Godot Classroom

Challenge 1

What will we be making?

For our first Challenge we'll be making the old arcade favourite Pong. Pong is a fairly simple game, however it will allow us to try out a few of Godot's most important features.

All assets required will be provided, this is purely intended to be an editor based project.

All of the steps you will need to successfully create and finish this project have been included.

What will we be covering?

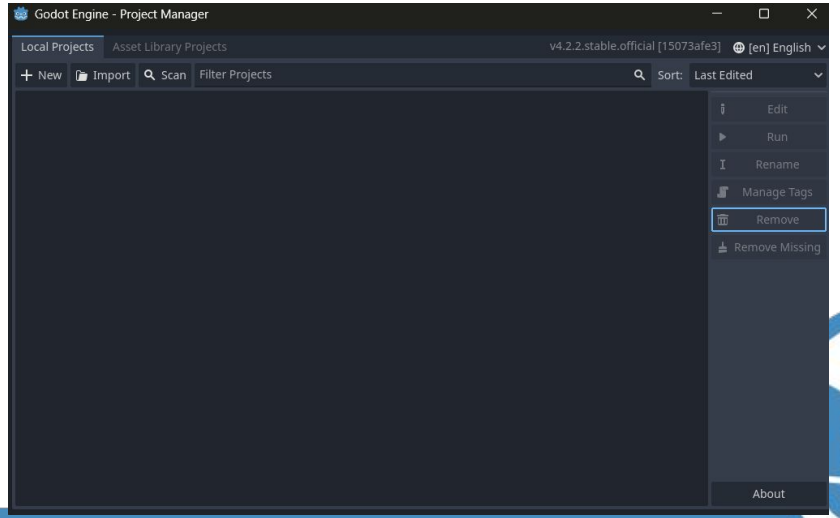
This Challenge will take you through:

- Organising your folder structure
- Creating scenes
- 2 player keyboard input
- Making scenes move
- Keeping track of scores
- Clickable buttons
- Connecting signals through code
- Creating a global script
- Switching between scenes
- A simple end game screen with a quit and play again button

Creating the Project.

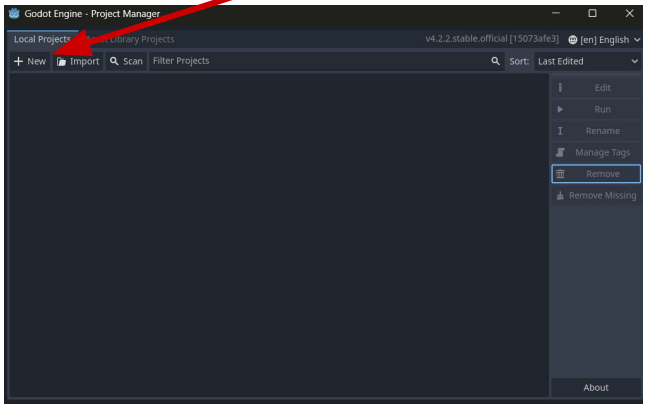
If you haven't already, download Godot. This project was created using Godot 4.2.2 however this will also work for other versions of Godot, some small changes may be required.

When you open Godot you will see this window.

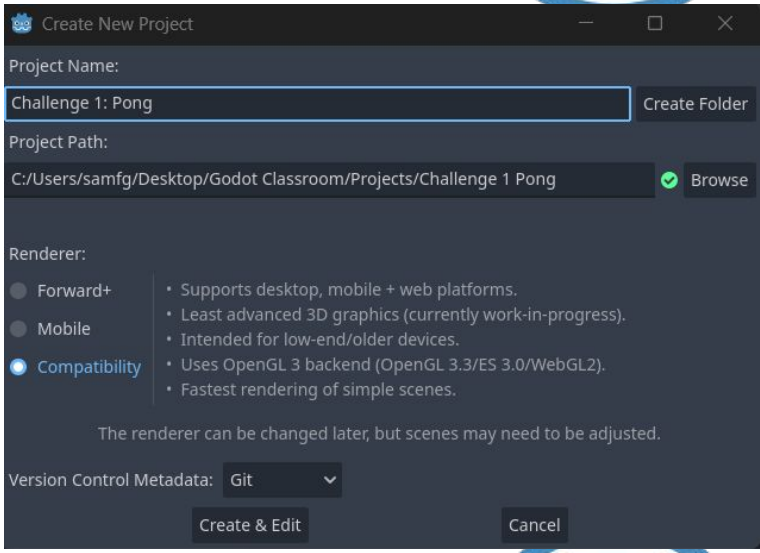


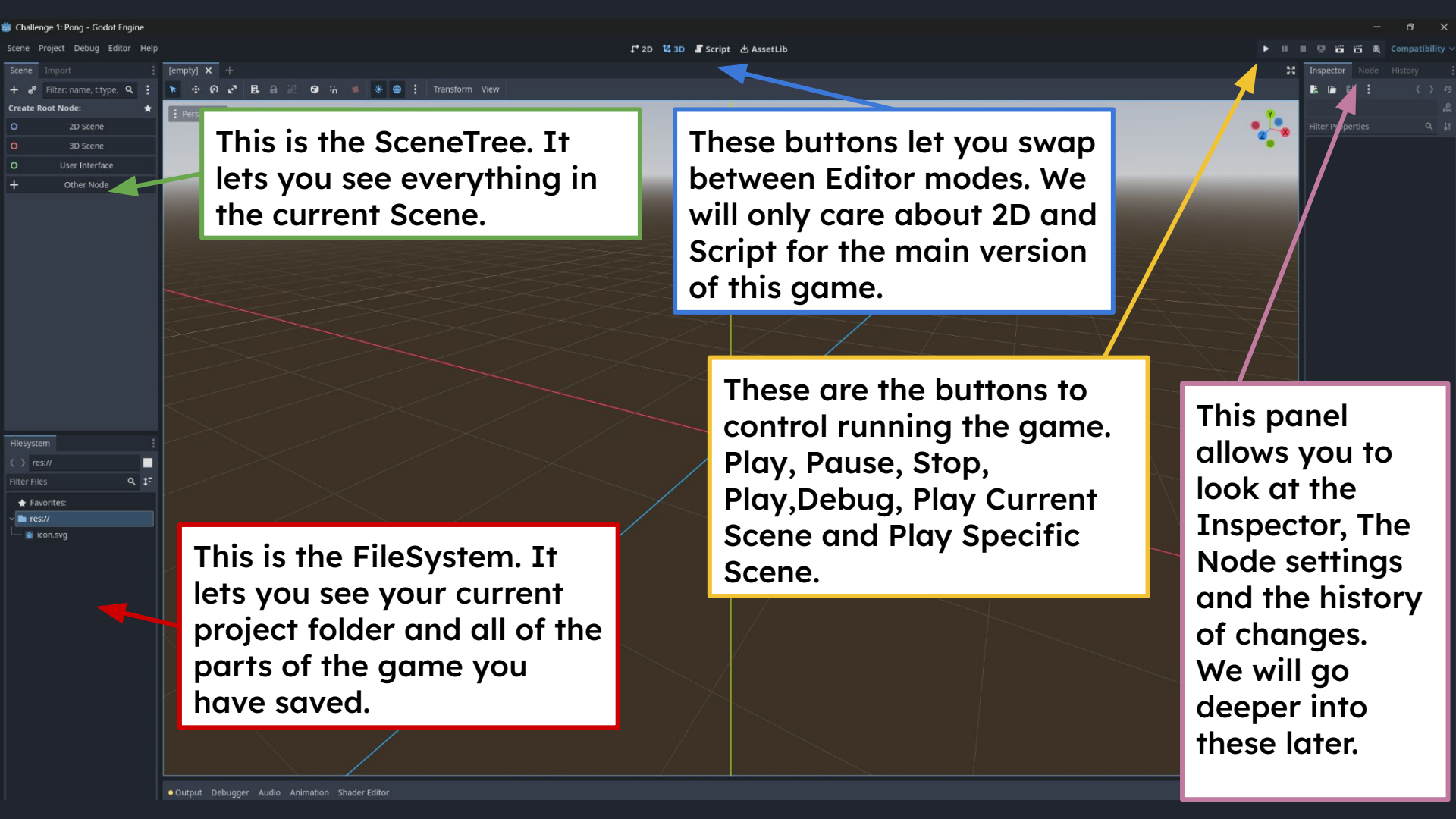
Creating the Project.

Step 1:
Click on “+ New”



Step 2:
Fill in the project name and make a folder for the project.
Choose **Compatibility** mode.




This is the SceneTree. It lets you see everything in the current Scene.
These buttons let you swap between Editor modes. We will only care about 2D and Script for the main version of this game.
These are the buttons to control running the game. Play, Pause, Stop, Play,Debug, Play Current Scene and Play Specific Scene.
This panel allows you to look at the Inspector, The Node settings and the history of changes. We will go deeper into these later.
This is the FileSystem. It lets you see your current project folder and all of the parts of the game you have saved.

File Organisation

With Godot it's important to keep your file structure organised so you can find things.

This is important because you'll be using file names or

needing to know the filepath to different assets like art or sounds. It’s important to note Godot really cares about capital letters and spelling!

File Organisation

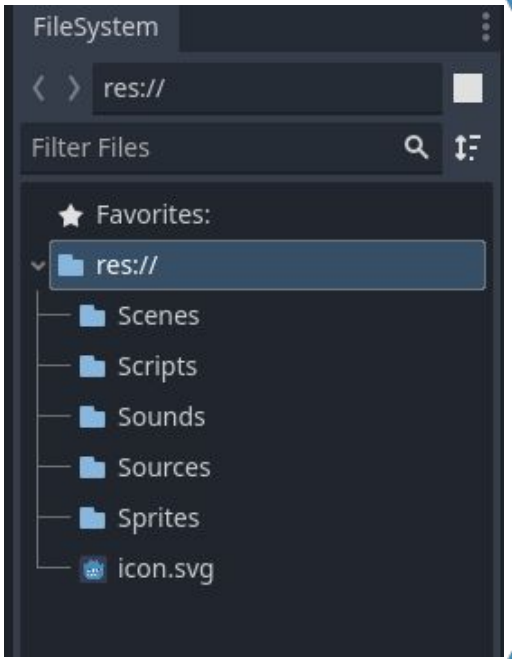
There are 5 main categories of things we care about. The 5 S's

- **Scenes** – Godot "objects" for every piece of our game we'll need to make a scene. The Ball will be its own scene, the score user interface will be its own scene.
- **Scripts** – These are text files that contain all the scripting and logic for the game to run.
- **Sounds** – Beep boop sounds, you get this one.
- **Sources** - The raw source files for assets you create, we'll keep these with the project so we don't get confused. You can keep these elsewhere but I find it useful to have a folder for this especially in small projects.
- **Sprites** – Visual assets either single images, tilesets or animation spritesheets.

File Organisation

Let’s make a folder for each one of the 5s topics.

To do so, right click in the FileSystem panel and choose “Create Folder” then you can name them. Make sure none of the new folders are inside each other. It should look exactly like the picture.



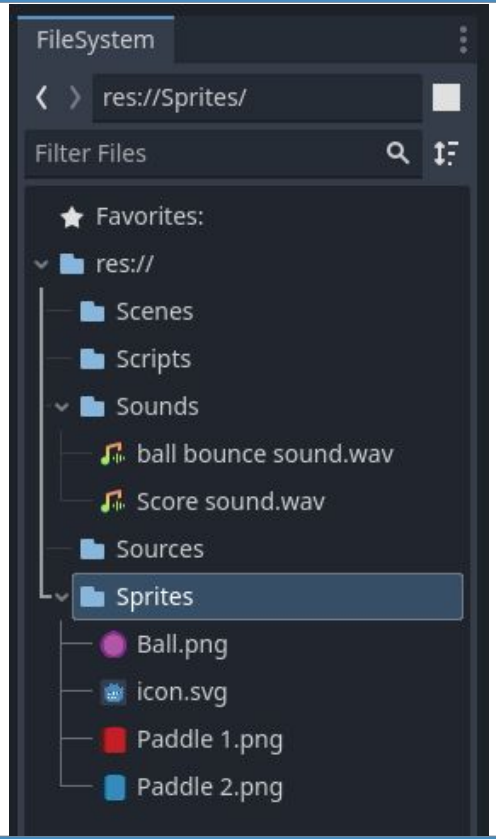
Adding the Assets

Copy the Sprites and Sounds provided in the challenge folder into the appropriate folders.

Sprites: Paddle1, Paddle2, Ball

(You can also move the icon.svg in)

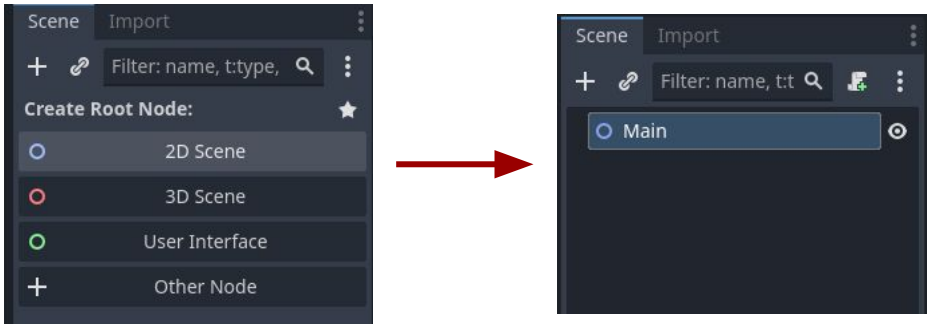
Sounds: Ball Bounce, Score Sound



Creating the Main Scene

The Main Scene is where all the action is going to happen. Think of it like the playing field of the game. In here we decide where all of the objects will go on the screen and which nodes need to be included.

In the SceneTree, click on 2D Scene. It'll make a new node, double click this to rename it "Main"



The Paddle

We will next create the Paddle Scene. Let's clarify the things we want it to be able to do:

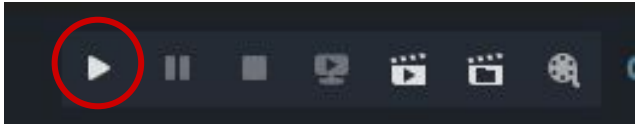
1. Be visible on the screen, using the assets we saved.
2. Act as a barrier to collide with the ball.
3. Move when the appropriate player presses their movement keys.
4. Cannot leave the screen and float off into space.

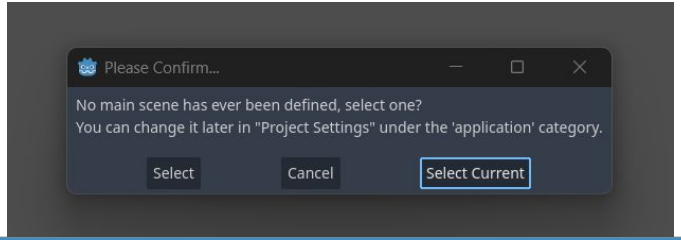
Selecting the Main Scene

We've created our Main scene, however, we need to tell Godot that this is the scene we want to run when the game starts. To do this we can just hit the play button.

A popup will appear, choose "Select Current", a save window will pop up. Save this scene in the Scenes folder as "main.tscn".

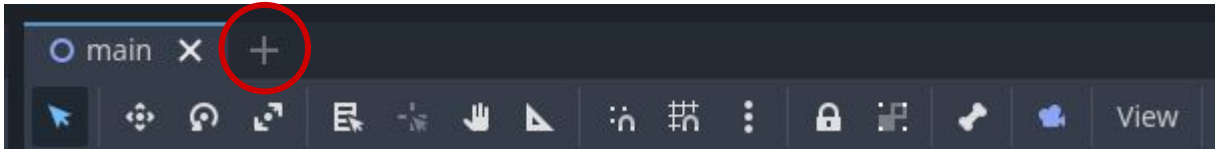
For now, the game will open but be blank. Close the game and let's add our Paddle's scene.



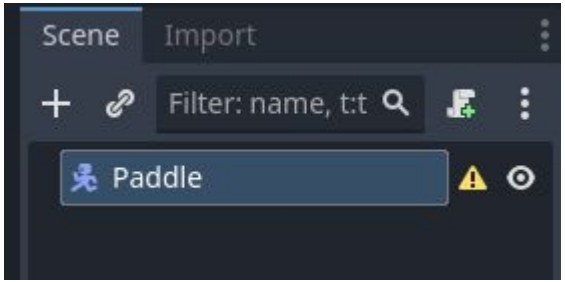


Creating The Paddle Scene

At the top of the main window, click the plus button to add a new Scene.



In the SceneTree, choose “Other Node” and add a CharacterBody2D from the list. Double click on this node to rename it “Paddle”

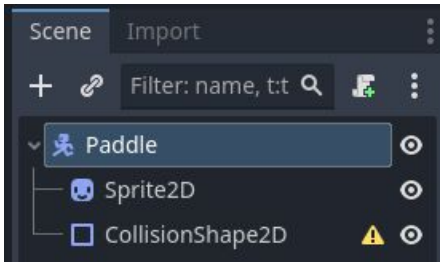


Structuring The Paddle Scene

Now we need to add some Children to the Paddle Node.

In the SceneTree, click on the + button and add:

- A Sprite2D
- A CollisionShape2D

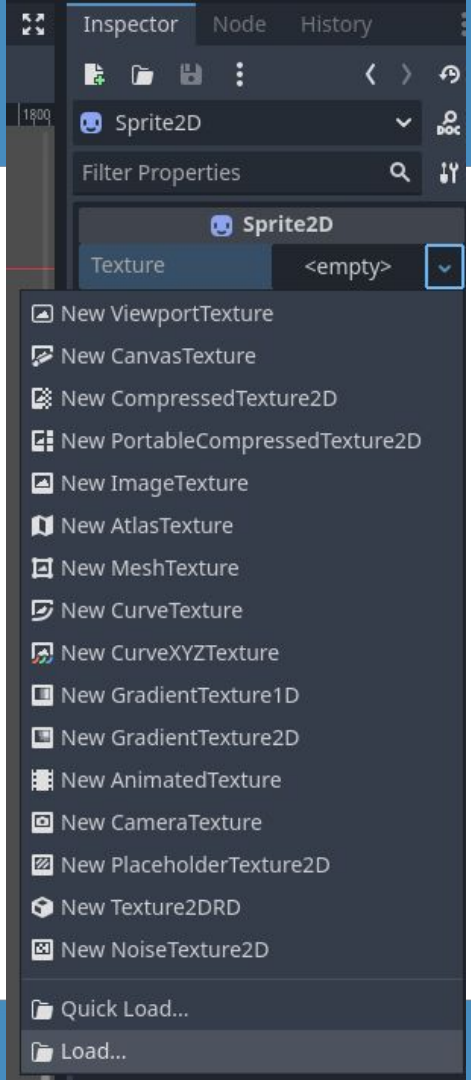


Setting up the Paddle Scene

We need to set up these new Children make them functional.

For the Sprite2D, click on it in the SceneTree and look at the inspector. It has a Texture that is currently empty. Click on the Empty box and choose Load, find the Sprites Folder and choose the red paddle.

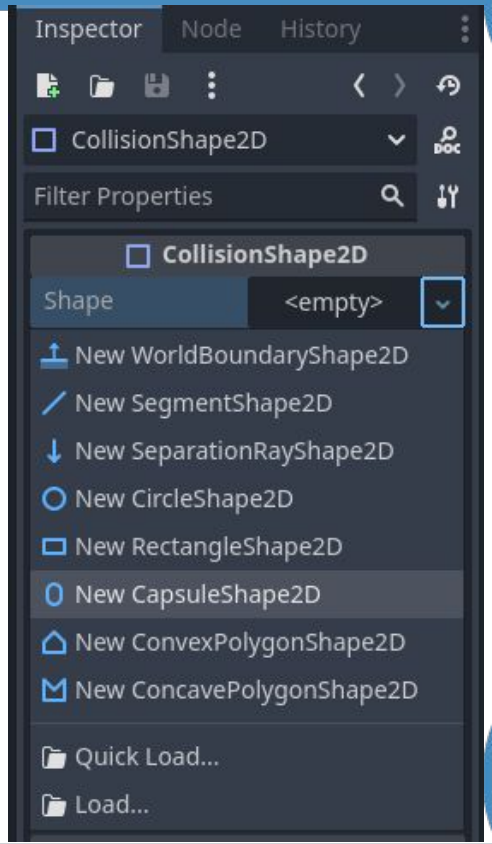
We should now see a red paddle in the main window.



Setting up the Paddle Scene

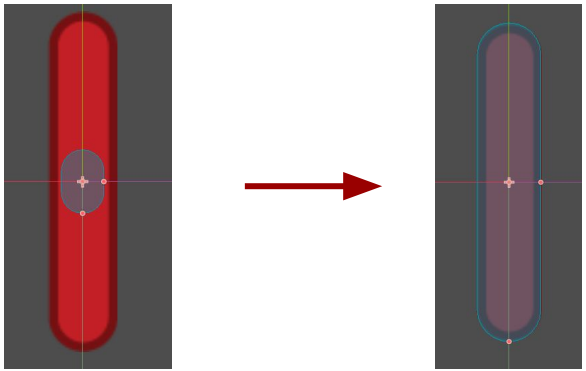
For the CollisionShape2D we need to define the area that things can bounce off of.

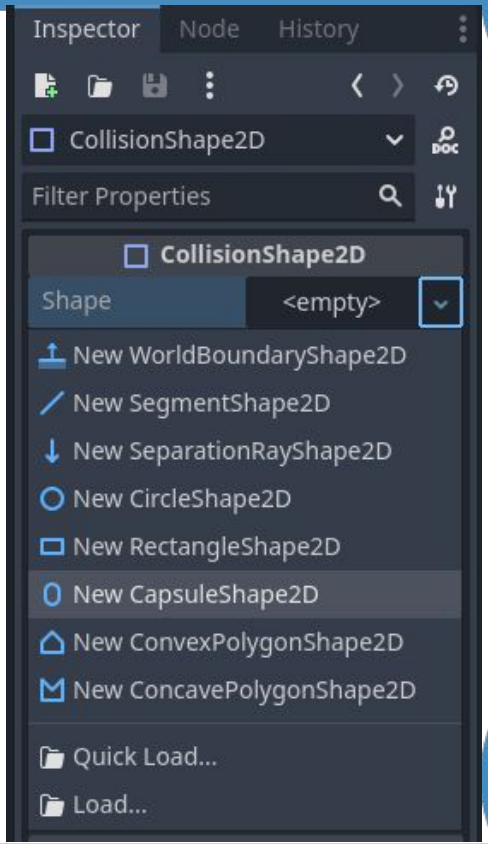
Click on the CollisionShape2D and in the Inspector you will see that it has an empty Shape field. Click on that empty field and add a new Capsule Shape.



Setting up the Paddle Scene

That capsule shape doesn't quite fit, grab those red circle handles and resize it to fit the paddle's sprite. It doesn't need to be pixel perfect.

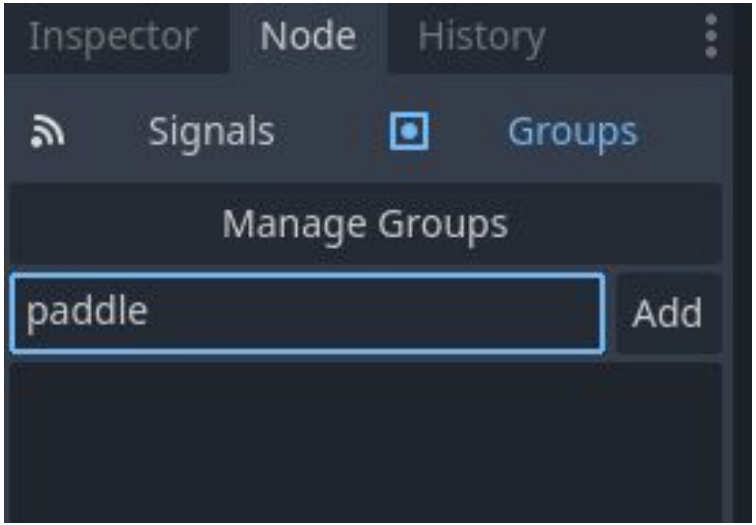




Setting up the Paddle Scene

To help us keep track of what is and isn't a paddle later, we will add it to a group.

Click on the Paddle Node and in the Node Panel (next to next to the Inspector) click on Groups and type in "paddle" to the textbox then press Add.



Saving the Paddle Scene

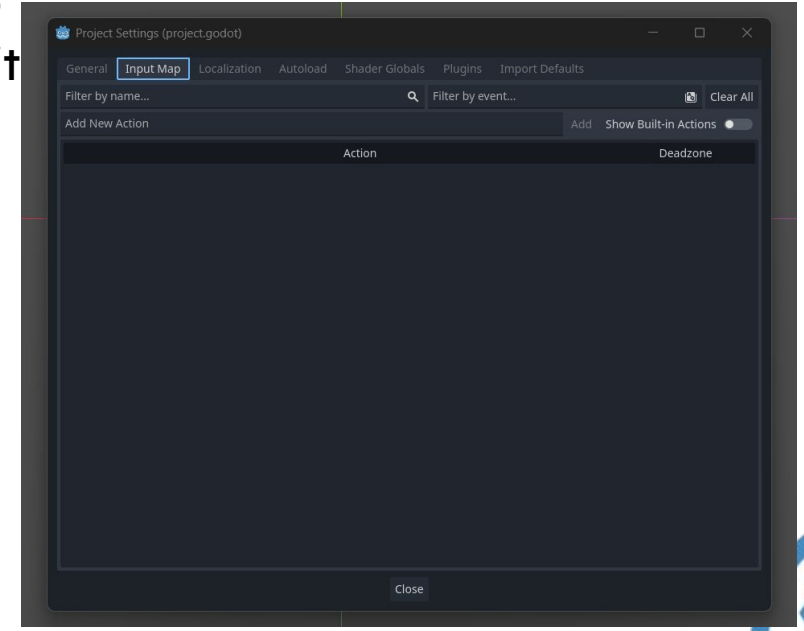
Once the Paddle looks good, go to the Scene option at the very top left of the screen and choose Scene > Save Scene.

Save this scene in the Scenes folder as “paddle.tscn”

Choosing our controls

We've got a paddle, now we need to decide how we are going to control it Both players will be playing with the same keyboard, there isn't going to be any sort of online multiplayer in this game.

At the top left of the window choose Project > Project Settings > Input Map

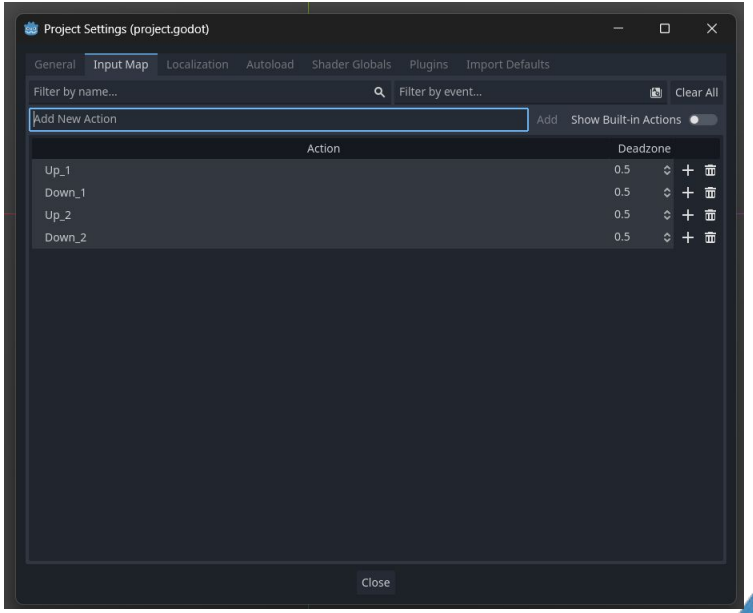


Choosing our controls

This window lets us add all the different input keys, controllers, etc we want in our project. You can even assign multiple keys to one action!

In the Add New Action field, add:

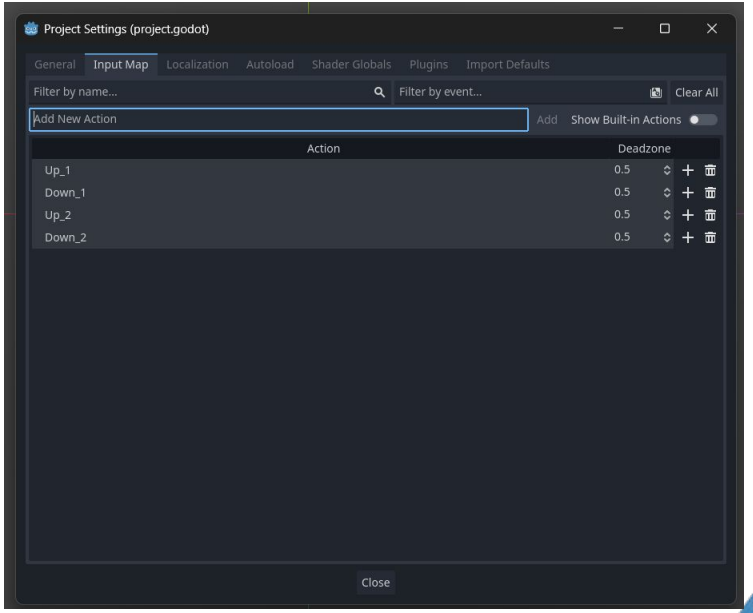
- “Up_1”
- “Down_1”
- “Up_2”
- “Down_2”



Choosing our controls

Click the Plus at the end of each control to open the key detection. Once the window opens you can just press the key you want to tie to that action. We'll set them up like this:

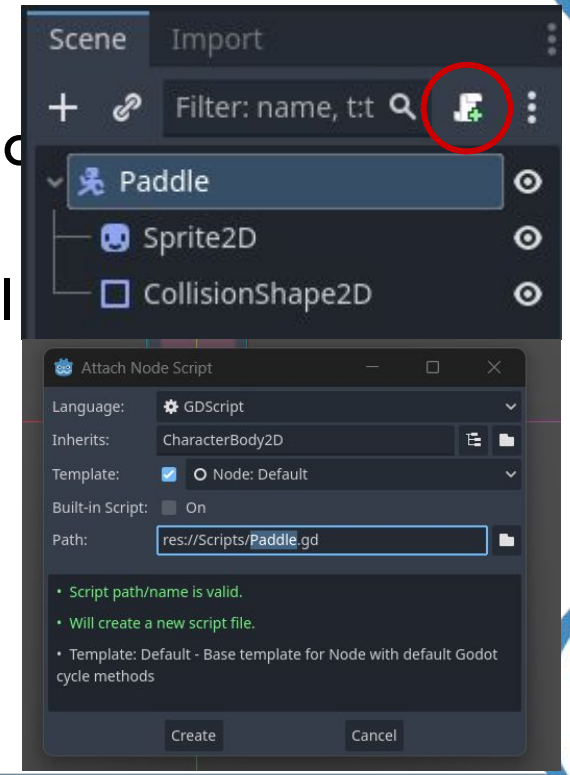
Action	Key
Up_1	W
Down_1	S
Up_2	Up Arrow
Down_2	Down Arrow



Making the Paddle Move

We need to add some code to our paddle scene to make it move. To do so click on the Paddle node and choose the Add Script button. Clicking this will have a menu pop up.

In this pop up, change the Path to be inside the Scripts folder.



Making the Paddle Move

When writing our first Godot Script it's important to remember a few things:

- Spelling and capitalisation is VERY important, be careful.
- Variable names must not contain spaces.
- Make sure you read any errors, they usually tell you what is wrong!
- The greyed out text that comes after a # is a comment, this is for your understanding and labeling your code, the engine does not try to read it in as game logic.

With that in mind we can copy the script in. On the next few slides I'll highlight what's happening in each bit of code.

```gdscript
extends CharacterBody2D

#Variables
@export var paddleID: int =1
var speed: int =10
@onready var paddleSprite :Sprite2D = $Sprite2D

#When the paddle is loaded into the game, on the main Scene starting, this code will run.
func _ready():
    if(paddleID == 1):
        paddleSprite.texture = load("res://Sprites/Paddle 1.png")
    elif(paddleID == 2):
        paddleSprite.texture = load("res://Sprites/Paddle 2.png")
    else:
        printerr("The paddleID has been assigned to something other that 1 or 2, whoops!")

#The process function runs every frame the game runs, many times a second.
func _process(_delta):
    if(Input.is_action_pressed("Up_" + str(paddleID))):
        global_position.y -= speed

    if(Input.is_action_pressed("Down_" + str(paddleID))):
        global_position.y += speed

    global_position.y = clamp(global_position.y, 80, get_viewport_rect().size.y -80)
```

```gdscript
extends CharacterBody2D

#Variables
@export var paddleID: int =1
var speed: int =10
@onready var paddleSprite :Sprite2D = $Sprite2D

#When the paddle is loaded into the game, on the main Scene starting, this code will run.
func _ready():
    if(paddleID == 1):
        paddleSprite.texture = load("res://Sprites/Paddle 1.png")
    elif(paddleID == 2):
        paddleSprite.texture = load("res://Sprites/Paddle 2.png")
    else:
        printerr("The paddleID has been assigned to something other that 1 or 2, whoops!")

#The process function runs every frame the game runs, many times a second.
func _process(_delta):
    if(Input.is_action_pressed("Up_" + str(paddleID))):
        global_position.y -= speed

    if(Input.is_action_pressed("Down_" + str(paddleID))):
        global_position.y += speed

    global_position.y = clamp(global_position.y, 80, get_viewport_rect().size.y -80)
```

This tells Godot that this script is attached to a CharacterBody2D and that it should have access to thing's like a velocity and collision information.

```gdscript
extends CharacterBody2D

#Variables
@export var paddleID: int =1
var speed: int =10
@onready var paddleSprite :Sprite2D = $Sprite2D

#When the paddle is loaded into the game, on the ma
func _ready():
    if(paddleID == 1):
        paddleSprite.texture = load("res://Sprites/
    elif(paddleID == 2):
        paddleSprite.texture = load("res://Sprites/
    else:
        printerr("The paddleID has been assigned to

#The process function runs every frame the game runs, many times a second.
func _process(_delta):
    if(Input.is_action_pressed("Up_" + str(paddleID))):
        global_position.y -= speed

    if(Input.is_action_pressed("Down_" + str(paddleID))):
        global_position.y += speed

    global_position.y = clamp(global_position.y, 80, get_viewport_rect().size.y -80)
```

A Variable is a programming concept that allows us to store values and assign names to them.
The var keyword is used to tell Godot we are declaring a new variable. The bright green Int is the Data Type of a variable in this case PaddleID and Speed have Integer data types which hold Whole Numbers like 1,2,3,4. They cannot hold numbers with decimal points.

We are setting them to an initial value using the = operator. So paddleID holds 1, speed holds 10.

```
1  extends CharacterBody2D
2
3  #Variables
4  @export var paddleID: int =1
5  var speed: int =10
6  @onready var paddleSprite :Sprite2D = $Sprite2D
7
8  #When the paddle is loaded into the game, on the main Scene starting, this code will run.
9  func _ready():
10     if(paddleID == 1):
11         paddleSprite.texture = load("res://Sprites/Paddle 1.png")
12     elif(paddleID == 2):
13         paddleSprite.texture = load("res://Sprites/Paddle 2.png")
14     else:
15         printerr("The paddleID has been assigned to something other that 1 or 2, whoops!")
16
17
18
19
20
21
22
23
24
25
```

This is our first Function. A Function is denoted by the func keyword followed by a name(): Functions are snippets of code that can be run. Some functions are triggered by certain events like _ready() runs when the Paddle Node is added into a scene and is ready to act. This typically only runs once at the start of the game loading.

The next bit of code is really simple If the paddleID is set to 1 it will load the Paddle 1 texture onto the Paddle Sprite2D, if it's 2 then it loads the Paddle 2 texture. If paddleID is set to some other number by accident then we get a neat little error message informing us of the mistake.

```
1  extends CharacterBody2D
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17 #The process function runs every frame the game runs, many times a second.
18 func _process(_delta):
19     if(Input.is_action_pressed("Up_" + str(paddleID))):
20         global_position.y -= speed
21
22     if(Input.is_action_pressed("Down_" + str(paddleID))):
23         global_position.y += speed
24
25     global_position.y = clamp(global_position.y, 80, get_viewport_rect().size.y -80)
```

The _process() function runs every single frame, if a game is running at 60 FPS it runs 60 times a second.

We are using this to say If the player is pressing the up button then move the paddle's position up the screen.

If the player is pressing the down button then move the paddle's position down the screen.

The last line limits the paddle from going off of the screen, it "Clamps" the position between 80 units from the top of the screen and 80 units from the bottom of the screen.

(80 because that's half the length of the paddle. If we didn't add that in then it could move halfway off the screen before it stops.)

Testing out the Paddle

Once you've got all your code typed in and there are no obvious glowing red errors then it's time to test your hard work.

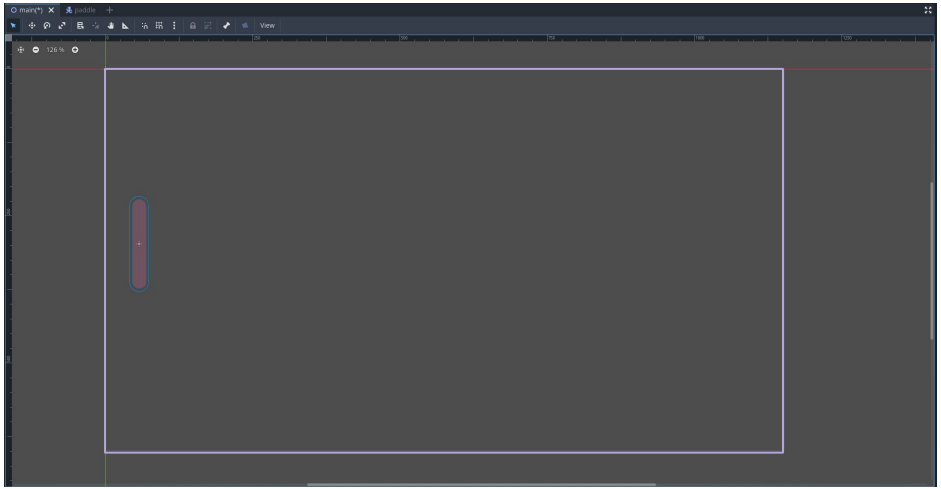
To do this save the scene again (Scenes > Save Scene)

Then navigate over to the Main Scene at the top of the screen.

Testing out the Paddle

In the Main Scene, switch into 2D mode.

You should see a blank screen with a purple box. From the FileSystem find the paddle.tscn file we saved and drag and drop it into the purple box. Something like image on the right.



I made my purple box a bit more obvious for the slide, yours won't be that bright.

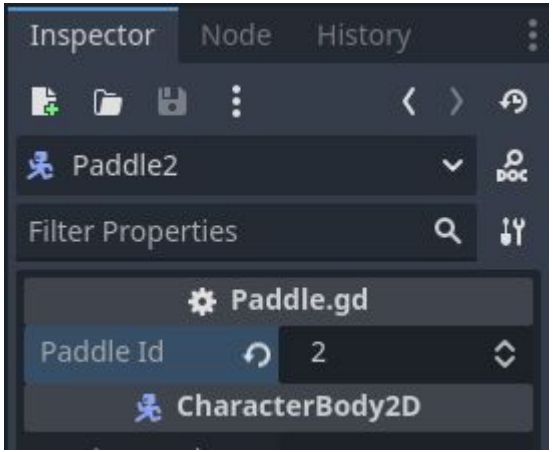
Testing out the Paddle

Hit the play button at the top right and the game should start. You'll be able to move the red paddle up and down with the W and S keys, it shouldn't be able to leave the play area.

Test that all works as expected, if not, revisit the code and make sure you've got it all correct and there's no errors.

Adding the second paddle

We need two paddles for pong! Repeat the process of dragging and dropping a Paddle from the FileSystem, put it on the right side. With Paddle2 selected, change it's Paddle ID in the inspector to be 2 instead of 1.



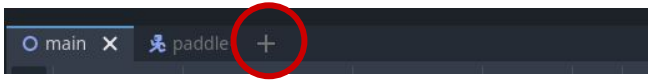
Run the game and test, it should move when you press the Up and Down arrows but not the W and S keys. The red paddle should still work.

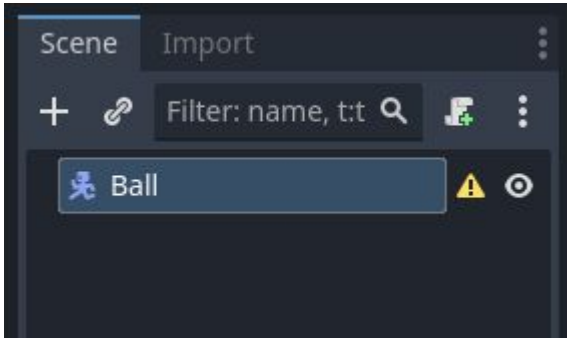
Creating the Ball Scene

Two paddles! Time for the next step, the Ball.

We need to create another new Scene so click on the + next to the Paddle Scene at the top.

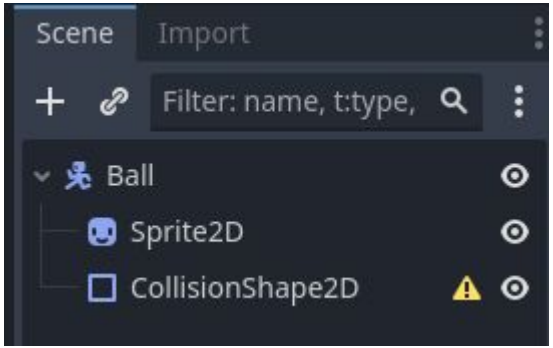
Choose “Other Node” and pick a CharacterBody2D again. Rename this to Ball.





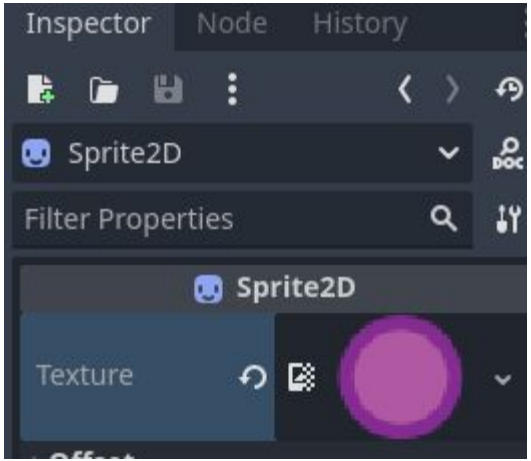
Structuring the Ball Scene

Next we need to add the Children. The Ball is simple, only a sprite and some collision. Add a Sprite2D and a CollisionShape2D



Setting up the Ball Scene

Click onto the Sprite2D and in the inspector, click on the empty texture and Load the Ball Sprite from the Sprites folder.

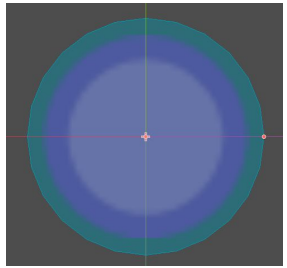


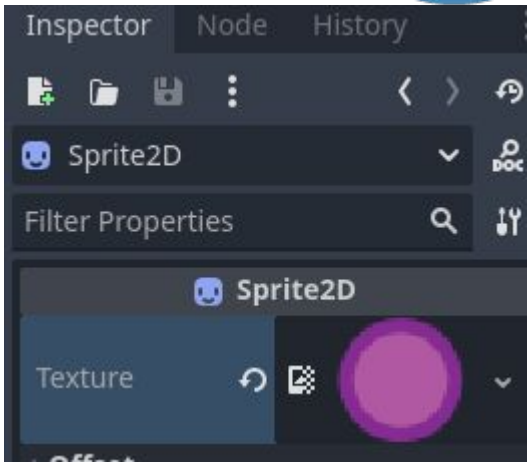
Setting up the Ball Scene

Click onto the CollisionShape2D and in the inspector, click on the empty Shape and add a new CircleShape2D.

Resize it using the orange handle to be just a little bigger than the ball itself.

Now go to Scene > Save Scene and Save it in the Scenes folder as ball.tscn

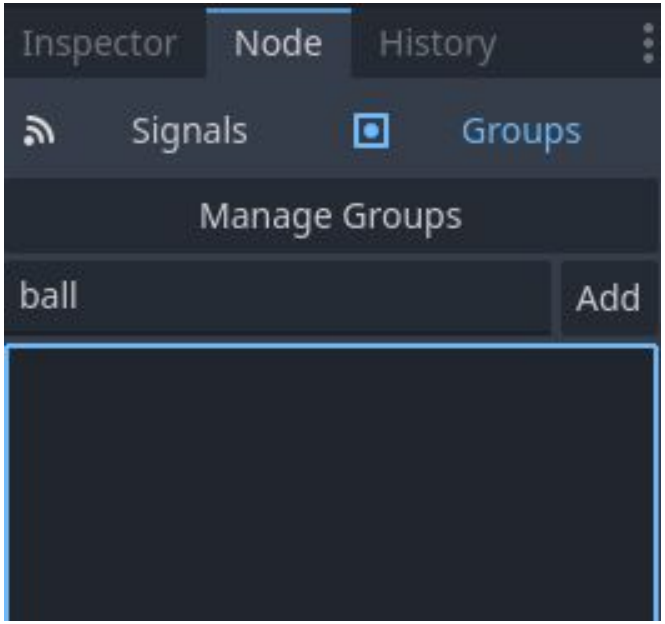




Putting the Ball in a Group

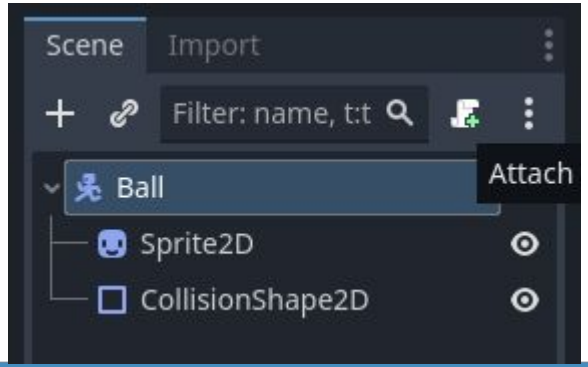
To help us keep track of what is and isn't a ball later, we will add it to a group.

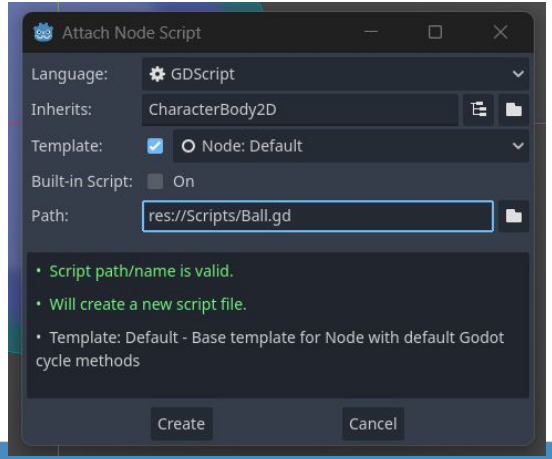
Click on the Ball Node and in the Node Panel (next to next to the Inspector) click on Groups and type in "ball" to the textbox then press Add.



Making the Ball Move

That was a quick setup! Now we need to add a script to the Ball CharacterBody2D to make it move and bounce off of the paddle.Click on the Attach Script button again and make a new script. Don’t forget to change the path to save it in the Scripts folder!





```gdscript
extends CharacterBody2D

#Variables
var speed:int = 500

#When the ball is loaded into the game, this code will run.
func _ready():
    velocity = Vector2.LEFT * speed

#The process function runs every frame the game runs, many times a second.
func _process(delta):
    var collision = move_and_collide(velocity*delta)
    if (collision):
        velocity = velocity.bounce(collision.get_normal())
```

This tells Godot that this script is attached to a CharacterBody2D and that it should have access to thing's like a velocity and collision information.

```gdscript
extends CharacterBody2D

#Variables
var speed:int = 500

#When the ball is loaded into the game, this code will run.
func _ready():
    velocity = Vector2.LEFT * speed

#The process function runs every frame the game runs, many times a second.
func _process(delta):
    var collision = move_and_collide(velocity*delta)
    if (collision):
        velocity = velocity.bounce(collision.get_normal())
```

```gdscript
extends CharacterBody2D

#Variables
var speed:int = 500

#When the ball is loaded into the game, this code will run.
func _ready():
    velocity = Vector2.LEFT * speed

#The process function runs every frame the game runs, many times a second.
func _process(delta):
    var collision = move_and_collide(velocity*delta)
    if (collision):
        velocity = velocity.bounce(collision.get_normal())
```

This sets the speed of the ball. Our ball will move at a constant speed all the time. Bigger number, faster ball.

```gdscript
extends CharacterBody2D

#Variables
var speed:int = 500

#When the ball is loaded into the game, this code will run.
func _ready():
    velocity = Vector2.LEFT * speed

#The process function runs every frame the game runs, many times a second.
func _process(delta):
    var collision = move_and_collide(velocity*delta)
    if (collision):
        velocity = velocity.bounce(collision.get_normal())
```

The _ready() function for the ball is really simple this time. We are setting the velocity of the ball (it's speed and direction of travel) to give it an initial kick towards the left at a speed of 500.

```
1   extends CharacterBody2D
2
3   #Variables
4   var speed:int = 500
5
6   #When the ball is loaded in
7   func _ready():
8       velocity = Vector2.LEFT
9
10  #The process function runs every frame the game runs, many times a second.
11  func _process(delta):
12      var collision = move_and_collide(velocity*delta)
13      if (collision):
14          velocity = velocity.bounce(collision.get_normal())
```

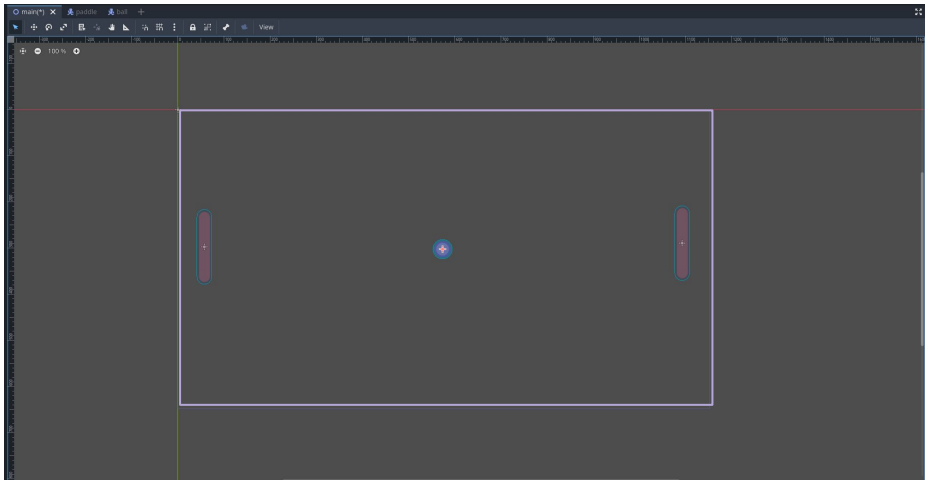
The _process() function for the ball tries to move the ball according to its current velocity. If it hits something that collision is recorded in the collision variable.

If the ball hits something like the paddle or a wall during the current frame then line 14 gets triggered and reflects the ball away from the surface it hits.

Testing out the Ball

Switch back to the Main Scene, switch into 2D mode.

From the FileSystem find the ball.tscn file we saved and drag and drop it into the middle of the play area. Line it up with the paddles. Something like image on the right.



I made my purple box a bit more obvious for the slide, yours won't be that bright.

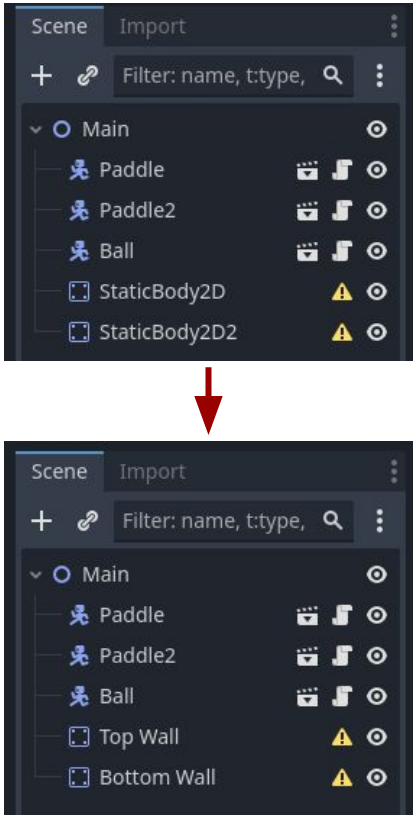
Hit play and give it a go, don't drop the ball! It'll fly away forever! We need walls.

Creating the walls

In the main scene, we need to add some simple walls so that the ball can bounce off of the top and bottom of the screen.

To do this, click on the Main node at the top and add two new StaticBody2D nodes.

Rename them.



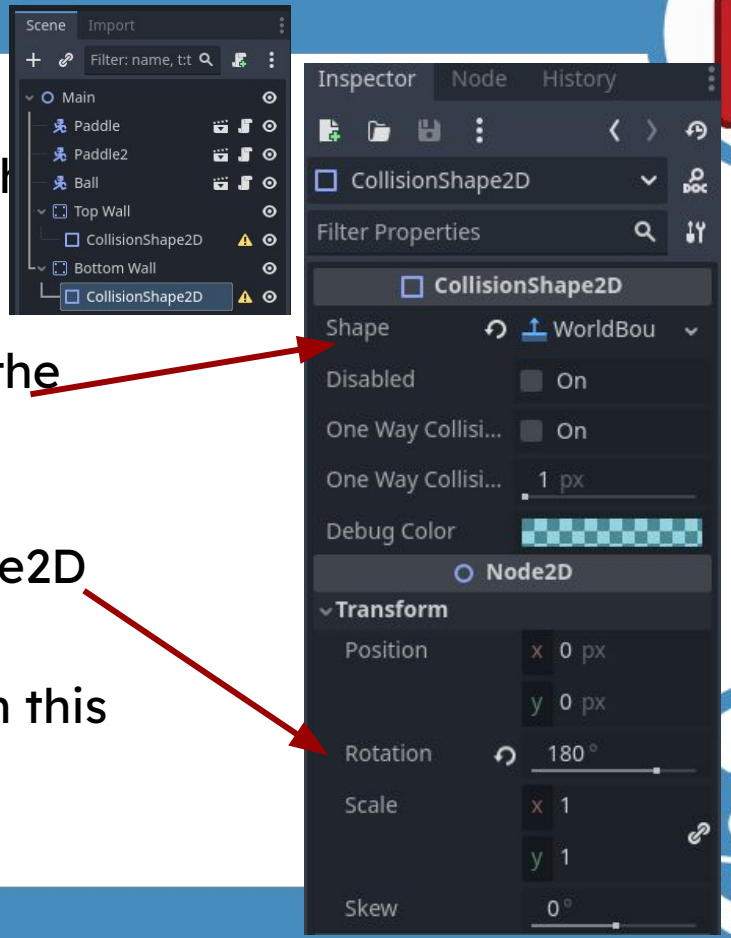
Creating the walls

Add a CollisionShape2D as a child for both Walls.

On the Top Wall CollisionShape2D, using the inspector panel add a new WorldBoundryShape2D for the shape.

Change the Rotation of the CollisionShape2D to be 180

Leave the Bottom Wall inspector alone on this step!



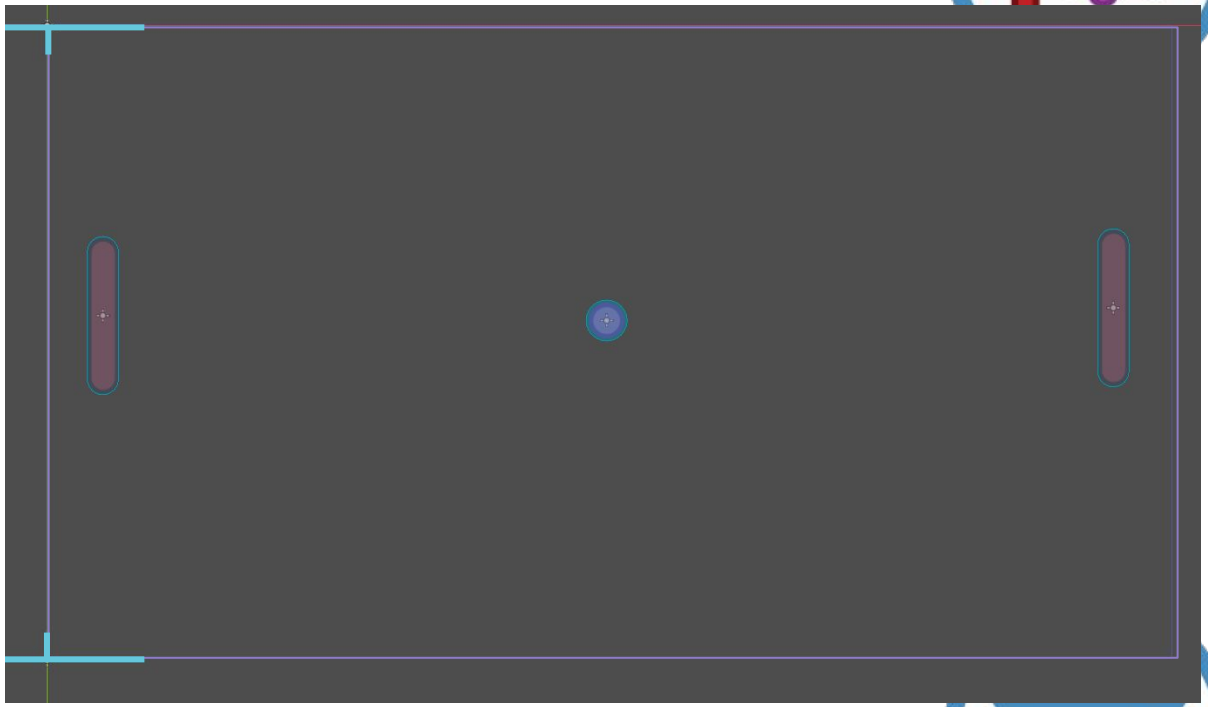
Creating the walls

On the Bottom Wall CollisionShape2D

The rotation should still be 0

Use the Move tool to drag the CollisionShape2D down to the bottom of the purple box.

Again for ease of seeing I will exaggerate the lines so you can check everything is in the right place and rotated correctly.



Testing the Walls

Hit play and see if you can get the ball to bounce off the top and bottom of the play area now.

Tip: Try nudging the ball with the rounded end of the paddle! It'll deflect the ball better. We'll give the ball a bit more variance in initial angle shortly to let it hit the wall more naturally.

Keeping track of the Score

Right now we don’t have any concept of a score. We’ll want to have some variables nestled away that keep track of how many points each player has.

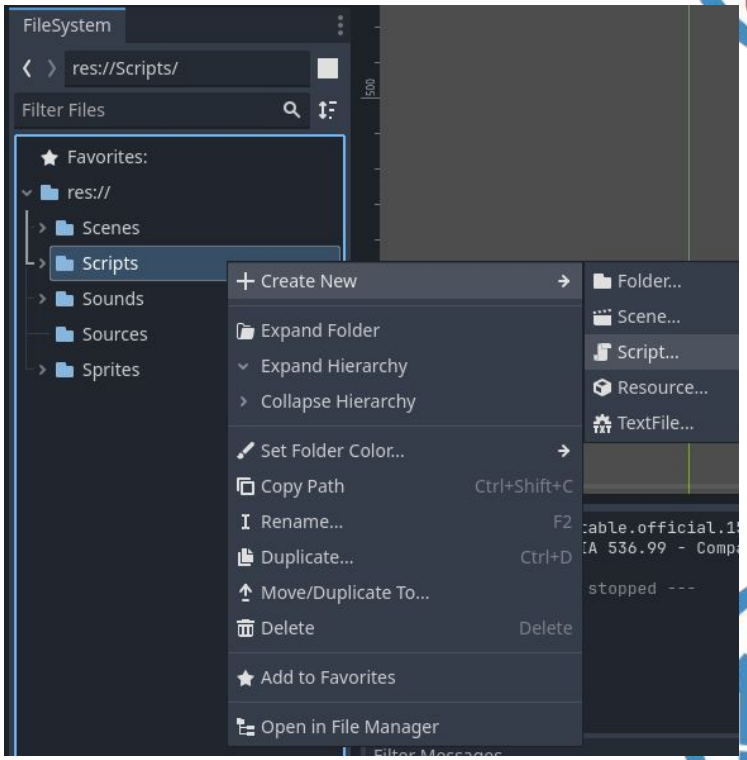
Eventually we will want to have easy access to these variables in any script, with any scene loaded.

A good way to do this is a Global Script or an Autoload.

Creating a Global script.

We'll make this script differently than before as it isn't attached to a node that we can see in the SceneTree.

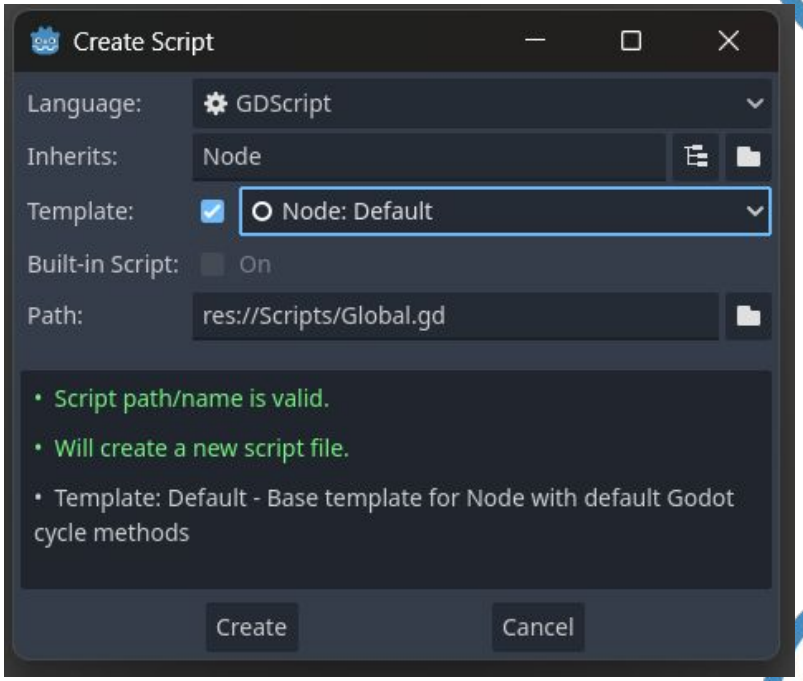
Right click the Scripts folder in the FileSystem and choose Create New > Script



Creating a Global script.

With the window that pops up, call the new script “Global”

Hit create to make the script then double click it in the FileSystem to edit it.



```
1   extends Node
2
3   #Variables
4   var P1_Score: int = 0
5   var P2_Score: int = 0
6
```

```
1  extends Node
2
3  #Variables
4  var P1_Score: int = 0
5  var P2_Score: int = 0
6
```

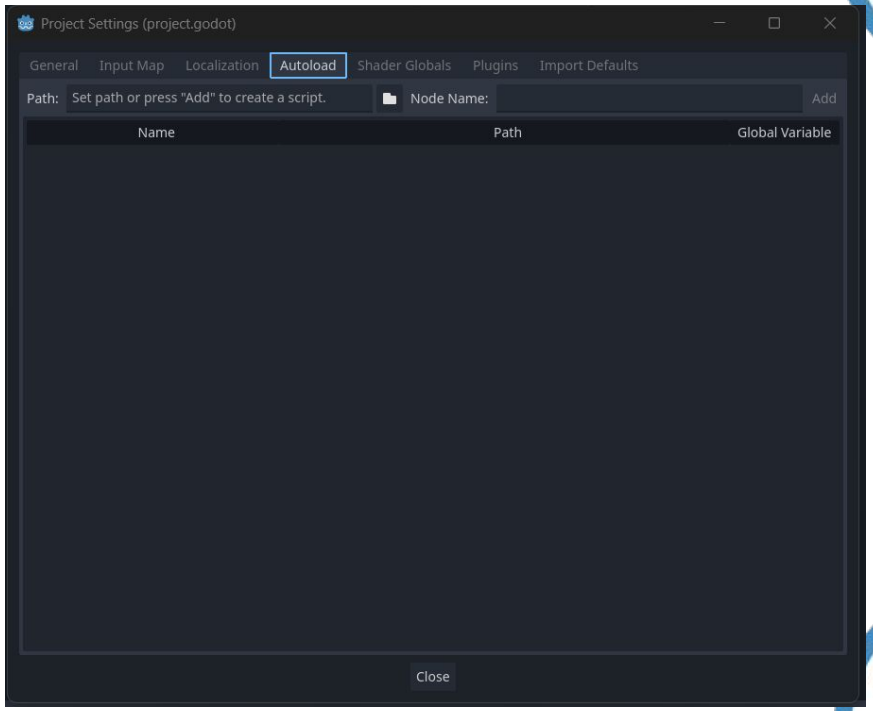
This script will belong to a blank Node that is essentially hidden and is constantly loaded.

All we need to do here is add in two variables. One to keep track of each score. They'll be integer data types which means they'll only count in whole numbers (0,1,2,3,...)

Activating the Autoload

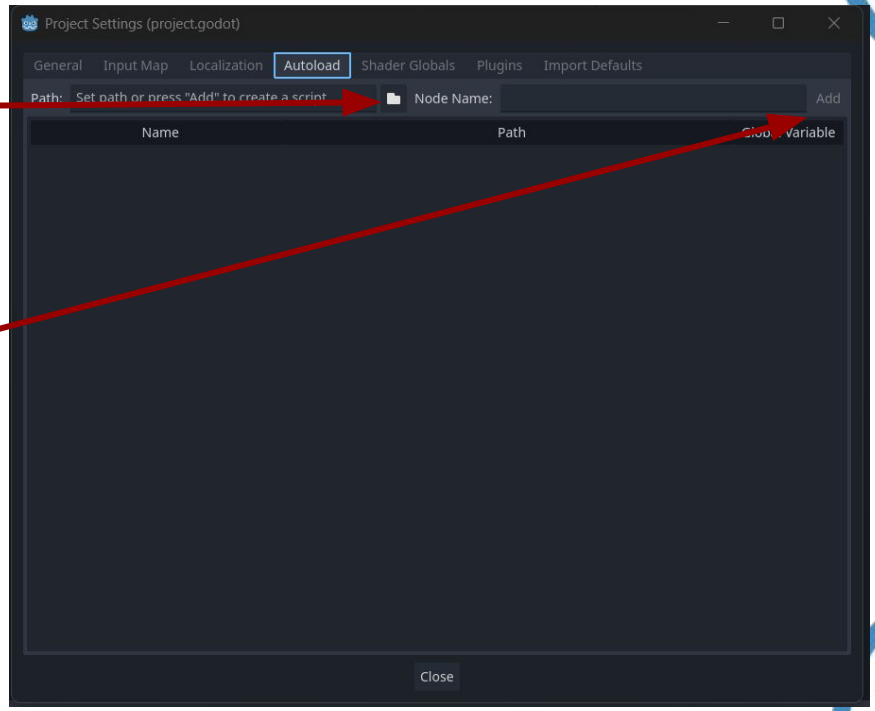
To tell Godot about the new script and ensure it loads it constantly we need to make it an Autoload.

At the top left open Project> Project Settings > Autoload



Activating the Autoload

Choose the folder icon and find our new Global script in the Scripts folder. Select it then choose Add at the right.



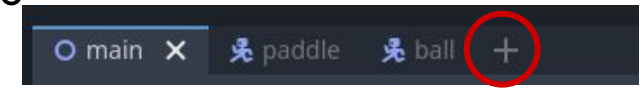
Next up we need Goals

We’ve got a paddle, balls, walls and a score system. Now we need a way to actually increase the scores.

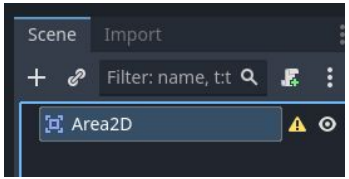
For that we will create two Goals, invisible detection zones that will increase our score and destroy the ball when it enters.

Creating the Goal Scene

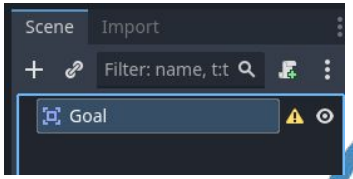
We need to create another new Scene so click on the + next to the Ball Scene at the top.



Choose “Other Node” and pick an Area2D. Rename this to Goal.



An Area2D is a trigger area, it can detect when characters or other Area2Ds enter or leave it’s area. It does not physically collide and bump into things, think of it like an automatic door you walk up to and it and it opens when you are detected but you don’t physically touch the trigger.



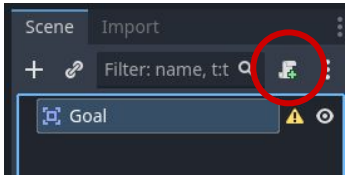
Save the scene using Scene > Save Scene as “goal.tscn”

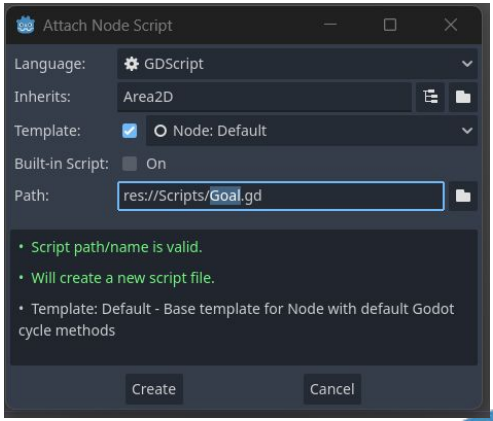
Adding a script to the Goal Scene

We'll handle the collision shapes in the main scene as both goals will have different rotations.

Add a script using the add script button.

Make sure you change the path so it goes into the Script folder and name it "Goal"





```gdscript
extends Area2D

#Variables
@export var GoalID : int = 0

func _ready():
	body_entered.connect(scored)
	
func scored(body):
	if(body.is_in_group("ball")):
		if(GoalID == 1):
			Global.P2_Score +=1
		elif(GoalID == 2):
			Global.P1_Score +=1

	body.queue_free()
```

```gdscript
extends Area2D

#Variables
@export var GoalID : int = 0

func _ready():
    body_entered.connect(scored)

func scored(body):
    if(body.is_in_group("ball")):
        if(GoalID == 1):
            Global.P2_Score +=1
        elif(GoalID == 2):
            Global.P1_Score +=1

    body.queue_free()
```

This tells Godot that this script is attached to a Area2D and that it should have access to thing's like body_entered signals.

```gdscript
extends Area2D

#Variables
@export var GoalID : int = 0

func _ready():
	body_entered.connect(scored)

func scored(body):
	if(body.is_in_group("ball")):
		if(GoalID == 1):
			Global.P2_Score +=1
		elif(GoalID == 2):
			Global.P1_Score +=1

	body.queue_free()
```

A variable that we can change from the inspector, to keep track of who's goal this is. If it's 1 it belongs to P1 and will give a point to P2 when they knock the ball into it.

```gdscript
extends Area2D

#Variables
@export var GoalID : int = 0

func _ready():
    body_entered.connect(scored)

func scored(body):
    if(body.is_in_group("ball")):
        if(GoalID == 1):
            Global.P2_Score +=1
        elif(GoalID == 2):
            Global.P1_Score +=1

    body.queue_free()
```

When the goal is added to the scene the _ready() is run once.

We connect the body_entered signal to the scored() Function. This means that whenever ANY body (paddle, ball) enters the goal area then the scored() function will run.

```gdscript
extends Area2D

#Variables
@export var GoalID : int = 0

func _ready():
	body_entered.connect(scored)

func scored(body):
	if(body.is_in_group("ball")):
		if(GoalID == 1):
			Global.P2_Score +=1
		elif(GoalID == 2):
			Global.P1_Score +=1

		body.queue_free()
```

When a body enters the goal area, this function runs. Our first step is to check IF the body that has entered the area is the ball.

If it is the ball then it adds to the score of the correct player.

body.queue_free() deletes the ball after the score is added.

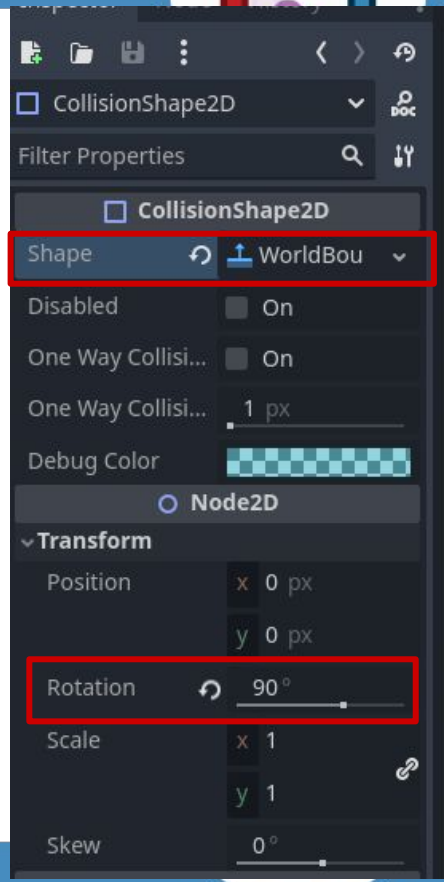
Adding Goals to the Main Scene

In the Main Scene, in 2D mode.

Drag the Goal.tscn from the Scenes folder in the FileSystem onto the Main Scene. Put it over towards the left hand side.

Add a CollisionShape2D as a child of the Goal.

On the CollisionShape2D, add a new WorldBoundaryShape2D as the shape and change the rotation to **90**.

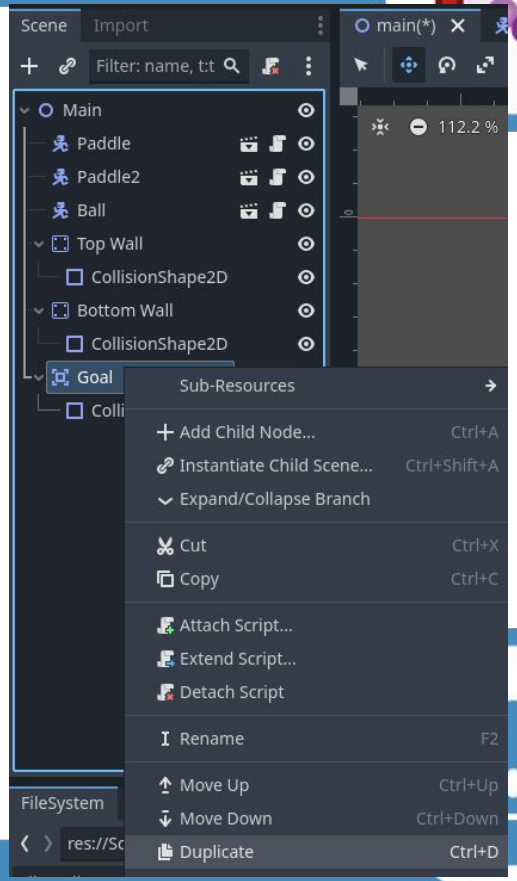


Making a second Goal in the Main Scene

Right Click the Goal node, choose duplicate.

You should now have Goal and Goal 2, both with collision shapes.

Don’t forget to change the Goal IDs in the Inspector. Left Goal should be 1, right goal should be 2.

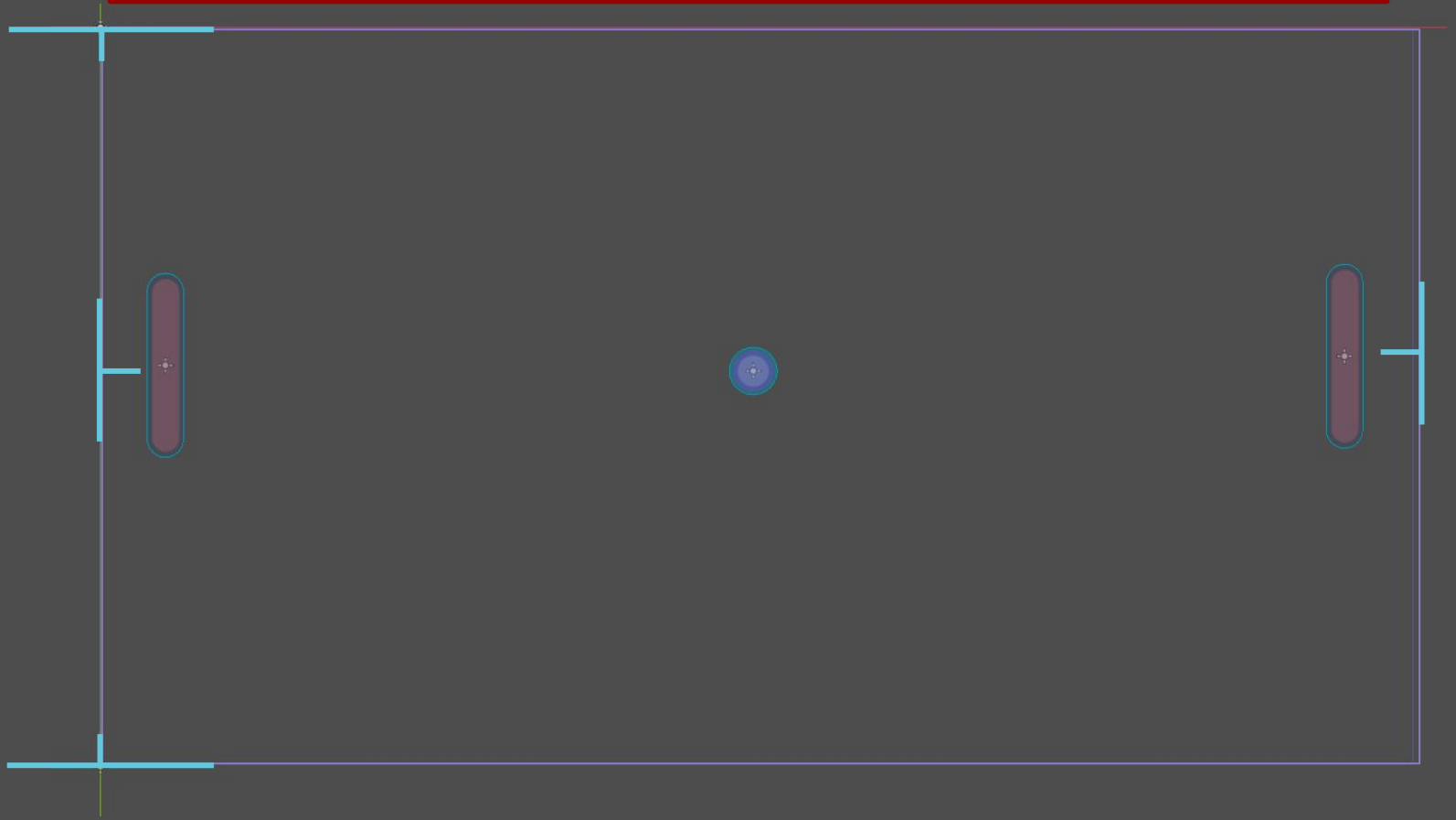


Making a second Goal in the Main Scene

Move Goal2 over to the right hand side using the move tool.

In the inspector, open the Transform Sub menu and change the rotation of Goal2 to 180. This will make it face the right way.

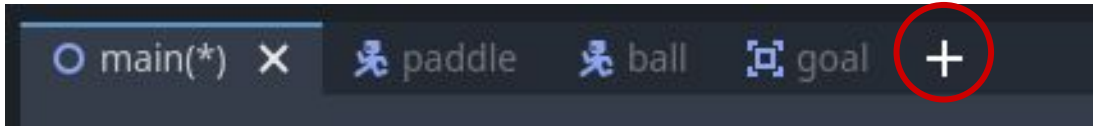
You should end up with a scene that looks something like this. The direction that the T shape faces matters.



Creating the Countdown Scene

It’s a bit jarring for the player’s to have to react instantly when the game begins. We should have a countdown each time the ball is about to spawn to give people time to figure out what they will do.

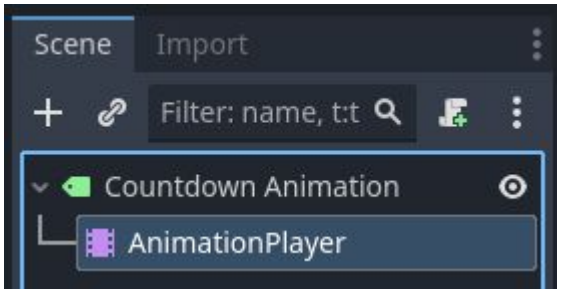
You know the drill, click on the + button next to Goal to add a new scene.



Creating the Countdown Scene

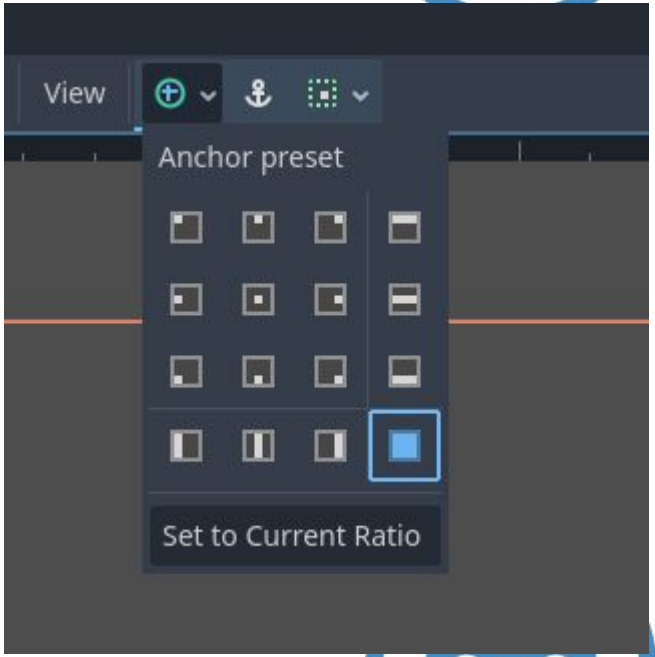
In the new scene, we will choose Other Node > Label. Rename the Label to be “Countdown Animation”.

Add an “Animation Player” as a child of this Countdown Animation node.



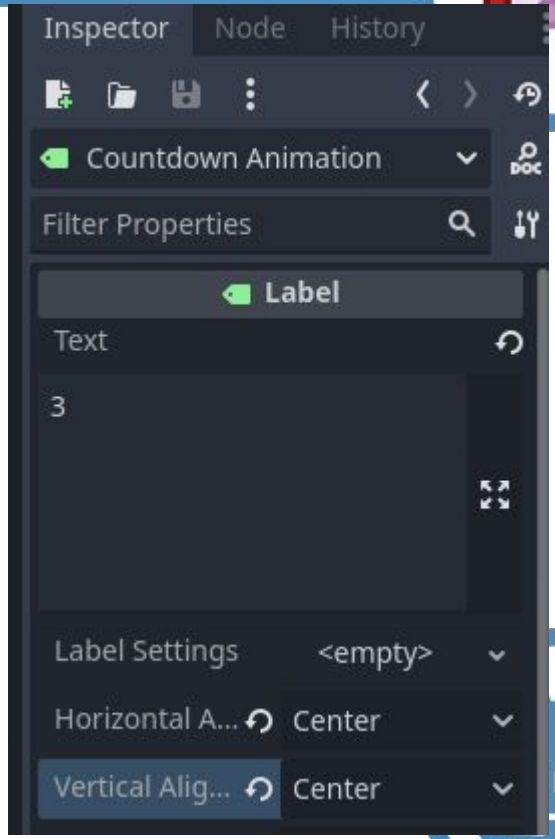
Setting up the Countdown Scene

Switch to 2D mode. Highlight the Countdown Animation Node and click on the Anchor Preset > Full Rect as shown on the right. This will stretch our label out so it fits over the whole screen and allows us to center our text perfectly.



Setting up the Countdown Scene

In the Text field in the Inspector write the number **3** and change the Horizontal Alignment and Vertical Alignment to both be “**Center**”

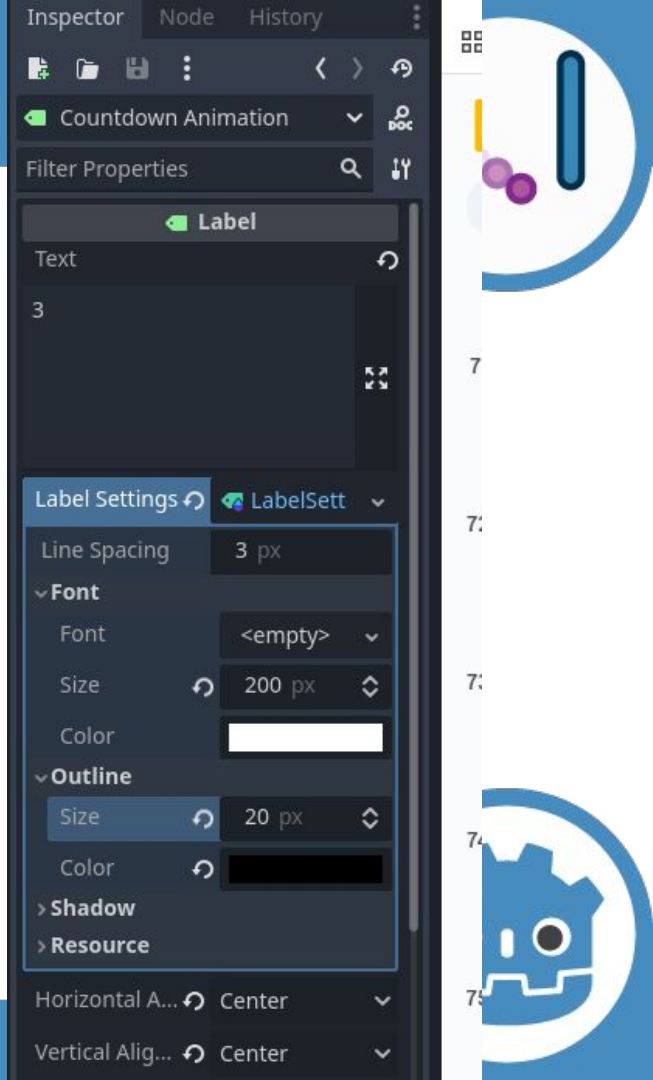


Setting up the Countdown Scene

For Label Settings, click on the empty section and choose “new Label Settings”

Under Font, change the size to be **200px**.

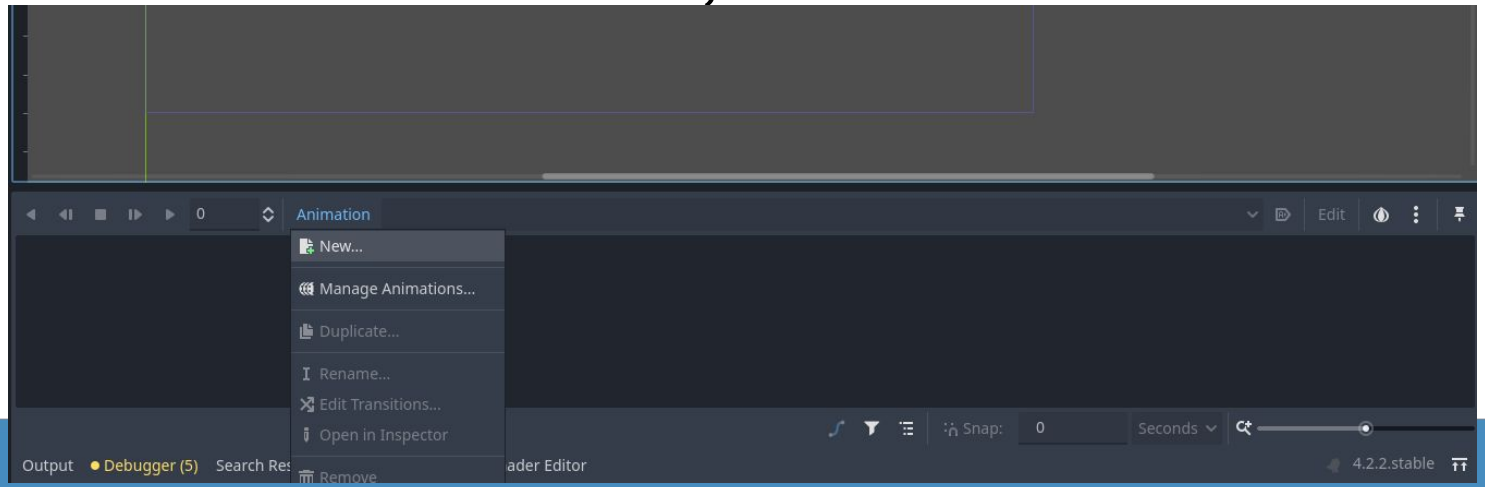
Under outline, change the size to **20px** and the color to **black**.



Setting up the animation player

Click on the Animation Player. A new window should pop up at the bottom of the screen. If not click on the Animation tab at the bottom.

Choose Animation > New, call it “countdown”



Setting up the animation player

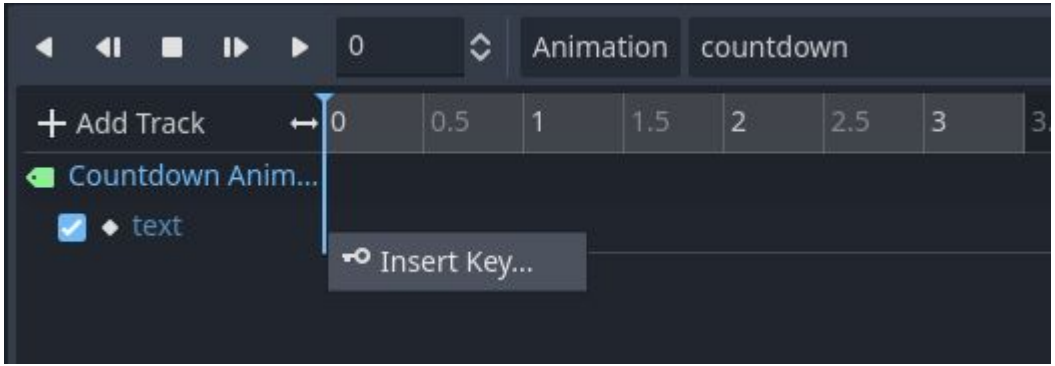
The animation player is very powerful, it allows us to plan a set of actions to trigger at certain times or change values at certain times. We're going to make a simple 3,2,1,Go! Countdown.

Change the length of the countdown to be 3.5 seconds

Setting up the animation player

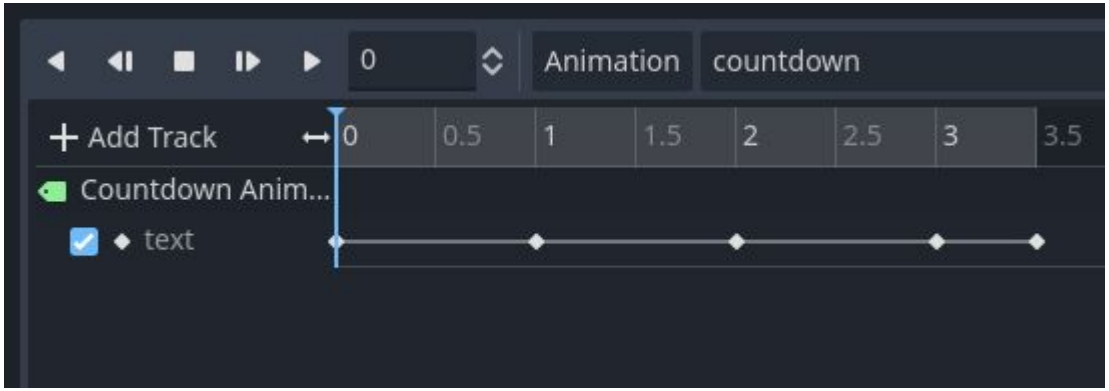
Choose Add Track > Property Track > Countdown Animation > Text. This lets us change the text value.

At the 0 mark on the timeline, right click and choose “Insert Key”

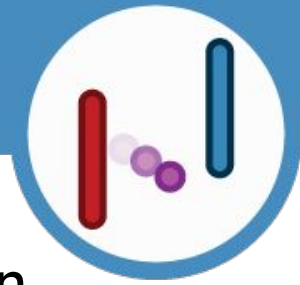


Setting up the animation player

At each timeline mark 1, 2, 3 and 3.5 Insert another key.



Setting up the animation player



For each diamond, click on it. Then change the value to in the inspector to the matching one from the table below

Key Time	Value
0	3
1	2
2	1
3	Go!
3.5	

Leave this one **blank**! It's an easy way to make the countdown go away once it's complete!

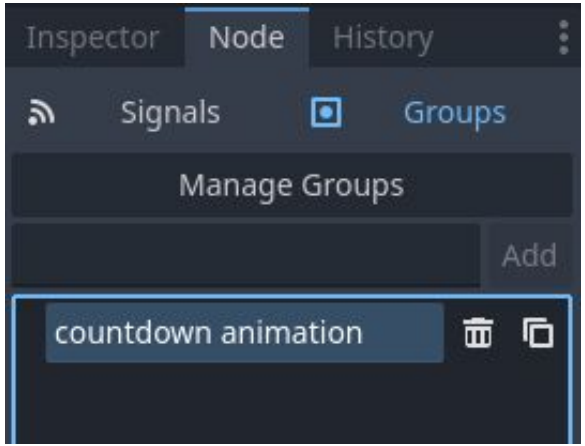
Testing the Countdown

Hit the Animation Play button to check it looks okay.

If all is well, go to Scene > Save Scene. Save this scene into the Scenes Folder and call it “Countdown_Animation”

Adding the Countdown to a group

Select the Animation Player Node and in the Node Panel > Group Add it to the “countdown animation” group by typing the name and pressing Add.



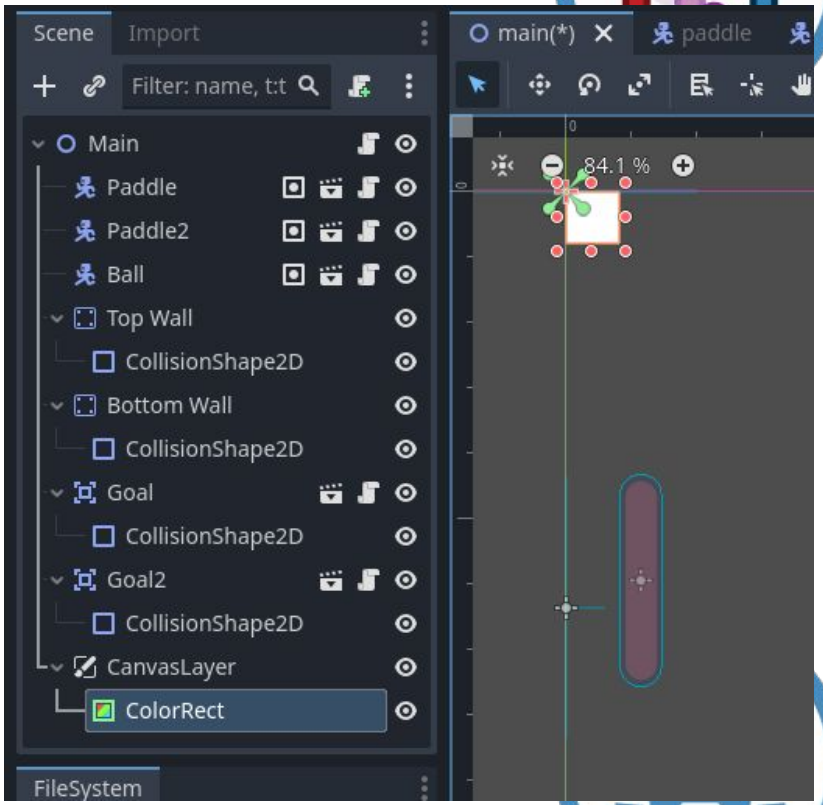
Adding the Countdown to the Main Scene

Go back to the Main Scene.

Highlight the Main node and add a “CanvasLayer” as a child.

With the CanvasLayer selected add a “Color Rect” as a child of it.

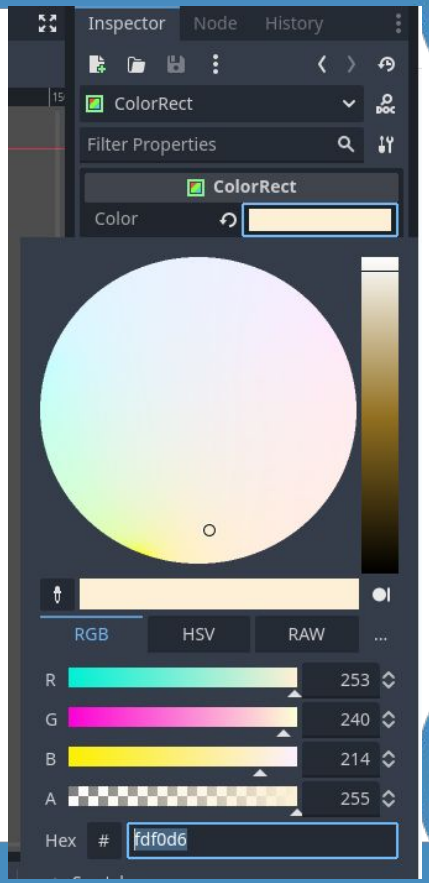
A white square should show up.



Adding the Countdown to the Main Scene

Change the Anchor Preset to Full Rect so it covers the whole screen.

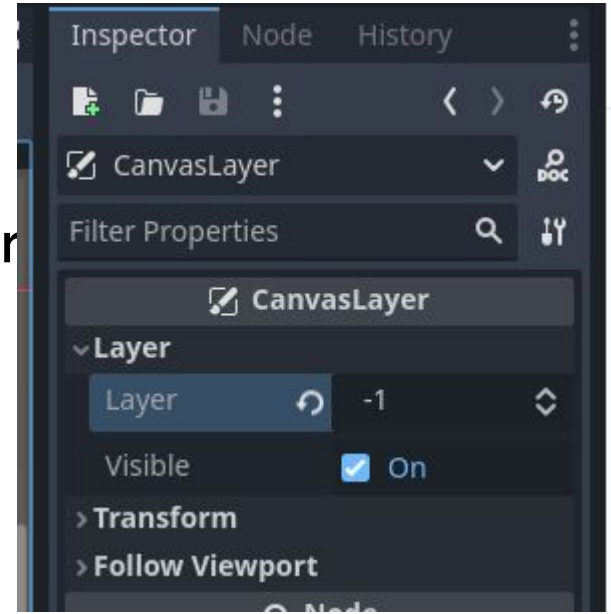
In the inspector, change the Color to something lighter that won't clash. I picked **#fdf0d6**



Adding the Countdown to the Main Scene

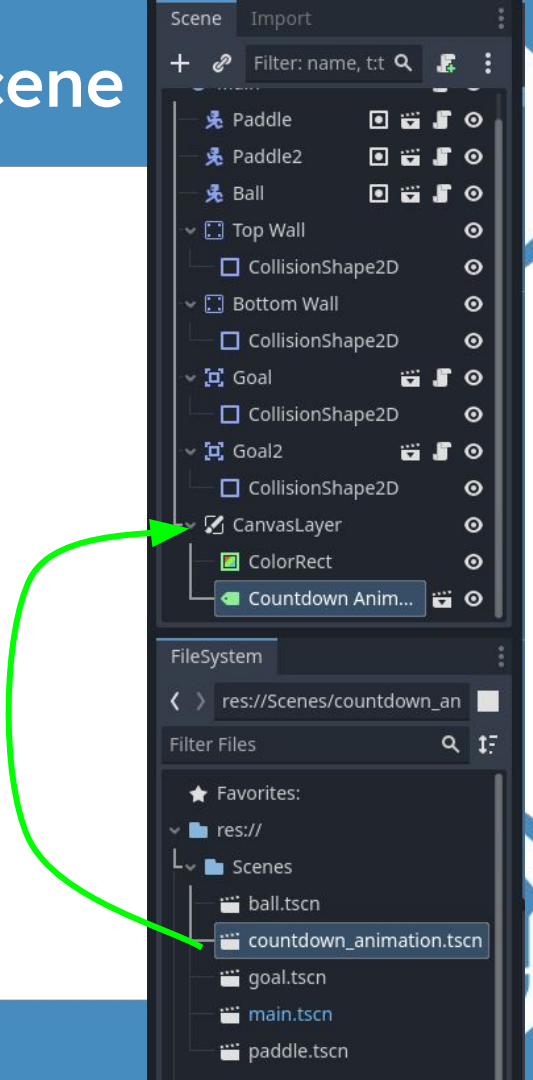
Our background is in front of the actual game!

To fix this choose the CanvasLayer then in the Inspector change the Layer to **-1**.



Adding the Countdown to the Main Scene

Finally, from the Scenes folder drag the countdown_animation.tscn onto the CanvasLayer to add it to the main game.



Minor upgrade to the Paddle Script

Change to Script mode at the top. Click on the Paddle.gd script. At the bottom, after the process function add these lines of code.

It does not need to be on exactly line 27-28.

```
27  func reset():
28      global_position.y = get_viewport_rect().size.y /2
29
```

This little function lets us move the paddle back to the center. Currently it’s not being called but we’ll do that in another script!

Minor upgrade to the Ball Script

Change to Script mode at the top. Click on the Ball.gd script. Replace the _ready() function with this new launch function.

```
6   #When the ball is created in the GameController Script it calls this function.
7   func launch(direction:Vector2):
8       velocity = direction * speed
```

The entire upgraded script can be seen on the next slide.

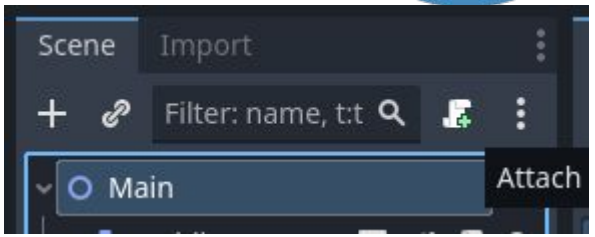
```
1   extends CharacterBody2D
2
3   #Variables
4   var speed:int = 500
5
6   #When the ball is created in the GameController Script it calls this function.
7   func launch(direction:Vector2):
8       velocity = direction * speed
9
10  #The process function runs every frame the game runs, many times a second.
11  func _process(delta):
12      var collision = move_and_collide(velocity*delta)
13      if (collision):
14          velocity = velocity.bounce(collision.get_normal())
15
16
```

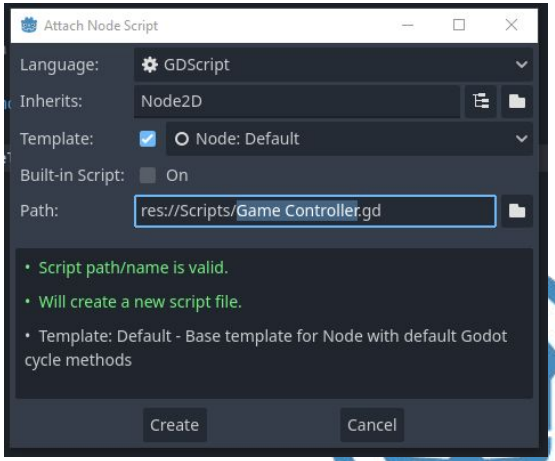
Controlling the whole game

For this to really feel like a game we need to implement a way to respawn start the game properly, respawn the ball and even give us space to add a win condition.

In the Main Scene, add a script to the Main node

Change the path to the Scripts folder and name it “Game Controller”





The Game Controller Script

This script is a little bit longer than the other ones! I'll split it up into two slides. All of this should flow one after another. Please have a look at the attached code listing's in the Challenge folder to see how it's laid out if you are confused.

I'll explain the code as usual on the slides following the script.

```gdscript
extends Node2D

#variables
var countdownAnim: AnimationPlayer
var ball:PackedScene = load("res://Scenes/ball.tscn")
signal resetPaddle

func _ready():
	StartGame()

func StartGame():
	countdownAnim = get_tree().get_first_node_in_group("countdown animation")
	var paddles = get_tree().get_nodes_in_group("paddle")

	for paddle in paddles:
		connect("resetPaddle",paddle.reset)

	Global.P1_Score = 0
	Global.P2_Score = 0
	resetPaddle.emit()

	startNextRound()
```

```
23
24 func startNextRound():
25     if(Global.P1_Score==5 or Global.P2_Score ==5):
26         call_deferred("endGame")
27         return
28     countdownAnim.play("countdown")
29     await countdownAnim.animation_finished
30     spawnBall()
31
32 func endGame():
33     get_tree().change_scene_to_file("res://Scenes/end screen.tscn")
34
35 func spawnBall():
36     var newBall= ball.instantiate()
37     newBall.global_position = get_viewport().get_visible_rect().size/2
38     var direction = Vector2.from_angle(randf_range(0,359))
39     newBall.launch(direction)
40     get_tree().root.get_child(1).add_child(newBall)
41
```

```gdscript
extends Node2D

#variables
var countdownAnim: AnimationPlayer
var ball:PackedScene = load("res://Scenes/ball.tscn")
signal resetPaddle

func _ready():
    StartGame()

func StartGame():
    countdownAnim = get_tree().get_first_node_in_group("countdown animation")
    var paddles = get_tree().get_nodes_in_group("paddle")

    for paddle in paddles:
        connect("resetPaddle",paddle.reset)

    Global.P1_Score = 0
    Global.P2_Score = 0
    resetPaddle.emit()

    startNextRound()
```

Our variables for this script:

- A reference to the countdownAnimation's AnimationPlayer.
- A reference to the Ball scene, so we can create more.
- A custom signal, this will let us send messages to the Paddles to tell them when to return to the center.

```gdscript
extends Node2D

#variables
var countdownAnim: AnimationPlayer
var ball:PackedScene = load("res://Scenes/ball.tscn")
signal resetPaddle

func _ready():
    StartGame()

func StartGame():
    countdownAnim = get_tree().get_first_node_in_group("countdown animation")
    var paddles = get_tree().get_nodes_in_group("paddle")

    for paddle in paddles:
        connect("resetPaddle",paddle.reset)

    Global.P1_Score = 0
    Global.P2_Score = 0
    resetPaddle.emit()

    startNextRound()
```

When the Main scene is run, it calls the StartGame() Function.

```gdscript
extends Node2D

#variables
var countdownAnim: AnimationPlayer
var ball:PackedScene = load("res://Scenes/ball.tscn")
signal resetPaddle

func _ready():
    StartGame()

func StartGame():
    countdownAnim = get_tree().get_first_node_in_group("countdown animation")
    var paddles = get_tree().get_nodes_in_group("paddle")

    for paddle in paddles:
        connect("resetPaddle",paddle.reset)

    Global.P1_Score = 0
    Global.P2_Score = 0
    resetPaddle.emit()

    startNextRound()
```

We assign the countdownAnim using the groups we'd set up earlier.

We get an array of all of the Paddles that are in the scene.

We then loop through each paddle and connect that resetPaddle Signal so that the paddles know to listen for it.

```gdscript
extends Node2D

#variables
var countdownAnim: AnimationPlayer
var ball:PackedScene = load("res://Scenes/ball.tscn")
signal resetPaddle

func _ready():
    StartGame()

func StartGame():
    countdownAnim = get_tree().get_first_node_in_group("countdown animation")
    var paddles = get_tree().get_nodes_in_group("paddle")

    for paddle in paddles:
        connect("resetPaddle",paddle.reset)

    Global.P1_Score = 0
    Global.P2_Score = 0
    resetPaddle.emit()

    startNextRound()
```

At the start of the game we need to reset both player's scores to 0.

We broadcast the resetPaddle signal which runs the reset() function we wrote for the paddles and moves them back to the center of the field.

We then call the startNextRound() function.

```
23
24  func startNextRound():
25      if(Global.P1_Score==5 or Global.P2_Score ==5):
26          call_deferred("endGame")
27          return
28      countdownAnim.play("countdown")
29      await countdownAnim.animation_finished
30      spawnBall()
31
32  func endGame():
33      get_tree().change_scene_to_file("res://Scenes/er
34
35  func spawnBall():
36      var newBall= ball.instantiate()
37      newBall.global_position = get_viewport().get_visible_rect().size/2
38      var direction = Vector2.from_angle(randf_range(0,359))
39      newBall.launch(direction)
40      get_tree().root.get_child(1).add_child(newBall)
41
```

The startNextRound() function will be called

- When the Game Begins

And

- Once a player has scored.

First thing we do is check if either player has 5 points. If they do, they win. endGame() function is called and the rest of the code in this function is skipped.

If nobody has won then it starts the countdown animation we made, waits for it to finish then calls the spawnBall() function.

```
23
24  func startNextRound():
25      if(Global.P1_Score==5 or Global.P2_Score ==5):
26          call_deferred("endGame")
27          return
28      countdownAnim.play("countdown")
29      await countdownAnim.animation_finished
30      spawnBall()
31
32  func endGame():
33      get_tree().change_scene_to_file("res://Scenes/end screen.tscn")
34
35  func spawnBall():
36      var newBall= ball.instantiate()
37      newBall.global_position = get_viewport().get_visible_rect().size/2
38      var direction = Vector2.from_angle(randf_range(0,359))
39      newBall.launch(direction)
40      get_tree().root.get_child(1).add_child(newBall)
41
```

The end game function simply tries to switch the scene to the end screen. We've not actually made this yet! So it won't do anything until we do. This means the game will just keep going!

This bit of code looks tricky. Let's break it down.
The spawnBall() function is run after a player scores, that means the ball has been deleted already. We need to create a new ball, put it in the middle of the screen and then send it flying in a random direction.

Line 36: Creates a brand new copy of the Ball scene. This is called "Instantiating a scene"
Line 37: Moves the ball to the center of the screen. We get that visible purple rectangle and find out exactly half way by dividing it by 2.
Line 38: we pick a random angle between 0 and 359 for the ball to shoot off in.
Line 39: we call the launch function to shoot the ball off in the chosen direction
Line 40: This line of code looks at the current scene. It then picks the top node, in this case it will be the Main Node. We use this line to add the newBall we've made to the scene so it can actually be used, before you do this the ball does not actually appear in the game.

```
35 func spawnBall():
36     var newBall= ball.instantiate()
37     newBall.global_position = get_viewport().get_visible_rect().size/2
38     var direction = Vector2.from_angle(randf_range(0,359))
39     newBall.launch(direction)
40     get_tree().root.get_child(1).add_child(newBall)
```

The Game Controller Script

One more little upgrade! Now that we've got a way to start a new round lets make the Goal actually trigger the new round.

Switch to Script mode, go to the Goal.gd Script

At the bottom of the scored() function add

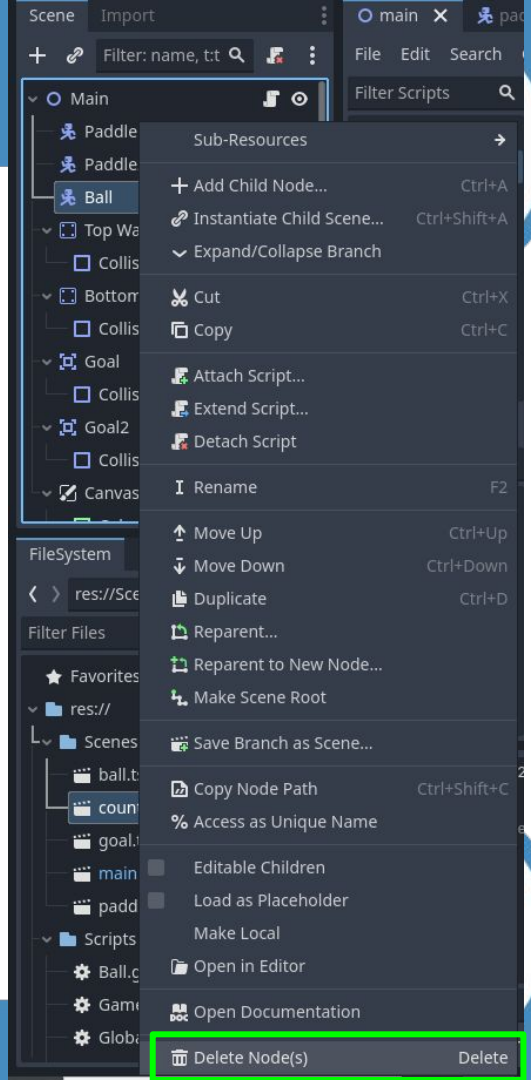
```
9  func scored(body):
10     if(body.is_in_group("ball")):
11         if(GoalID == 1):
12             Global.P2_Score +=1
13         elif(GoalID == 2):
14             Global.P1_Score +=1
15
16         body.queue_free()
17         owner.startNextRound()
```

Test the game!

Go back to the main scene, switch to 2D mode.

Time to test out all of our recent changes. Delete the ball from the SceneTree in the main Scene by right clicking then choosing Delete Node(s). We'll create these automatically now.

Then hit the Play Button and see if it works!

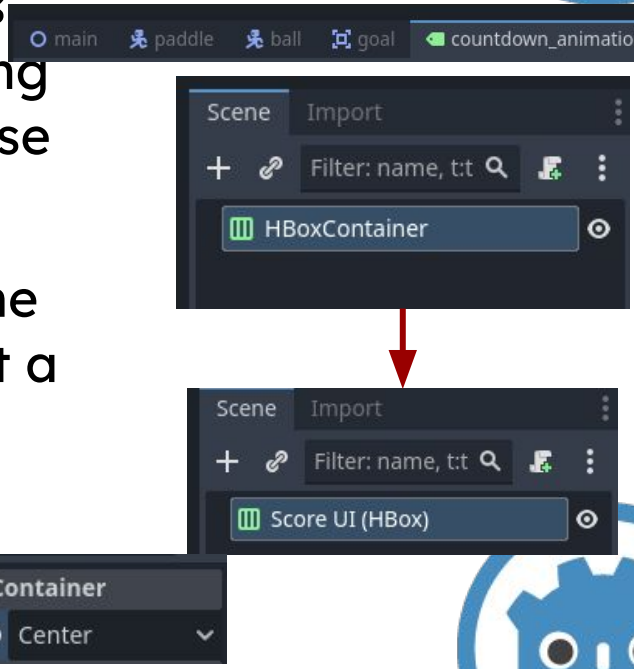


Creating the Score UI Scene

Right now we're keeping track of the scores for both players but we're not doing anything with them. It would be useful to display these scores at the top of the screen.

Let's make a new Scene. Click on the + at the top, add an "Other Node" this time we want a "HBoxContainer". Rename it to "Score UI (HBox)"

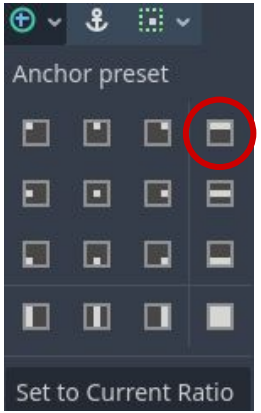
Change the Alignment to "Center"

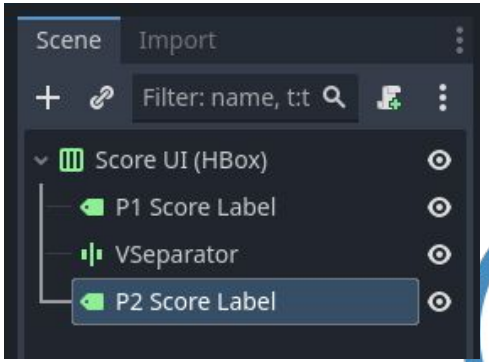


Structuring the Score UI Scene

Change the Score UI (HBox) Anchor Preset to be Top Wide. Add the following children nodes to the Score UI(HBox)

- A Label, renamed to P1 Score Label
- A VSeperator
- A Label, renamed to P2 Score Label





Setting up the Score UI Scene

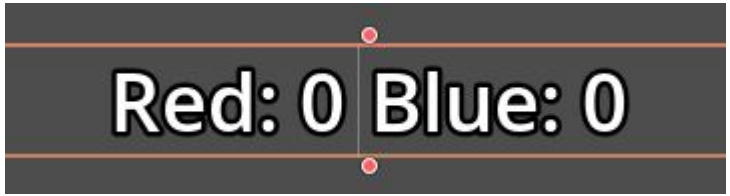
Some small changes are required for both Labels.

P1 Score Label: In the Text field add “Red: 0”

P2 Score Label: In the Text field add “Blue: 0”

Both Labels: Add a Label Setting with

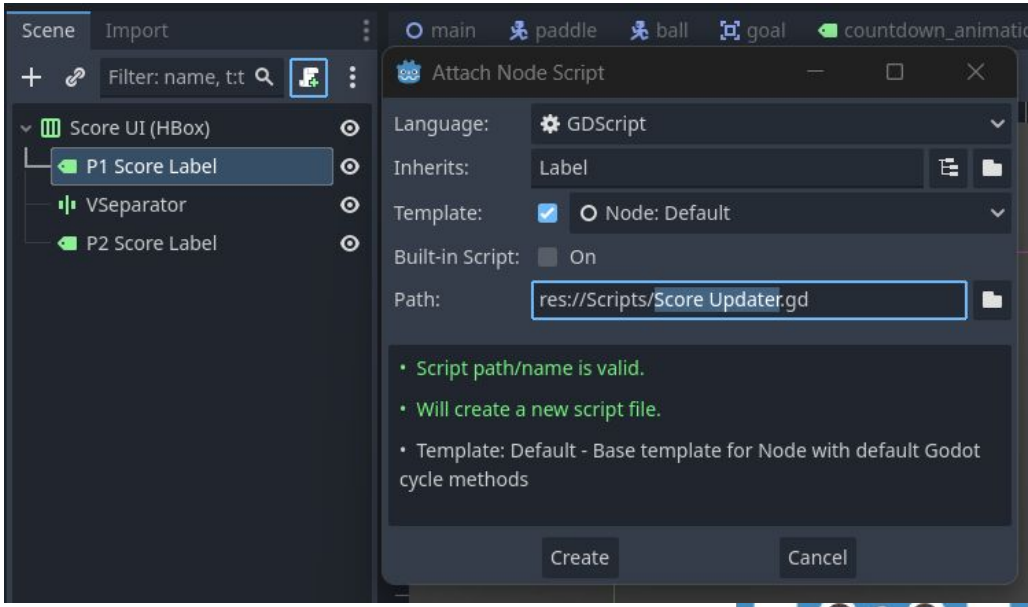
(Font Size: **40px**. Outline Size: **10px**, Color: **Black**)



Making the UI update

Add a script to the P1 Score Label. Make sure you change the path to be in the Scripts folder. Call it “Score Updater”.

This script will also be attached to the P2 Score Label later.



```
extends Label

@export var playerID:int = 1

# This function is called every frame, many times a second.
func _process(delta):
    if(playerID ==1):
        text = "Red: " + str(Global.P1_Score)
    elif(playerID == 2):
        text = "Blue: "+ str(Global.P2_Score)
    else:
        printerr("The playerID of ", name, " is incorrect.")
```

Keeps track of who's score the label should display. Should be 1 or 2.

```
1   extends Label
2
3   @export var playerID:int = 1
4
5   # This function is called every frame, many times a second.
6   func _process(delta):
7       if(playerID ==1):
8           text = "Red: " + str(Global.P1_Score)
9       elif(playerID == 2):
10          text = "Blue: "+ str(Global.P2_Score)
11      else:
12          printerr("The playerID of ", name, " is incorrect.")
```

The _process() function runs every frame. This is a bit wasteful as our score certainly won't be changing that fast but it really doesn't matter at all for project this small.

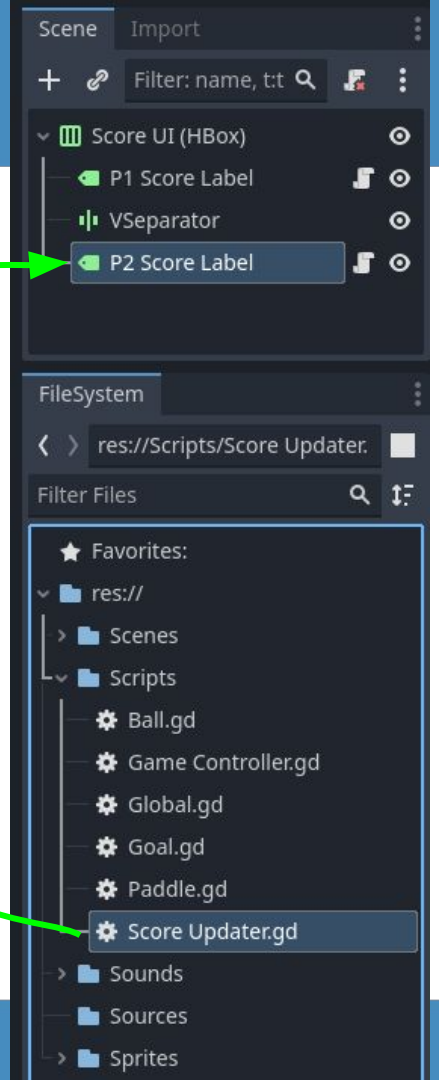
In the _process() function the text is simply updated to match the appropriate player's score based on the playerID.
The str() function changes whatever information you give it into a string that can be displayed in the text field of a label.

```
1   extends Label
2
3   @export var playerID:int = 
4
5   # This function is called every frame, many times a second.
6   func _process(_delta):
7       if(playerID ==1):
8           text = "Red: " + str(Global.P1_Score)
9       elif(playerID == 2):
10          text = "Blue: "+ str(Global.P2_Score)
11      else:
12          printerr("The playerID of ", name, " is incorrect.")
```

Making the UI update

Now drag and drop the new Score Updater script from the Scripts folder onto the P2 Score Label to attach it to this script as well.

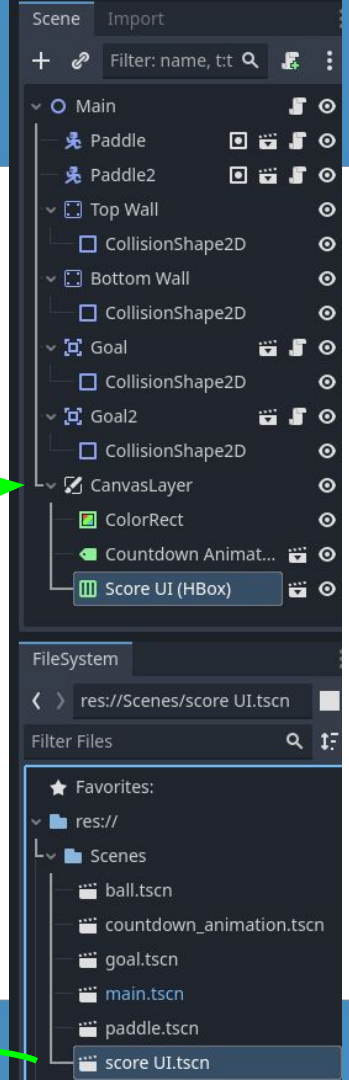
In the P2 Score Label Inspector change the playerID to **2**.



Add the Score UI to the main scene

Save the Score UI Scene with Scene > Save Scene.

Navigate to the Main Scene and drag and drop the new Score UI.tscn file onto the CanvasLayer Node in the Main Scene.



Creating the EndScreen Scene

We've already got the code setup to try to send us to an end screen when a player reaches 5 points. Now would be a good time to actually make that scene!

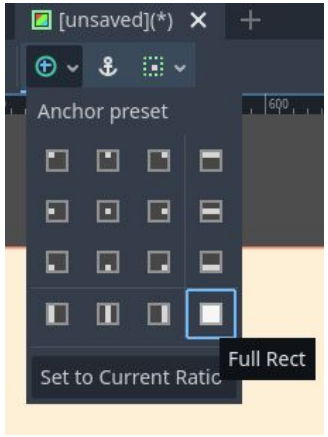
At the top of the screen, click the + to add a new scene.

Choose a Other Node > ColorRect.

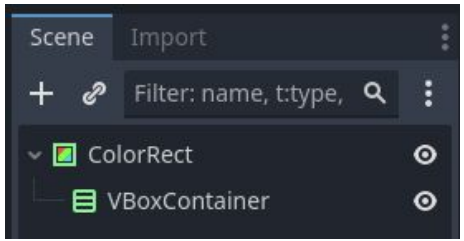
Save the Scene as "end screen.tscn" if the name is not correct then it will fail to load.

Structuring the End Scene

Change the Anchor settings on the Color Rect to be Full Rect and in the Inspector change its color to be something more pleasant. I'm using #fdf0d6 again.



Add a VboxContainer as a child of the ColorRect. Change it's anchor settings to Full Rect too.



Structuring the End Scene

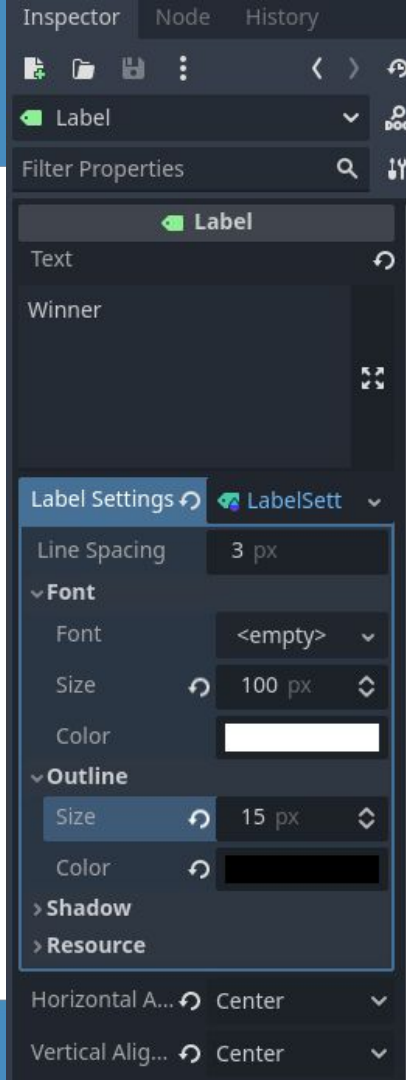
We'll need a few things to be children of this VBoxContainer. We'll go through them from top to bottom and the Inspector settings you should change.

Setting up the End Scene - Winner Label

Highlight the VboxContainer and add a Label as a child.

Using the inspector, set these properties:

- Change the Text Field to say “**Winner**”
- Add a new label setting with font size: **90px**, Outline size **15px**, Outline Color **Black**
- Horizontal and Vertical Alignment: **Center**



End Scene - Play Again Button

Highlight the VboxContainer and add a button as a child.

Using the inspector, set these properties:

- Change the Text Field to say “**Play Again**”
- Change Layout > Container Sizing > Horizontal to **Shrink Center**
- Change Theme Overrides > Font Sizes to **50px**

End Scene - Quit Button

Highlight the VboxContainer and add a button as a child.

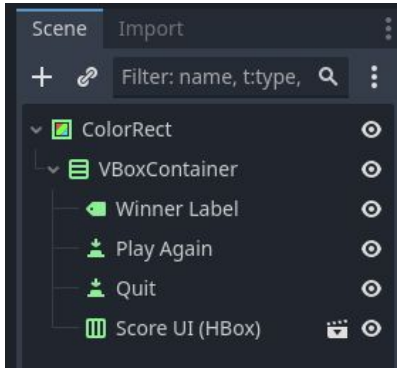
Using the inspector, set these properties:

- Change the Text Field to say “**Quit**”
- Change Layout > Container Sizing > Horizontal to **Shrink Center**
- Change Theme Overrides > Font Sizes to **50px**

Structuring the End Scene - Scores

From the FileSystem drag and drop the “Score UI.tscn” into the VboxContainer at the bottom of the list.

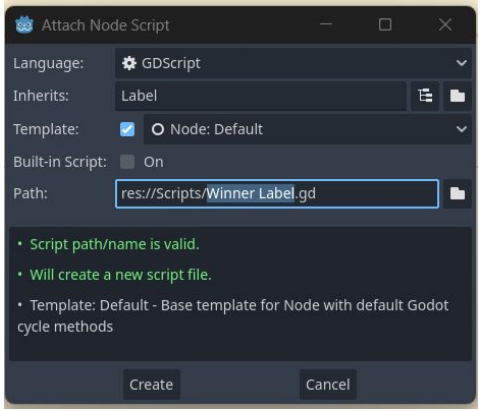
Rename all of the nodes to match below.



Making the End Scene Functional

Add a script to the **Winner Label**. Save it in the Scripts folder as **Winner Label**.

If Player 1 has more points than Player 2, it shows “Red Wins!” otherwise, “Blue Wins!”

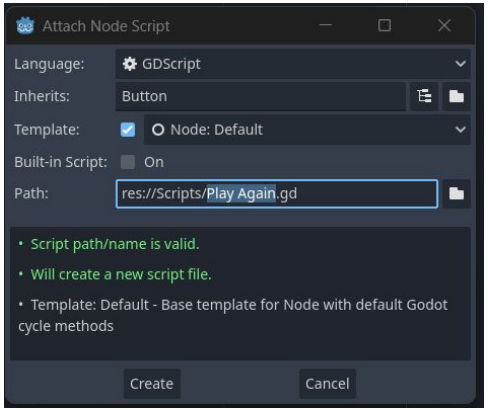



```gdscript
extends Label

#Called when the winner label is loaded into the scene
func _ready():
    if(Global.P1_Score > Global.P2_Score):
        text= "Red Wins!"
    else:
        text= "Blue Wins!"
```

Making the End Scene Functional

Add a script to the **Play Again Button**. Save it in the Scripts folder as **Play Again**.

This button listens for when the button is pressed. Then changes scene back to the main.

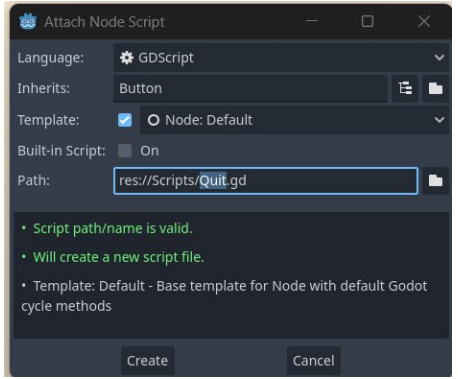



```gdscript
extends Button

#Called when the button is loaded into the scene
func _ready():
    pressed.connect(playAgain)

#Called when the button is pressed.
func playAgain():
    get_tree().change_scene_to_file("res://Scenes/main.tscn")
```

Making the End Scene Functional

Add a script to the **Quit Button**. Save it in the Scripts folder as **Quit**.

This button listens for when the button is pressed. Then quits the game entirely.




```gdscript
extends Button

#Called when the button is loaded into the scene
func _ready():
    pressed.connect(quit)

#Called when the button is pressed.
func quit():
    get_tree().quit()
```

Final Test!

Hit play and test out the game!

- You should be able to move both paddles independently.
- Score points when the ball gets past either paddle.
- The points should update on the UI.
- When either player scores a total of 5 points the game should end and display a winner.