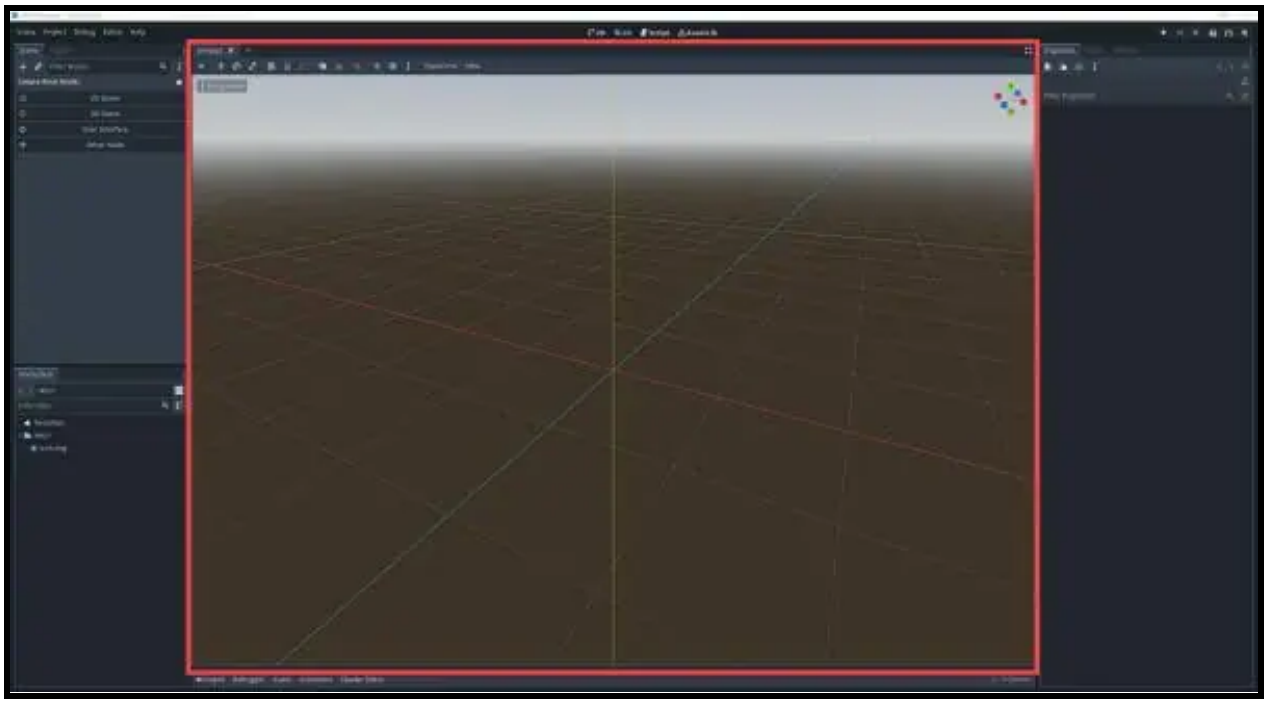


Editor Overview

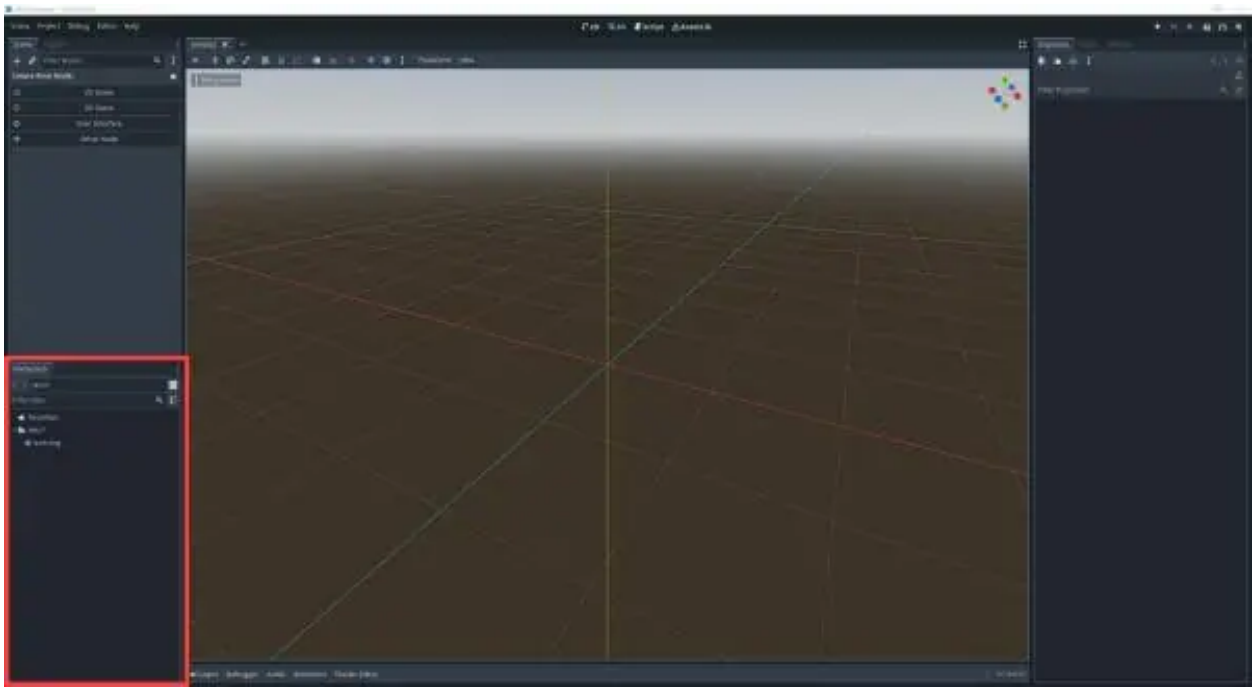
After you create the project, the Godot editor should open up. This is the window where we'll be spending most of our time in creating our games. This is where we will manage our assets, build our levels and code our gameplay behaviors. At first, this may seem daunting. There are a number of different panels, buttons, and labels. But hopefully, as time goes on, it will all become second nature to you.



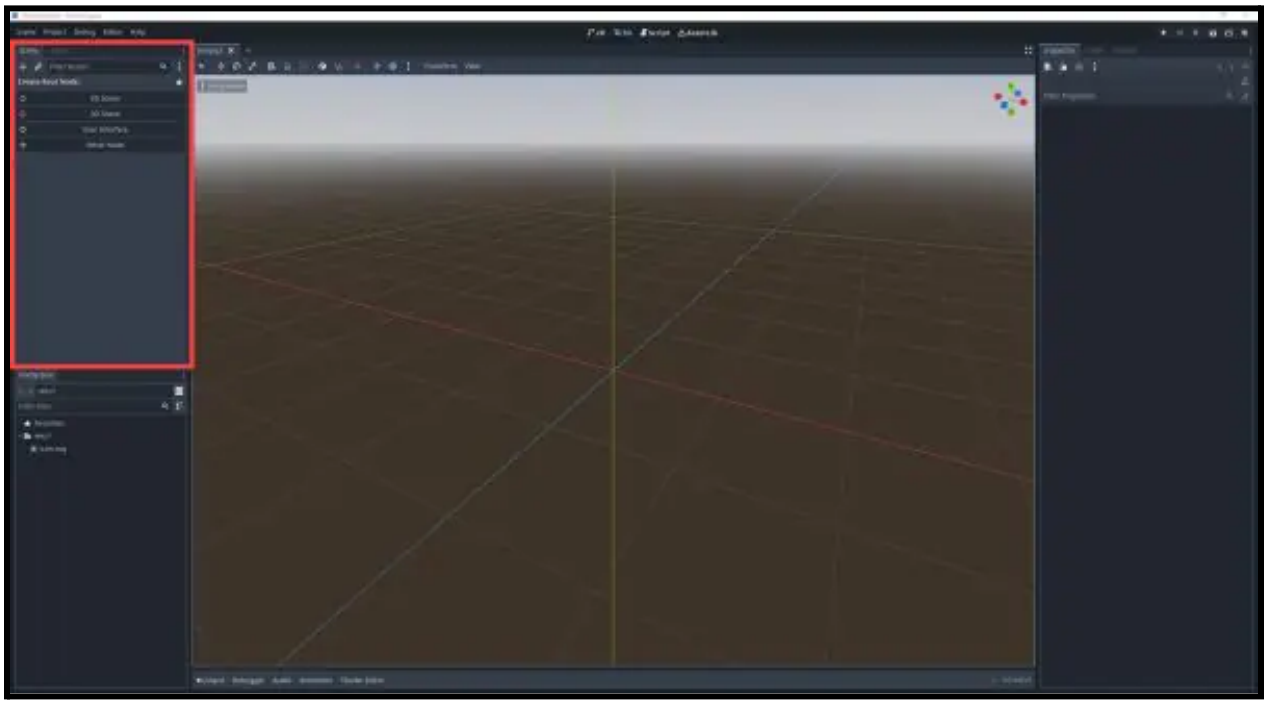
Let’s start by looking at what each of these panels do. First, in the center, we have our scene view. This is a window into our game and where we’ll be clicking on objects, dragging them in, and building our levels. At the moment, the engine is set to 3D, so that’s why we have a 3D view.



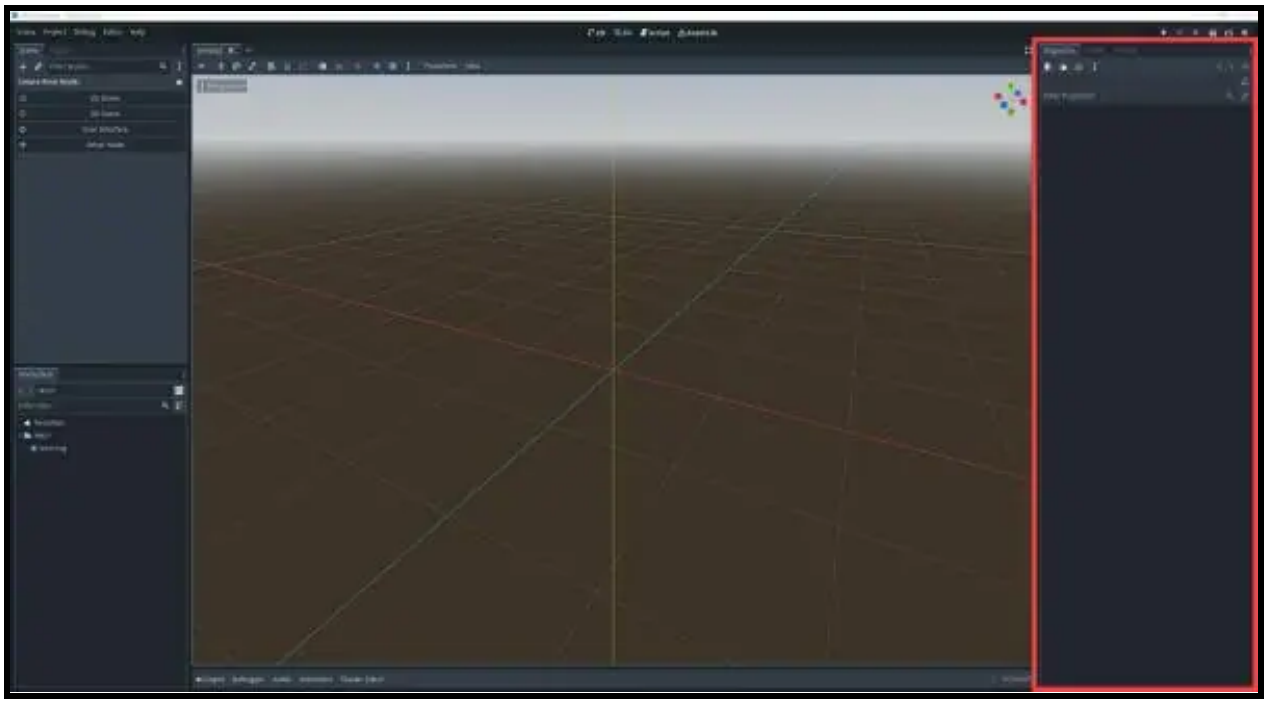
On the bottom left, we have our **File System**. This panel contains all of our assets (textures, models, sprites, sound effects, scripts, etc). It's very similar to the file explorer on your computer.



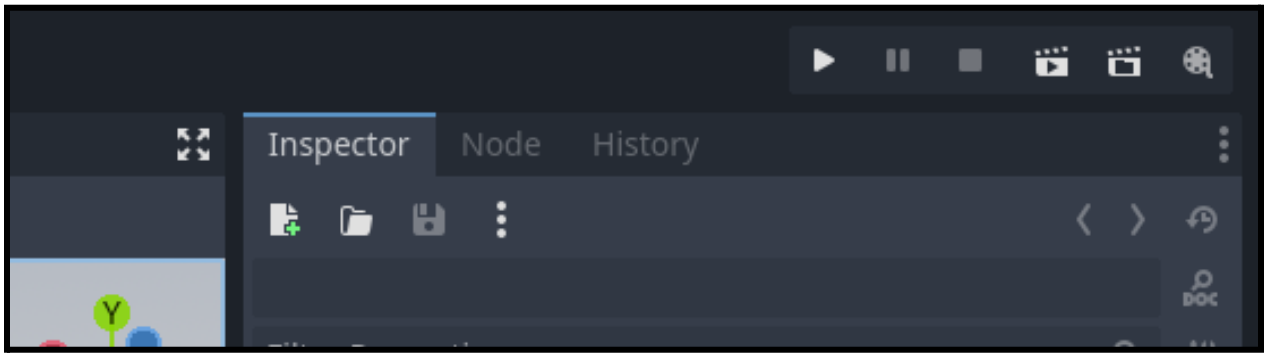
At the top left, we have the **Scene** panel. This here is a list of all the nodes currently in our scene. The way Godot works is with a node system. We'll get more into that later, but it's important to know that everything in your game (the player, camera, objects, lights, etc) are all nodes. And a scene is a collection of nodes.



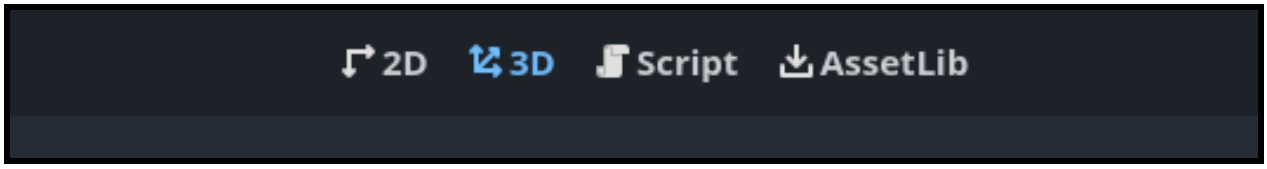
On the right-hand side of our editor, is the **Inspector**. Whenever we select a node, this panel will populate to show us all of that node's properties. Its position, rotation, scale and any other things that we might want to modify. If it's a light node, then the inspector will allow us to change the color, brightness, range, angle, etc.



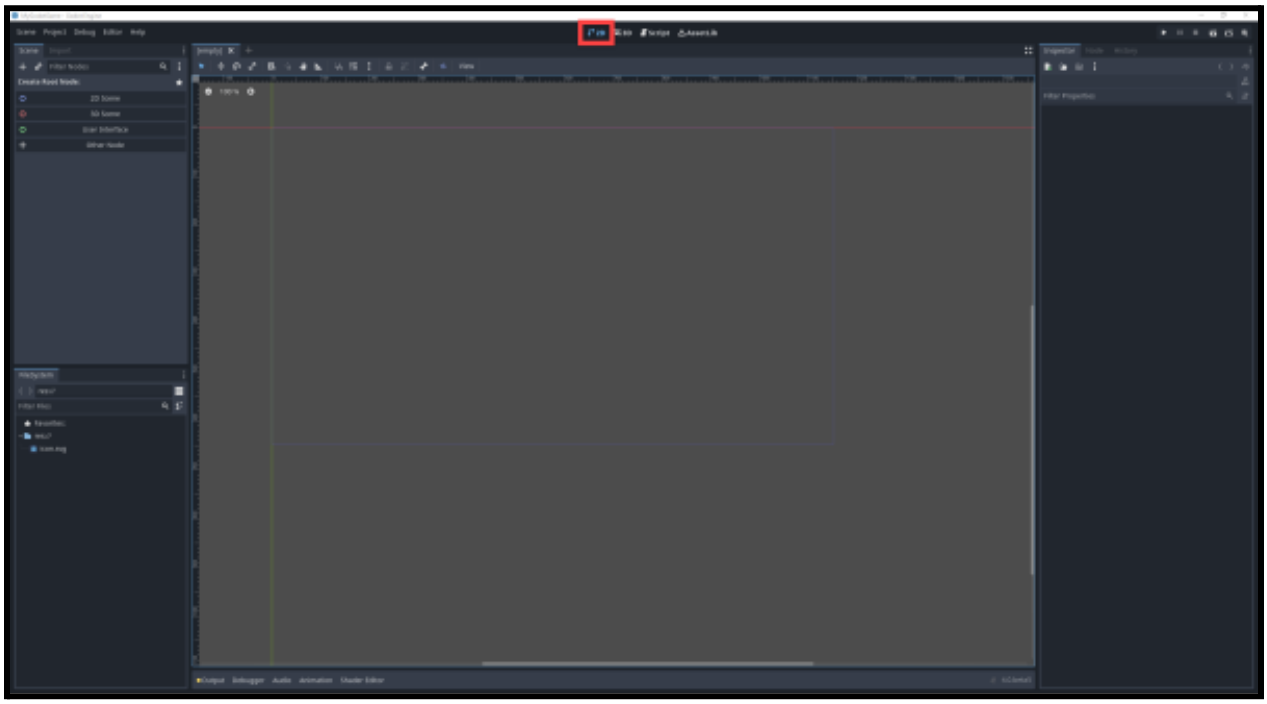
At the top right corner of the screen, we have our play tools. These allow us to test out our game while we're developing it. We can play, pause, stop, run the current scene, run a specific scene, or use the movie maker.



At the top center of the screen, we have a few buttons. These allow us to change what we see down in the large, center-scene viewport. Right now, it's set to 3D, so that's why we see our 3D scene. But we can easily switch over to 2D if that's the type of game we want to create. When we start to code, we'll be hopping over to the Script tab, as that's where the text editor is contained.

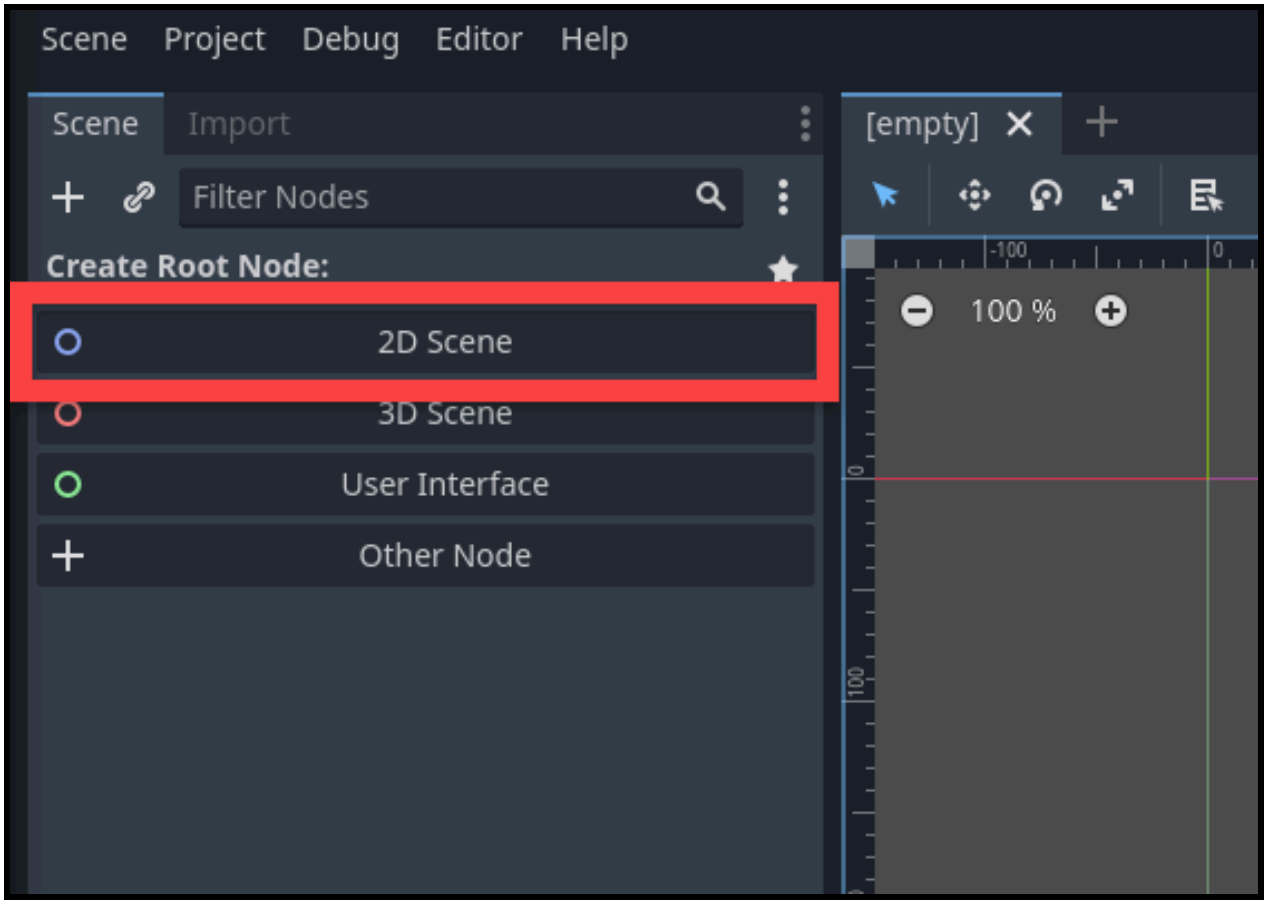


For this tutorial though, we'll be creating a 2D game. So let's click on the 2D button to switch the window over to 2D mode.



Nodes

Now that we're ready to begin, let's go over to the **Scene** panel and create our first root node. Select the **2D Scene**.



This is going to create our root node, which is now visible in the list. The way Godot works is through nodes. Everything in your game is nodes. 3D models, tiles, blocks, the player, enemies, UI – everything that can have a position in space or be instanced is going to be a node. Now with these nodes, they can be nested inside of each other. For example, a player node will also have a child node that acts as a collider. Another child node renders its sprite to the screen and another child node for the camera.

If you wish to learn more about Nodes and Scenes, check out the Godot documentation.

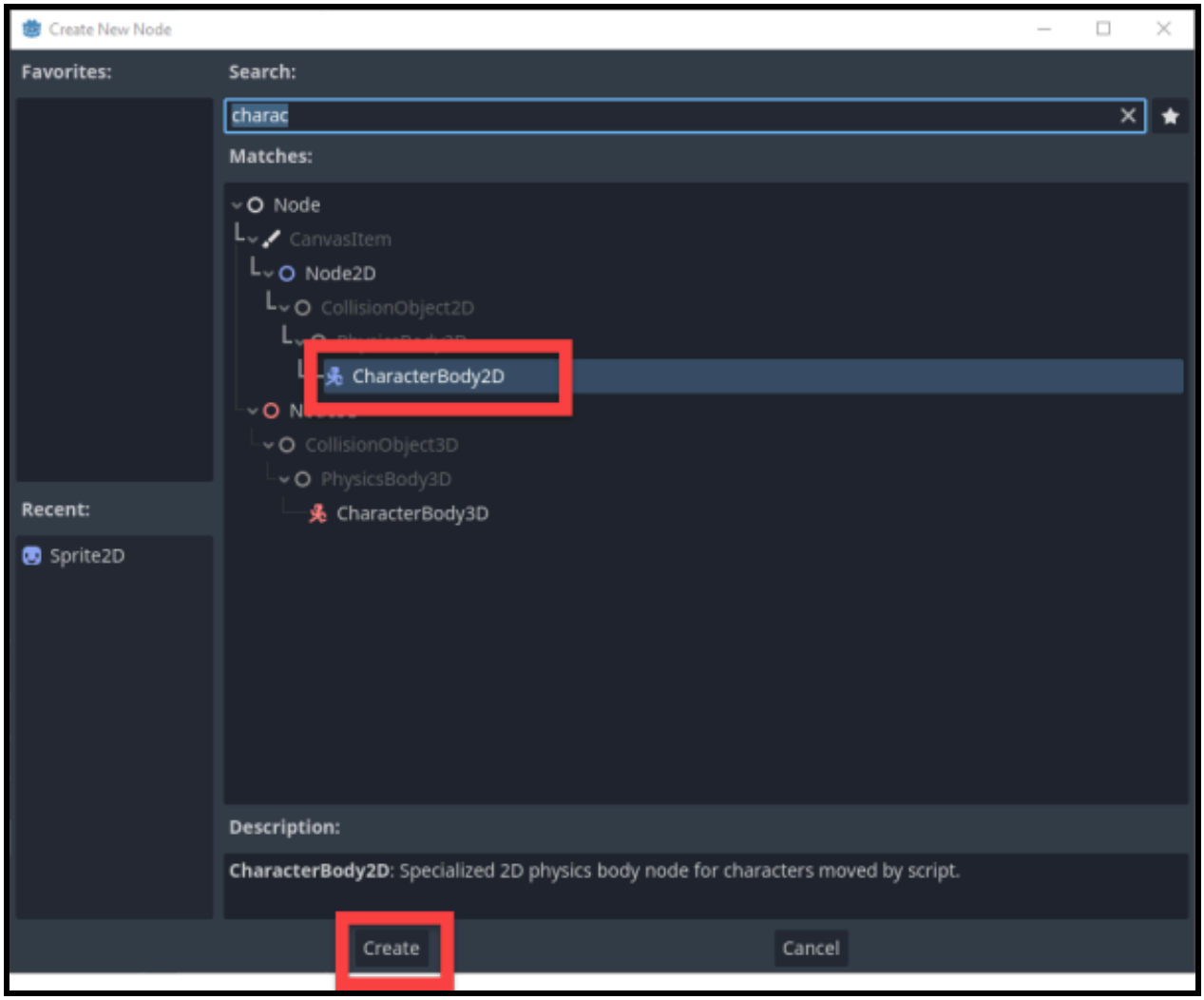
Now before we continue, let's save our scene. A scene is basically a collection of nodes that can be loaded into our game. Press *CTRL + S* or go to *Scene > Save Scene As* to save. Let's call it **Level1.tscn**, then click **Save**.

Creating the Player

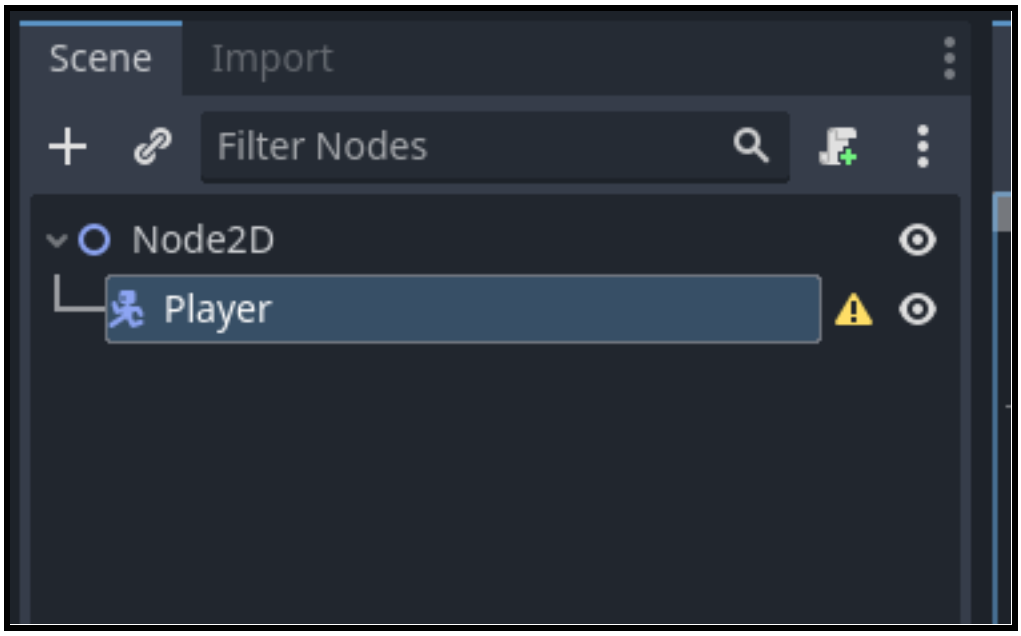
Let's get started on creating our player. We, first of all, need to figure out what node we want our player to be, because each node has a specific type. Each node type serves a different

purpose, and for our player, we'll be using a **CharacterBody2D** node. So to get started, go to the top right corner of the **Scene** panel, and click on the + icon.

In *the Create New Node* window, search for **CharacterBody2D**, select it, then click **Create**.

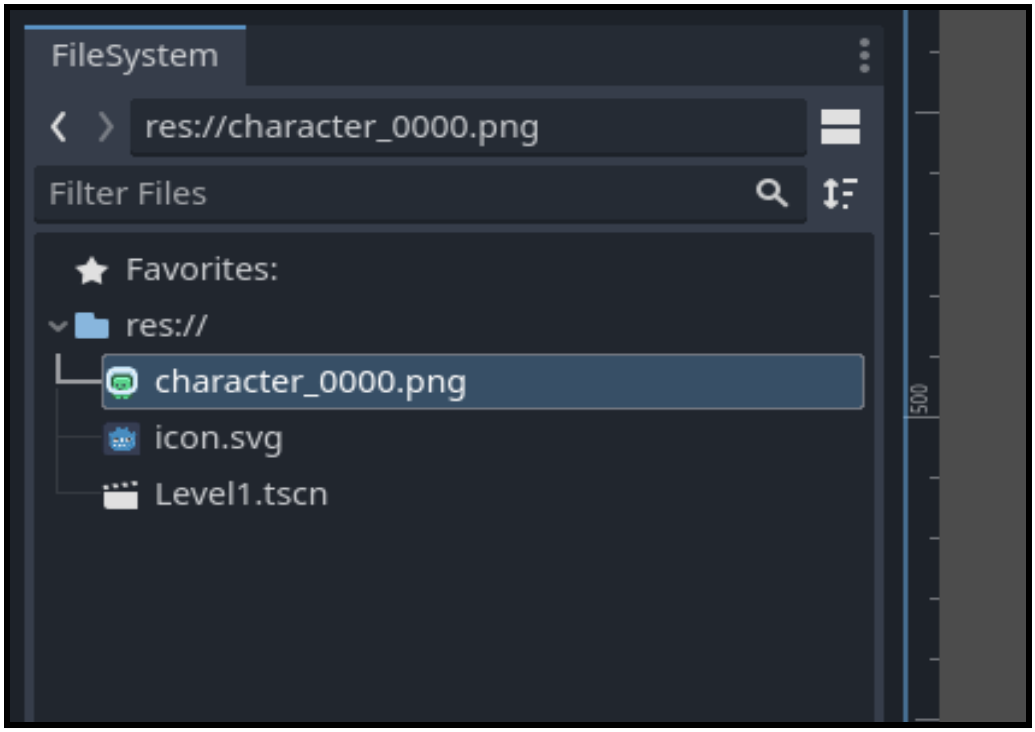


This will create the node. Now we can double-click on it to rename the node. Rename it to *Player*.

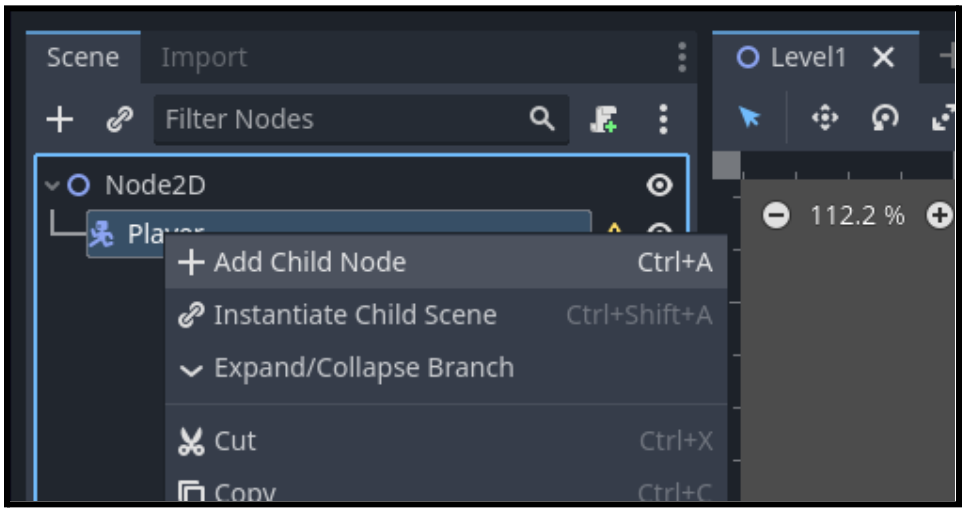


Now that we have our root player node, we need to give them a visual. Download this ZIP and extract the PNG inside.

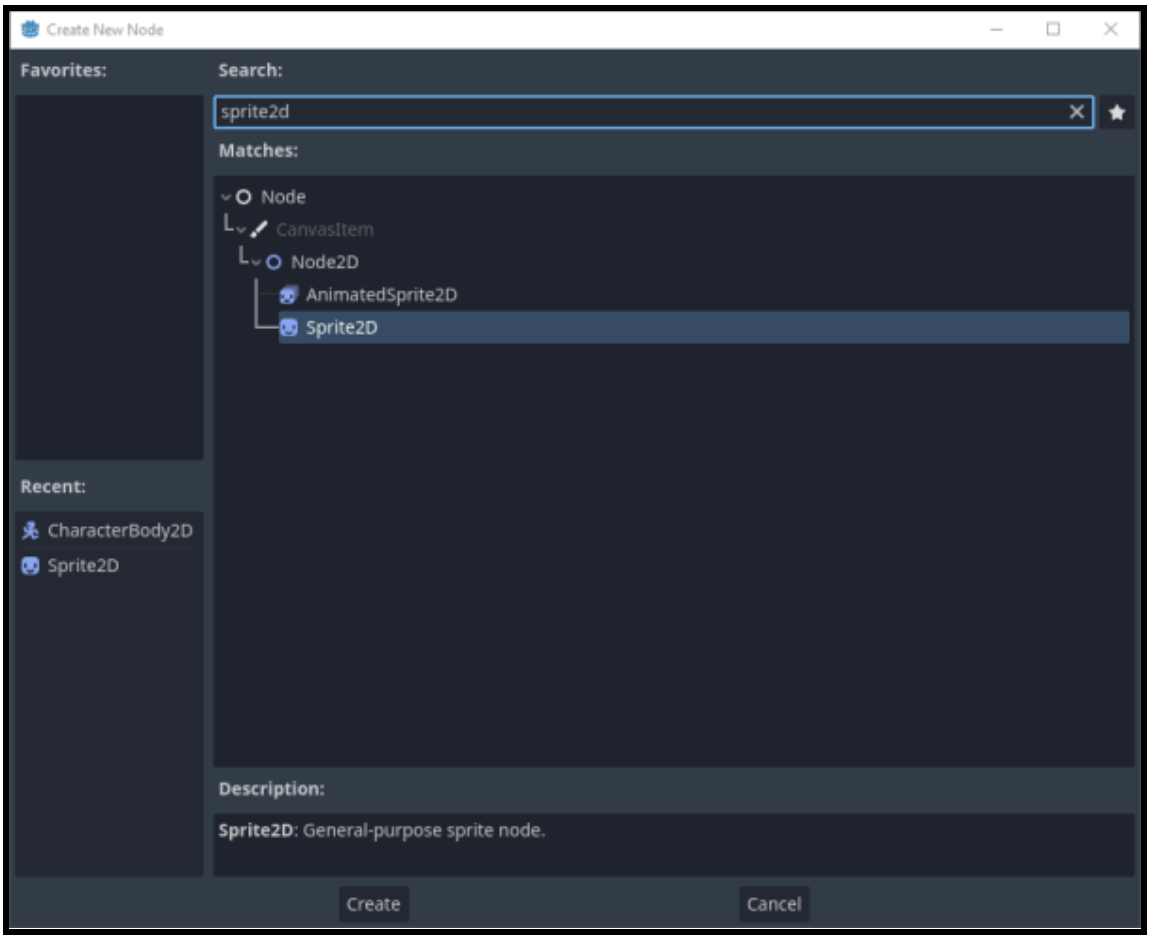
With the image downloaded and extracted, drag it into the **FileSystem** panel to import.



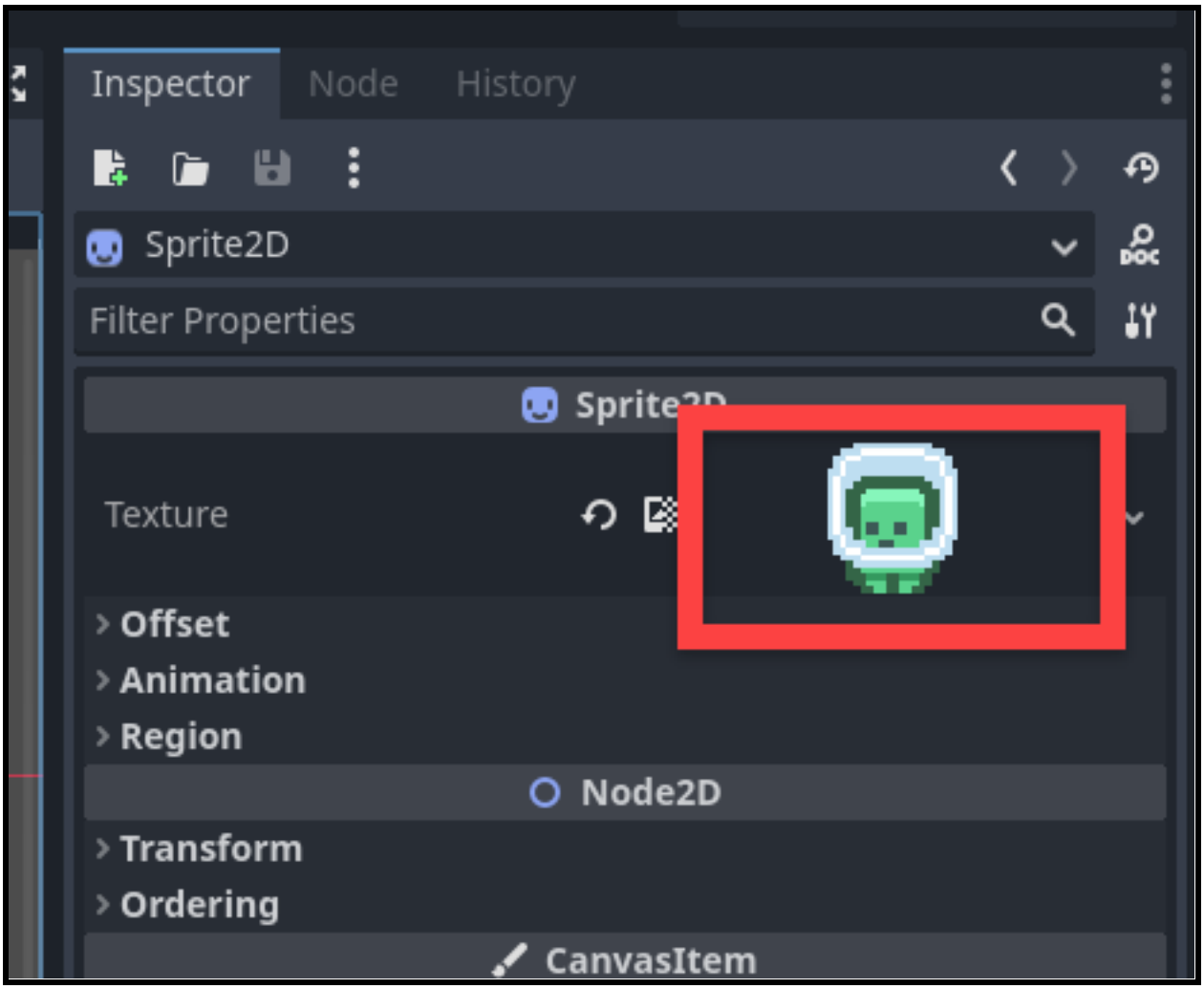
Now on our player node, right click it and select *Add Child Node*.



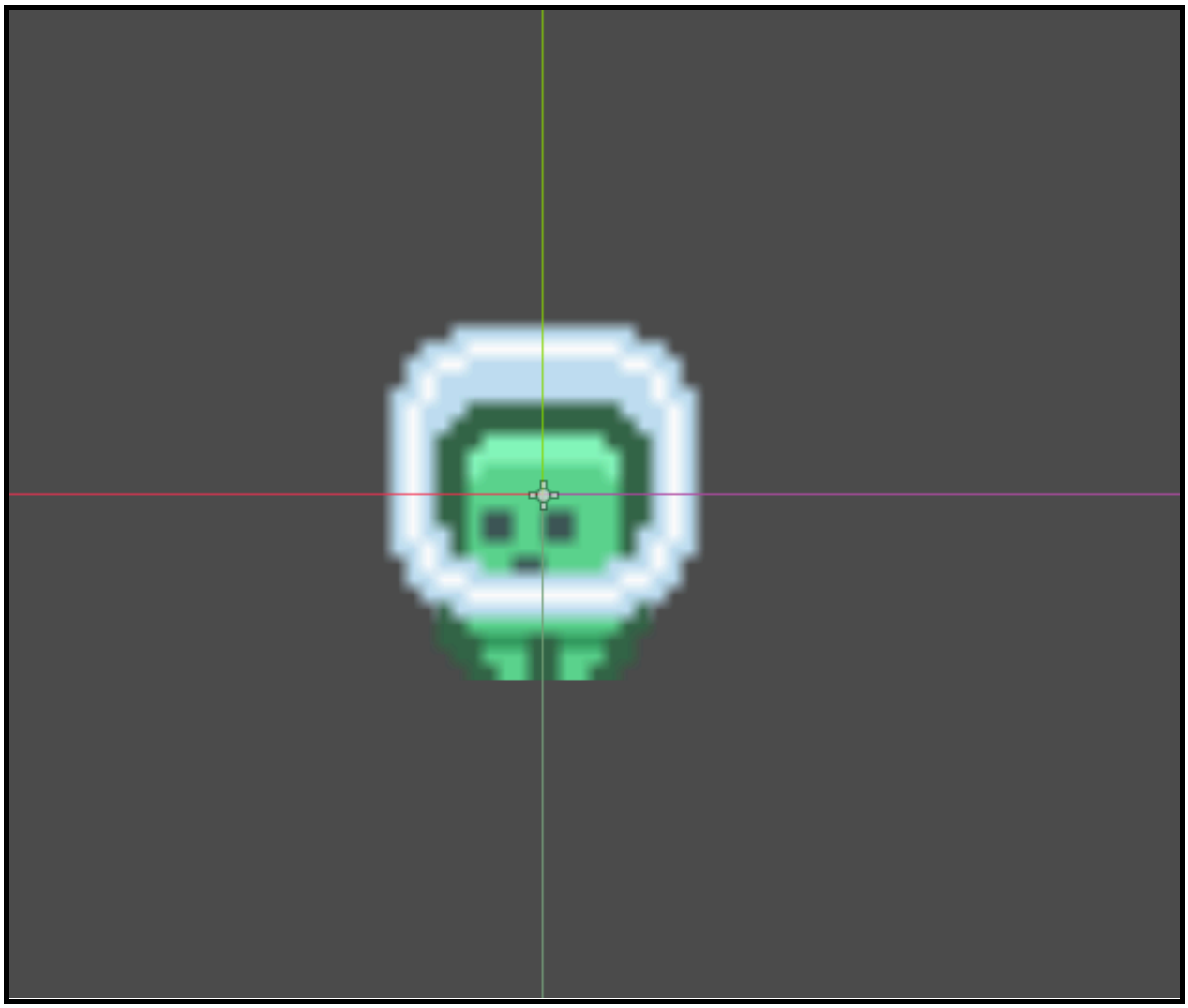
Search for **Sprite2D**, select it and then click **Create**.



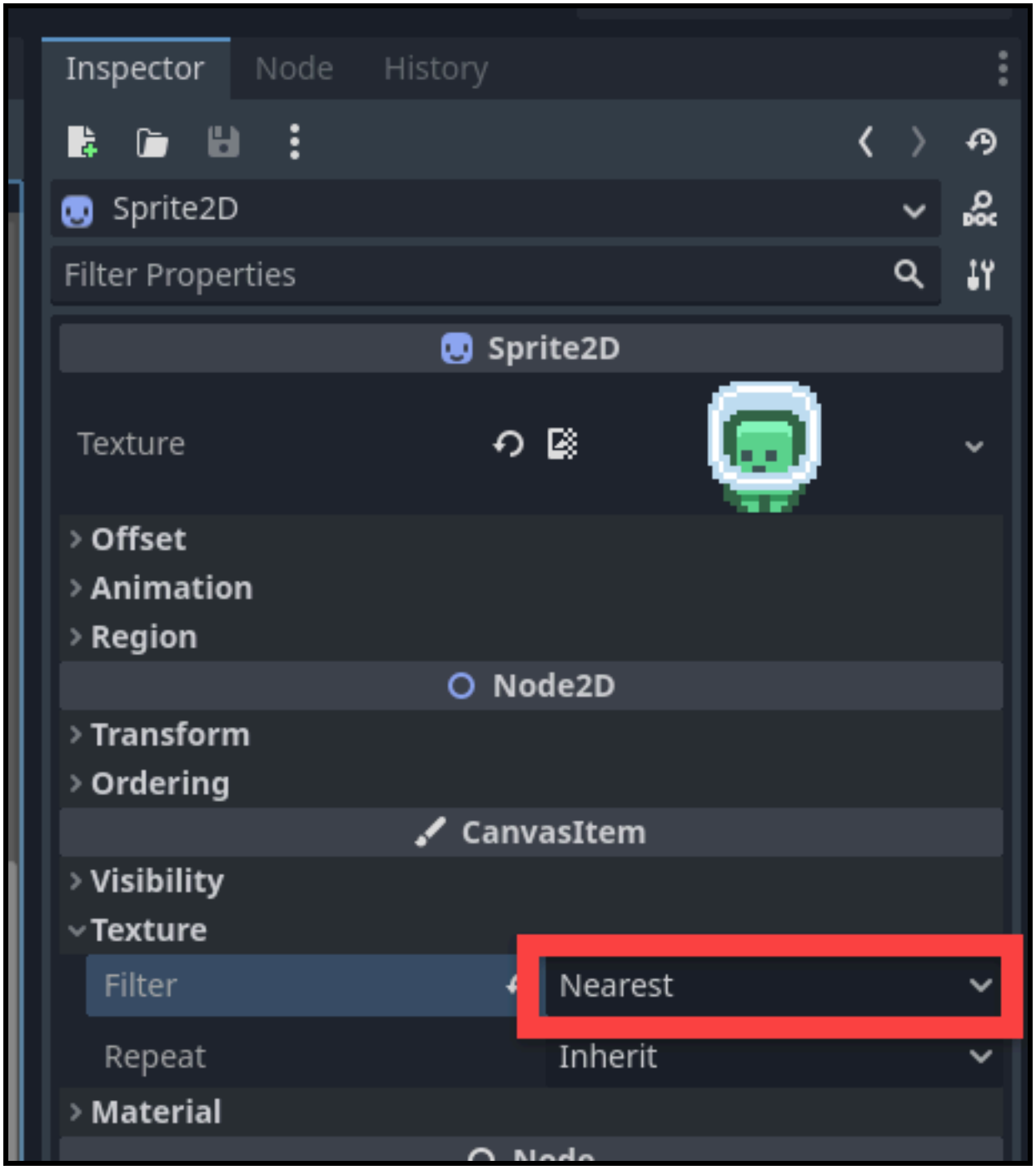
With the new child node selected, look over at the **Inspector** and you should see that the panel has a bunch of information. There is a texture property that we need to enter. From the **FileSystem**, click and drag the character sprite over into the field to assign it.



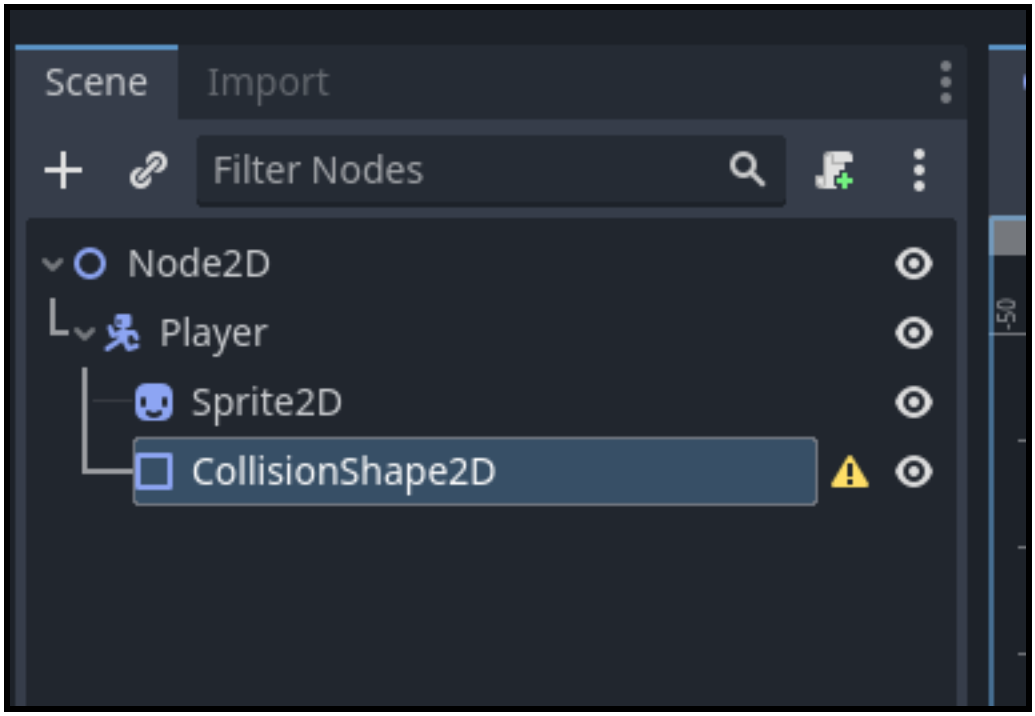
Now we can zoom into the player in the scene view, but you'll notice that it's blurry.



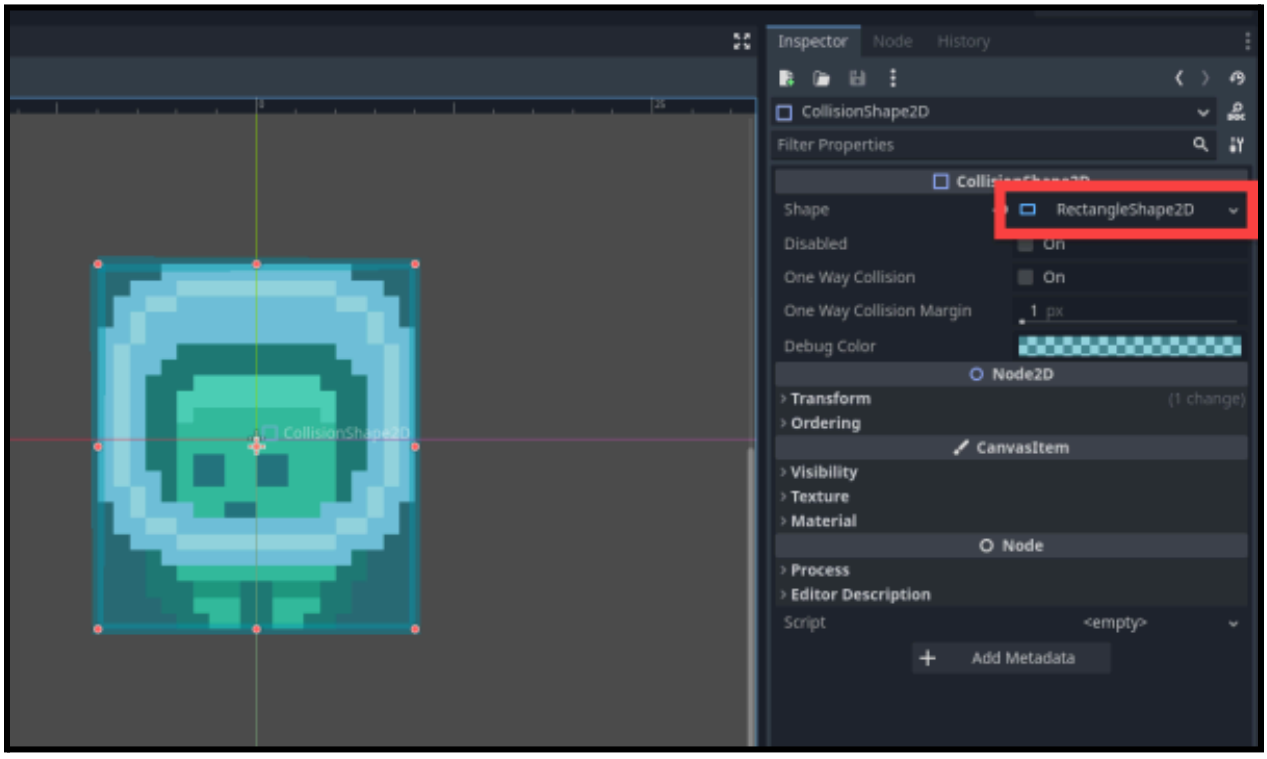
To fix this, select the sprite node, go to the inspector, and under the **Texture** drop-down, set **Filter** to *Nearest*.



Our player also needs a collider, so add a new child node of the type **CollisionShape2D**.



In the inspector, you can set the **Shape** to be a *RectangleShape2D*. Then in the scene view, click and drag on the orange dots to fit it to our sprite.

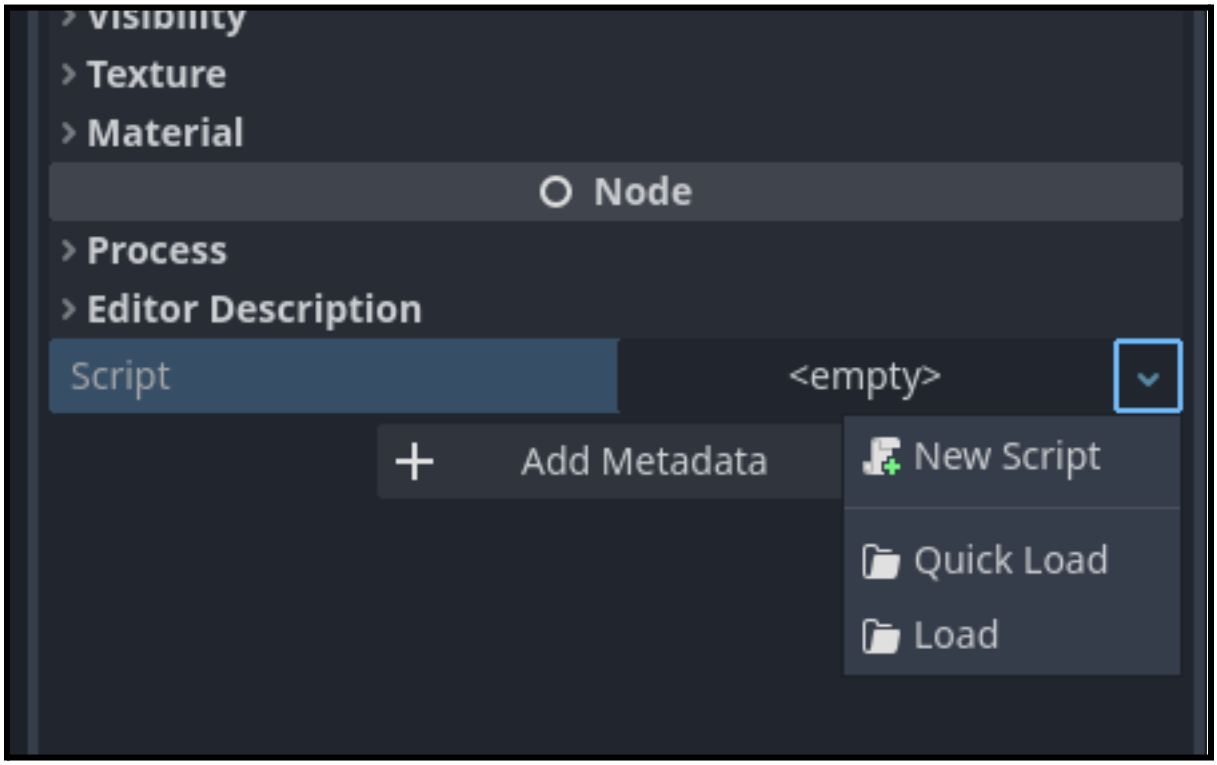


And there we go! We now have our player created. Let's move on to coding, where we'll get our player moving around.

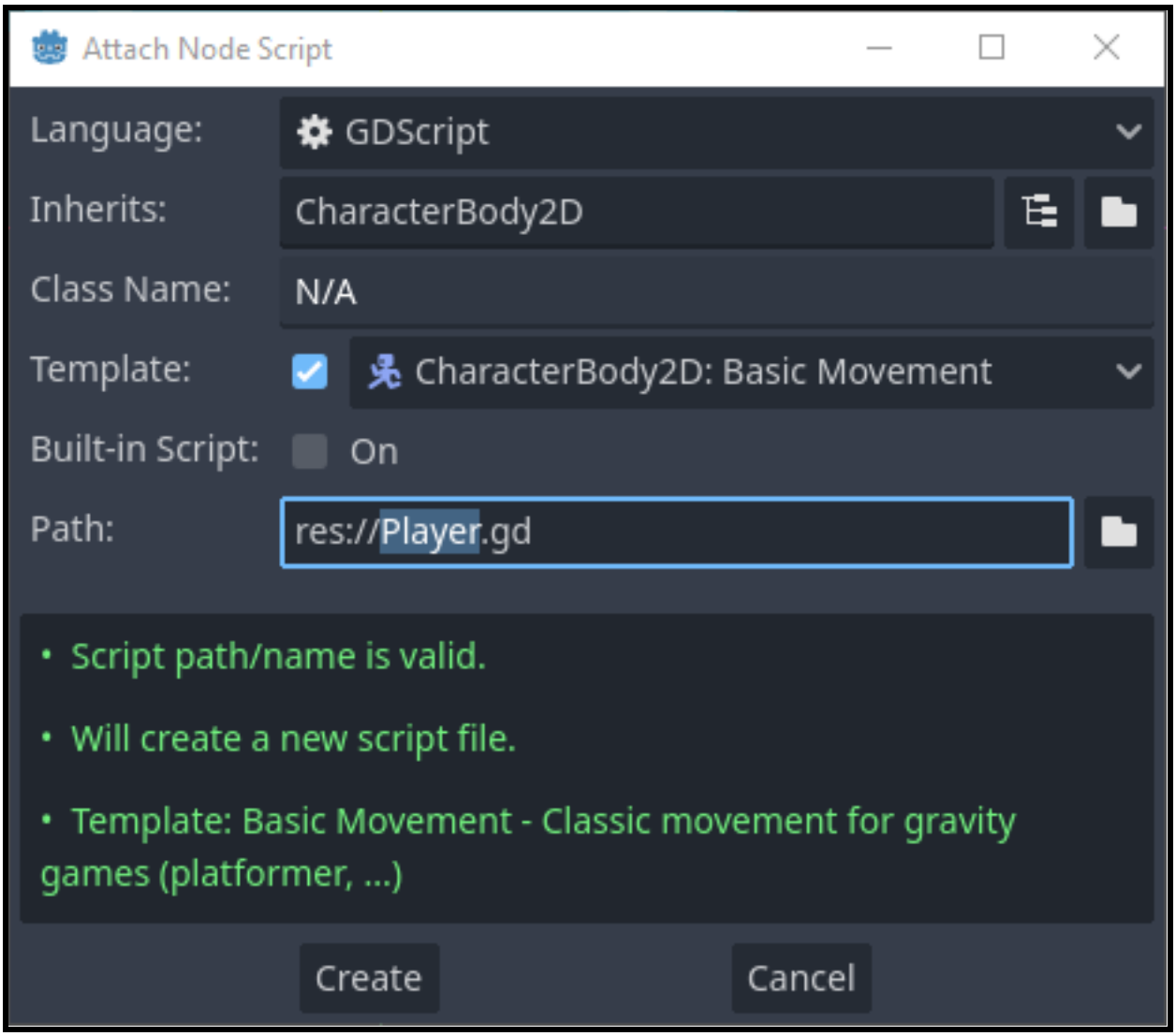
Moving the Player

To move the player, we need to create a script. This is basically a text file that contains code. The engine can read that and apply whatever gameplay behaviors we define.

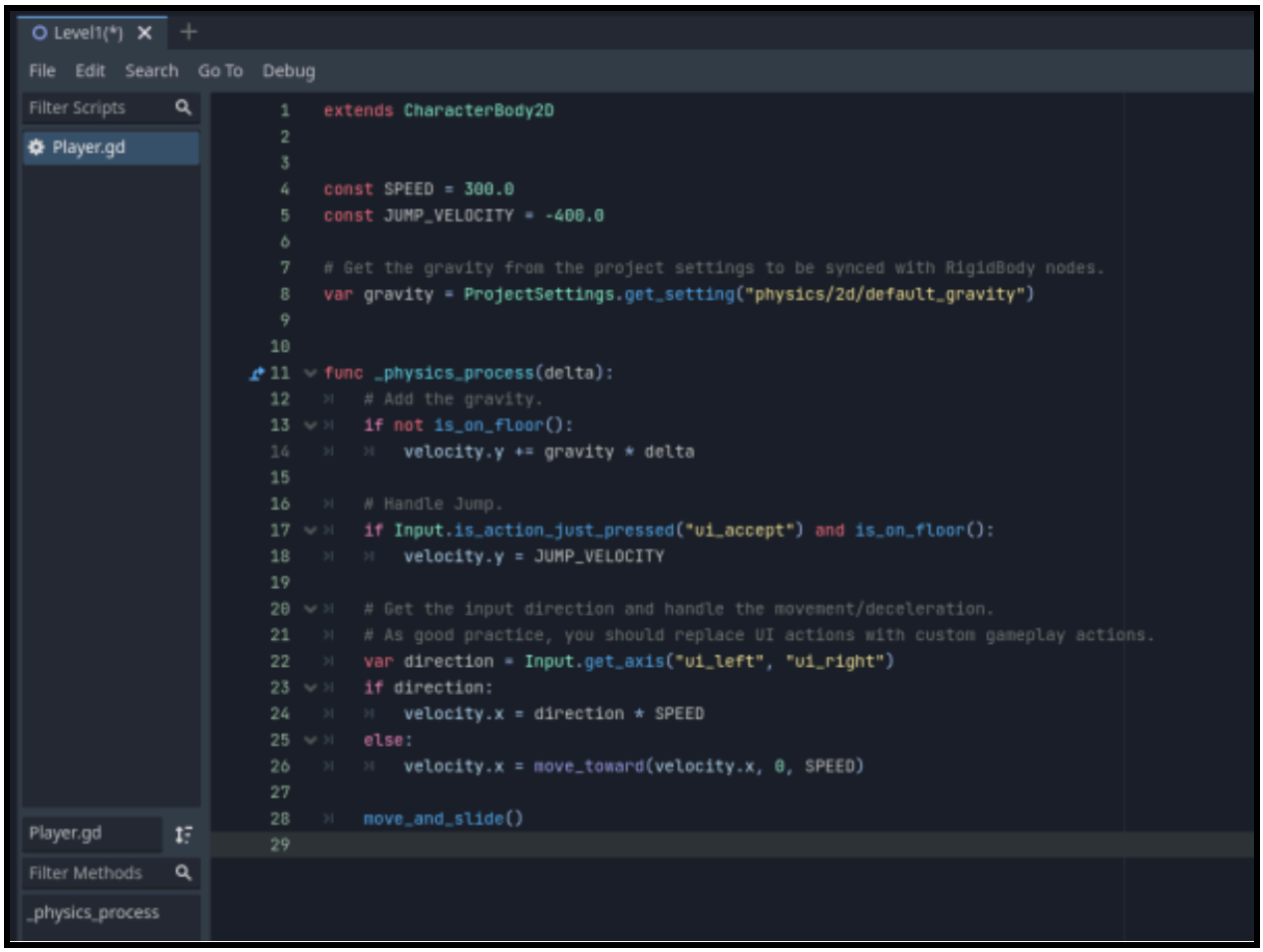
With the **Player** node selected, go down to the bottom of the inspector. Click on the down arrow next to Script and select **New Script**.



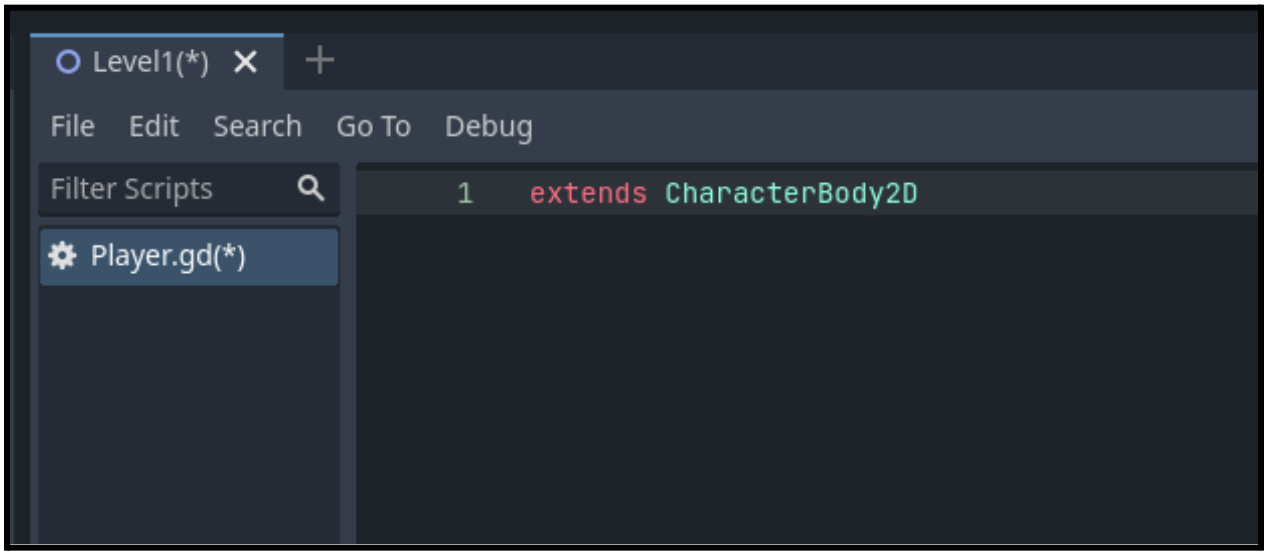
Call it **Player.gd** then click **Create**.



This will then switch us from 3D mode to Script mode and we'll see some default code here:



Delete everything except for the first line.



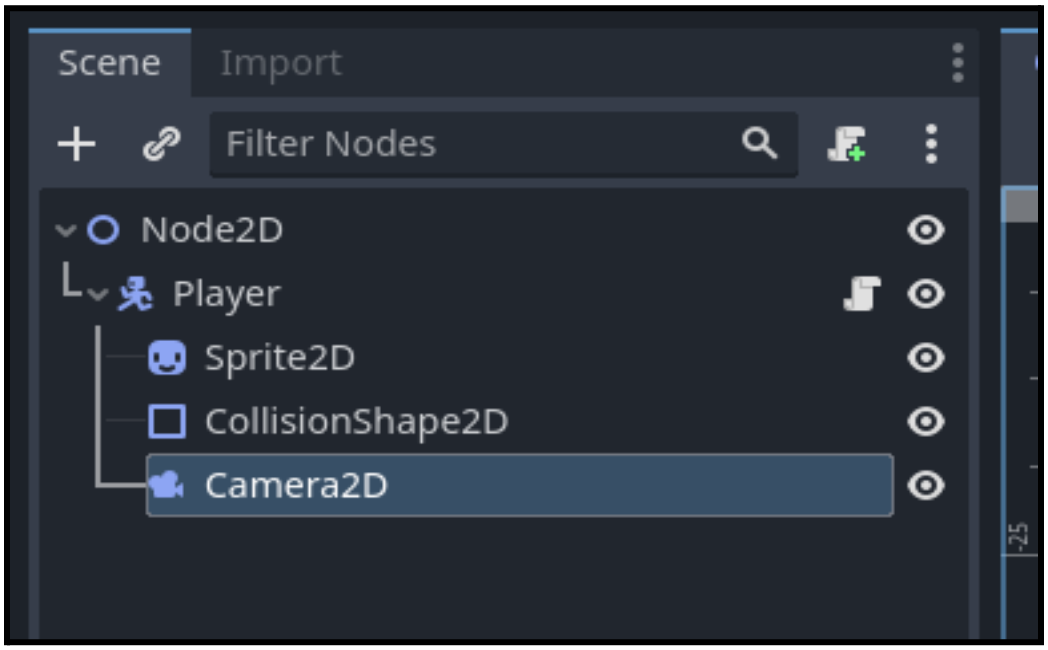
Now let's add in some code:

```
1.  func _physics_process(delta):
2.
3.      velocity = Vector2()
4.
5.      if Input.is_key_pressed(KEY_LEFT):
6.          velocity.x -= 1
7.      if Input.is_key_pressed(KEY_RIGHT):
8.          velocity.x += 1
9.      if Input.is_key_pressed(KEY_UP):
10.         velocity.y -= 1
11.     if Input.is_key_pressed(KEY_DOWN):
12.         velocity.y += 1
13.
14.     velocity *= 50
15.
16.     move_and_slide()
```

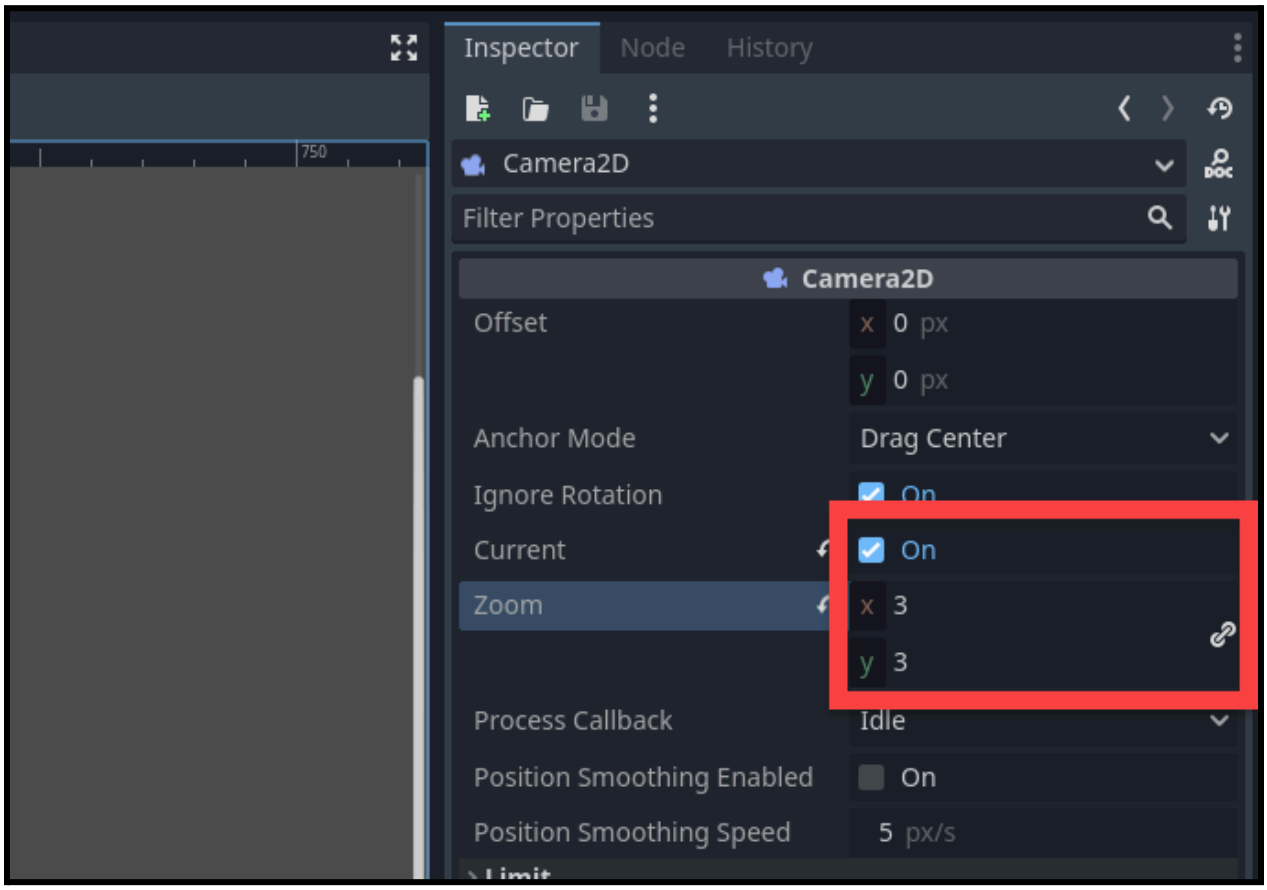
What we're doing here is detecting inputs from the arrow keys, then changing the value of our velocity based on that. At the end, we're multiplying it all by 50, so that we move at 50 pixels per second. And finally, move_and_slide will apply those changes to our CharacterBody2D.

If you now go to the top right corner of the screen and click on the **Play** button, you can test it out.

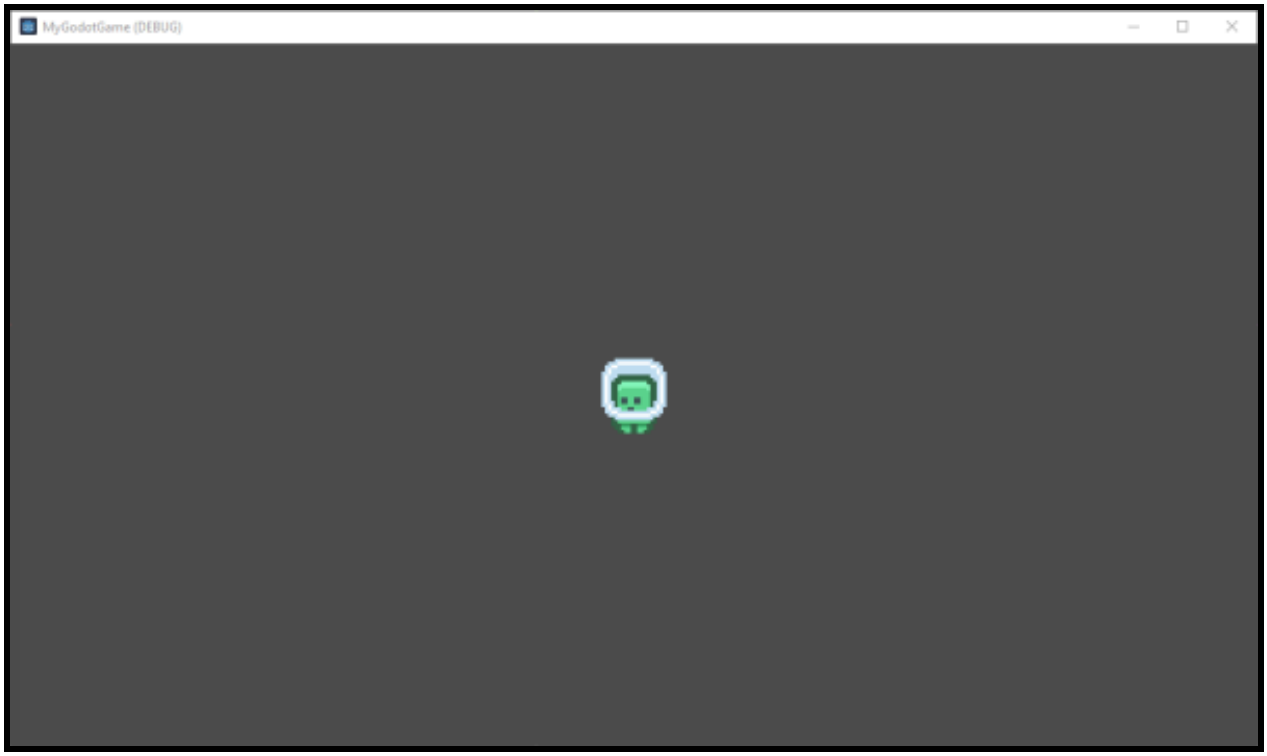
One thing you will notice, is that after you press play, a window will open. But our player is so small! And it's in the top left corner of the screen. To fix this, we can go to our player node and add a new child node of type **Camera2D**.



Then in the inspector, make sure to enable **Current**. And for the **Zoom**, set that to 3, 3. This will make it so the camera is closer to the player and we can get a much better view of them.



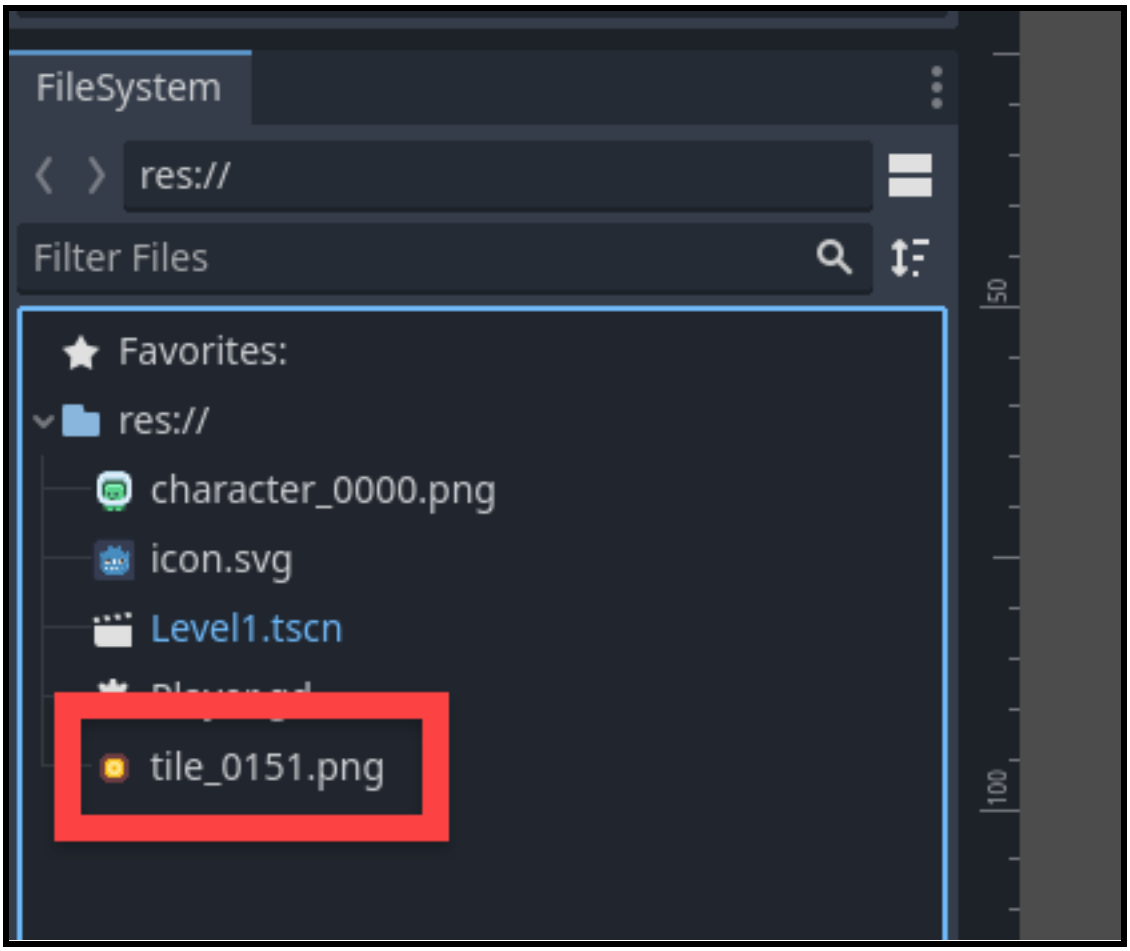
Now when you press play, it should look like this:



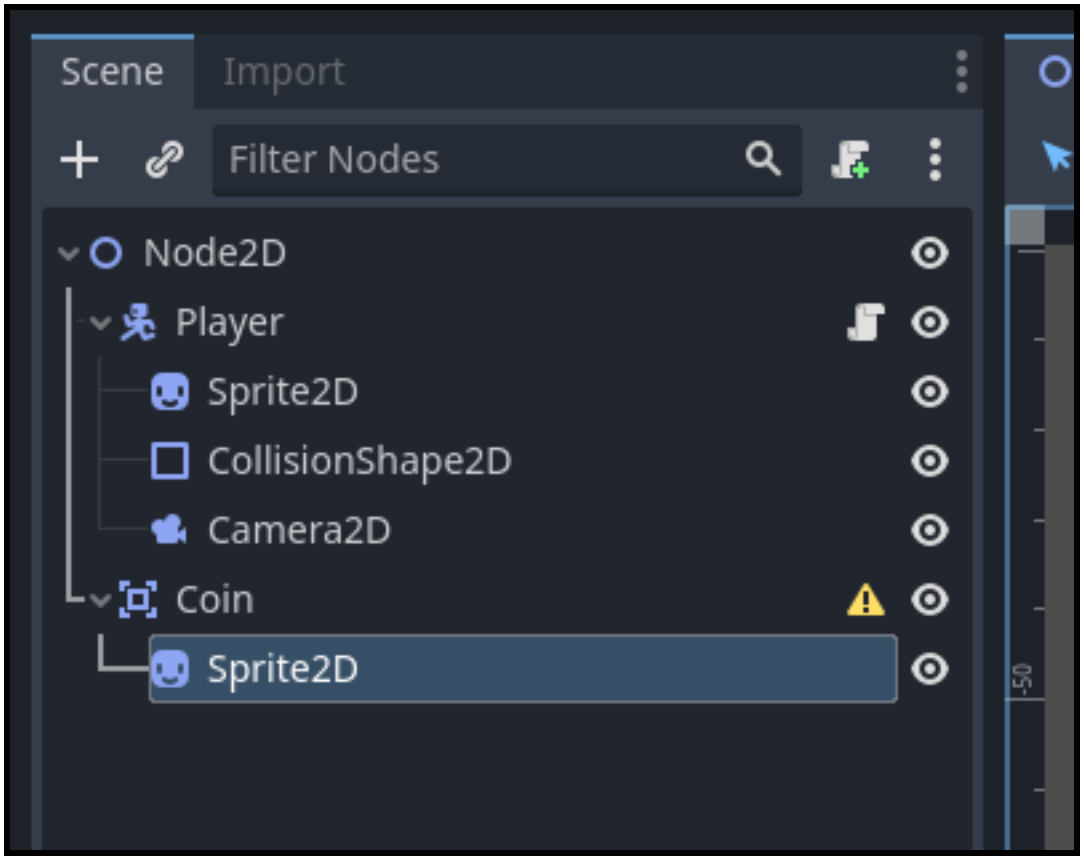
Collectibles

When we play our game, there's one problem. There's nothing to do! So let's add in some coins that the player can collect.

To begin, download the coin sprite and import it into your project.



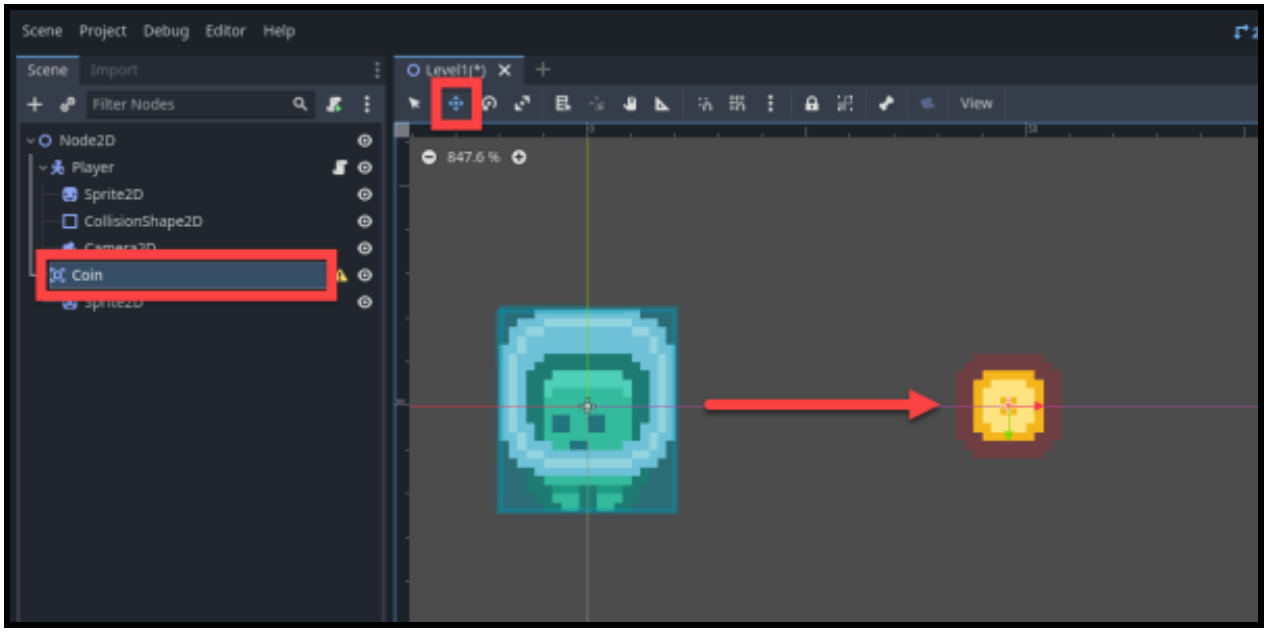
Next, in the Scene panel, click on the + and create a new node of type **Area2D**. Make sure to rename it to *Coin*. Then as a child of Coin, add a **Sprite2D** node.



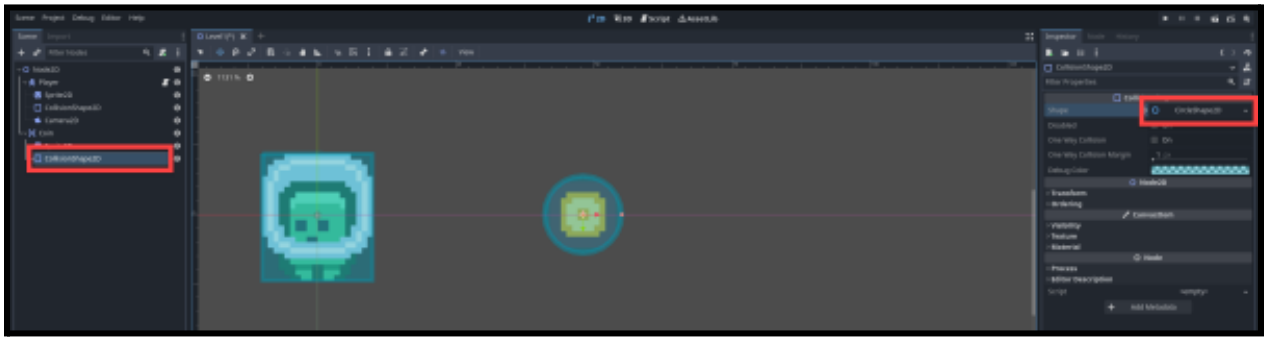
In the Inspector:

- Set the **Texture** to be the coin sprite.
- Set the **Position** to 0, 0.
- Set the **Filter** to *Nearest*.

Select the *Coin* node, then at the top of the scene window, select the **Move Tool**. This will allow us to click and drag on the coin to move it around our scene.



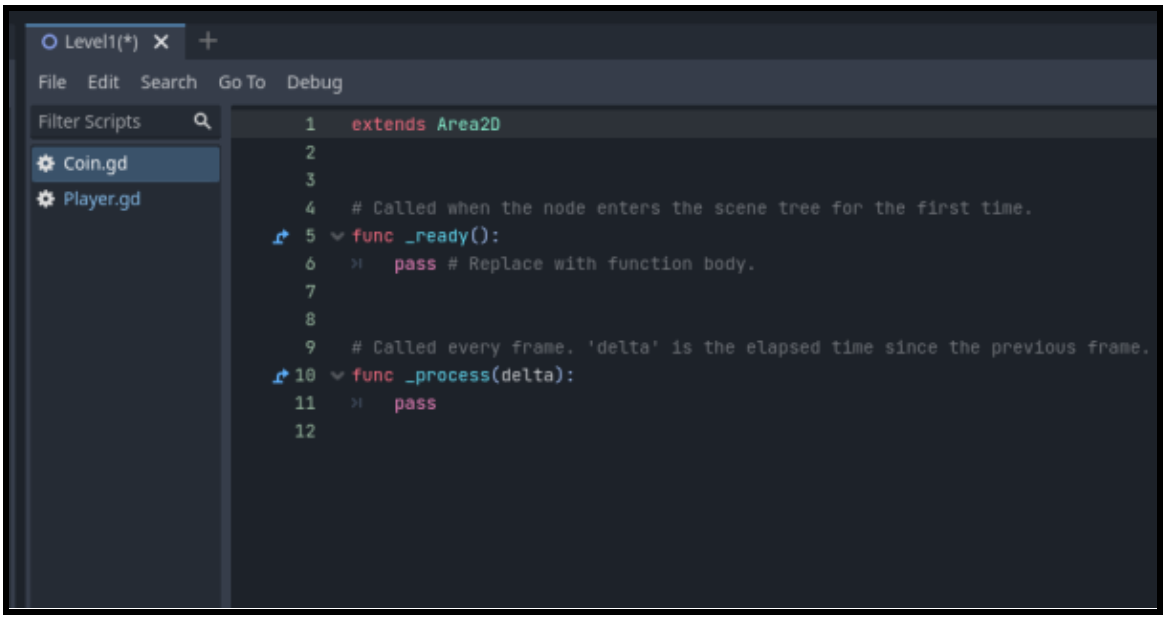
As a child of the Coin, create a **CollisionShape2D** node. Then in the Inspector, set the **Shape** to *CircleShape2D*. We can then click and drag on the orange circle to resize the collider.



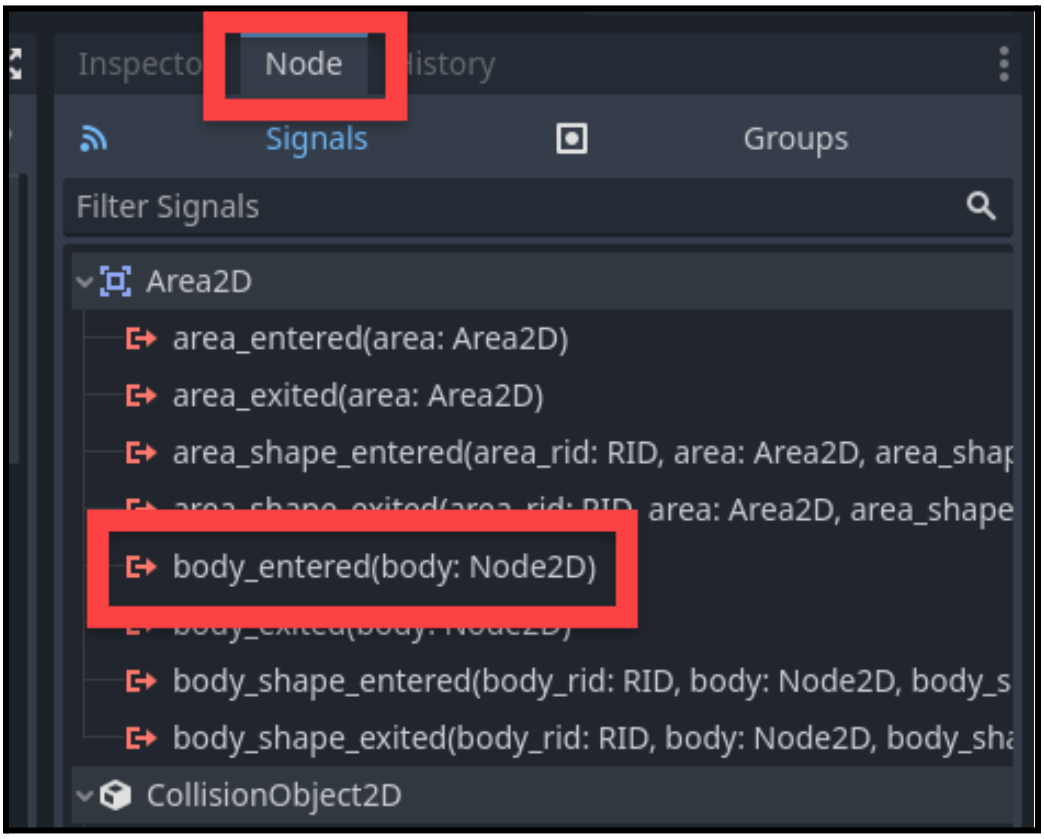
Finally, let's create a script for the coin.

1. Select the *Coin* node.
2. In the inspector, create a new script called *Coin*.

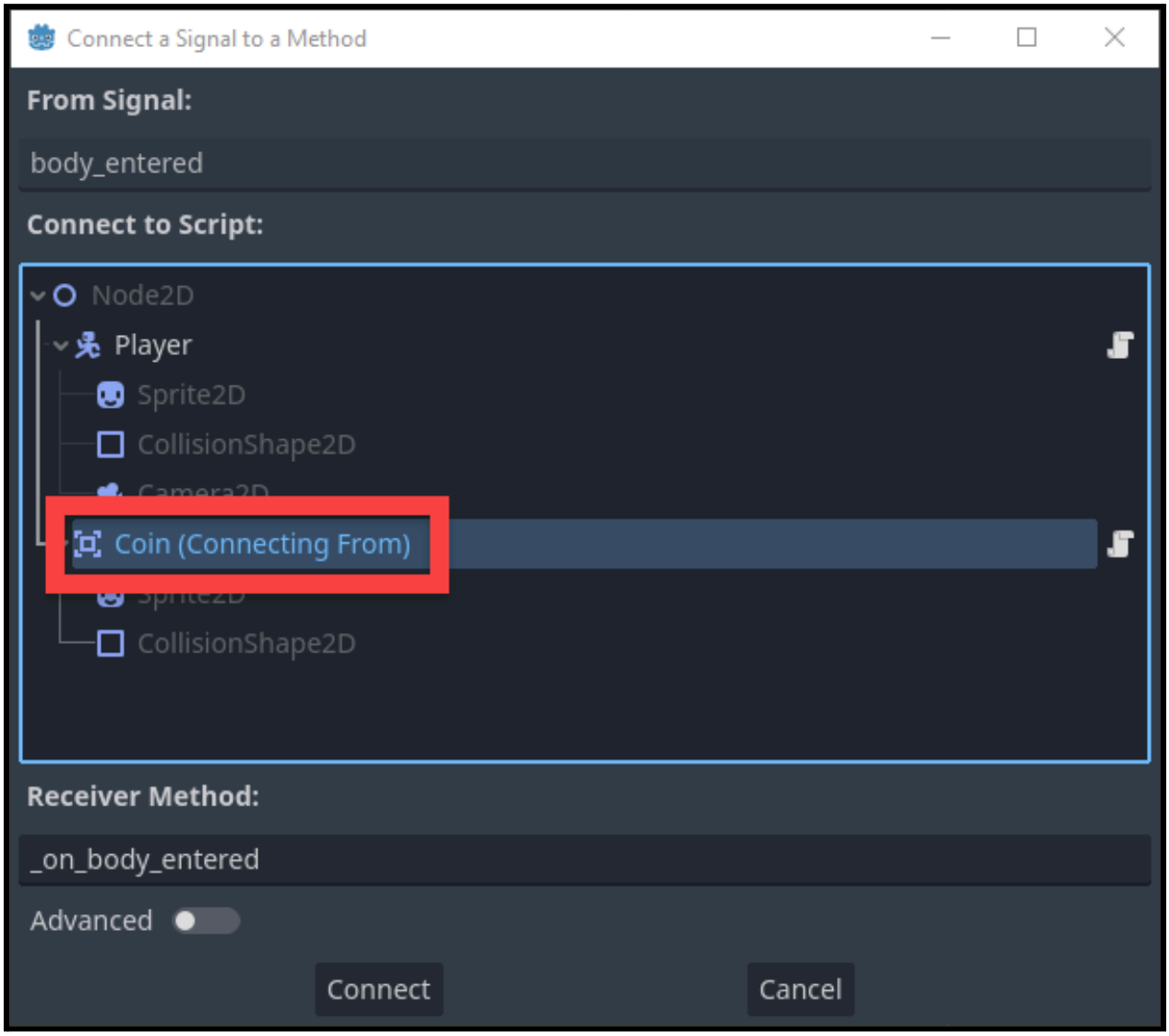
Here's what our Script panel should look like now.



With the Coin selected, go over to the inspector and swap over to the **Node** tab. Here, we want to double-click on *body_Entered(body: Node2D)*.



In this window, select the Coin and click **Connect**. What we're doing here is basically setting it up so that the coin script gets informed when a body hits the coin collider.



This new bit of code should then be added to the script automatically.

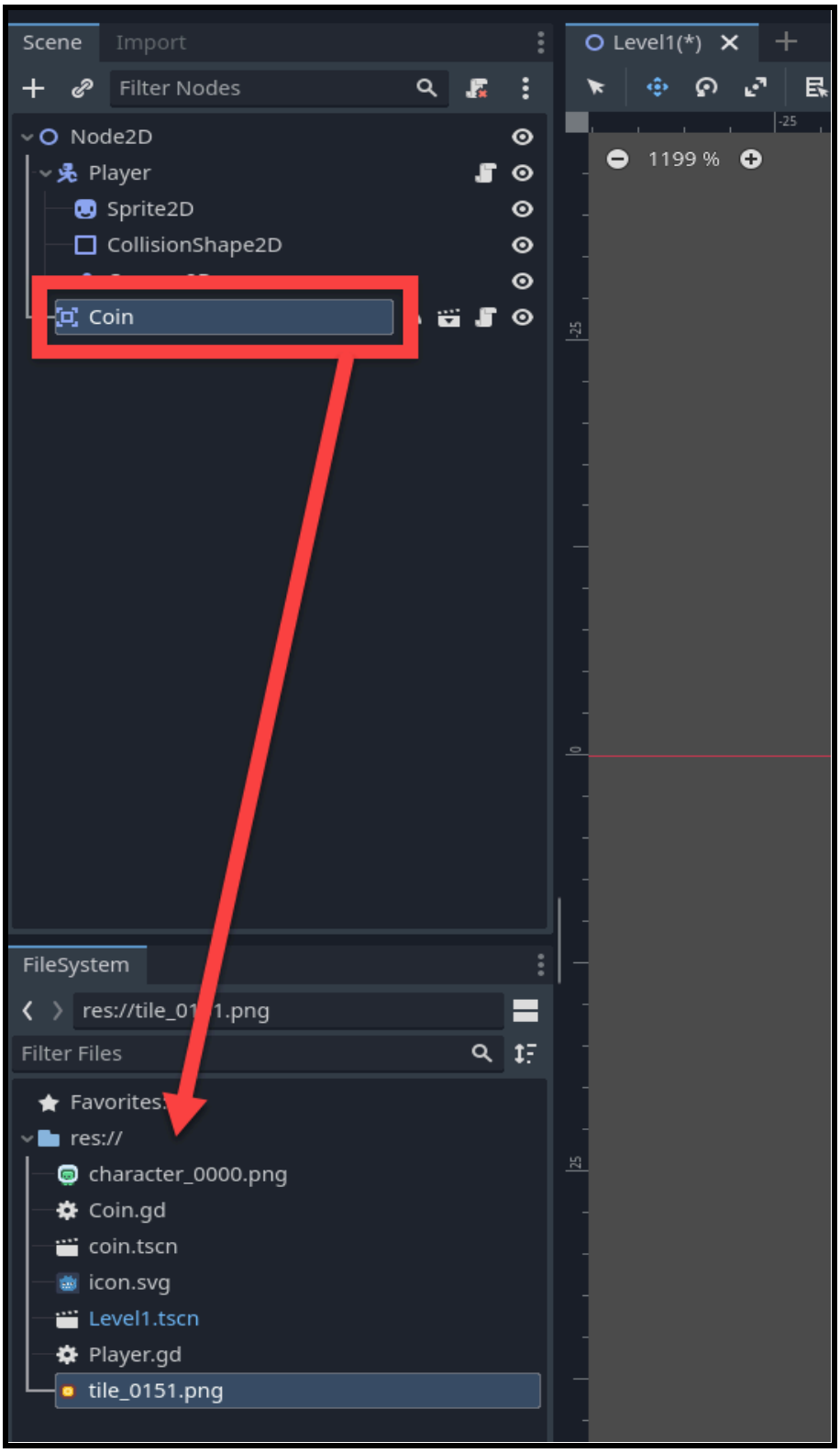
```
13
14 func _on_body_entered(body):
15     pass # Replace with function body.
16
```

Delete the “pass” line of code and add these three new lines:

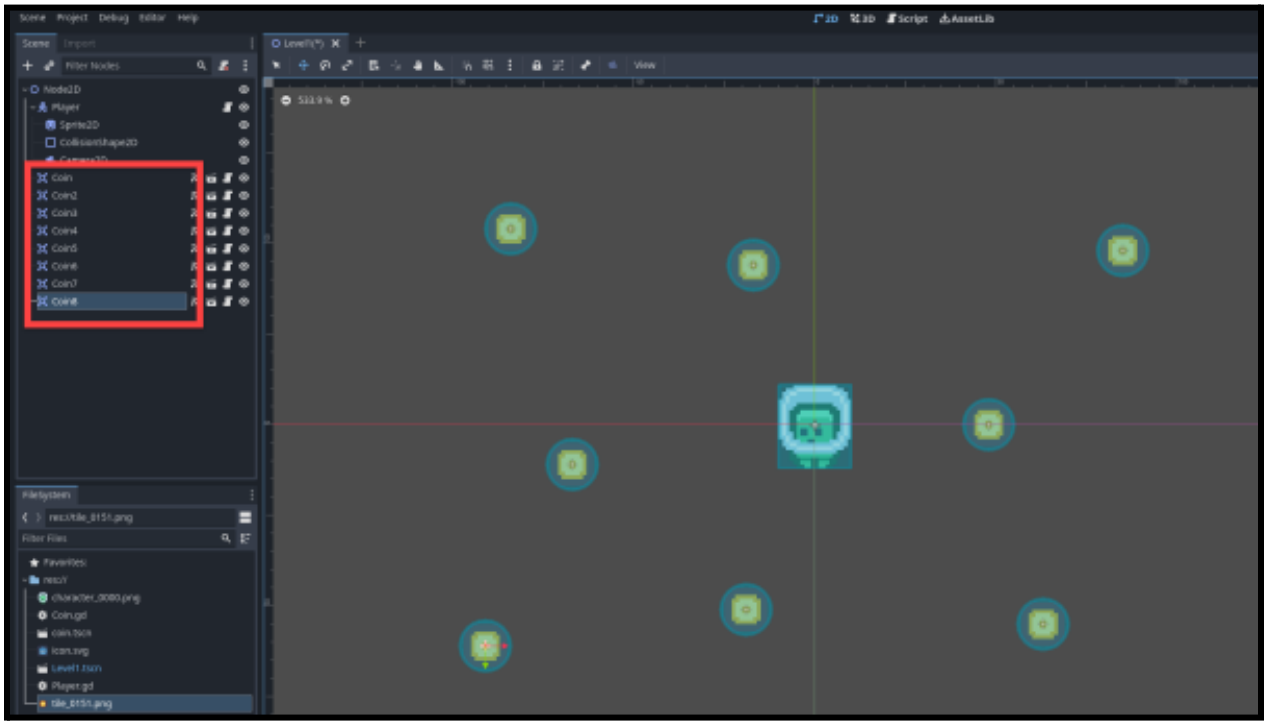
```
1. func _on_body_entered(body):
2.     body.scale.x += 0.2
3.     body.scale.y += 0.2
4.     queue_free()
```

What we’re doing here, is increasing the size of the colliding object (player) when they hit the coin. Then with *queue_free()*, we are destroying the coin so the player can only collect it once.

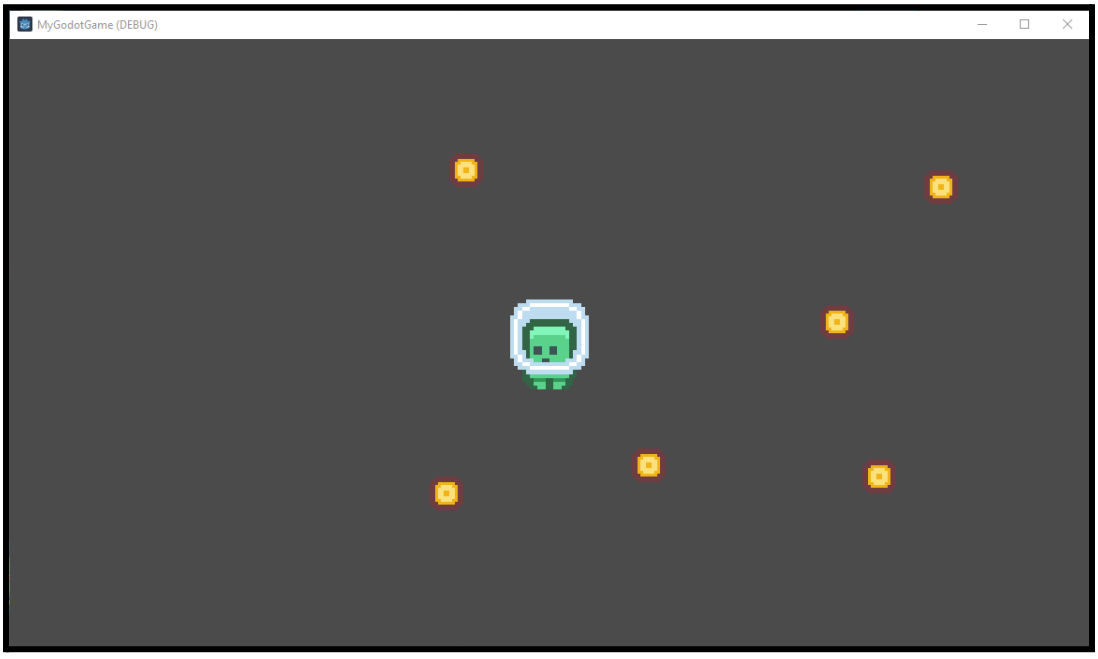
Back in our **2D** panel, let’s make some more Coins. The best practice is to turn it into a scene. Much like how our level is saved as *Level1.tscn*, we are going to make our coin a scene. This means we can have many different instances of it in our scene which all inherit from the same blueprint. So to do this, click and drag the coin node into the **FileSystem**. Call it coin and click save.



Now we can duplicate the coin *(Ctrl + D)* and move them around like so.



Press play and we can test it out!



Conclusion

And there we go! We've made a small, introductory game with the Godot game engine. While this game is very simple, it does serve to not only show you what Godot 4 is all about, but some of the core tools you'll be using in any Godot game project.

From here, you can expand upon our project, or create an entirely new game with the knowledge that you've learned. Perhaps you'll want to turn this into a platformer with tons of jumping mechanics. Or maybe you want to make a bullet hell or even a strategy game. There is a ton left to explore, but hopefully these foundations will help you in taking that next step.

If you're interested in learning more about the specifics with Godot, then you can read their documentation here. Thanks for following along, and good luck with your future Godot 4 games!

How to Create a Skiing Mini-Game in GODOT – Godot 4 Tutorial

In this tutorial, you'll be building a simple skiing game using Godot's 3D collision system. We will design our scene step by step, starting with setting up the ski slope, creating a player character, and adding some trees as obstacles to avoid. You'll learn how to create collidable objects, handle collisions between objects in the game, and reload the scene upon a specific event, such as hitting a tree while skiing down the slope.

To follow this tutorial, you are expected to have a basic understanding of the following concepts:

- Godot game engine
- GDScript programming
- Creating and manipulating 3D models in Godot

Project Files

While you'll need to set up your own Godot project, we have included the asset files used in this tutorial. For the tree models, you can choose to create your own or use simple 3D shapes available in Godot.

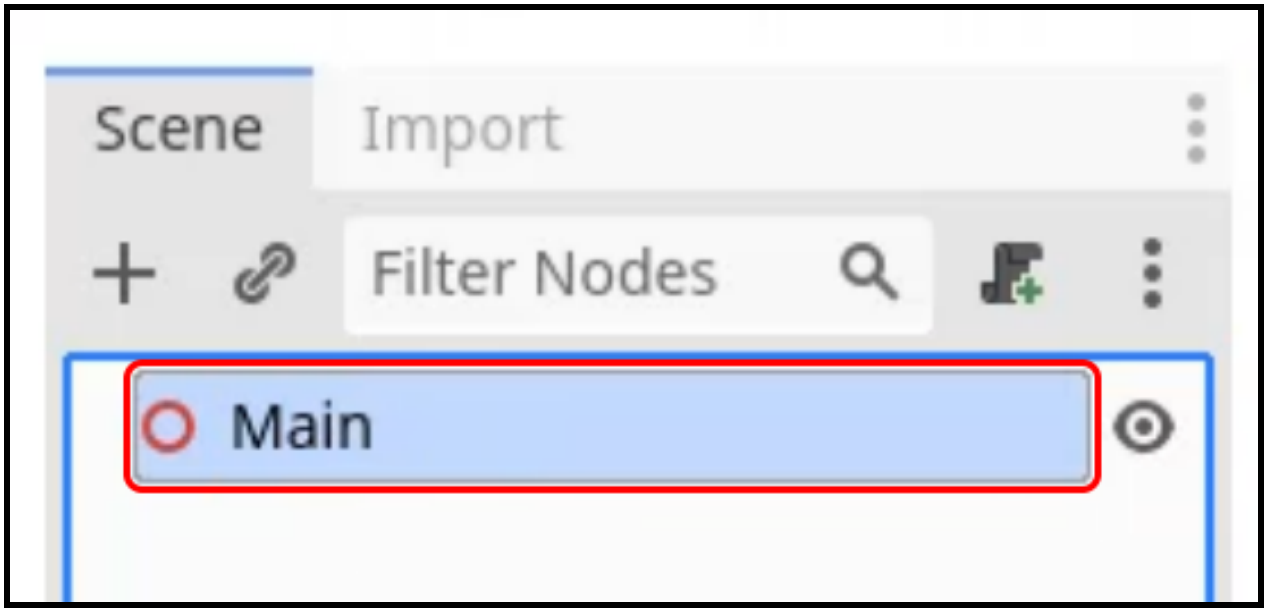
Download Project Files Here

Collision – Part 1

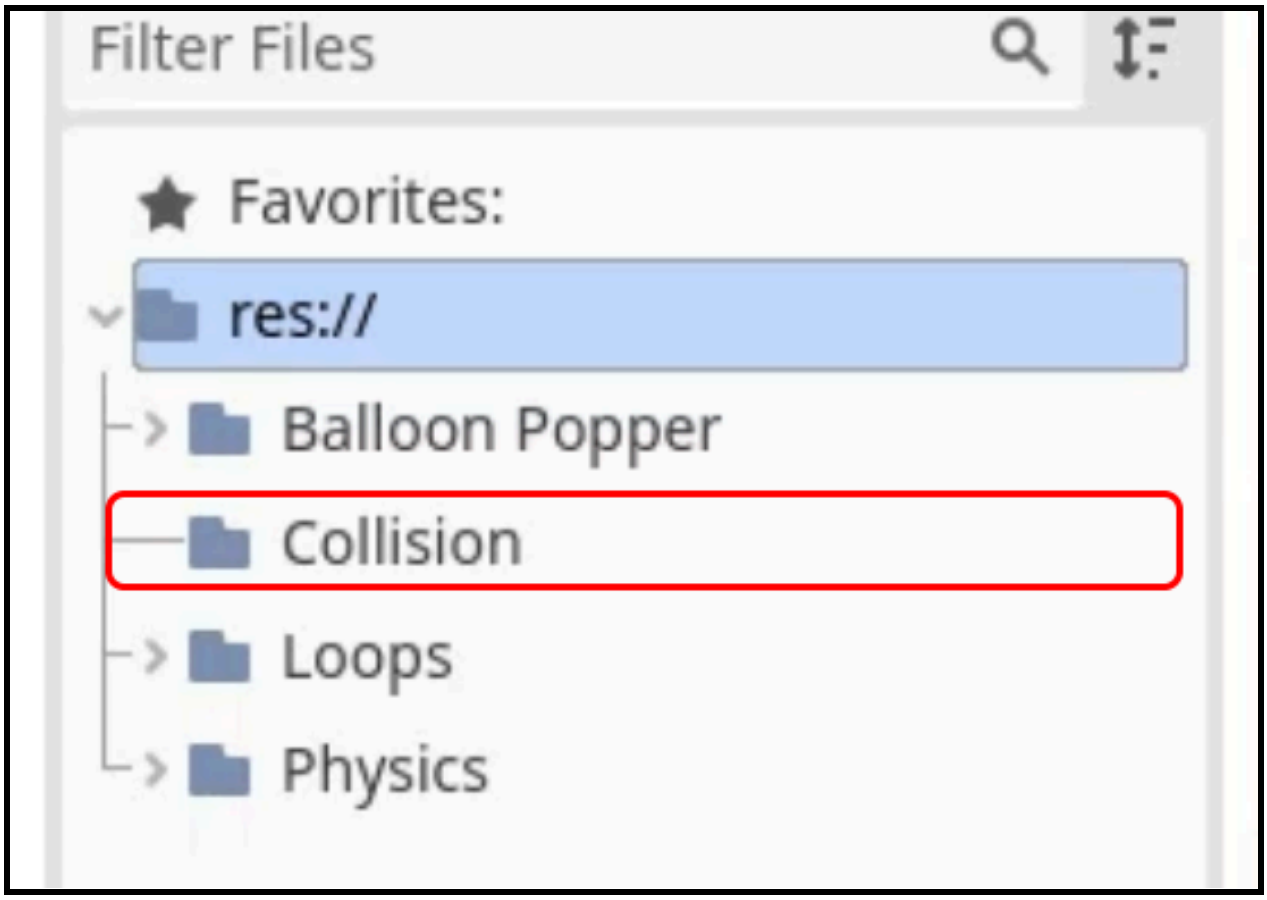
In this lesson, we are going to start the fourth mini-game project which will focus on Godot's collision interactions using a skiing game.

Setting up the Scene

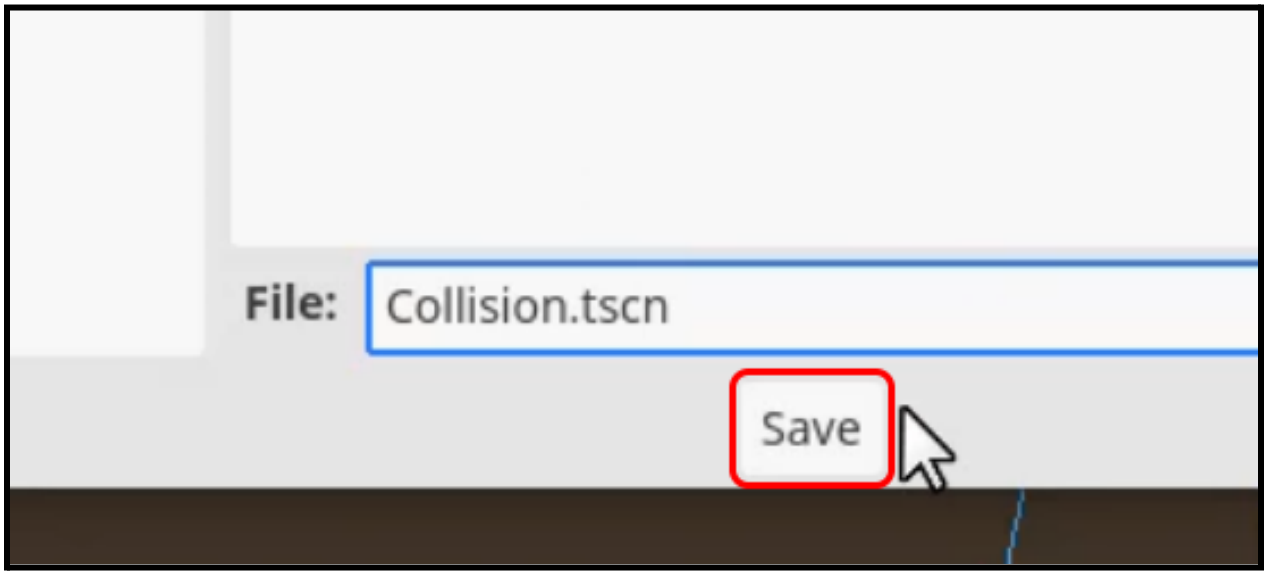
As with the other mini-projects, we will start by creating a new scene. This game will be a **3D** project, so we will use a 3D root node, which we will name *Main*.



We will create a new folder called *Collision* for this project.



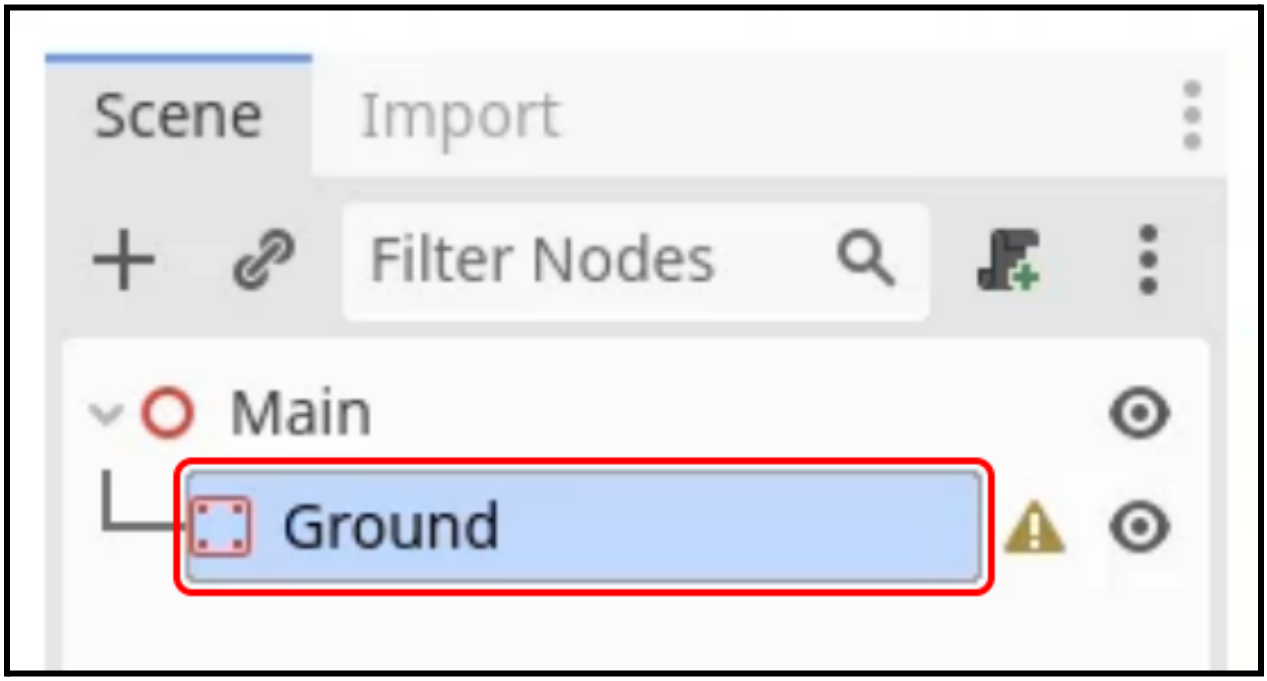
Inside the new folder, we will save the scene as *Collision.tscn*.



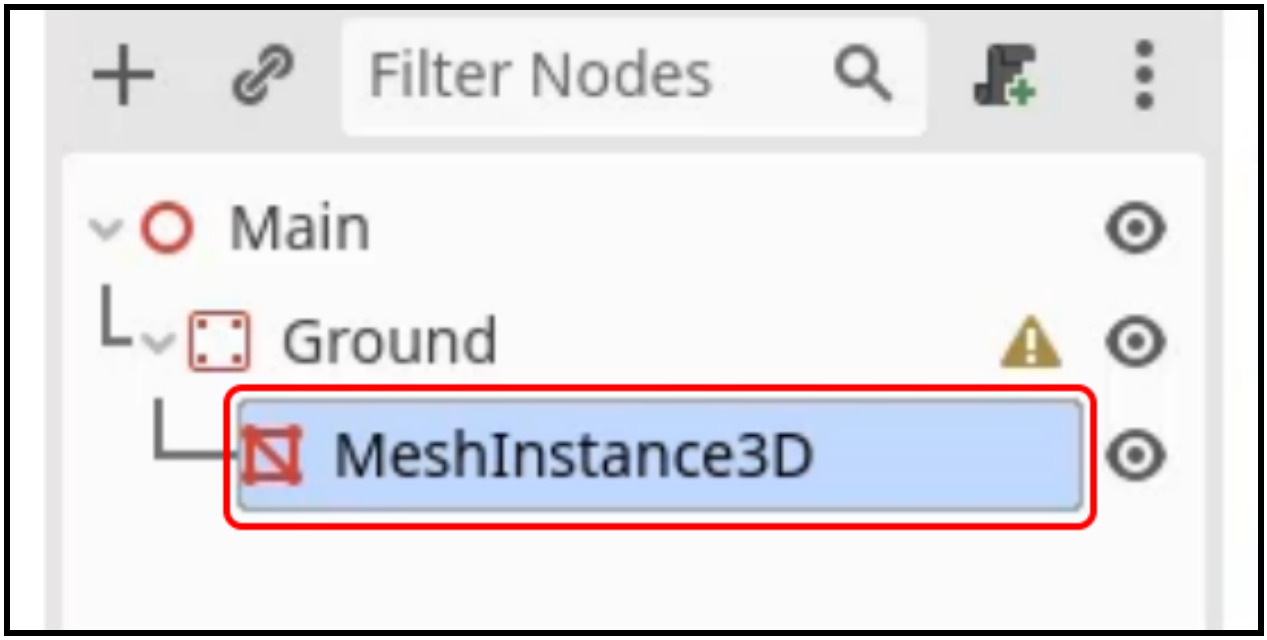
Creating the Ski Slope

We are going to begin by creating our *Ski Slope*. This will be a **StaticBody3D** node, which is used for having physical, static objects in a 3D game world.

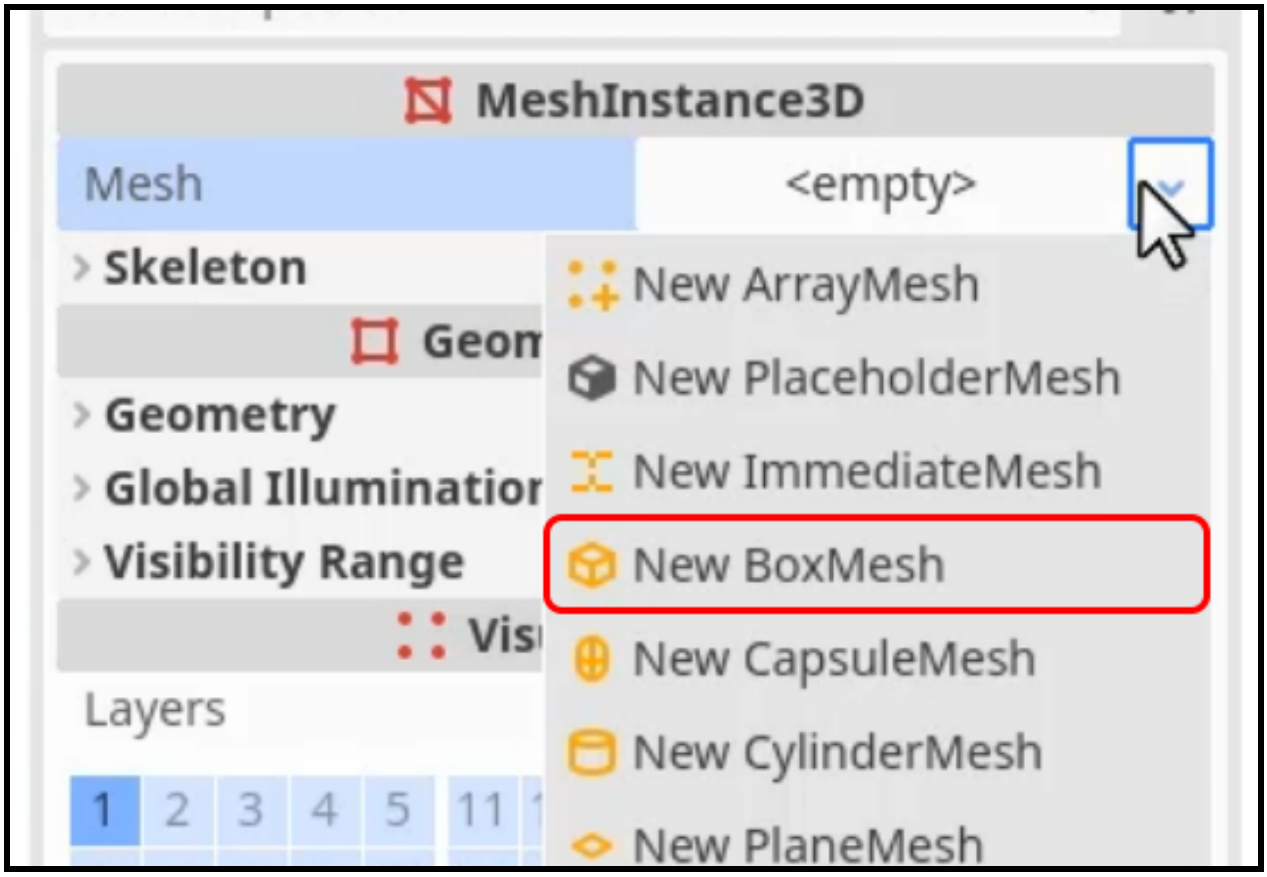
We can then rename this to *Ground* to represent its use in the scene.



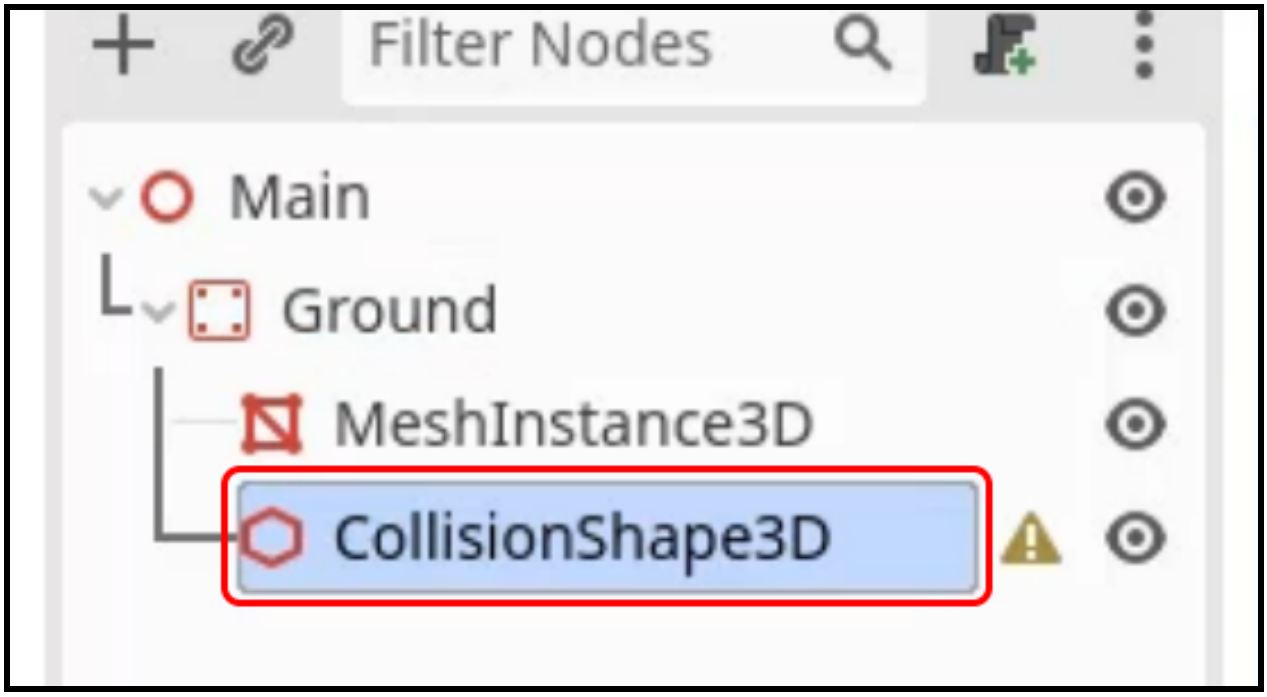
Next, we will add a **MeshInstance3D** node as a child of our *Ground* node.



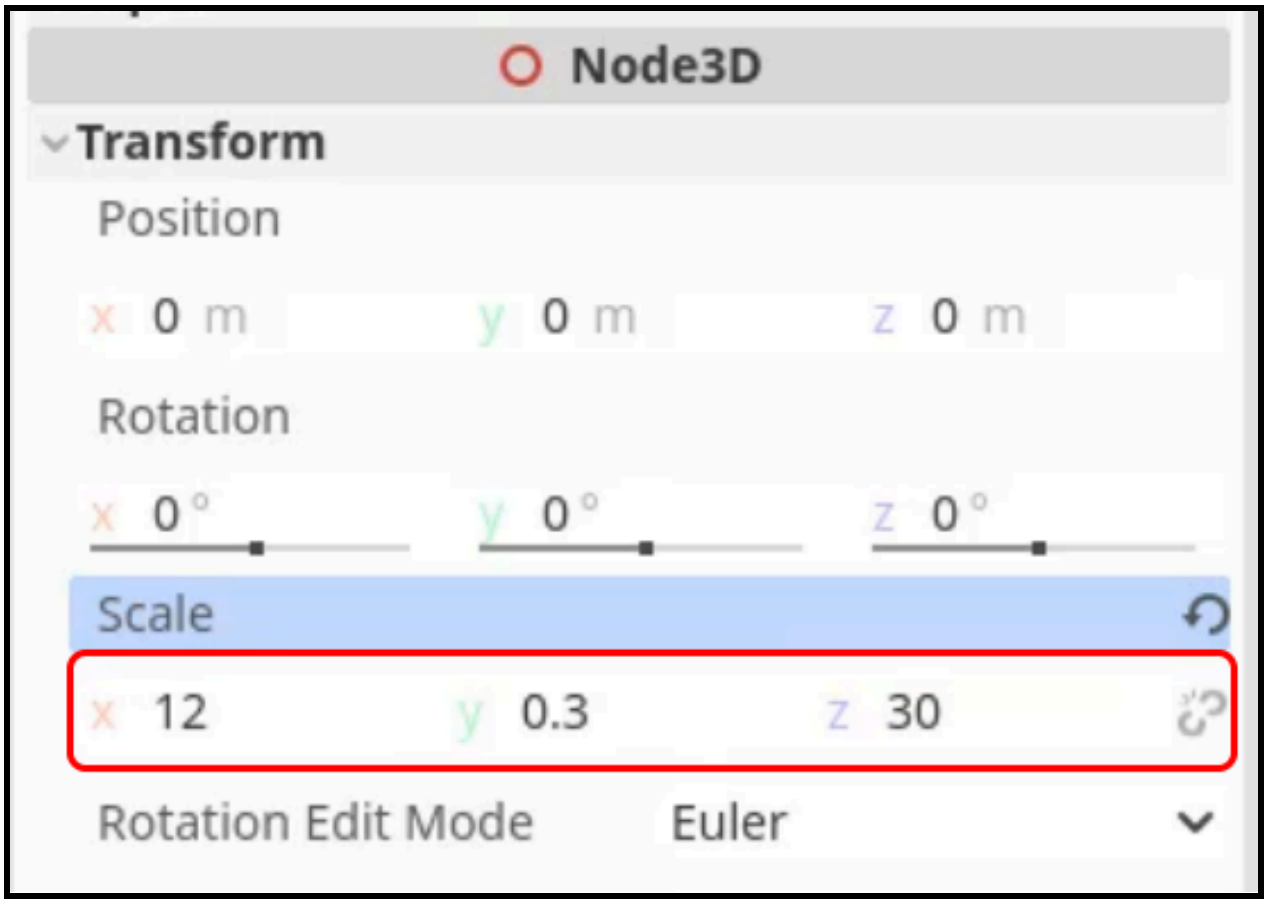
This will be used to render the mesh to the screen, and we will use a **New BoxMesh** as the value for the **Mesh** property**.**



We will then add a **CollisionShape3D** to match this.



For the **Shape** property, we will use a **New BoxShape3D** value to match the *MeshInstance3D* node. We then need to change the size of the cube. Select the static body parent, and select the **Scale Tool** (**R**). Scale the cube on the X-axis to 12, the Y-axis to 0.3, and the Z-axis to 30.

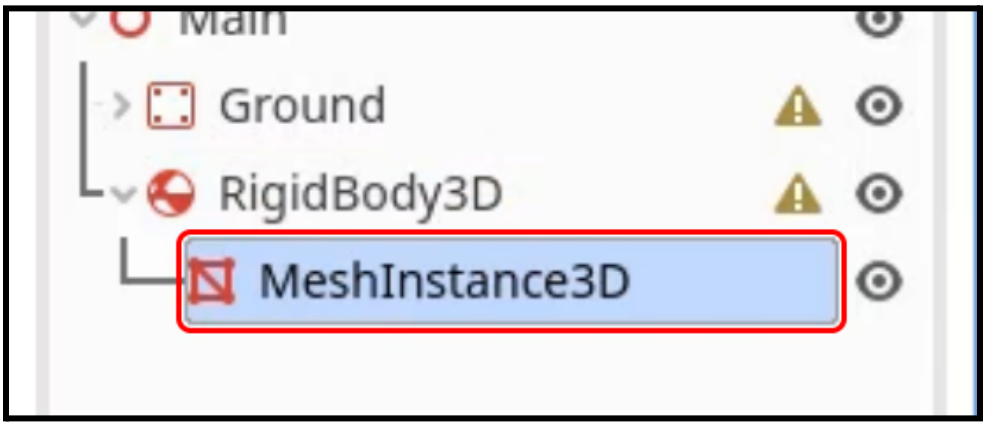


Then select the **Rotate Tool** (**E**), and rotate the *Ground* node around -20 on the X-axis.

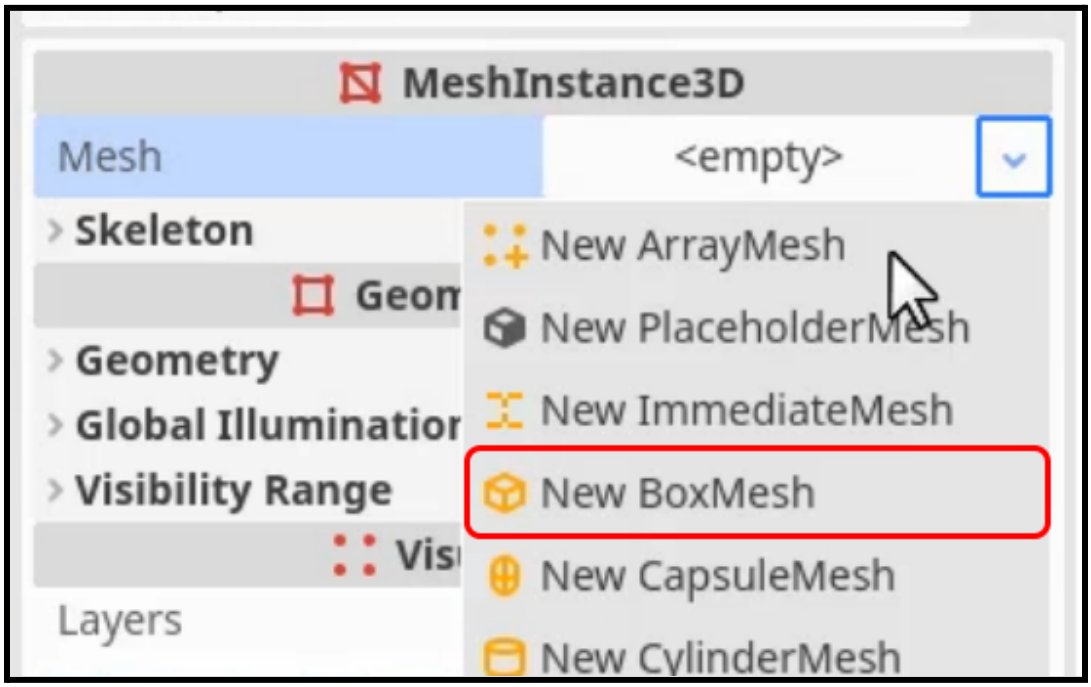
Creating the Player

We now need to create our player. For the parent of our *Player,* we will use a **RigidBody3D** node.

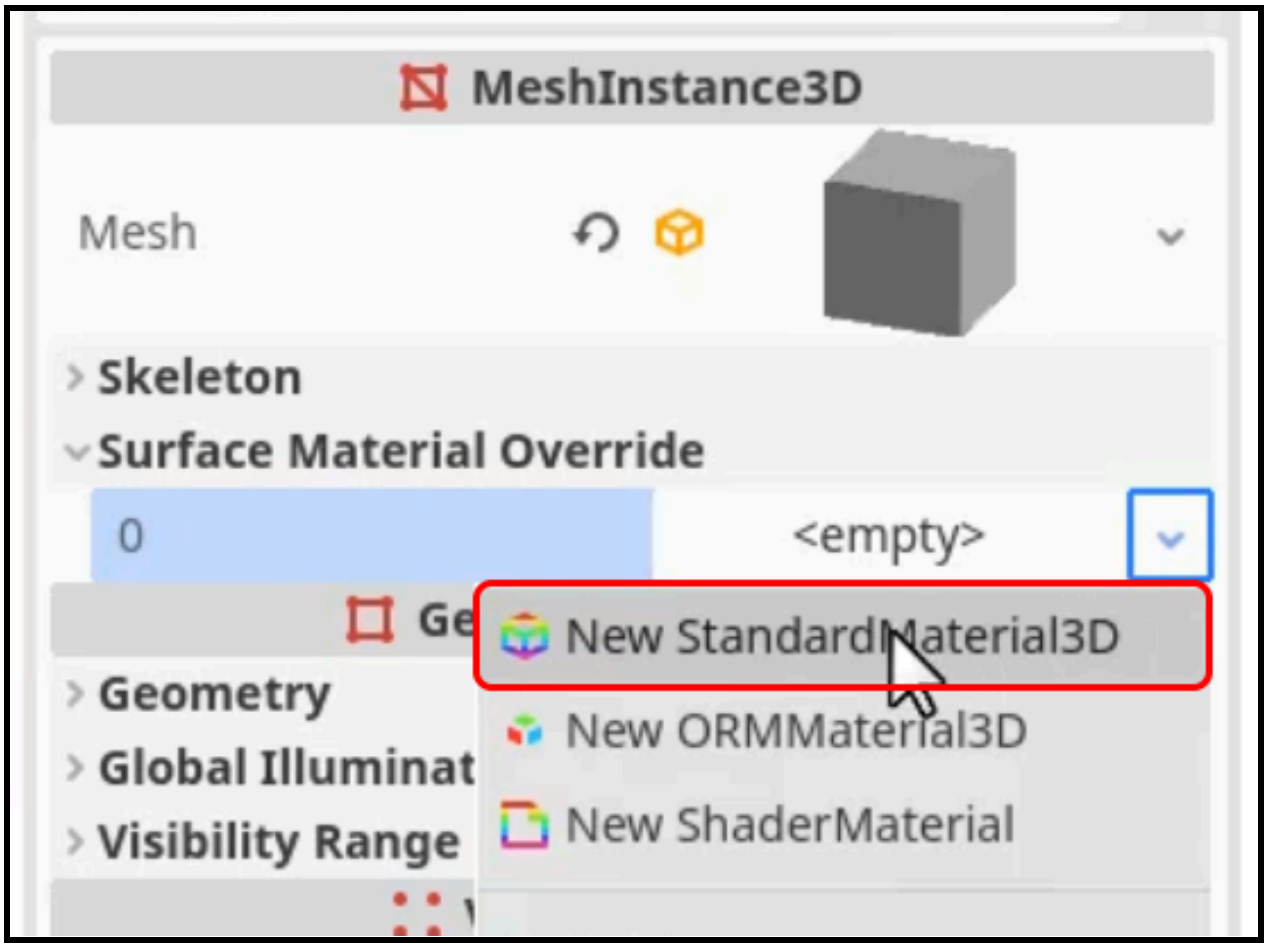
We will then add a **MeshInstance3D** node as a child so that we can see it.



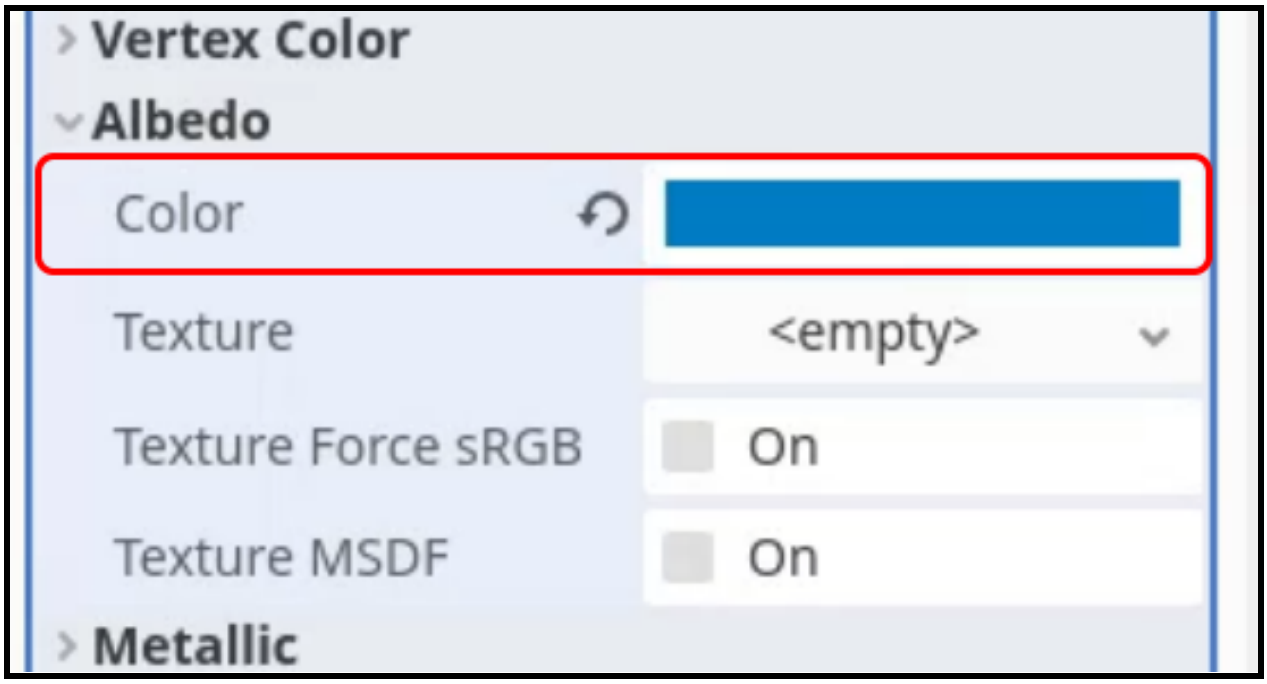
The **Mesh** property will be a **New BoxMesh**.



We will be using multiple *MeshInstance3D* nodes to create a model of our player. For this cube, we want to change the color by adding a **New StandardMaterial3D** in the **Surface Material Override** of the mesh instance.



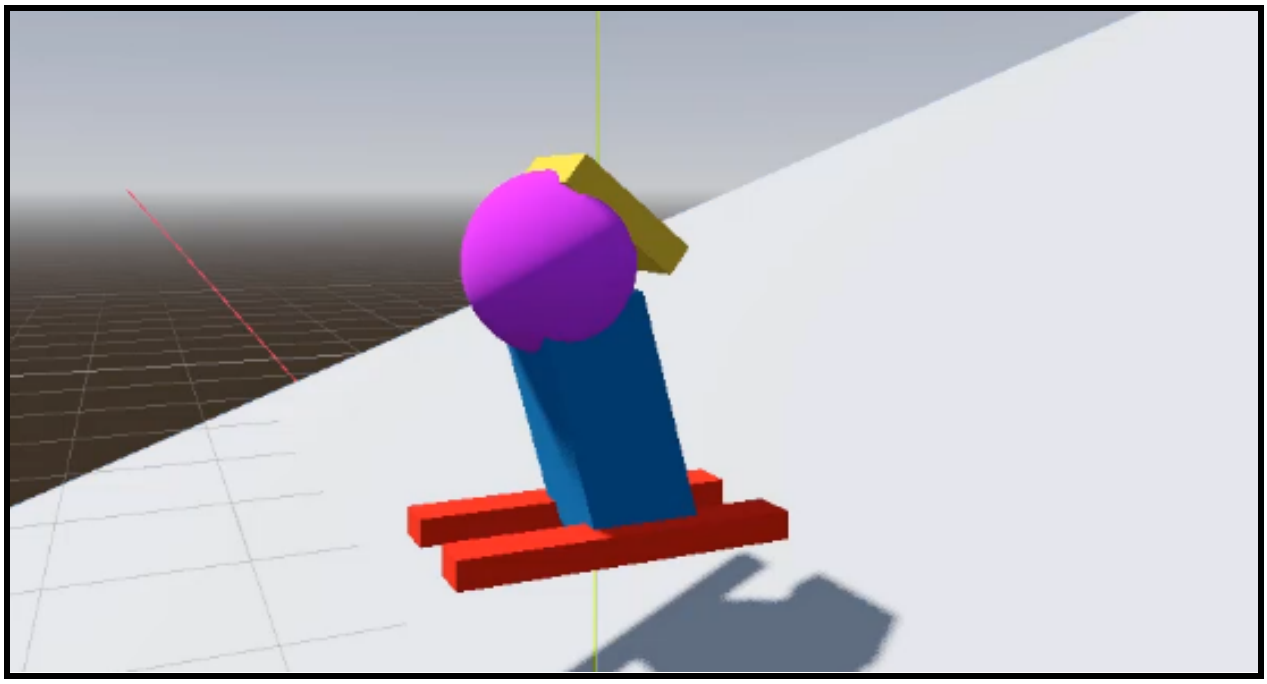
From here we will change the **Albedo Color** to be a color you like, we will use blue.



From here you can duplicate, resize, rotate, and move the cubes to design your own player character model.

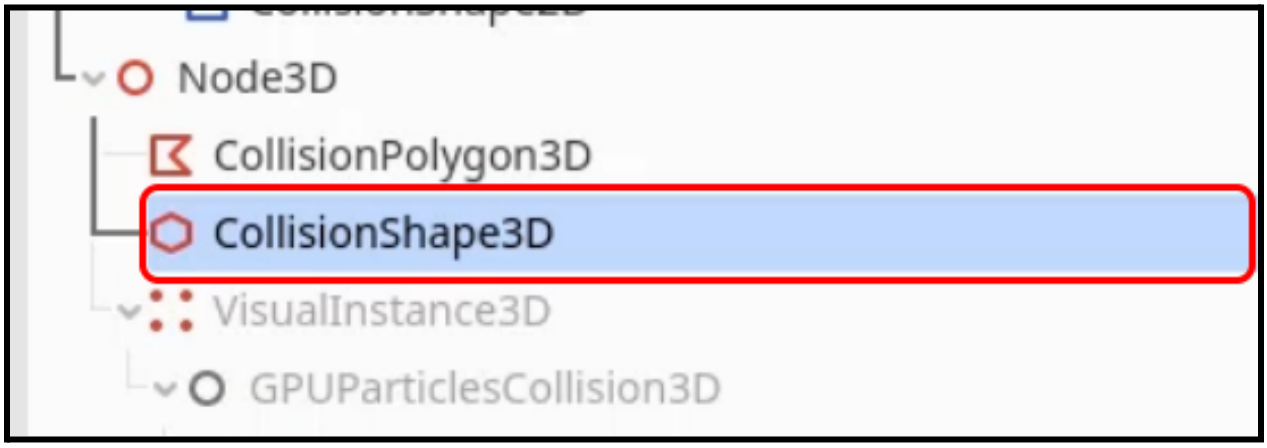
Collision – Part 2

In this lesson, we will be setting up our character's movement and environment in the game engine Godot. Before doing this, however, in the last lesson, we set the challenge to create a character model using different mesh instances with their own materials. Here is the example we made using five different *MeshInstance3D* nodes to create our skiing character.

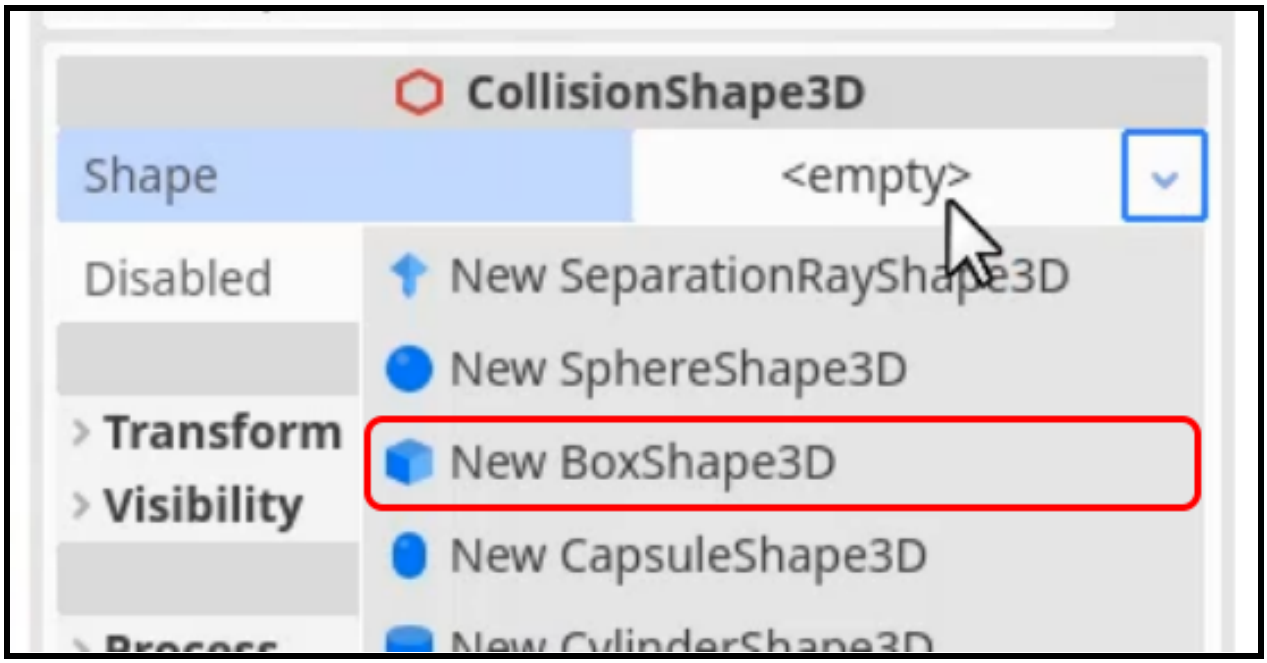


Creating the Player

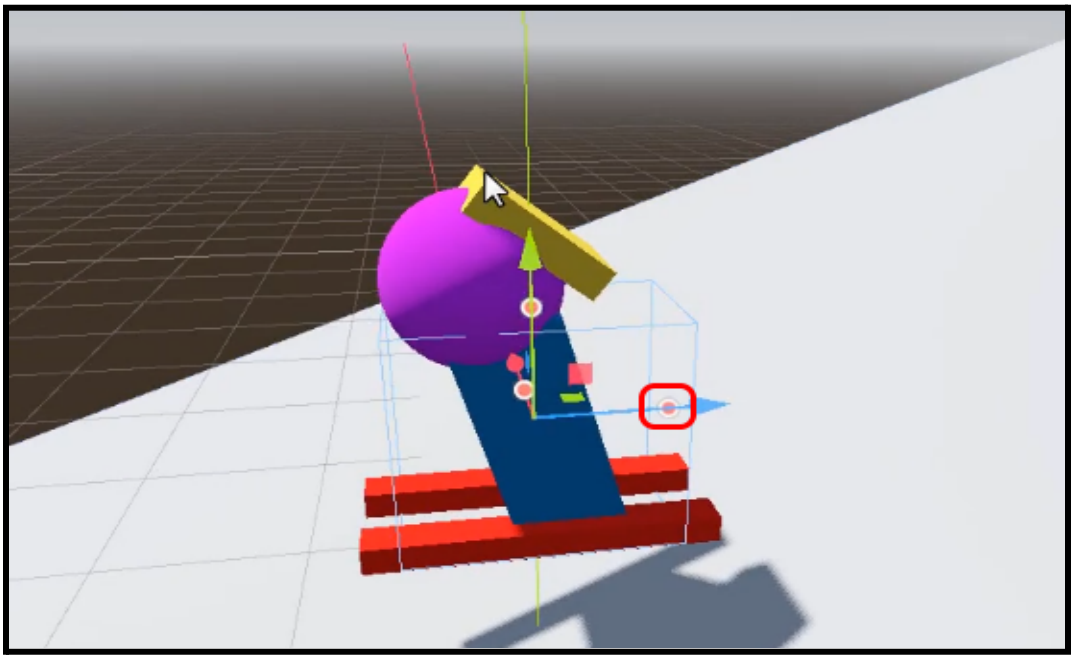
First, we will create a **CollisionShape3D** as a child of the *RigidBody3D* node.



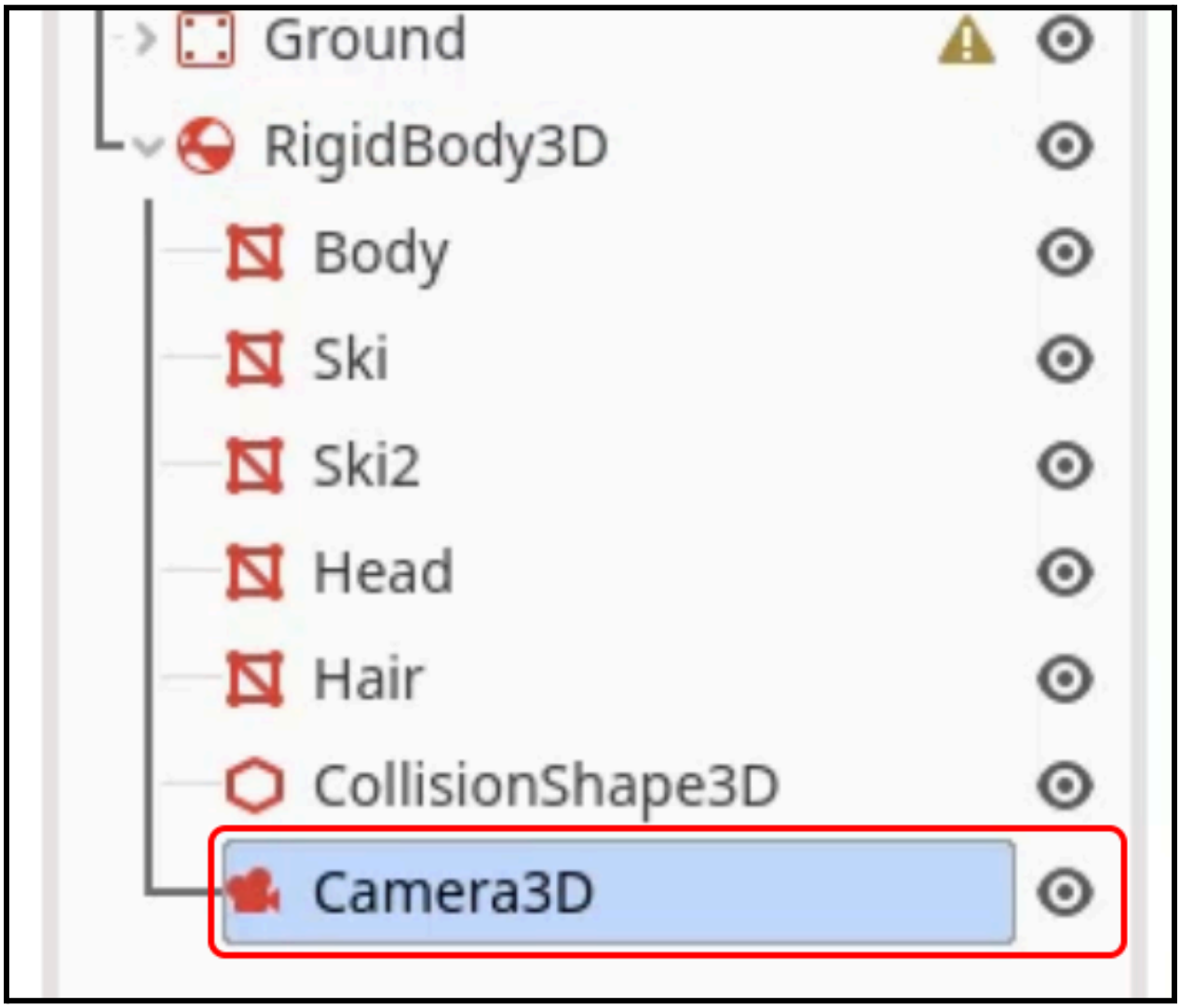
We will make the **Shape** property a **New BoxShape3D** to match our character.



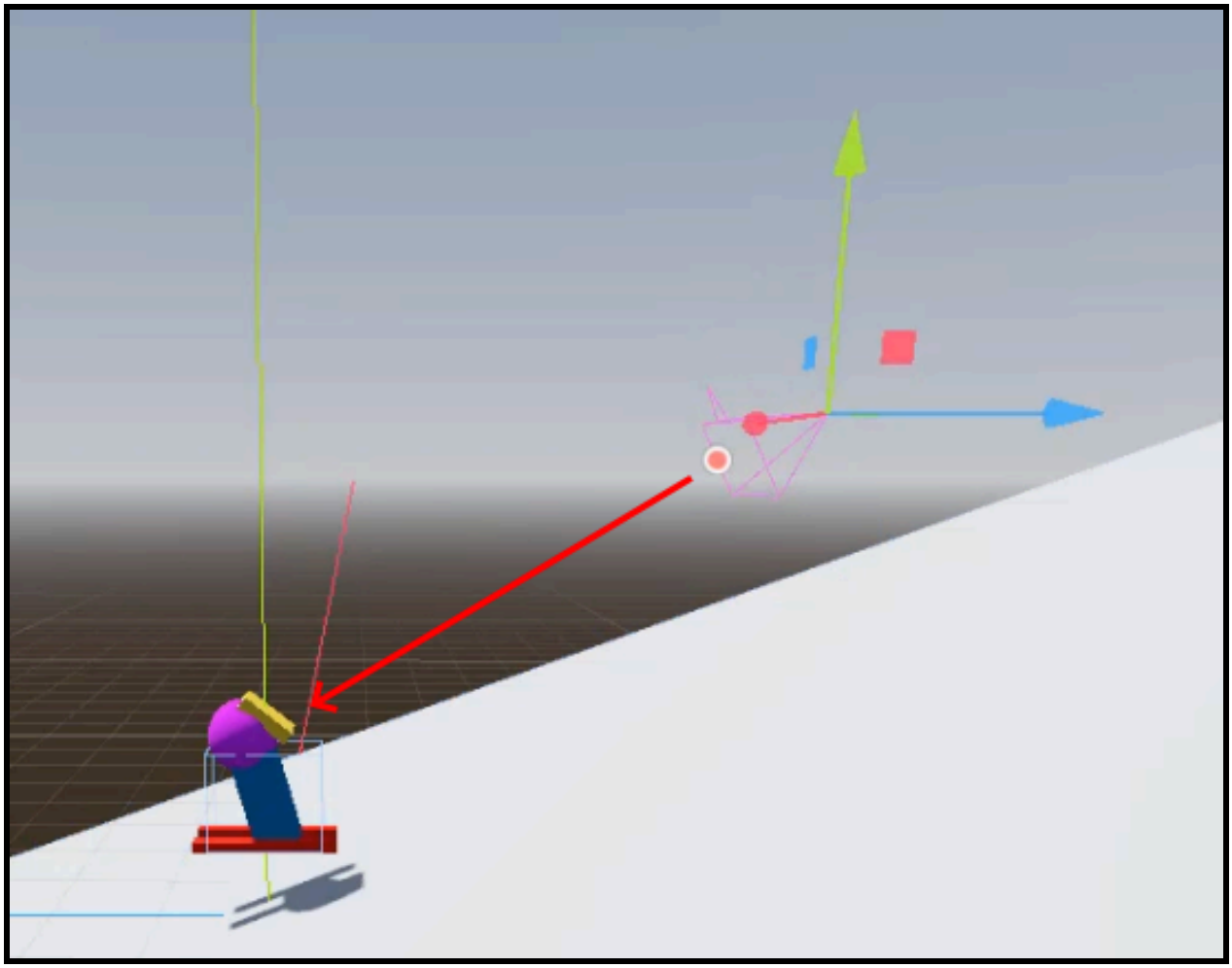
We then want to fit this to the bounds of our player model, using the orange circles to scale the box.



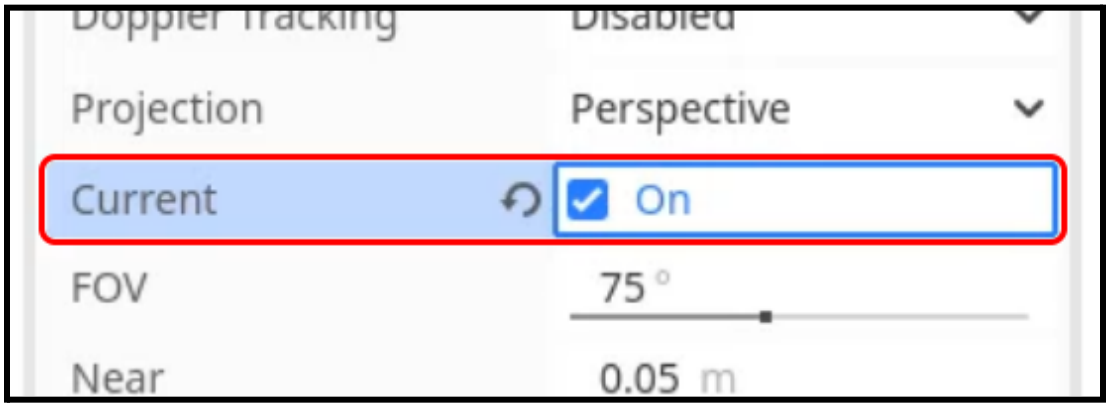
We then want to add a **Camera3D** as another child node of the *RigidBody3D* node, so that we can track the player down the slope.



Then we want to **move** and **rotate** our camera so that it is looking down on our player.

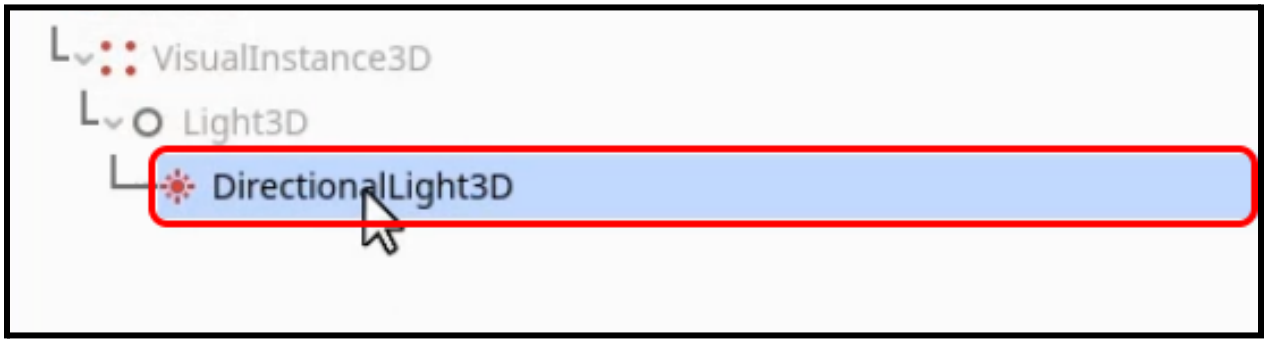


We then want to enable the **Current** property in the inspector, to tell Godot to use this as the camera for the game window.

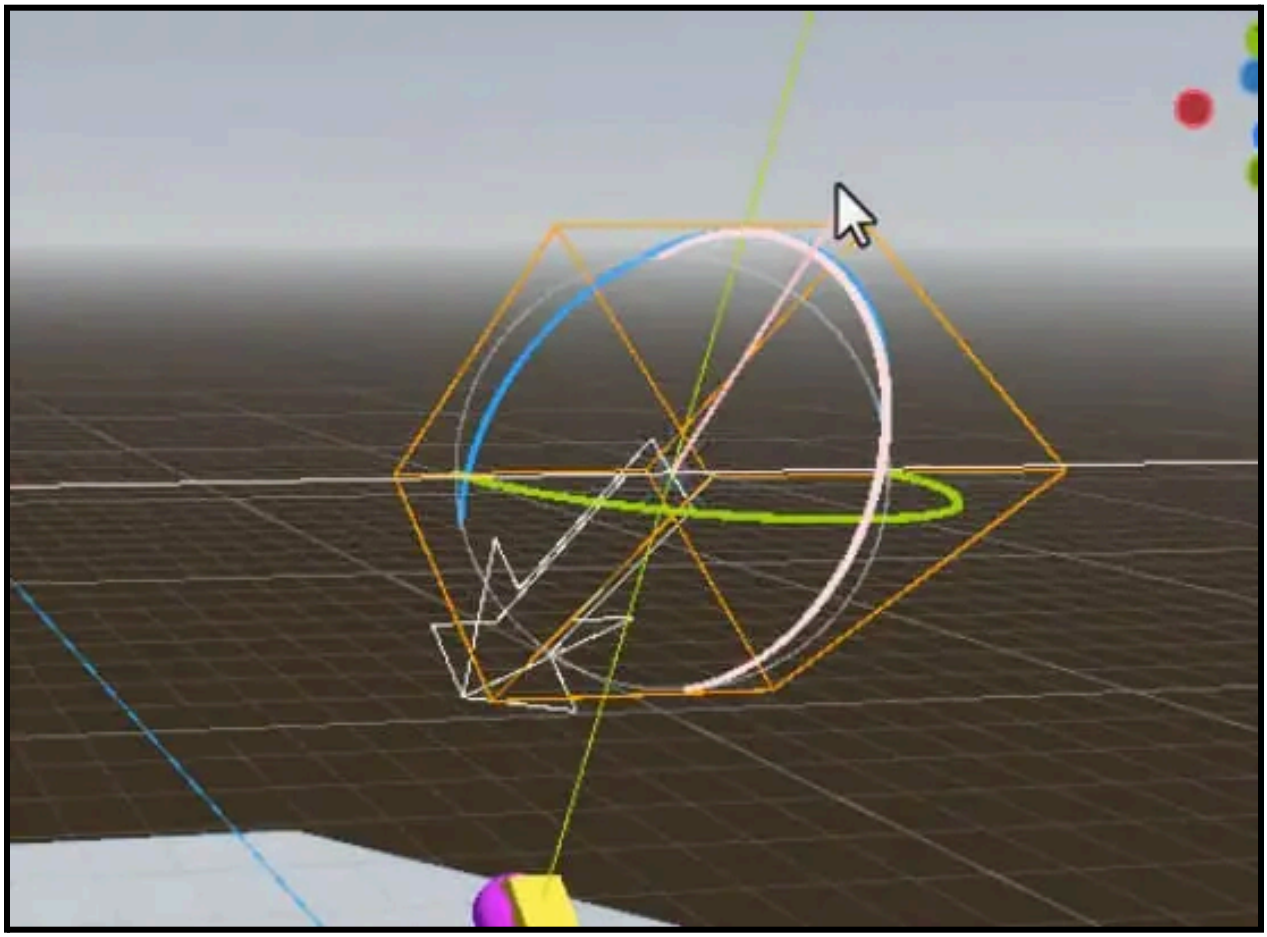


Creating the Environment

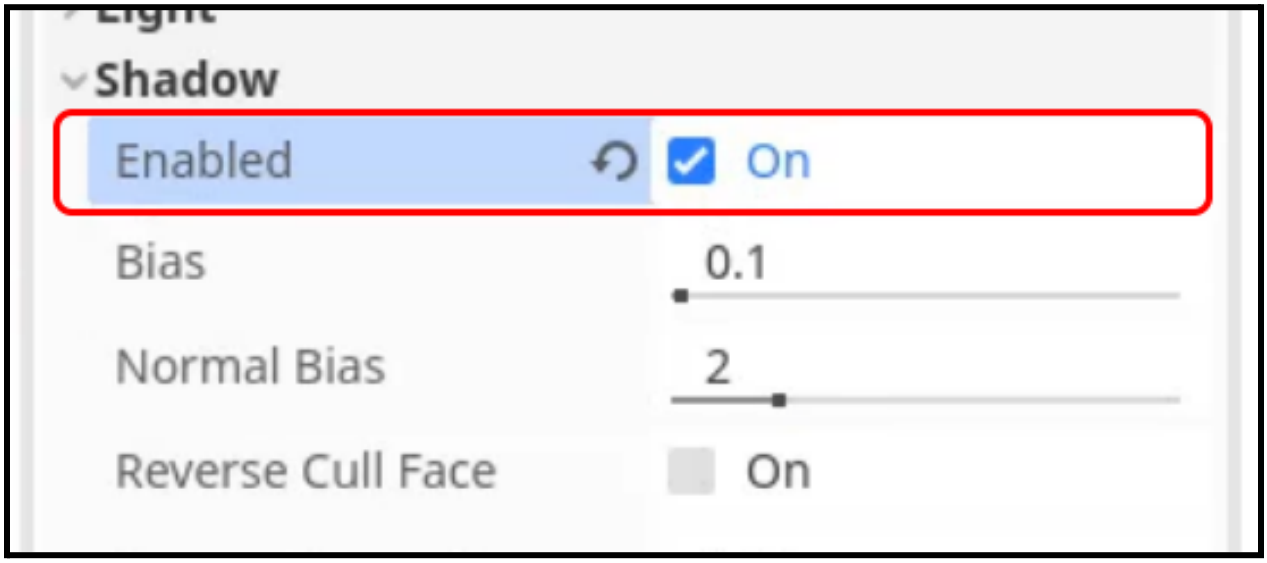
We will then create a **DirectionalLight3D** node to provide lighting in our scene.



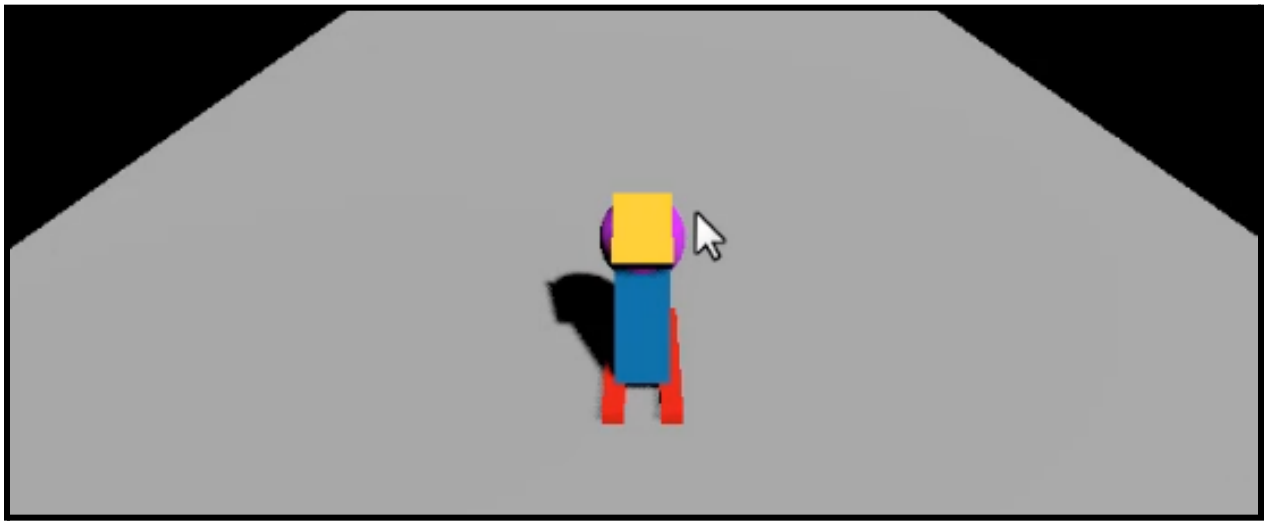
From here, position the light to shine down on an angle at our player.



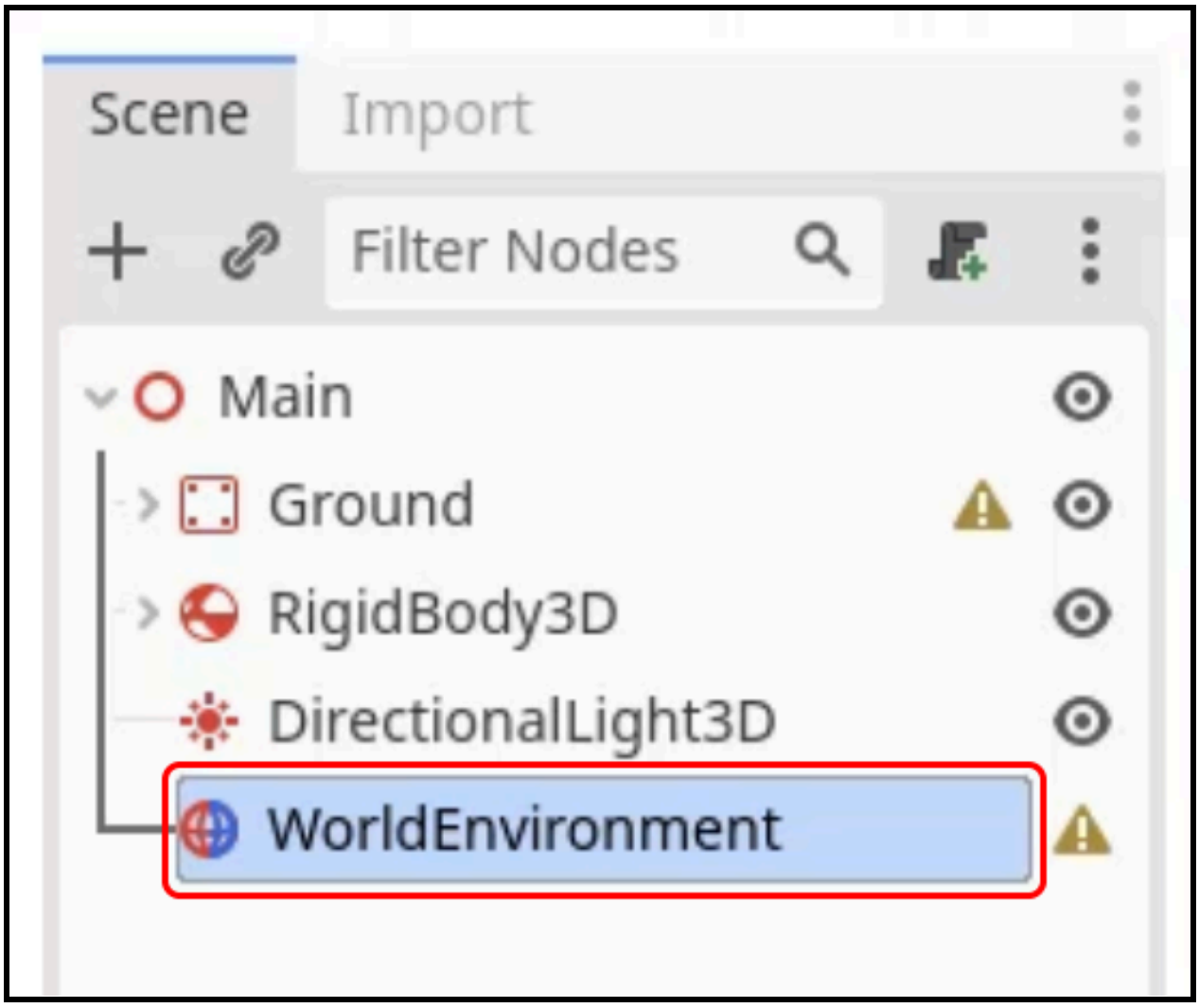
Finally, enable the **Shadow** property of the *DirectionalLight3D*.



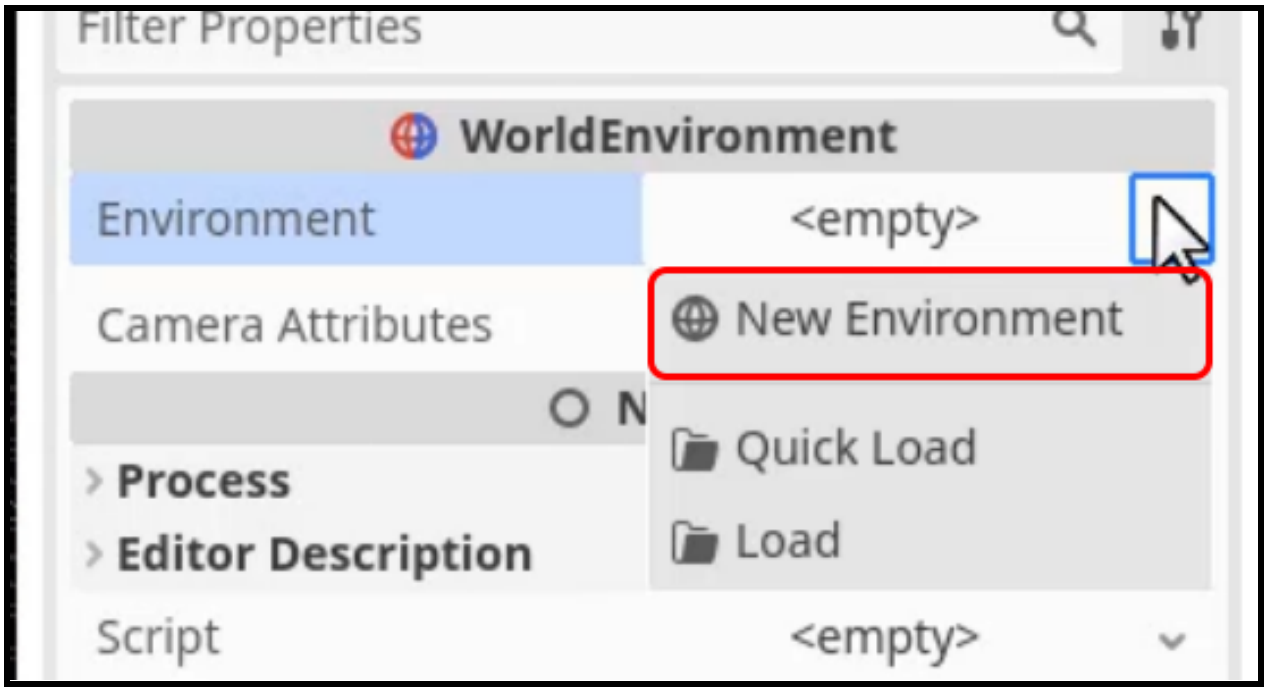
Now, when you press **Play** you will see the scene is lit and we can see our character.



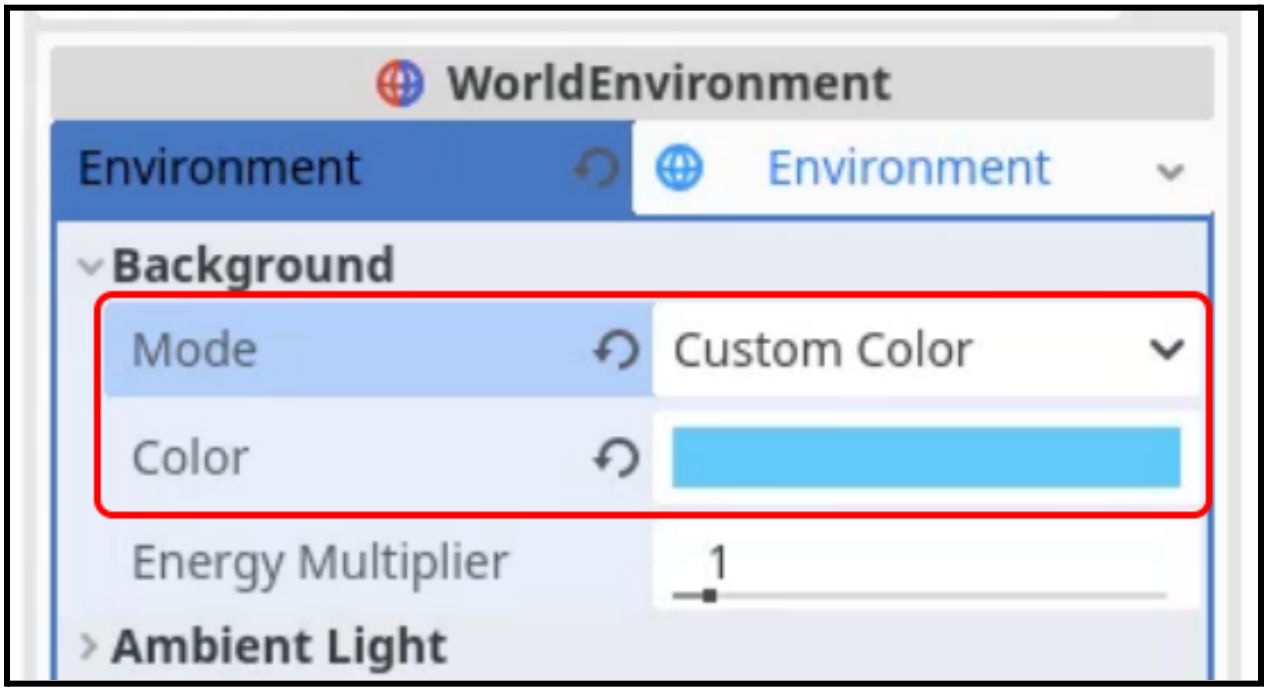
Currently, however, it doesn't really look like a skiing environment. To fix this we will use a **WorldEnvironment** node, which will allow us to change things such as background color, fog, and lots more.



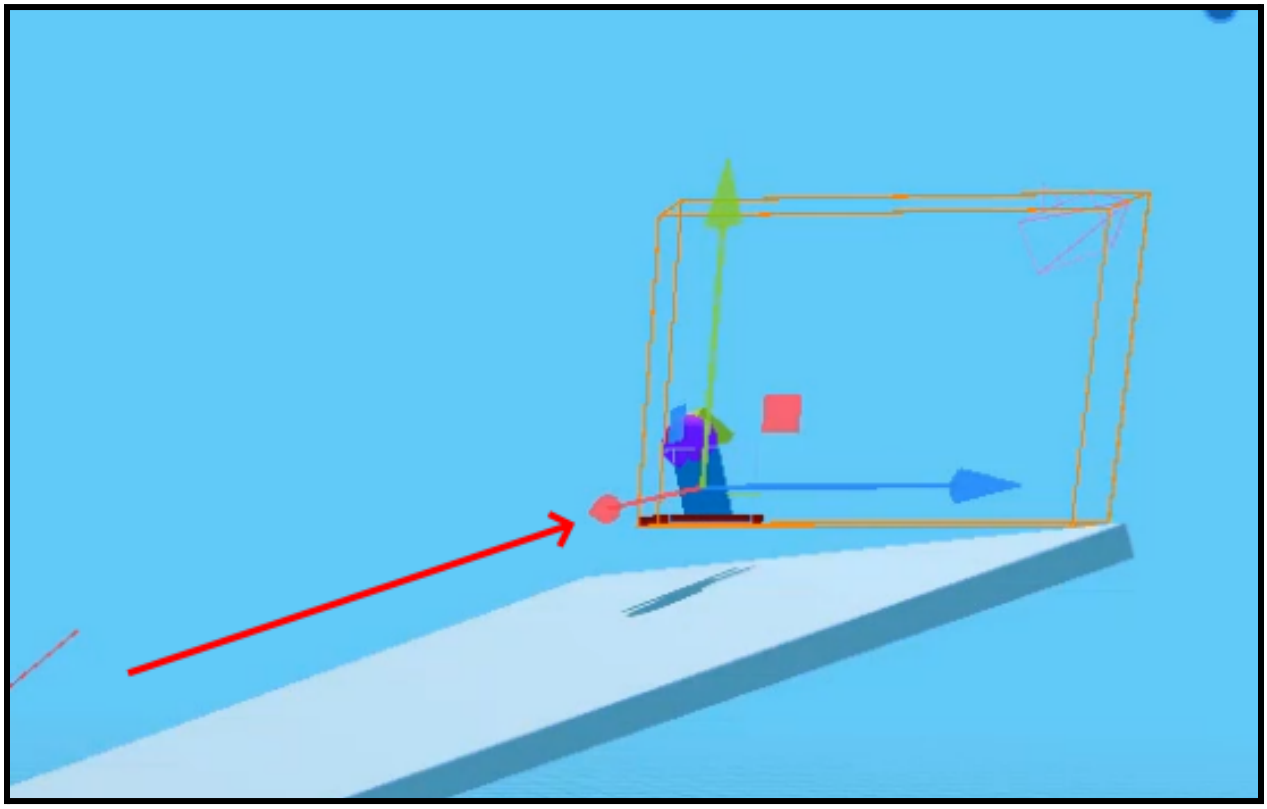
We will create a **New Environment** for the **Environment** property.



To begin with, we will change the *Mode* to **Custom Color** in the **Background** tab. We will change this to a light blue hue.

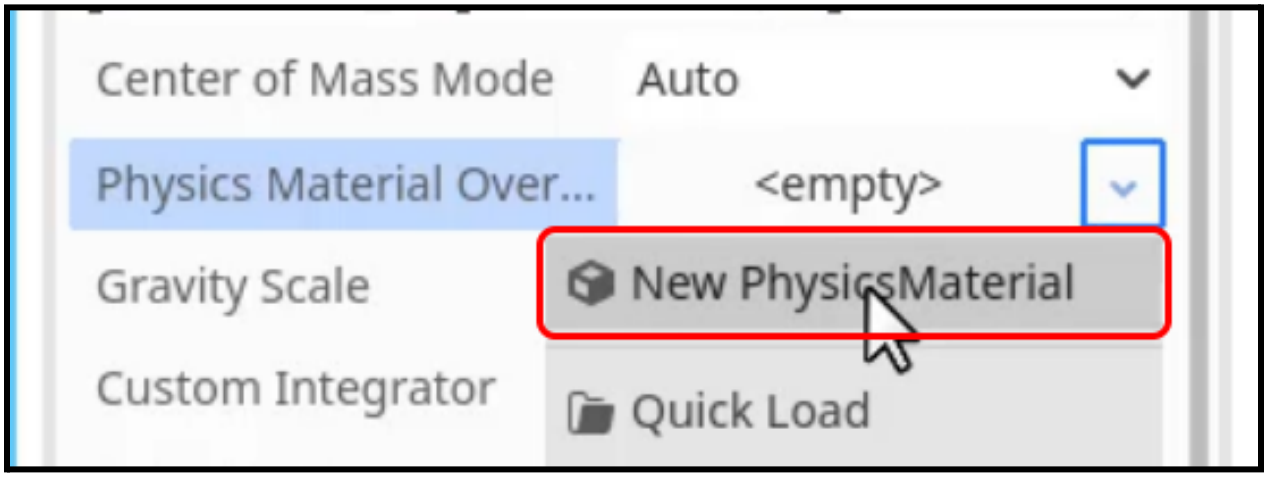


Finally, we will move the *RigidBody3D* player node to the top of the slope, so that when we press *Play* it starts at the top.

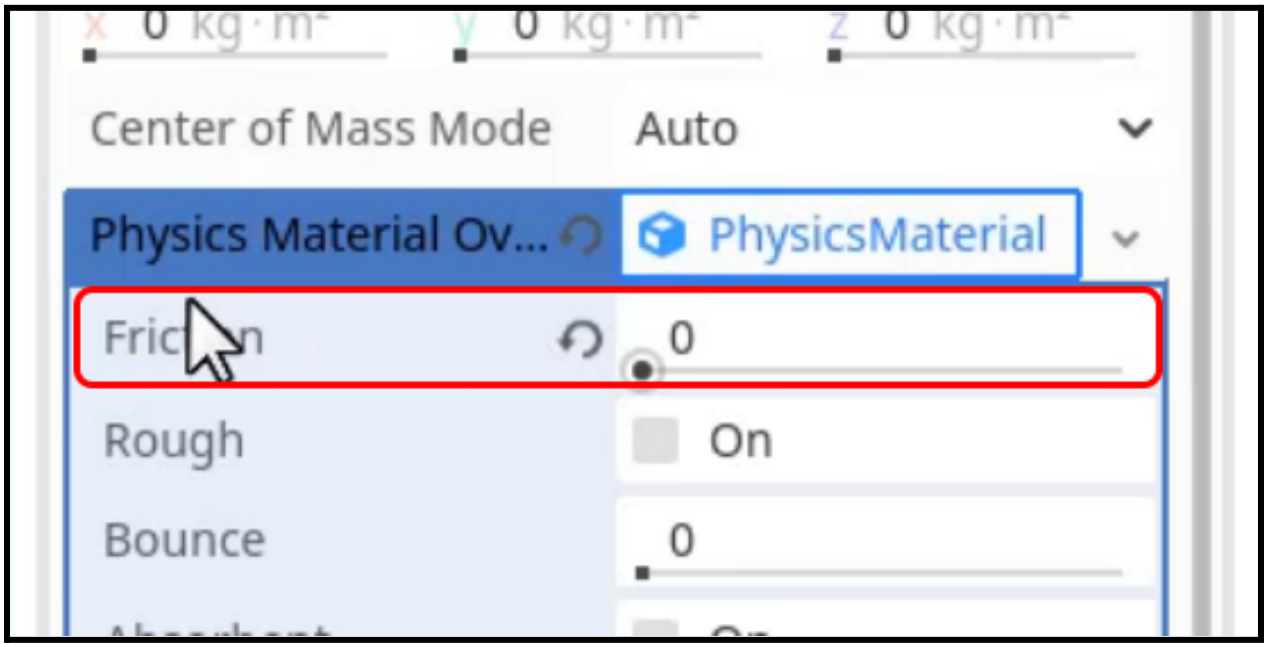


We will also rename the *RigidBody3D* node to *Player* so that we can easily identify it.

To make the Player slide down the slope, we will also add a **Physics Material Override** value.



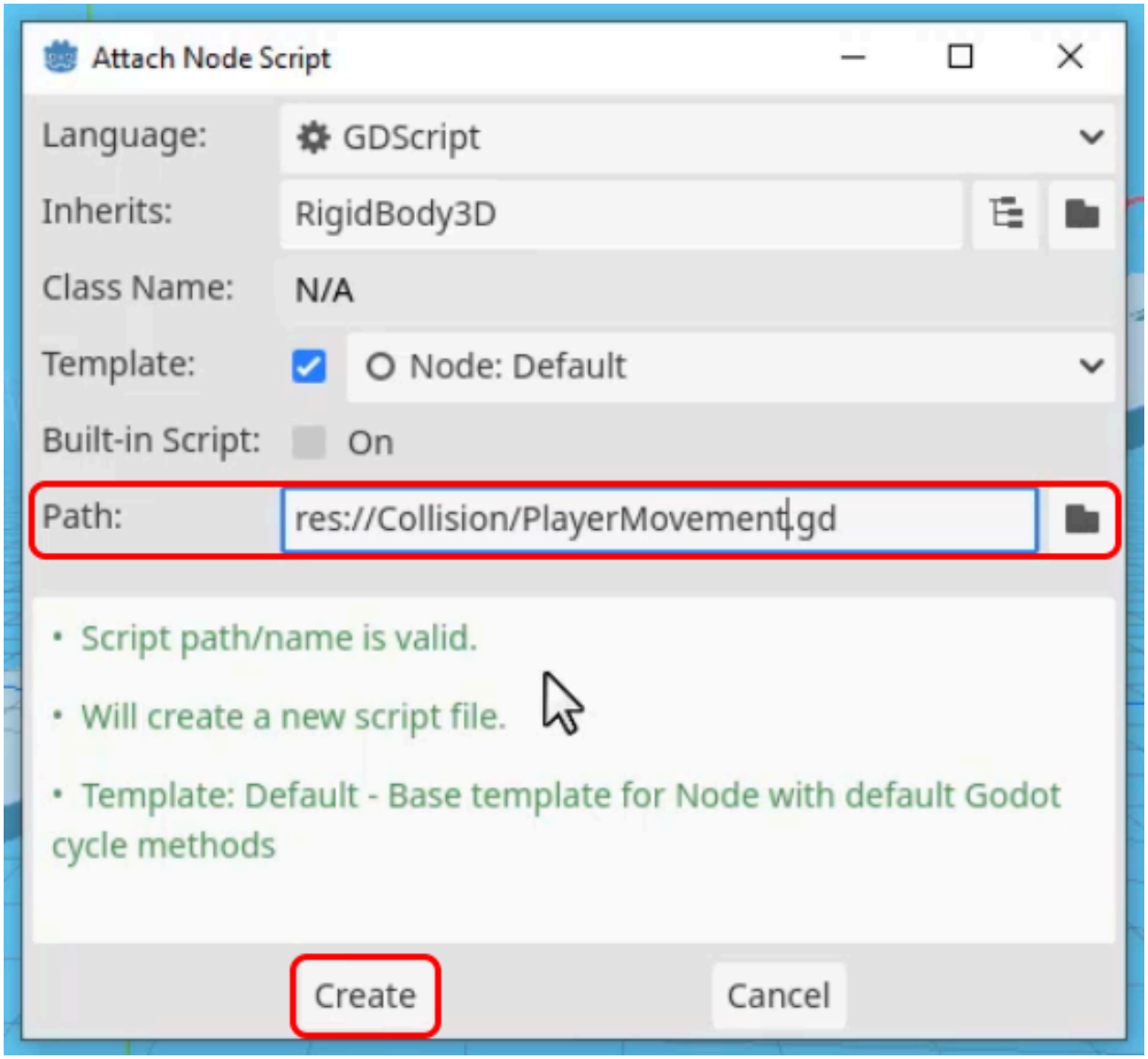
We will then set the **Friction** property to **0** so that our *Player* slides down the slope.



Now when you press **Play** you will see the *Player* slides down the hill as intended.

Creating the Movement

We will then select our **Player** node and create a script called *PlayerMovement.gd,* and make sure it inherits from *RigidBody3D*.



We will delete the *_ready* and *_process* functions and create an exported variable called **moveSpeed** of type **float** that has a default value of **2.0**.

```
1.  @export var move_speed : float = 2.0
```

We will then create a function called *_physics_process* which will detect our key presses. The *_physics_process* function is useful for physics because it runs at a fixed rate, unlike *_process* which can change depending on the FPS of the game.

```
1. func _physics_process(delta):
```

We will then set our **linear velocity** on the **x-axis** to be negative or positive *move_speed* depending on whether the left or right arrow key is pressed.

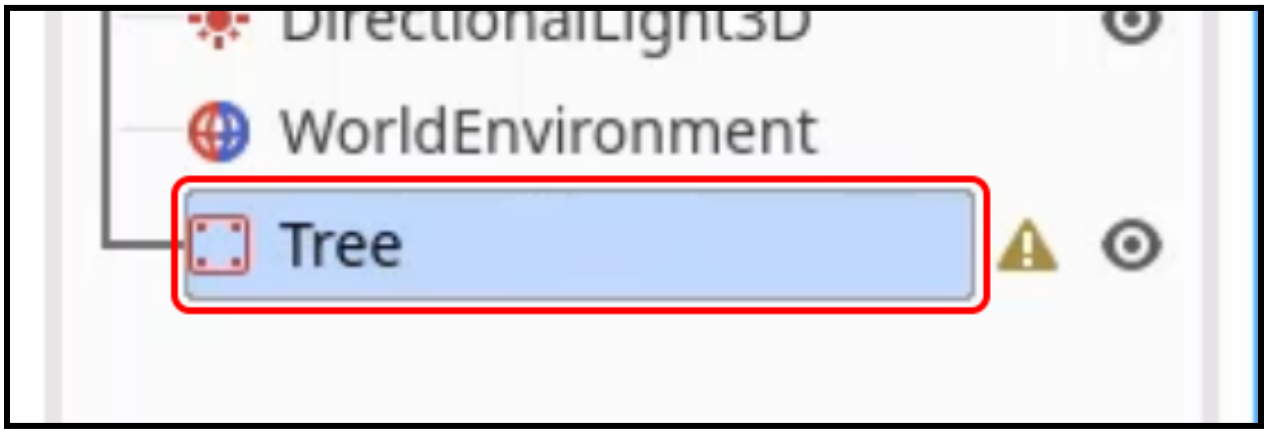
```
1. func _physics_process (delta):
2.     if Input.is_key_pressed(KEY_LEFT):
3.         linear_velocity.x = -move_speed
4.     if Input.is_key_pressed(KEY_RIGHT):
5.         linear_velocty.x = move_speed
```

Now when you press **Play** you will be able to move left and right as we ski down the hill. In the next lesson, we will begin implementing our trees for the *Player* to collide with.

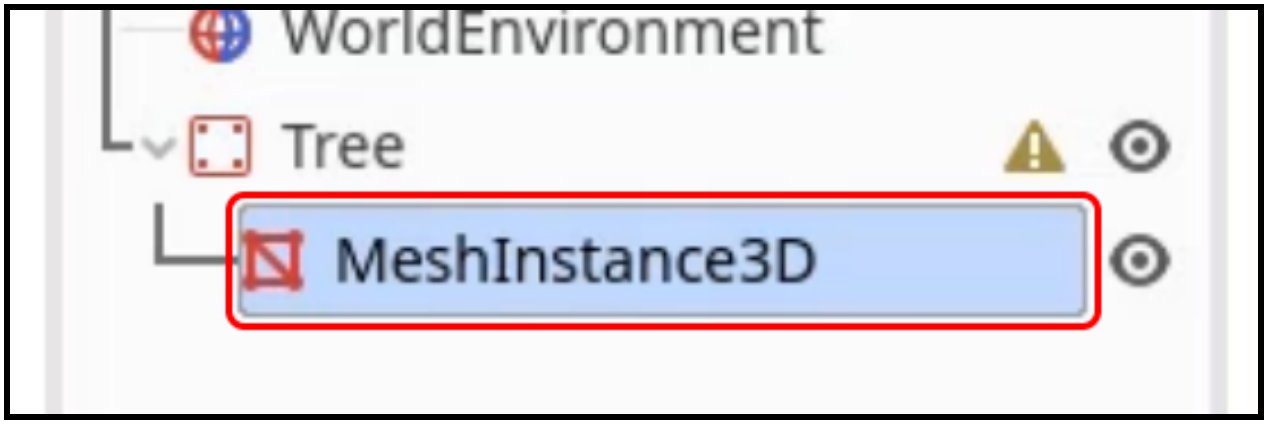
Collision – Part 3

In this lesson, we will be creating our tree obstacle.

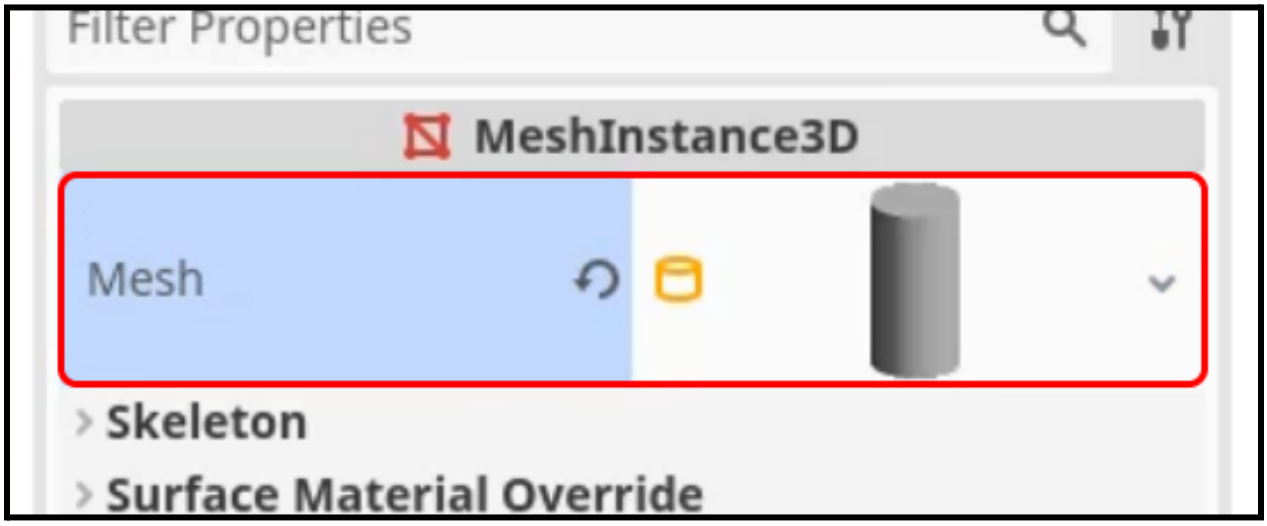
The first step is to create a new **StaticBody3D** node, which we will call *Tree*.



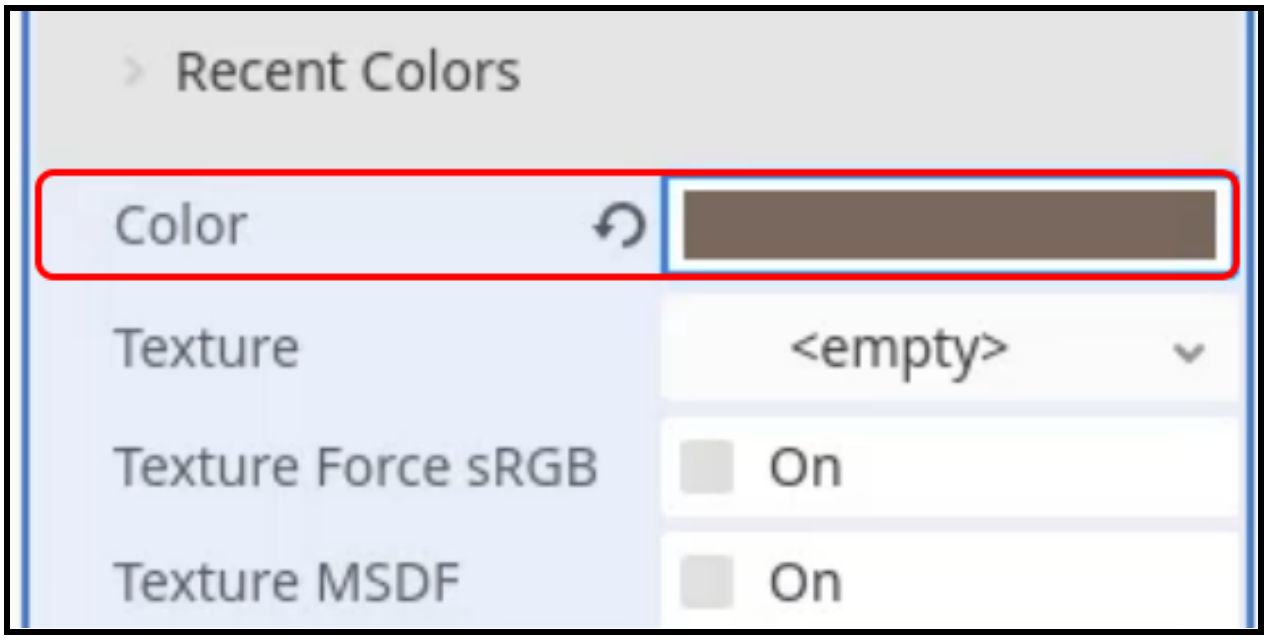
We will then add a **MeshInstance3D** node as a child of *Tree*.



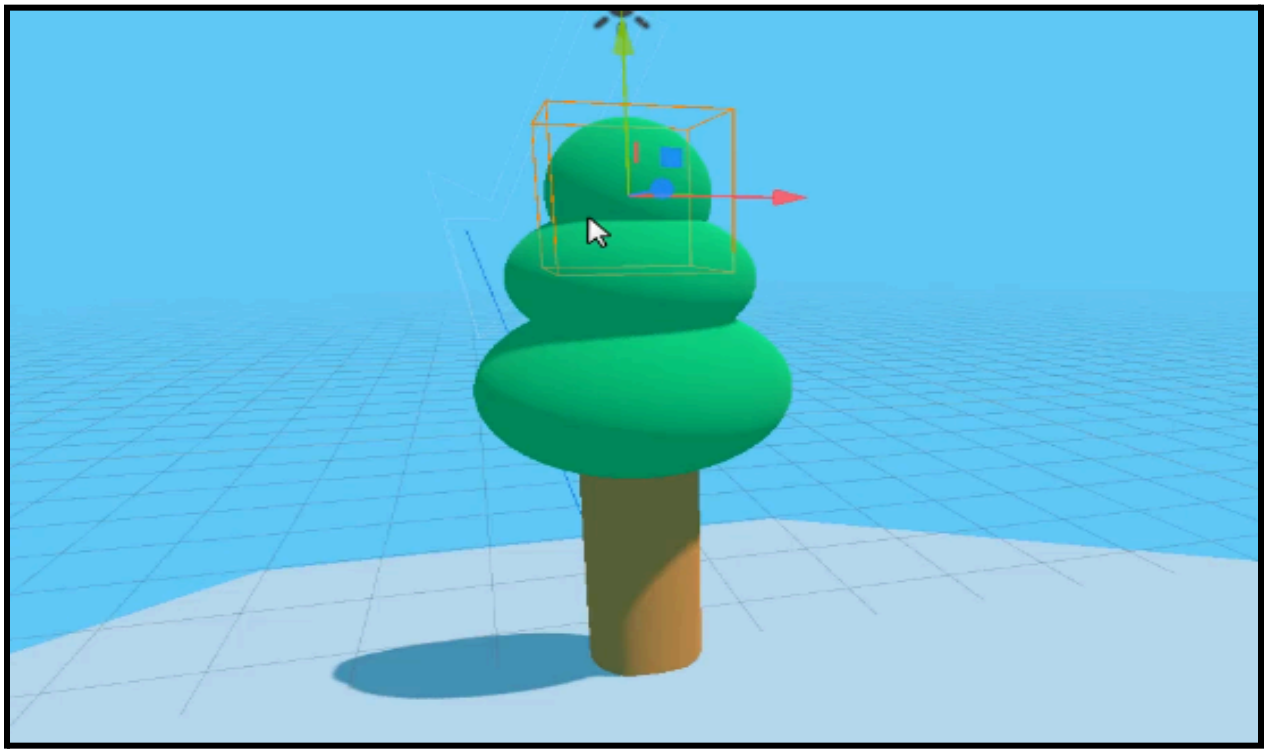
This will be a **Cylinder** for the trunk of the tree.



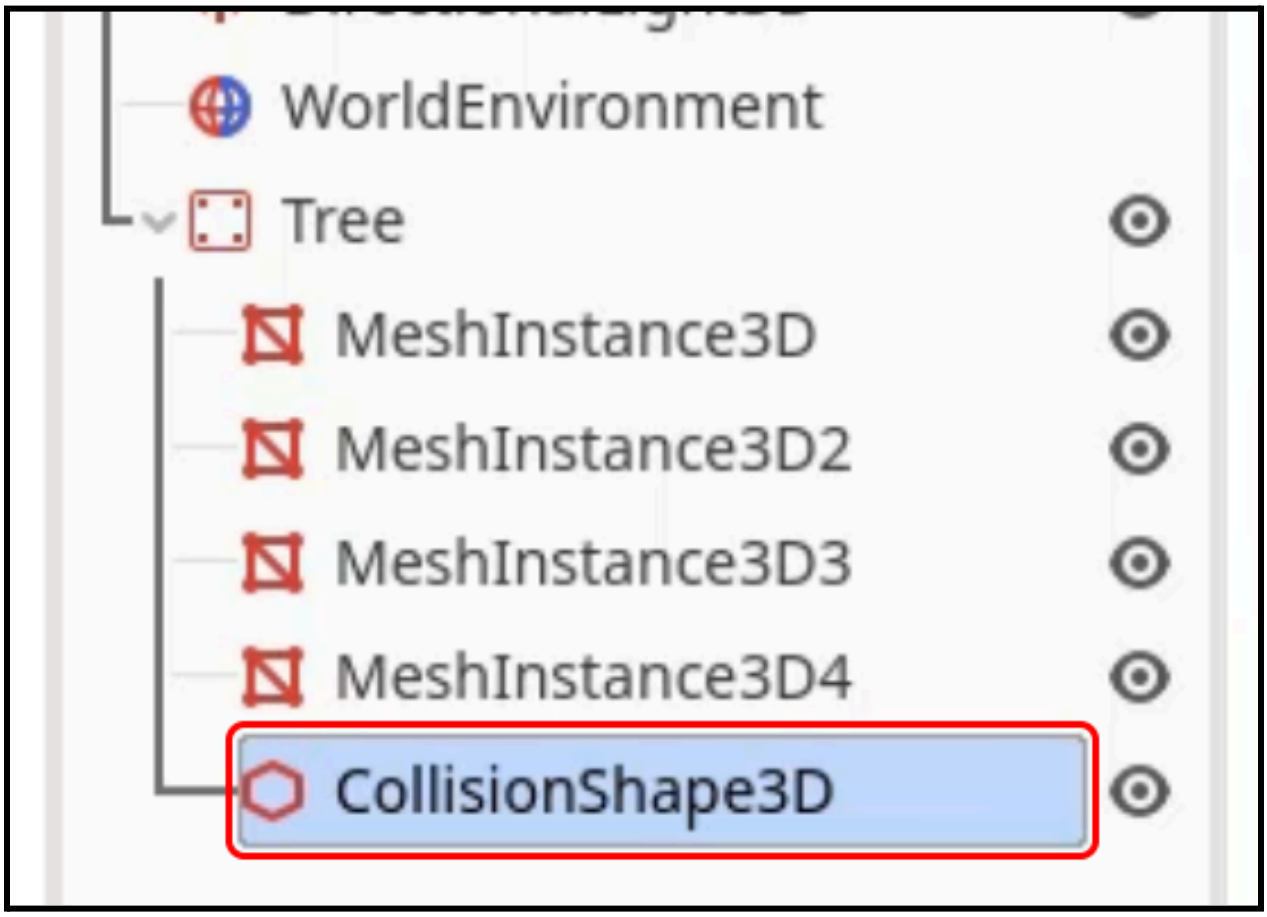
We will then give this **brown material** to be the color of bark.



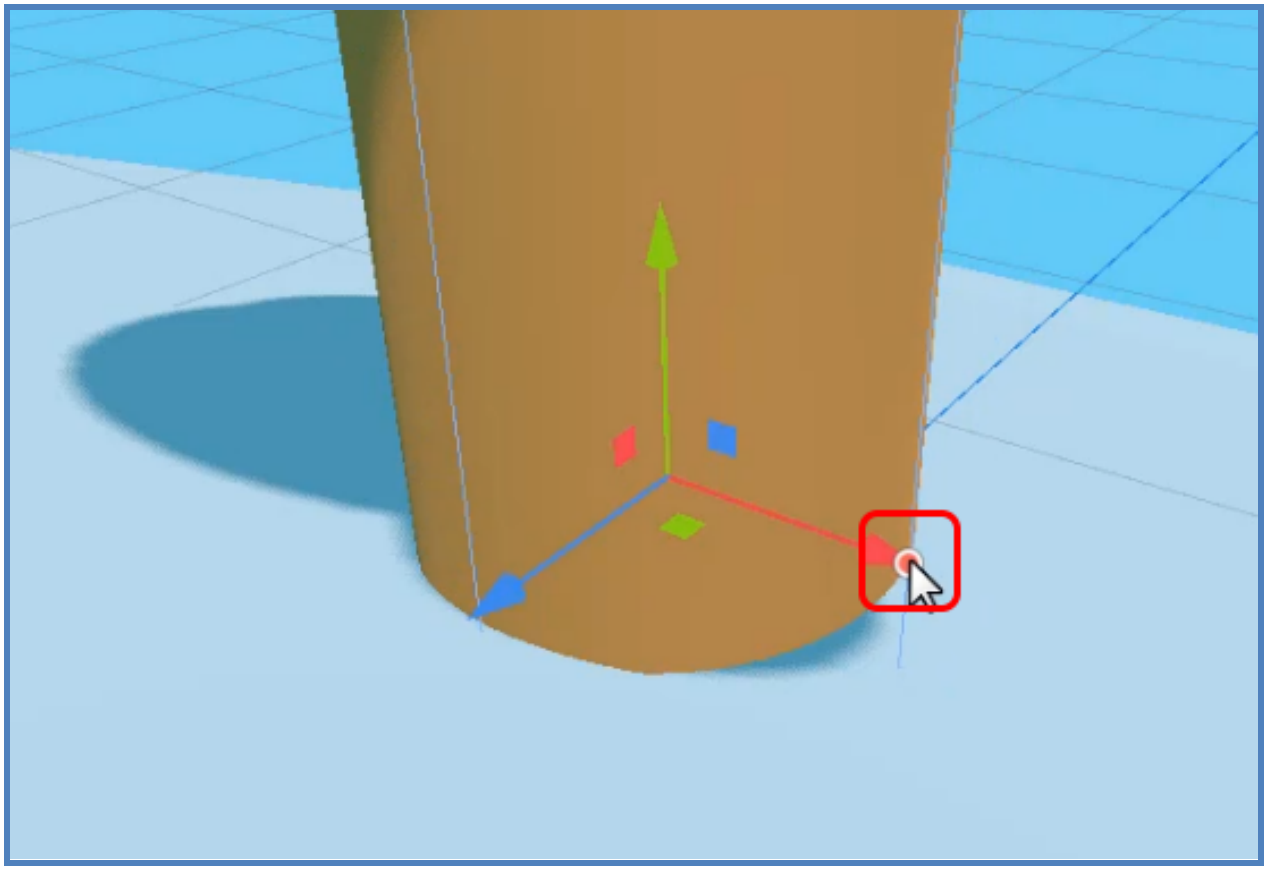
From here you can add more *MeshInstance3Ds* to create leaves on the tree, or alternatively create an entirely different model to be an obstacle.



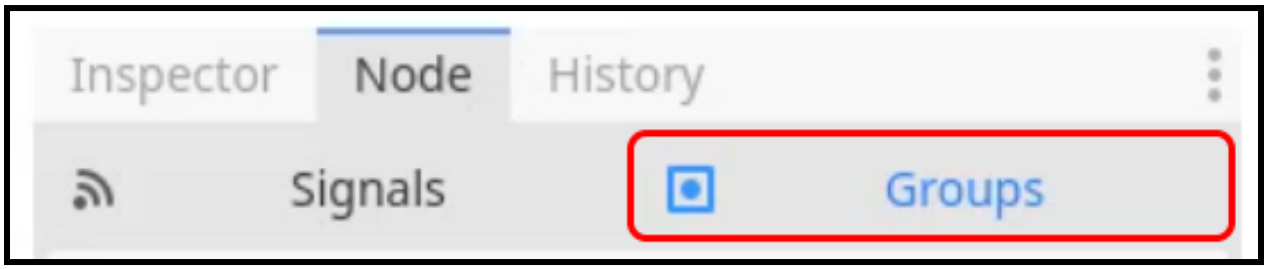
We also want to add a **CollisionShape3D** as a child of our tree, so that our *Player* has something to collide with.



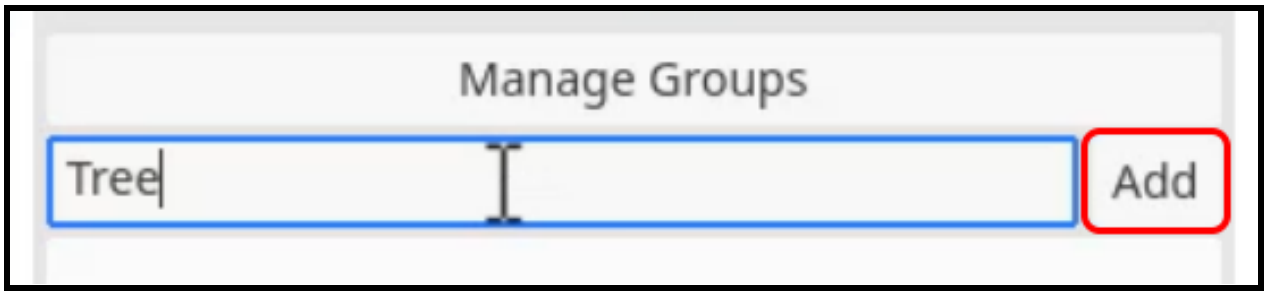
This will be a **CylinderShape3D**, which we can fit to the bounds of our tree trunk.



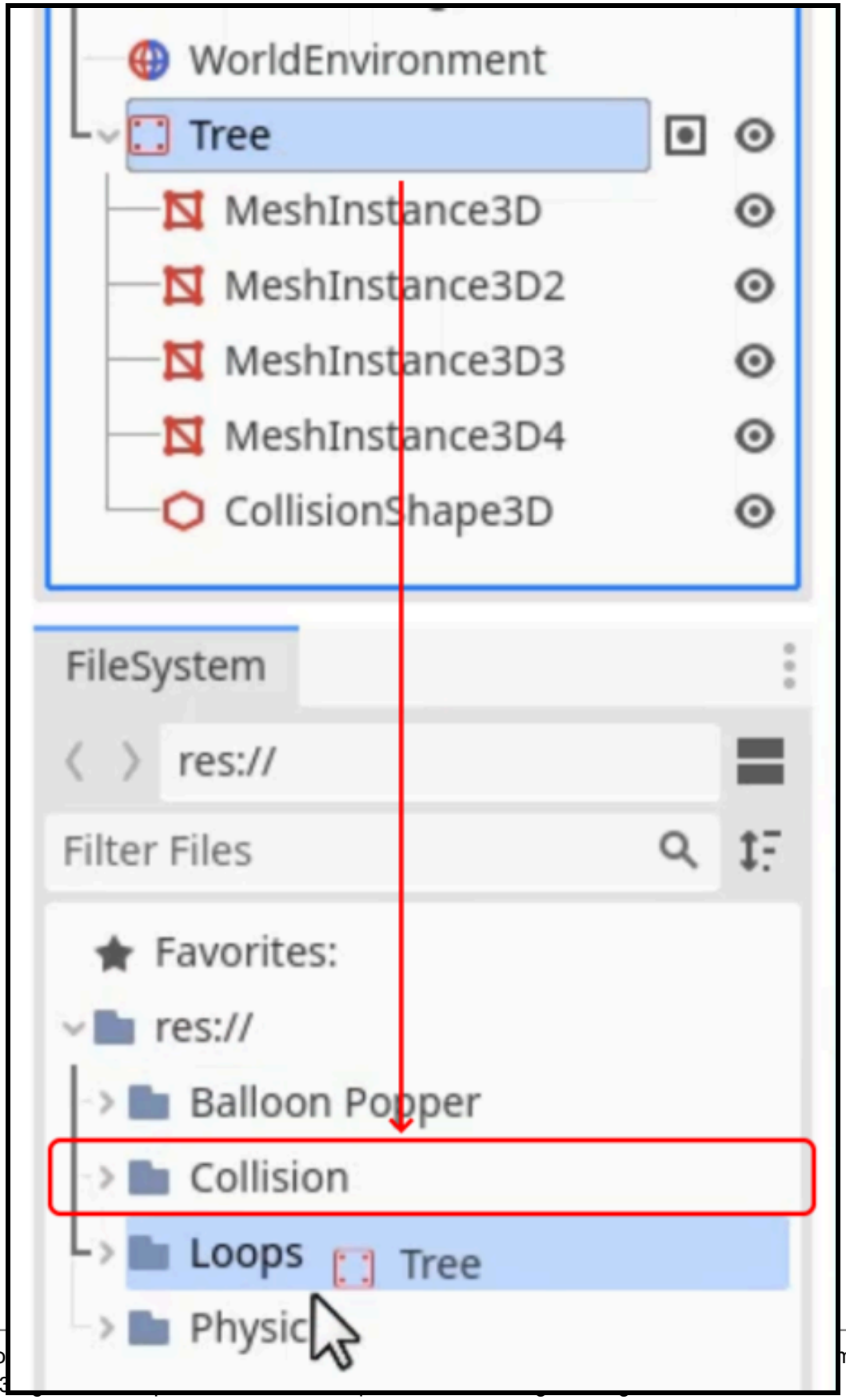
We then add a group to our tree node, we will do this in the *Node* tab with the *Tree* node selected. We will use a group to add a label to our tree so that we can detect if it is a tree our *Player* has hit. This will allow us to detect the difference between the *Tree's* and the *Ground's StaticBody3D* nodes.



We will add a new group to our node with the name *Tree*.

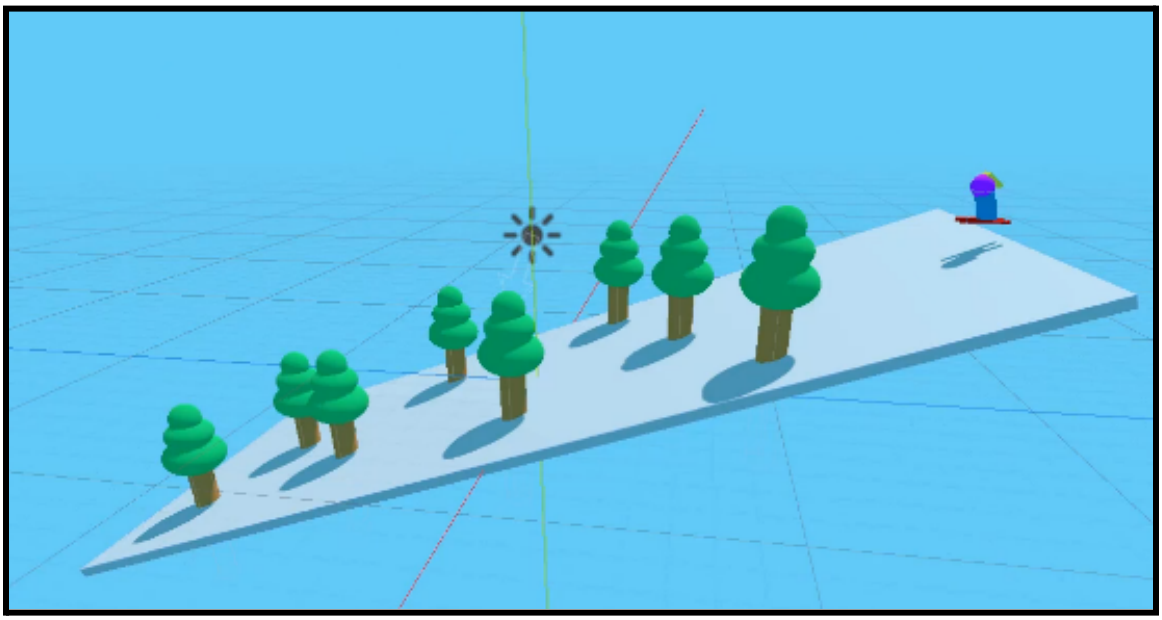


Finally, we will drag the *Tree* node into the *FileSystem* to turn it into a scene, allowing us to edit multiple instances at once. Remember to save the scene in the *Collision* folder. We will be using the name *Tree.tscn.*



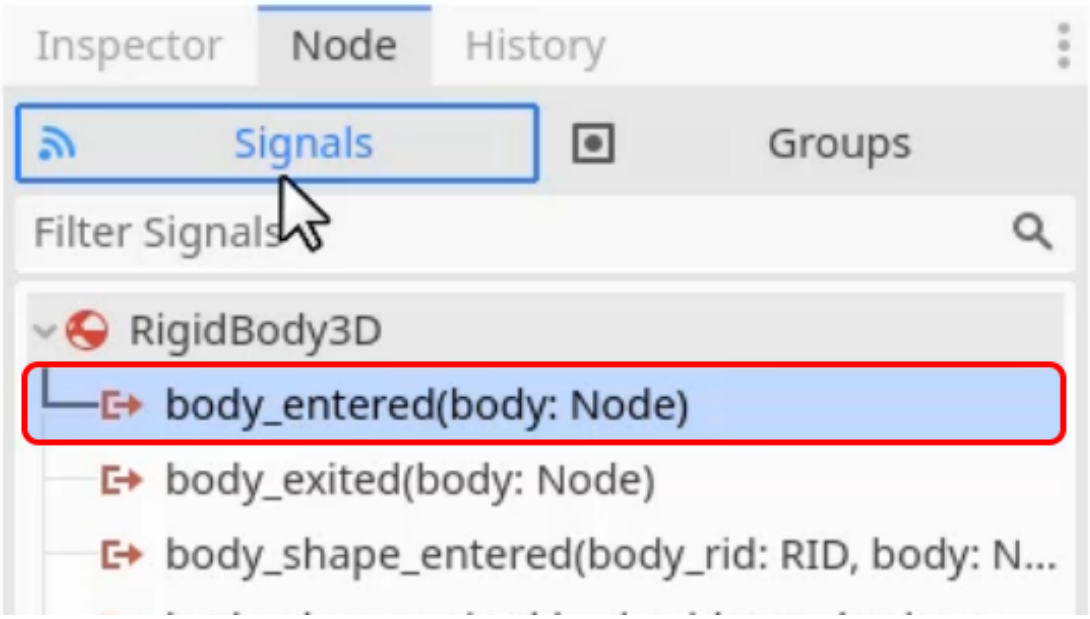
This boo master
2D and 3

You can now **duplicate** (**CTRL+D**) the obstacles around the level, to give the *Player* something to dodge as they slide down the slope.

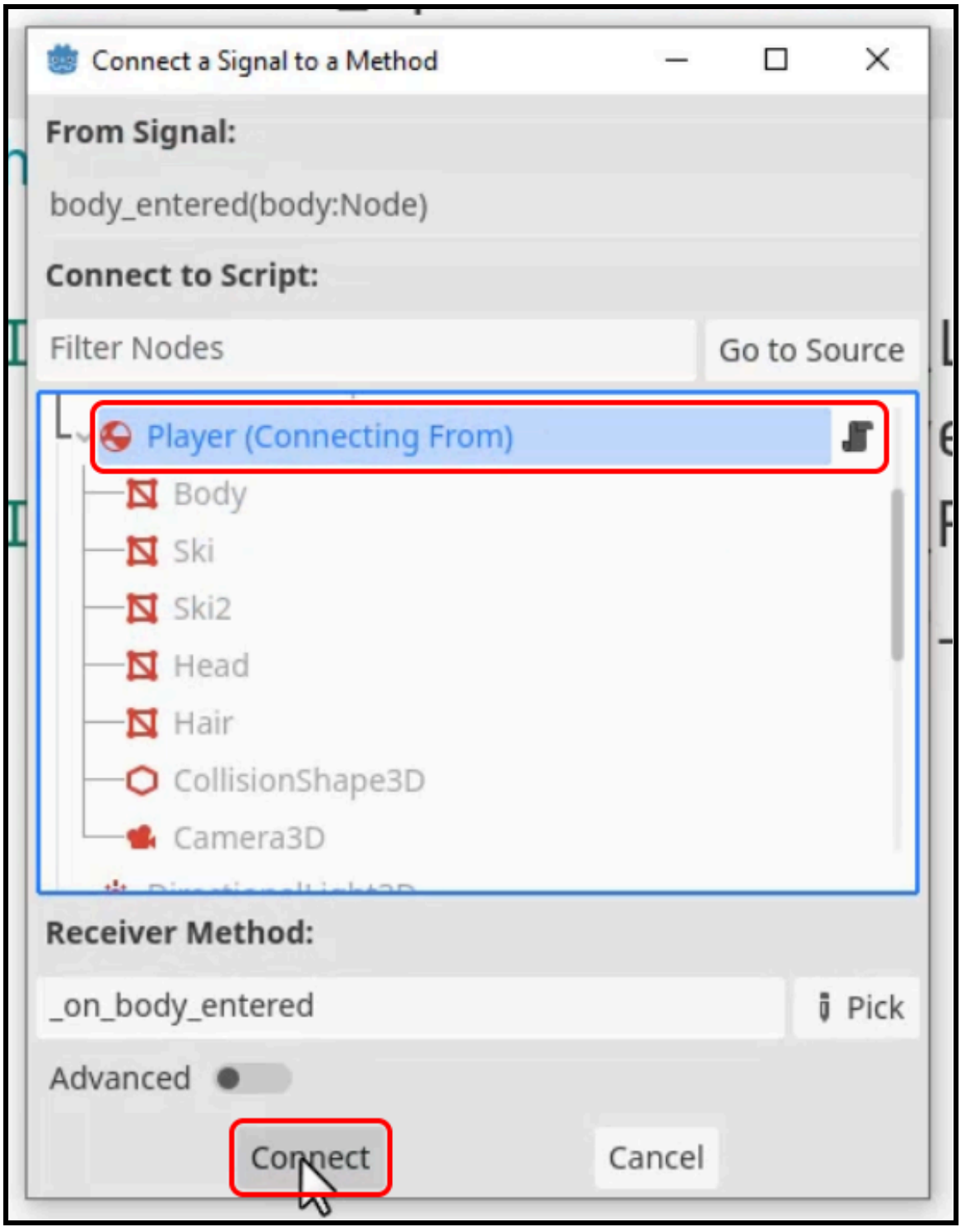


Detecting Collisions

Currently, if you press *Play* you will notice when you touch a tree nothing happens. To fix this, we will select our *Player* rigid body node and go to the **Signals** tab. We can then connect the **body_entered** signal to our *Player* node. A *signal* is a way of calling a function when an event happens, in this case when the *Player* makes a collision with another object.



In the pop-up window, make sure the *Player* node is selected and press **Connect**.

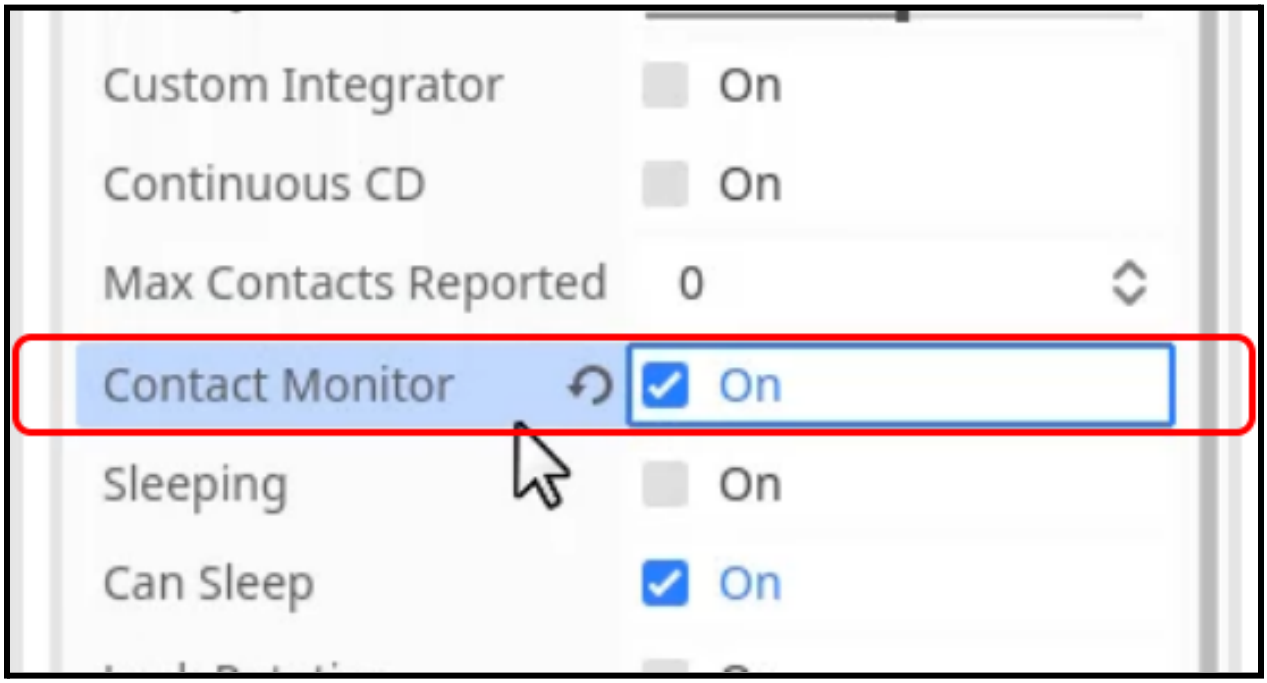


Inside the new *_on_body_entered* function, we can now check to see if the body that we have collided with is within the *Tree* group. If so, we then reload the scene.

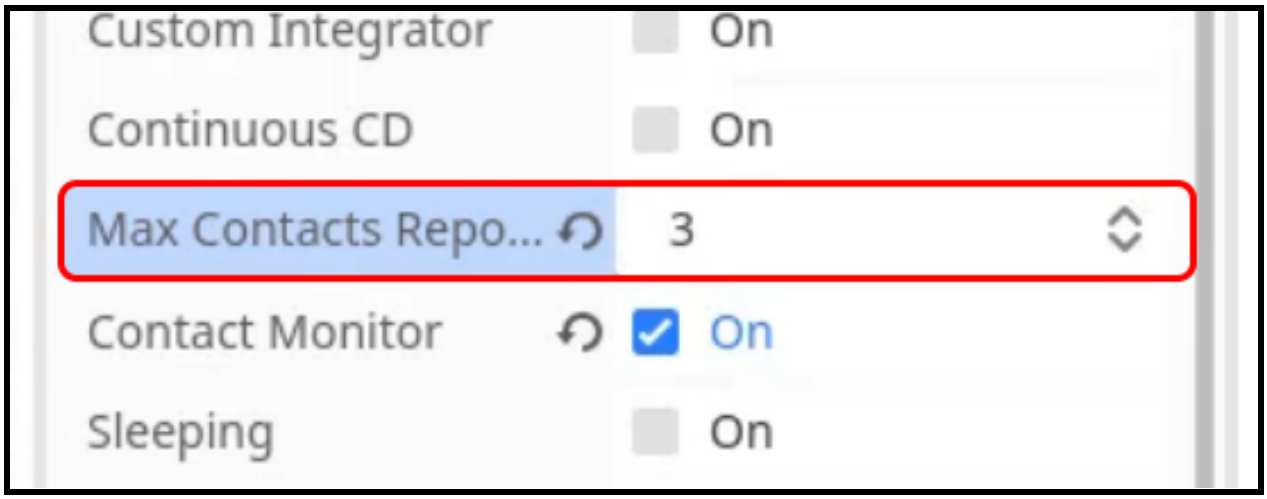
```gdscript
func _on_body_entered(body):
    if body.is_in_group("Tree"):
        get_tree().reload_current_scene()
```

Enabling Contact Monitoring

If you press *Play* now, you will notice nothing happens still. This is because we need to enable the **Contact Monitoring** property on our *Player* node.



This will allow us to detect contacts with other colliders. We will also change **Max Contacts Reported** to **3**. This will mean we can track multiple collisions at once, which is necessary as we will have one collision with the ground so we need more than one collision report at a time to be able to hit the trees.



Now if you press **Play** you will see you can ski down the hill and collide with the *Trees*, which will reset the *Player* to the top of the slope. As a challenge, feel free to add a collider at the end to detect when the *Player* has reached the end of the slope.

That is everything for this mini-game, which has taught us to add colliders to our objects, detect collisions between the objects, and check the type of an object by the groups applied to it.

Conclusion

This tutorial concludes our simple skiing game using Godot's 3D collision system. We've covered how to create collidable objects in the 3D environment, detect and handle collisions between these objects, and reload the scene upon specific events like hitting a tree. While the game is small and simple, you can expand on this concept by adding more obstacles, improving the player movement controls, and even creating a procedural level generation system for endless skiing experience.

You can also explore more advanced game development topics in Godot, such as creating multiplayer games, improving your 3D art skills, or working with more complex physics interactions to keep expanding your game development knowledge and portfolio.

We hope you've learned a lot from this tutorial, and we wish you the best of luck in your future game development projects!

Want to learn more? Try our complete GODOT 4 MINI-PROJECTS course.