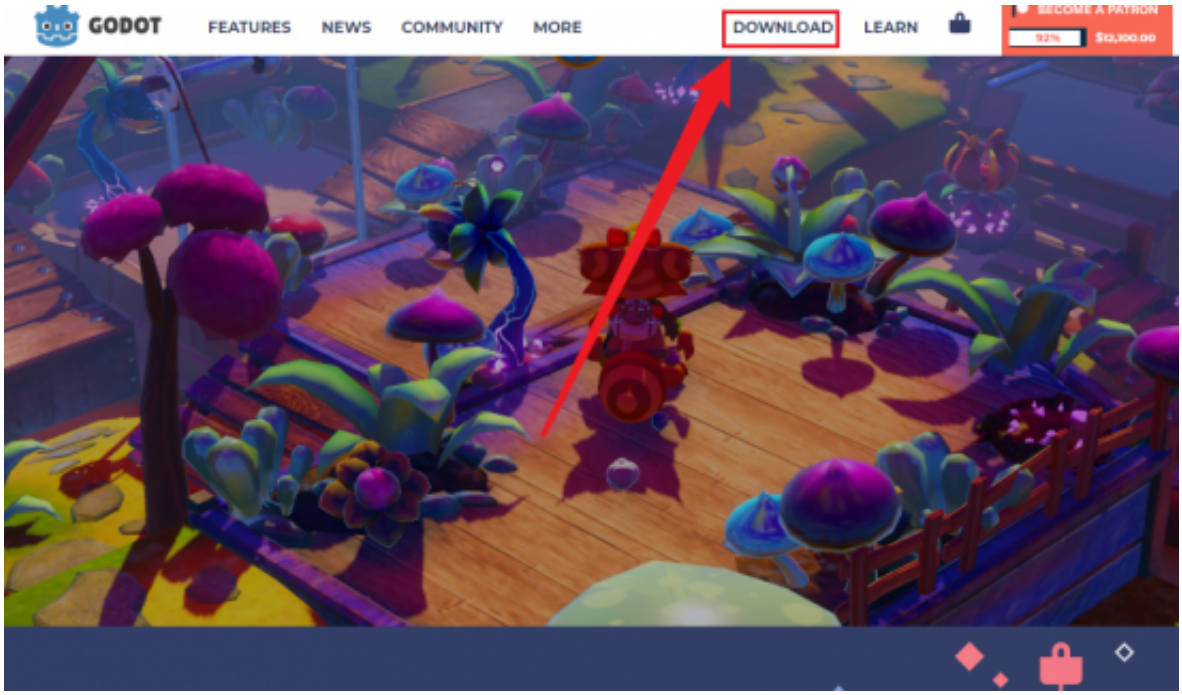
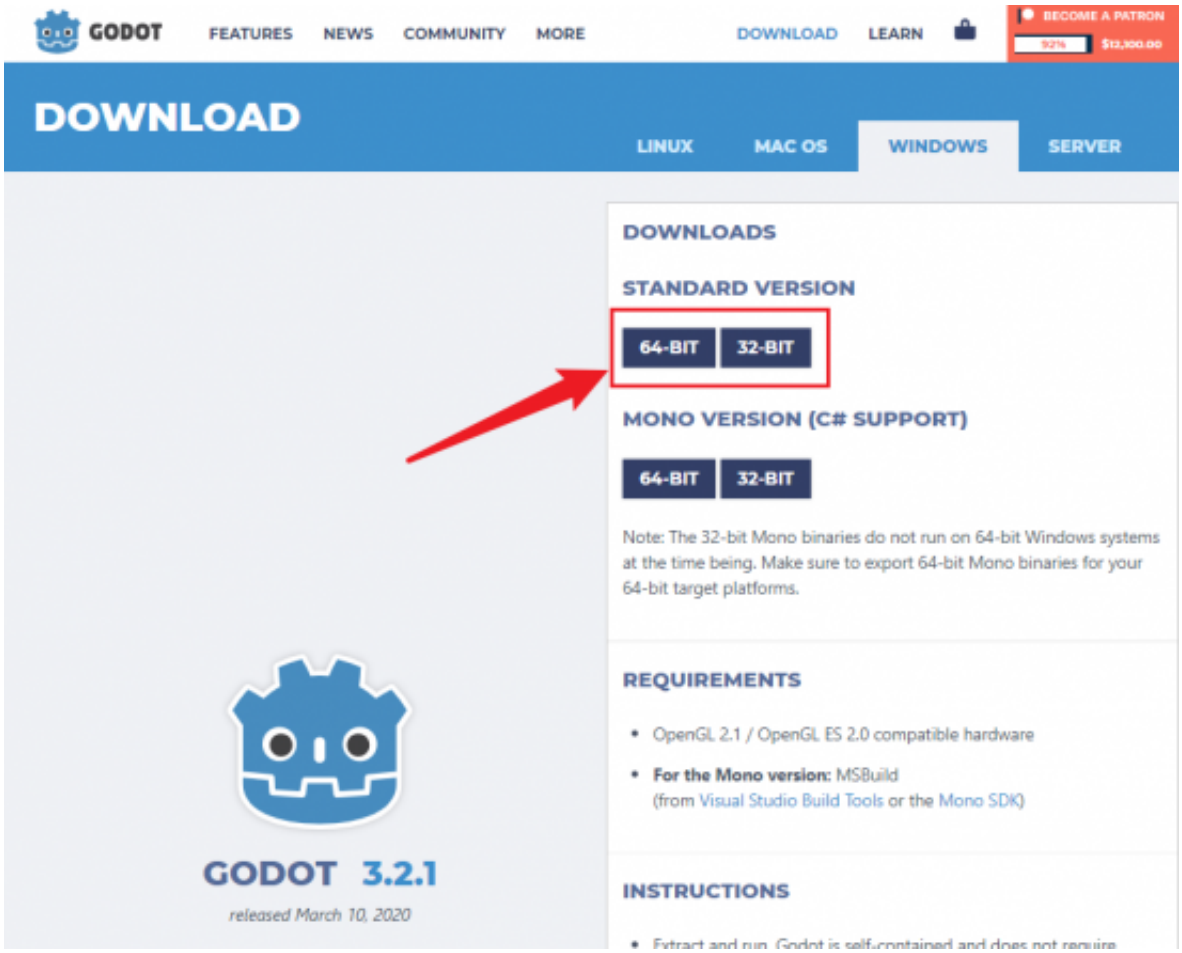
How to install and set-up Godot

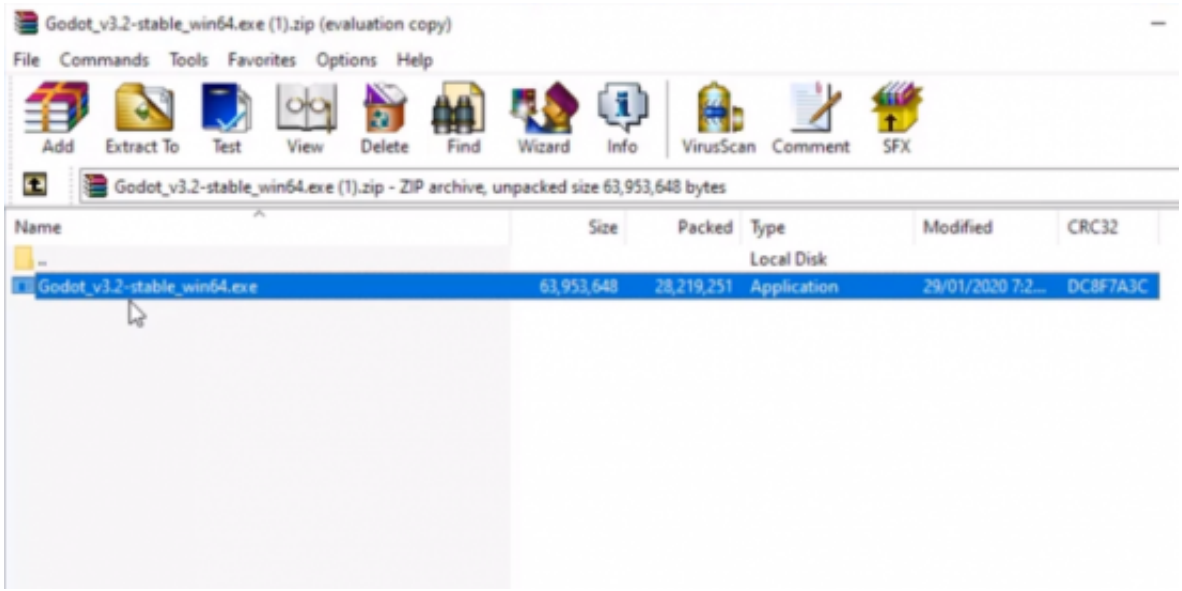
1. Go to Godotengine.org
2. Click on the **DOWNLOAD** button



3. Download the **Standard** version. (Check your OS to choose between a 32-bit / 64-bit)



4. Extract and run the **.exe** file.

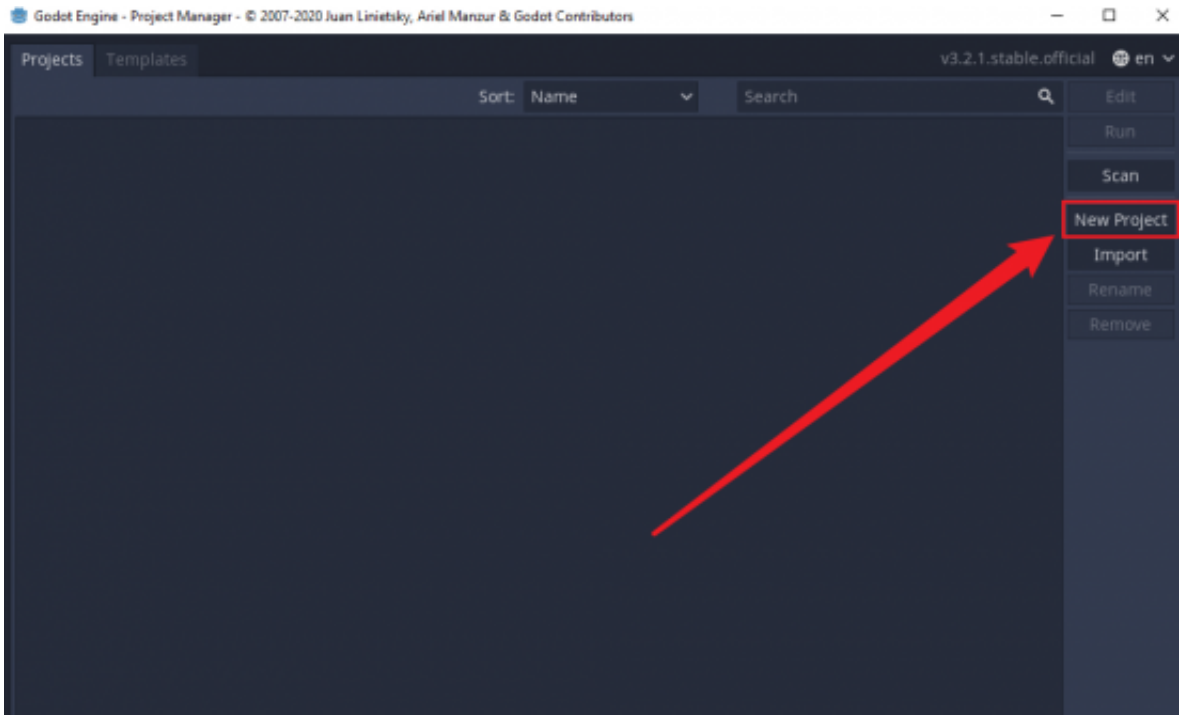


Godot Editor

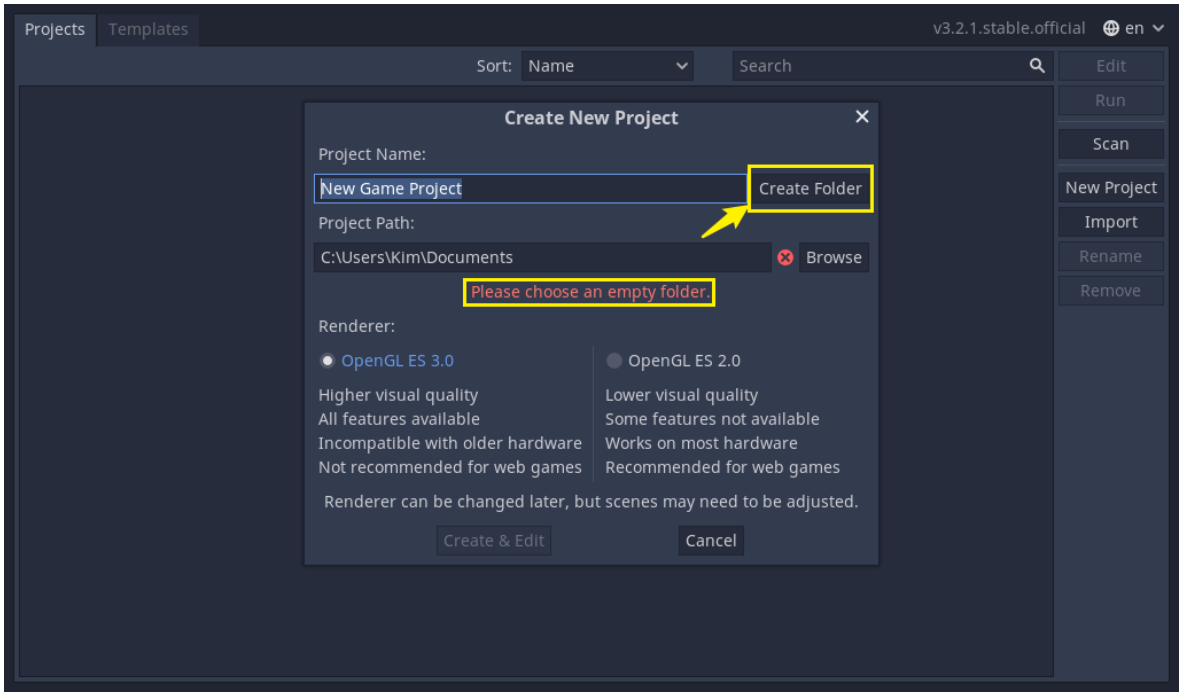
In this lesson, we are going to be creating our first Godot project.

Creating a new project

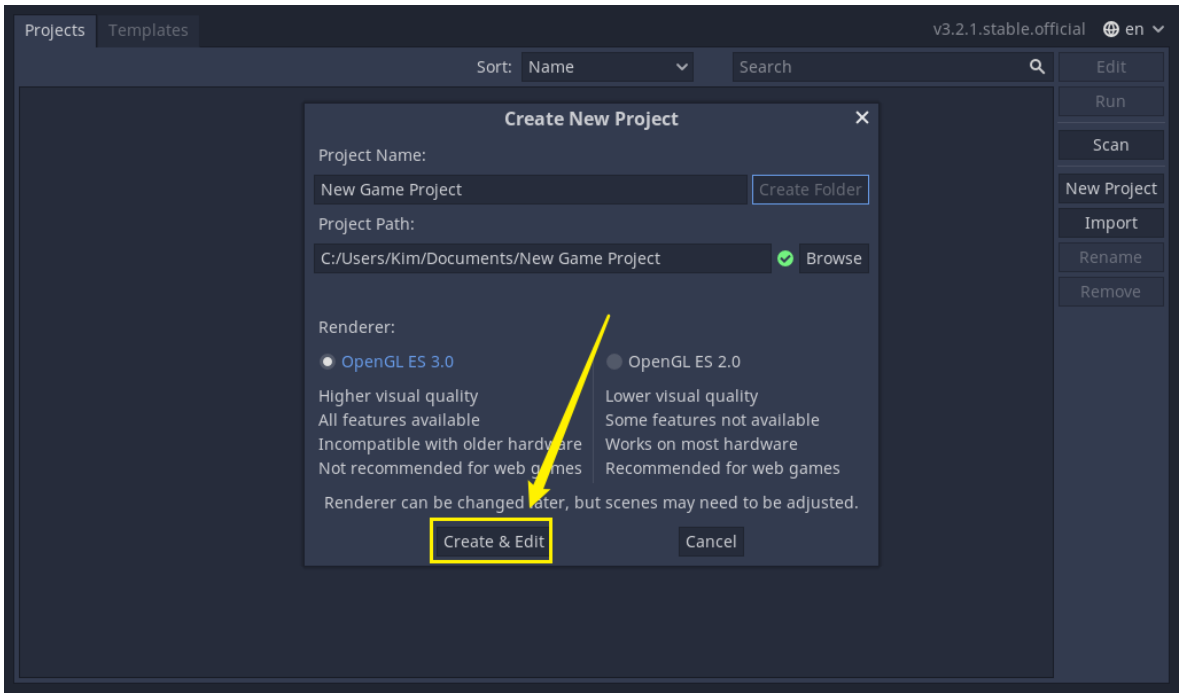
1. Open up the Godot Editor, and click on the **New Project** button.



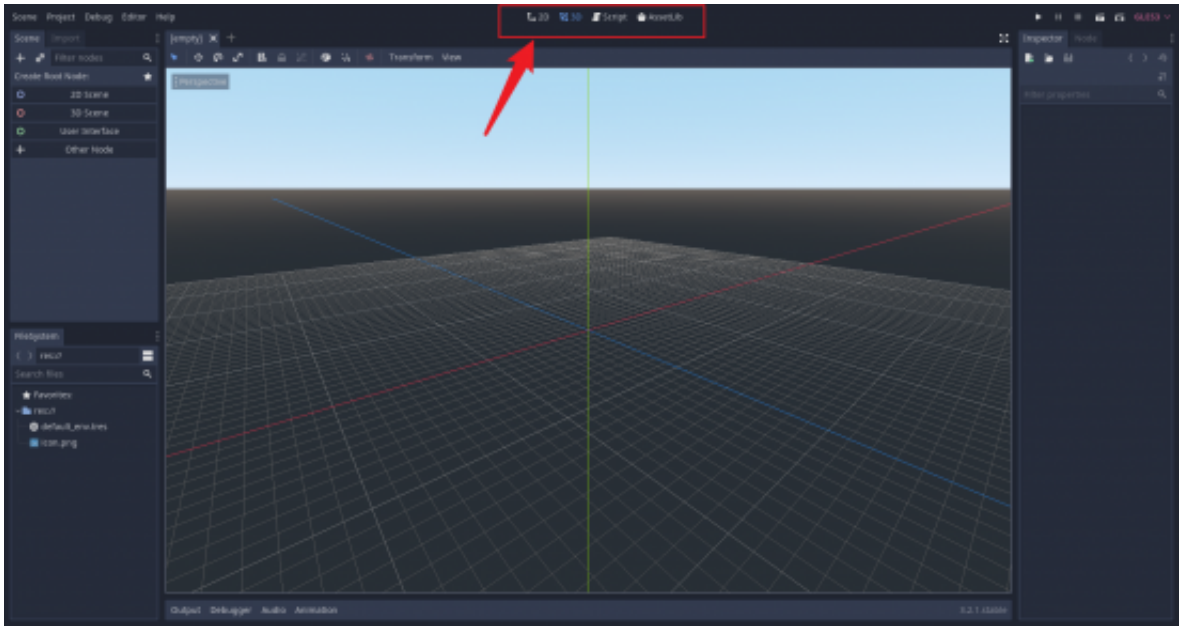
2. You need to either browse an empty folder or click on **Create Folder**.



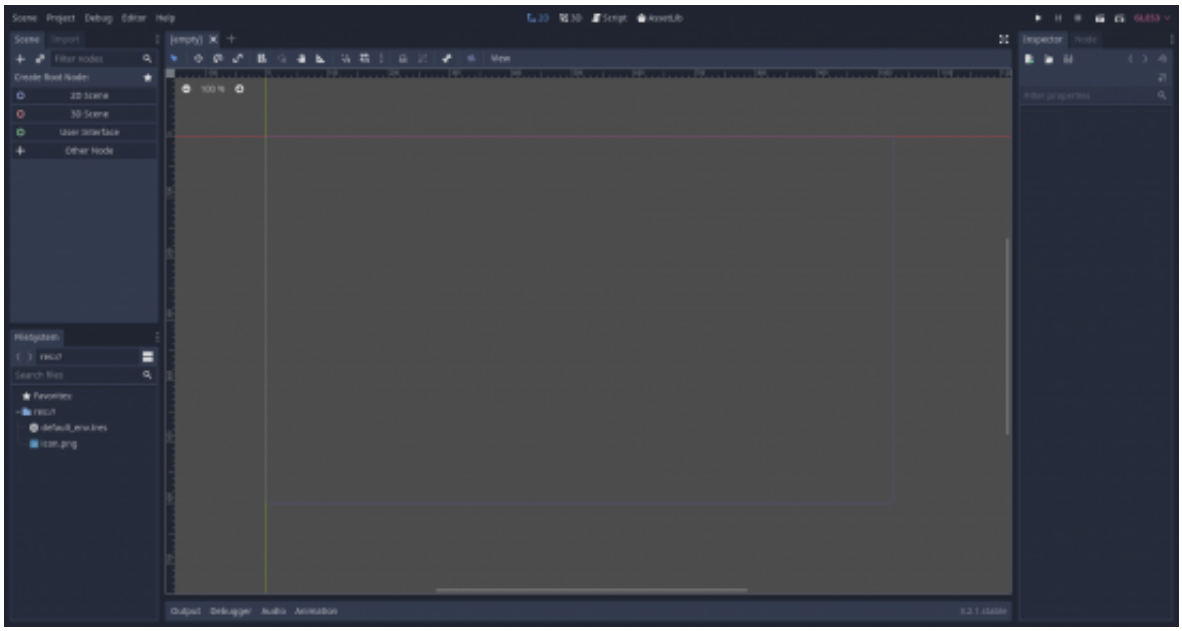
3. Click on **Create & Edit**.



With your project open, you should see the editor's interface with the 3D viewport active. You can change the current workspace at the top of the interface.

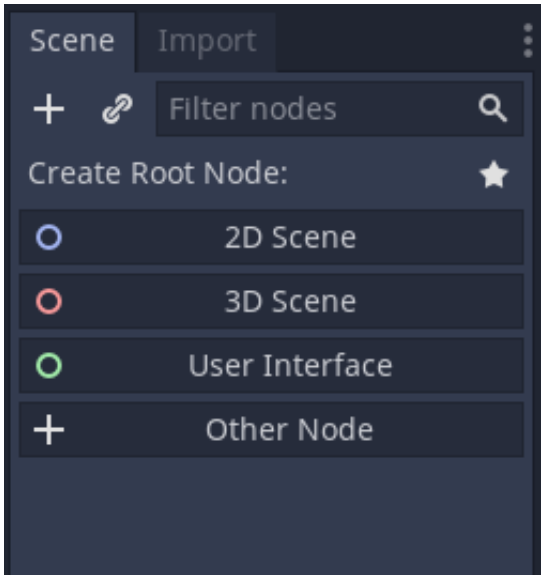


Click on **2D** to switch to the **2D workspace.**



Scene Panel

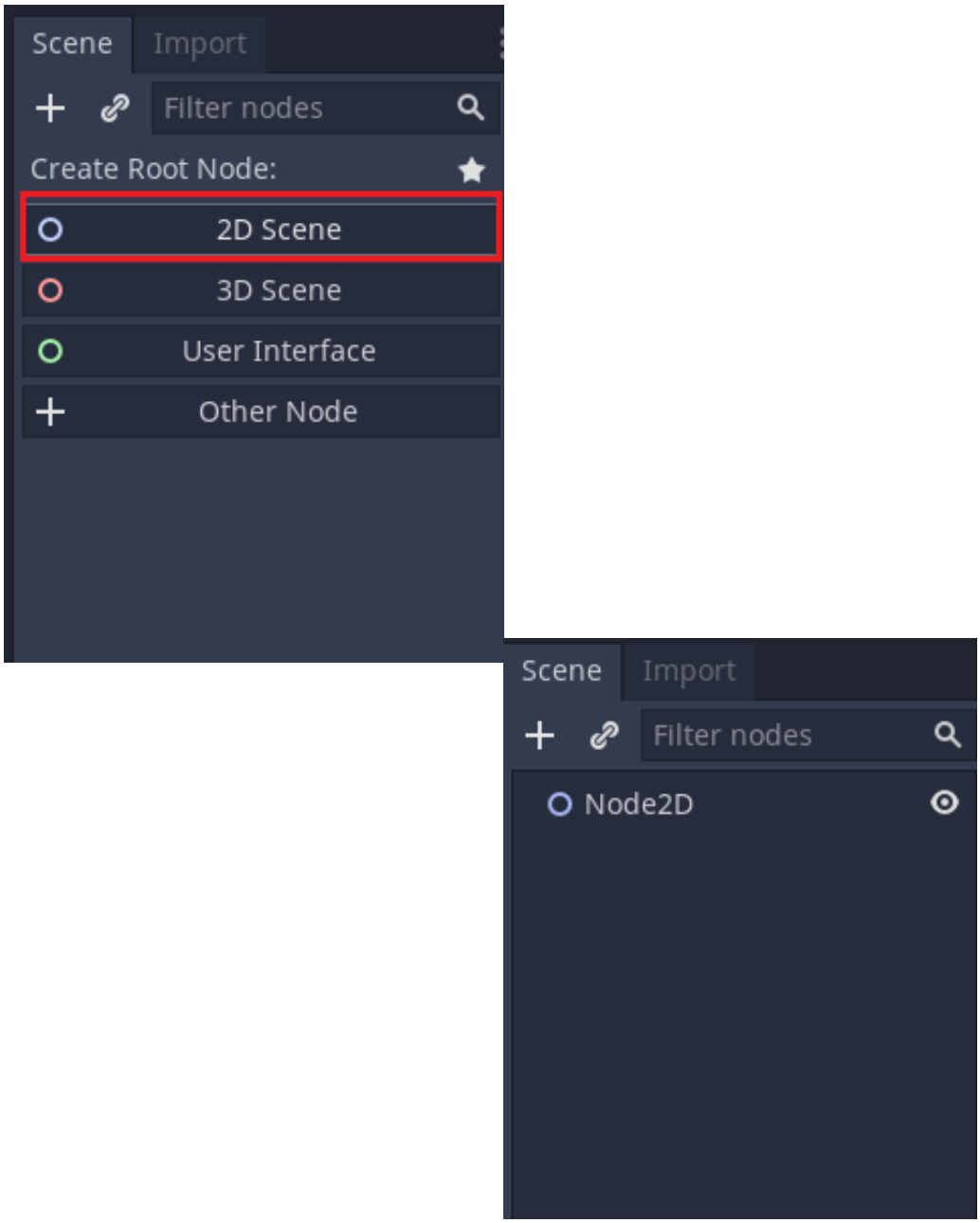
On the top right corner of the interface is the **Scene Panel**. You can see the list of all the **nodes** in our current scene.



Nodes

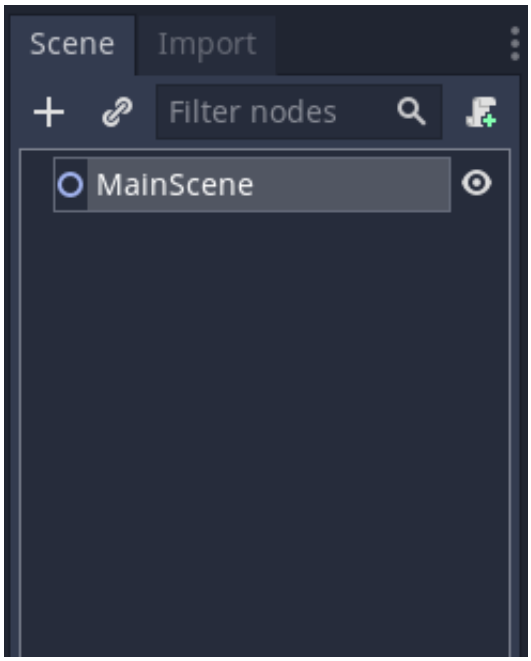
Nodes are the basic building blocks for creating games in Godot. They encapsulate data and behaviour, and they can inherit properties from other nodes.

To create a **Root Node** in the scene, you can simply click on 2D Scene.



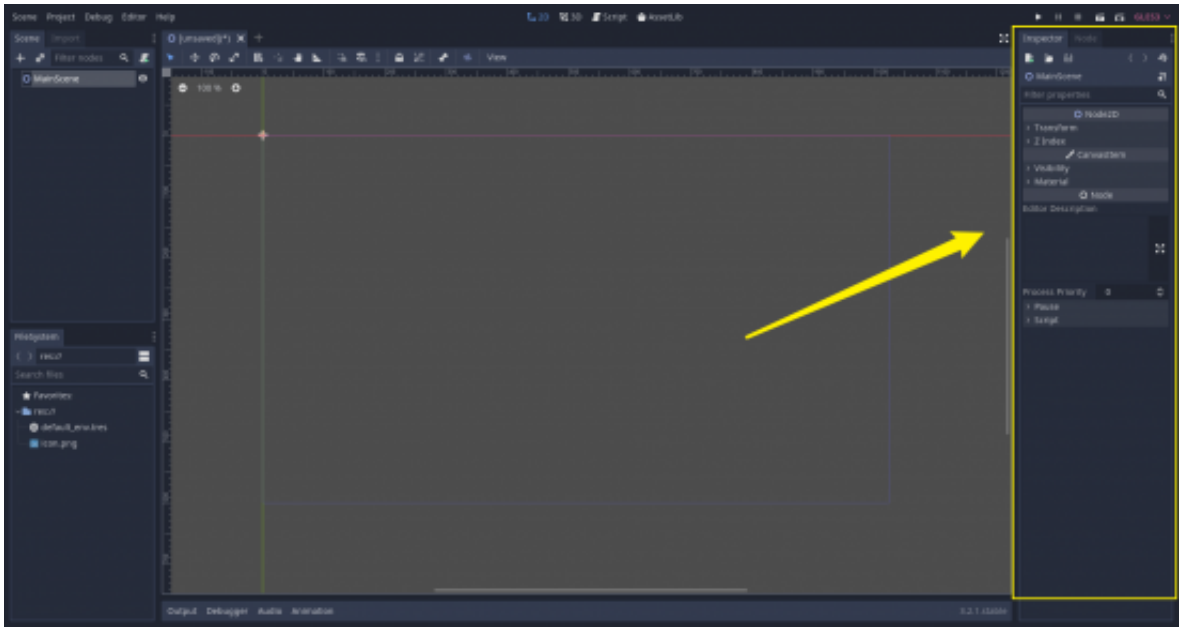
A new root node is created.

We can rename this node to ‘MainScene’.



Inspector

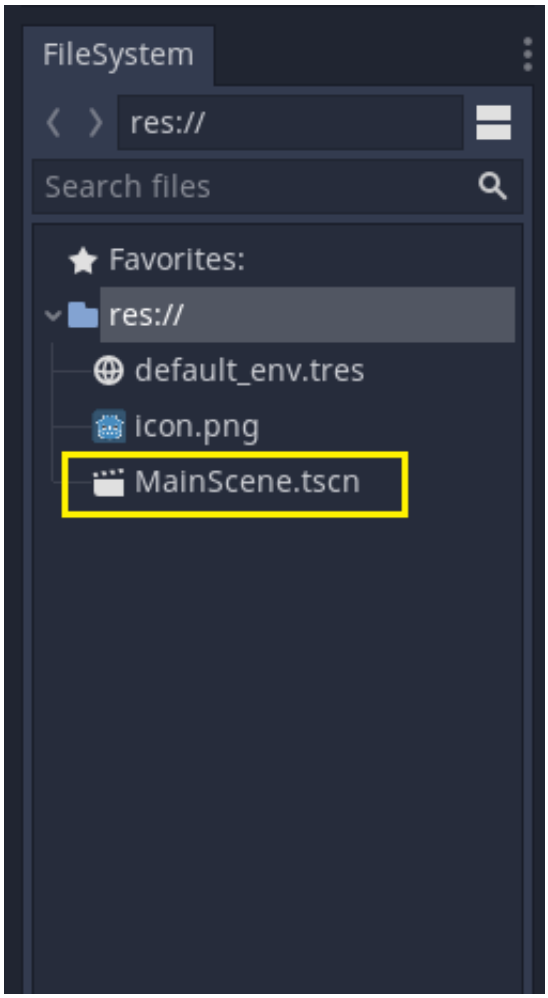
On the right-hand side, in the **Inspector panel**, a list of all the properties of the node is shown.



File System

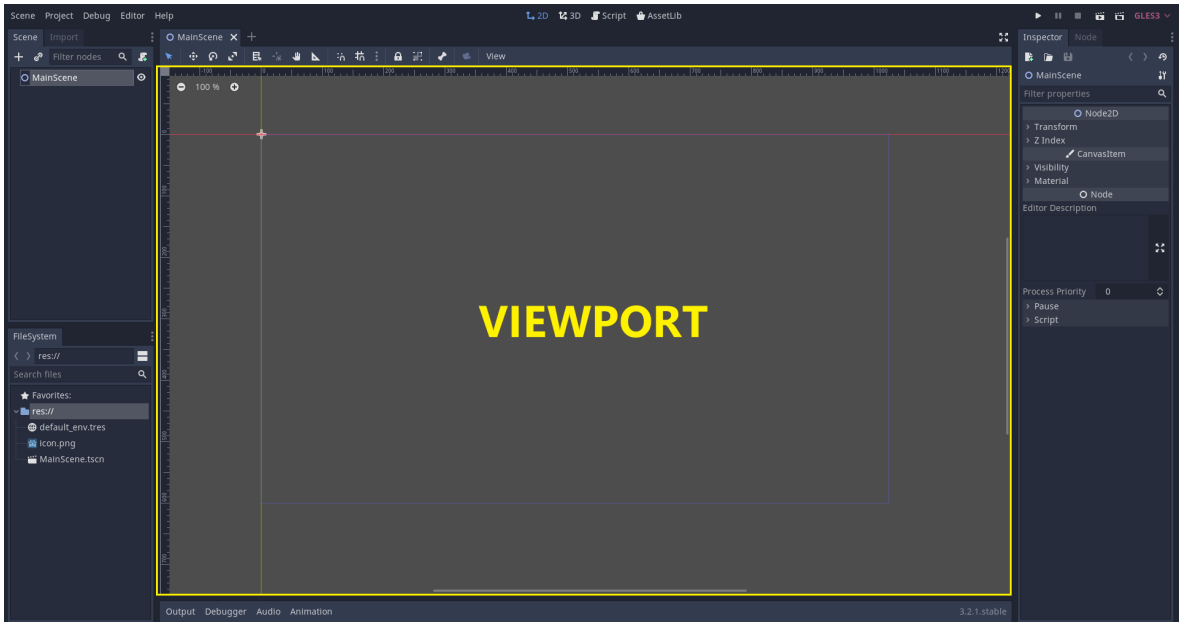
On the bottom left-hand side is the **FileSystem** panel, where we can view our file structure.

Let's save this scene by pressing **Ctrl+S**. You will now see that the new MainScene object is added to FileSystem.



Scene Viewport

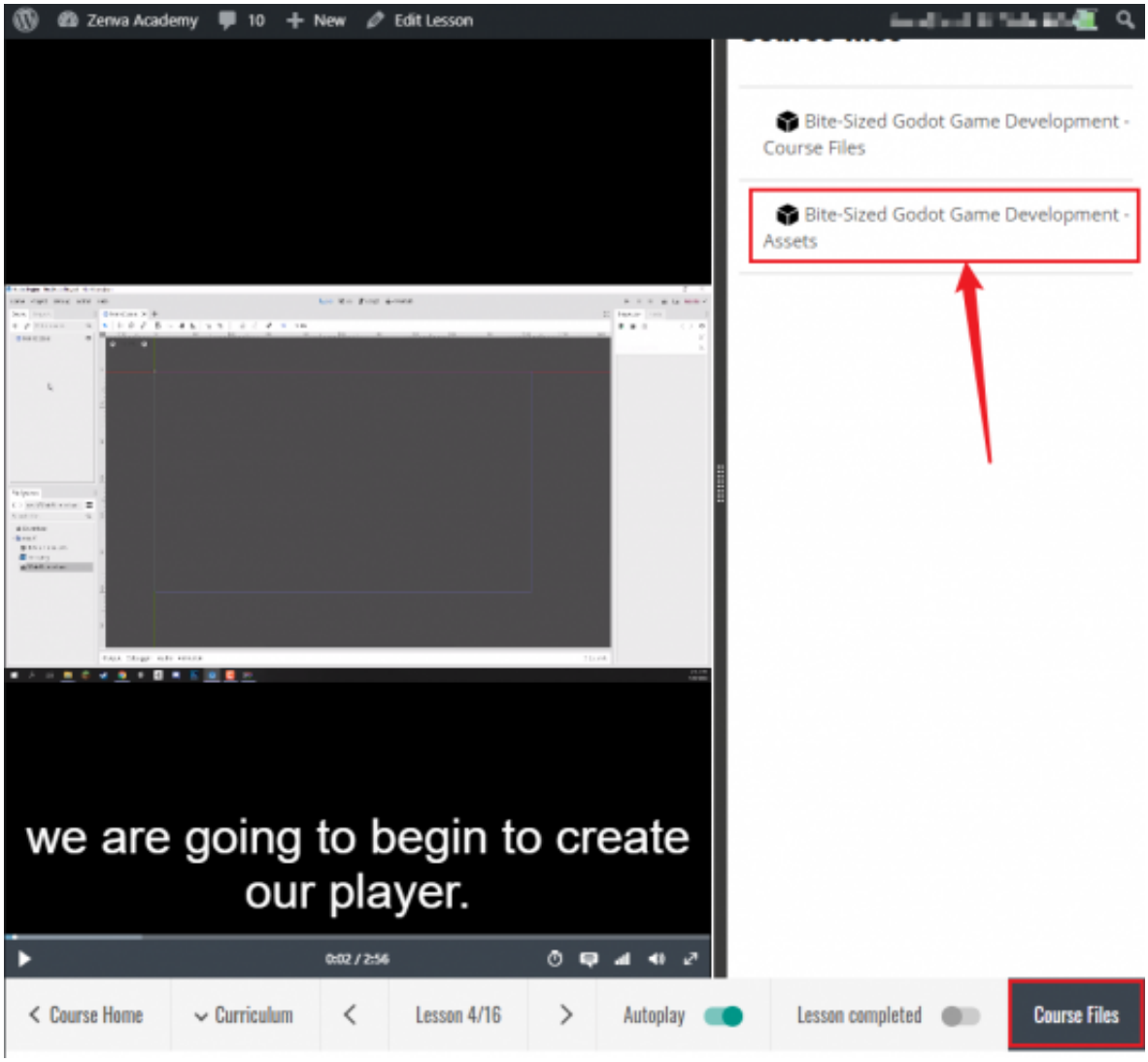
The center of the screen is the scene viewport. This is where we can see and interact with the rendered objects.



In this lesson, we are going to begin to create our player.

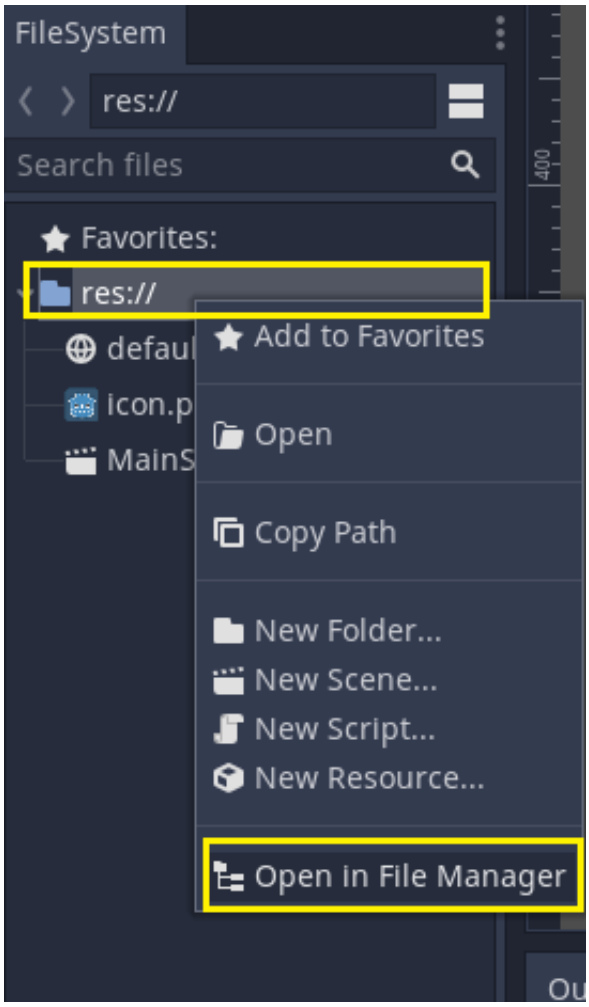
Importing assets

1. Download the included assets which come with this course.

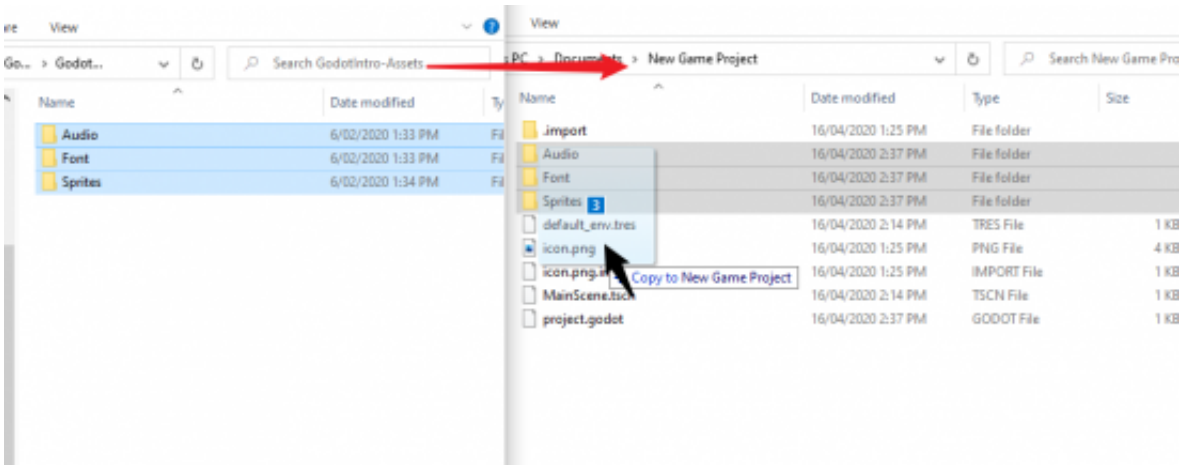


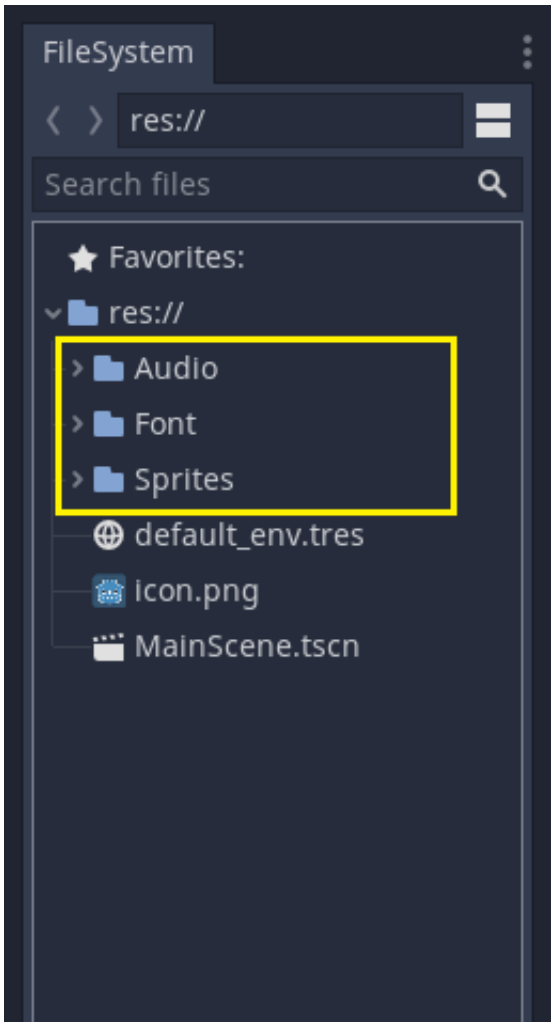
2. Select the root folder in **FileSystem**.

3. Right-click > **Open in File Manager**:

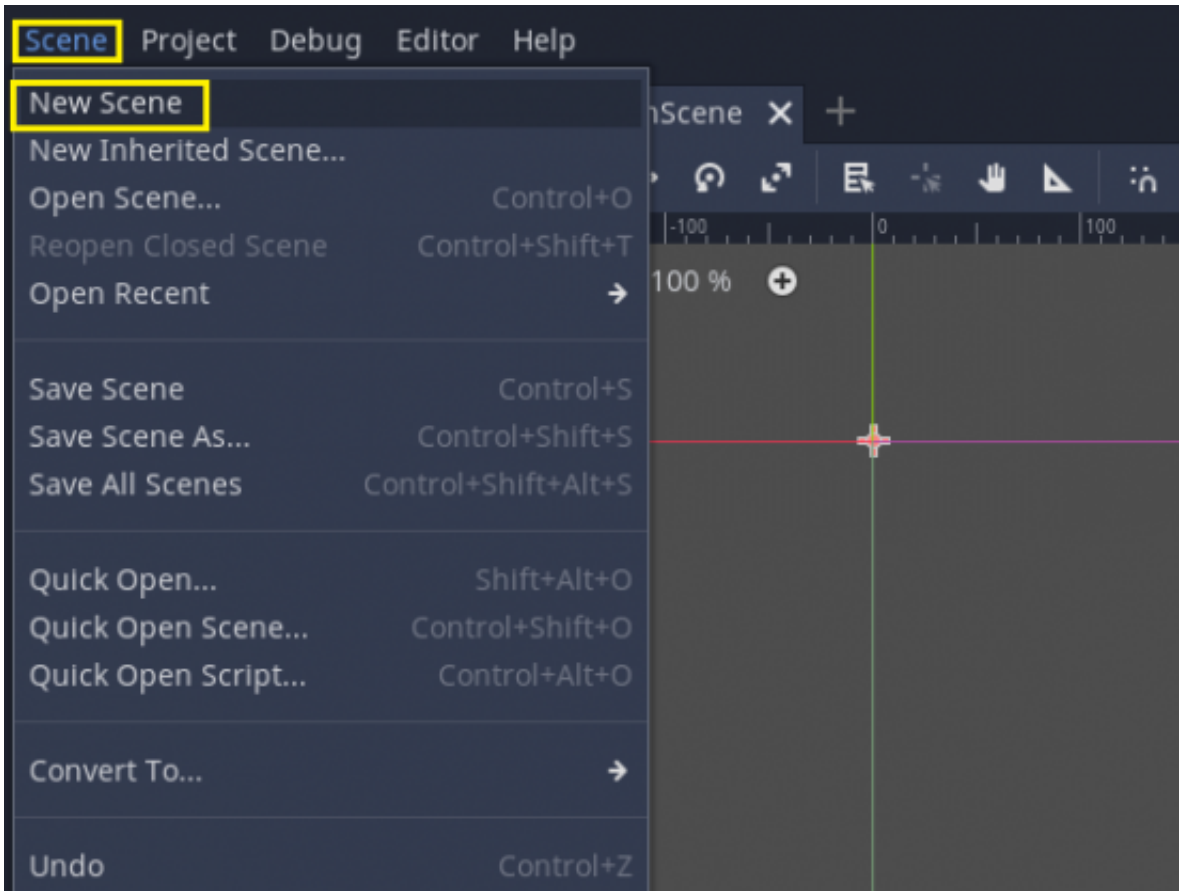


4. Drag and drop the downloaded assets (Font, Sprites, Audio) into the project folder.

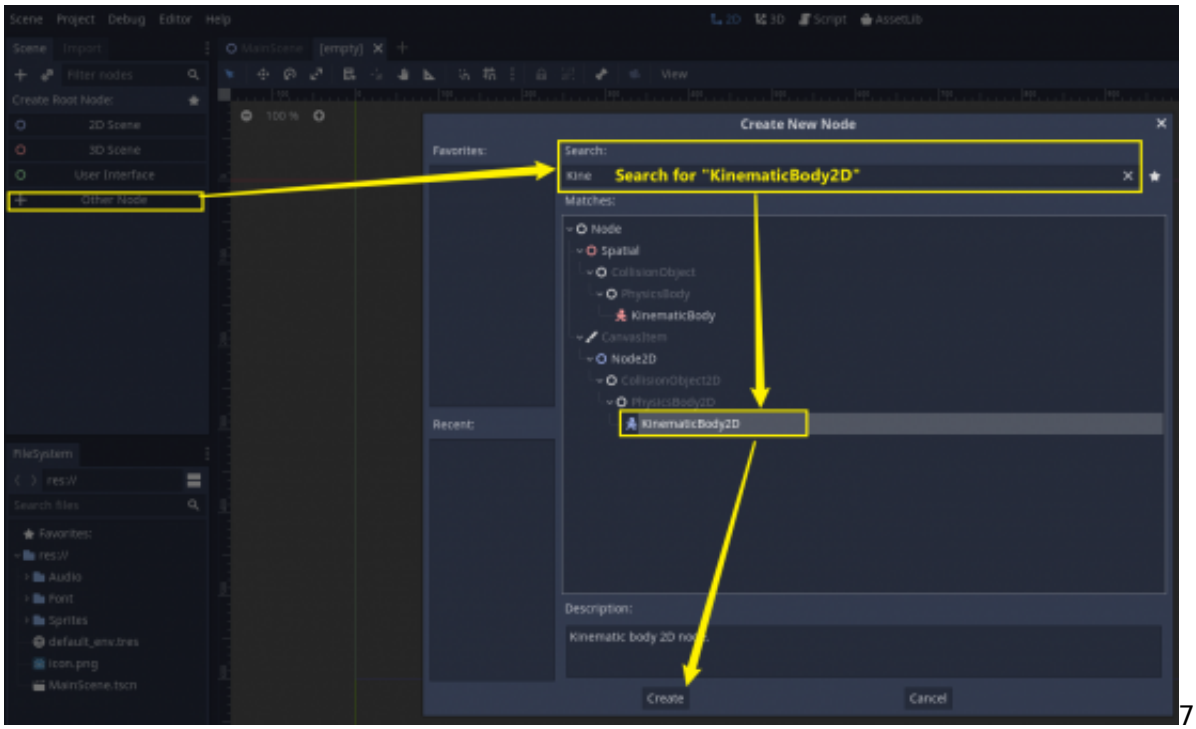




5. Click on **Scene** > New Scene



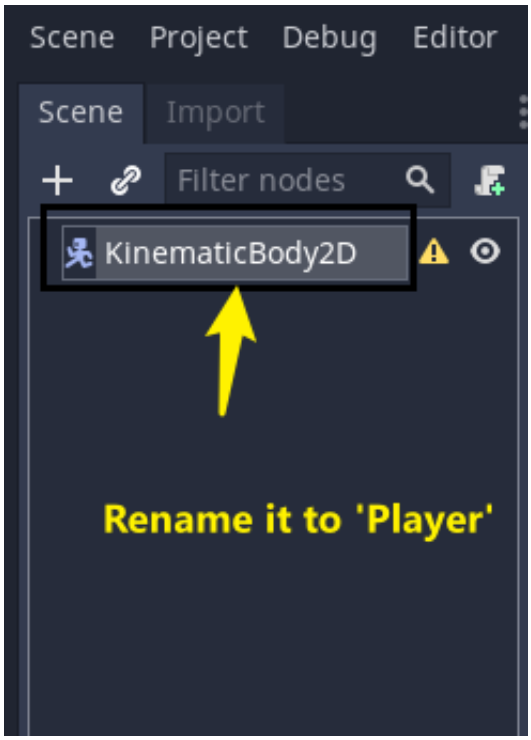
6. Click on **Other Nodes** in the Scene window.



7. Search for ‘KinematicBody2D’ in the Create New Node window.

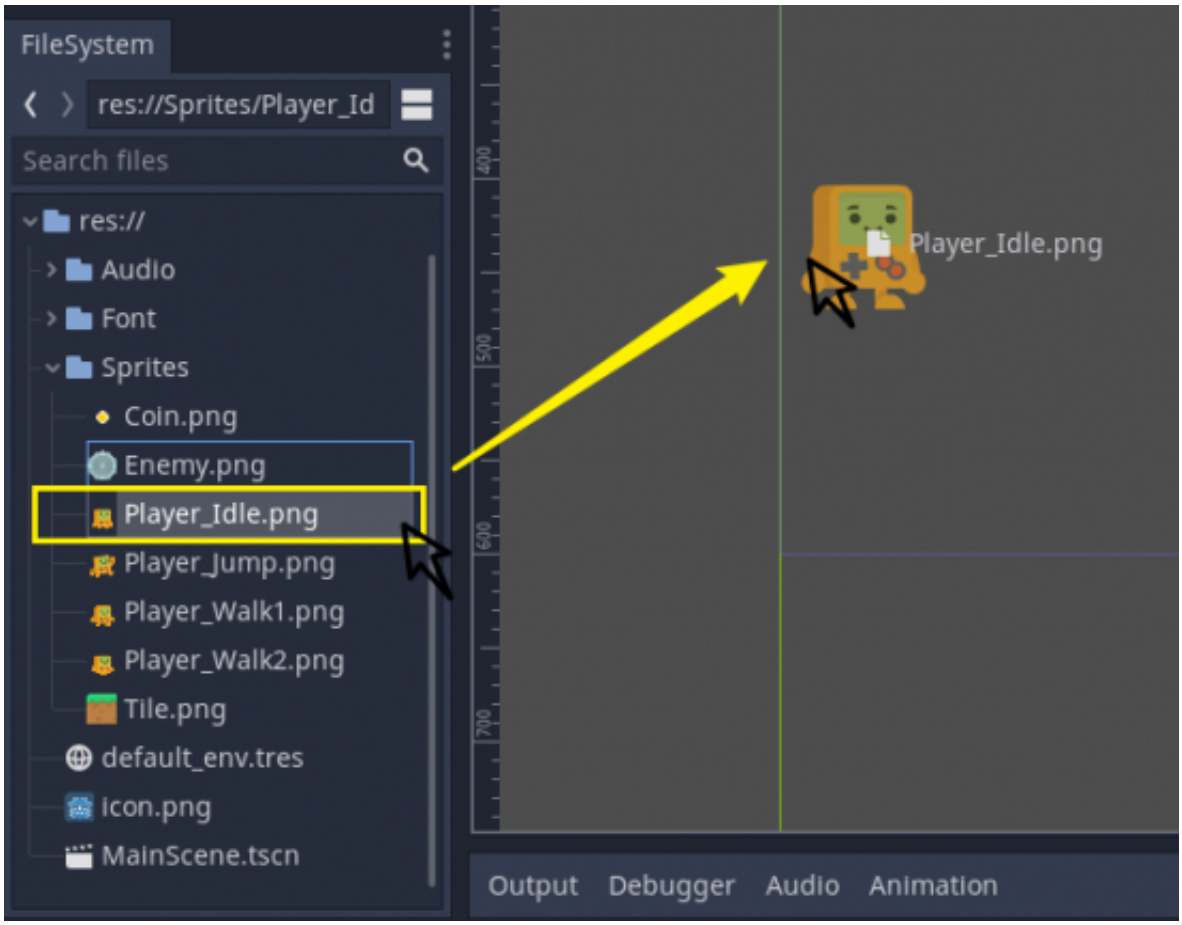
8. Select the ‘KinematicBody2D’ node and hit ‘Create’.

9. The new node is now visible in the Scene window.

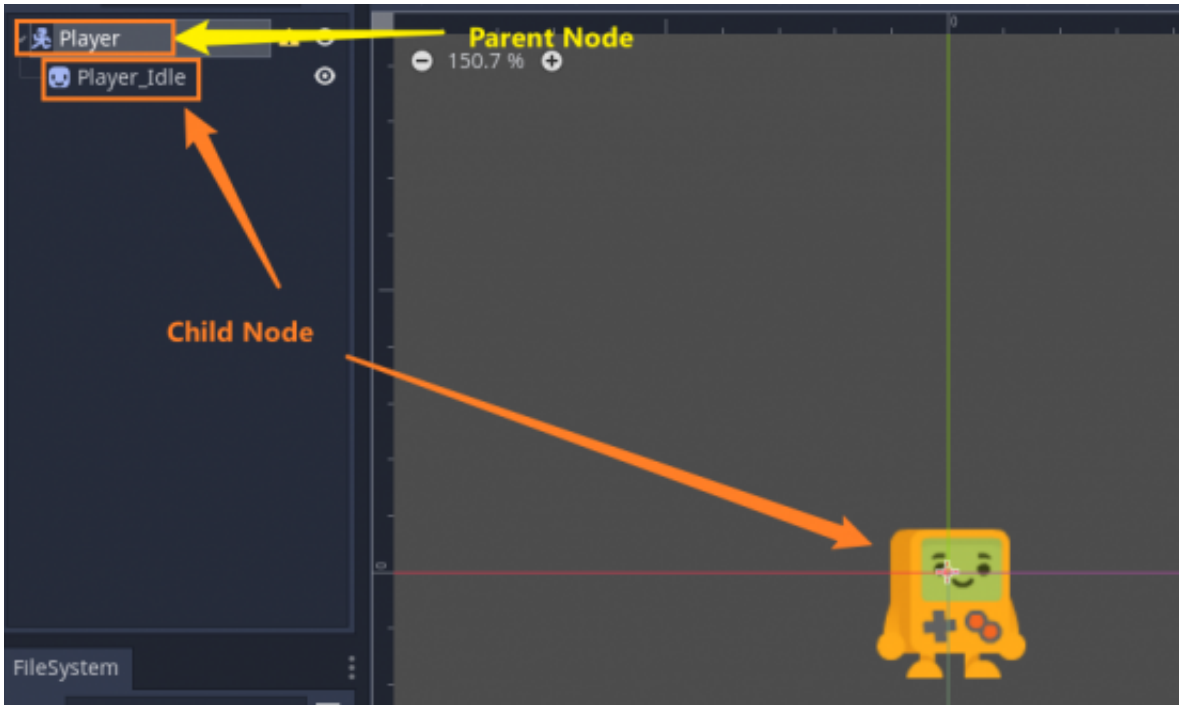


10. In FileSystem, double-click and open Sprites folder.

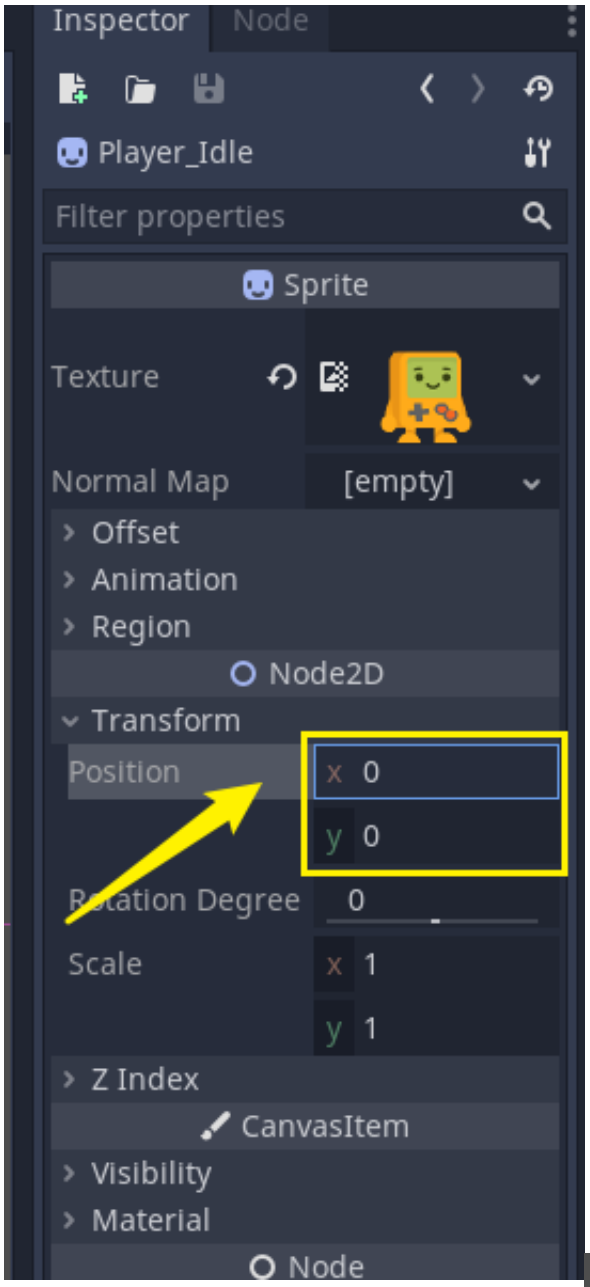
11. Select the ‘Player_Idle’ file and drag it into the viewport.

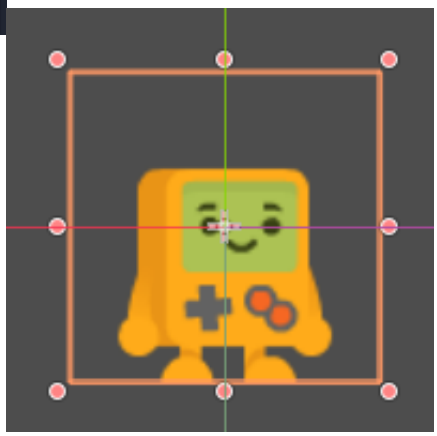


12. This creates a new node as a child of the Player node.



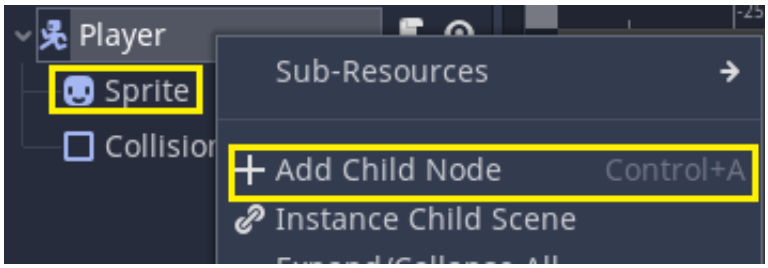
13. In the inspector, go to Transform > Position. Set x and y to 0.



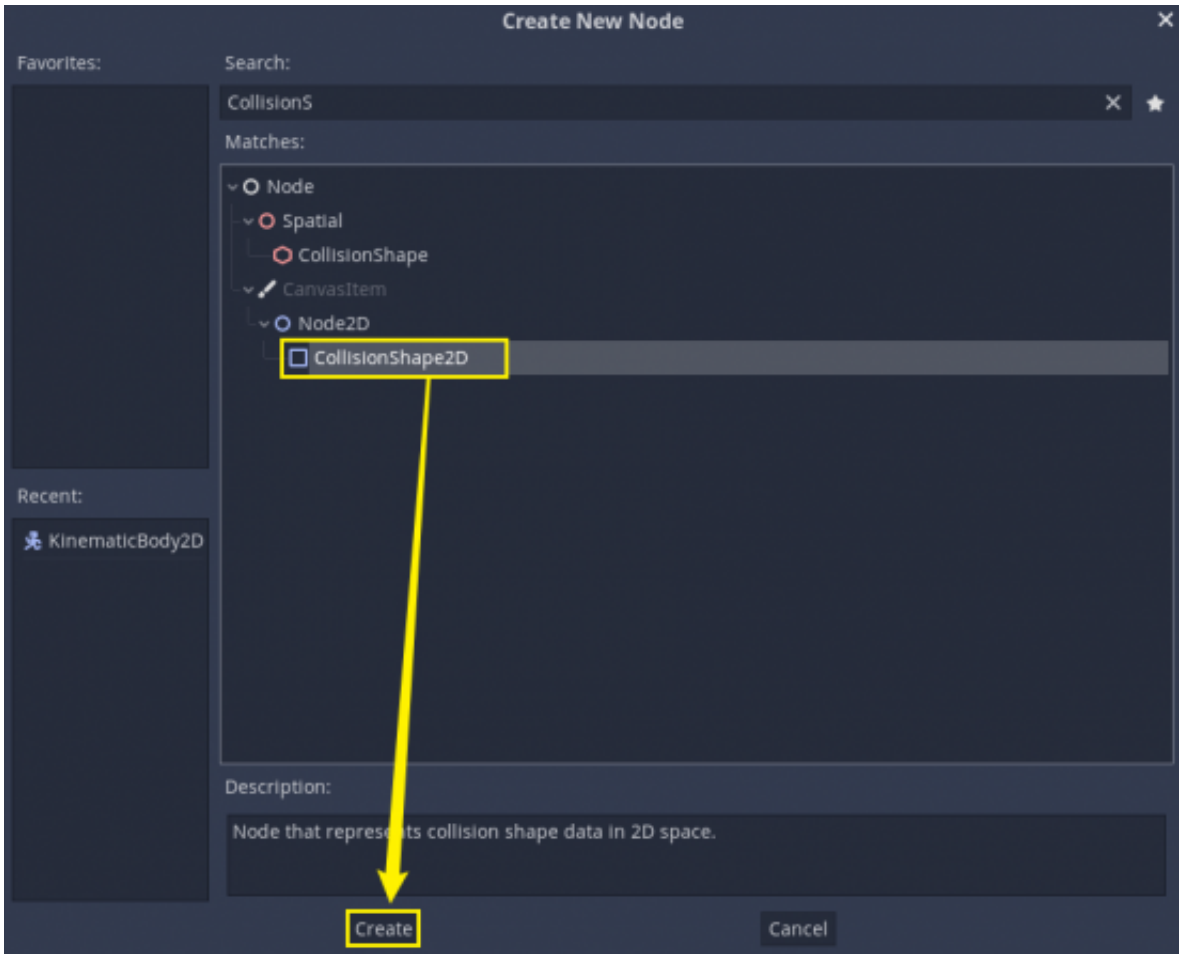


*The sprite is now centered in the origin (0, 0). Hit **Ctrl+S** to save the scene.*

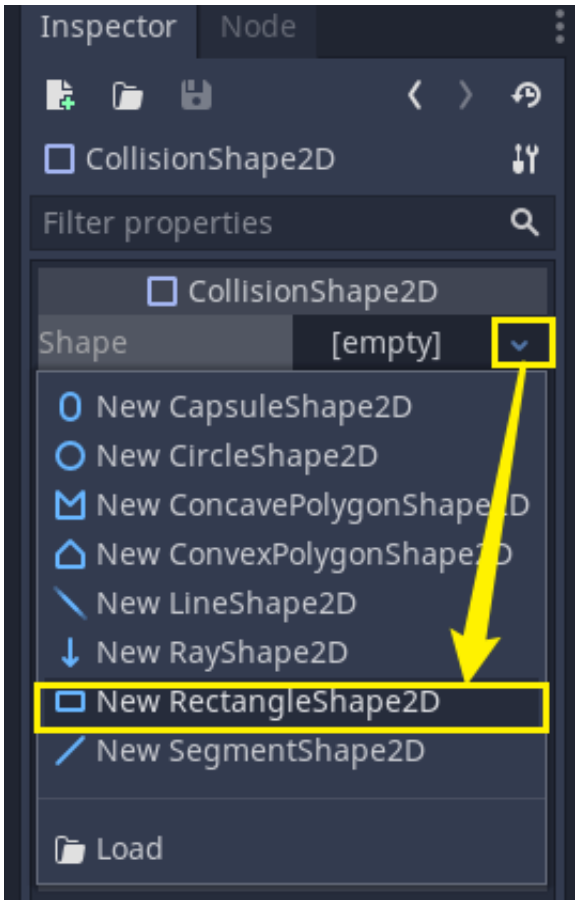
14. Rename the sprite from 'Player_Idle' to 'Sprite'. Right-click on the Player, and go 'Add Child Node':



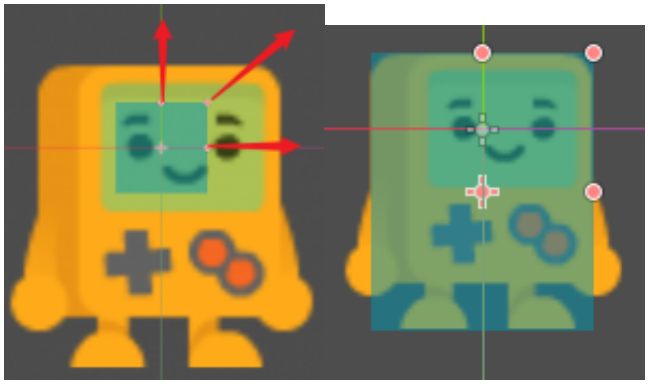
15. Create a CollisionShape2D:



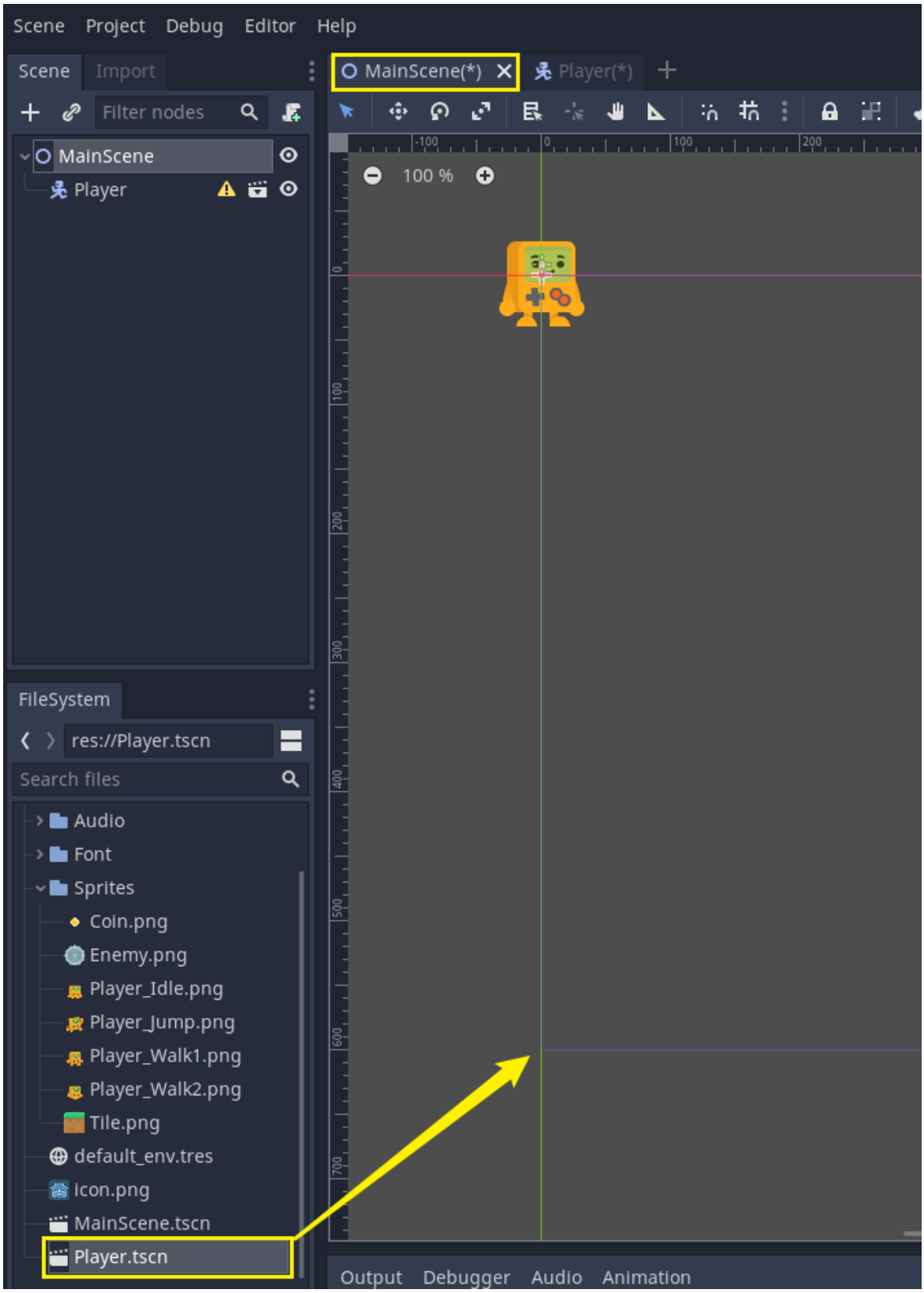
16. Go to Inspector > Click on ‘Shape’ > Select ‘New RectangleShape2D’



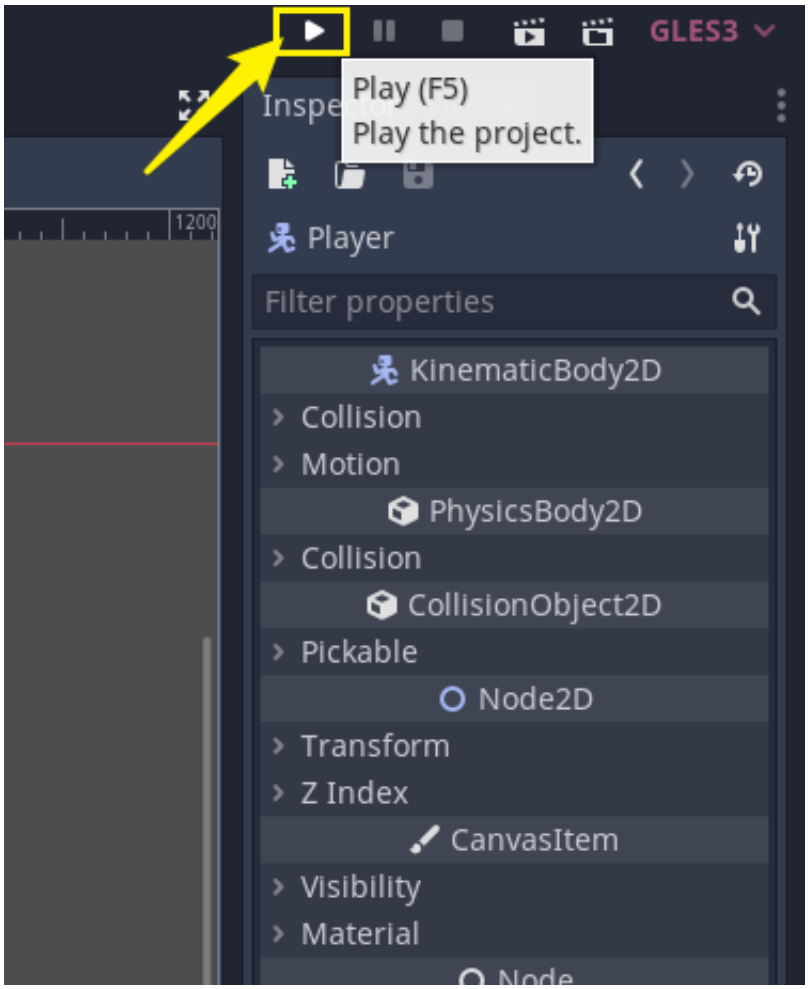
17. With the collider selected, drag one of the red dots to adjust the size so that it aligns with the sprite.



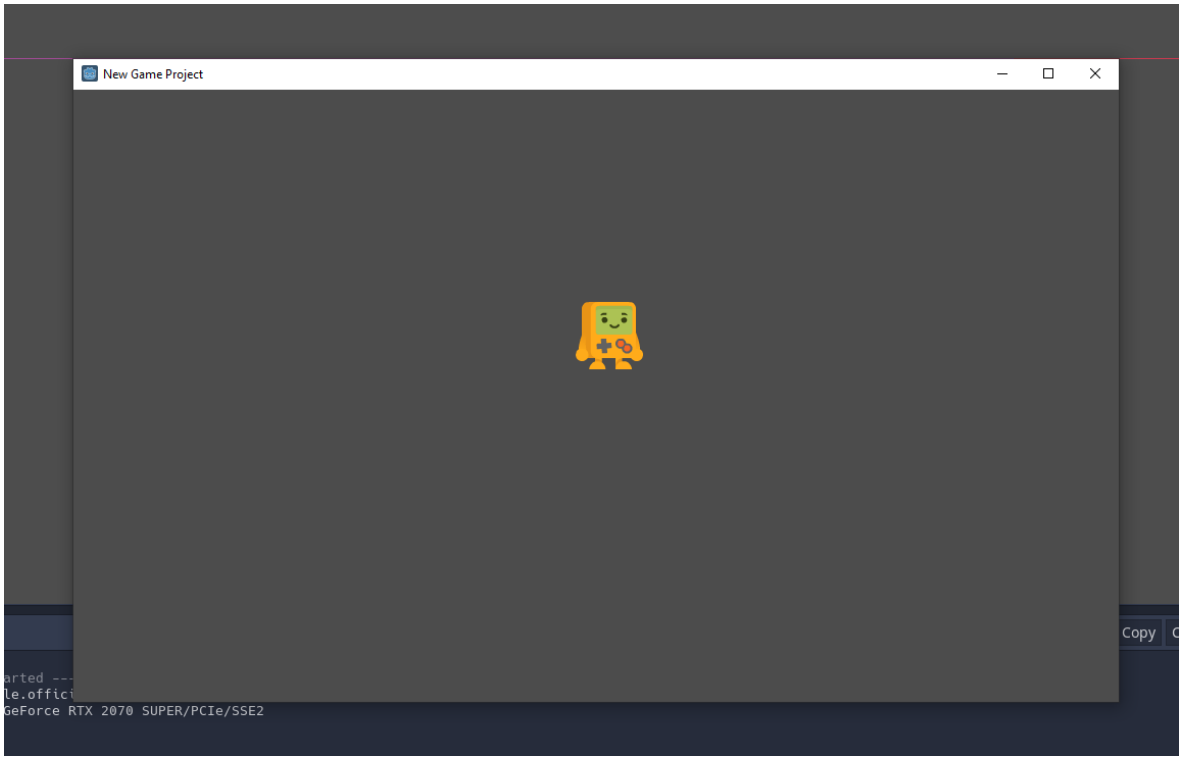
18. Click on 'MainScene'. Go to FileSystem and drag the **Player.tscn** into the main scene.



19. Save the scene and click on the '**Play**' button.

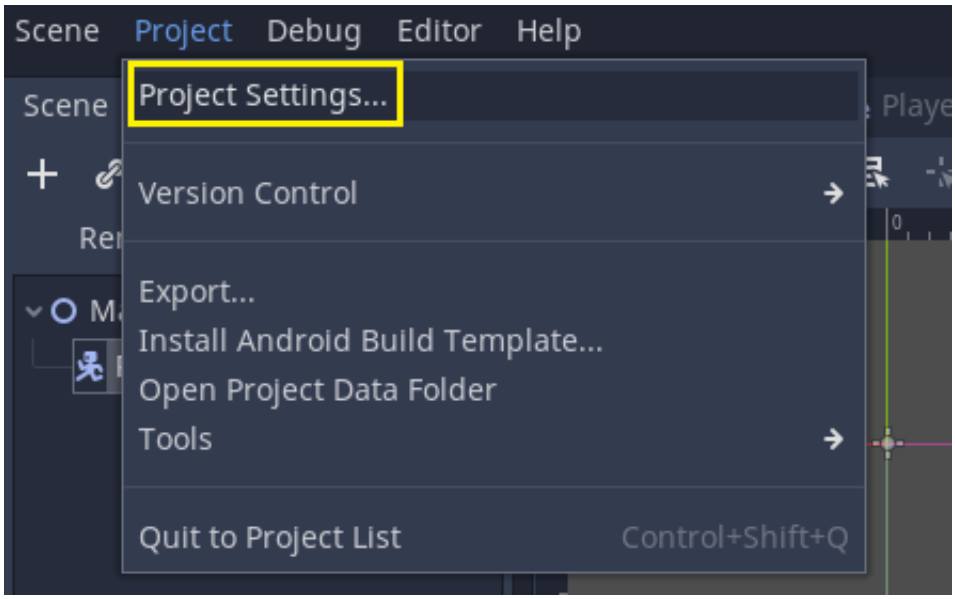


20. Our player is showing up in the game window.



In this lesson, we are going to begin scripting the player.

Go to **Project** > **Project Settings** > **Input Map.**

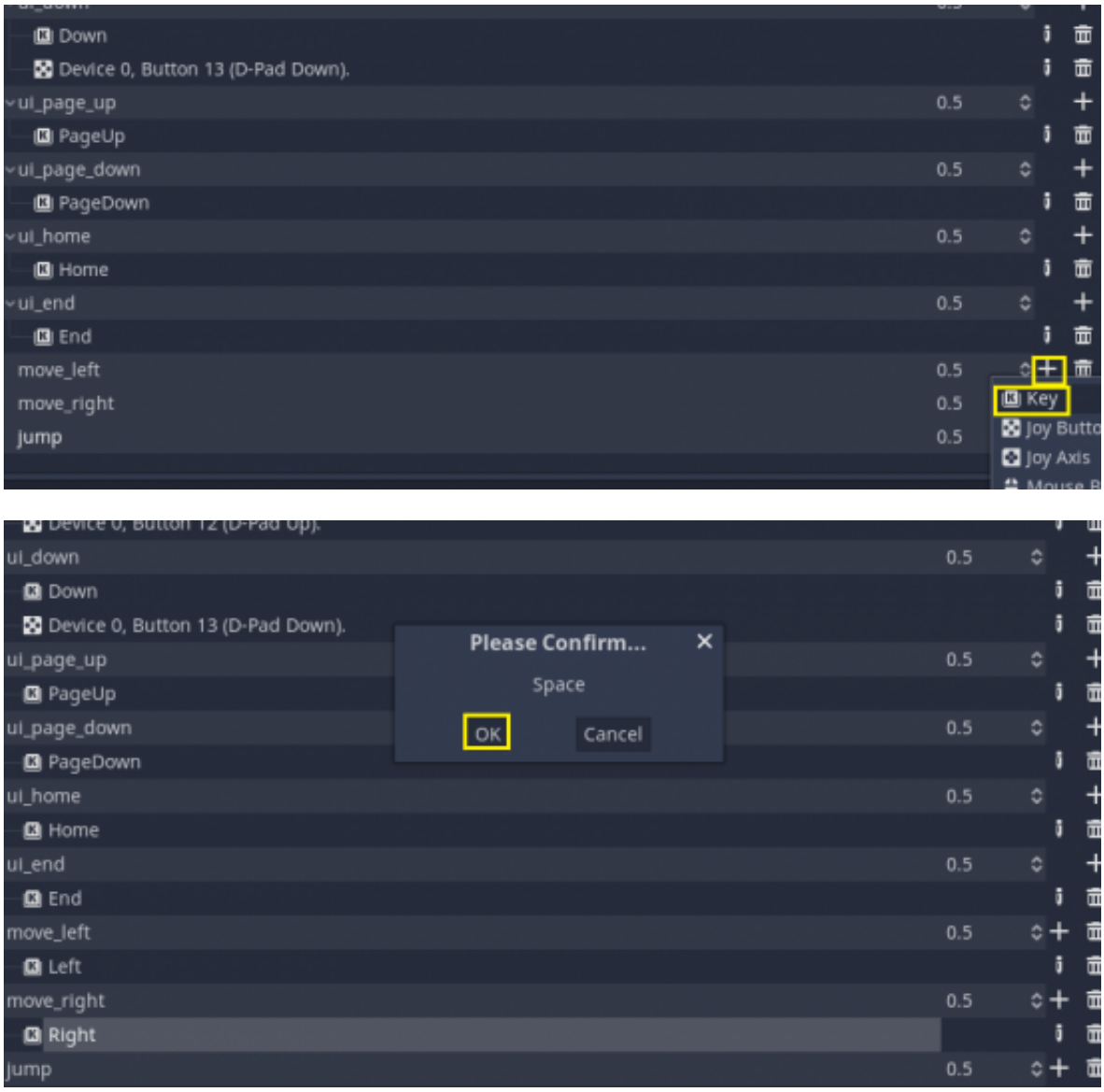


Creating New Actions

We can create new actions by giving the action names.

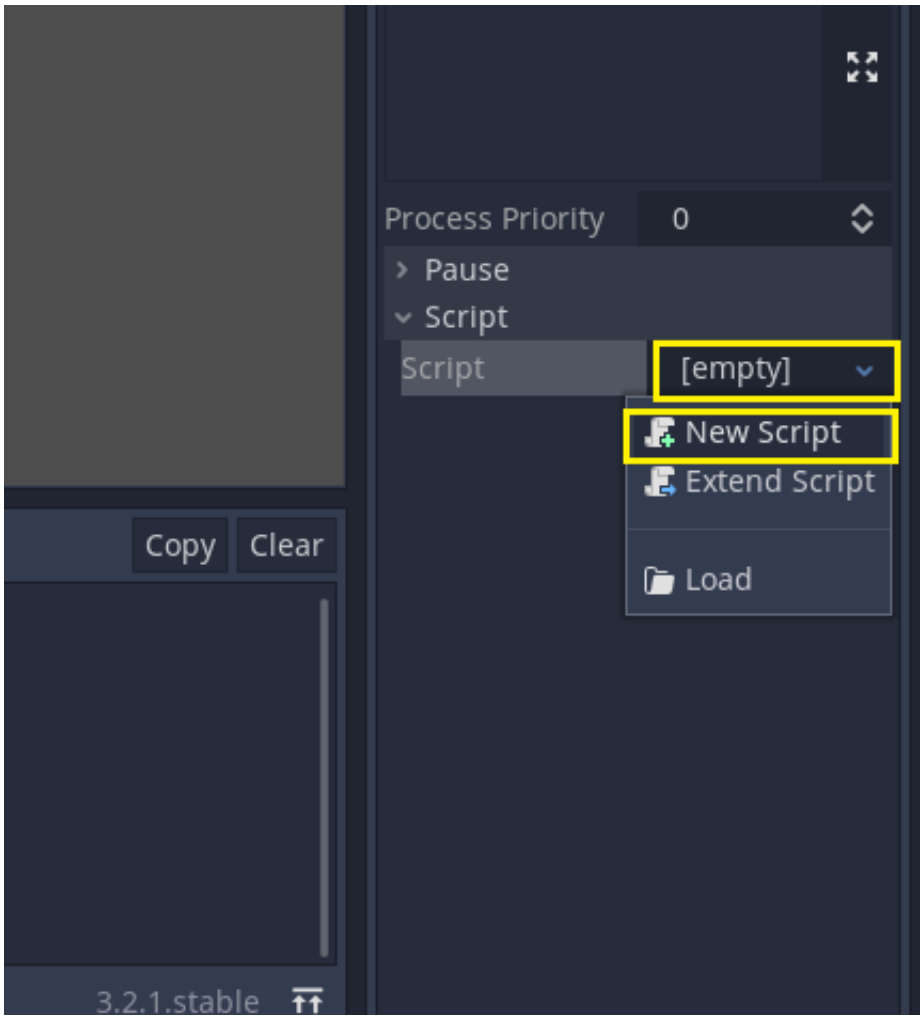
We will add ‘move_left’, ‘move_right’ and ‘jump’ then click on **Add.**

We now need to assign each action a unique key. Simply click on the ‘**+**‘ button and press the relevant key on your keyboard.

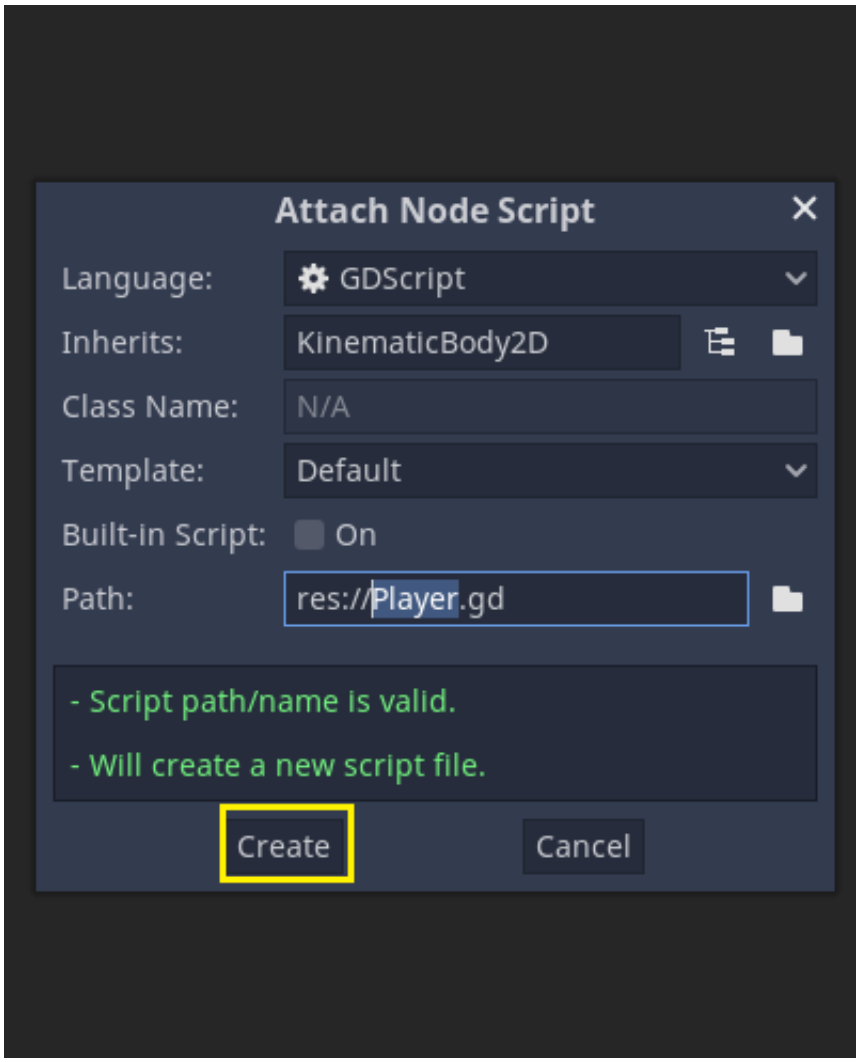


Close the project settings window.

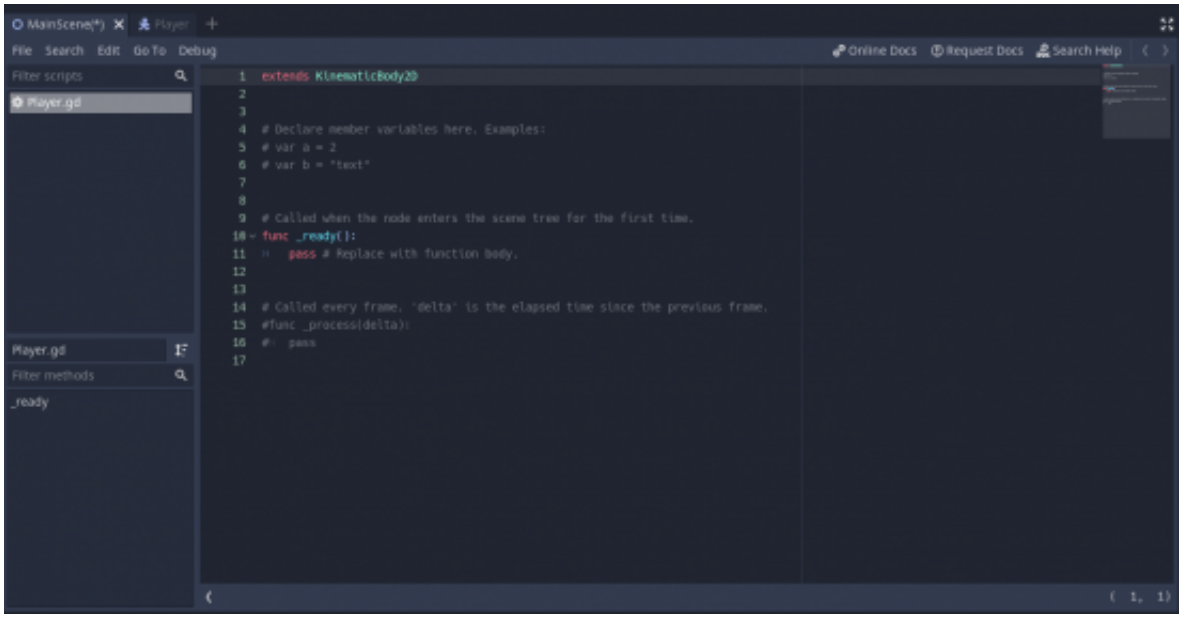
With the Player node selected, if you go to the Inspector panel, you will see that there is a **Script** tab. Expand it and click on **[empty]**. Then select 'New Script' in the dropdown.



Hit on the ‘Create’ button.



This will automatically open up a **Scripting Interface**.



Creating a variable

(What is a **variable**? A variable is basically a container that can hold data. The value may change depending on conditions or on commands passed to the program.)

To create our first variable, type '**var**' and then enter a **variable name**. Then, we can give it an **initial value** of zero by entering ' **= 0**'.

```
extends KinematicBody2D

var score = 0
```

So now the score variable has a data type of **int**. You can also add a **type hint** with a colon (:) after the name to keep it as an integer, which is a whole number.

```
extends KinematicBody2D

var score : int = 0
```

You can also set a different data type, such as **Vector2** or **sprite**. Vector2 holds two numeric values that represent the x, y coordinates of a point.

```
extends KinematicBody2D

var score : int = 0

var speed : int = 200
var jumpForce : int = 600
var gravity : int = 800

var vel : Vector2 = Vector2()

var sprite : Sprite = $Sprite();
```

Note that the **dollar sign ($) operator** is used in front of Sprite();. This basically tells the script to look through all of the children nodes and find the node of whatever type is after the $, which is Sprite in this case.

Scripting The Player – Part 2

The alternate syntax to the dollar sign ($) operator is to use the **get_node** function. Simply replace $Sprite() with **get_node("Sprite")**.

```
var sprite : Sprite = get_node("Sprite")
```

We can also add a keyword called '**onready**' before 'var' to make sure the asset is ready when the scene loads.

```
onready var sprite : Sprite = get_node("Sprite")
```

This delays the initialization of the variable until the node of Sprite is found.

Moving The Player

Next, we will implement the ability for the player to move around and jump. First, we need to apply gravity. This is going to be done inside of a function called **_physics_process()**:

```
func _physics_process(delta):
```

This is a built-in function of Godot, which gets called 60 times a second to process physics. First, we want to set out velocity on the x value to be 0.

```
func _physics_process(delta):

  vel.x = 0
```

Then we will check our movement inputs: if the Input system detects 'move_left' action.

```
func _physics_process(delta):

  vel.x = 0

  #movement inputs
  if Input.is_action_pressed("move_left"):
```

Our velocity on the x should be negative speed when we press the "move_left" button, and positive speed when we press "move_right".

```
func _physics_process(delta):

  vel.x = 0
```



```
  #movement inputs
  if Input.is_action_pressed("move_left"):
     vel.x -= speed
  if Input.is_action_pressed("move_right"):
     vel.x += speed
```

Move_and_slide()

When moving a KinematicBody2D, you should not set its position property directly. Rather you should use the **move_and_collide()** or **move_and_slide()** methods. (**Note that these methods should be used within the **_physics_process()** callback.*)

move_and_slide() method moves the body along a given vector, and if the body collides with another, it will slide along the other body. This is especially useful in platformers or top-down games.

To use the **move_and_slide()**, we need to first send over the velocity and the direction in which the floor is pointing. Since the ground is going to be below us, this would be **Vector2.Up**.

```
func _physics_process(delta):

  vel.x = 0

  #movement inputs
  if Input.is_action_pressed("move_left"):
     vel.x -= speed
  if Input.is_action_pressed("move_right"):
     vel.x += speed

  #applying the velocity
  move_and_slide(vel, Vector2.UP)
```

Flipping The Sprite

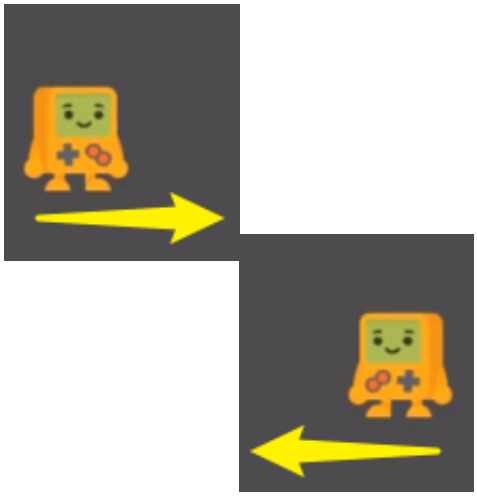
Now, we can assign it to our velocity and check which direction our player sprite should be facing towards.

```
func _physics_process(delta):

  vel.x = 0

  #movement inputs
  if Input.is_action_pressed("move_left"):
     vel.x -= speed
  if Input.is_action_pressed("move_right"):
     vel.x += speed

  #applying the velocity
  vel = move_and_slide(vel, Vector2.UP)

  #sprite direction
```



```
  if vel.x < 0:
     sprite.flip_h = true
  elif vel.x > 0:
     sprite.flip_h = false
```

Save the script and press Play:



You can now move the Player horizontally with arrow keys. The sprite will flip as the moving direction changes.

Applying Gravity

```
func _physics_process(delta):

  vel.x = 0

  #movement inputs
  if Input.is_action_pressed("move_left"):
     vel.x -= speed
  if Input.is_action_pressed("move_right"):
     vel.x += speed

  #applying the velocity
  vel = move_and_slide(vel, Vector2.UP)

  #gravity
  vel.y += gravity * delta

  #sprite direction
  if vel.x < 0:
     sprite.flip_h = true
  elif vel.x > 0:
     sprite.flip_h = false
```


Applying gravity is pretty straightforward: we can add gravity directly onto our y velocity. Note that when you add/subtract from the velocity, it needs to be multiplied by delta so that it accelerates at the correct speed.

```
#gravity

vel.y += gravity * delta
```

Jump Input

We need to check if the 'jump' key is detected, and we also need to check if the player is on the floor:

```
  #jump input
  if Input.is_action_just_pressed("jump") and is_on_floor():
```

We then want to apply the jump force in the opposite direction of gravity:

```
  #jump input
  if Input.is_action_just_pressed("jump") and is_on_floor():
    vel.y -= jumpForce
```

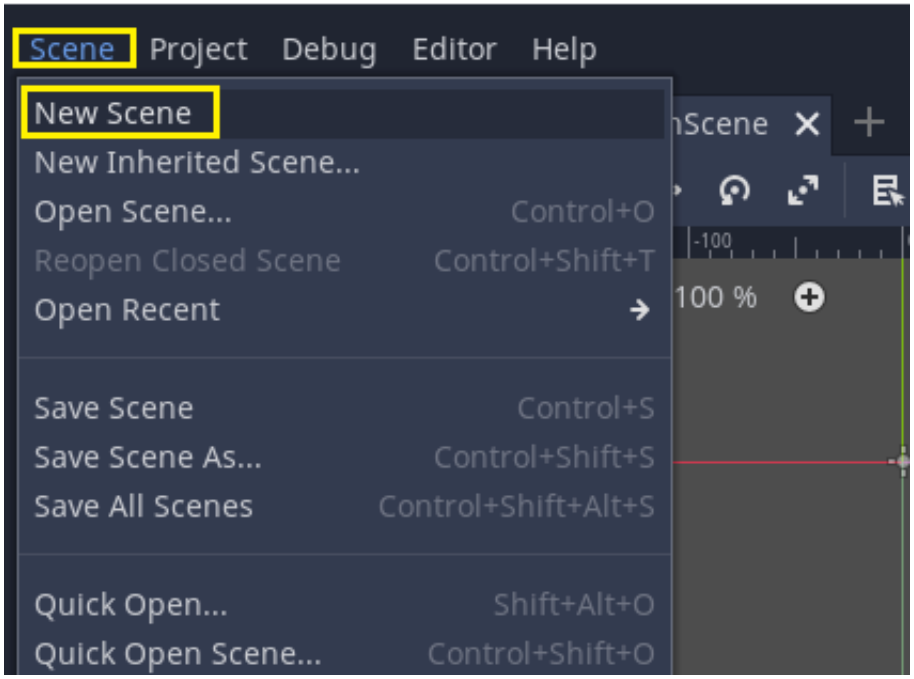
Make sure you have saved the script.

```
extends KinematicBody2D

var score : int = 0

var speed : int = 200
var jumpForce : int = 600
var gravity : int = 800

var vel : Vector2 = Vector2()

onready var sprite : Sprite = get_node("Sprite")

func _physics_process(delta):

 vel.x = 0

 #movement inputs
 if Input.is_action_pressed("move_left"):
  vel.x -= speed
 if Input.is_action_pressed("move_right"):
  vel.x += speed

 #applying the velocity
 vel = move_and_slide(vel, Vector2.UP)
```

```
#gravity
vel.y += gravity * delta

#jump input
if Input.is_action_just_pressed("jump") and is_on_floor():
 vel.y -= jumpForce

#sprite direction
if vel.x < 0:
 sprite.flip_h = true
elif vel.x > 0:
 sprite.flip_h = false
```

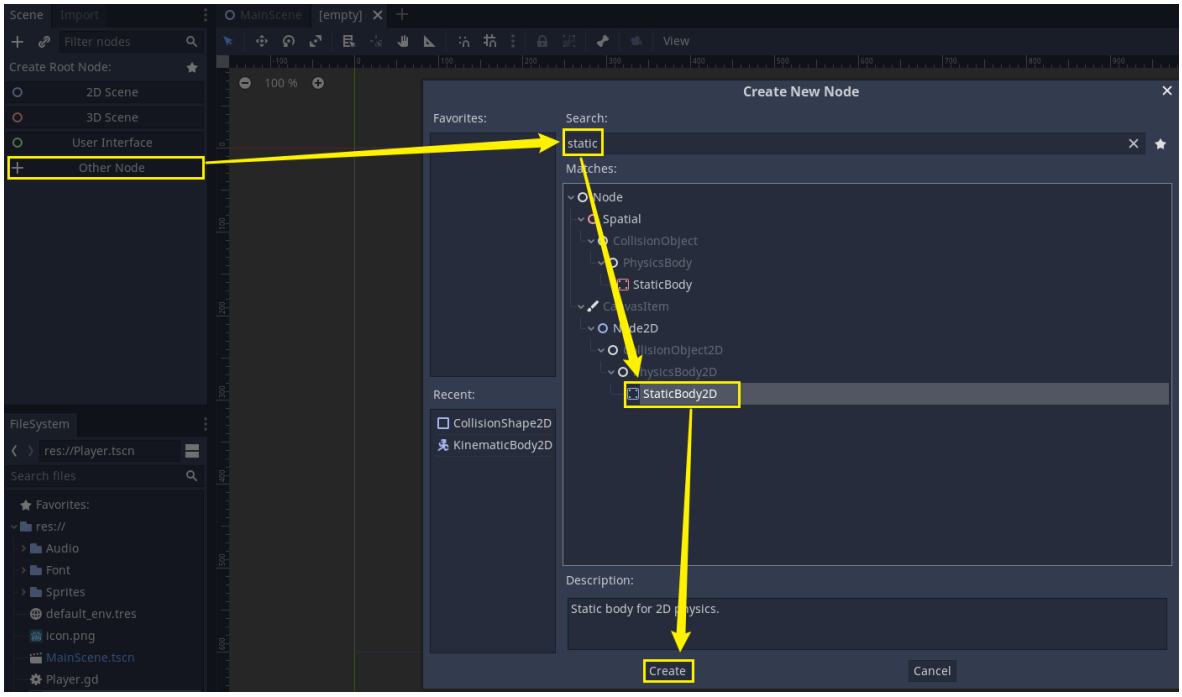
In this lesson, we are going to create tiles.

Create a new scene by going to Scene > New Scene



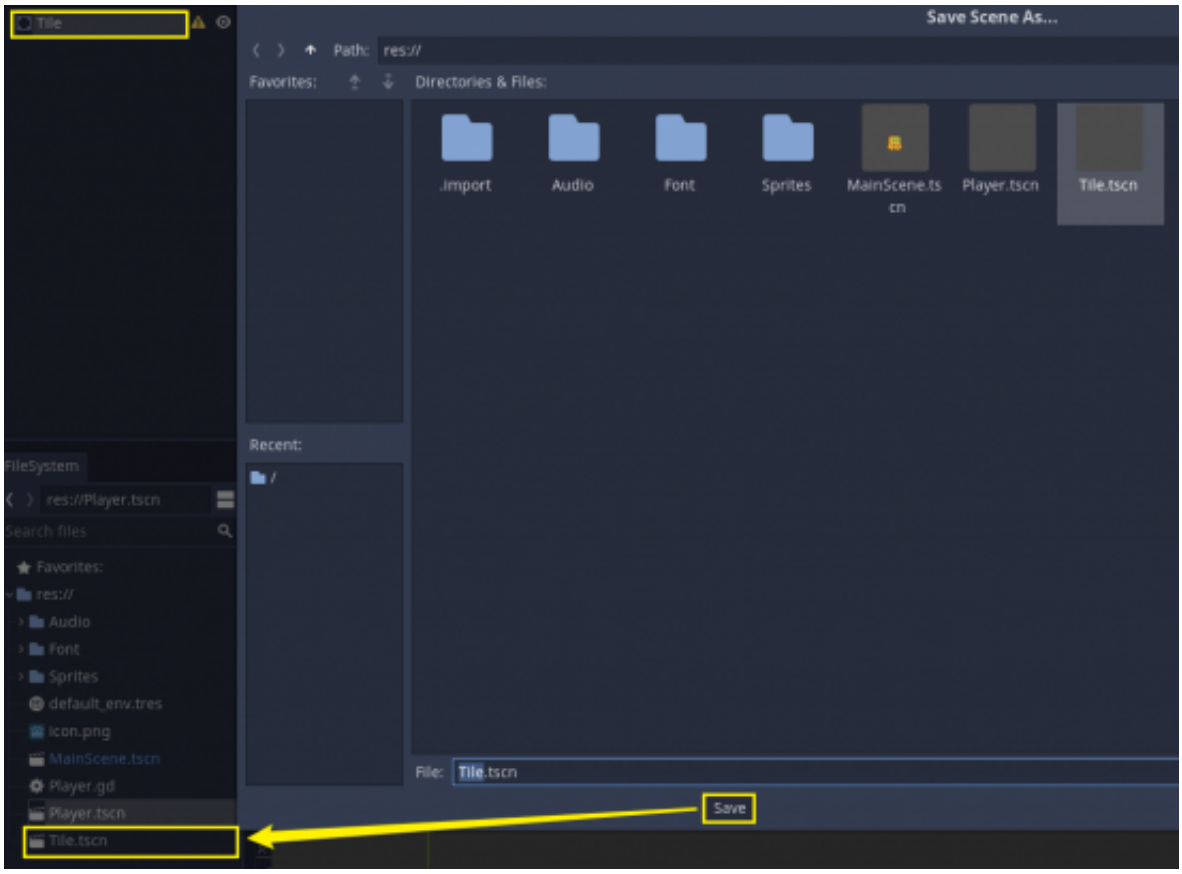
Later we will put multiple instances of the tile scene in our main scene.

Go to Create Root Node: **Other node** and choose **StaticBody2D**.

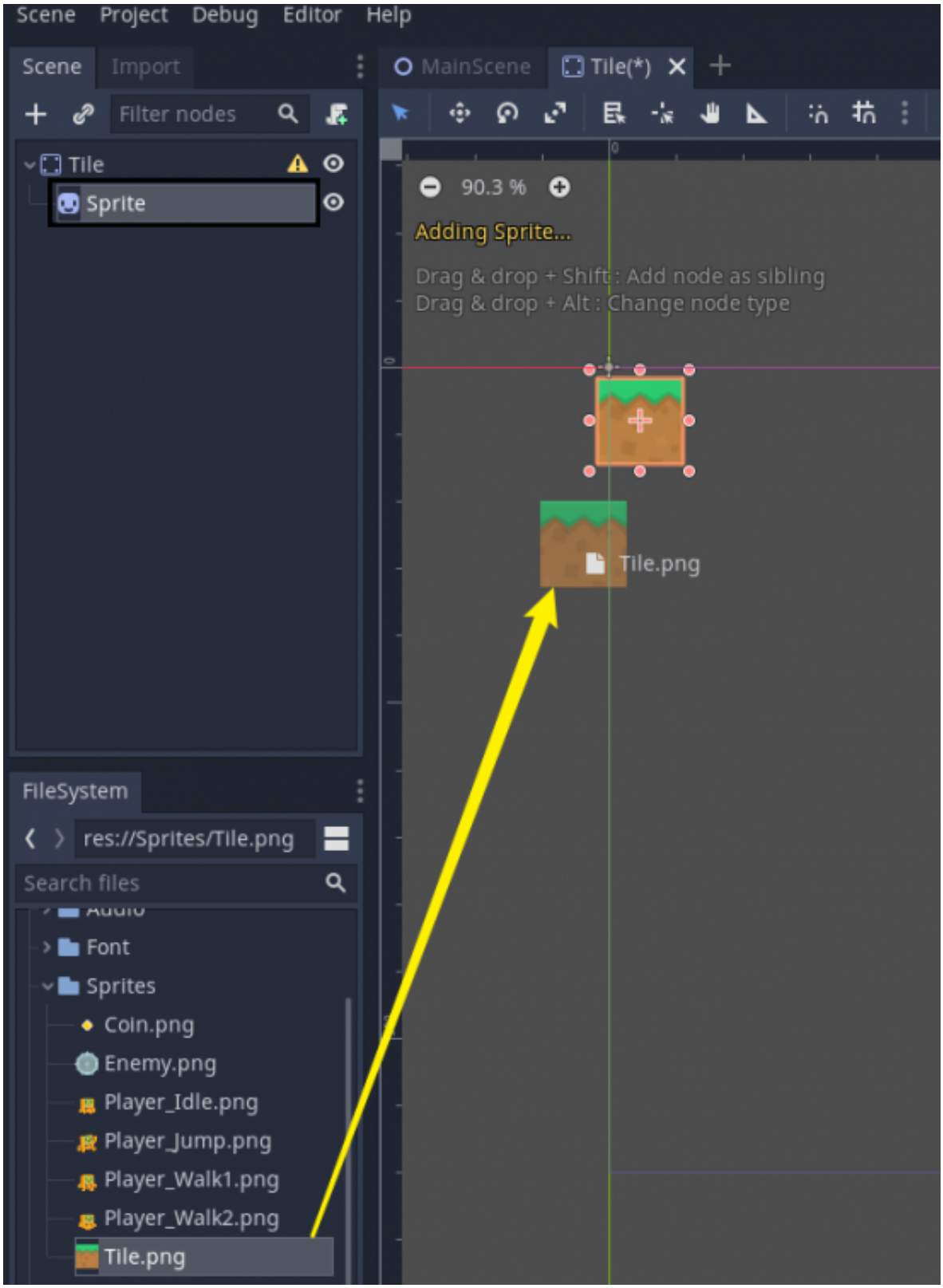


StaticBody2D is a body that is not intended to move. It is ideal for implementing static objects such as walls or platforms.

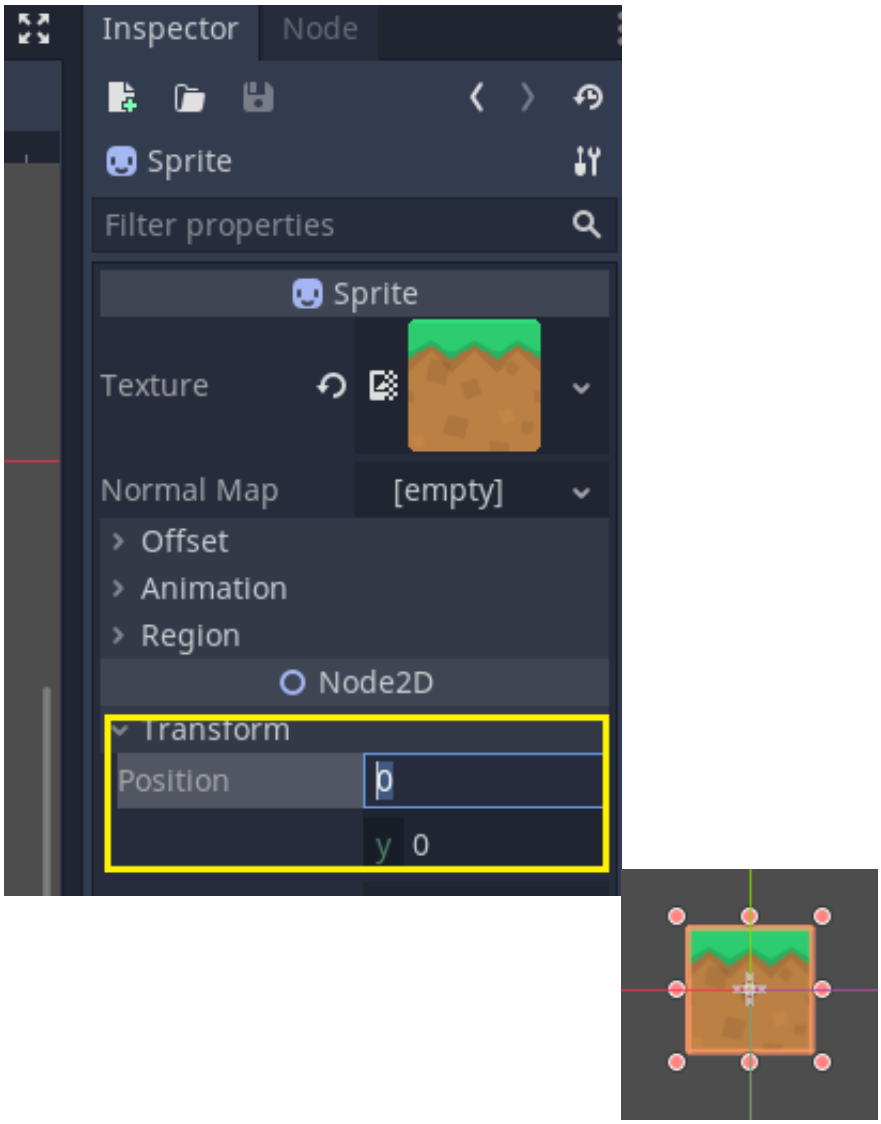
Rename it to ‘Tile’ and save it in our main project folder.



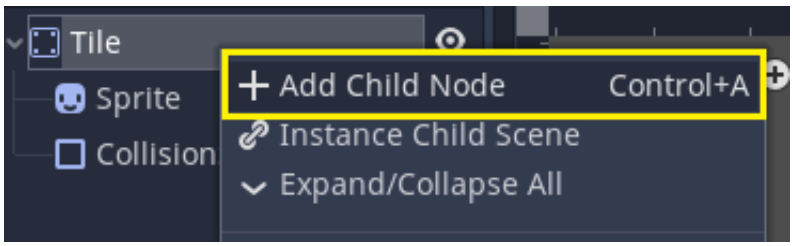
Drag in the tile sprite from FileSystem, and rename the node to ‘Sprite’.



Set the transform position to be 0 on both the x and y.

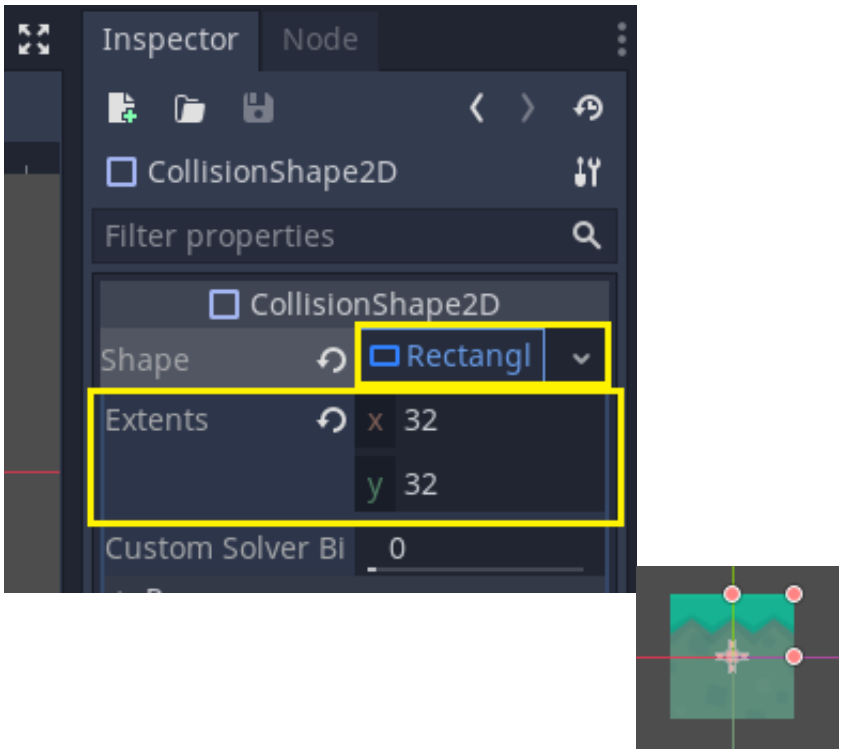


Right-click on the root note ‘Tile’ and add a new child node (Ctrl+A). Add a **CollisionShape2D**.



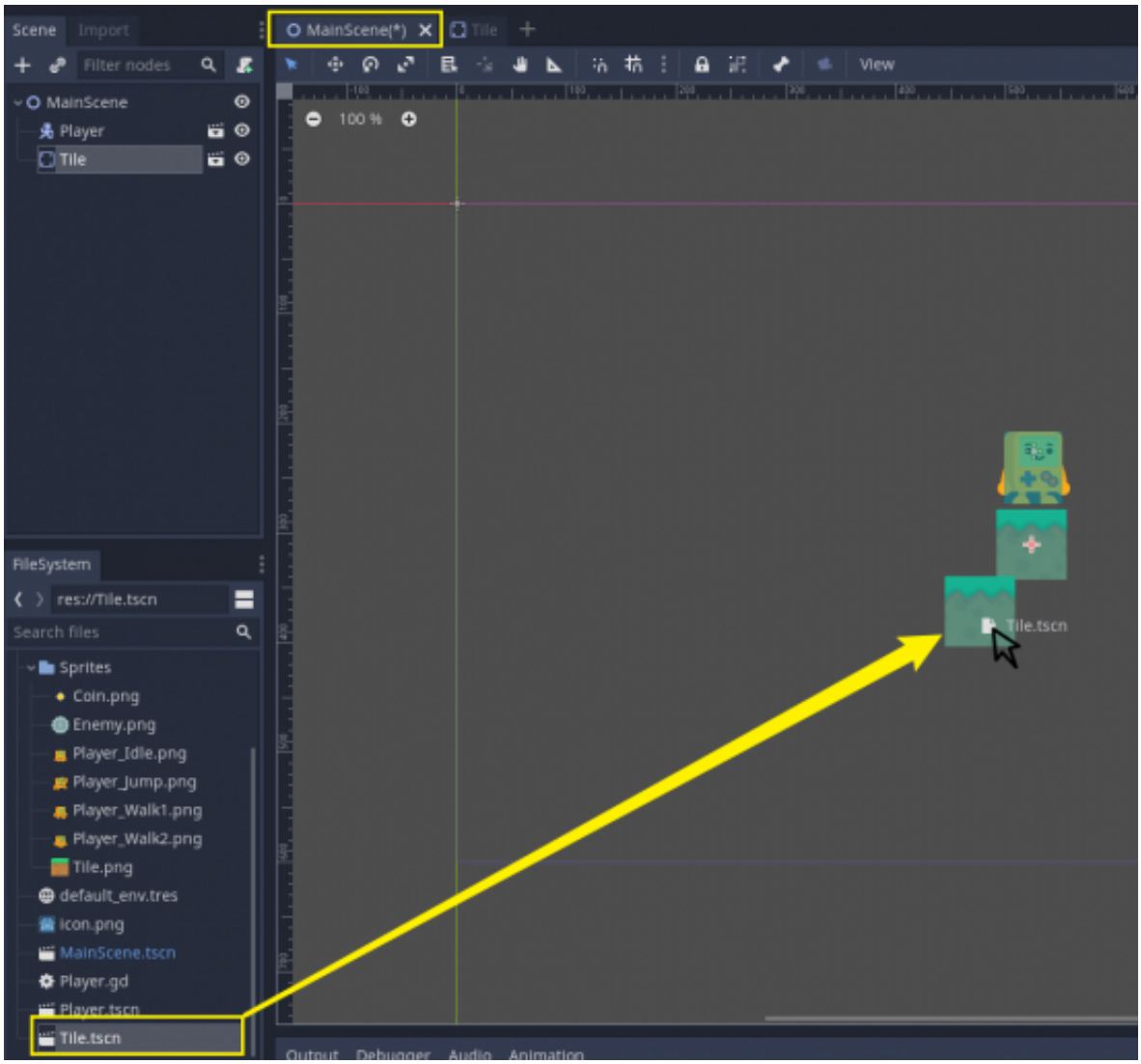
With the CollisionShape2D selected, choose Shape > **RectangleShape2D**.

Expand the shape tab and set the **Extents** property to 32 for both x and y.



Save the scene and open up the main scene.

Drag and drop the **Tile.tscn** into the scene.

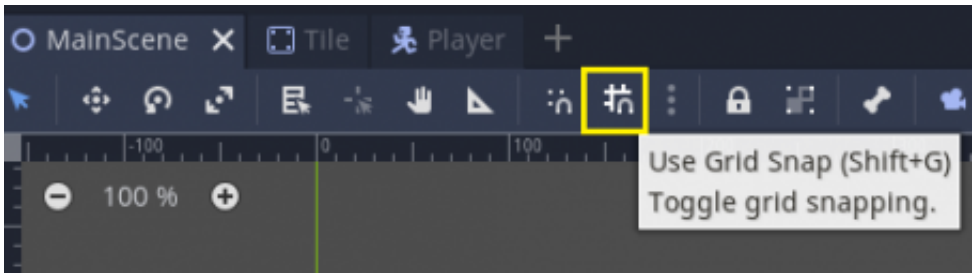


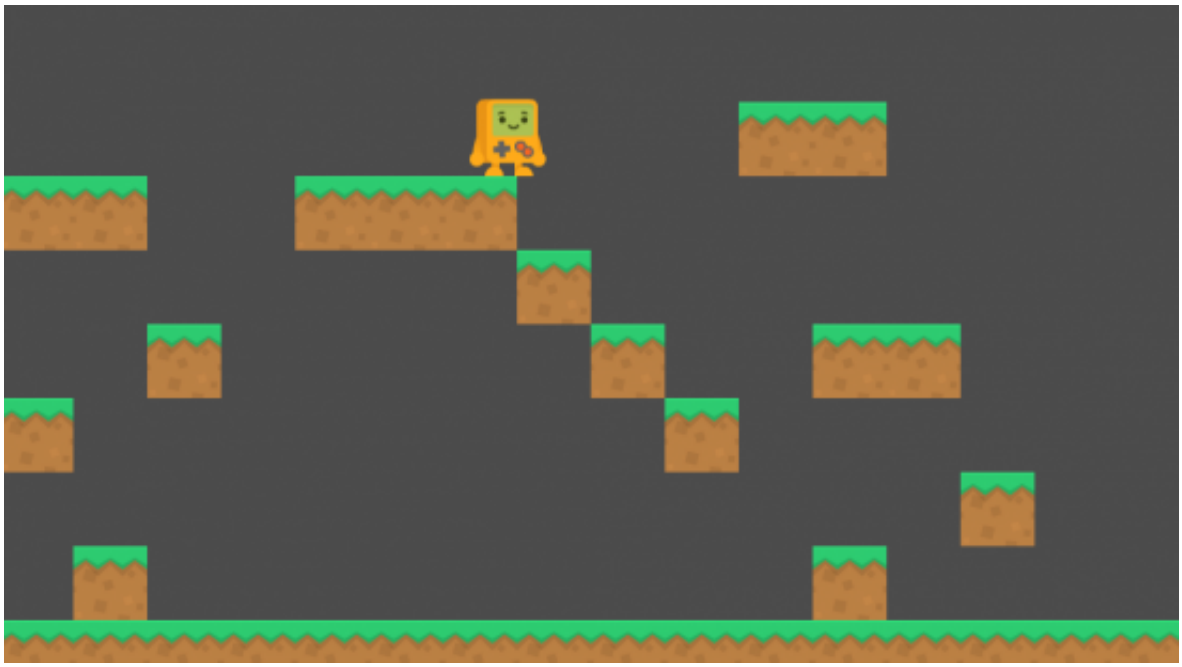
Save the scene and press Play.

The player falls down onto the tile now.

Now we can start building a level.

Simply press **Ctrl+D** to duplicate the tile. For more precision, you can toggle **Grid Snapping** on the toolbar (**Shift+G**)

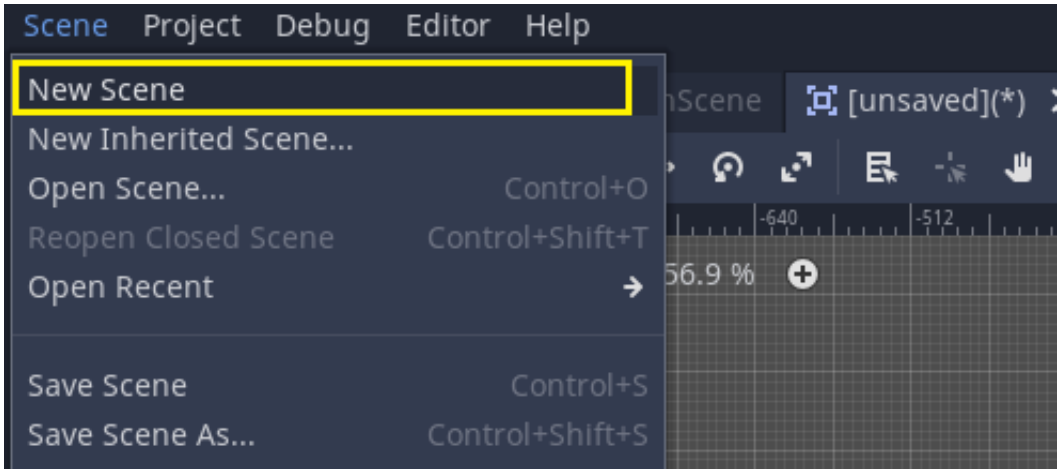




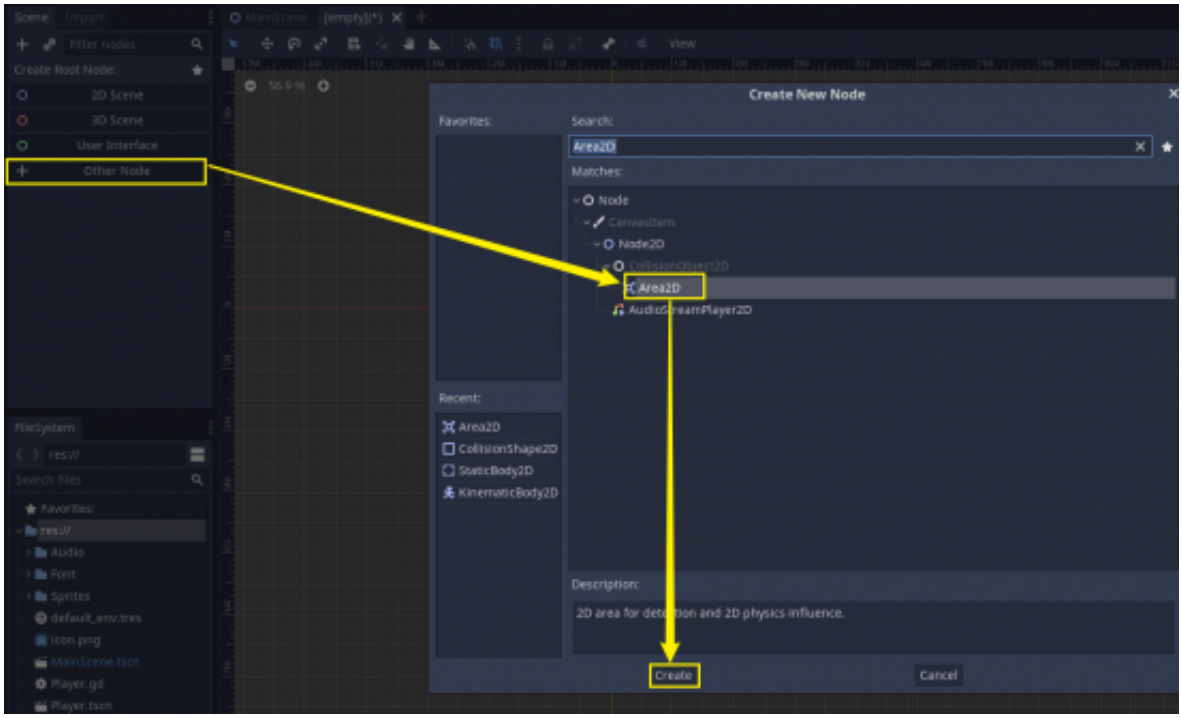
Enemy Part 1

In this lesson, we are going to create our enemy, which can hit the player and restart the scene.

Let's start by creating a new scene.

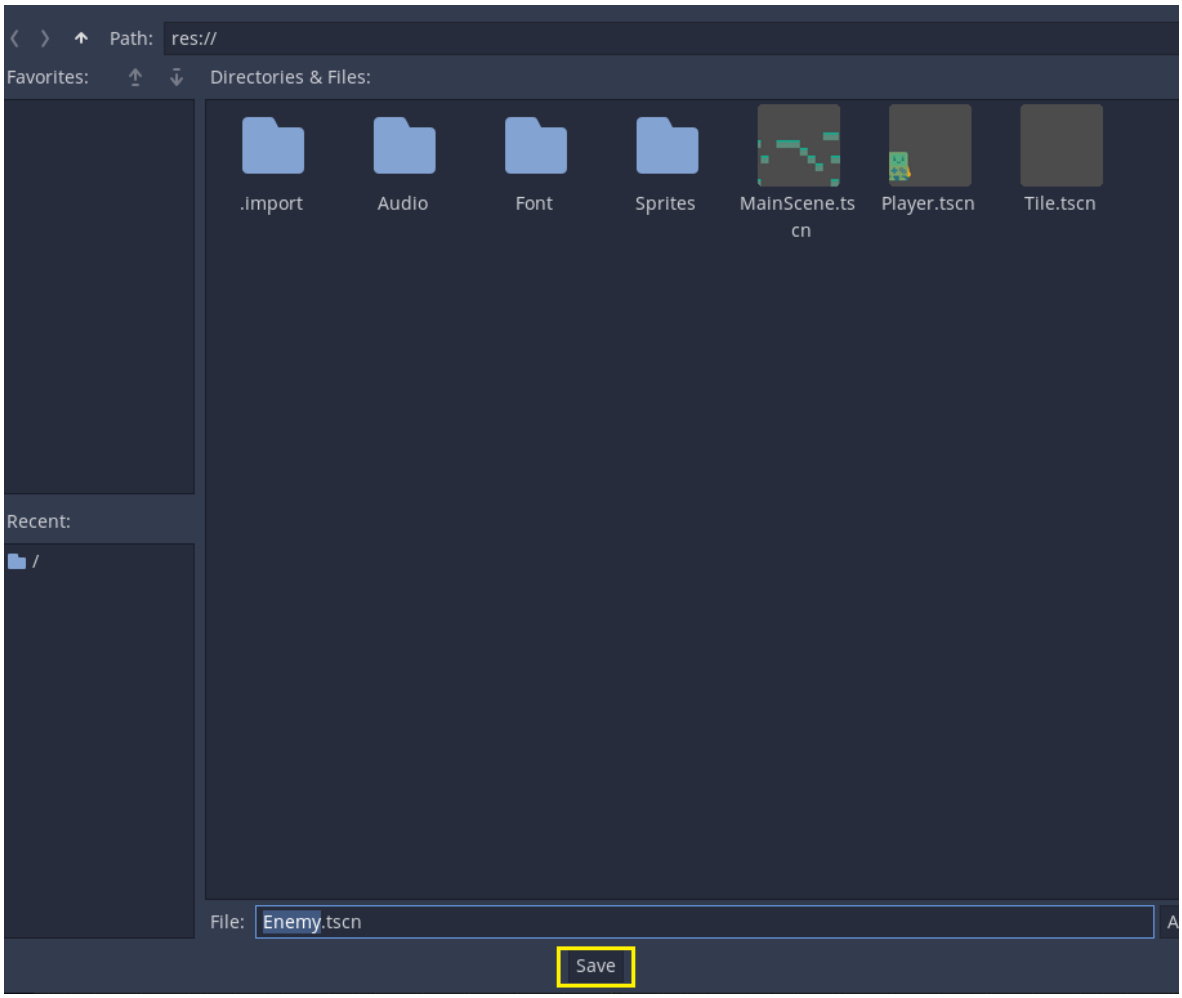


Create a new root node: Other Node > **Area2D**

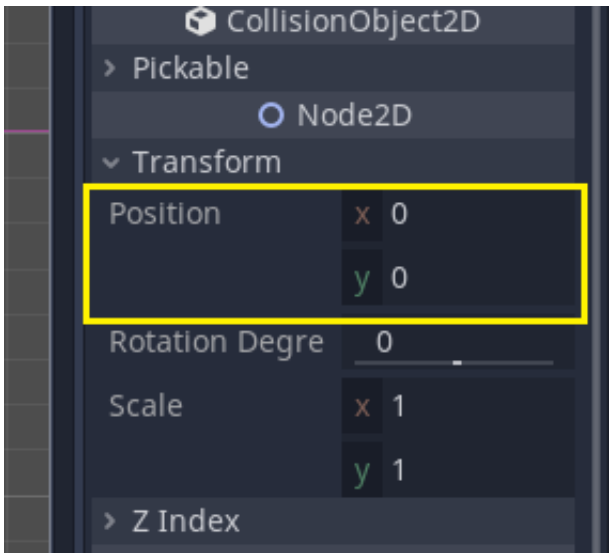


Area2D is an area that detects CollisionObject2D nodes entering, overlapping, or exiting. It is useful for detecting collision and applying physics influence.

Rename the node from **Area2D** to **Enemy** and save the scene (Ctrl+S).

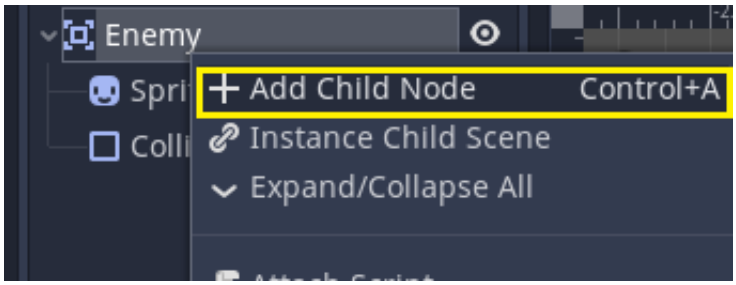


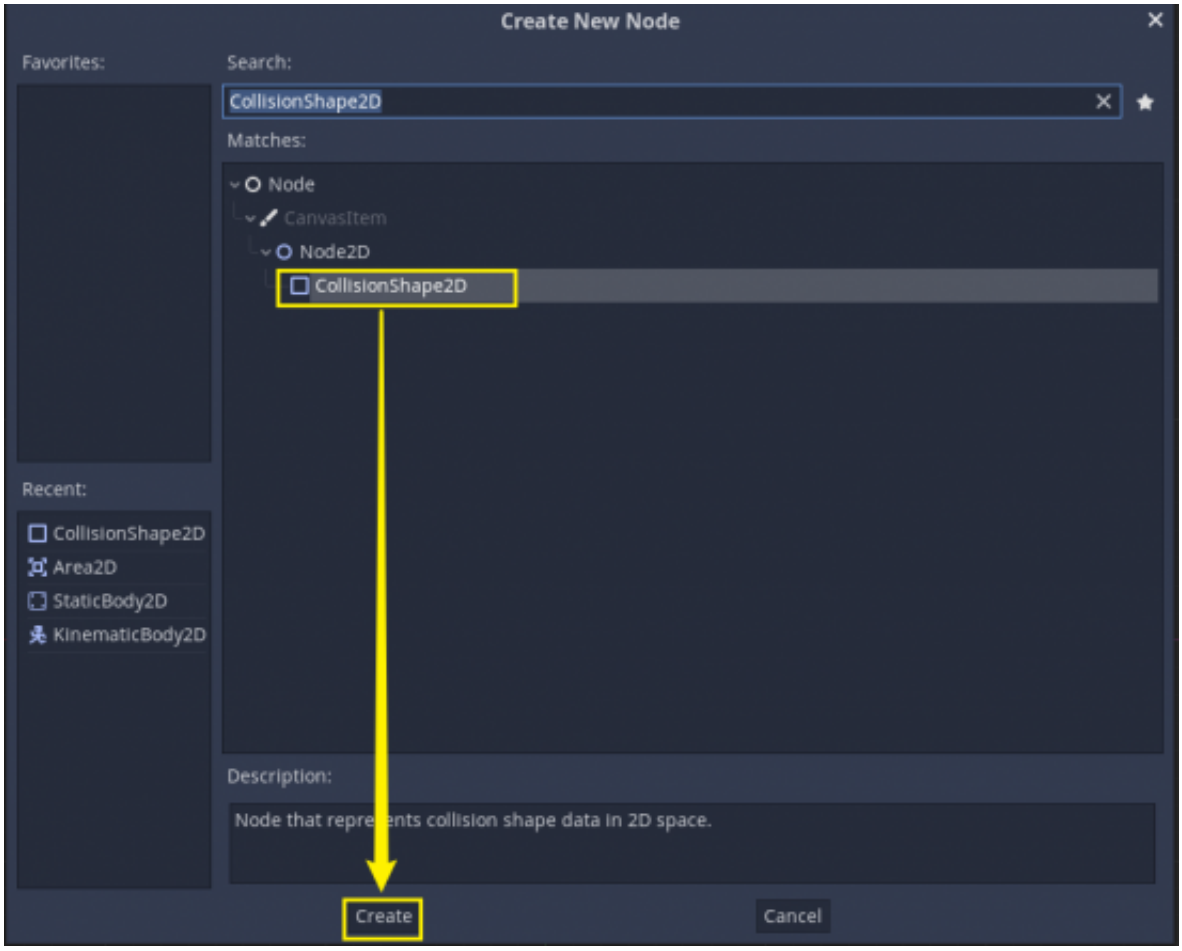
Drag in the Enemy sprite (FileSystem > Sprites > Enemy.png) into the viewport. Make sure that the position is set to (0,0).



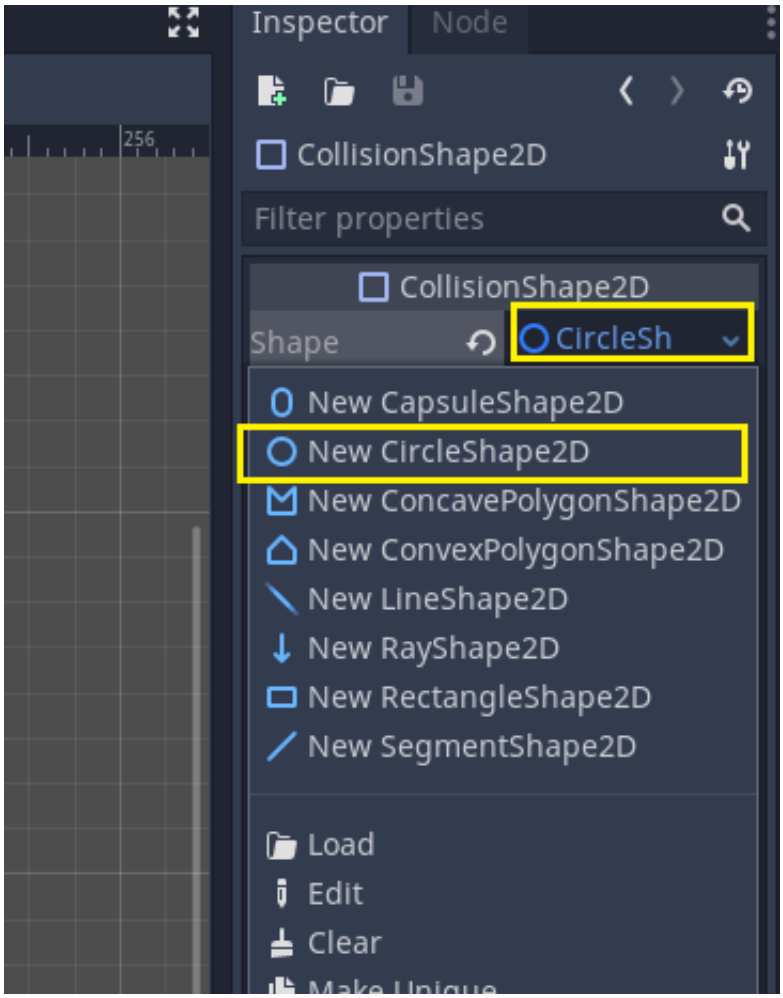
Rename the sprite node to **Sprite**.

Add a new child node (Ctrl+A) : CollisionShape 2D.

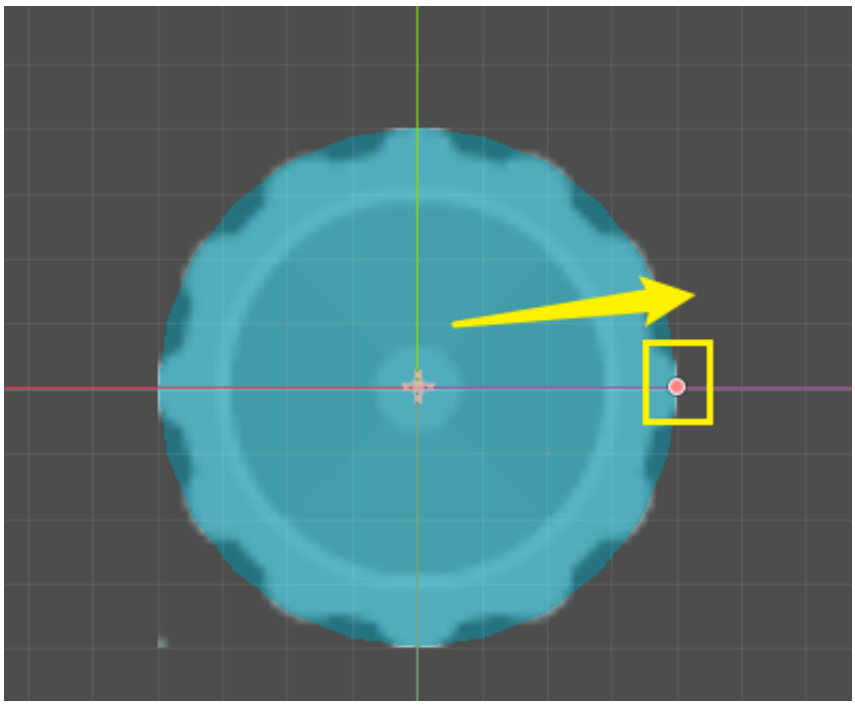




In the Inspector, we can set the shape of the collision node to be a **New CircleShape2D**.

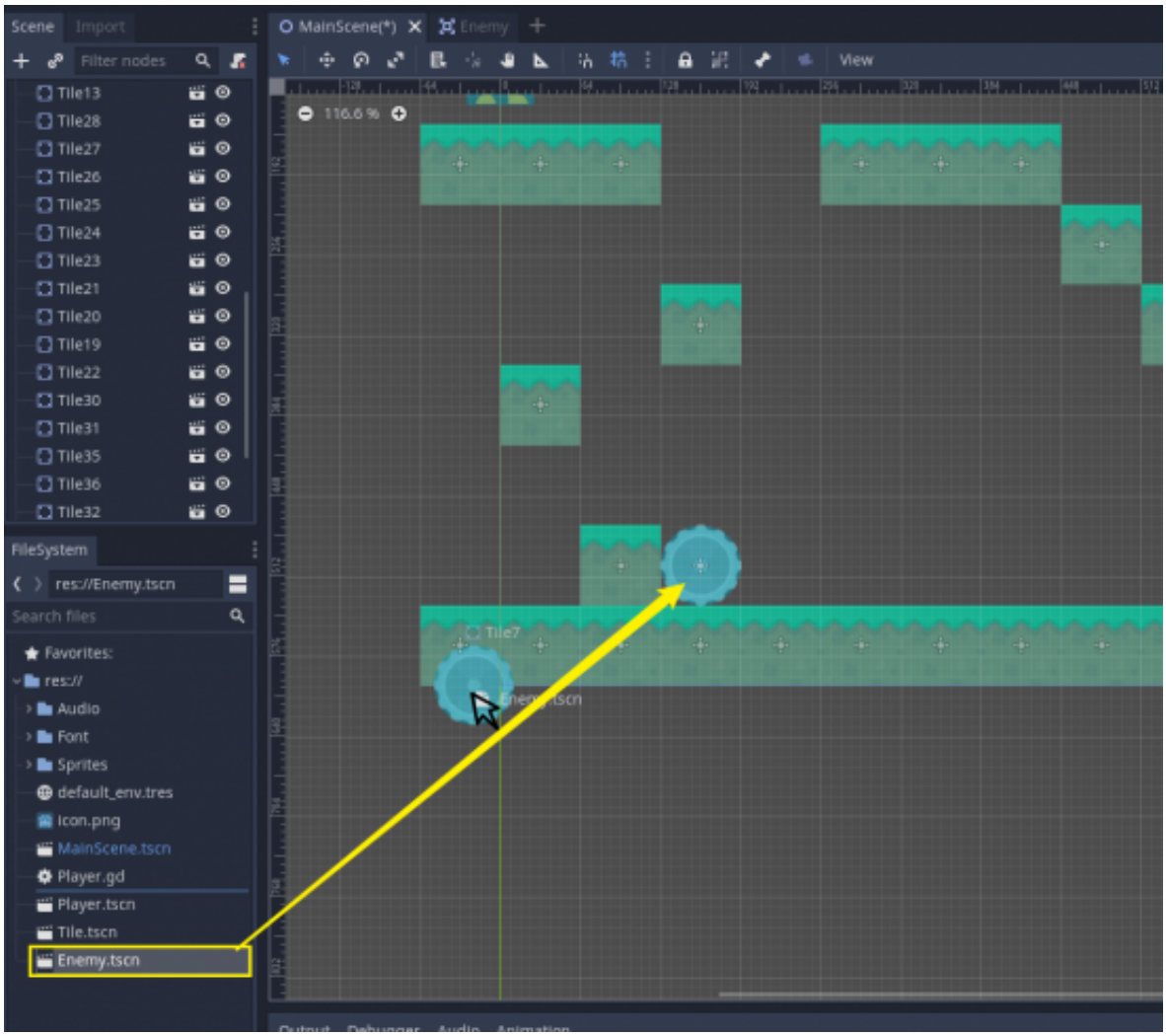


We can then click and drag the orange dot to resize the circle.

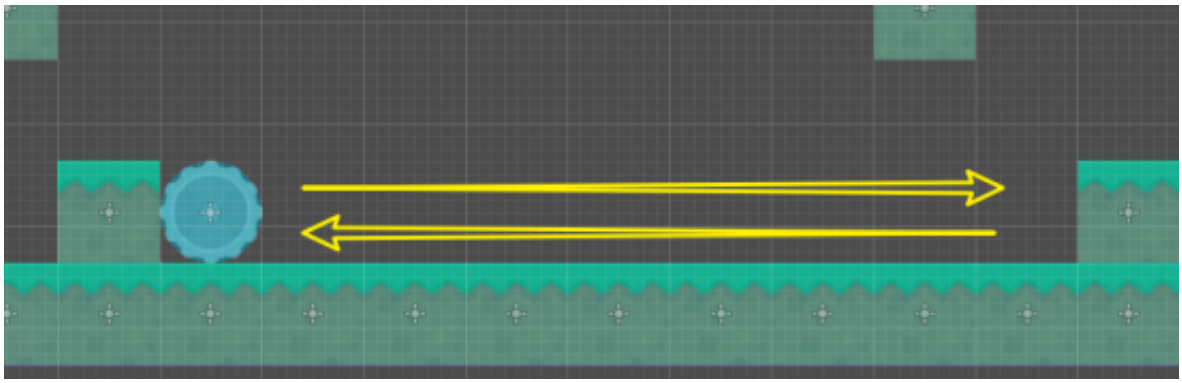


Save the scene and hop over to our main scene. We can now drag in the **Enemy.tscn** into the

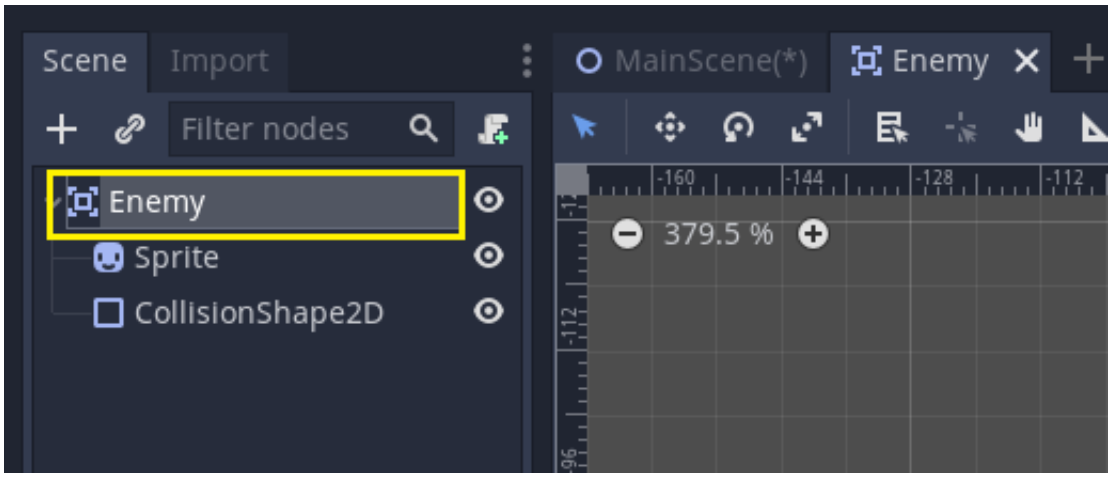
scene.



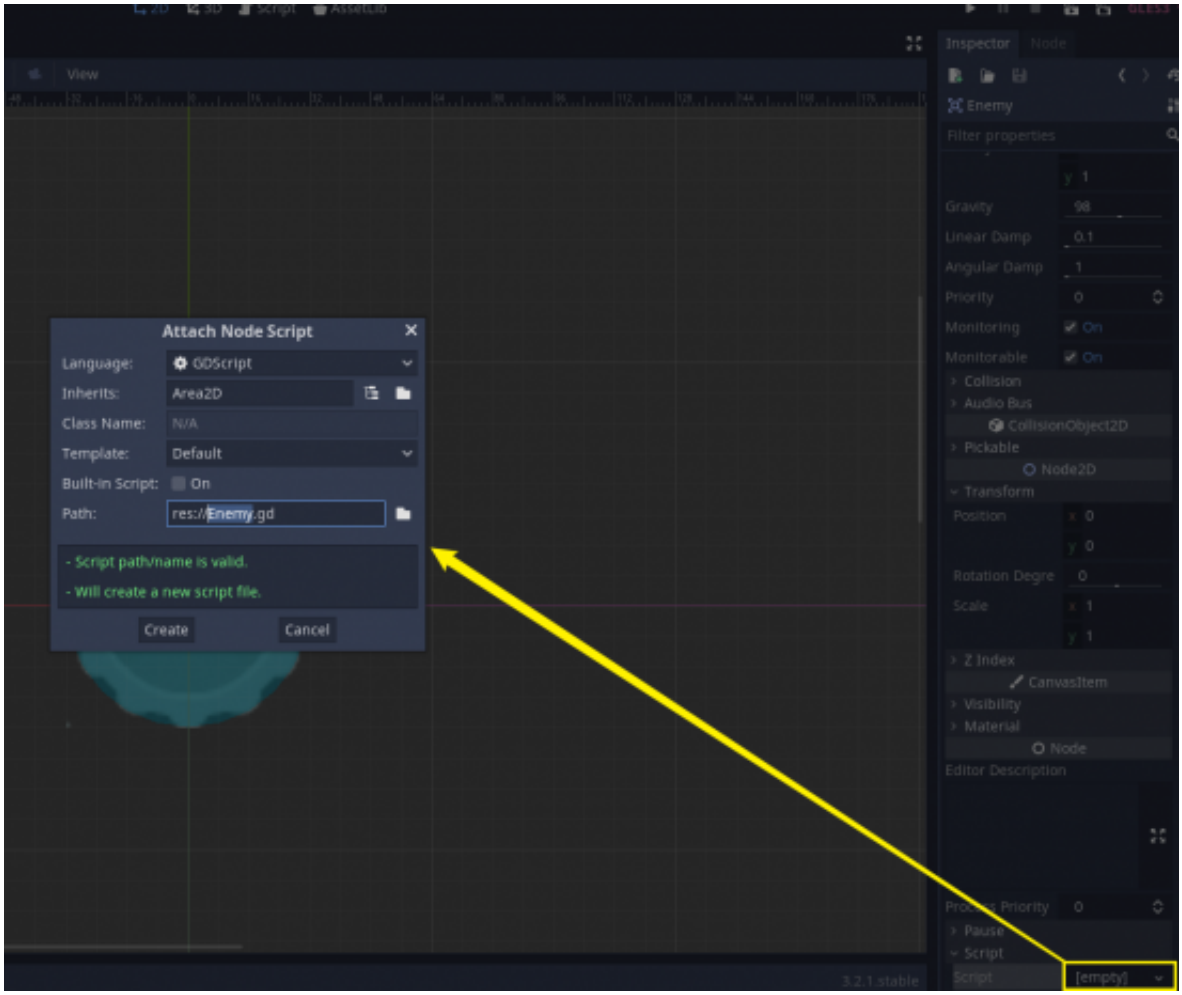
Now, we are going to create a script on this enemy so that it can move back and forth between two points.



Let’s go back to the Enemy scene, and select the root node.



In the inspector, click on **New Script** under the Script property.



Hit ‘Create’, and the script editor will pop up automatically.

The first thing we will create is integer variables that will store the **speed** value and **move distance**. We will make their default values to be 100.

```
extends Area2D
```



```
var speed : int = 100
var moveDist : int = 100
```

To move from one position to another, we need to know our starting position and the target position. Our starting position will be stored in a var of type int called: **startX**

```
extends Area2D

var speed : int = 100
var moveDist : int = 100

onready var startX : int = position.x
```

Note that the **onready** keyword is used here to make sure that the value stores the position information before the variable is initialized.

Then we need to set the target position, which is simply **position.x** + **moveDist**.

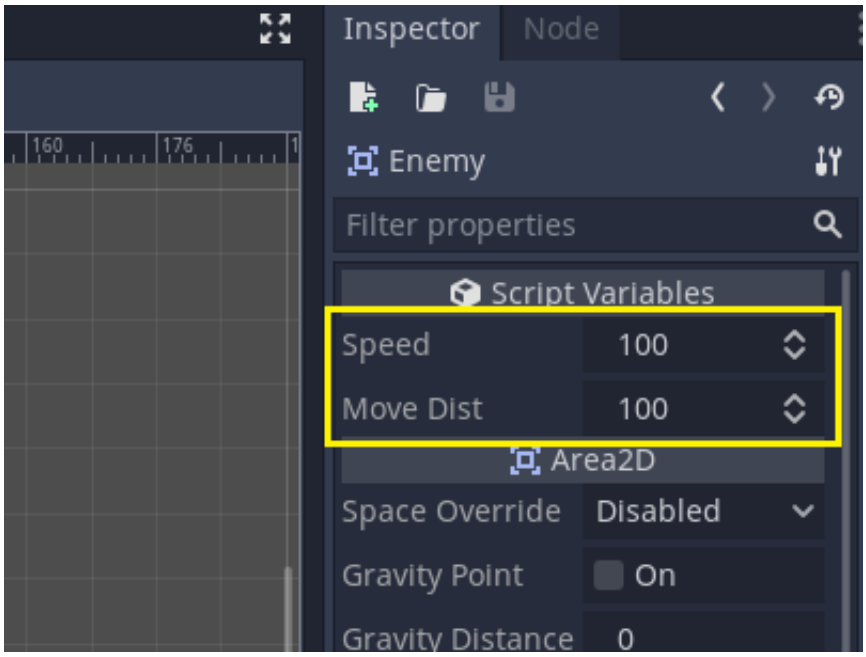
```
extends Area2D

var speed : int = 100
var moveDist : int = 100

onready var startX : int = position.x
onready var targetX : int = position.x + moveDist
```

The problem we have with this script right now is, when we make a modification to the values, it is going to affect **every Enemy instance** in the scene.

We want to vary the speed and the distance that each enemy moves at. To do so, we can use an **export** keyword at the start of the variables.

```
extends Area2D

export var speed : int = 100
export var moveDist : int = 100

onready var startX : int = position.x
onready var targetX : int = position.x + moveDist
```

The **export** keyword allows the variables to appear in the inspector.



Now, these values will only change on **that instance** of the enemy. This means we can have two enemies that have different speed/move distance.

We will create a function called **move_to**, which will move the enemy to the target position. The function needs to take in 3 parameters: the current position, the destination, and how much we move in this function call.

```
#The function takes in three parameters: current position, destination, and how much
it moves in this function call.
func move_to (current, to, step):
```

We will then create a local variable within this function called **new**. This will help us store and modify the current position.

```
#The function takes in three parameters: current position, destination, and how much
it moves in this function call.
func move_to (current, to, step):

 #We will use the new variable to store and modify the current position.
 var new = current
```

If **new** is less than **to**, it means we haven't reached the destination yet (Given that we are moving positively).

```
#The function takes in three parameters: current position, destination, and how much
it moves in this function call.
func move_to (current, to, step):

 #We will use the new variable to store and modify the current position.
 var new = current
```



```
 #If we haven't reached the destination yet
 if new < to:
```

So if **new < to**, we need to move it from its current position. We can simply add **step** to **new** to increment it each frame.

```
#The function takes in three parameters: current position, destination, and how much it moves in this function call.
func move_to (current, to, step):

 #We will use the new variable to store and modify the current position.
 var new = current

 #If we haven't reached the destination yet
 if new < to:
  #Move positively by step
  new += step
```

If we have gone over the destination (i.e. if **new > to**), we need to stop moving further by going: **new = to**

```
#The function takes in three parameters: current position, destination, and how much it moves in this function call.
func move_to (current, to, step):

 #We will use the new variable to store and modify the current position.
 var new = current

 #If we haven't reached the destination yet
 if new < to:
  #Move positively by step
  new += step
  #If we've gone over the destination
  if new > to:
   #Cap it out at the destination
   new = to
```

Once we have reached the target position, we need to move in the opposite direction, and stop once we return to the starting position.

```
#The function takes in three parameters: current position, destination, and how much it moves in this function call.
func move_to (current, to, step):

 #We will use the new variable to store and modify the current position.
 var new = current

 #If we haven't reached the destination yet
 if new < to:
```



```
  #Move positively by step
  new += step
  #If we've gone over the destination
  if new > to:
   #Cap it out at the destination
   new = to
 #If we've reached the destination
 else:
  #Move negatively by step
  new -= step
  #If we've returned to our starting position
  if new < to:
   #Cap it out at the starting position
   new = to
 return new
```

And finally, outside of this if statement, we can return **new**.

In the next lesson, we're going to continue on with this script by implementing the ability to detect the collision with the player.

In this lesson, we are going to continue on with the **Enemy** script and implement the ability to detect the collision with the player.

```
extends Area2D

export var speed : int = 100
export var moveDist : int = 100

onready var startX : int = position.x
onready var targetX : int = position.x + moveDist

#gets called every frame
func _process (delta):

 position.x = move_to(position.x, targetX, speed * delta)

 #is our position equal to the target?
 if position.x == targetX:
  if targetX == startX:
   targetX = position.x + moveDist
  else:
   targetX = startX

#moves 'current' towards 'to' at a rate of 'step'
func move_to (current, to, step):

 var new = current

 #are we moving positive?
 if new < to:

  new += step
  if new > to:
   new = to
 #are we moving negative?
 else:
  new -= step
  if new < to:
   new = to
 return new
```

First, we will create a function called **_process(delta)**, which is a built-in function in Godot that gets called every frame. The **delta** will become the duration between each frame.

We can call the **move_to** function that we created in the previous lesson here and assign it to our position.x to actually affect the enemy instance. Note that **move_to** takes in the three parameters: **current**, **to**, **step**.

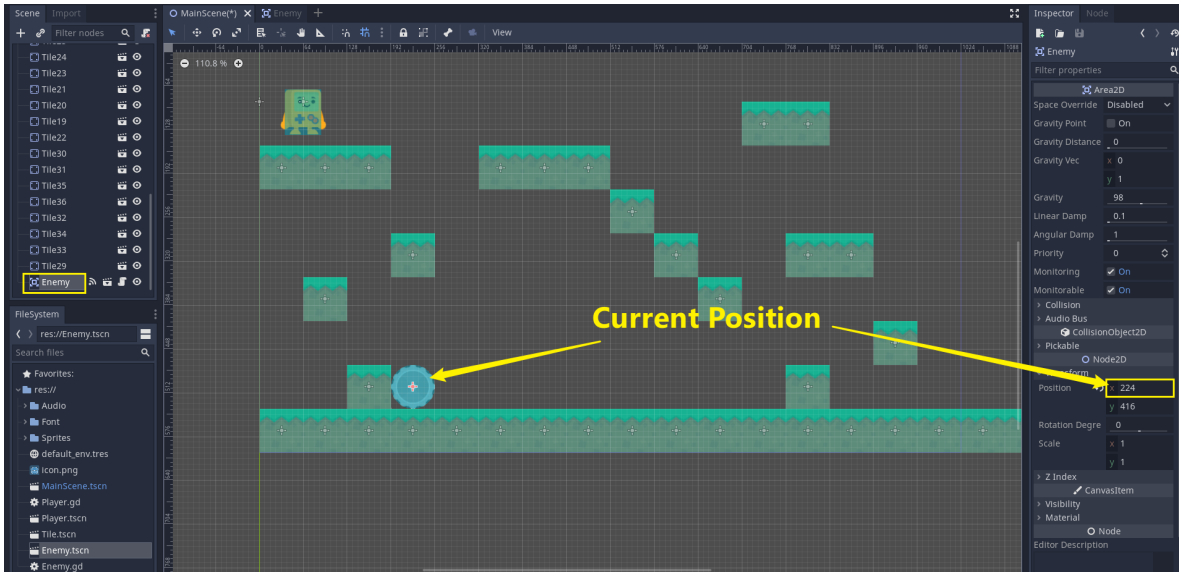
```
func _process (delta):

        #Passing position.x to 'current' / targetX to 'to' / and speed*delta to 'step
'.
 position.x = move_to(position.x, targetX, speed * delta)
```

Within the **_process(delta)**, we can also check if our position is equal to targetX. We can then adjust our targetX, depending on our current position.

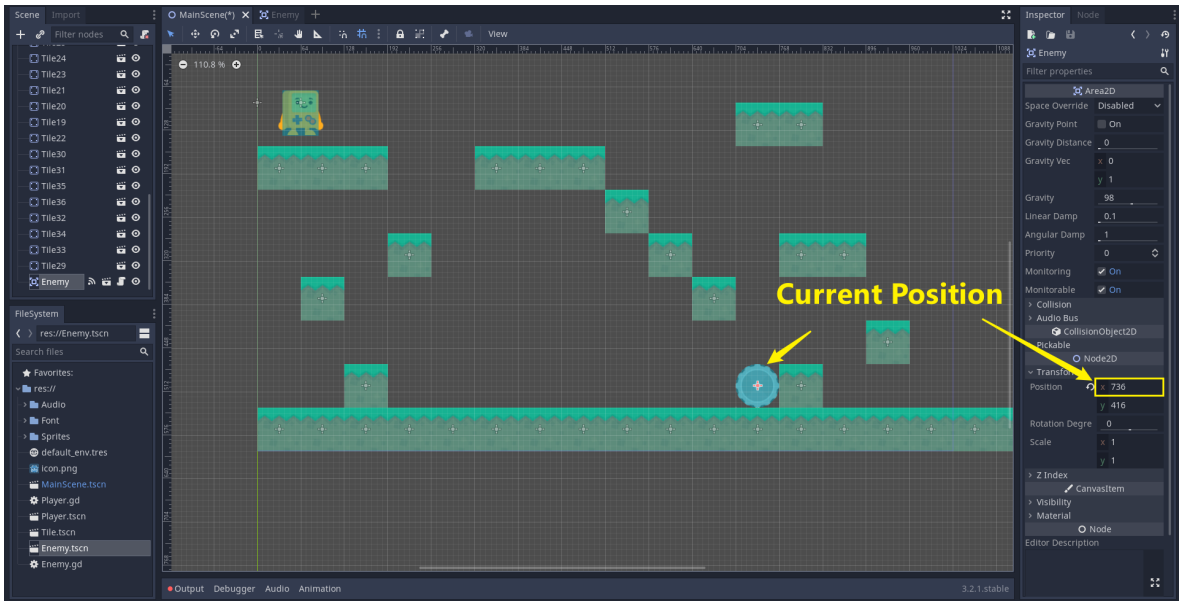
```
func _process (delta):

position.x = move_to(position.x, targetX, speed * delta)

#is our position equal to the target?
if position.x == targetX:
 if targetX == startX:
  targetX = position.x + moveDist
 else:
  targetX = startX
```

Now, let's go back into our main scene and select the **Enemy** node.

Open up **Transform** in the **Inspector,** and check the current x position. This will be our starting position: 224

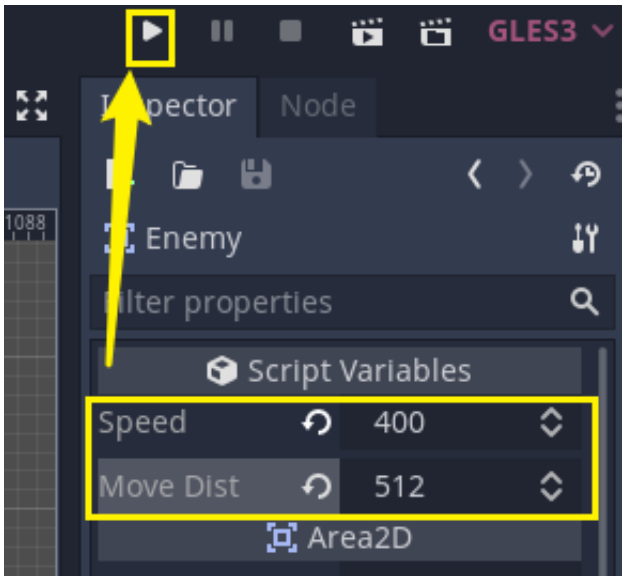


Click and drag the enemy all the way to the end. This will be our target x position: 736

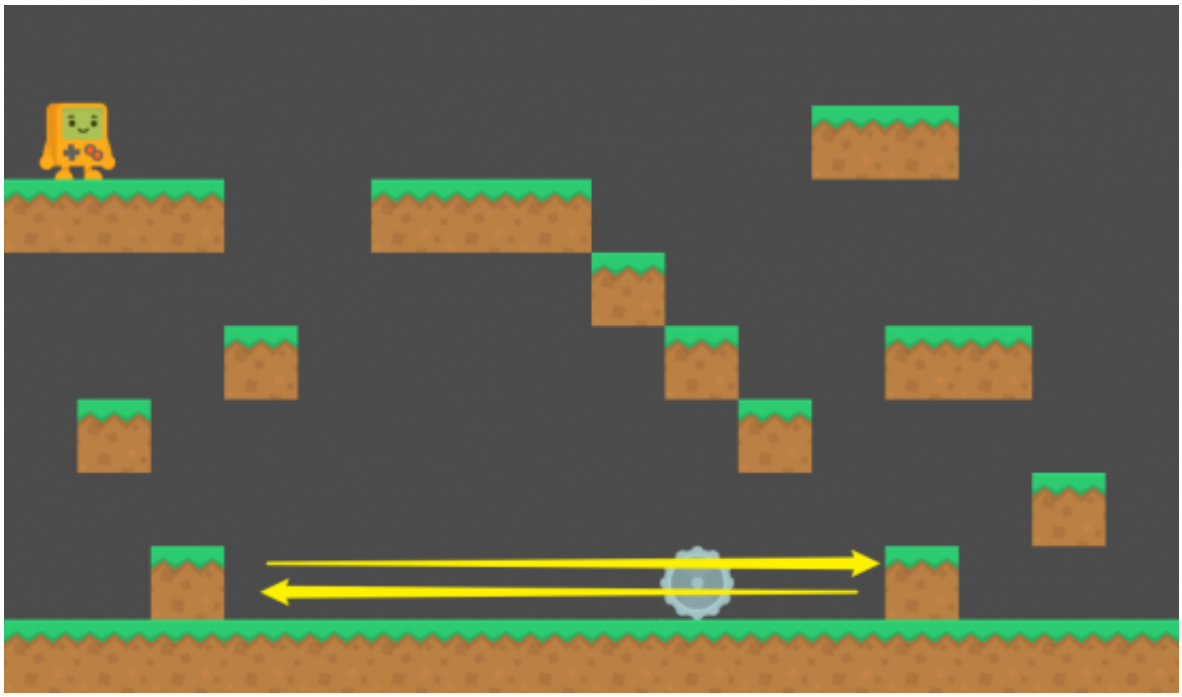


From here we can calculate that our move distance is going to be 736 – 224 = 512. Set the **moveDist** to 512 in the inspector, and the **speed** to 400 so it takes just over a second until you reach the end.

Click and drag the enemy instance to the starting position. Save and press Play.



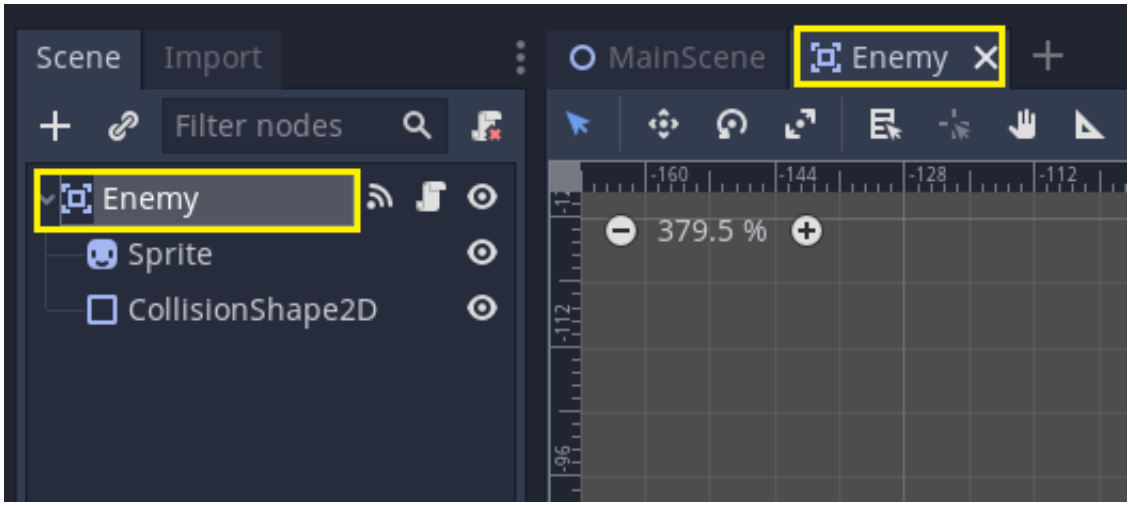
We can see now that the enemy is moving between two points.



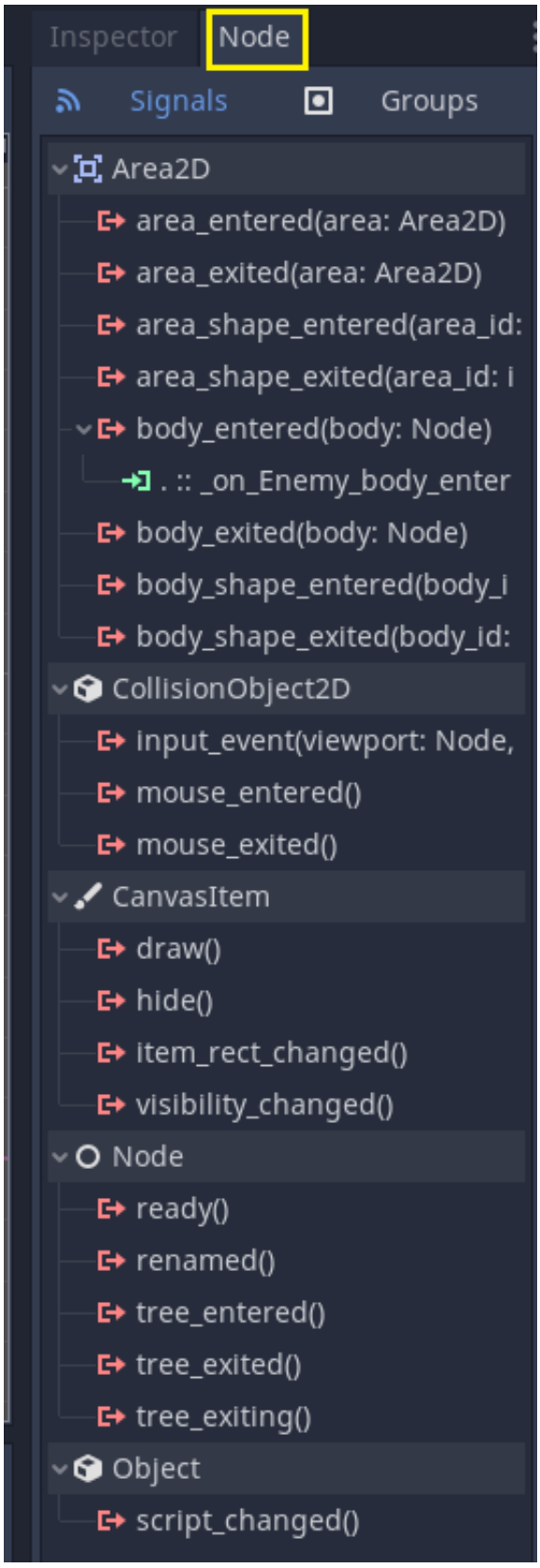
When we get hit by it, nothing happens at the moment. We are now going to make the scene restart by using '**signals**'.

Signals are events that can be called when a certain thing happens to a node. For example, rather than continuously checking a button to see if it's being pressed, the button can emit a signal when it's pressed.

Switch to the **Enemy** scene and select the root node.

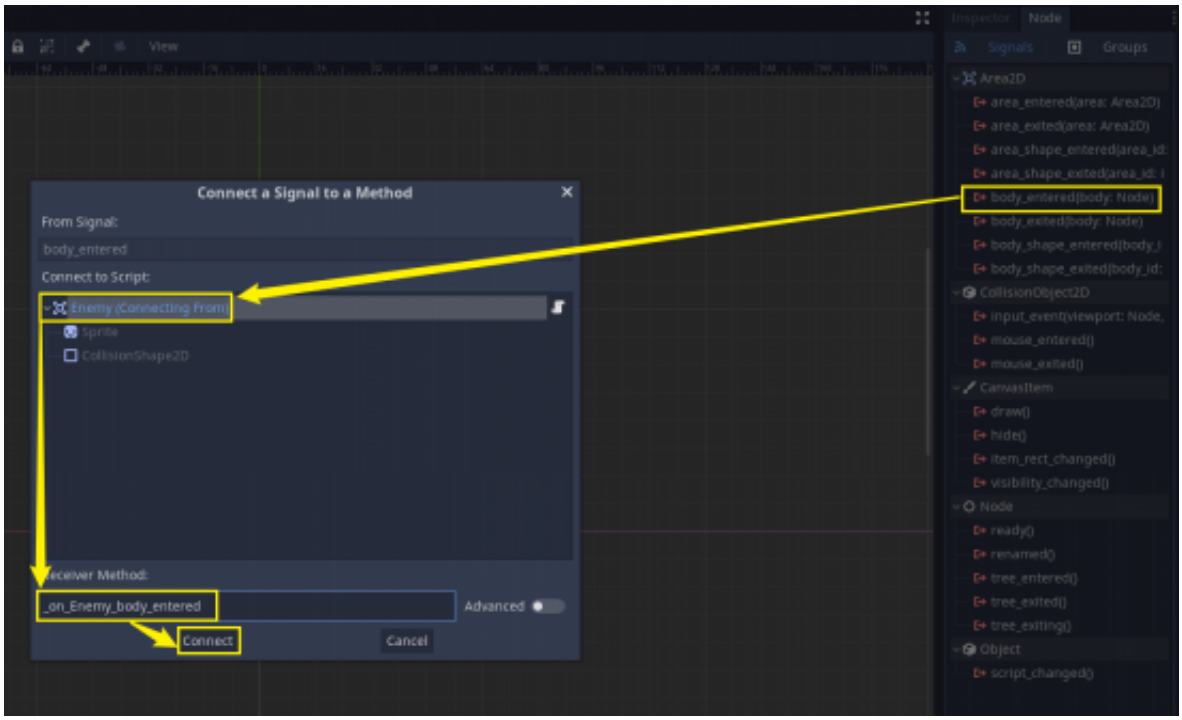


Select the **Node** tab next to the Inspector.



There is a list of various different signals that this node can emit. We want to detect when we have entered another body, and call a function that restarts the scene.

Double-click on '**body_entered**' under **Area2D** Node.



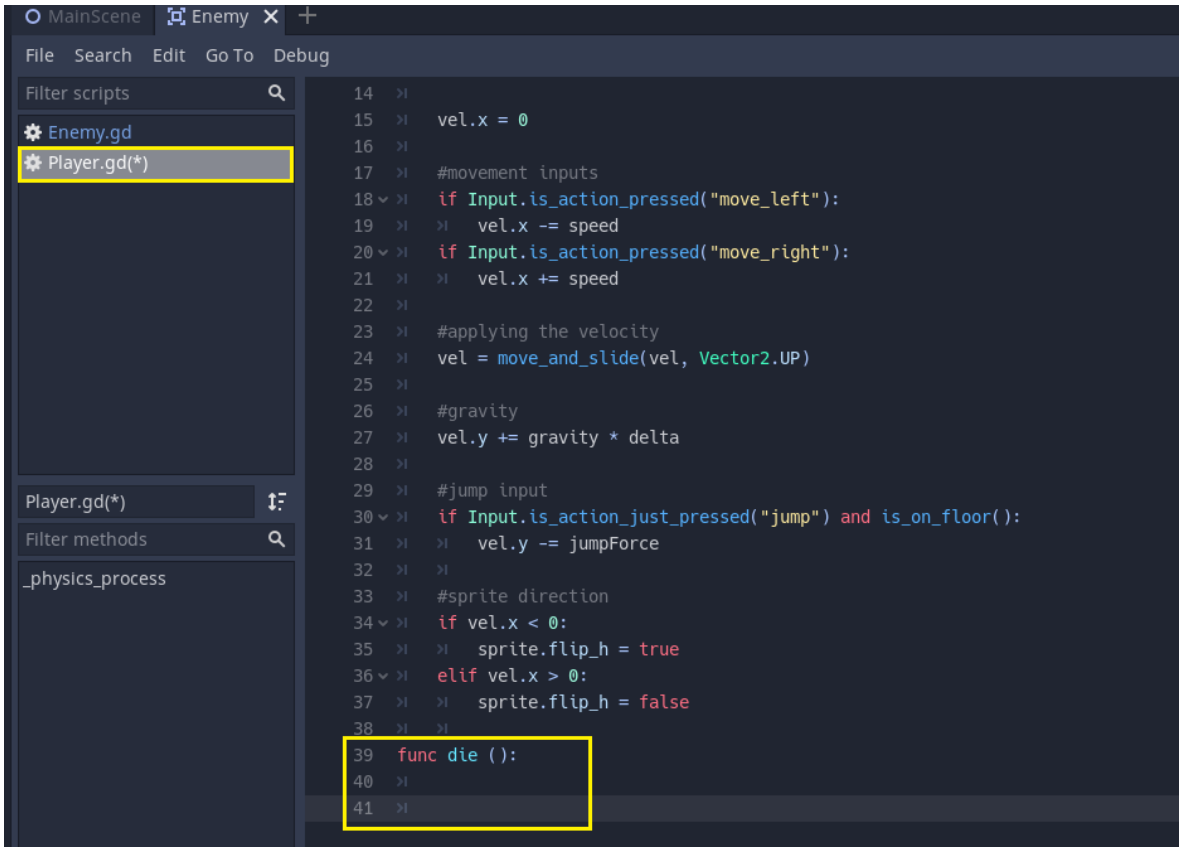
Make sure you have the enemy object selected. Under the **Receiver Method** tab, you can put down the name of the function to call when the signal gets called. Click the '**Connect**' button.

You'll see that it's created a brand new function called **_on_Enemy_body_entered** in the script.

```
func _on_Enemy_body_entered(body):
```

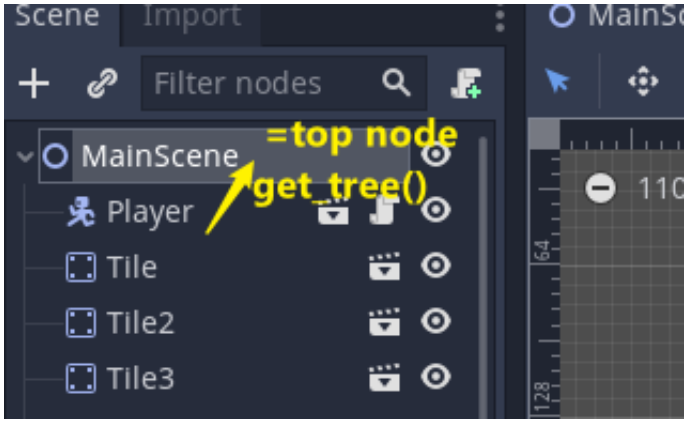
It carries over a parameter of the **body**, which will be the node that we have collided with.

Click on **Player.gd** and create a new function, which will get called when the enemy hits the player.



We're going to restart the scene when this function gets called. To do that, we can use **get_tree()**, which fetches the top parent node ('MainScene' in this case). Then, we can get it to reload by using the built-in function: **.reload_current_scene()**.

```
func die ():

 get_tree().reload_current_scene()
```



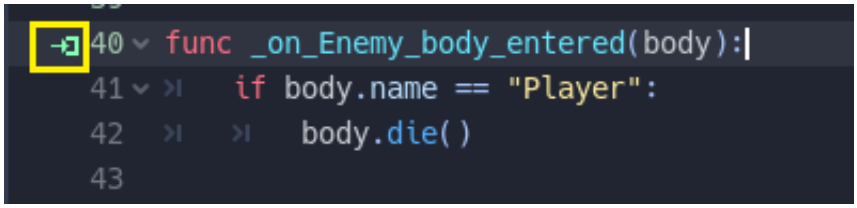
Now, inside of our **Enemy** script, we can call the function if the name of the body that has entered is "**Player**".

```
func _on_Enemy_body_entered(body):
 if body.name == "Player":
```

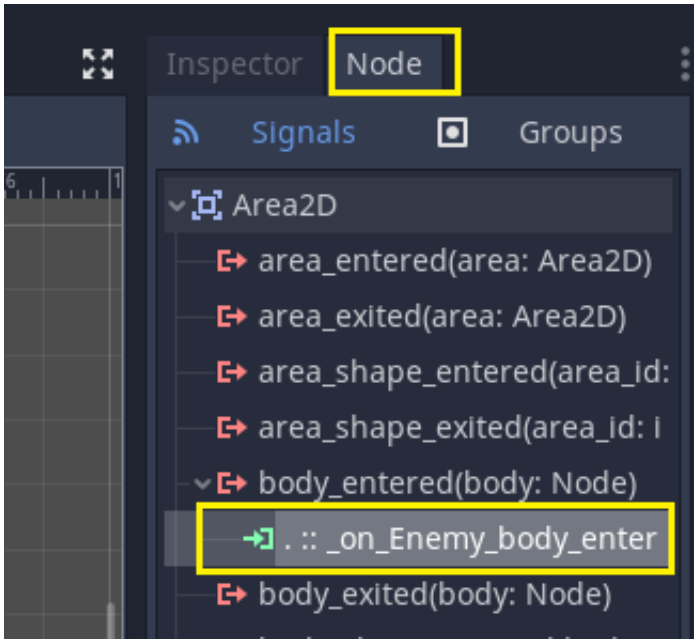


```
body.die()
```

To indicate that this is a signal, there's a green little icon on the left which notes that this here is the signal function.



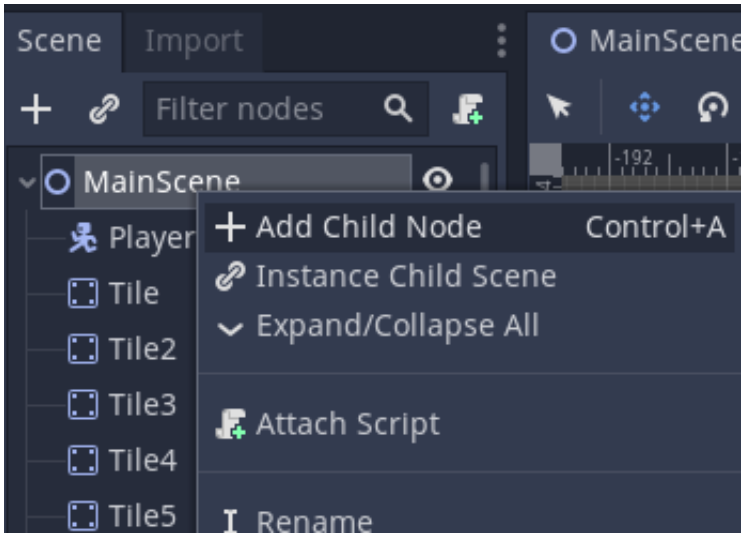
You'll see that our function appears under **body_entered** in the **Node** panel as well.

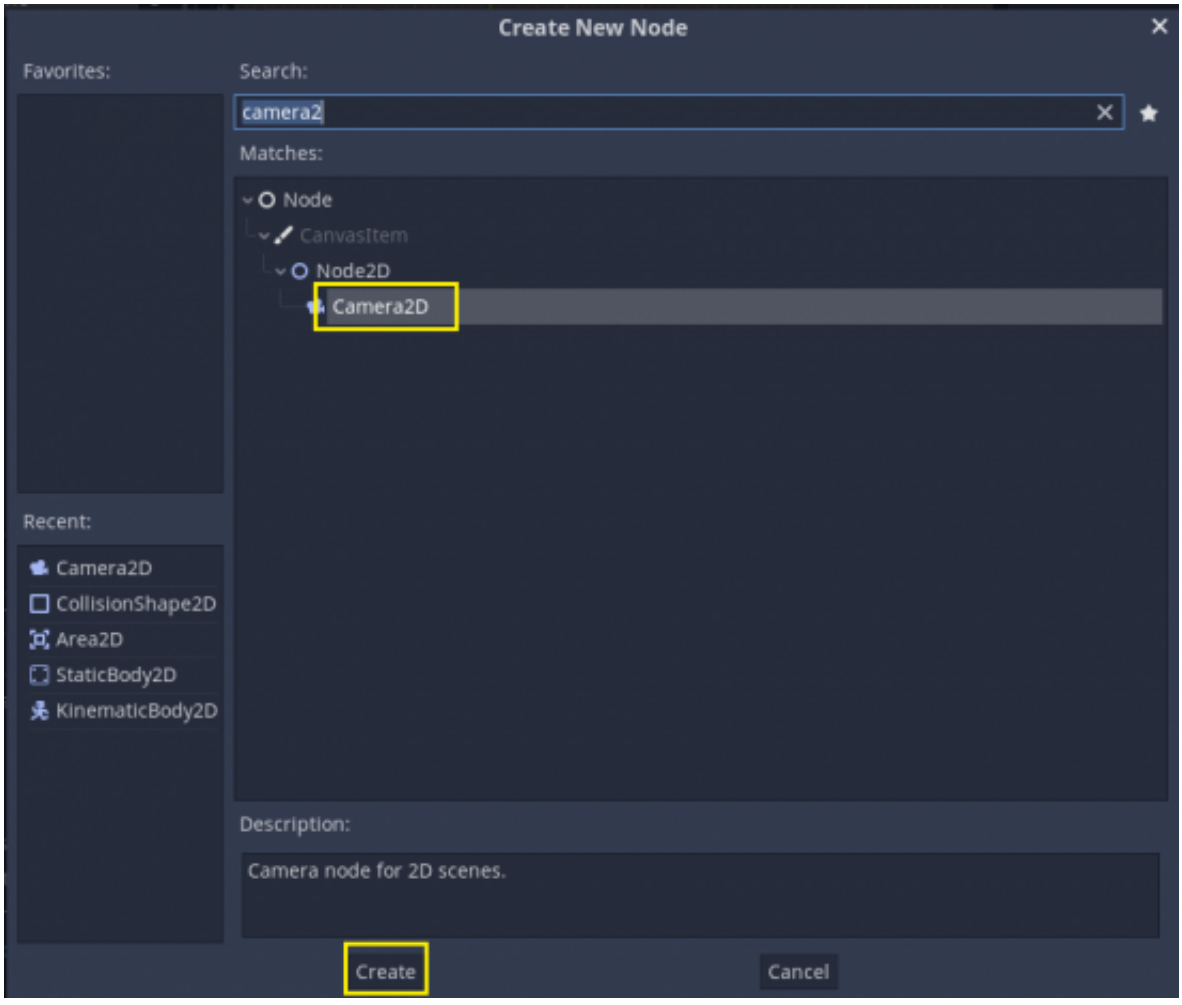


Now if we press **Play**, we can see that the scene reloads when we collide with this enemy.

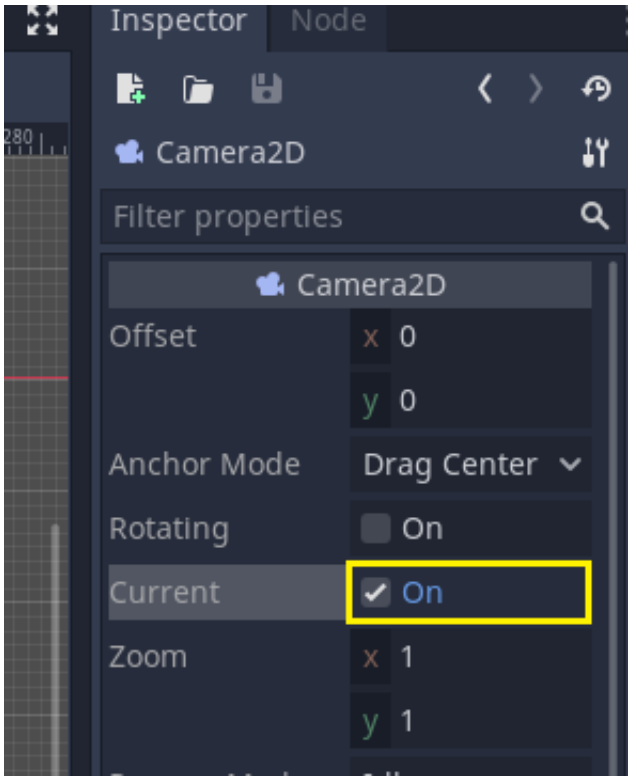
In this lesson, we're going to be creating a camera in Godot and having it track our player horizontally.

Create a new node of type **Camera2D** in the main scene.

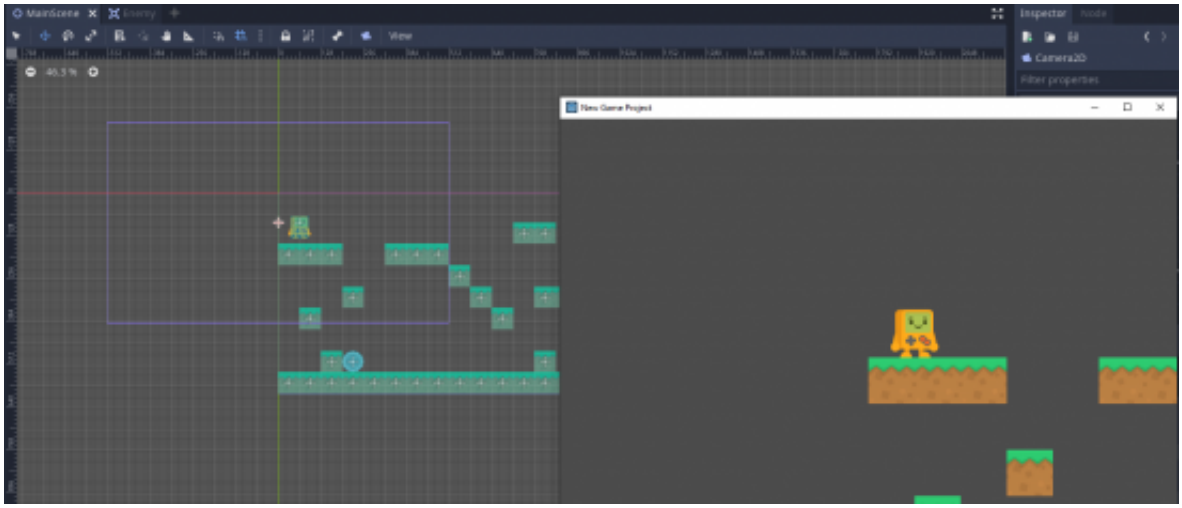




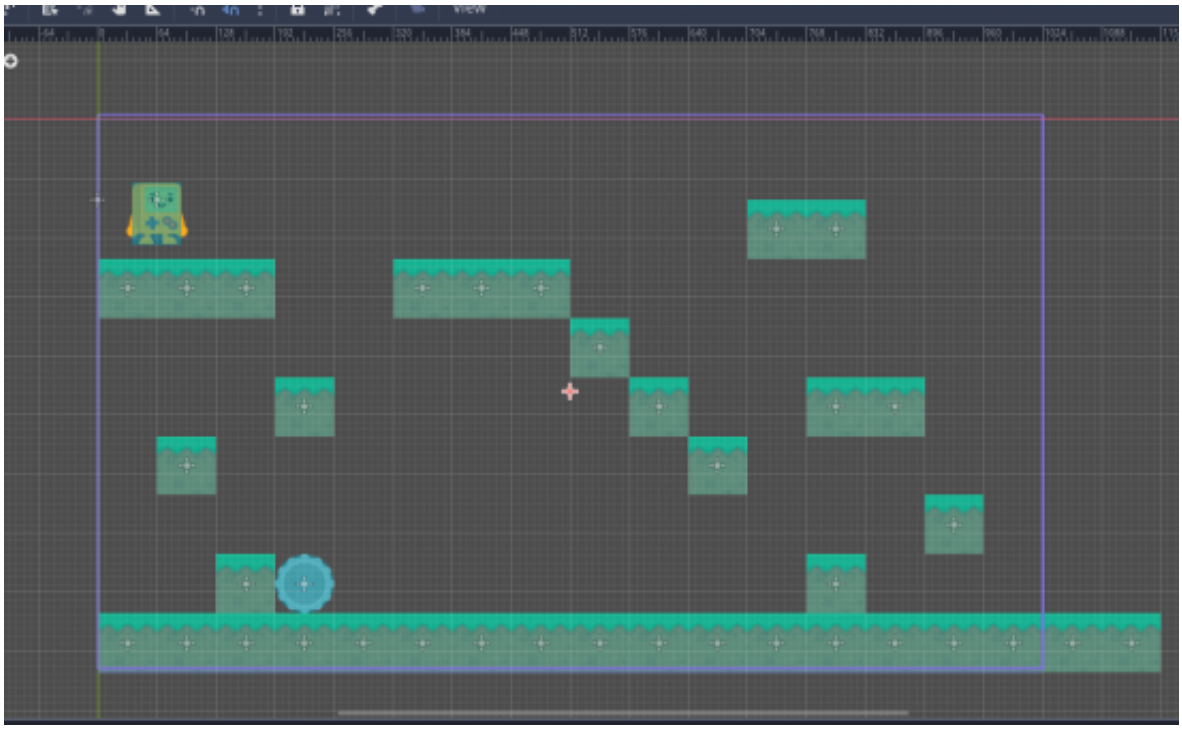
Select the **Camera2D** and enable the '**Current**' property in the inspector.



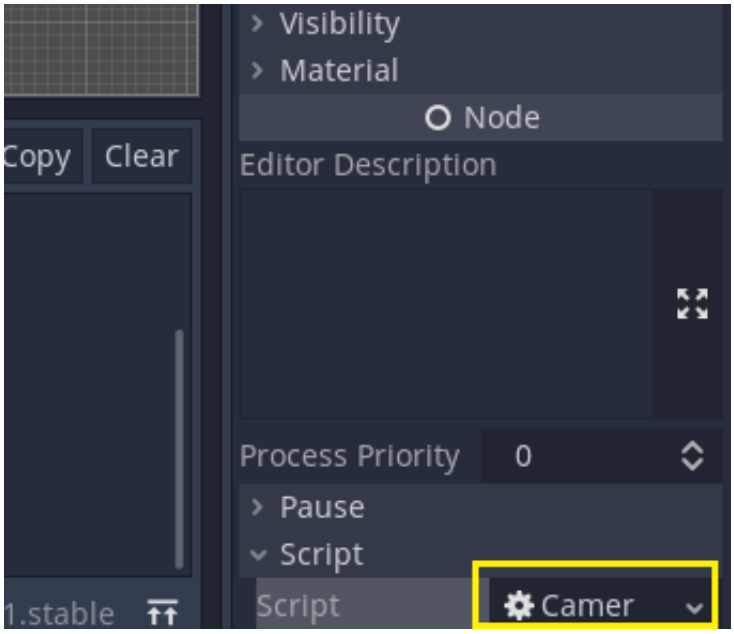
You will see that the blue outline appears in the viewport, showing the range of the current camera.

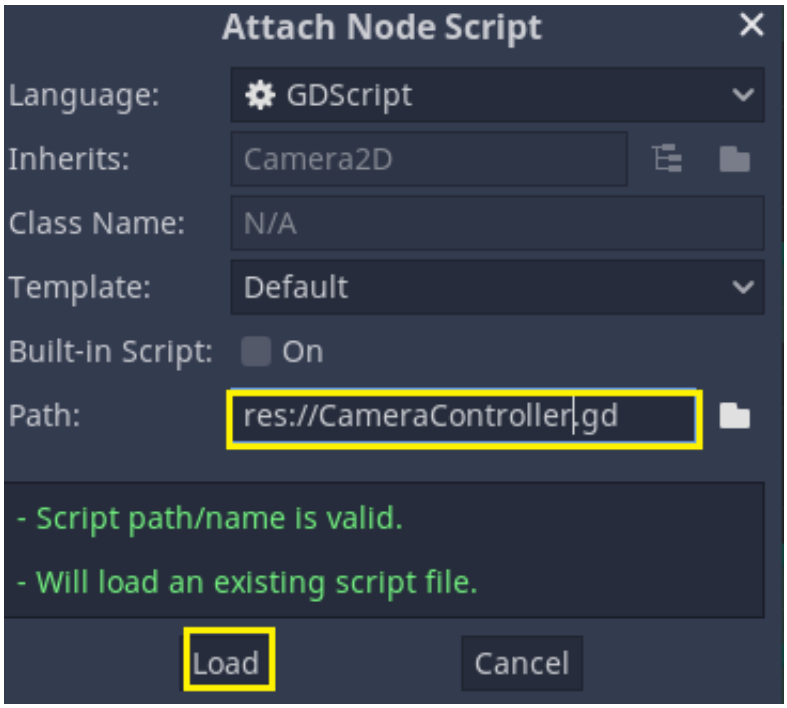


Now, let's make our camera to track our player. Click and drag the camera to where we want it to be initially.



Create a new script inside **Camera2D** and name it as **CameraController**.

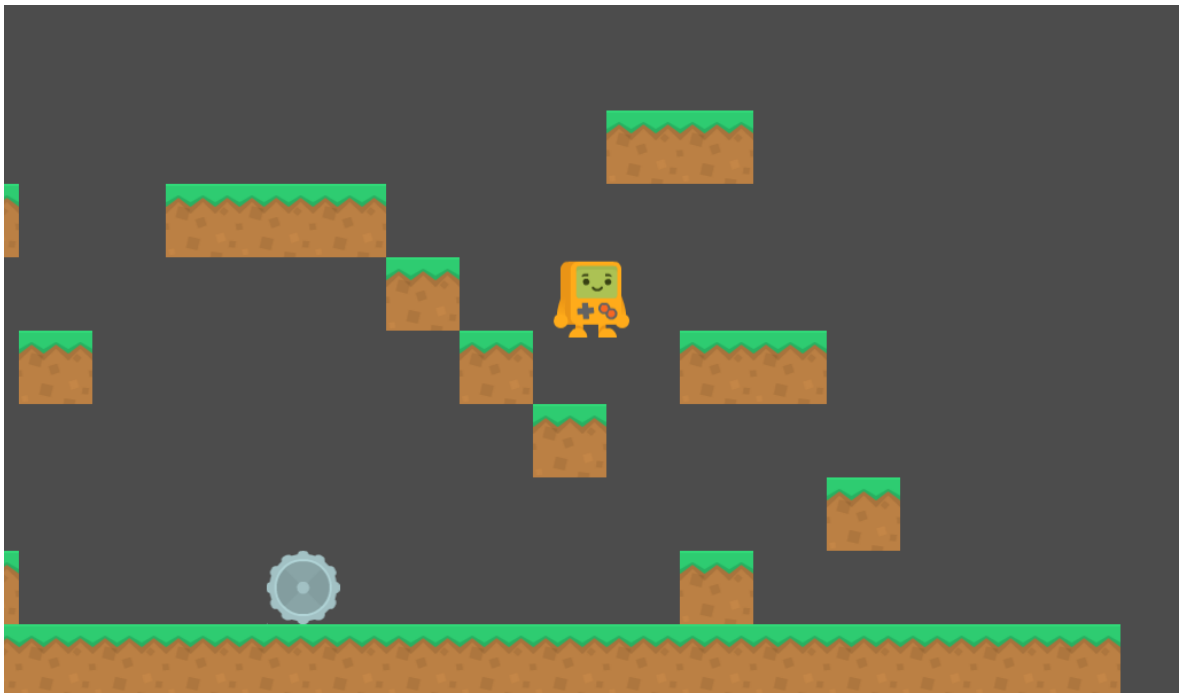




What we have to do inside the script is simply find our player by using **get_node()**, and then set the camera's x position to be the same as the player's x position.

```
extends Camera2D

onready var player = get_node("/root/MainScene/Player")

#This function gets called every frame
func _process (delta):

 position.x = player.position.x
```

Save the script and press the **Play** button.



The camera now tracks us along the x position.

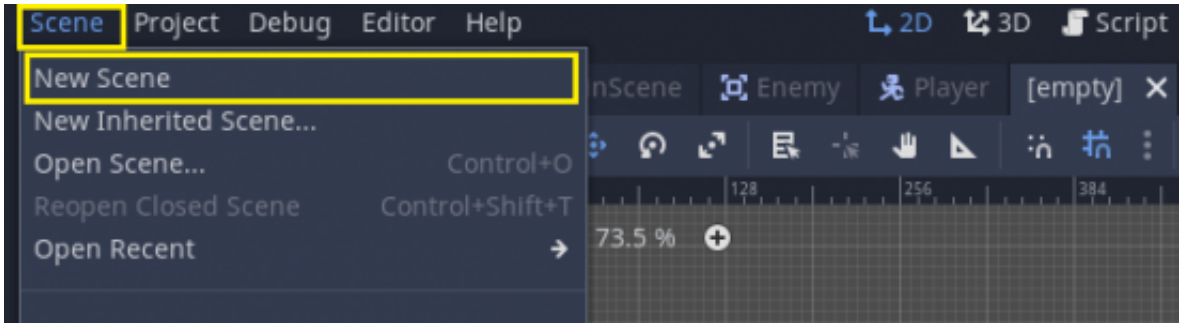
In this lesson, we are going to implement the ability to collect coins.

Let's start by opening up the **Player** script.

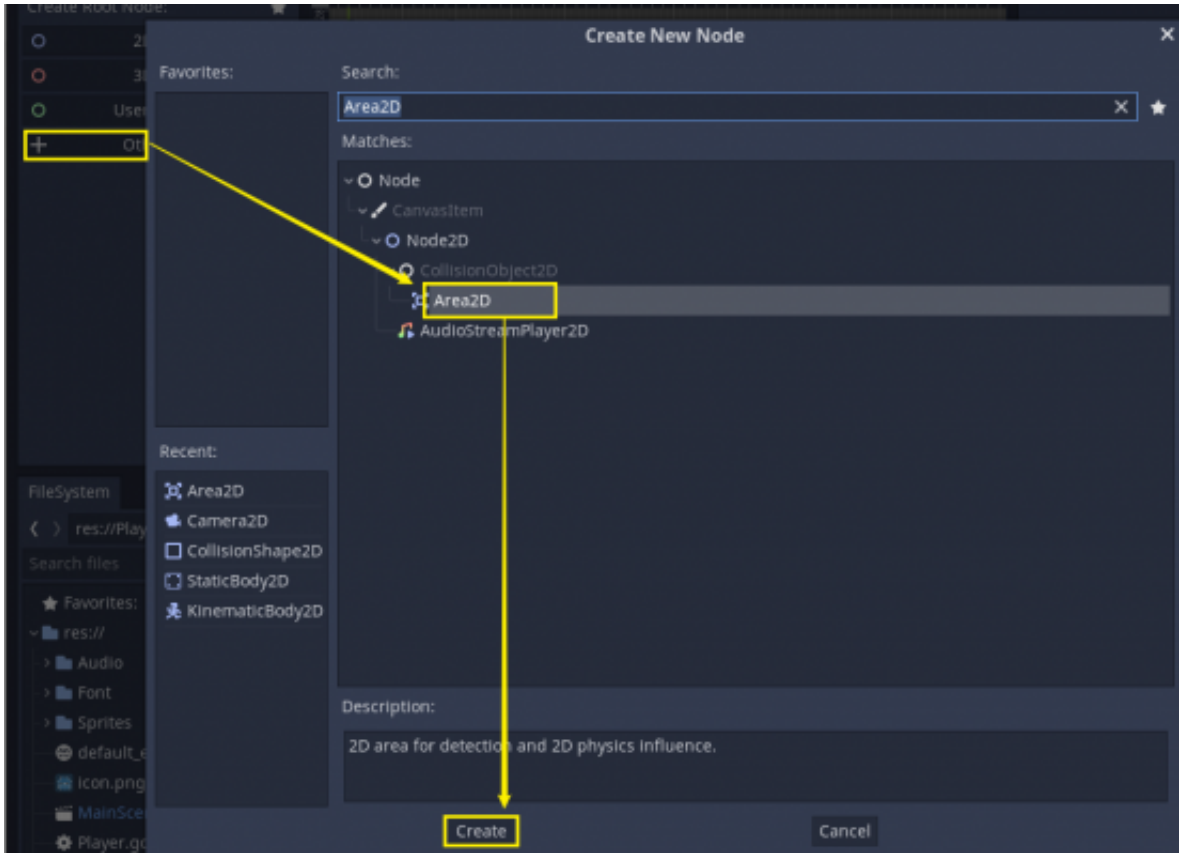
```
func collect_coin (value):
 score += value
```

Create a new function called **collect_coin()**. This function will be called when we hit a coin, and the value of the coin will be added to the player's score.

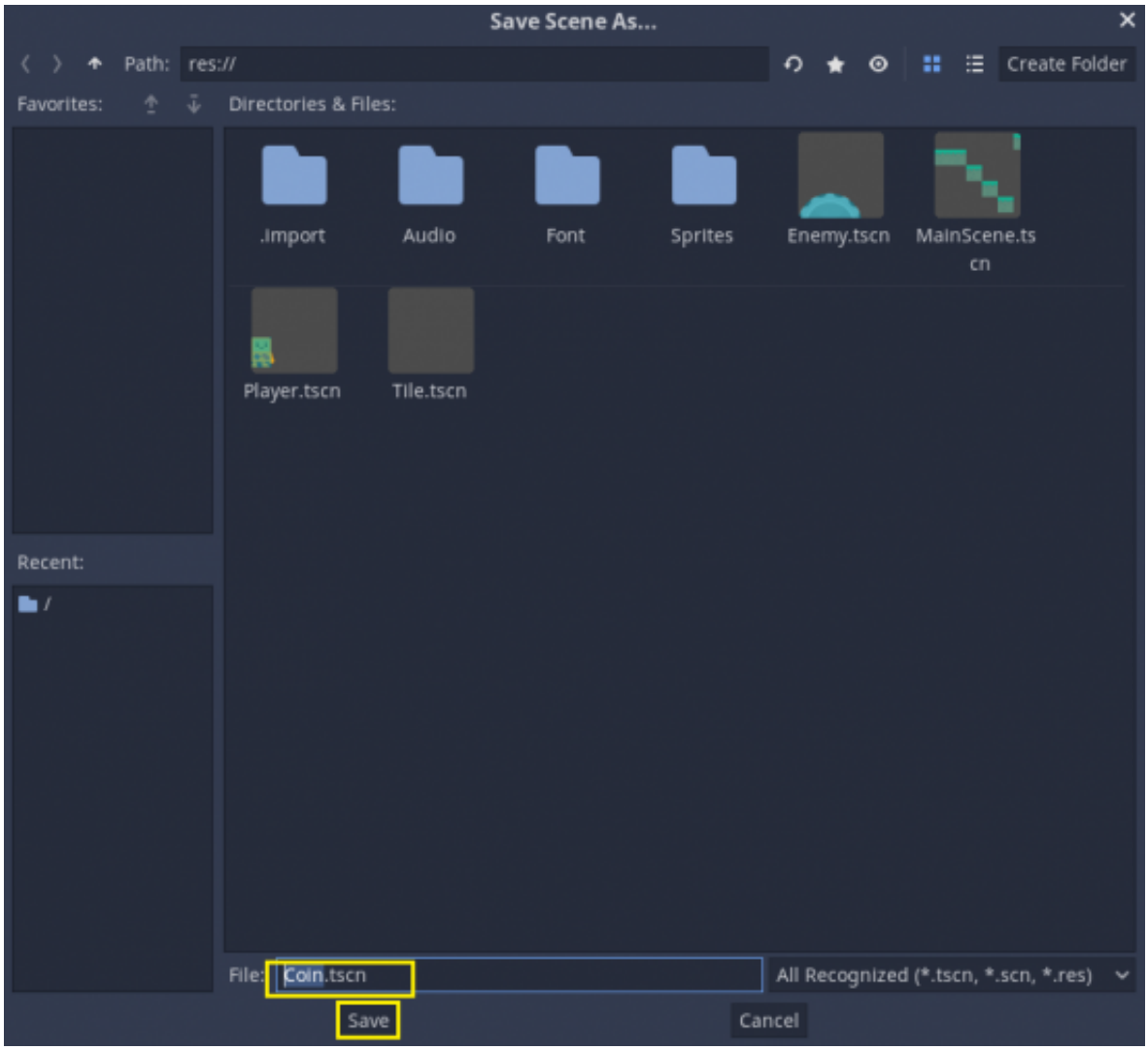
Now, we need to set up a coin object. Go to Scene > **New Scene**.



Create a new root node of type **Area2D**. Note that Area2D can detect a collision, but it doesn't have a physical body so you can't hit it and bounce off.

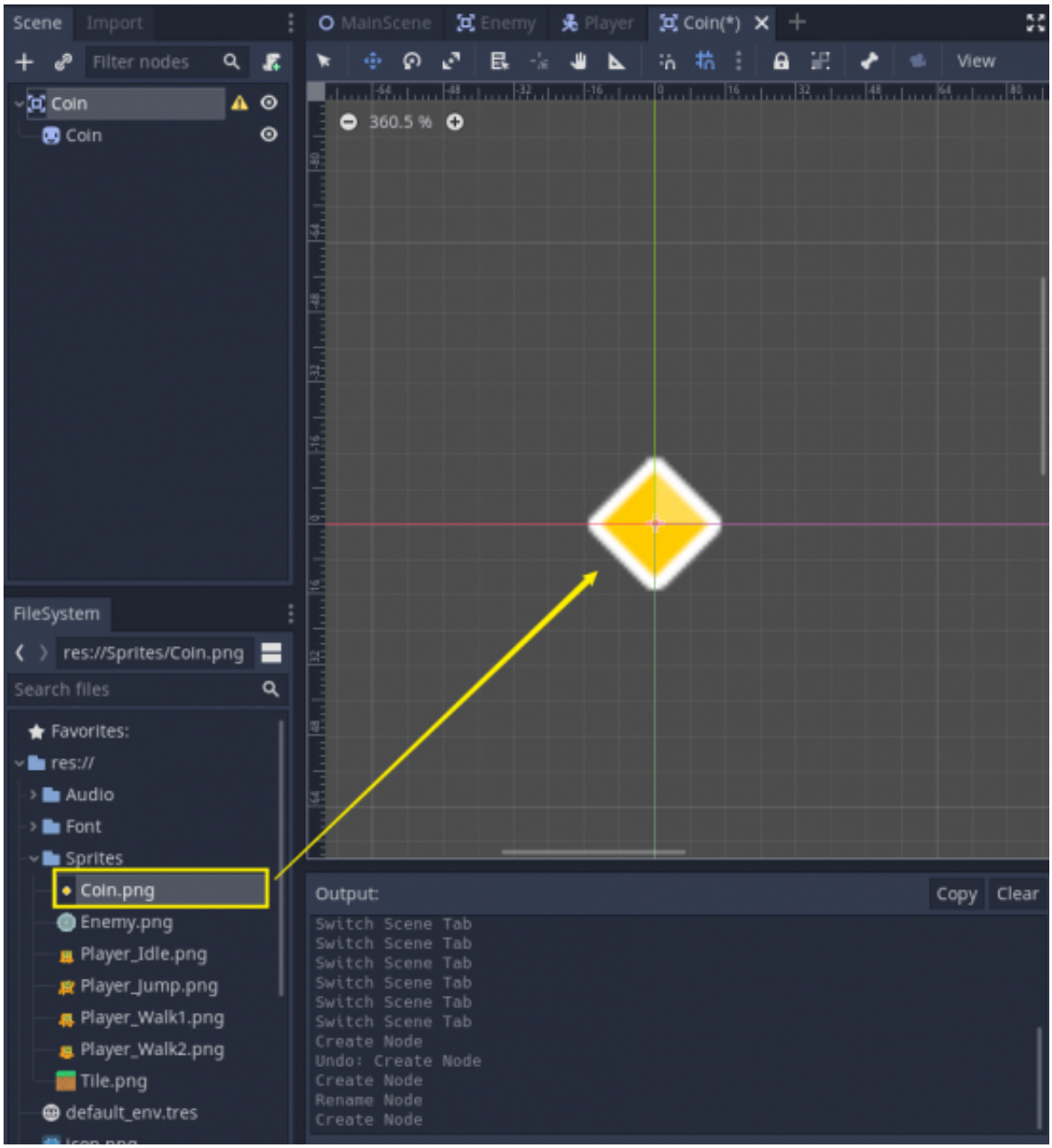


Rename the **Area2D** node to 'Coin', and save it as **Coin.tscn**.

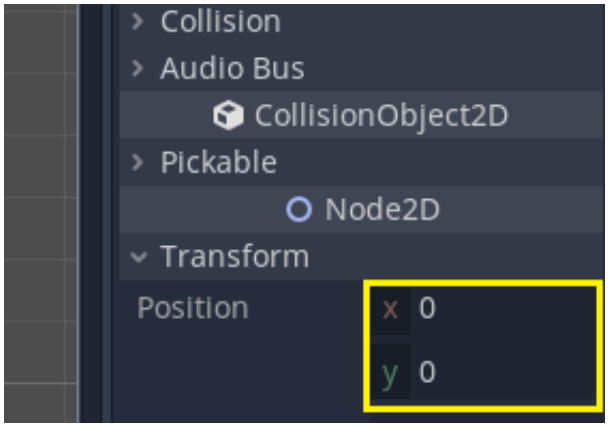


As a child of this node, we want to add two things: a **sprite** and a **collider**.

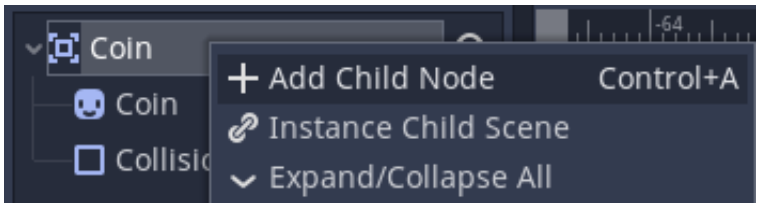
First, select and drag the **Coin.png** from FileSystem into the viewport.



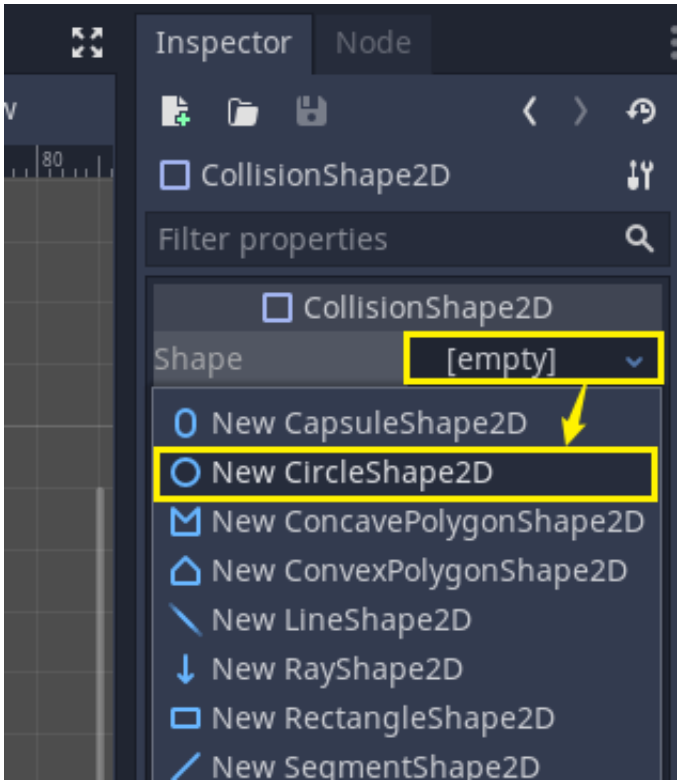
Rename the node to ‘Sprite’ and make sure that the position is set to (0,0).



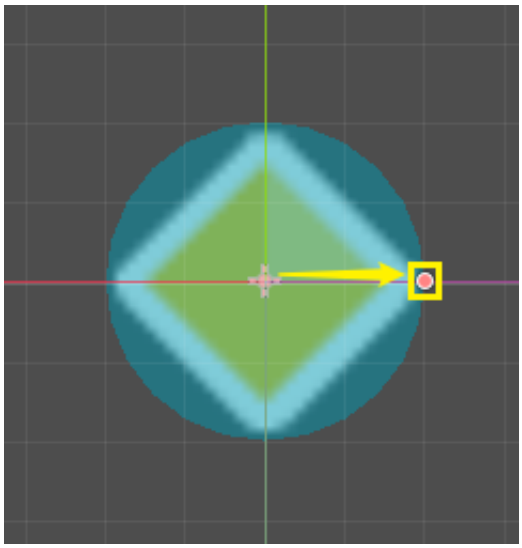
Then, add a child node (Ctrl+A) of type **CollisionShape2D**.



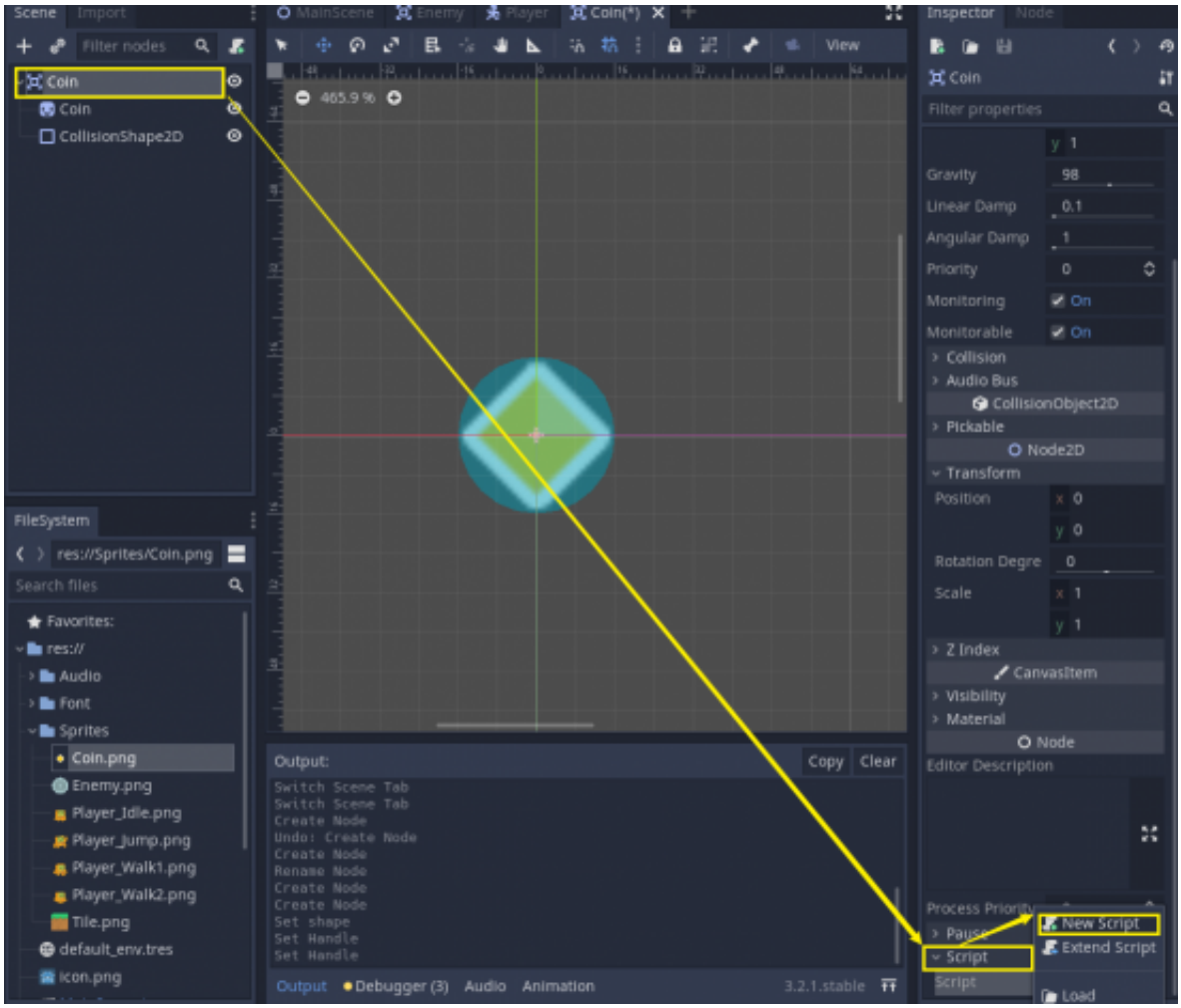
Click on the ‘Shape’ property in the Inspector, and select New **CircleShape2D**.

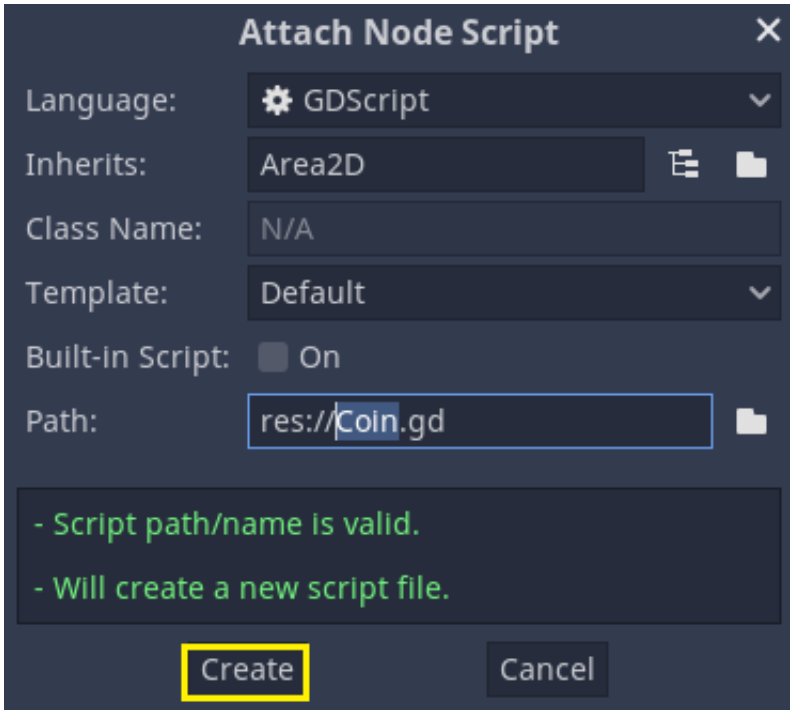


Click and drag the little orange circle to resize the collider.



Select the root ‘Coin’ node, and go to **Script > New Script** in the Inspector.



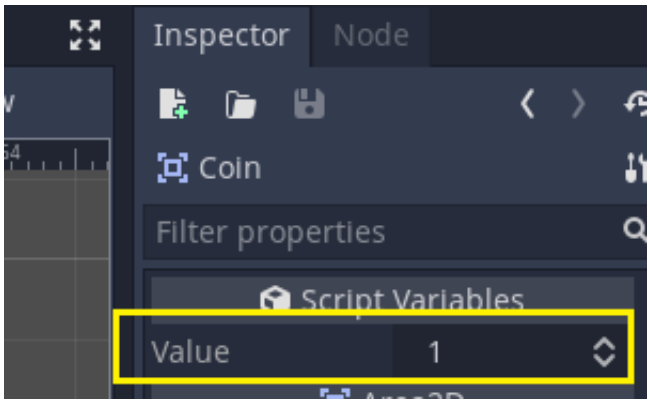


Within the script, we want to detect the collision with the player and give them the corresponding value that we have on this coin.

Let's start by creating a variable of type integer to store the value.

```
extends Area2D

export var value : int = 1
```

The '**export**' keyword allows us to modify the value in the Inspector.



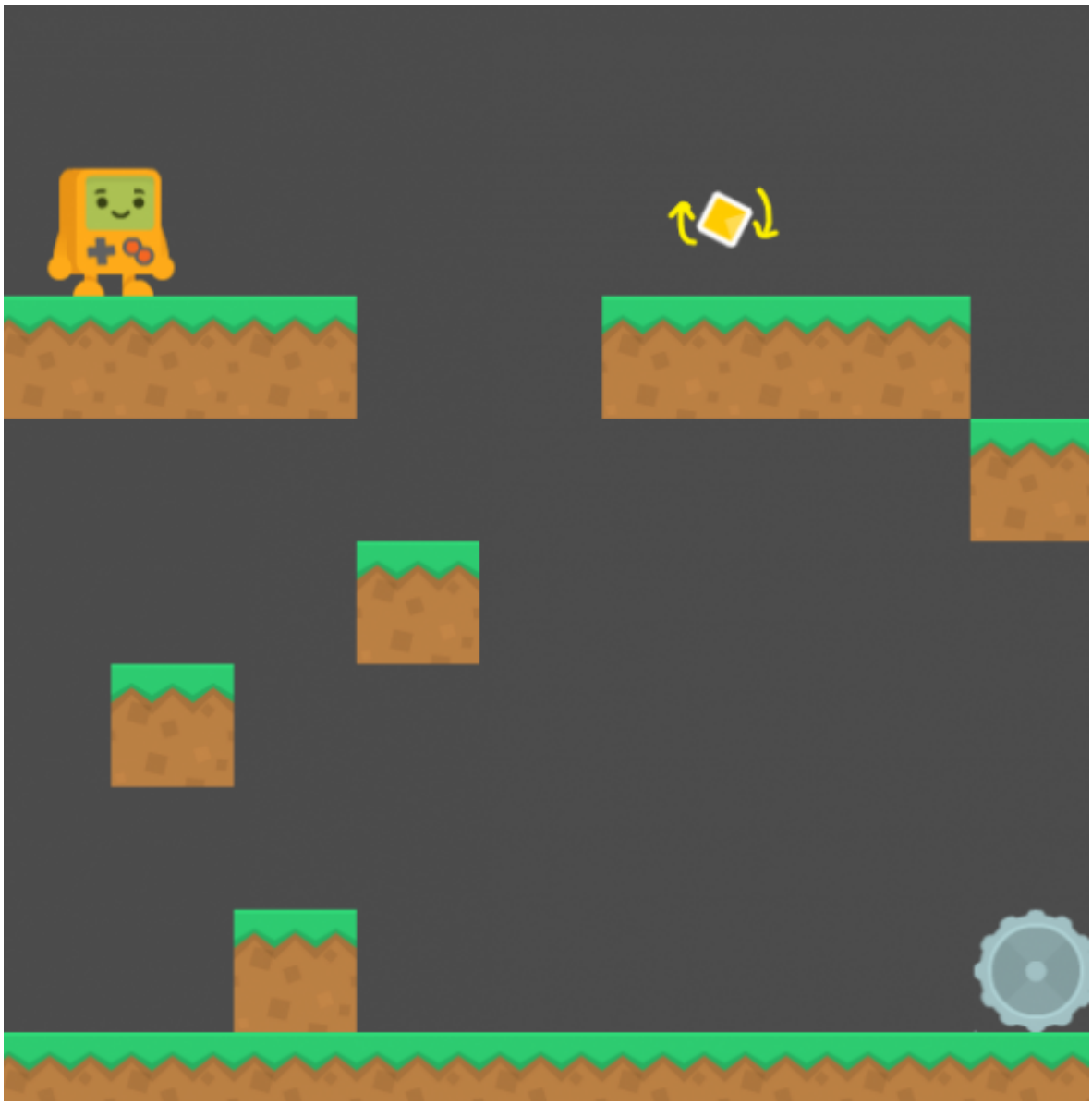
This value may vary for each instance, which means there can be multiple coins that have different values.

Now, we would like to rotate the coin over time. You can access the rotation property directly by using **rotation** or **rotation_degrees**.

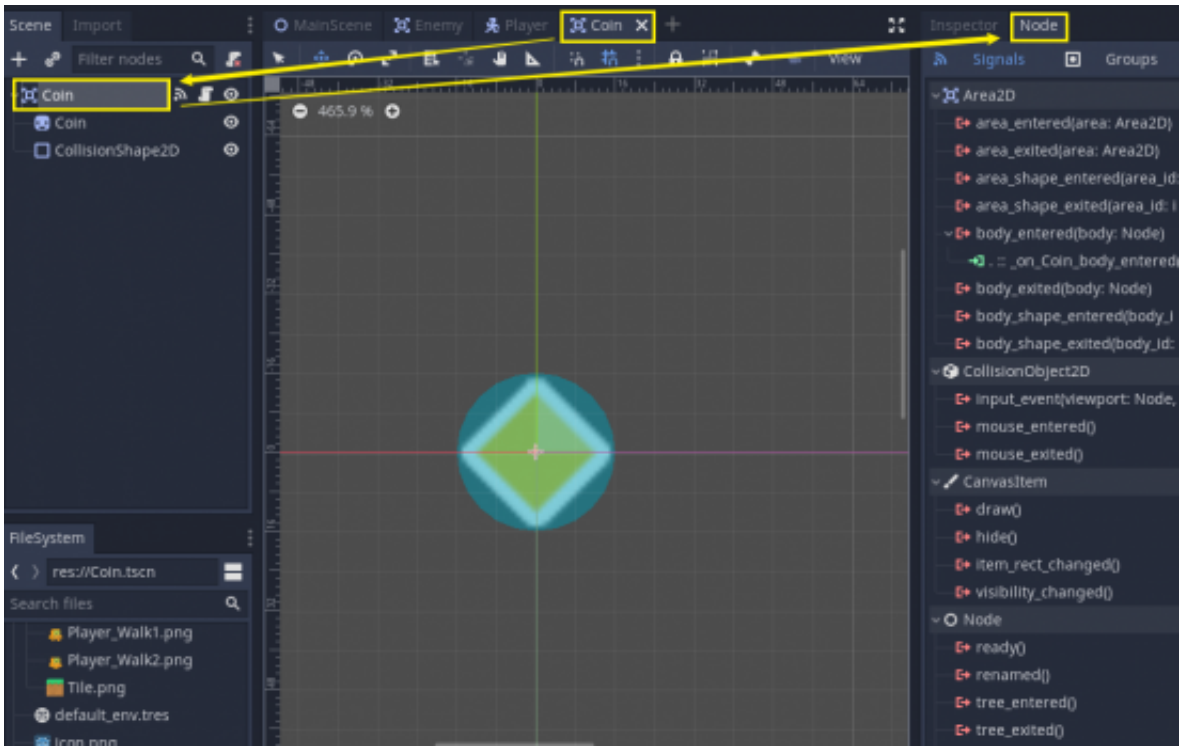
```
extends Area2D

export var value : int = 1
var rotationSpeed : float = 90.0

func _process (delta):

 rotation_degrees += rotationSpeed * delta
```

Note that the function **_process()** gets called every frame, and while we can change the rotation property every frame, it is not easy to find the adequate rotation speed to apply. So we multiplied **rotationSpeed** by **delta** to convert from 'rotation **per frame**' to 'rotation **per** second'.

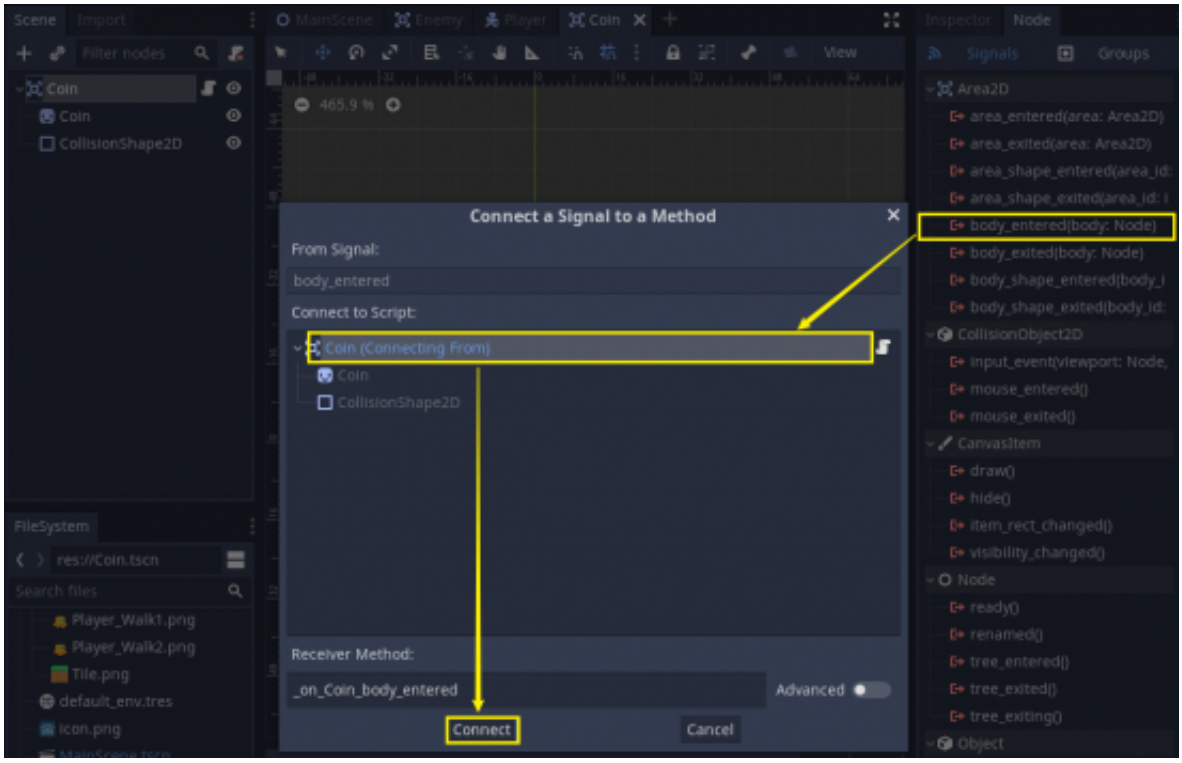
If we drag and drop the coin object into the scene and press Play, we should see that the coin is spinning around.



We still need to give it an ability to detect a collision with the player. Let’s go back to our coin scene, and open up the **Node** panel.



Create a **signal** of type **body_entered** by double-clicking on it.



Again, this will create a function inside the coin script.

```
func _on_Coin_body_entered(body):
```

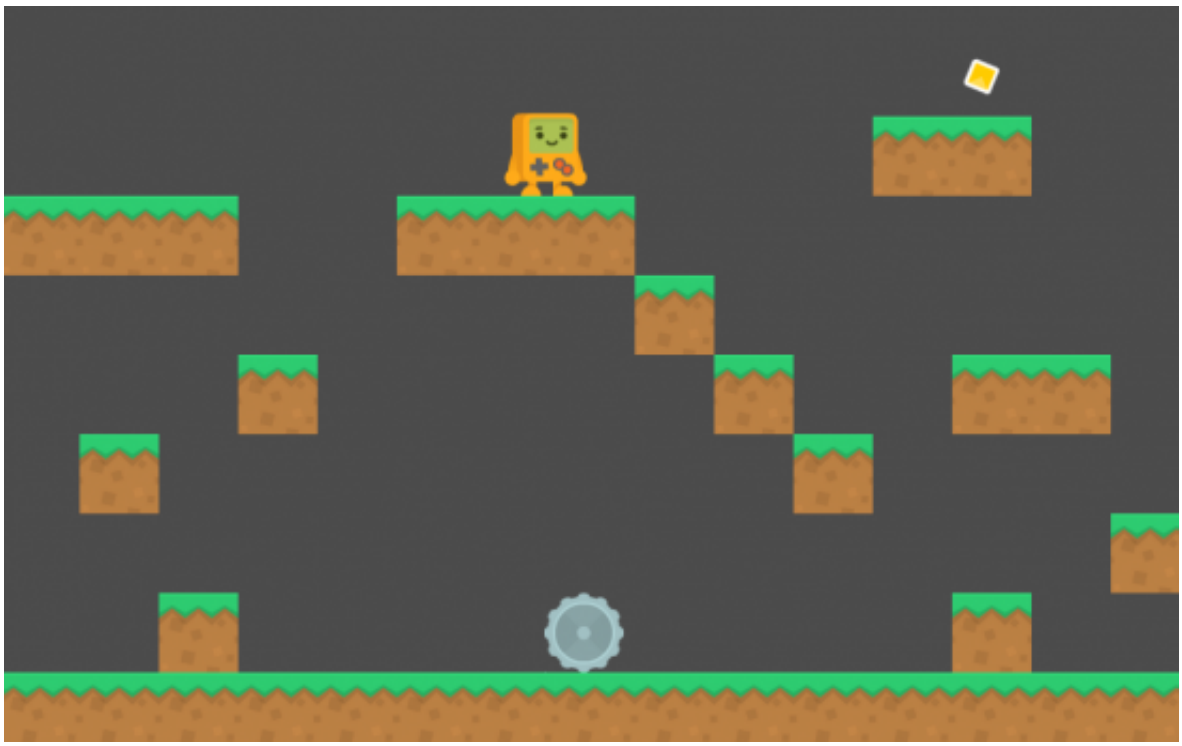
This function should detect if the player has entered the body and call the function called **collect_coin** that we created earlier. (Don't forget to send over the **value** of the coin.)

We then need to destroy the coin off once it has been picked up, using **queue_free()**.

```
extends Area2D

export var value : int = 1
var rotationSpeed : float = 90.0

func _process (delta):

 rotation_degrees += rotationSpeed * delta


func _on_Coin_body_entered(body):

 if body.name == "Player":
  body.collect_coin(value)
  queue_free()
```

queue_free() is another Godot's built-in function that destroys the current node and all the children nodes along with it.

Save the script and hit Play.

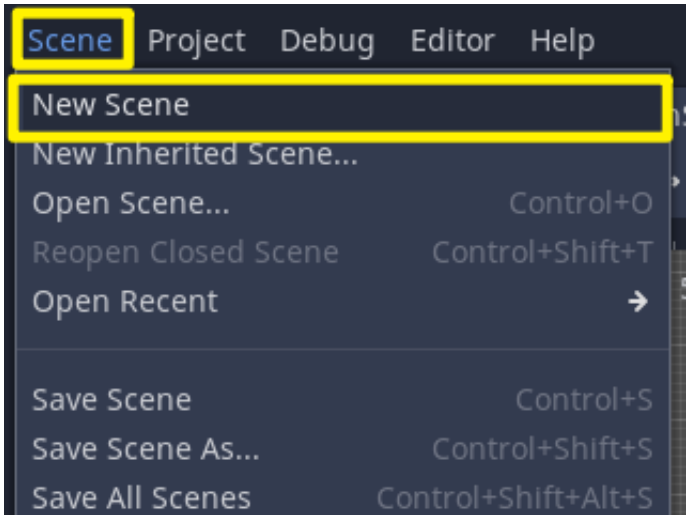


We can now see that the coin gets destroyed as soon as we collide with it.

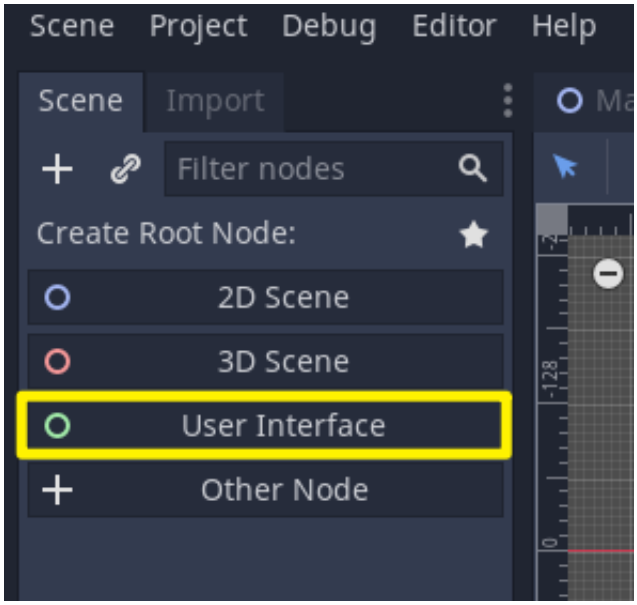
In this lesson, we are going to be setting up a user interface so that we can see our current amount of score.

Control Node

Create a New Scene by going to **Scene > New Scene**.

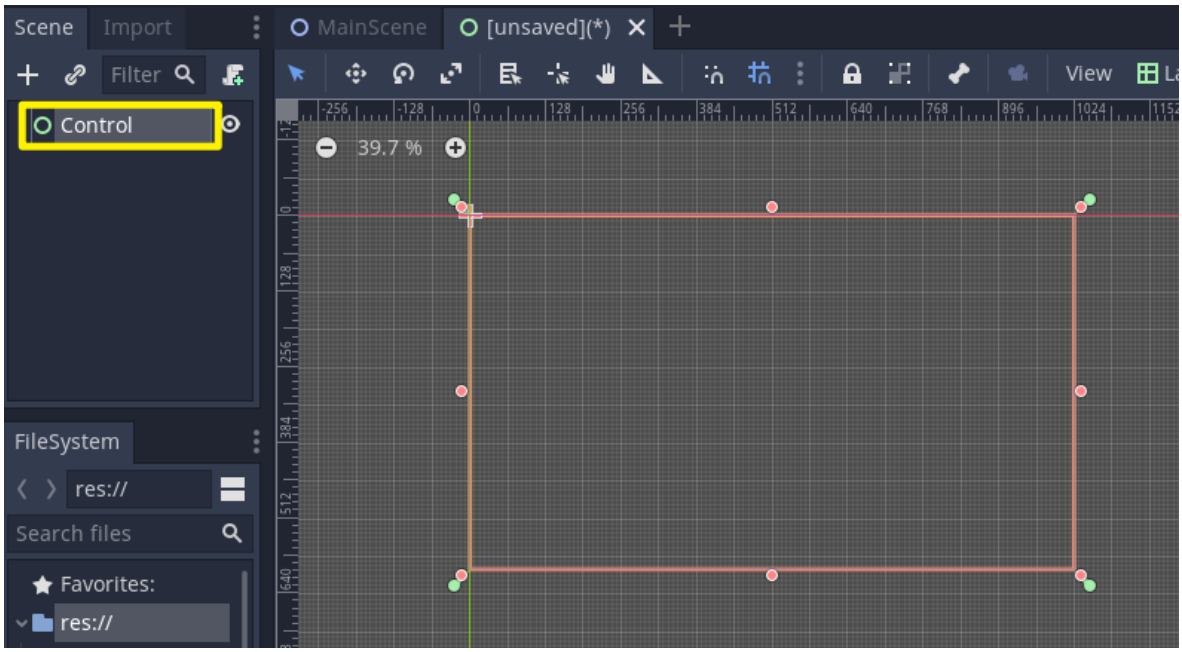


Create a new Root Node of type **User Interface**.

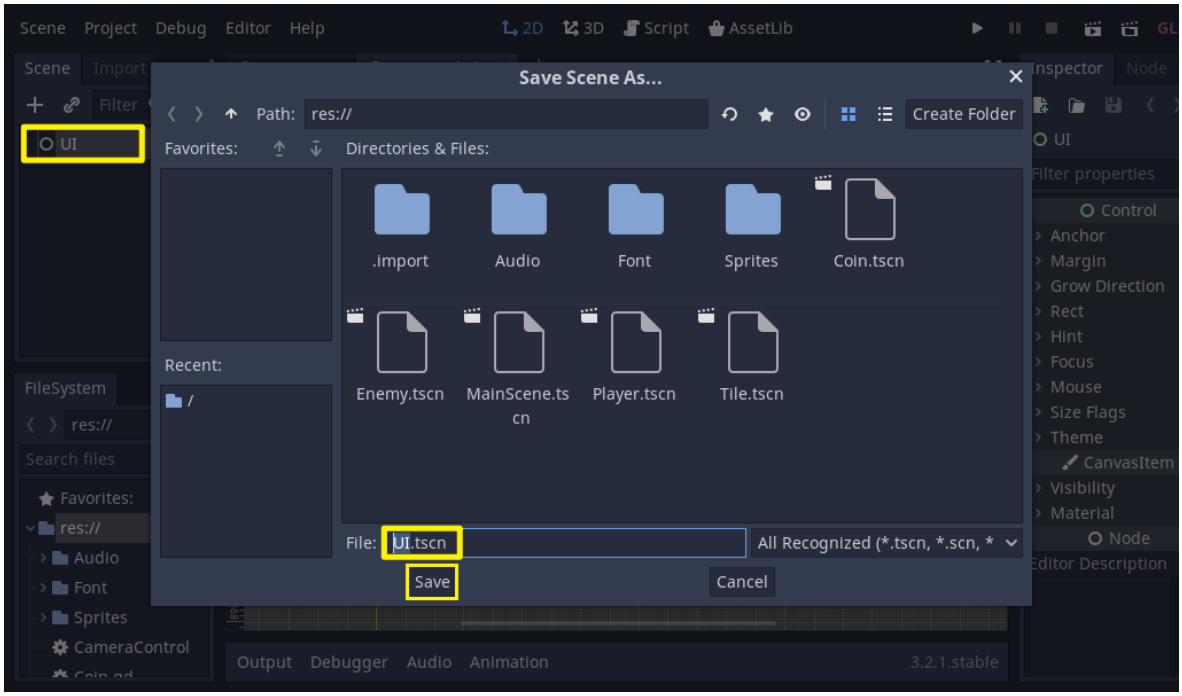


All the UI nodes are made up of **Control nodes**, which manage how they appear on the screen (e.g. size).

For more information, check out the documentation: https://docs.godotengine.org/en/stable/getting_started/step_by_step/ui_introduction_to_the_ui_system.html



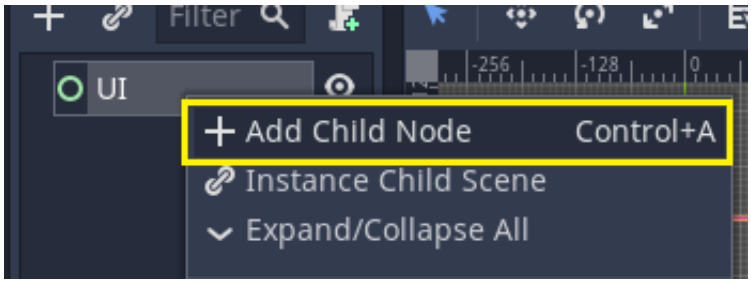
Let's rename the node to "**UI**" and save it (CTRL+S) as a scene called "**UI.tscn"**.

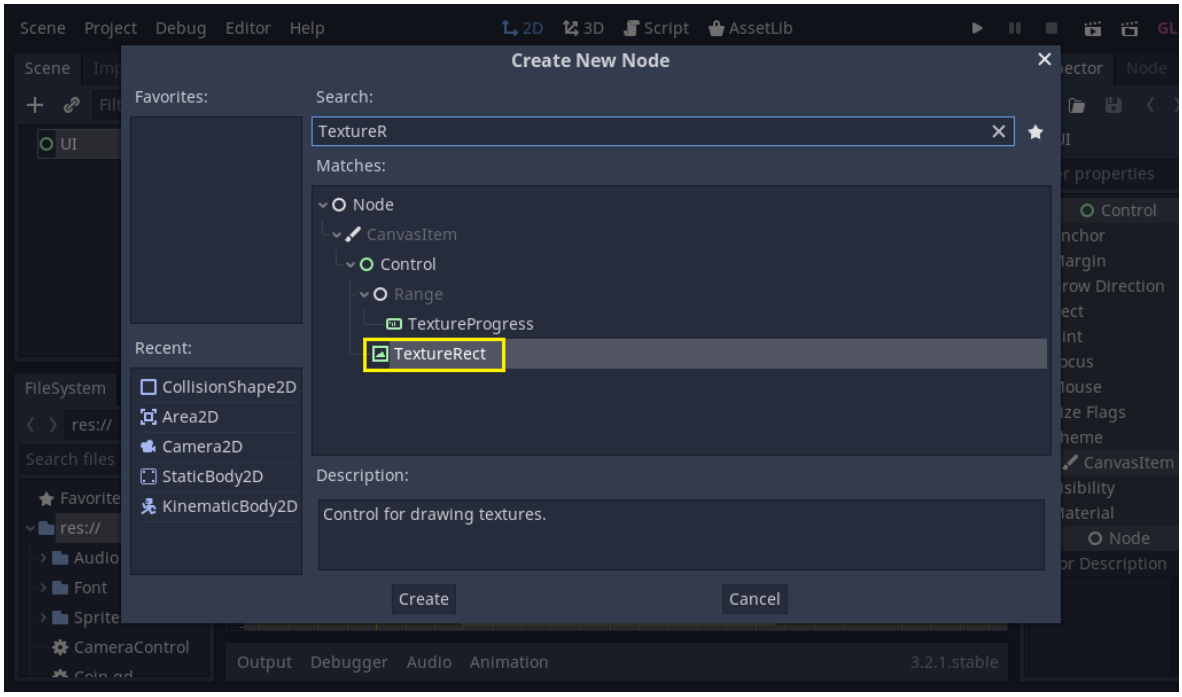


Setting up a texture

We now need to set up a texture to display on the screen.

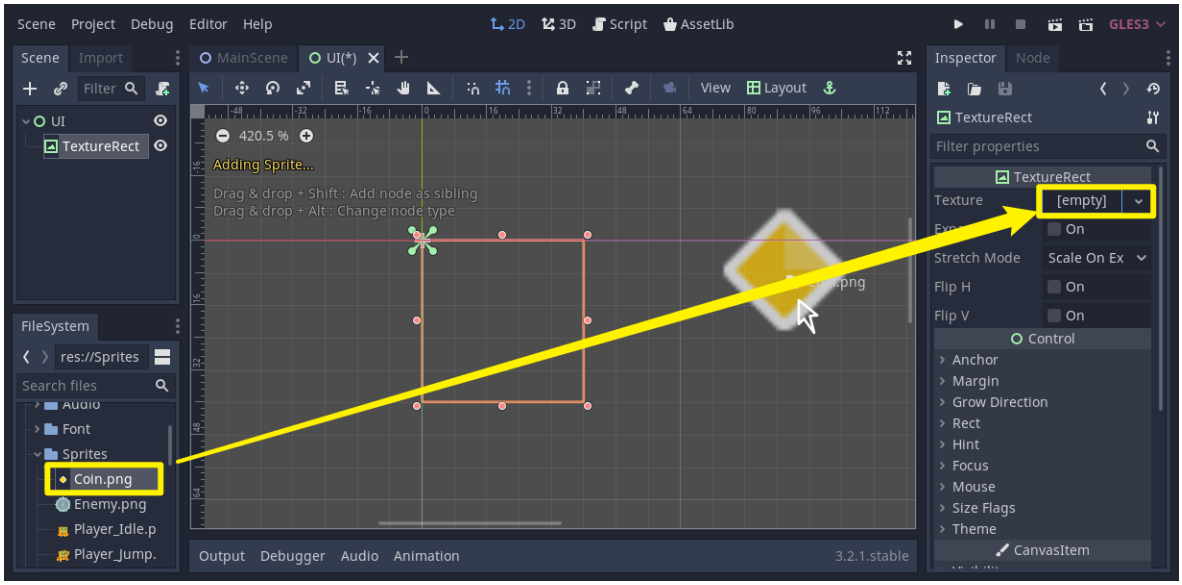
Right-click on UI, select '**Add Child Node**' and look for '**TextureRect**'.



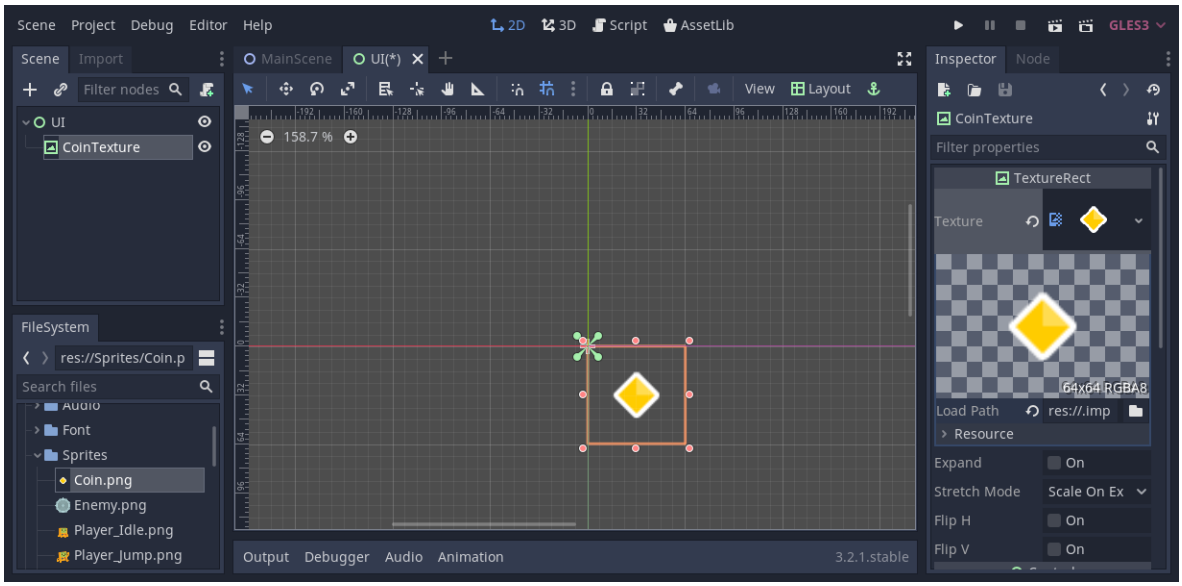


In the inspector, we can see that the texture is currently set as [empty].

Simply drag in the image (Coin.png) from our project folder into the texture field.



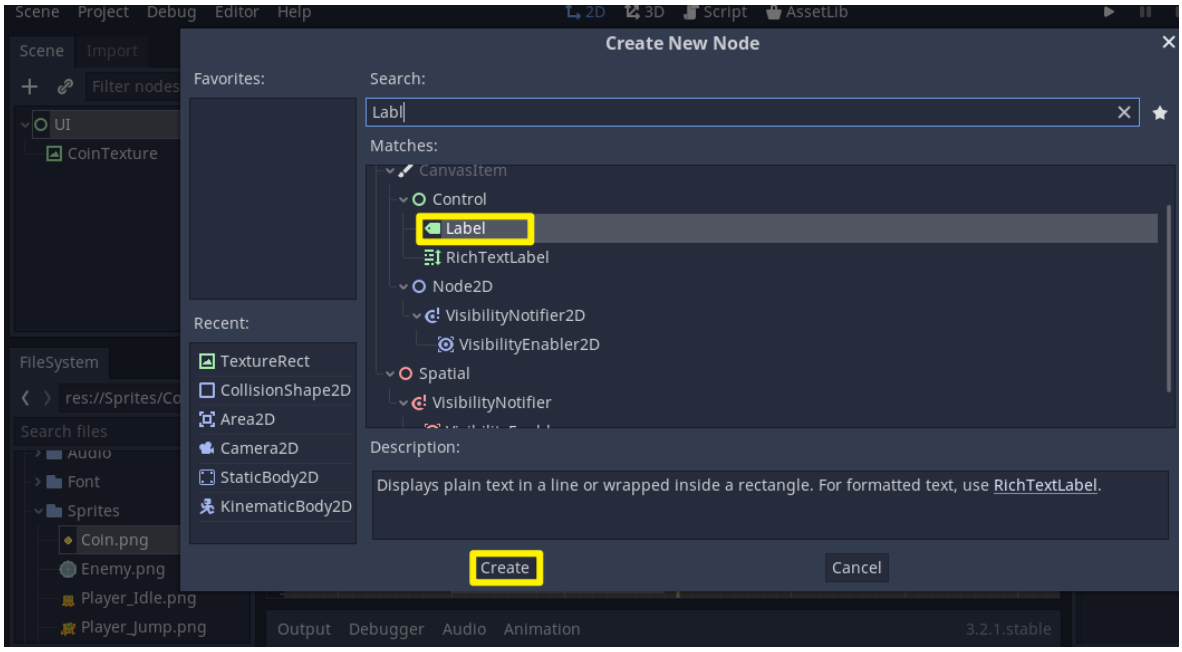
The image is now appearing in the scene. Let’s rename this node to “CoinTexture”.



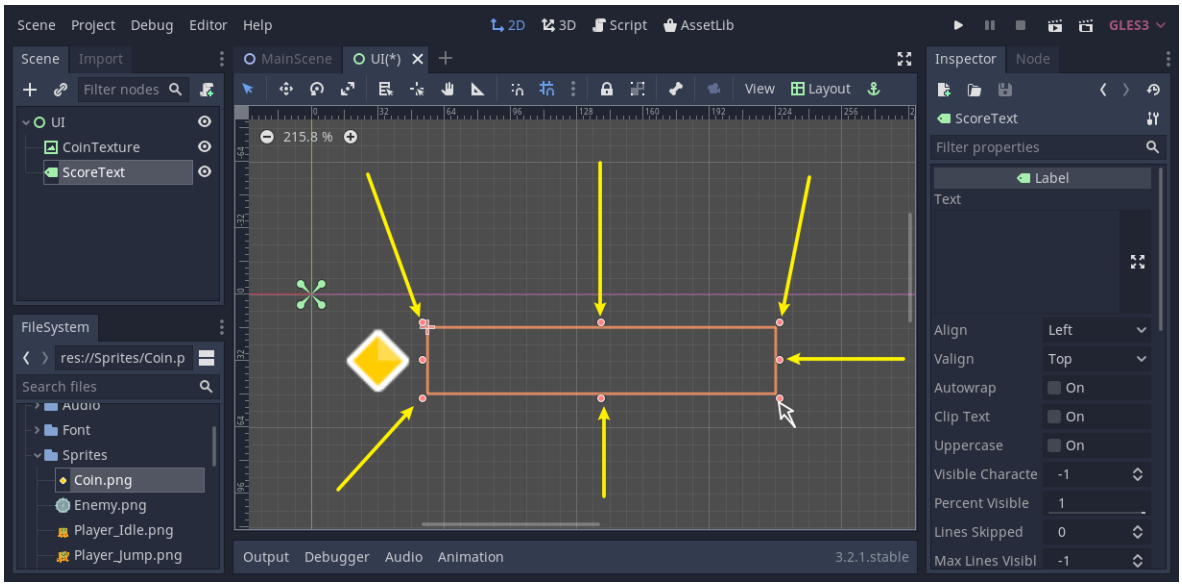
Creating a Label

Now, we need a piece of text to display our current score.

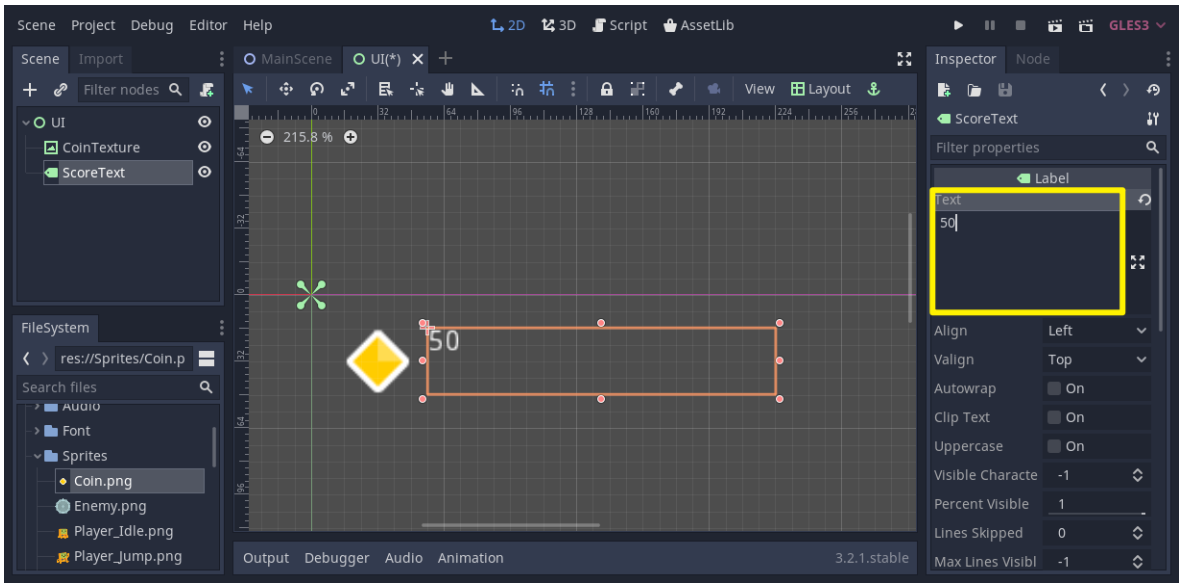
Right-click on UI, Add Child Node and select '**Label**', which displays plain text on the screen.



Rename the label node to 'ScoreText'. We can adjust the position/size of the text box by dragging on the control points.

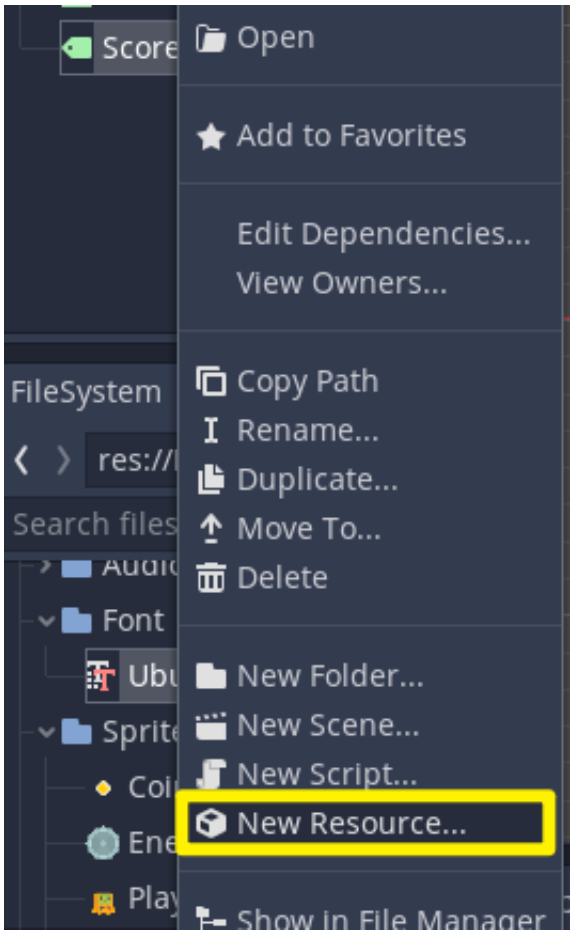


In the inspector, we can type something (e.g. 50) in the Text field.

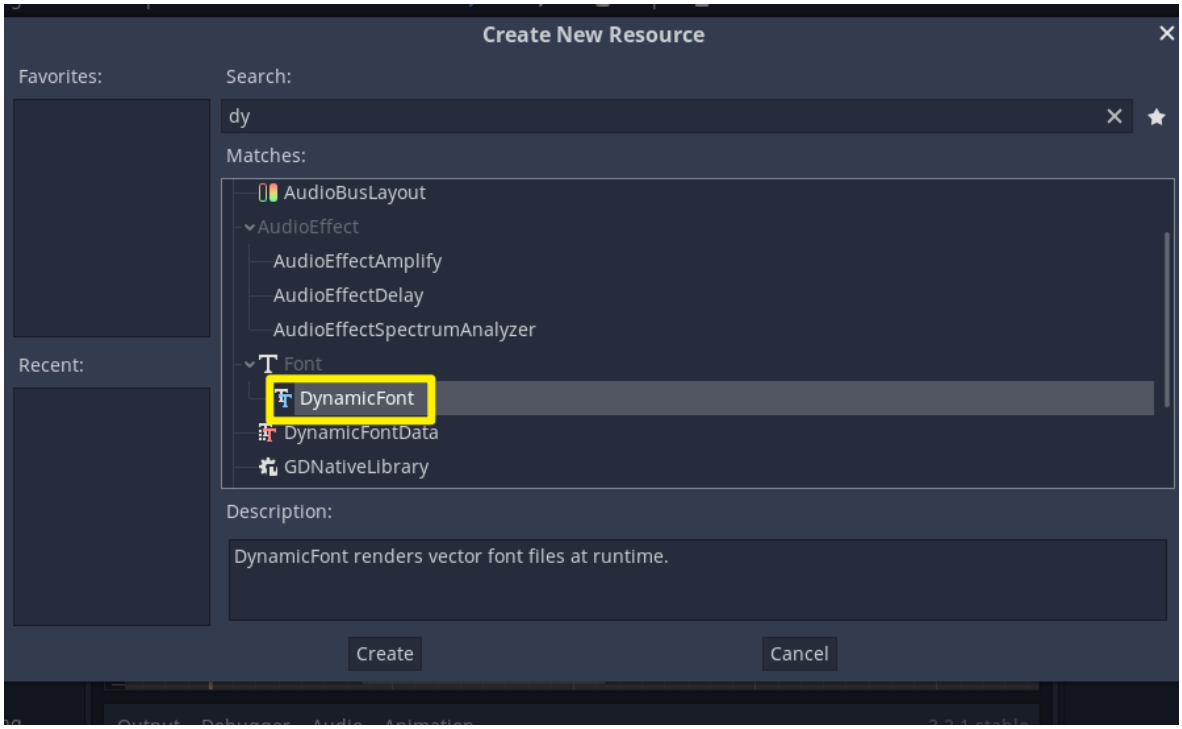


Let's assign a **Custom font** to the text.

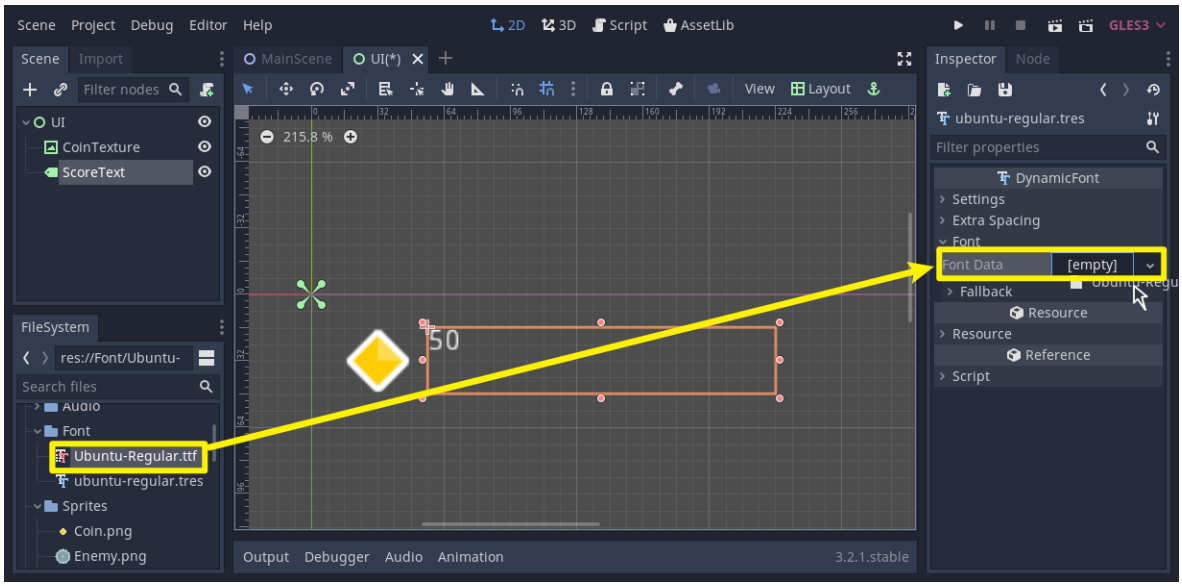
Open up the 'Font' folder, **right-click** on the font file (Ubuntu-Regular.ttf), and select '**New Resource**'.



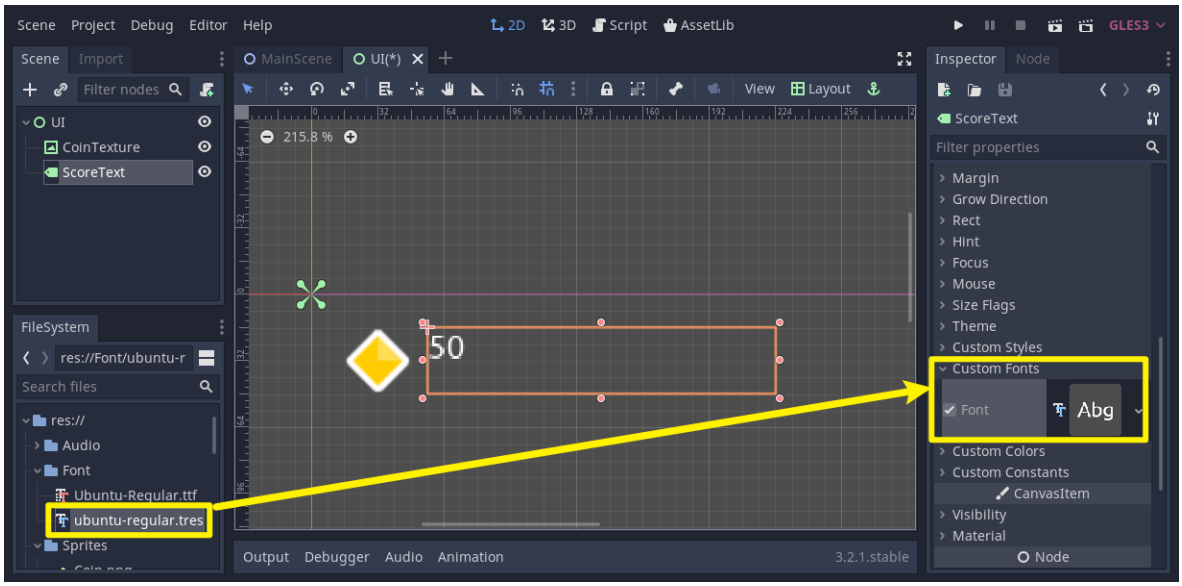
Select '**Dynamic Font**', and save it as "ubuntu-regular.tres".



Open up the Font folder in the inspector and drag in the font file (ubuntu-regular.ttf) to the **Font data**.

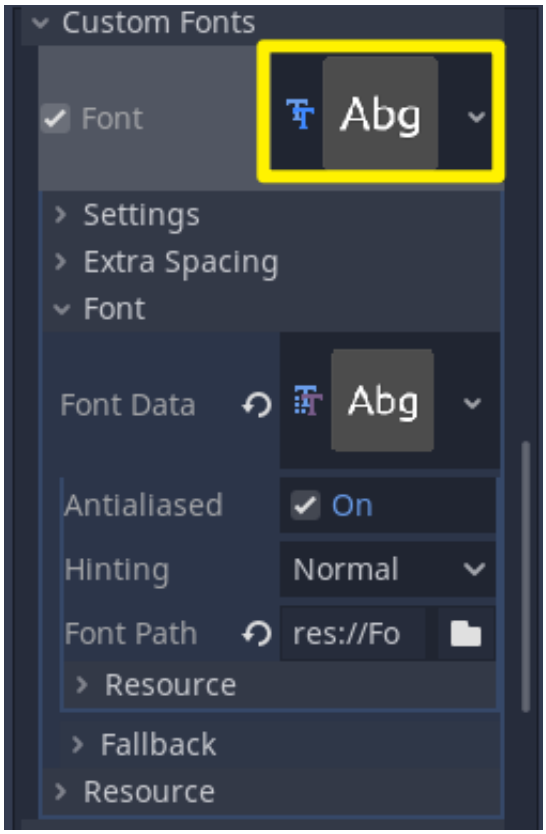


Double-click on 'ScoreText', and click on the **Custom Fonts** dropdown. We will drag in our Dynamic Font (ubuntu-regular.tres) here.



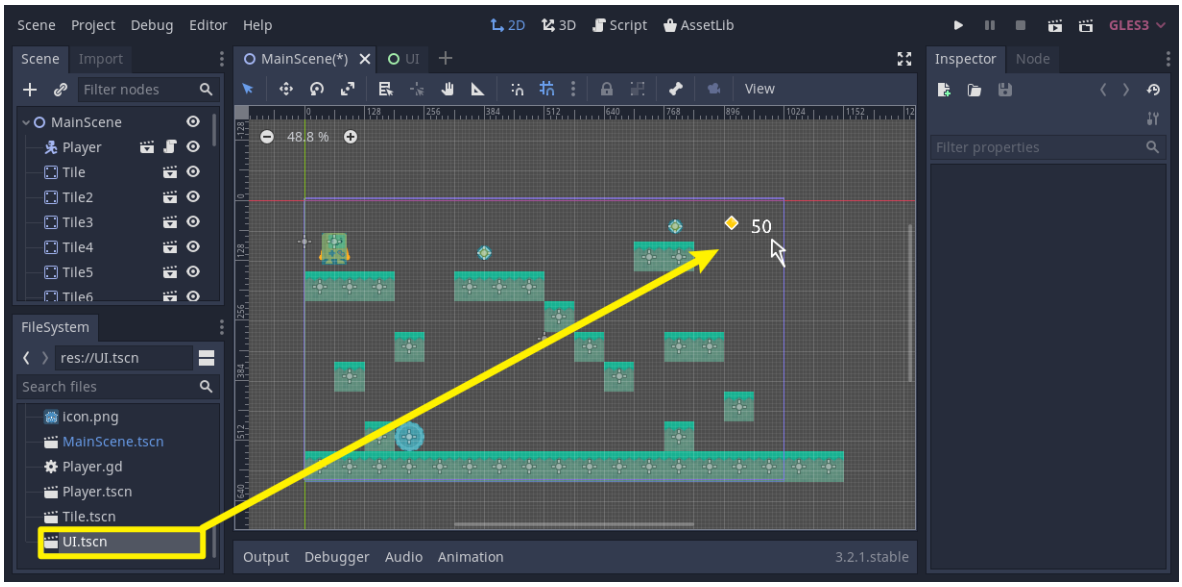
Now you'll see that the text appears with our new font.

Click on the Font property, and open up the **Settings** dropdown. We will set the size of the font to 40.

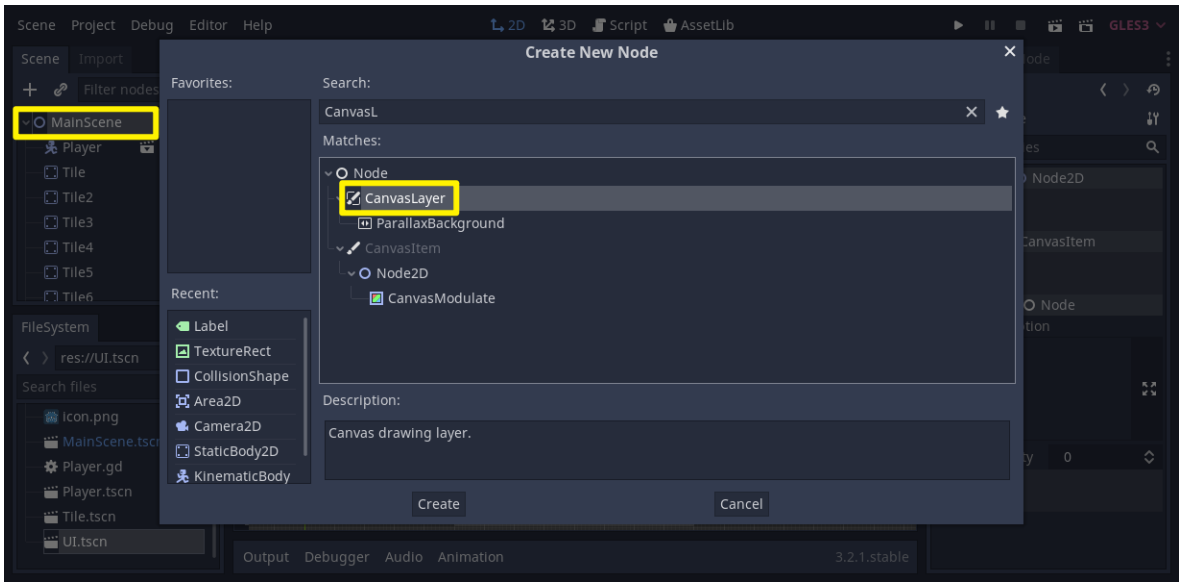


Let's **save** the scene (Ctrl+S) and switch over to our **main scene**.

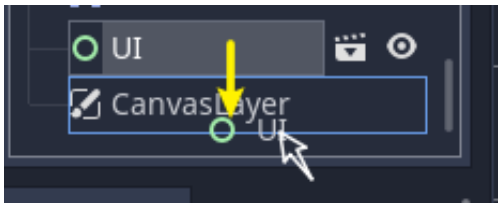
We can now drag in our UI scene (**UI.tscn**) into our main scene.

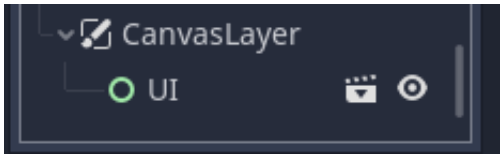


To make it align with our camera, we need to create a new node called **CanvasLayer**.

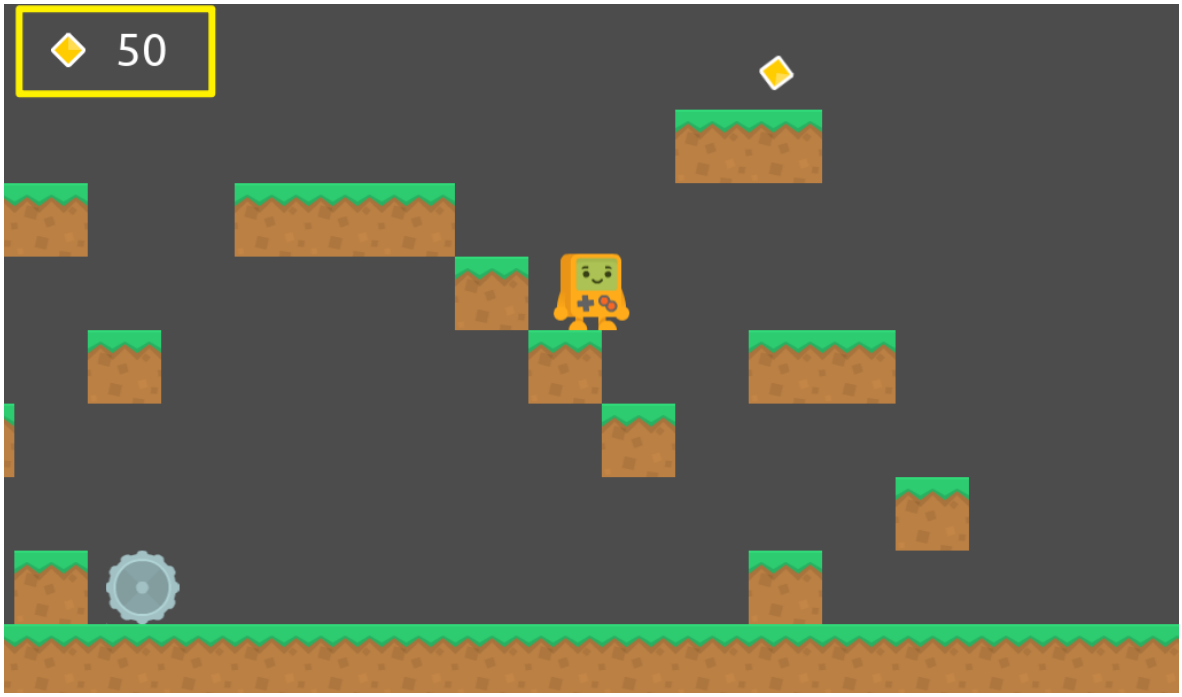


We then need to make the UI node a child of **CanvasLayer**.



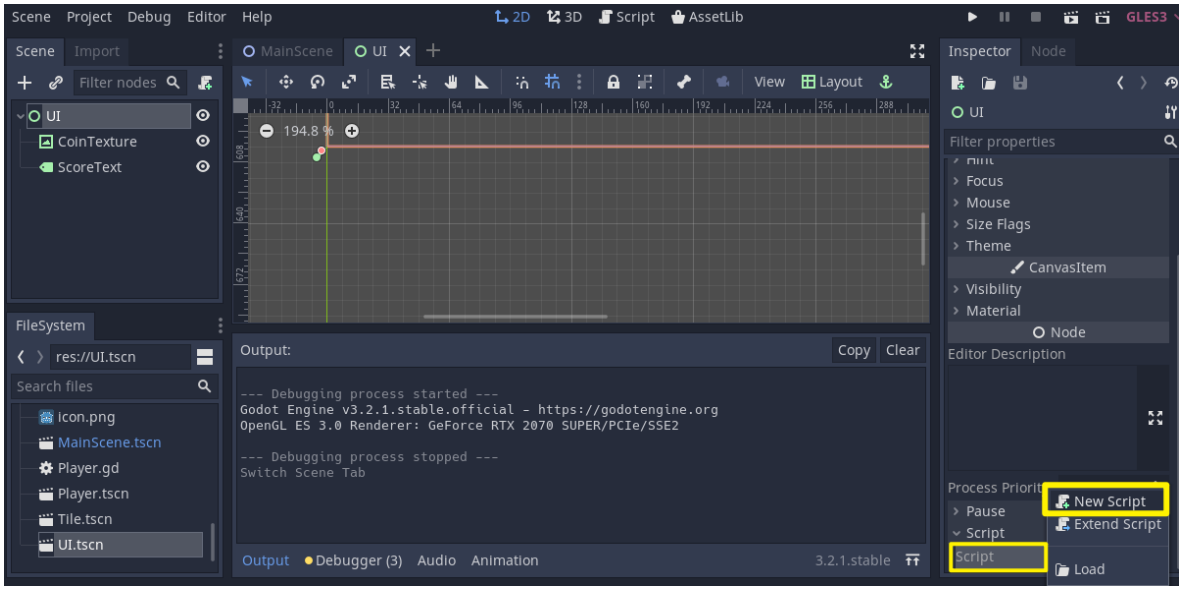


Now save the scene and hit **Play**. You will see that the UI follows along with the camera.



In this lesson, we are going to **create a script for UI**, so we can update our text when our score changes.

Open up the UI scene, select the root node, and create a **New Script**.



First up, we need to create a variable to keep track of our score text node.

```
extends Control

onready var scoreText = get_node("ScoreText")
```

The **onready** keyword is used here because the variable should be found once this node is initialized.

In the **_ready** function, we will set this text to be initially 0, since that should be the initial value of our score.

```
extends Control

onready var scoreText = get_node("ScoreText")

func _ready():

 scoreText.text = "0"
```

Now, whenever our score changes (i.e. get a coin), we are going to be calling the set_score_text function.

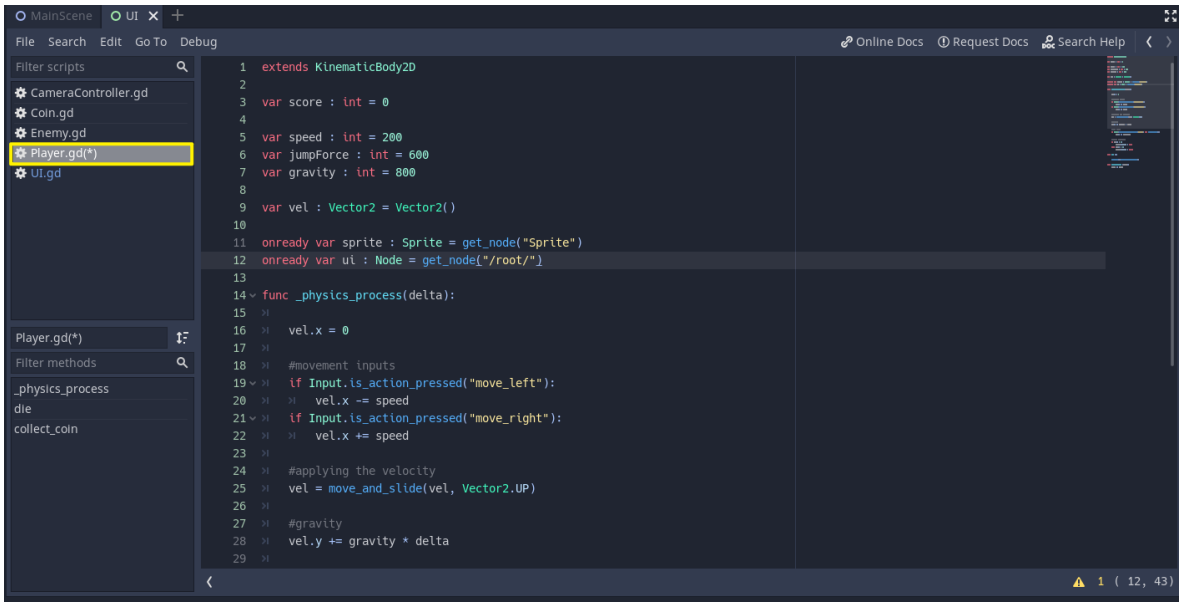
This function takes in an integer value (score) and sets the **scoreText.text** equal to the value (score).

```
extends Control
```

```
onready var scoreText = get_node("ScoreText")

func _ready():

 scoreText.text = "0"

func set_score_text (score):

 scoreText.text = str(score)
```

Note that **str() function** is used here to convert the integer value to a string.

Let's **save** the script, and move on to our Player script. (Player.gd)



Next, we will need to get a reference to the UI node.

At the top of the script where variables are defined, we need to create an **onready** variable of type **Node**.

We can then assign the default node path to the UI, using **get_node("path/")**.

```
onready var ui : Node = get_node("/root/MainScene/CanvasLayer/UI")
```

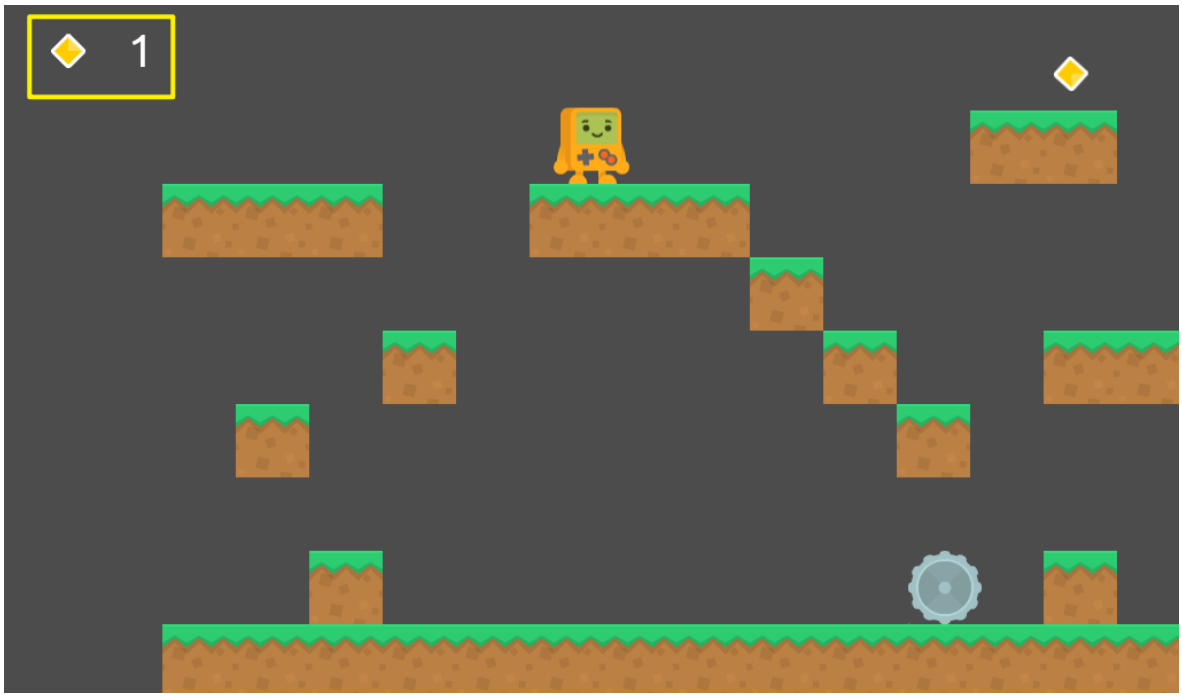
Then, down here where we collect the coin, we need to call the **set_score_text** function in our UI script and send over our score value.

```
func collect_coin (value):
 score += value
 ui.set_score_text(score)
```

Save the script and hit **Play**.

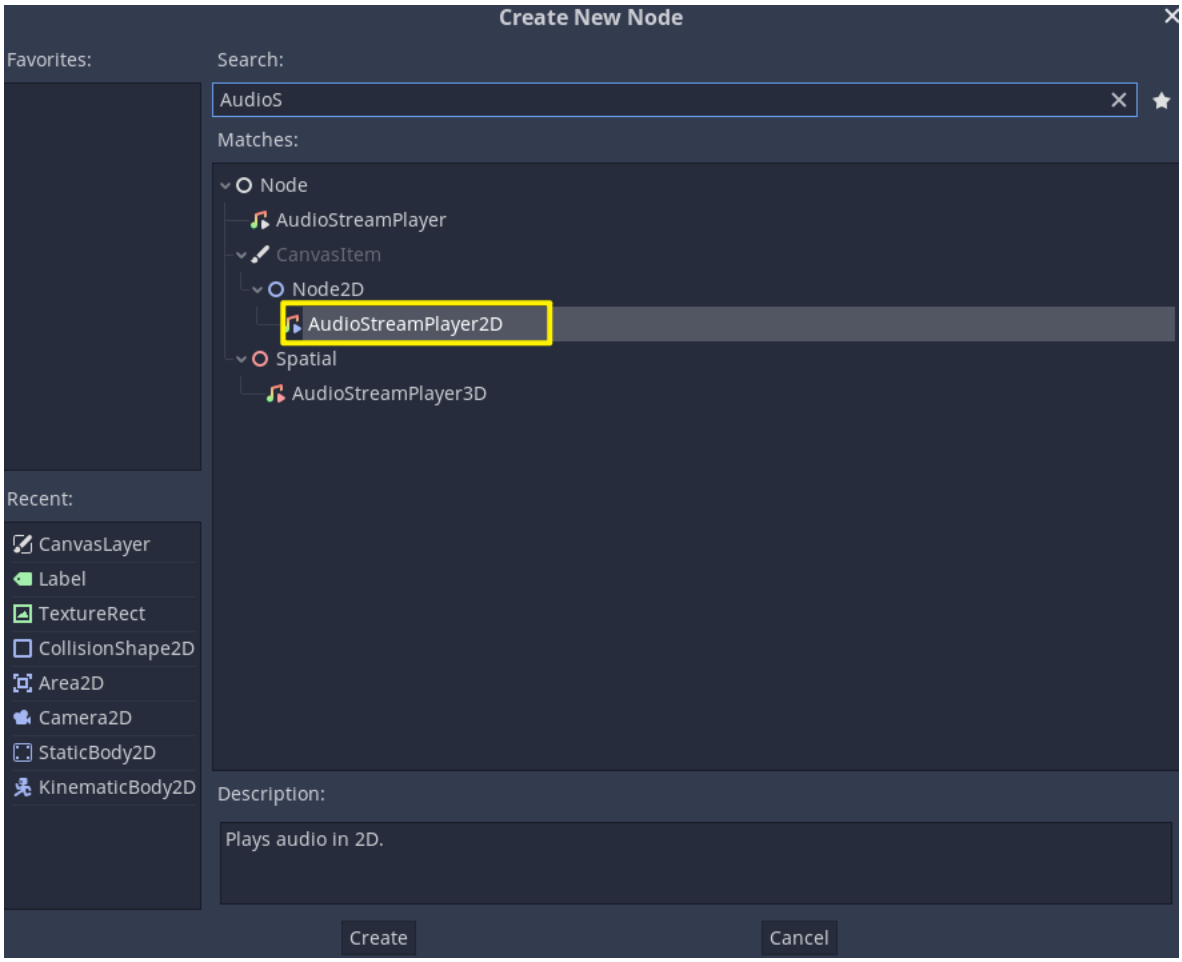
```
extends KinematicBody2D

var score : int = 0

var speed : int = 200
var jumpForce : int = 600
var gravity : int = 800

var vel : Vector2 = Vector2()

onready var sprite : Sprite = get_node("Sprite")
onready var ui : Node = get_node("/root/MainScene/CanvasLayer/UI")

func _physics_process(delta):

 vel.x = 0

 #movement inputs
 if Input.is_action_pressed("move_left"):
  vel.x -= speed
 if Input.is_action_pressed("move_right"):
  vel.x += speed

 #applying the velocity
 vel = move_and_slide(vel, Vector2.UP)

 #gravity
 vel.y += gravity * delta

 #jump input
 if Input.is_action_just_pressed("jump") and is_on_floor():
  vel.y -= jumpForce

 #sprite direction
 if vel.x < 0:
  sprite.flip_h = true
 elif vel.x > 0:
  sprite.flip_h = false

func die ():

 get_tree().reload_current_scene()

func collect_coin (value):
 score += value
 ui.set_score_text(score)
```

We should see that our score text is 0 at the start, and if we collect a coin, it goes up to 1.



In this lesson, we are going to be setting it up so that we can play audio clips through Godot.

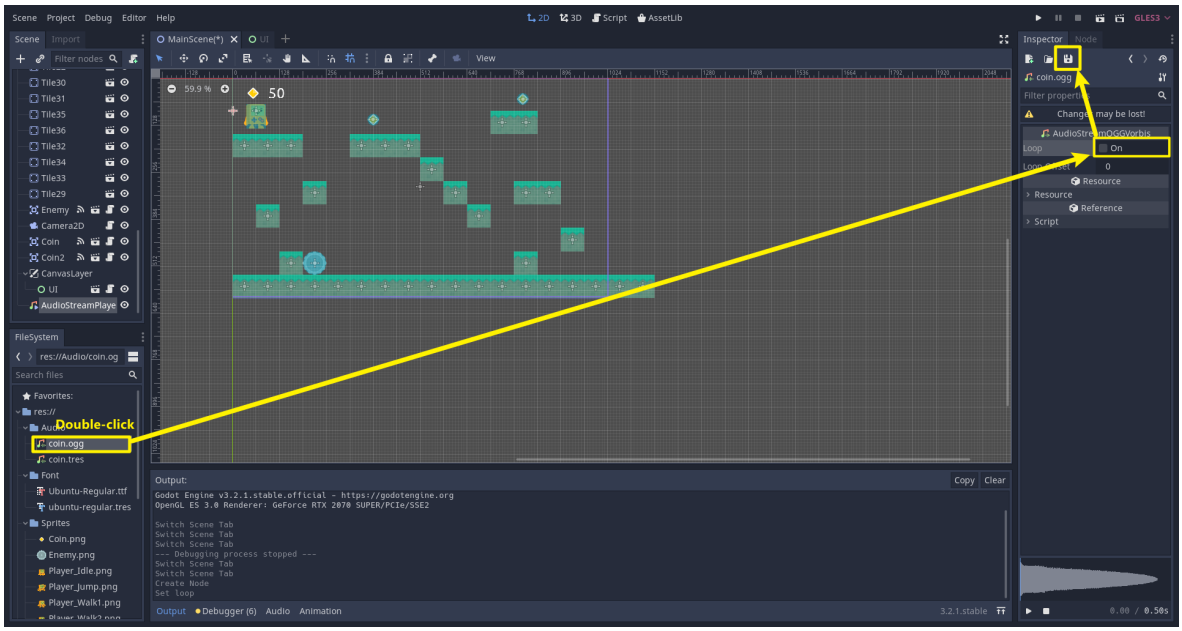
First, add a new node called an **AudioStreamPlayer2D**.



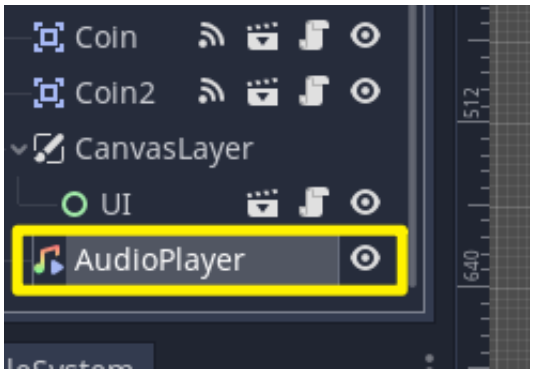
We want to now go ahead and convert that **AudioClip** that we brought in with us at the start of the course.

Go to FileSystem, open up Audio folder, and double-click on **coin.ogg**.

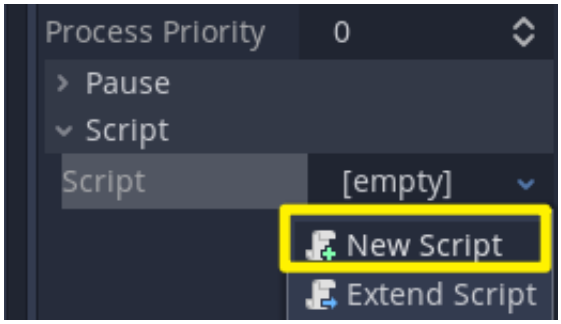
Disable the 'Loop' property in the Inspector and save it as "**coin.tres**".



Rename the **AudioStreamPlayer** to 'AudioPlayer'.



Create a new script called AudioPlayer.gd.

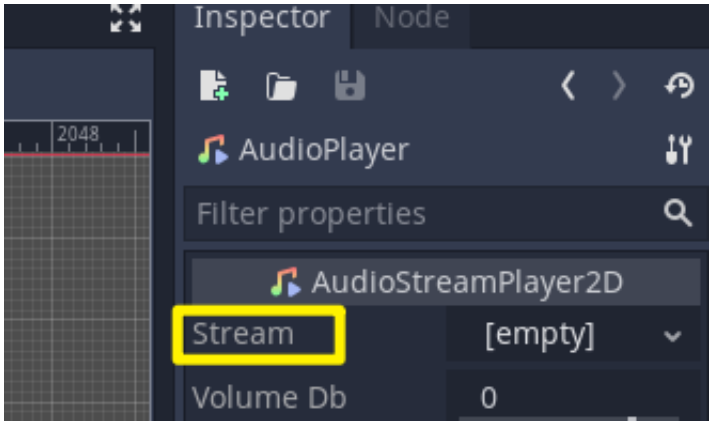


First of all, we need to create a variable for our sound effect. We will then be using **preload()**, which finds a resource file in our FileSystem and assign it to the variable.

Then, let's create a function that plays the coin pick-up sound effect.

```
extends AudioStreamPlayer2D

var coinSFX = preload("res://Audio/coin.tres")

func play_coin_sfx ():

 stream = coinSFX
 play()
```

We can simply assign 'coinSFX' to **stream**, which is the property we can assign in the inspector. This controls what sound should be currently playing.



Let's **save** the script and move on to our **Player** Script.

We're going to create a new variable that stores this **audioPlayer**.

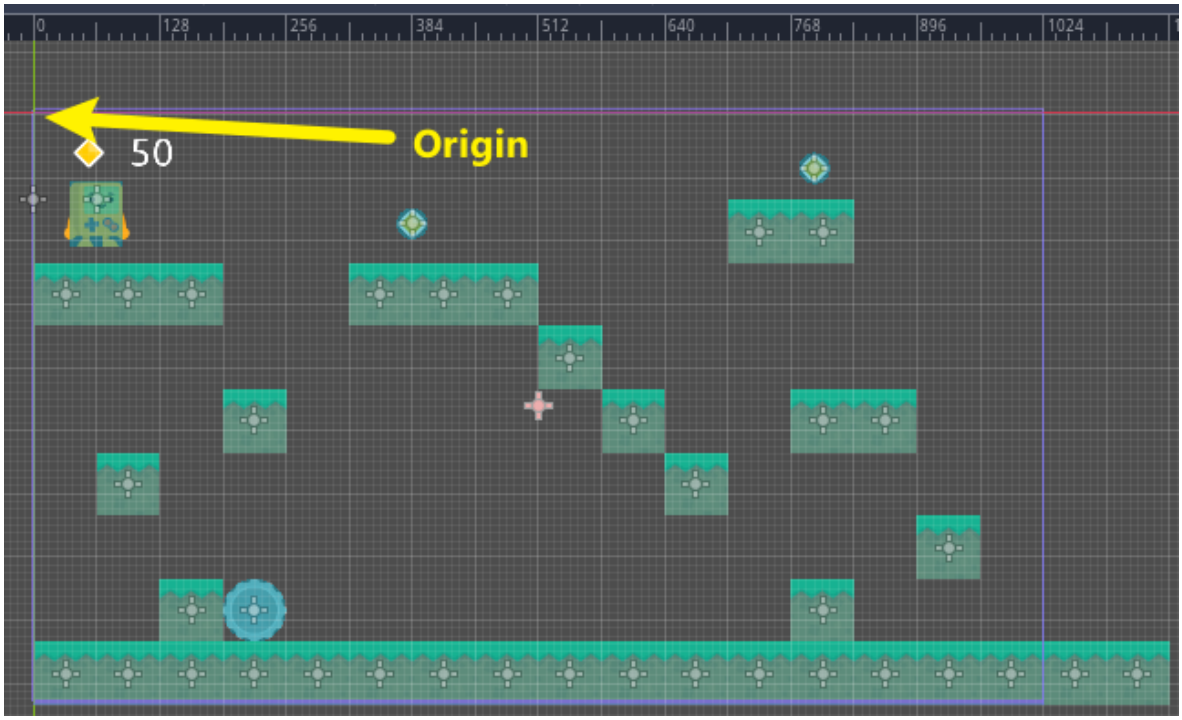
```
onready var audioPlayer : Node = get_node("/root/MainScene/AudioPlayer")
```

Then, we can make this **audioPlayer** play a sound clip whenever we collect a coin.

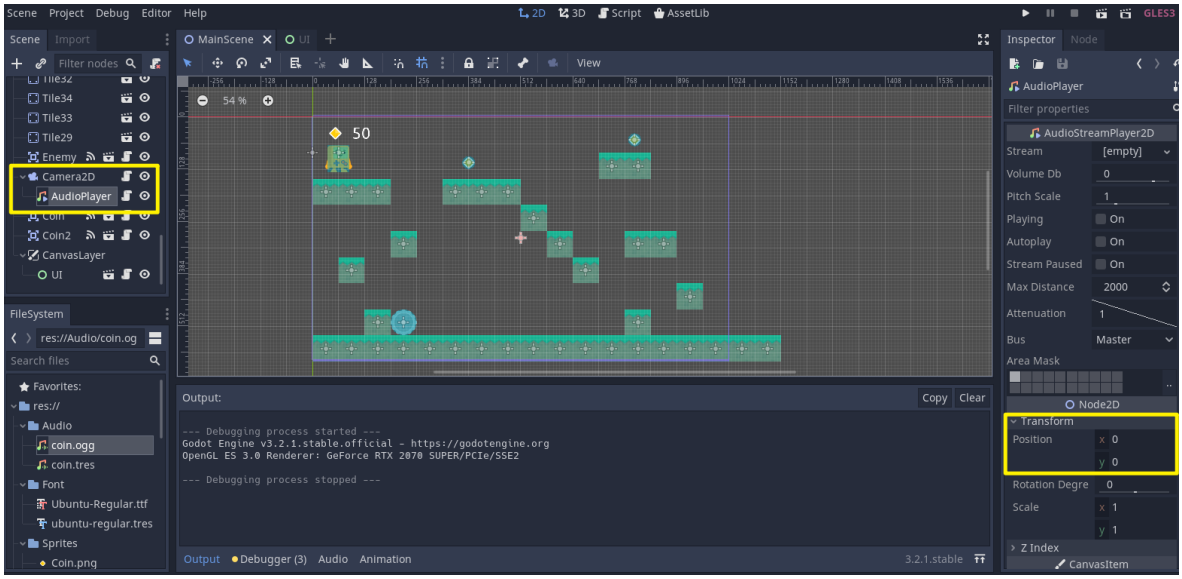
```
func collect_coin (value):
 score += value
 ui.set_score_text(score)
 audioPlayer.play_coin_sfx()
```


Save it and press **Play**. We should now hear the sound effect (beep) whenever we collect a coin.

Currently, the position of our **AudioPlayer** is at the origin, so the audio sounds a bit to our left ear.



We can fix this by making the audioPlayer a **child** of **Camera2D** and setting the **position** to be **(0,0)** in the inspector.



Lastly, we need to edit our script and correct the path to “/root/MainScene/**Camera2D**/AudioPlayer”.

```
onready var audioPlayer : Node = get_node("/root/MainScene/Camera2D/AudioPlayer")
```

