# Introduction to Godot

By Lyhdyr Esquerdo Teixidó

# Some considerations

- The workshop will be in English since it is part of the FairGame project and it's streamed to several schools across Europe, but if you have any question and feel more comfortable asking it in Spanish or Catalan, that's okay!

- I'll reply in English though, so everybody can understand the answer. If you need me to explain something in another language you can ask me at the end of the class or email me at lyhdyr09enderu@gmail.com

- The class will be recorded and you'll have all the materials available on both Google Drive and GitHub

# About me

**they/them** (en)
**elle** (es)
**elli** (ca)

- Producer, Programmer & Character Artist at Lemon Bird, indie studio currently under the preincubation & acceleration program PowerUp+
- Degrees in English Studies (2014, UB) & Game Design and Development (2024, UdG), Master in Gender Studies (2015, UB)
- Active member of FemDevs
- Also works in a museum, has worked in IT consultancies, schools and more
- Writer when life allows it ;_;
- Loves cooking & watching series, has 5 birbs

https://linktr.ee/lyhdyr

# Godot basics

# What is Godot

Open source game engine that allows you to develop games in 2D and 3D and export them to different platforms like Windows, iOS, Linux, Android, Web and even consoles*

(*with a development kit)

# FOSS

“Free and Open Source Software”

Source code is available to users so they can **study it, modify it and improve it**.

Software is improved and maintained by contributions of the **community**.

Other examples of FOSS software:

- Blender, GIMP, Mozilla Firefox, LibreOffice, VLC Media Player...

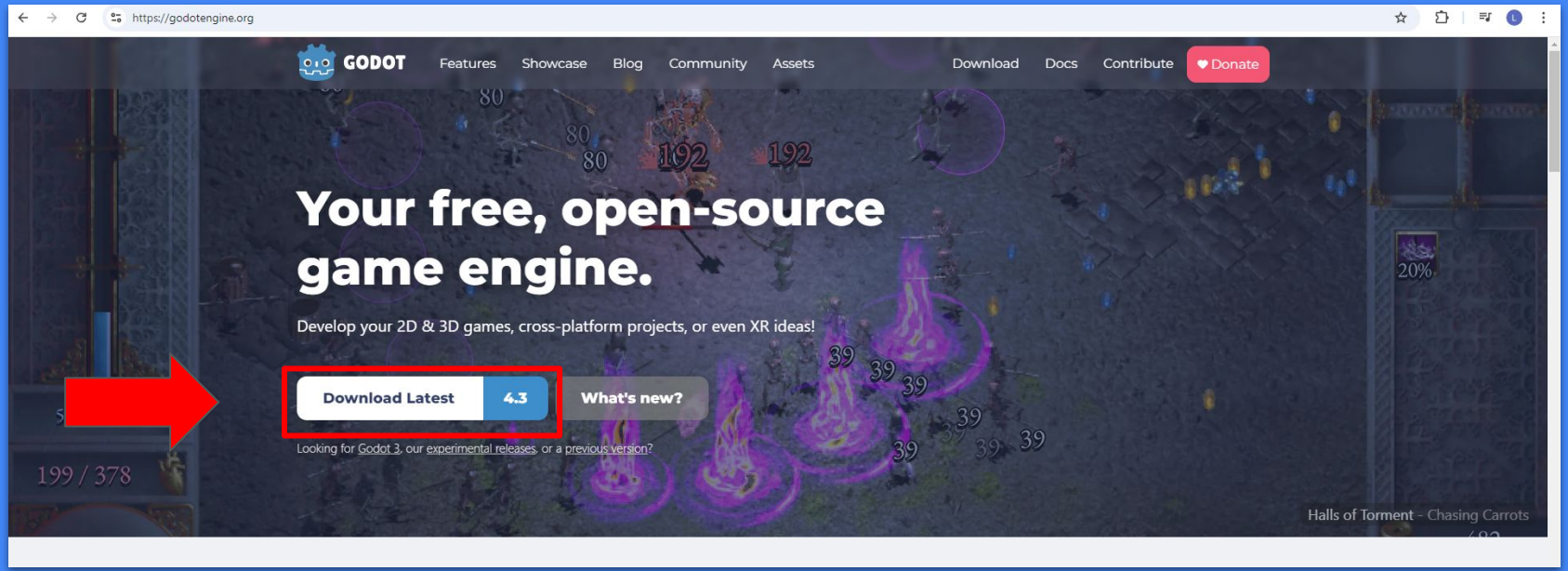
# Why Godot

No license fees, subscription fees or shady clauses!

- Free & open-source
- Clear and complete documentation
- Strong community
  - Contributes to improving the engine by reporting & fixing bugs, implementing new features, writing documentation
  - Shared knowledge, tutorials, forums...
- Light-weight
  - The 4.3 .exe file size is 126 MB!
- System requirements low compared to other engines
- No login to start
- No need to install
- Easy to learn

# How to download Godot

https://godotengine.org/



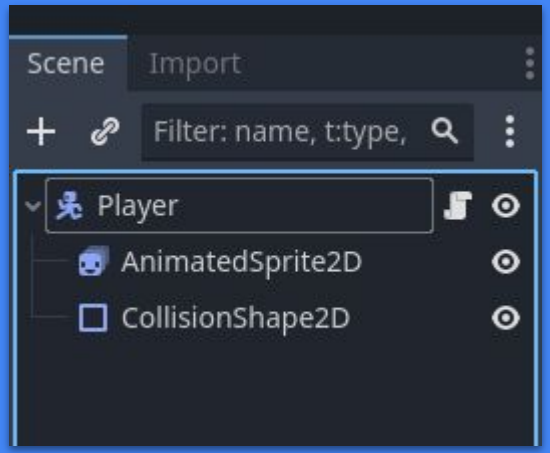
# Nodes

Nodes: the basic unit in Godot. Everything is made of nodes and you can combine them to make new elements.

- Ex. Node, Timer, AudioStreamPlayer, Button, Label, Sprite2D, TextureRect, AnimationPlayer, Camera3D, CollisionShape2D...

Nodes are organized in a tree system in the game.

Scenes: groups of nodes that are saved in the FileSystem so they can be easily reused and organized inside the project.

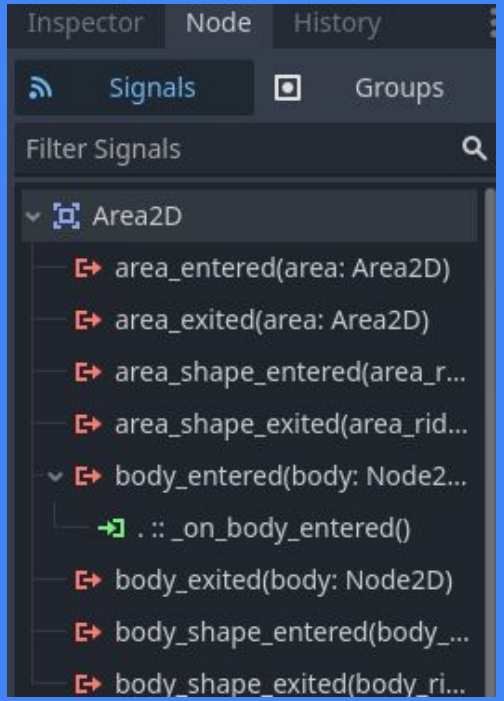


# Signals

Nodes can emit a signal informing about something that happened.

Ex. A timer reached its timeout time, a button was pressed…

We can also create our custom signals!



# Scripts

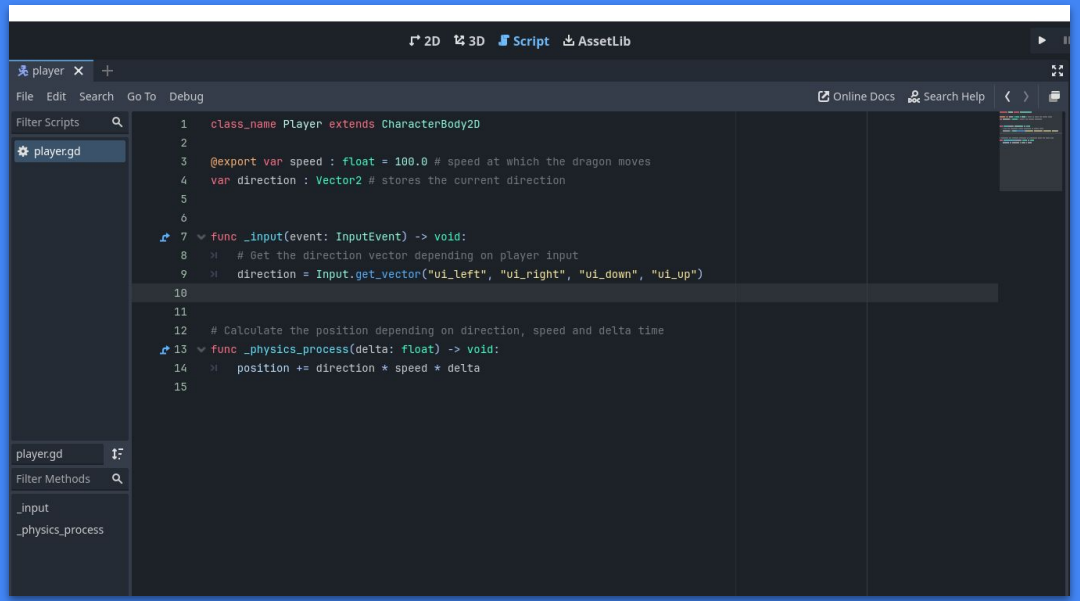
We define a node's behaviour by attaching scripts (code) to it.

- Godot has its own scripting language, GDScript, which is based on Python.
- It has also official support for C# and C++
- 3rd party integrations for C, Rust, D, Python, Nim, Go, Lua...
- You can use different languages in the same project
  - Ex. code most of the game logic with GDScript but certain parts with C+ for optimization
- However, it's highly recommended to learn GDScript, since other languages don't have complete support.
  - Ex. you can export to Web games made with GDScript but not in C#!

# Godot’s IDE

**Integrated Development Environment**

- Godot has its own built-in IDE where we can write our scripts without needing to use external IDEs like VS Code, Visual Studio or others…
- It can be accessed from the “Script” tab



# Resources

Resources are data containers.

Godot has its own resource types, like Animation, Mesh (3D model), Texture (image), Theme (style), AudioStream (audio)...

We can also create our own custom resource types by defining them on a script!

Ex. Item data, NPC data...

```
class_name Item extends Resource

@export var name : String
@export var description : String
@export var image : Texture2D
@export var price : int
```

**Example code defining an Item Resource**

# Tutorial project

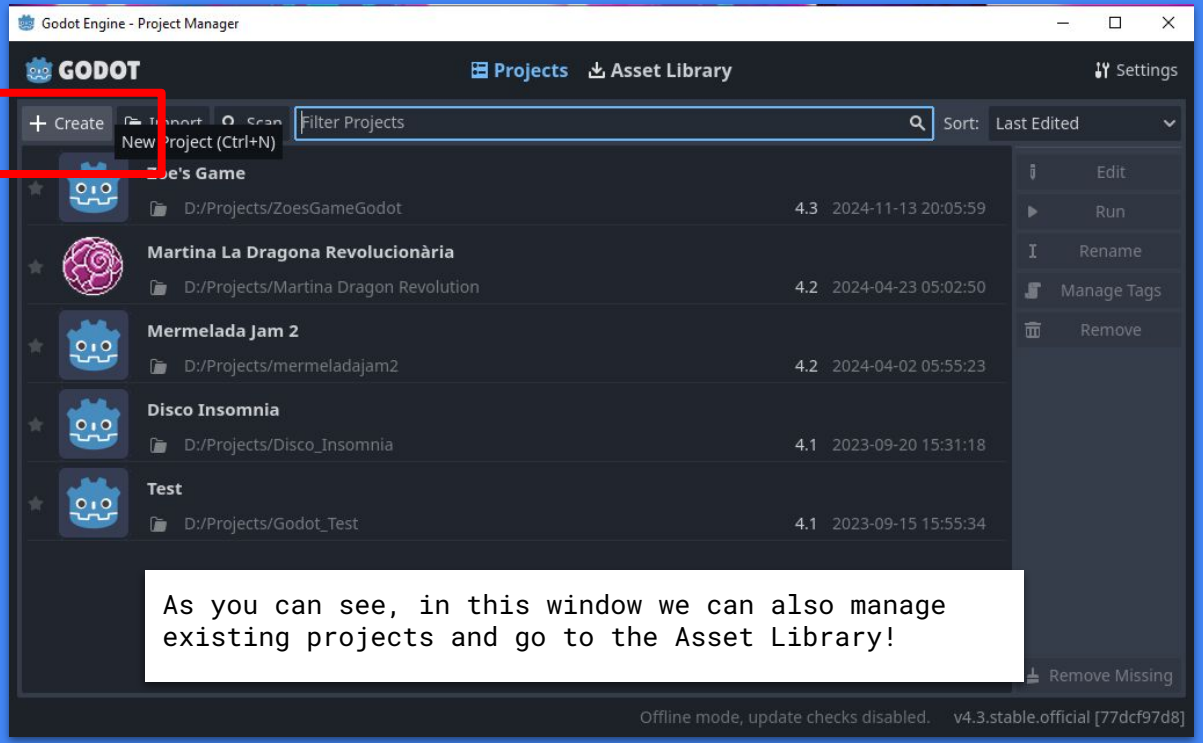
# What game will we make?

- A simplified version of the game “Martina La Dragona Revolucionària” I made as a solo dev in Sant Jordi Jam 2024
- Arcade game
  1. Player: a dragon that flies through the sky
  2. Level with parallax background
  3. Obstacles that hurt you
  4. Gameover
  5. Time
  6. Music
  7. Exporting the game


**The original game**

# Creating the project

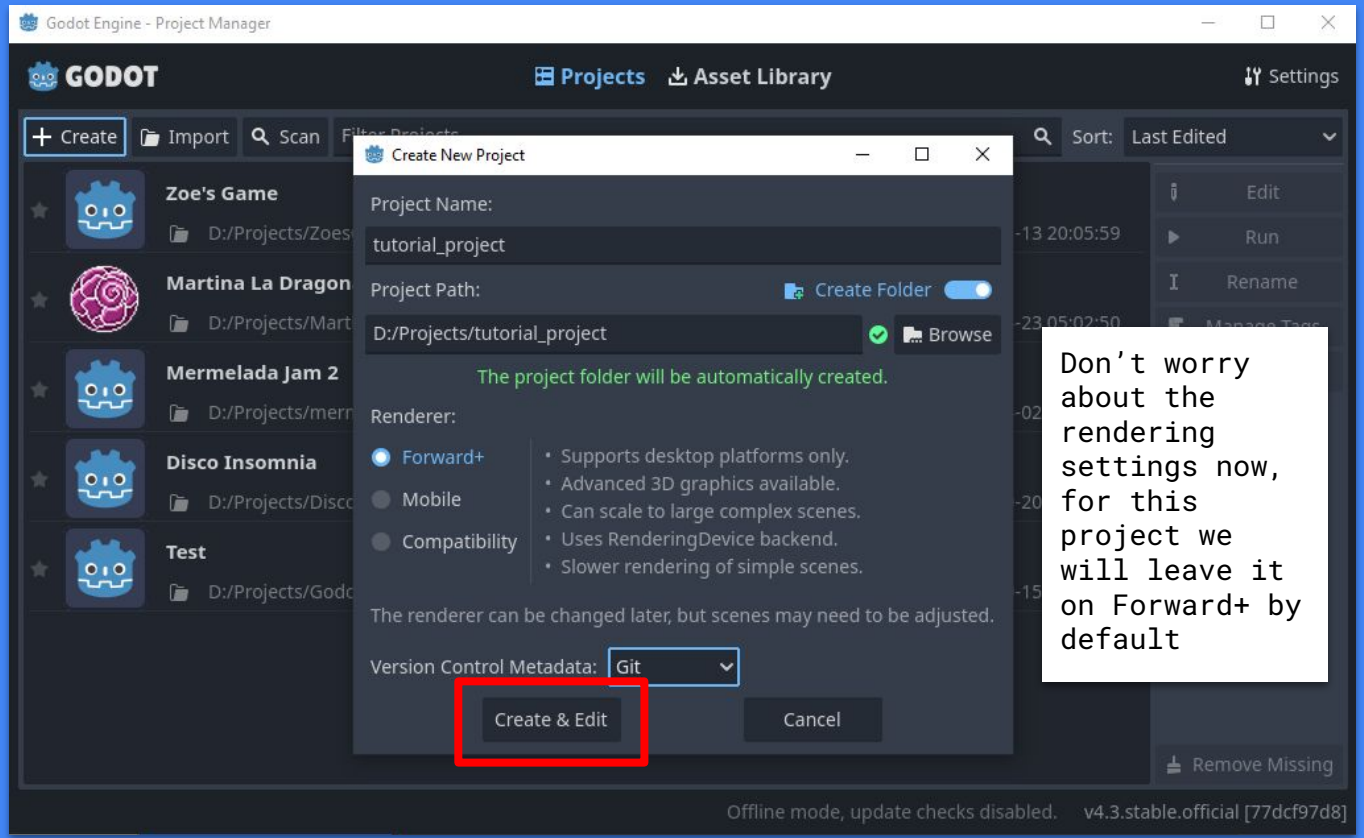
Simply click on the godot icon to execute the program and then on the +Create button on the Project Manager window.


As you can see, in this window we can also manage existing projects and go to the Asset Library!

Write a name for your project and choose the path where it will be created.

Then click on “Create & Edit” and the project will be created. It may take a while so be patient!




Don’t worry about the rendering settings now, for this project we will leave it on Forward+ by default

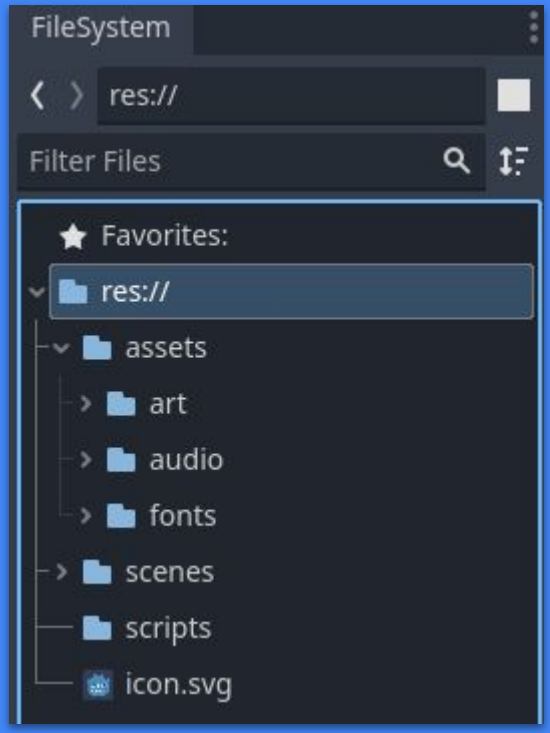
# Organizing the FileSystem

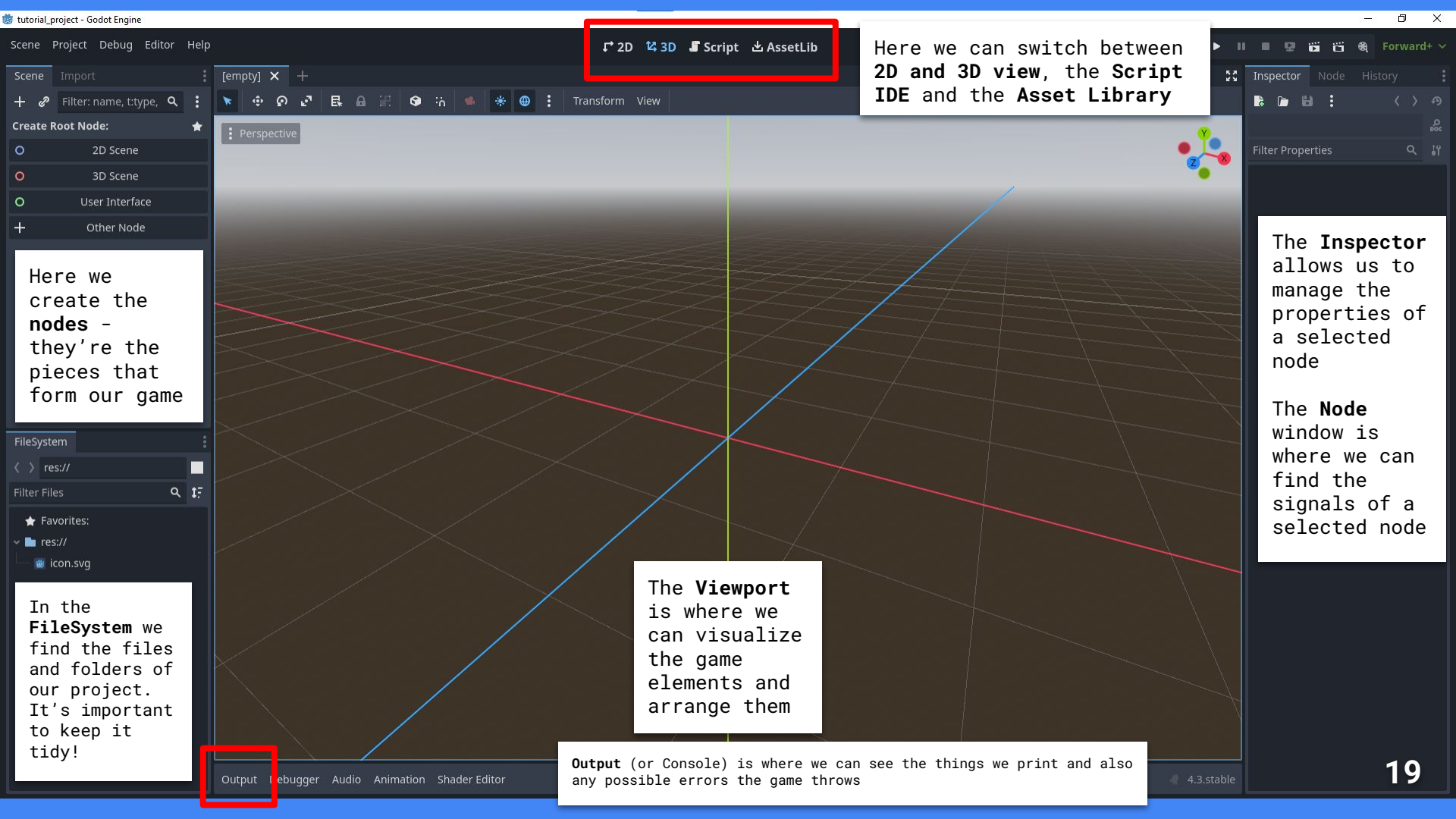
1. Now we should create a “scenes” and a “scripts” folders in the FileSystem so we can keep everything more organized.

2. We can also copy or drop the assets folder that contains the art, sounds and fonts necessary for this tutorial.

3. Projects can scale very fast so it’s good practice to organize elements in folders!

4. We can create more folders at any time. We can also rearrange elements, but we should be careful because that may affect our project




tutorial_project - Godot Engine
Scene Project Debug Editor Help
2D 3D Script AssetLib
Here we can switch between **2D and 3D view**, the **Script IDE** and the **Asset Library**
Forward+
Scene Import
[empty]
Transform View
Inspector Node History
Filter: name, t:type,
Create Root Node:
2D Scene
3D Scene
User Interface
Other Node
Perspective
Filter Properties
Here we create the **nodes** - they're the pieces that form our game
The **Inspector** allows us to manage the properties of a selected node
The **Node** window is where we can find the signals of a selected node
FileSystem
res://
Filter Files
Favorites:
res://
icon.svg
In the **FileSystem** we find the files and folders of our project. It's important to keep it tidy!
The **Viewport** is where we can visualize the game elements and arrange them
Output Debugger Audio Animation Shader Editor
4.3.stable
**Output** (or Console) is where we can see the things we print and also any possible errors the game throws
19

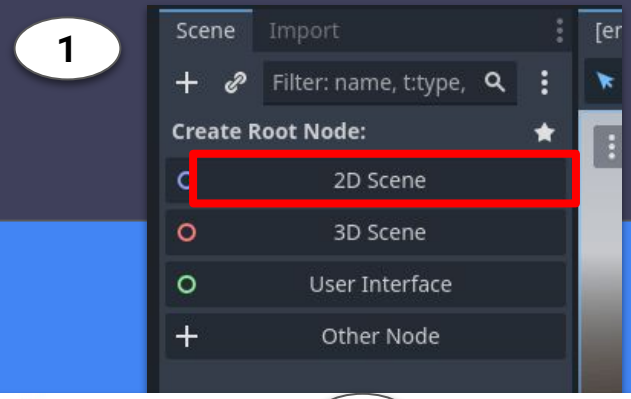
# Player: Creation

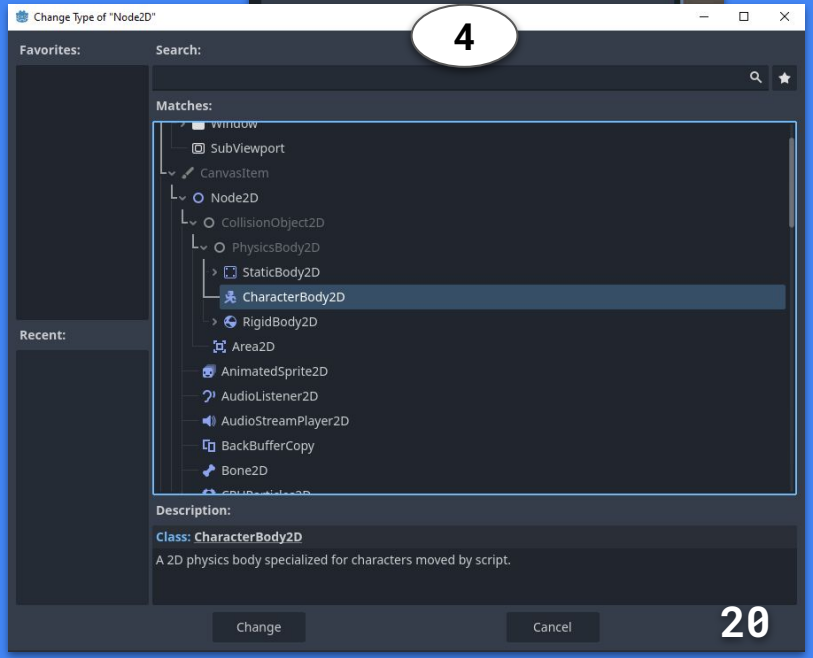
1. In the “Scene” window we choose the option “2D Scene” since we’re making a 2D game.

2. We save (Ctrl + S) the newly created node with the name “player.tscn” in our scenes folder

3. We double-click, or right click on the node and select “Rename”, to change its name to “player”

4. We right-click on the node and select “Change type…”, where we choose the option “CharacterBody2D”



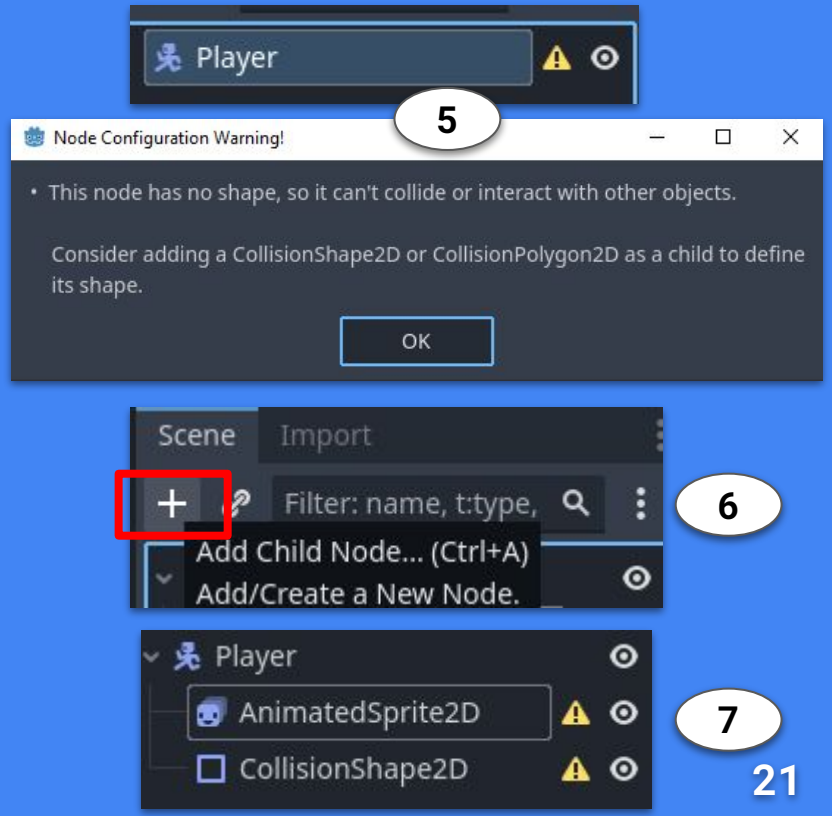



# Player: Adding nodes

5. You may notice the warning sign on the node. If we click on it, it will tell us the node needs a shape to be able to collide with others, so let's add it!

6. We click on + (or right-click on the node and select "Add child node...") to add a CollisionShape2D and AnimatedSprite2D (we want our dragon to fly!)

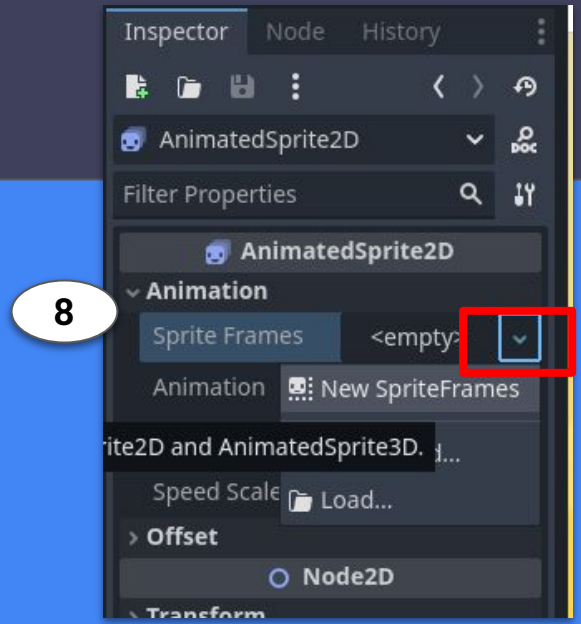
7. Oh, no, they have warnings too! AnimatedSprite2D needs a SpritesFrames resource and CollisionShape2D needs a shape.

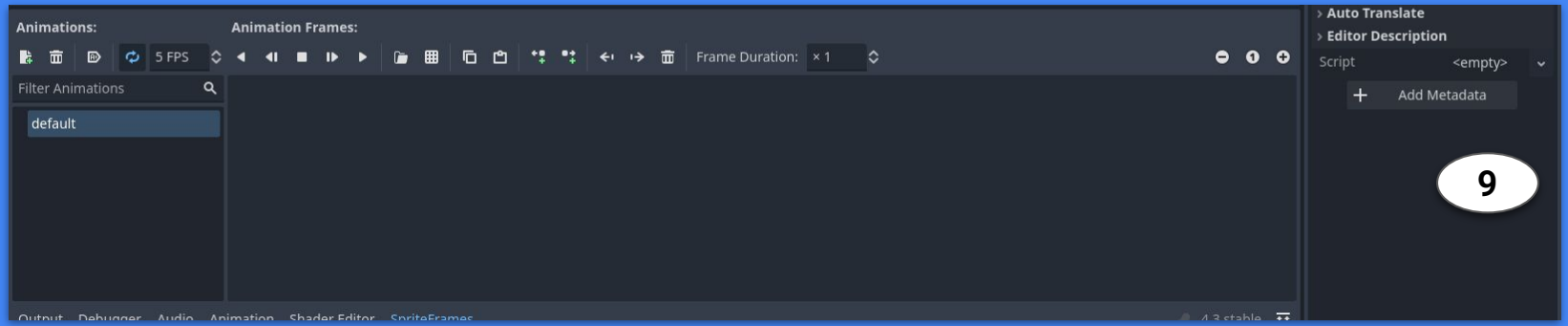


# Player: AnimatedSprite2D

8. We will focus on animating our dragon first since then it will be easier to set the collision. We select the AnimatedSprite2D we just created and then click on the arrow next to Animation > SpriteFrames on the Inspector and choose “New SpriteFrames”

9. We click on the newly created spriteframe and a window called “Animation Frames” will open in the area below:



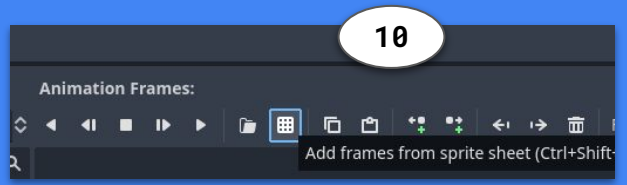



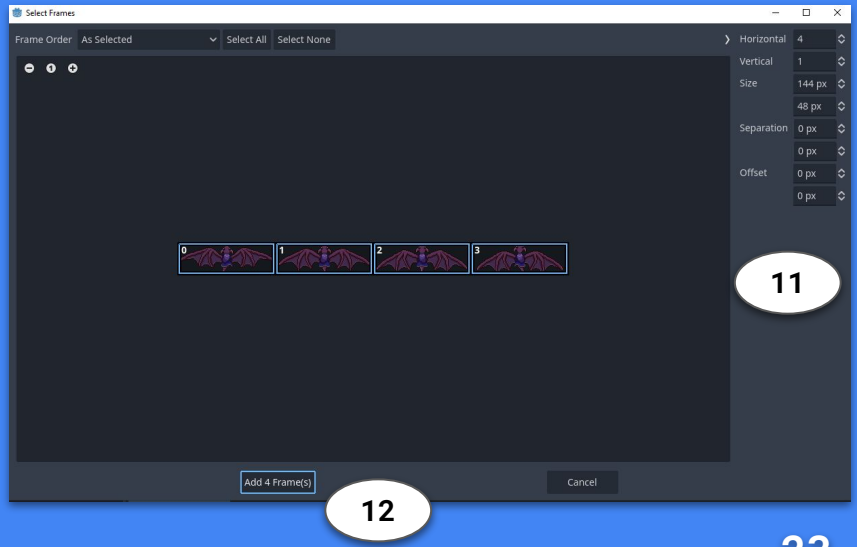
# Player: Building the animation

10. We click on the grid icon (“Add frames from spritesheet”). A file selection window will open. We go to assets > art and select “dragon.png”

11. As you can see, this is a picture that contains all the frames of the flying dragon. We have to set the values to Horizontal = 4 and Vertical = 1 so the image is correctly splitted.

12. Then we click on all the frames in order (you will see numbers on them) and then click on “Add 4 frames”.

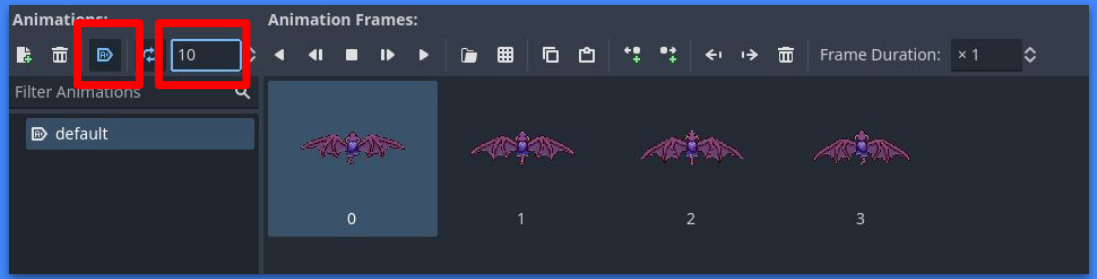


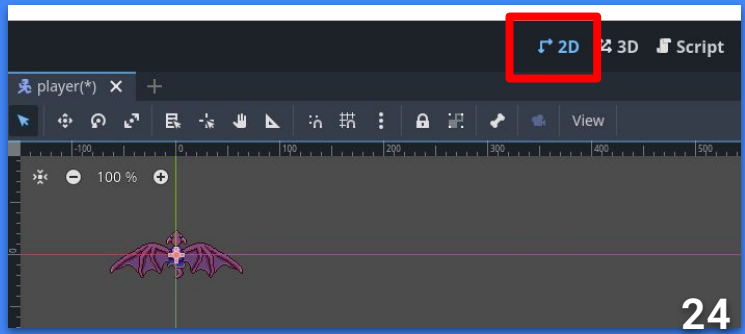



# Player: Animation details

11. Now on the Animation Frames window we can manage our dragon animation: we could change the animation name, change the speed, rearrange the frames, duplicate or add new frames, etc. However, for now we only need to change the speed to 10 FPS and click on “Autoplay on Load”

12. If we go to the 2D tab on the viewport, we will see our dragon flying!

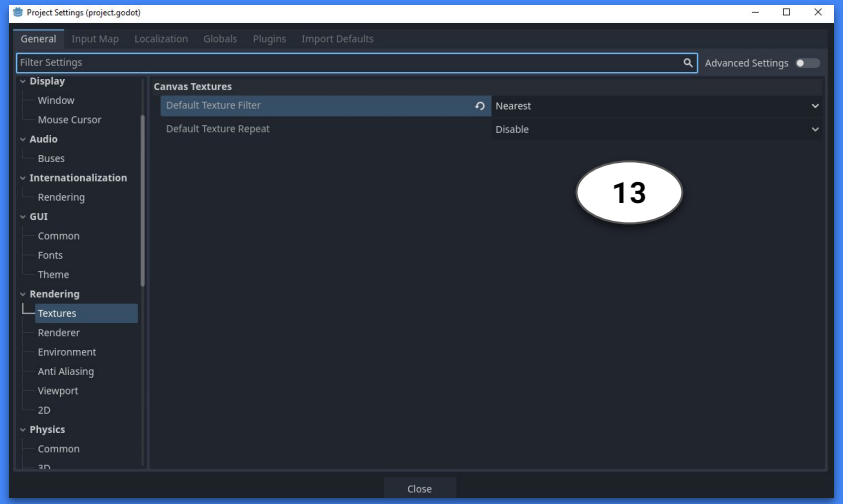


# Player: Making the sprite look crispy

Did you notice that the dragon looks blurry? We can change the settings so it looks better.

13. At the top bar, we go to Project > Project Settings > Rendering > Textures and on “Default texture setting” we choose “Nearest”. Then simply click “Close”. Now the pixels look defined!



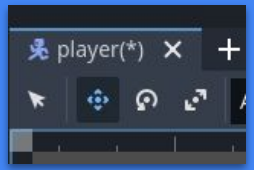

# Player: CollisionShape2D

Now that we can see our dragon, we can add a collision shape. We click on the node “CollisionShape2D” on the scene tree and then, on the inspector, we can choose a new shape on CollisionShape2D > Shape.

There are several types of shape we can choose: circle, capsule, rectangle, polygon… The election is a matter of design: do we want the wings to be a colliding spot or just the body?

14. For this game we will choose the option “New Capsule Shape 2D” and adjust it to the dragon body. We could also have separate colliders for the wings, but for now we are going to assume the only weak points this dragon has are her head and body.

We can use the Transform icons to modify the collider or change the values on the inspector

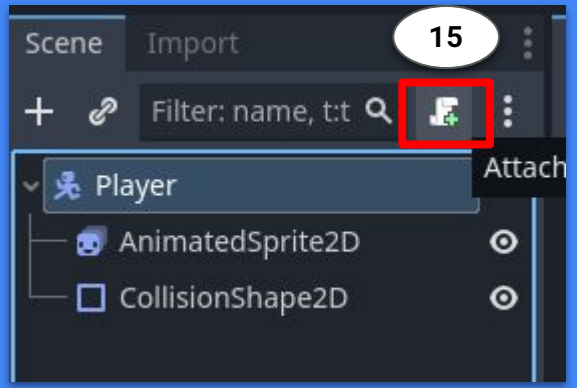


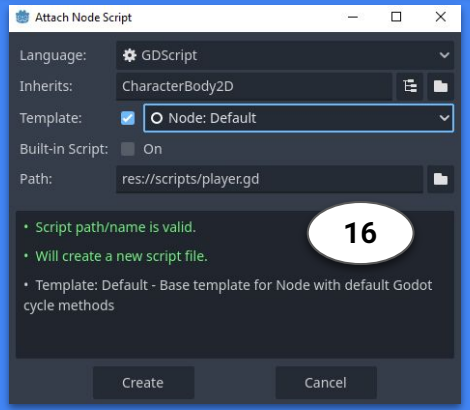

# Player: Movement

Now let's make the dragon move with user input!

15. We click on the Player node (top of the scene tree) and then on the icon "Attach a new or existing script to the node". We can also do this by right-clicking on the node and selecting the option "Attach script…"

16. We will save the new script on the "scripts" folder with the name "player.gd". Then we choose the template "Node: Default" and click on "Create".



# Player: Coding the movement

```gdscript
class_name Player extends CharacterBody2D


@export var speed : float = 100.0 # speed at which the dragon moves
var direction : Vector2 # stores the current direction


# Get the direction vector depending on player input (4 arrows)
func _input(event: InputEvent) -> void:
	direction = Input.get_vector("ui_left", "ui_right", "ui_up", "ui_down")


# Calculates the position depending on direction, speed and delta time
func _physics_process(delta: float) -> void:
	position += direction * speed * delta
```

Now the IDE will open, where we can write the code to make the dragon move.

Copy this code on the script.

The label **@export** makes the variable speed visible on the inspector so we can change it easily when testing the game

The function **_input** captures the input by the player. Here, the 4 arrow directions.

The function **_physics_process** is called a fixed amount of times each second to make physics calculations, like in this case, the player's movement. **Delta** is the amount of time that has passed between each call

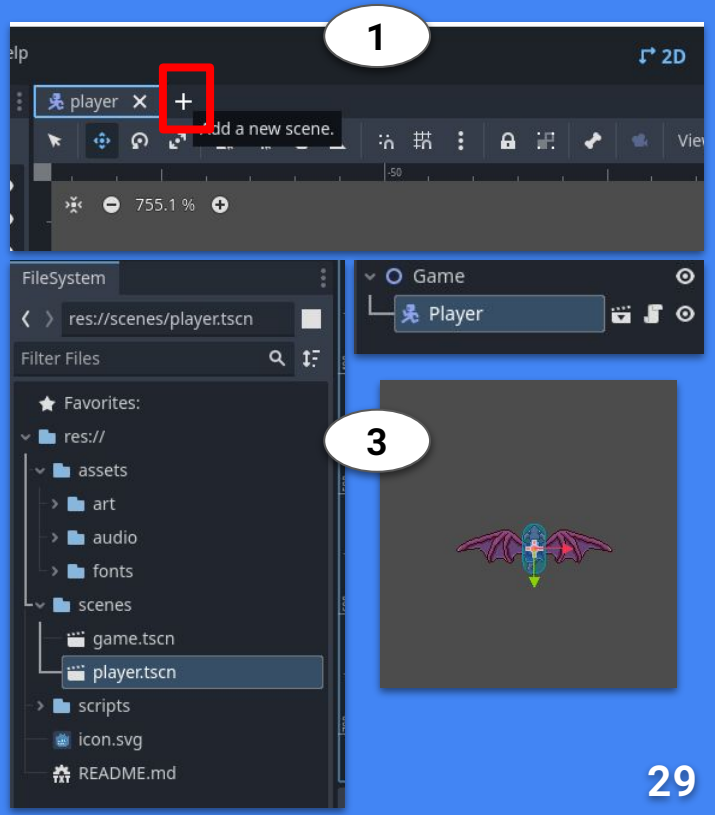
# Level: Creation and Player

You want to try to move the dragon, right? Let's create a level where she can fly.

1. First we choose the 2D tab again so we can see the level and its elements. Then we click on (+) ("Add a new scene") and choose 2D Scene.

2. We change the node name to "Game" and save the scene at the "scenes" folder with the name "game.tscn".

3. On the FileSystem window, we go to "scenes" and drag the Player into the node's tree system or into the viewport. Now the Player should be a child node of "Game" in the tree system.

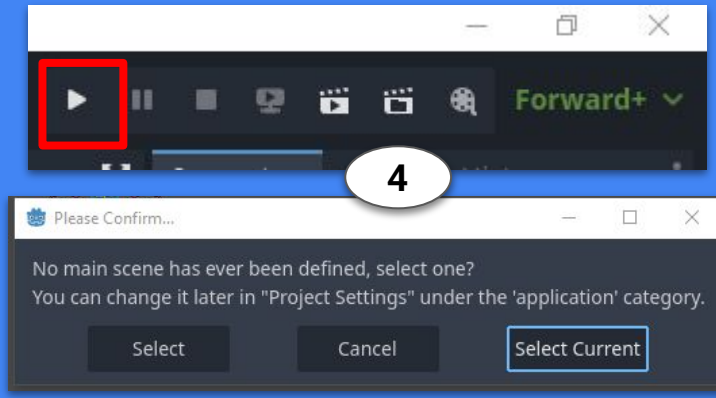


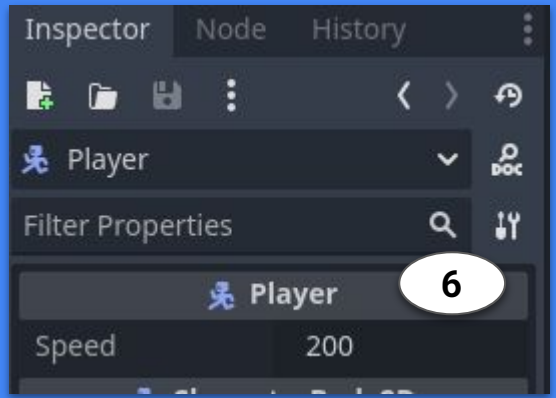
# Level: Testing the Player's movement

4. In order to test the scene, we click on the "play" icon at the top right. A pop-up message will tell us we have not defined a main scene, so we choose "Select Current".

5. The game executes and now you should be able to move the dragon across the empty level. You may want to change her speed and make her move faster.

6. You can do so by modifying the speed value on the Player's inspector.

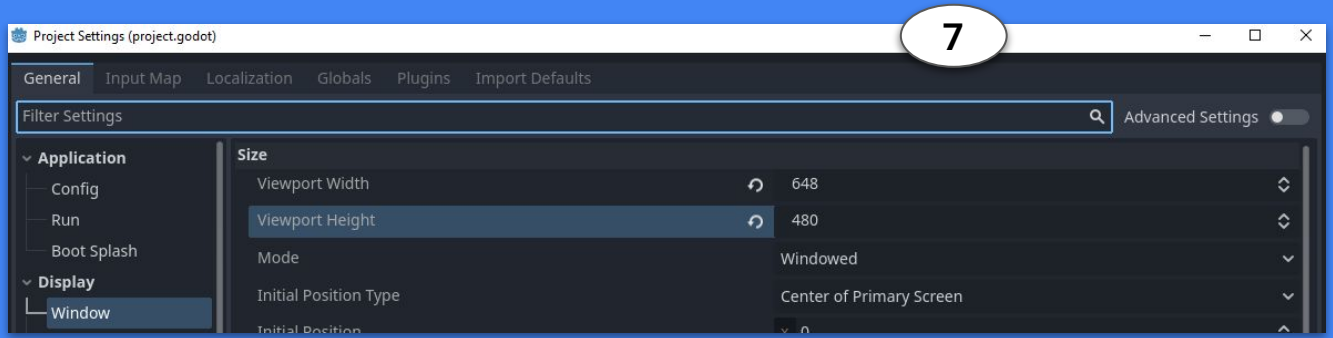


# Level: Adjusting the viewport size

Now that we have our dragon on the level we're going to add a background, but first let's adjust the viewport size.

7. In order to do so, we should go to Project > Project Settings > General > Window and set the Viewport Width to 648 and the Viewport Height to 480. This means the game window will be smaller, which is a good fit for our pixel art game, especially if we were to export it to web.



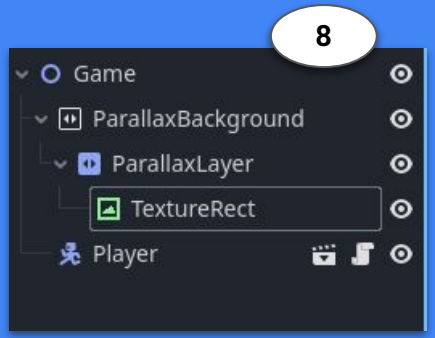
# Level: Adding a parallax background

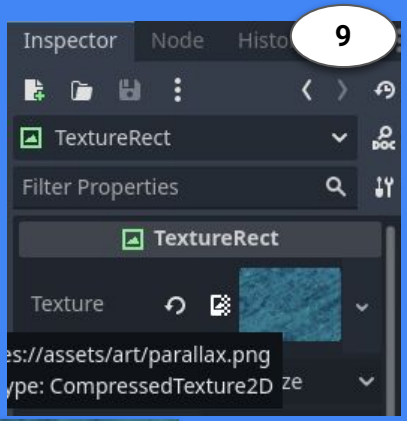
To create the effect that the dragon is flying over a field, we're going to make an endless scrolling background, also called parallax.

8. We will need to add the following nodes to the scene: ParallaxBackground, ParallaxLayer and TextureRect. They should be ordered like in the associated screenshot.

9. We click on the TextureRect node and then on Texture and choose the file called "parallax.png" from the art folder.

10. Now we should see the dragon on a field. Let's make it scroll.




# Level: Making the parallax scroll

There are multiple approaches to this. We could use a camera that follows the player or limit the player to the viewport size and move the parallax from a script.

In the original version of this game, I chose the script option, but to keep it simple we're going to use a camera here.

13. We add a Camera2D node as a child of Player.

14. Then, we select the node ParallaxLayer and modify the Mirroring values like this: x = 648, y = 480 (the viewport size). By doing this, we should now see the background duplicated on both axis.

15. Let's execute the game. Now the dragon is flying over an endless field!

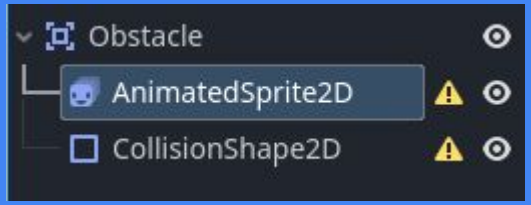
# Obstacles: Creation

Let's make the game more interesting by adding some danger! We're going to create moving obstacles that we should avoid.

1. We create a new Node2D and change its type to Area2D. It's a kind of collision node that detects collisions but allows elements to go through it.

2. We rename it as Obstacle and then we add two children nodes: AnimatedSprite2D and CollisionShape2D.

3. We save the new scene in the scenes folder as "obstacle.tscn"




This is very similar to the process of creating the **Player**, except it's an Area2D.

You can also check how you created the **level** scene (p.29) if you don't remember how to create a new scene.

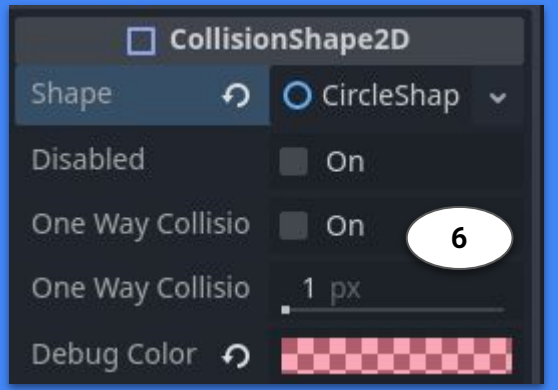
# Obstacles: Sprite and collision

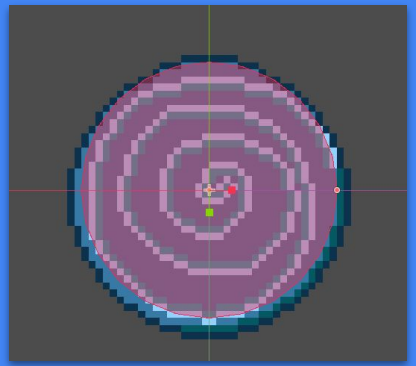
Now let's add the animated sprite and the collision shape.

4. Like we did with the Player, for the AnimatedSprite2D, we create a new SpriteFrame, click on the grid icon on the Animation Frames window, select "spiral.png" on the art folder and set Horizontal = 4 and Vertical = 1. Then we click on all the frames in order and finally on "Add 4 Frame(s)".

5. We click the Autoplay button and change FPS to 10.

6. Since the obstacle is round, we choose a new CicleShape for the CollisionShape2D's shape. We adjust the size so it matches the obstacle's size. We can change the Debug color so the shape is easier to see.

# Obstacles: Movement

7. We create a script named “obstacle.gd” and save it on the scripts folder. We’re going to make the obstacles move with the following code:

```
class_name Obstacle extends Area2D

var direction : Vector2
var speed : float

# Called when the node enters the scene tree for the first time.
func _ready() -> void:
	speed = randf_range(100, 300) # Random speed to add variety


# Calculates position based on speed, direction and delta time
func _physics_process(delta: float) -> void:
	position += direction * speed * delta
```

Movement is calculated the same way we did with **Player**, but we’re going to define the **direction** value externally, as you will see on the next slides.

# Obstacles: Creating a GameManager

8. We create a new script, this time attached to the Game node in the Game scene. We name it “game_manager.gd” and save it in the scripts folder. This script will be in charge of the game, including spawning the obstacles.

9. We right click on the Player in the Game scene and select “Access as Unique Name”. A % symbol will appear on its name.

10. Now we click again on Player and press control while we drop it to the Game Manager script. This will automatically add a new line on the script with a reference to the Player (we could also write this manually)

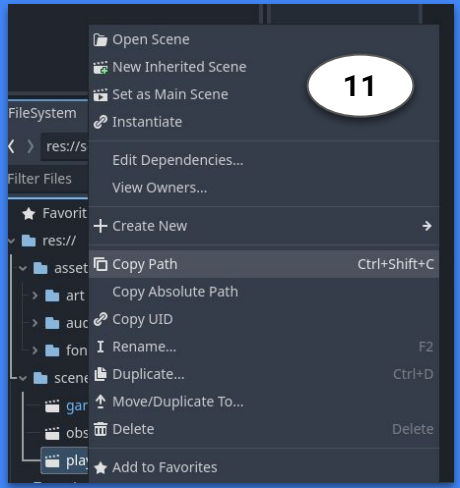
11. We also will need the path where the obstacle scene is saved. We can get it by right-clicking the scene on the FileSystem and choosing the option “Copy Path”


9

```
@onready var player: Player = %Player
```

10

**@onready** is a label that makes a variable to be initialized when the _ready function executes (at the beginning of a node’s life)



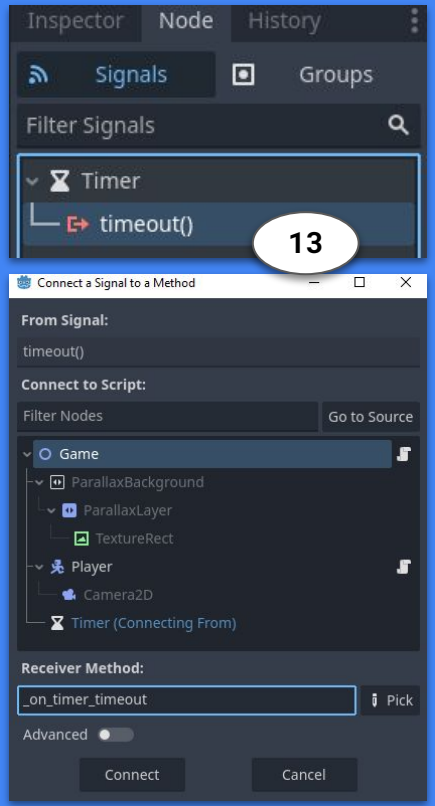

11

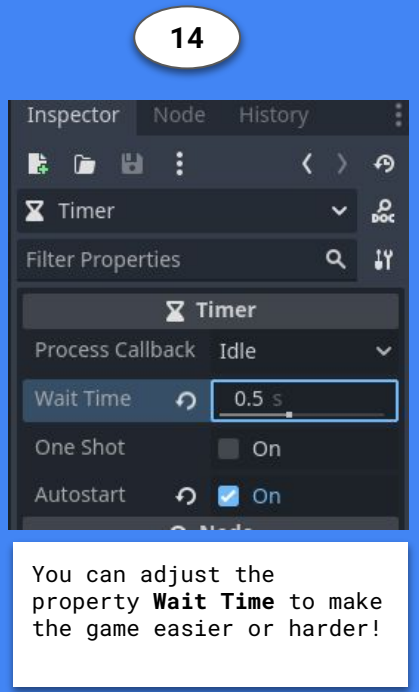
# Obstacles: Spawn timer

12. We create a Timer node inside the Game scene.

13. Then we select this node, click on the Node tag and on the timeout() signal. A window to connect the signal to the game manager will appear. We choose “Connect” and a new function will be created on the script.

14. We configure the Timer on the Inspector tag. Wait time will determine how many seconds have to pass to spawn a new obstacle. Autostart means the Timer activates automatically.



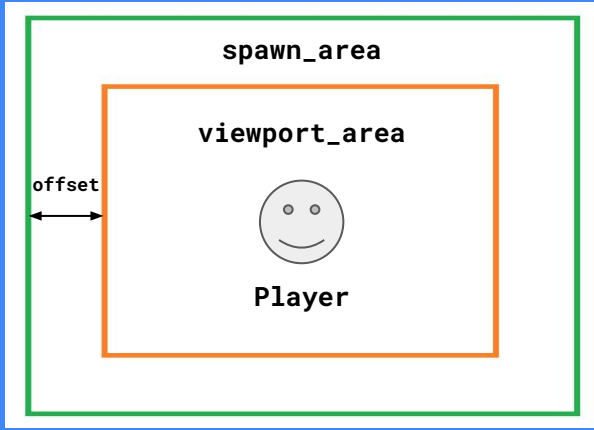




You can adjust the property **Wait Time** to make the game easier or harder!

# Obstacles: Calculating the spawn area

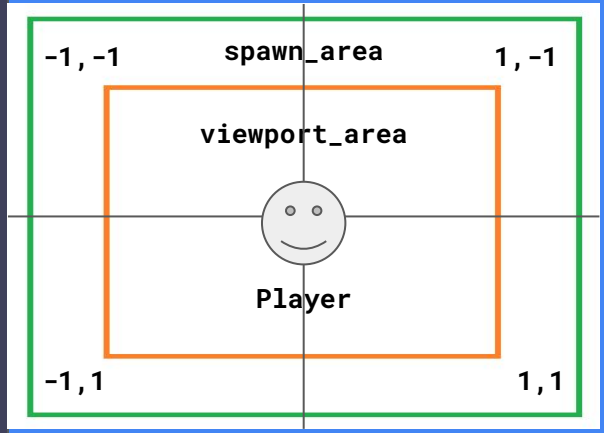
```gdscript
class_name GameManager extends Node2D

var viewport_area : Vector2 # This will store the viewport size
var spawn_area : Vector2 # Area where obstacles will spawn
var offset : float = 50.0 # An extra we add to the viewport size

# A reference to the Player
@onready var player: Player = %Player

# Preloads the Obstacle scene
const OBSTACLE := preload("res://scenes/obstacle.tscn")

# Called when the node enters the scene tree for the first time.
func _ready() -> void:
	# Gets the viewport size (The orange rectangle, now 648 x 480)
	# It's better to get it like this because
	# we might want change the viewport size in the future!
	viewport_area = get_viewport_rect().size

	# The green rectangle
	var max_area = Vector2(
		viewport_area.x + offset,
		viewport_area.y + offset)

	# The area between the green and orange rectangles
	spawn_area = max_area - viewport_area
```

Now we have all the ingredients needed to spawn obstacles every second!
They will appear on a random position outside the viewport and go towards the Player



# Obstacles: Position and direction

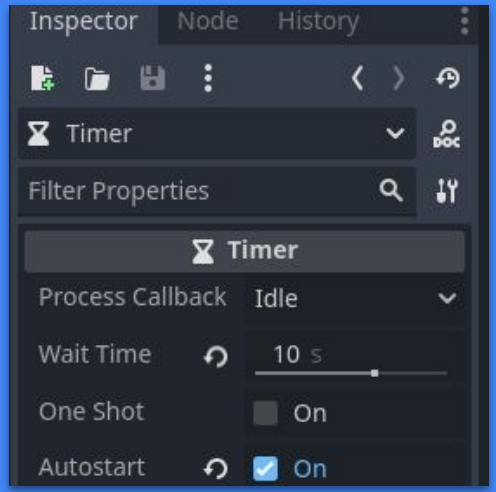
```gdscript
# Called every time the Timer reaches timeout
func _on_timer_timeout() -> void:
	var obstacle := OBSTACLE.instantiate()

	# The obstacle spawns in a random position outside the viewport
	# Here we get a random direction in one of the 4 quadrants
	var random_vec2 := Vector2(randf_range(-1, 1), randf_range(-1, 1))

	# Now we calculate the spawn position
	obstacle.position = Vector2(
		player.position.x + viewport_area.x/2 * sign(random_vec2.x)
			+ spawn_area.x/2 * random_vec2.x,
		player.position.y + viewport_area.y/2 * sign(random_vec2.y)
			+ spawn_area.y/2 * random_vec2.y)

	# The obstacle moves towards the Player!
	obstacle.direction = obstacle.position.direction_to(player.position)

	# We add the new obstacle to the Game scene
	add_child(obstacle)
```




From the Player's position, we add or subtract half the viewport_area (depending on the quadrant) and add or subtract half the spawn_area multiplied by the random direction (random_vec2), so the obstacle can spawn at any point of the spawn_area's randomly chosen quadrant

# Obstacles: Destroying the obstacles

Obstacles should be destroyed after some time so we don't run out of resources. To do so, we create a Timer node inside the Obstacle scene and connect its timeout signal to the Obstacle script. We can set the time to 10 seconds or so. Remember to check Autostart too!



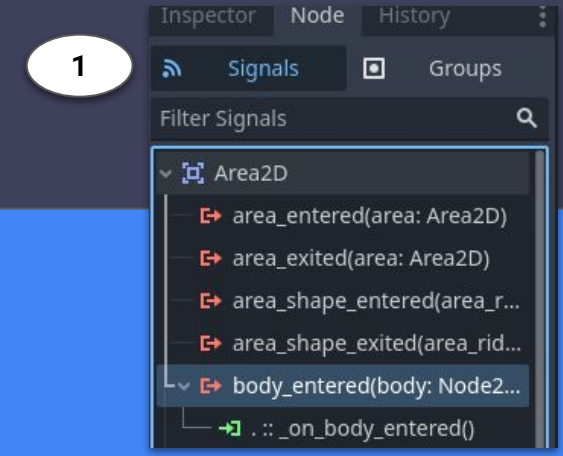

```
# Destroys the obstacle when timeout is reached
func _on_timer_timeout() -> void:
    queue_free()
```

# Game over

Let's now make that when the player touches an obstacle, it's game over.

1. We go to the Obstacle scene, click on the Obstacle node, go to the Node tab and choose the signal "body_entered", which we connect to the obstacle script.

2. We make the game scene reload. Optionally we can add a print to show a message on the console.





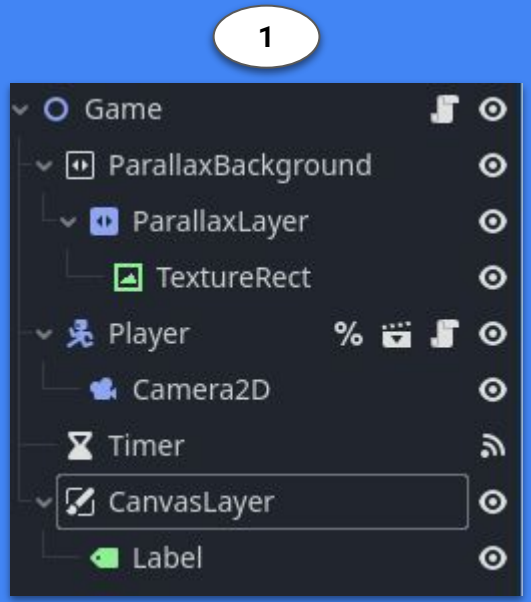
```
# Called when a body (like the Player) enters the Obstacle's collision area
func _on_body_entered(body: Node2D) -> void:
    print("Game over!")
    get_tree().reload_current_scene()
```

# Time

To make the game more interesting, we can add a Label to display the game time. When the player dies, time will reset to 00:00.

1. On the Game Scene we add a Canvas Layer node (used for UI elements) and then a Label node (used for text) as a child of Canvas Layer.

2. Like we did with the Player, we drop the Label node to the Game Manager script while holding Control to automatically create a reference.




```
@onready var player: Player = %Player # A reference to the Player
@onready var label: Label = $CanvasLayer/Label # Time label
```

2

# Time: Counting

There are several ways to count the time that has passed (timers, _process). Here we are going to use _process and delta time.

3. At the top of the script we define two variables, time, which will store all the time that has passed, and stored_delta, which will store the elapsed delta (time between frames).

4. Then we make the calculations on _process, which is called every frame.

```
# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta: float) -> void:
    stored_delta += delta #delta are milliseconds
    if stored_delta >= 1.0: #one second
        time += 1
        stored_delta = 0.0
        # Time should have a 00:00 format
        # Time / 60 = minutes
        # Time % 60 = seconds (%  gets the division remainder)
        label.set_text("%02d:%02d" % [time / 60, time % 60])
```

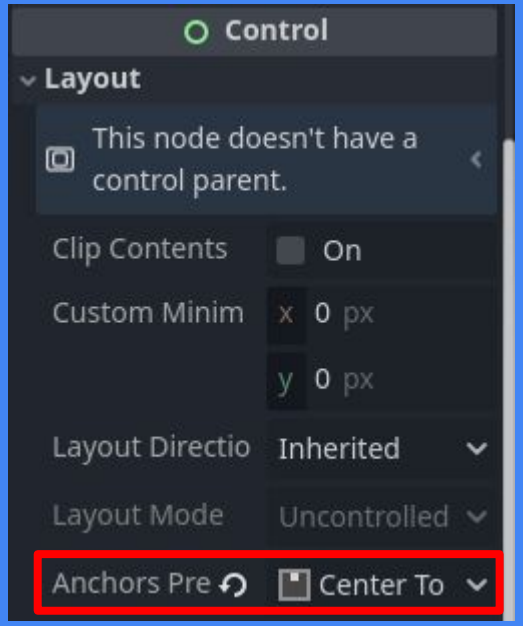
4

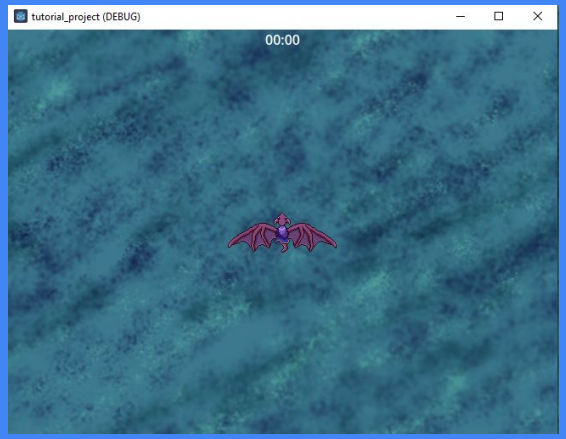
3

```
var time : int
var stored_delta : float
```

# Time: Displaying

5. Finally, we can adjust the anchors on the inspector so the Label is displayed at the center top. This option is found by clicking Label, going to the Inspector tab, and then going to the Control > Layout section, on “Anchor Presets”.



# Adding a shadow to the dragon

We can also add a shadow to the dragon to add depth to the scene. To do so, we can add a second AnimatedSprite2D to the Player Scene and then replicate what we did with the dragon animation, but this time by using the “dragon_shadow.png” asset.

We can also change its position and scale on Node2D > Transform, or its opacity and color on CanvasItem > Visibility > Modulate


# Music

Finally, let's add some music!

1. We create a new scene. This time we choose the option "Other Node" and then we select the type AudioStreamPlayer.

2. We change the node's name to "Music" and save the scene on the scenes folder as "music.tscn"

3. Now we click on the node and, on the inspector, we choose the song "pufino.mp3" on the Stream property.

4. We check the properties "Autoplay" and "Looping"

5. Finally, we go to Project > Project Settings > Globals and add the scene "music.tscn". By doing this, we prevent the music to start again when the scene reloads after gameover.

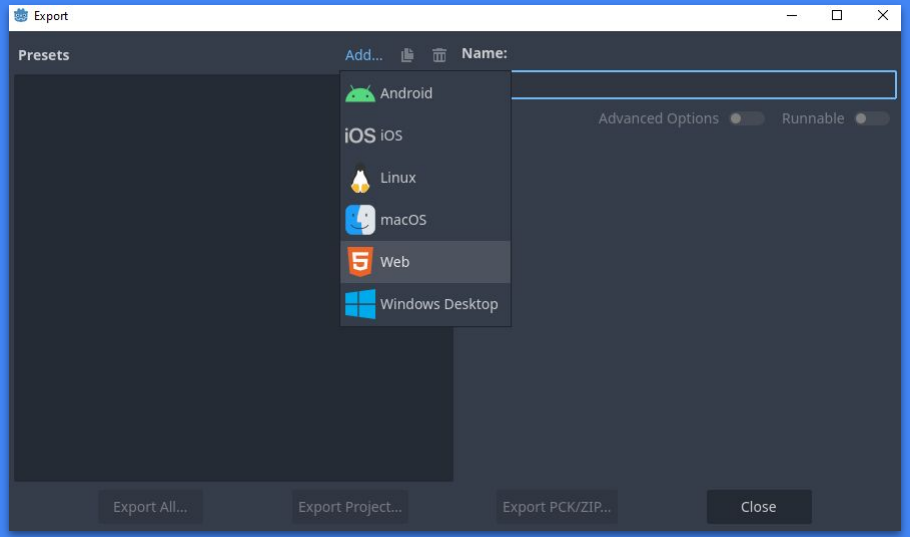
# Exporting the game

To export the game, we can go to Project > Export and there we choose the preset we want by clicking on Add.

The first time we do this, Godot will ask us to install the necessary packages.

I won't cover the details here, but there are lots of tutorials and documentation on this online.

You can easily publish your own games on itch.io.

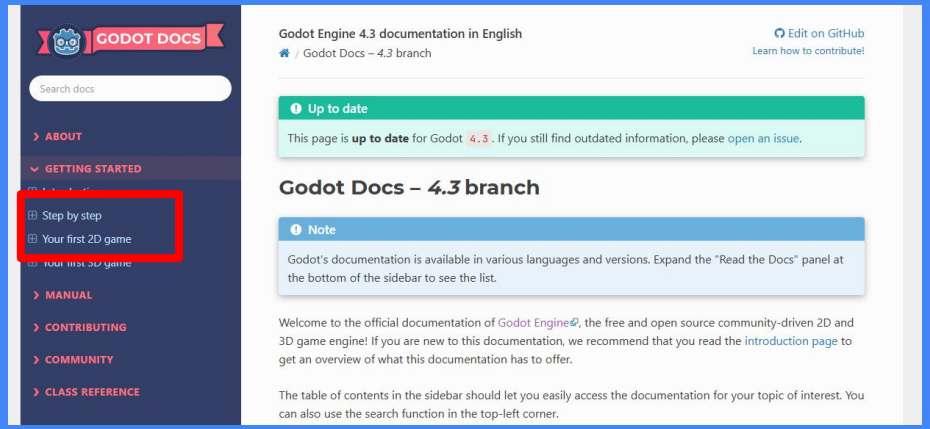


# Recommended tutorials

Beginner tutorial by Rothio Tomé (developer of Wax Heads) where she makes a 2D “walking simulator” game. The tutorial is in Spanish.

Beginner tutorial by Brackeys where he makes a platform game. The tutorial is followed by a guide on how to program in GDScript.

Godot documentation. Here you can find tutorials on how to make 2D and 3D games as well as very complete information on how the engine works, all the existing nodes and their properties and methods…



# Sources of knowledge

- Godot documentation
- Articles
- Videos
- Forums
- GitHub Repositories
- Online communities (Discord, Telegram…)
- Courses (paid & free)

If you don’t know how to do something, you can type “godot 4 how to X” on your searcher and you will find tutorials, people who already faced the same problem…

You can also take a look at the Asset Library. Maybe someone already made what you need, and you can download it for free!

# Credits

Sprites made by me :)

Music track:

- *Metal Is Trash* by Pufino
- Source: https://freetouse.com/music
- Copyright Free Music for Videoic

# Thank you very much!

Do you have any question?

You can also contact me at lyhdyr09enderu@gmail.com