

This is a compilation of small tutorials that are useful for beginners nonetheless!

Contents

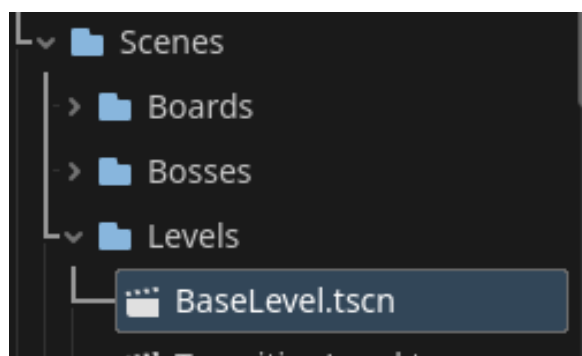
How to create a new level.....	1
How to create a new enemy.....	8

How to create a new level

First of all, how to create new levels. If you're not a beginner, you'll probably know it's simple: you just create an inherited scene from the base level and work from there. But I'm writing this tutorial for people who aren't familiar with Godot yet.

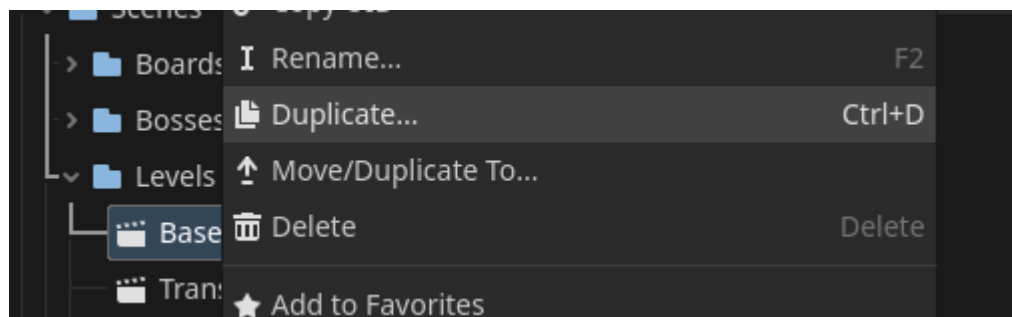
I've already mentioned that you'll need to create an inherited scene from the base level. But where is that "base level" located?

It's in "Scenes/Levels" folder:



Okay, now what's an "inherited scene"?

An inherited scene is basically when you take an existing scene with all its objects/nodes and create a new scene out of it. You might think it's the same as duplicating a scene like this:

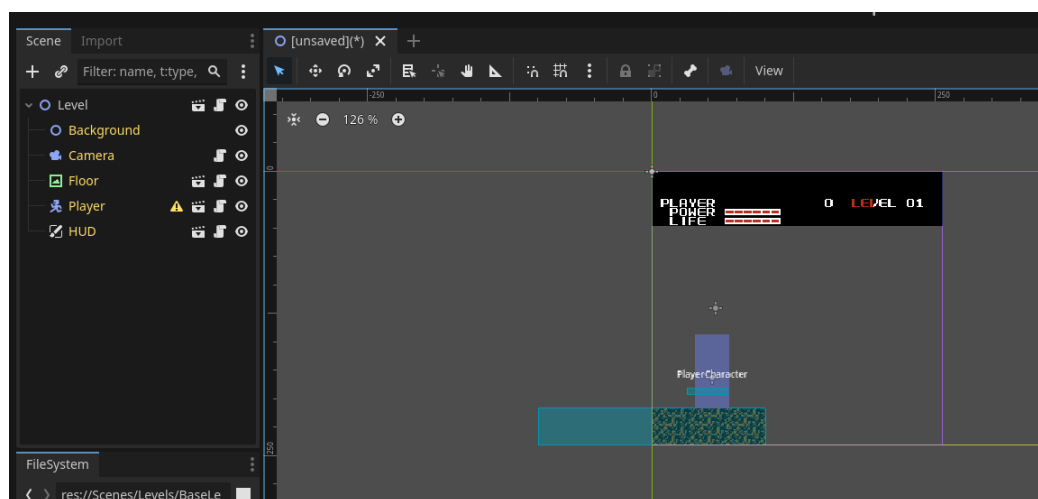
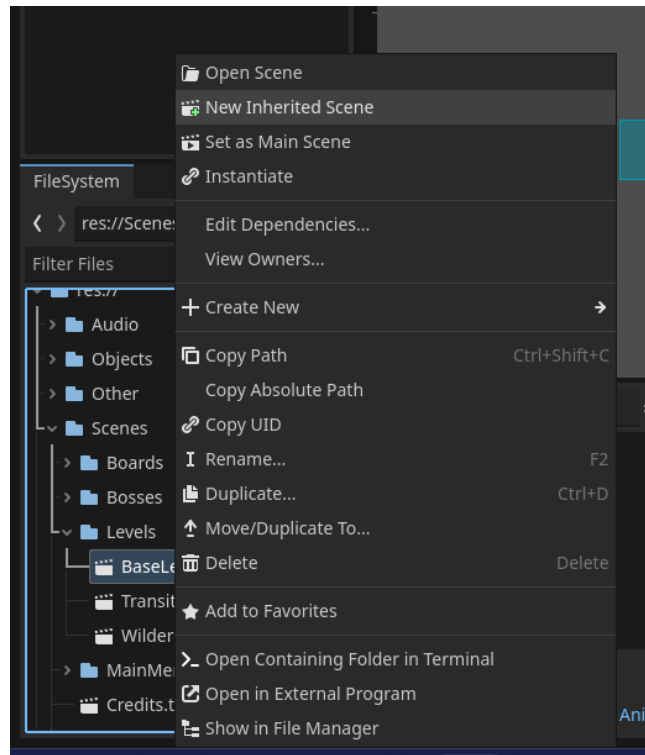


And yes, duplicating a scene and creating an inherited scene from it are pretty similar and you get seemingly the same result at first, but there's an important difference: when you change the base scene, an inherited scene changes accordingly.

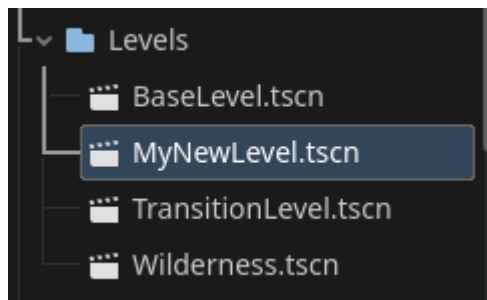
For example, you already have a lot of levels, and you decided to replace the HUD object with your own without affecting the original HUD object scene in the project, you can delete the original HUD object and add your own, and it will be added to all of the levels in your game! How cool is that?

Okay, now I think we can actually start creating a new level.

Right click on the base level scene and select “New Inherited Scene”.

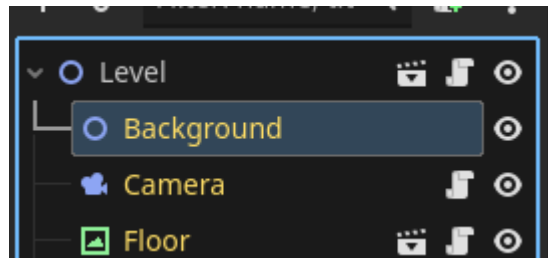


Now save it somewhere in your project, preferably in the same “Scenes/Levels” folder.

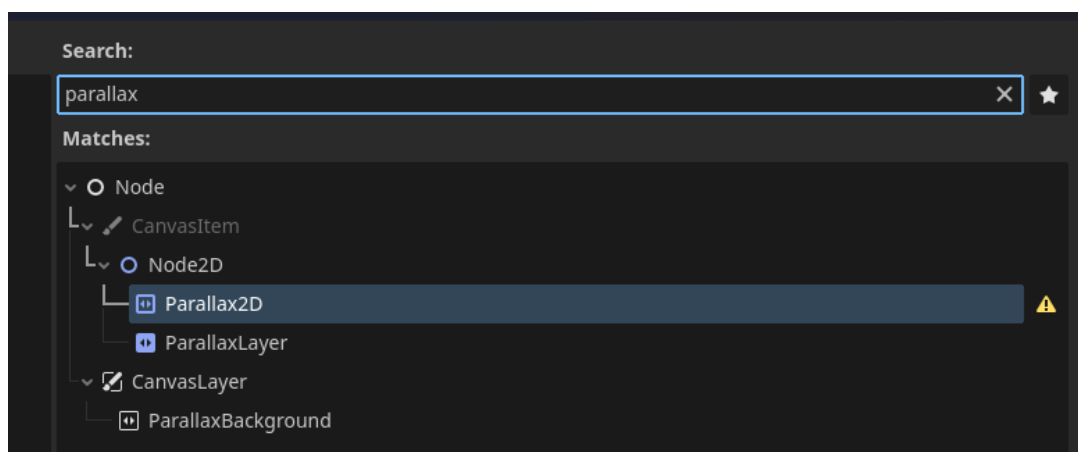


Nice, we have a new empty level. Let's create a simple background.

You may notice that there's a Node2D that's called "Background" in the scene:

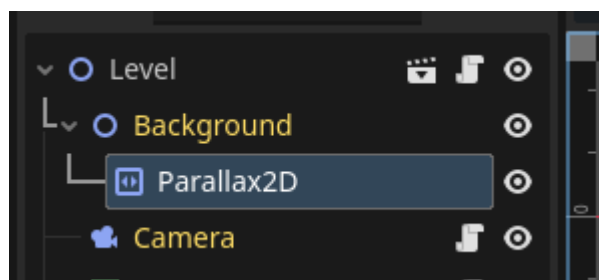


That's where we add our Parallax2D nodes.



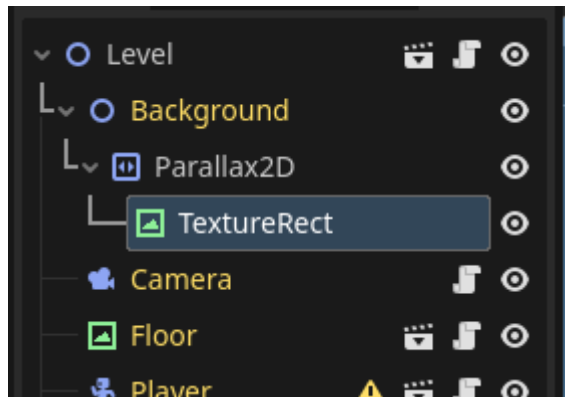
The difference between Parallax2D and ParallaxLayer is that Parallax2D is the newer solution that's more flexible I think, haven't checked those in quite some time but Parallax2D is definitely newer version of ParallaxLayer. And ParallaxBackground was used with ParallaxLayer.

We now have a parallax node:

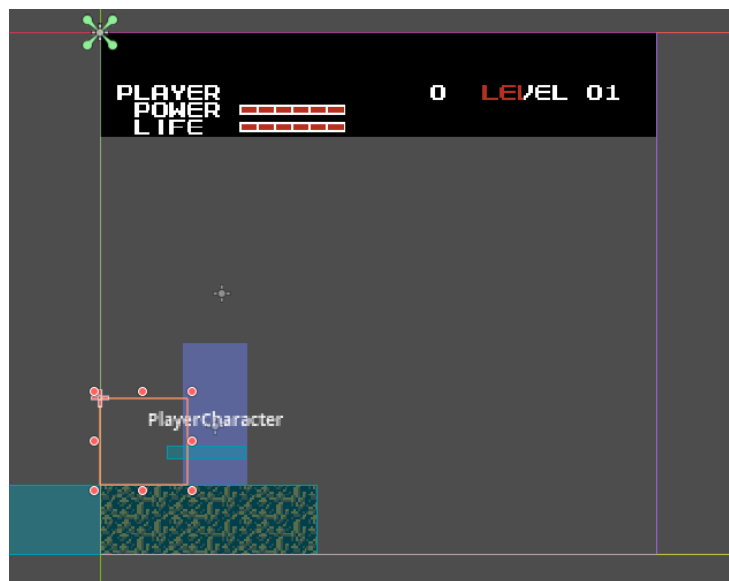


Let's make it show something. You can use any node that can be displayed, I will use TextureRect for this tutorial but something like Sprite2D can also work.

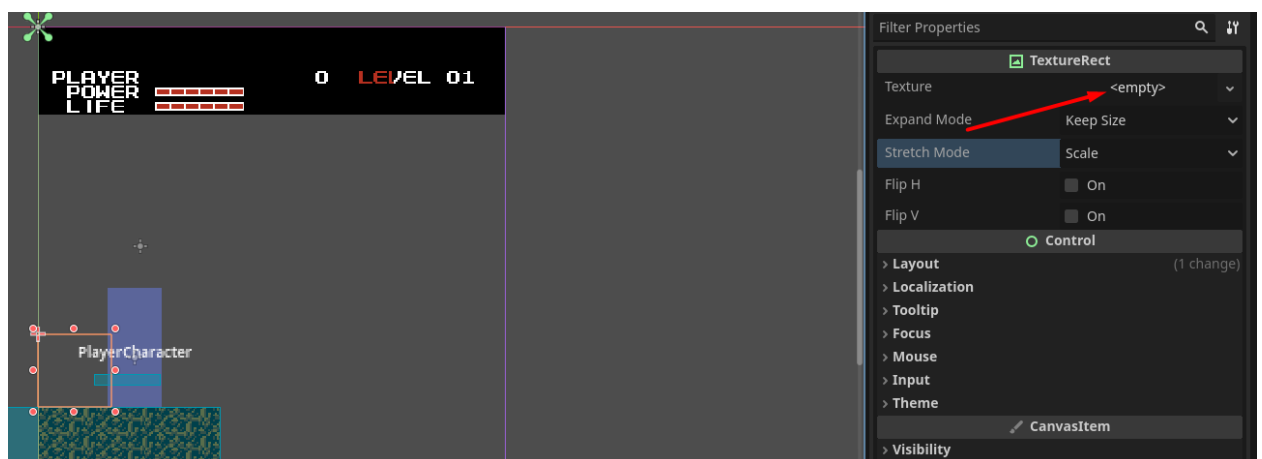
So first, I added the TextureRect node to the Parallax2D node as a child node.



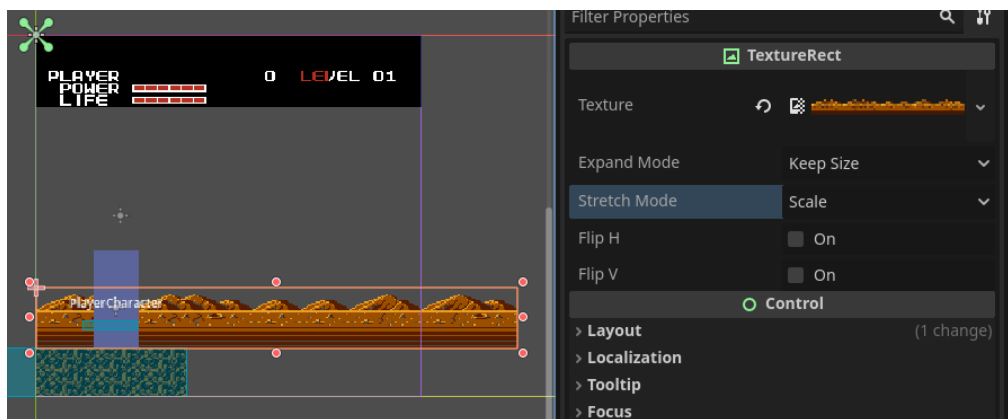
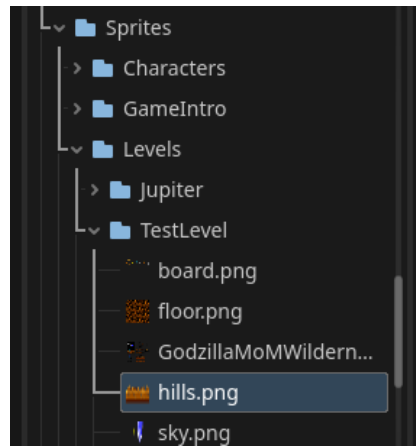
Then I moved the TextureRect node where I want to have my background.



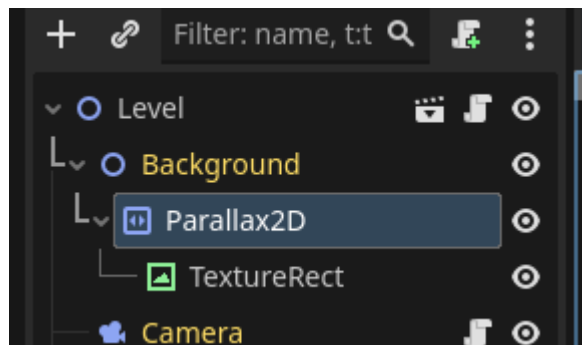
Now I need to add a texture to it.



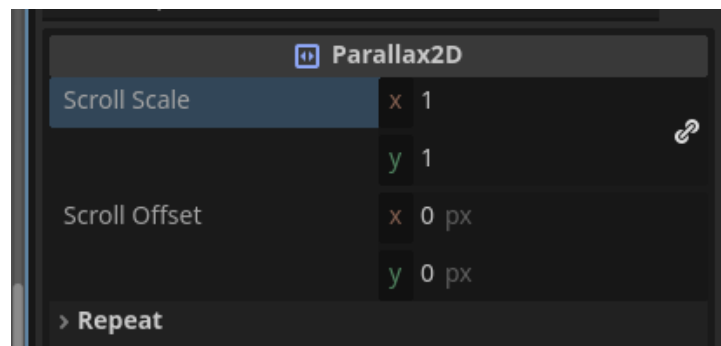
I will drag and drop the existing Wilderness background just as an example.



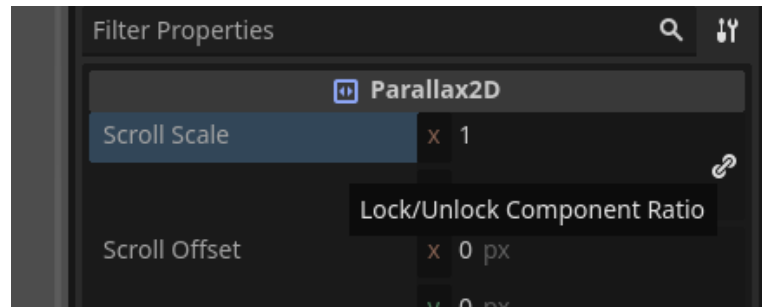
If you test the level now (check out “How to create a new character” tutorial, section “Testing our new character”) you will notice that this background piece has the same speed as the floor and it doesn’t repeat, so we need to change that. Select the Parallax2D node:



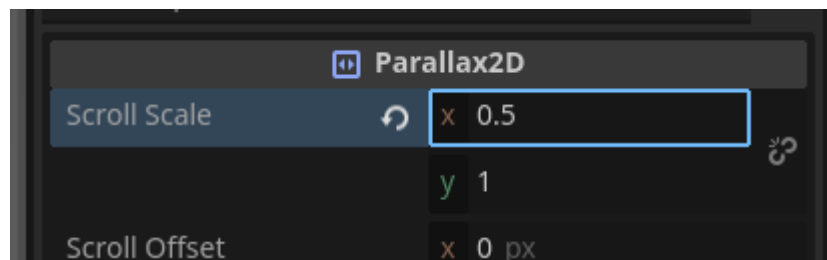
And here we need to change the scroll speed property:



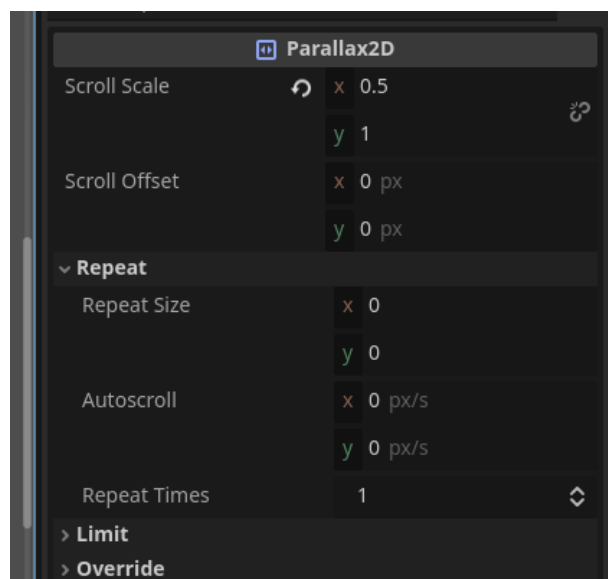
You may also want to untick this:



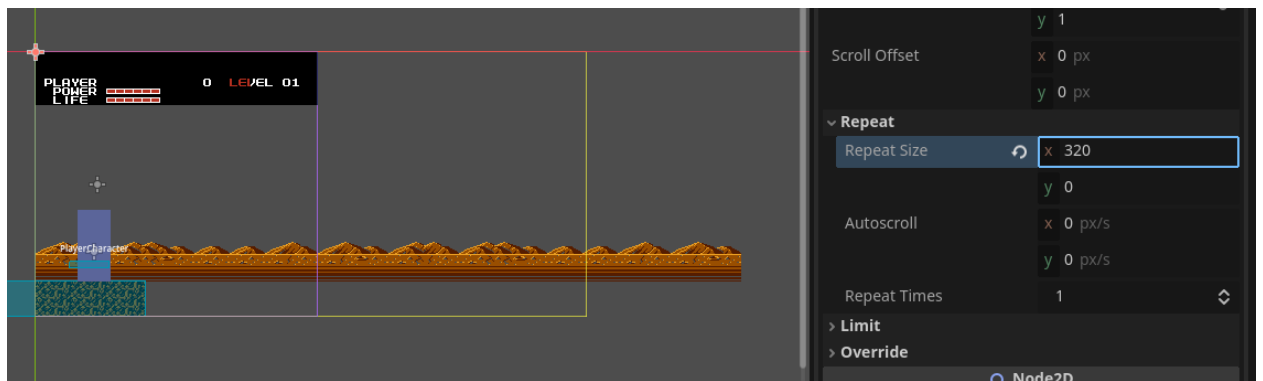
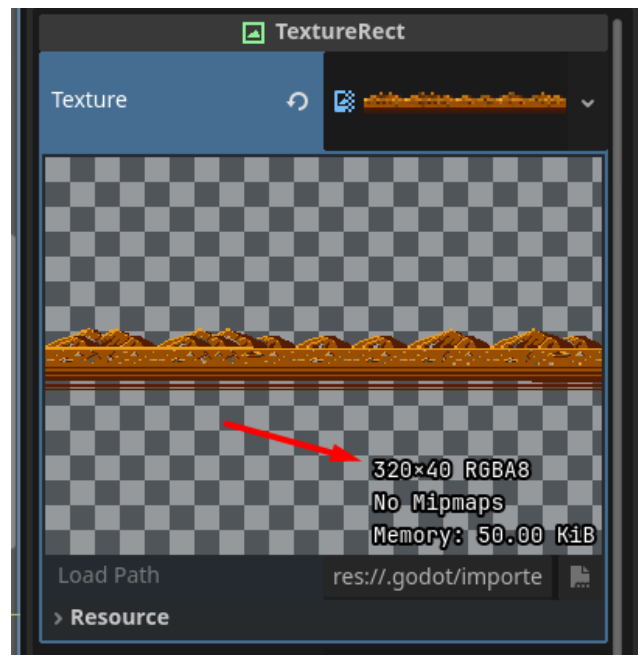
Now you can try something like 0.5 for Scroll Scale X value.



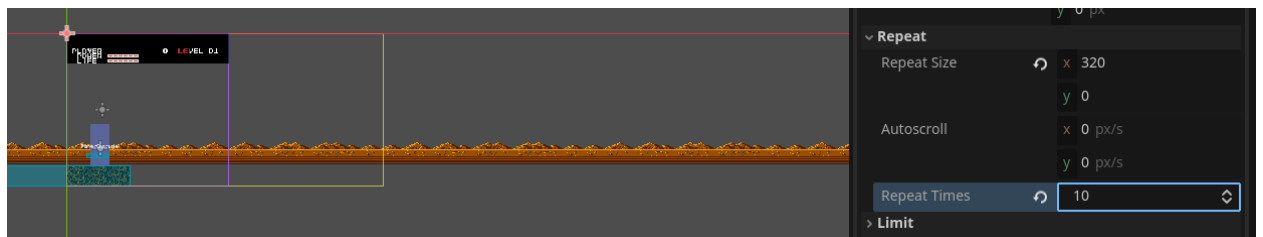
So now the problem about the image not repeating, open the “Repeat” group below and you will notice properties like “Repeat Size” and “Repeat Times”, the ones we need:



Change the Repeat Size X to the width of your image, you can check the image size inside of Godot editor by clicking on the texture inside of the property:



The image was doubled in size! But it's only doubled, it won't repeat past this point. The solution? Increasing "Repeat Times" property!



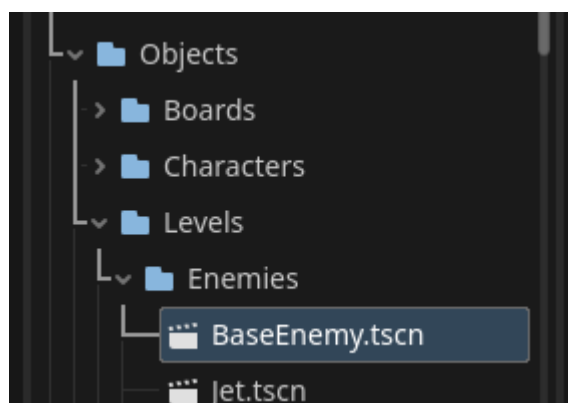
Here we go! Now you can test the level and see if the settings (primarily the Scroll Scale) suit you.



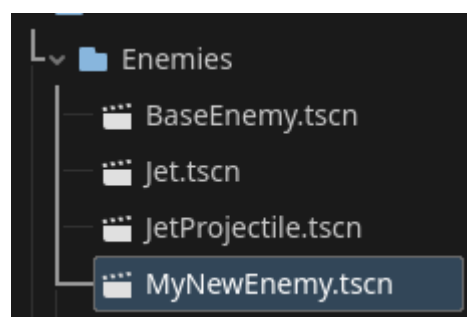
How to create a new enemy

I will create a simple enemy first and then add a small section about projectiles in case you need it!

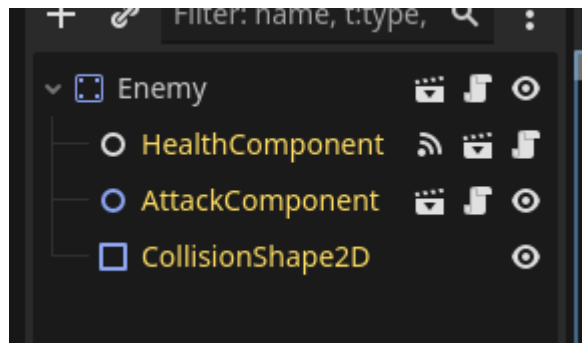
Just like the levels, the enemies also have a base enemy scene:



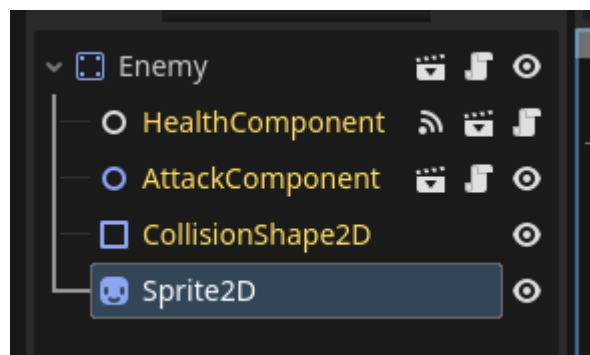
And just like the levels, you can create a new enemy by creating an inherited scene from it.



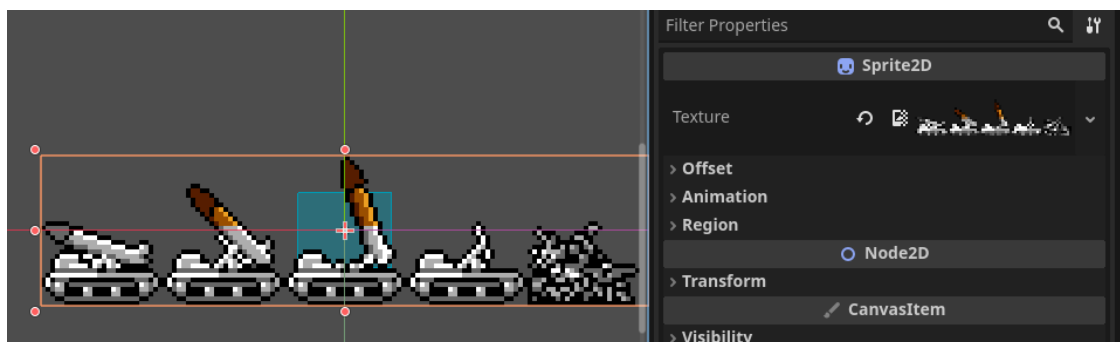
The base enemy scene is pretty barebones, but it has the components required for all enemies:



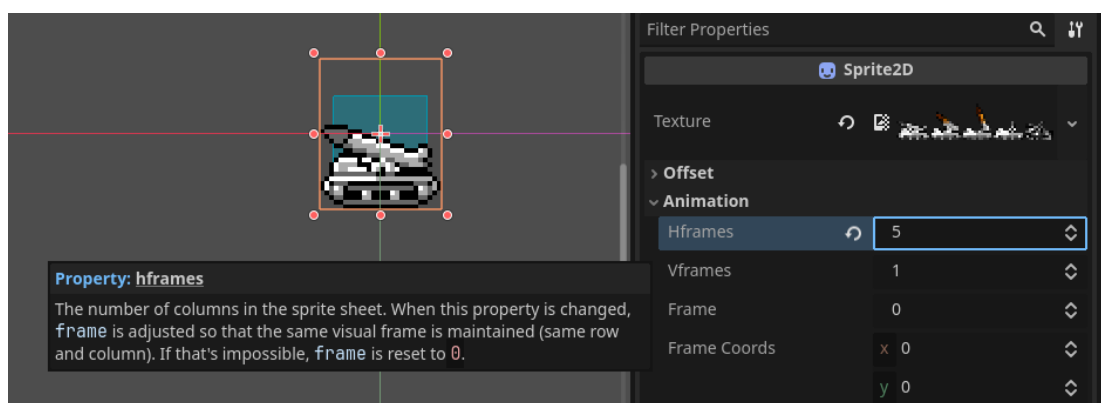
Let's make the enemy visible by adding a sprite to it. You can use Sprite2D, AnimatedSprite2D or other nodes if you want, I will use Sprite2D for this tutorial.



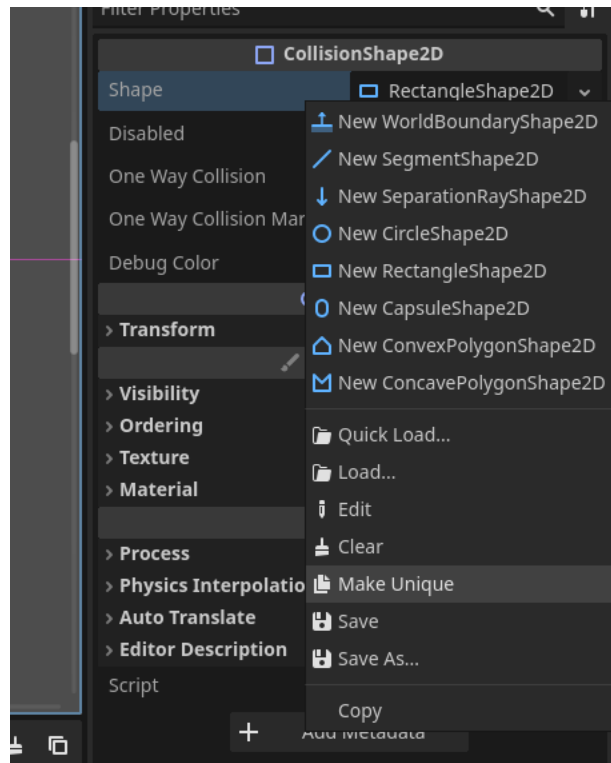
Also, for this tutorial I will use the sprites of an already existing enemy from the framework: the rocket launcher.



I will also need to adjust values like Hframes and Vframes for the spritesheet.



Now the collision shape. First, you need to make it unique, just click on the “CollisionShape2D” node, go to its properties, right click on the shape property and click “Make Unique”:

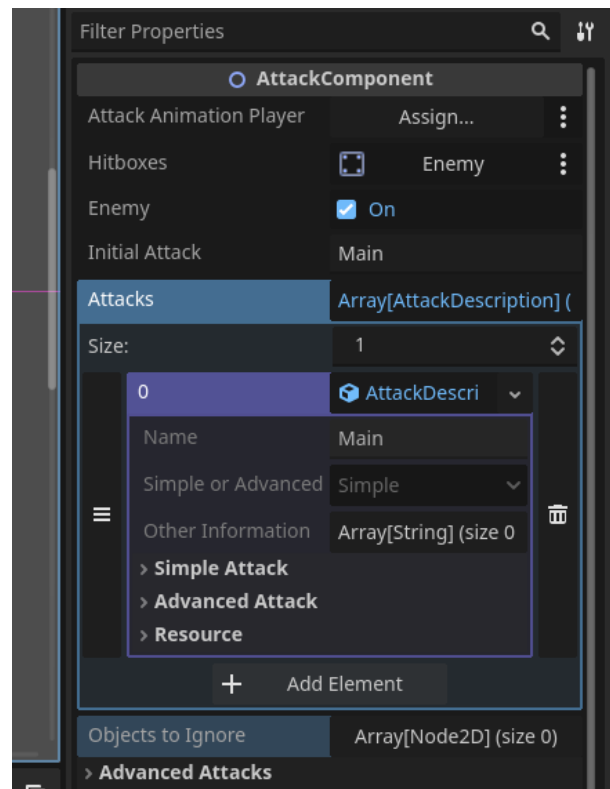


Why is it needed? That's needed in case you want to change the shape but since the inherited scenes also share resources (the shape is a Godot Resource), the changes in the inherited scenes may affect the base scene, so by making the shape unique we basically make a copy of the original shape that will not affect the shape in the base scene.

Now I will reposition the collision shape and/or the sprite (and you can also change the collision shape itself, if needed, it just so happens that the base enemy collision shape matches the rocket launcher).

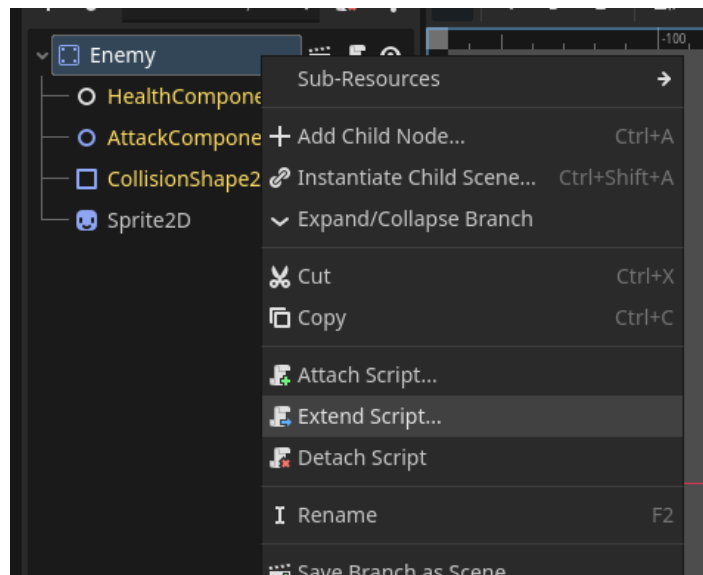


If you need more attacks, you should add them to the attack component (more on that in “How to create a new character” tutorial).

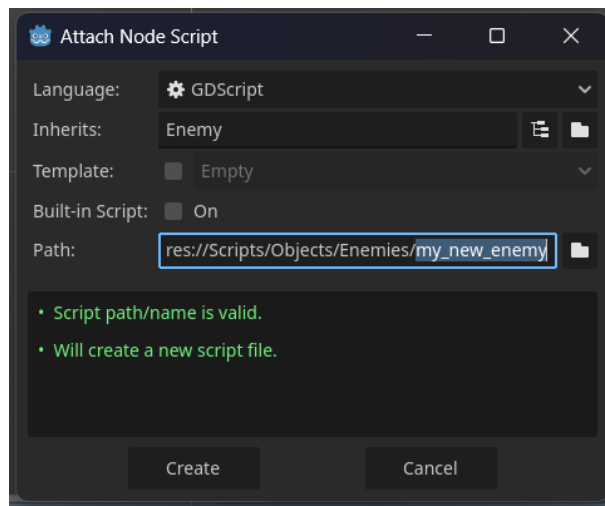


Now if you were to place this new enemy inside of a level it would do nothing except hit the player upon touching it and take damage. So now let's make it do something else as well.

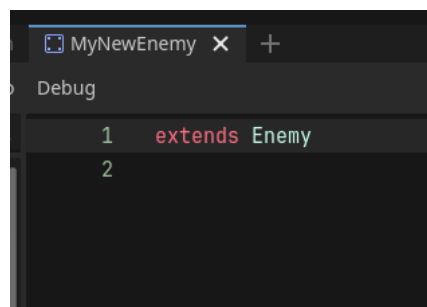
Right click on the "Enemy" node, i.e. the top level node in this scene, and click "Extend Script..."



Place your script somewhere, preferably in "Scripts/Objects/Enemies" folder.

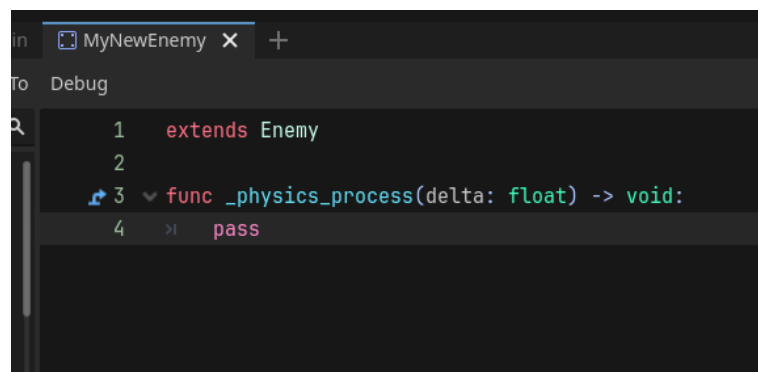


And hit “Create”.

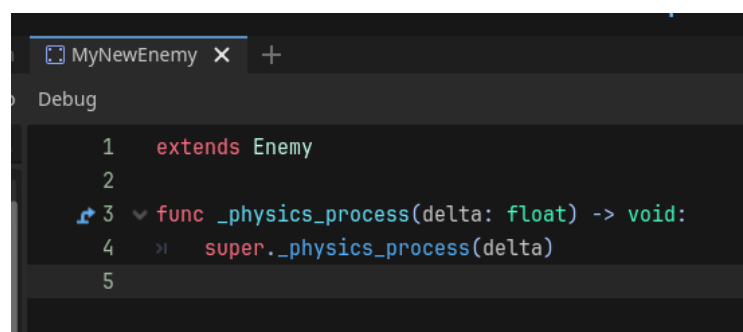


Not sure what I should make it do so I will make it move in a sine wave just as an example.

Just as usual, I will add a function that will be used for physics changes:

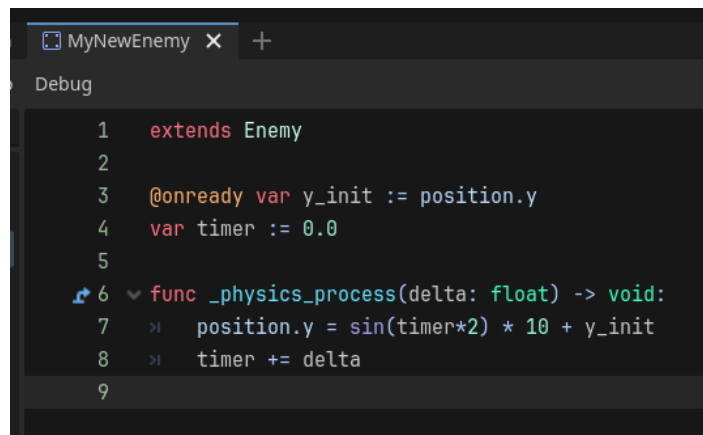


Since there's no “_physics_process” function in the Enemy class, it's fine, but if there was, we would need to call it like this:



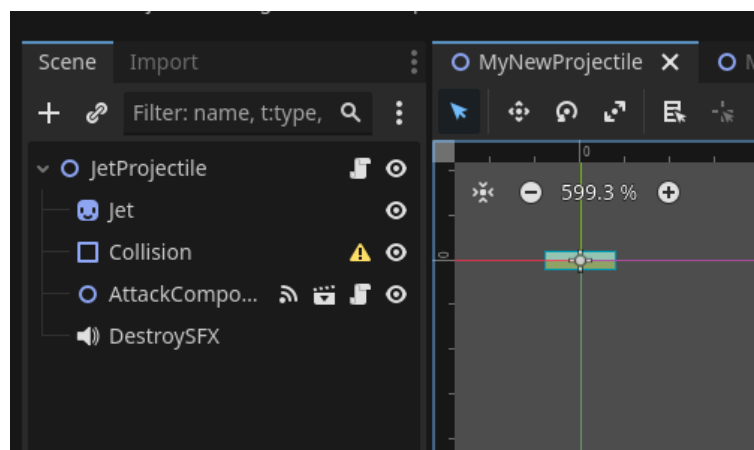
Why? Because if that function existed, there would probably be an important functionality that you would probably not expect to be gone.

So, the sine wave functionality:

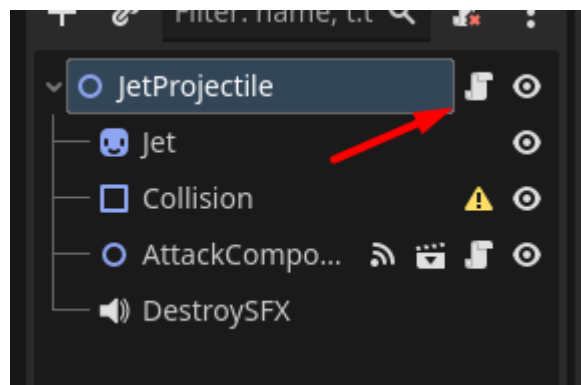


```
1 extends Enemy
2
3 @onready var y_init := position.y
4 var timer := 0.0
5
6 func _physics_process(delta: float) -> void:
7     position.y = sin(timer*2) * 10 + y_init
8     timer += delta
9
```

Now the projectiles. There's no base projectile scene, so we'll need to create our own or duplicate the existing one (they're very simple scenes, so it's not really important to have a base scene and inherit from it). For this tutorial, I will duplicate an already existing projectile, specifically, the jet projectile.



For this tutorial, I will basically duplicate the original jet projectile script so we can edit it a bit. To do that, open the script first by clicking on the scroll icon next to the top level node:

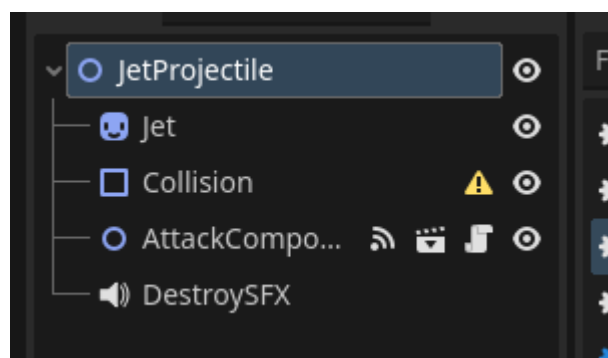
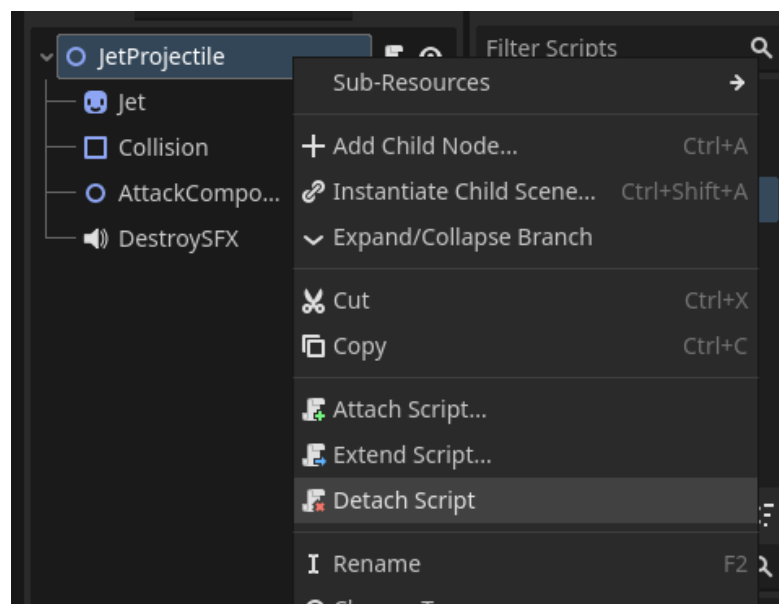


```

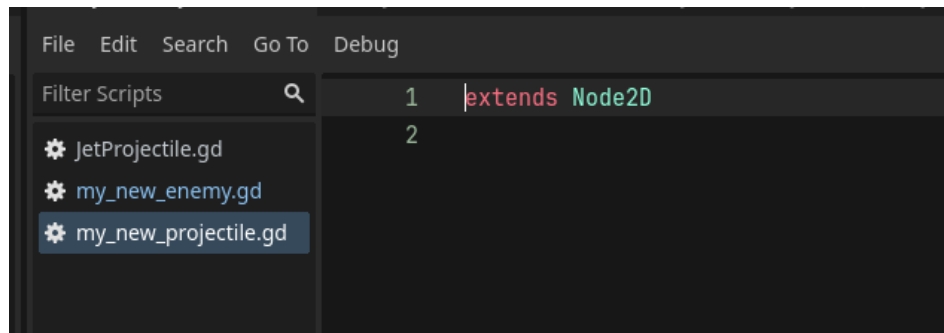
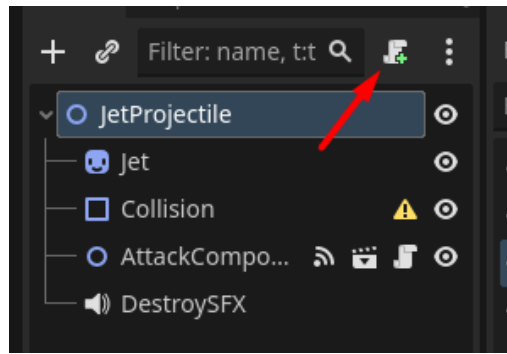
1  extends Node2D
2
3  const VELOCITY := Vector2(-1 * 60, 0.5 * 60)
4  const EXPLOSION := preload("res://Objects/Levels/Explosion.tscn")
5
6  @onready var destroy_sfx: AudioStreamPlayer = $DestroySFX
7  @onready var attack_component: AttackComponent = $AttackComponent
8
9  func _process(delta: float) -> void:
10     position += VELOCITY * delta
11
12  func _on_attack_component_attacked(_body: Node2D, _amount: float) -> void:
13     var explosion := EXPLOSION.instantiate()
14     explosion.global_position = global_position
15
16     destroy_sfx.play()
17     destroy_sfx.reparent(get_parent())
18     destroy_sfx.finished.connect(func() -> void: destroy_sfx.queue_free())
19     add_sibling(explosion)
20     queue_free()
21

```

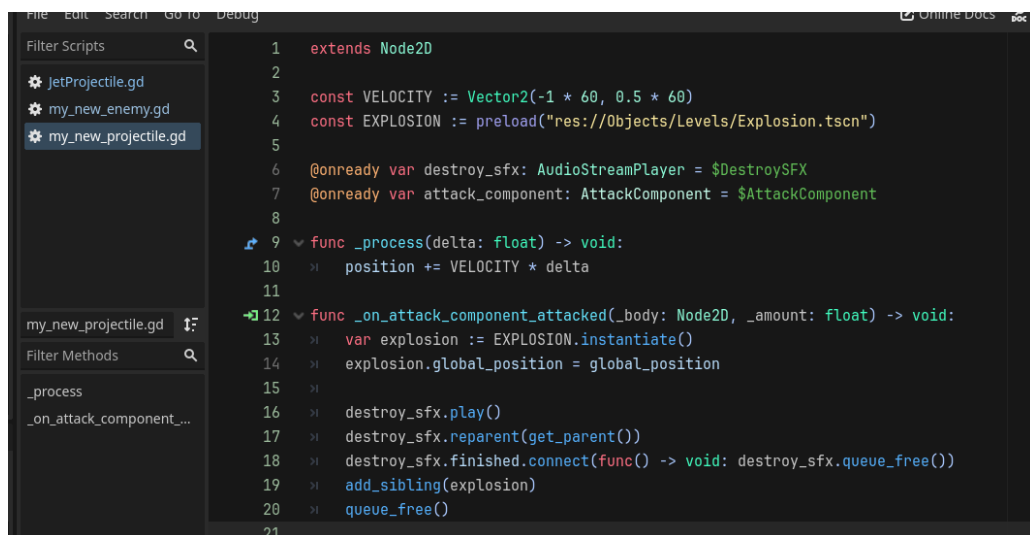
Select everything (you can use a keyboard shortcut for that: Ctrl+A) and copy.
Now detach the existing script:



Create a new one:



Remove the “extends Node2D” line and past the copied script here:

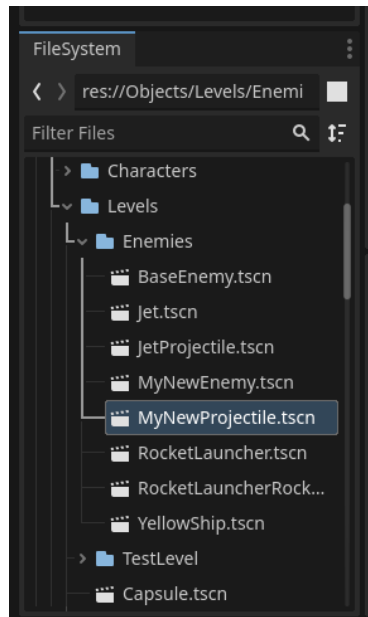


Nice, so now let’s change it a bit, for example, we can make it go up instead of down by changing the “VELOCITY” constant at the top of the script like this:

```

1 extends Node2D
2
3 const VELOCITY := Vector2(-1 * 60, -0.5 * 60)
4 const EXPLOSION := preload("res://Objects/Levels/Explosion.tscn")
5
  
```

Now let’s make our enemy shoot this projectile. First, let’s load it in the script. You can do that in the built in editor very easily this way: first, find the projectile scene in the FileSystem dock:



Drag and drop it somewhere in your script (preferably at the top) while holding Ctrl key:

 A screenshot of a Godot script editor. The script content is as follows:


```

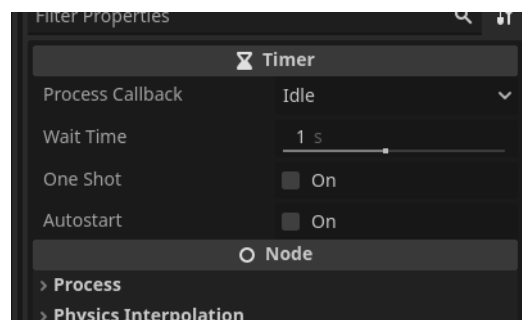
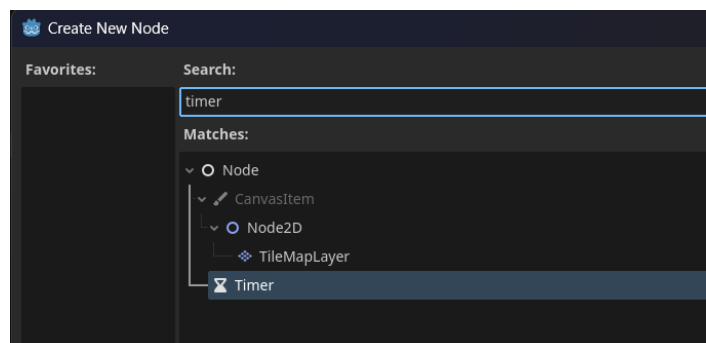
1 extends Enemy
2 const MY_NEW_PROJECTILE = preload("res://Objects/Levels/Enemies/MyNewProjectile.tscn")
3 @onready var y_init := position.y
4 var timer := 0.0
5

```

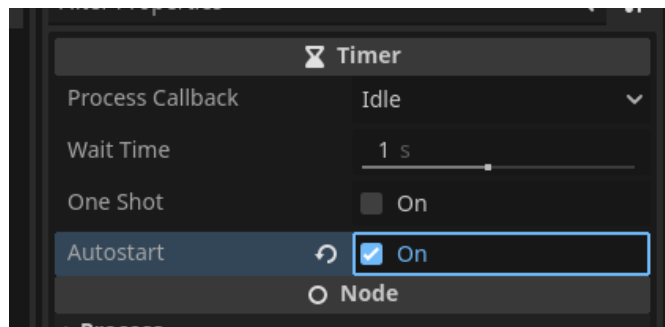
 The second line, which was added by dragging the file from the FileSystem, is highlighted.

Wow, it just magically added the exact line that loads the scene inside of the script, nice!

Now let's add a timer node that will fire every 1 second, and later we will connect the projectile creation to the timer.



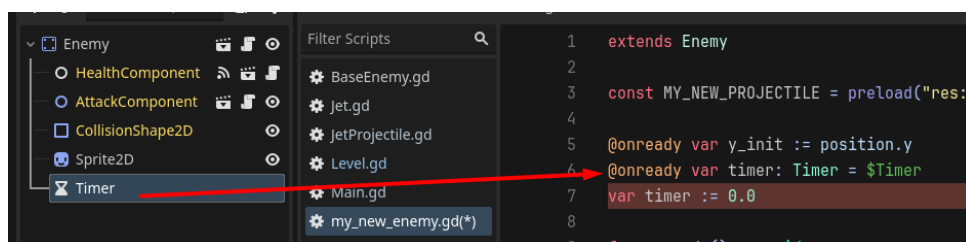
1 second is already the default value, so we can leave it like that, so now let's just check the "Autostart" property so it... starts automatically.



Now we can connect to the timer inside the code. First, let's create the "_ready" function.

```
5 @onready var y_init := position.y
6 var timer := 0.0
7
8 func _ready() -> void:
9     pass
10
11 func _physics_process(delta: float)
12     position.y = sin(timer*2) * 10 +
```

Now let's drag and drop the timer node into the script while holding Ctrl key:



This also works for nodes! It added the line that creates a new variable that references the node that was dragged, but now we have a name conflict, so we can change the timer node variable name a bit:

```
4
5 @onready var y_init := position.y
6 @onready var timer_node: Timer = $Timer
7 var timer := 0.0
8
```

Now let's create a placeholder function for the projectiles.

```

11
12 ▾ func _physics_process(delta: float) ->
13     > position.y = sin(timer*2) * 10 + y
14     > timer += delta
15
16 ▾ func shoot_projectile() -> void:
17     > pass
18

```

And now we can connect it to the timer signal in the “_ready” function like this:

```

8
9 ▾ func _ready() -> void:
10     > timer_node.timeout.connect(shoot_projectile)
11

```

We can copy the jet code that shoots projectiles, so open the jet scene and open its script.

```

33     > > > > > state = State.MOVING_RIGHT
34     > > > > > animation_player.play("flying_right")
35
36 ▾ > > > > > State.MOVING_RIGHT:
37     > > > > > velocity.x += 0.05 * 60 * 60 * delta
38     > > > > > if not launched_projectile and absf(velocity.x) < 0.04 * 60:
39     > > > > > var projectile := JET_PROJECTILE.instantiate()
40     > > > > > projectile.position = position
41     > > > > > add_sibling(projectile)
42     > > > > > projectile.attack_component.objects_to_ignore.append(self)
43     > > > > > launched_projectile = true
44
45     > position += velocity * delta
46
47 ▾ func _on_attack_component_attacked(_body: Node2D, _amount: float) -> void:

```

There’s the code we need. Let’s, ahem, *borrow* it and add it to our code:

```

15
16 ▾ func shoot_projectile() -> void:
17     > var projectile := JET_PROJECTILE.instantiate()
18     > projectile.position = position
19     > add_sibling(projectile)
20     > projectile.attack_component.objects_to_ignore.append(self)
21

```

...while also using the correct names:

```

15
16 ▾ func shoot_projectile() -> void:
17     > var projectile := MY_NEW_PROJECTILE.instantiate()
18     > projectile.position = position
19     > add_sibling(projectile)
20     > projectile.attack_component.objects_to_ignore.append(self)
21

```

Nice. Now if you place it inside a level, you will see it slowly moves up and down and shoots projectiles every second.

