What makes up a Game
Game loop
Check for key presses
Update the positions of everything
- based on user input, physics
draw the screen
Do it all again

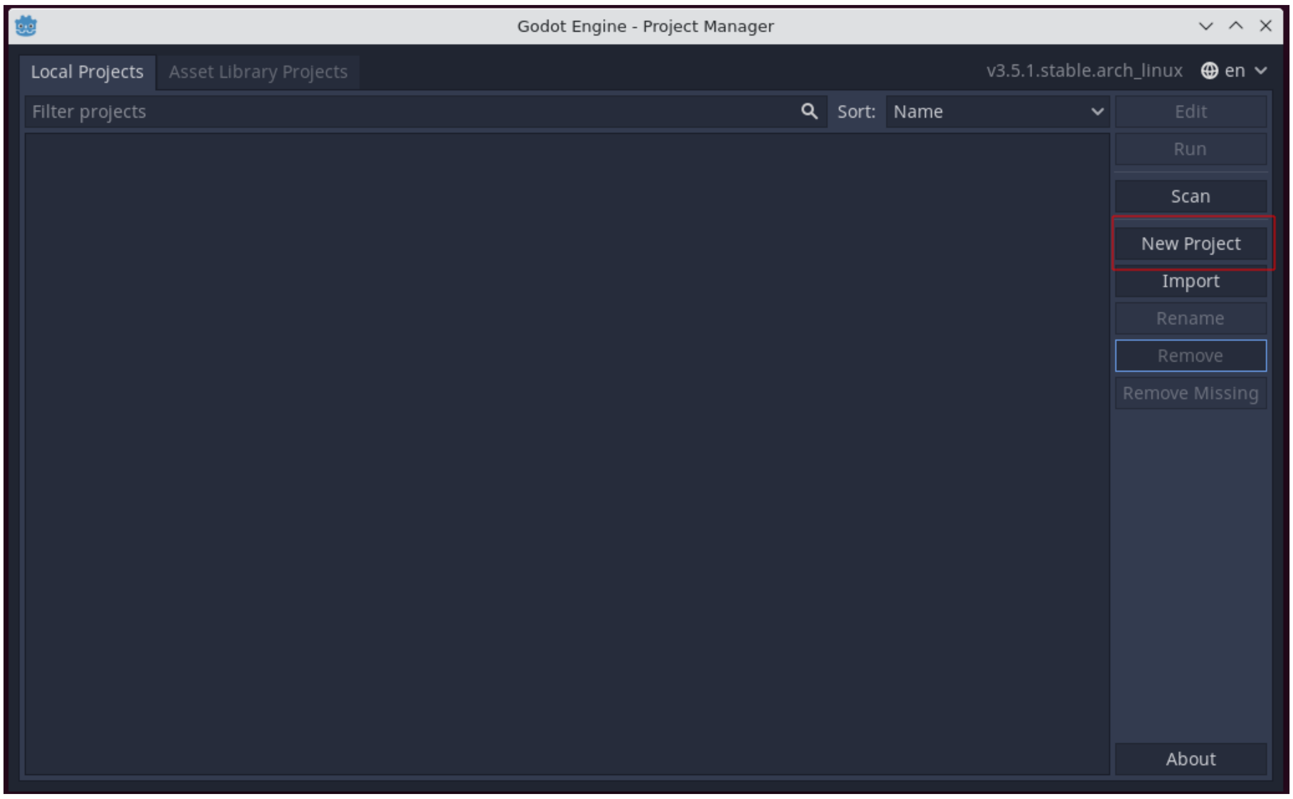
Hard - but Game Engines like Godot help

What is Godot

Aside: web editor
  https://editor.godotengine.org/releases/latest/
  All code stored in fake file-system in the browser.

Installing Windows/macOs
https://godotengine.org/download

After starting the Godot application, you will be prompted to explore official examples projects. Decline this. You will then seen the "Project Manager"

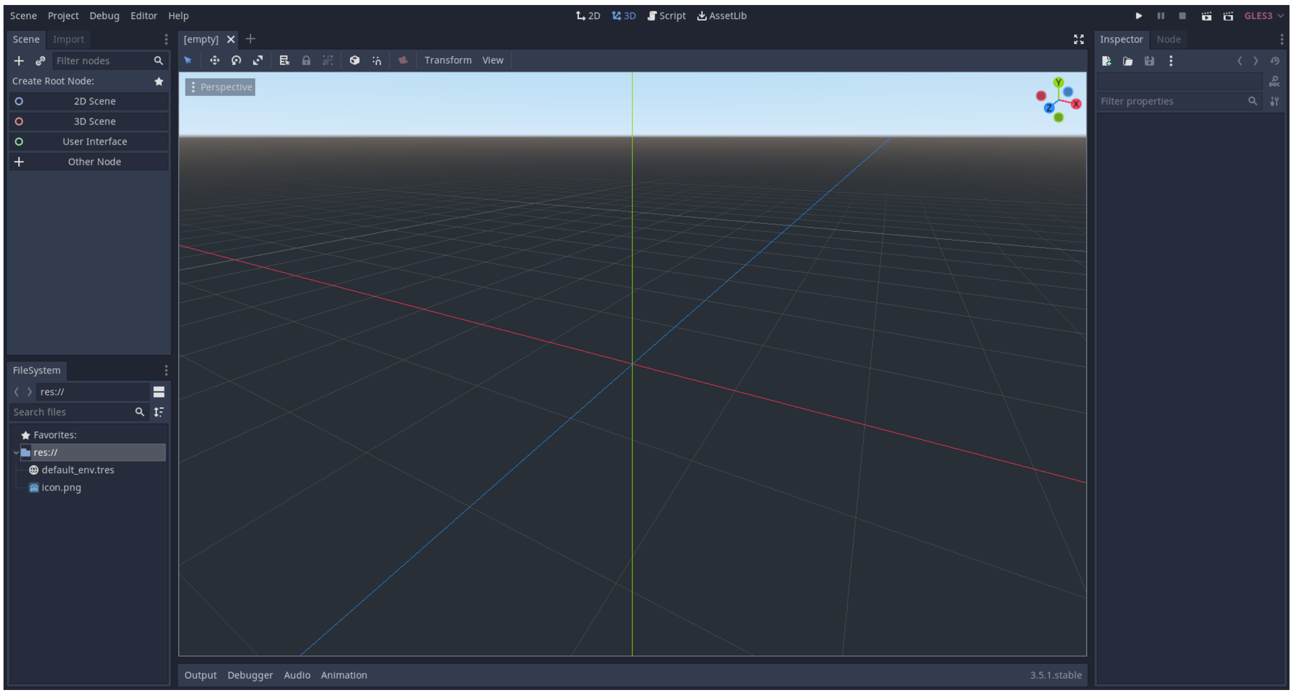


Make a "New Project", by clicking the relevant button
Click "Browse", navigate to a known, directory on your harddisk where you wish to save your Godot project, then click "Create Folder".

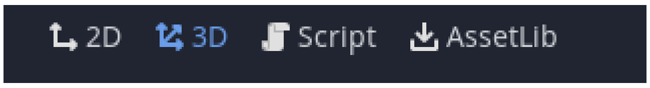
Create a folder called "Snvaders", then click "Select Current Folder"

You will then be presented with the “Create New Project” dialog, where you should select “Create & Edit”.

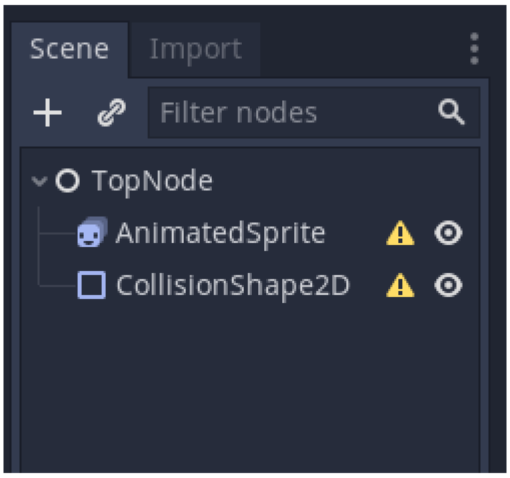
Quick look around the API



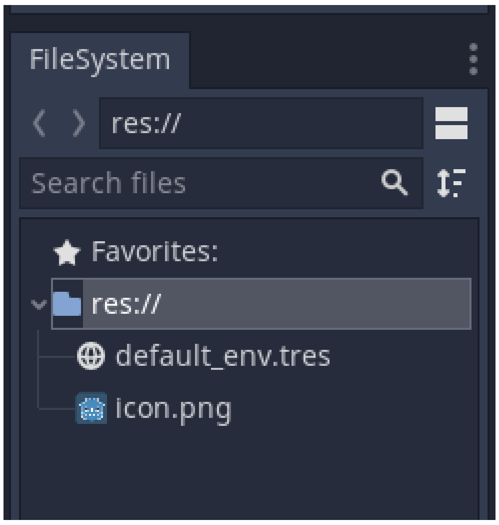
The initial view is the “3D” view. We will be creating a “2D” game today, so will not be using it. You will be switching between the “2D” and “Script” views mainly. To do this click on the “2D” and “Script” View selectors at the top of the screen:



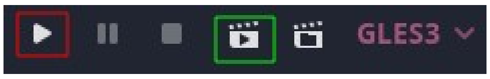
On the LHS of the screen, we see the “Scene” panel. This initially shows “Create Root Node”, but after choosing a node it shows the current Scene Node hierarchy:



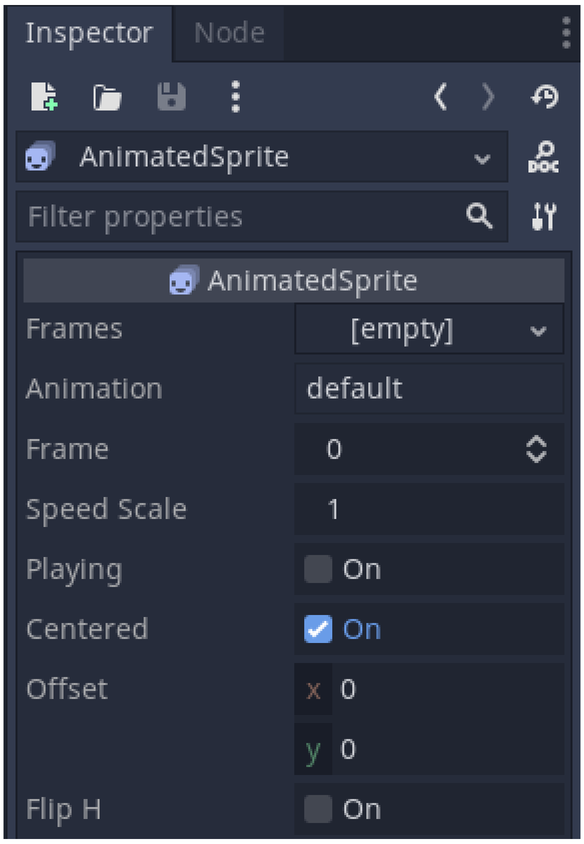
The “FileSystem” panel shows the contents of the directory containing the Godot project, you can click on items in here and open new files and scenes.



At the Top Right, we have the very important Play Game (red highlight) and Play Scene (green highlight) buttons.



The Inspector shows the current properties of the selected Node:



Frames	[empty]
Animation	default
Frame	0
Speed Scale	1
Playing	On
Centered	✓ On
Offset	x 0
	y 0
Flip H	On

The Centre of the screen will either be the 2D view, or the “Script” programming view

Programming - Argghhh!!! However, it’s not too hard and not too much of it. Here is our most

complicated math:
[TO BE CONTINUED]

Godot treats objects in Games as Scenes and Nodes

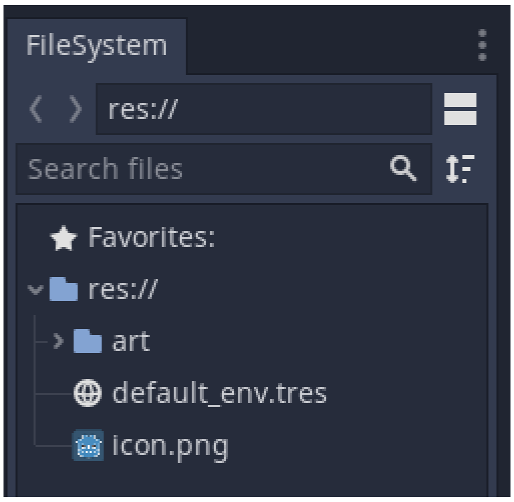
Scenes are things like - Gunship, Alien, Bullet

There is one “Main Scene” in a game, that is run when you press the play button.

Scenes are made up from Nodes, a tree like structure (imagine like the files on disk, or family tree)
Every node has properties, which you can see in the “Inspector” and also change through code

Get started

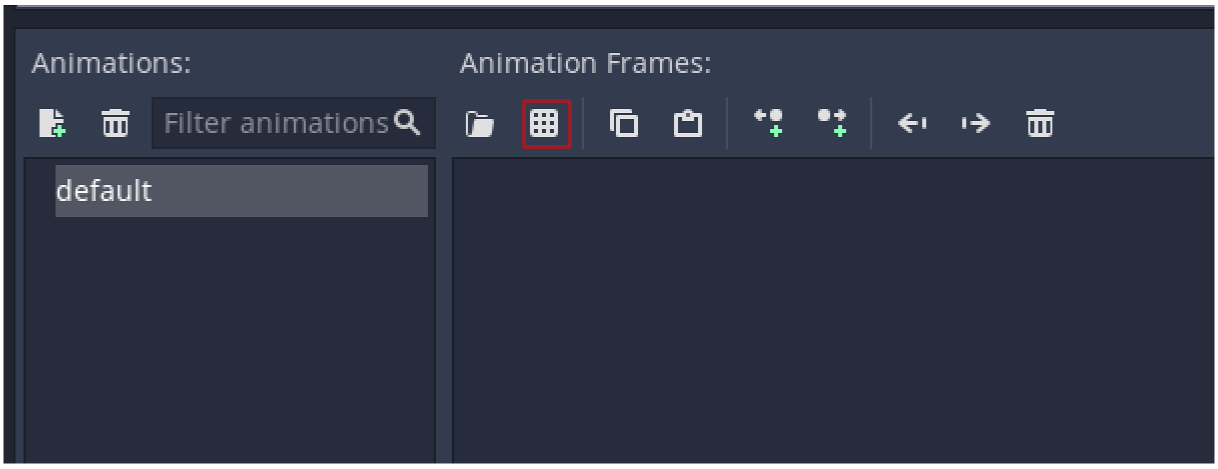
Copy the provided art folder into the SNvaders folder, it will appear in the FileSystem view:



Go to “2D” View.
Make a new Area2D Root Node (click “Other Node” and select from the dialog) - note the warning.
Rename it GunShip (by clicking on the name)
Save the Scene (always use the default name – in this case “GunShip.tscn”)

Area2D is an area that can be collided with. The image and the shape are separate nodes, the image “sprite” we will create now

Make a new AnimatedSprite as a child (right-click “Add Child Node”).
Click on Frames in the property inspector.
Select “New SpriteFrames” - then click on on the SpriteFrames (it will be highlighted in blue) - Look at the animation frames panel that appears at the bottom of the screen.



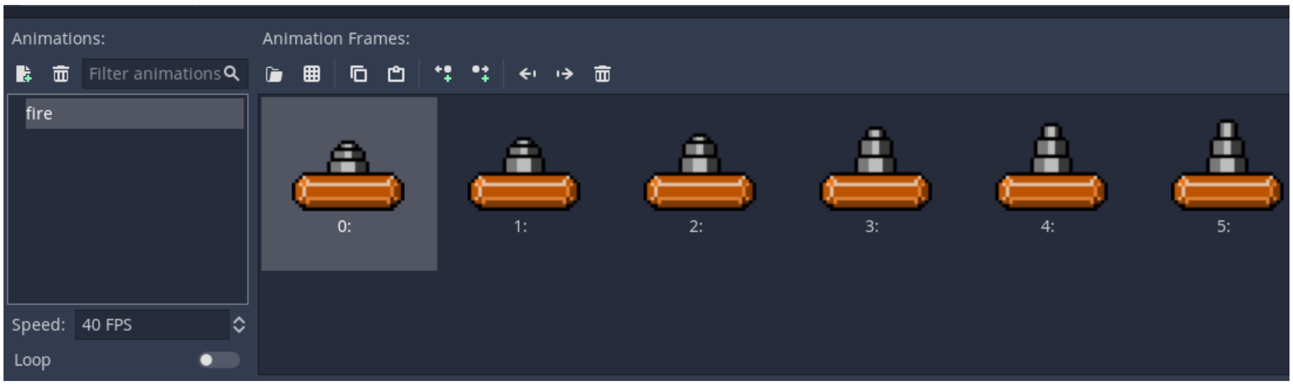
We are going to have sprite sheet:
Click Sprite sheet (griddy thing, highlighted in Red in screenshot above)

Choose “art/turret_frames_64.png” image
Change horiz to 6 and vertical to 1
Click “Select/Clear All Frames”, then “Add (6) Frames”

Select the AnimatedSprite in the Node-tree. Click on "Playing" in the properties- you can see the animation

It's too slow - so change FPS to 40
Turn loop off
Let's reverse the sprites (by dragging and dropping the frames), so that by default the gun recoils quickly and then ends up as it did before.

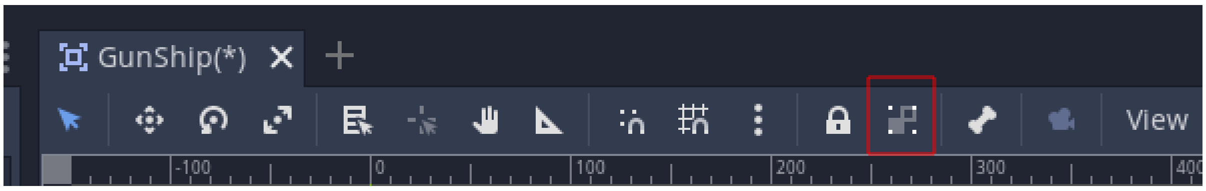
Change animation name from "default" to "fire"



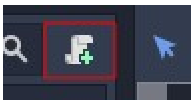
So, normally we'll be on frame 5, but when we play the animation, it'll quickly recoil and then end up where it was.

Make sure playing is off, and we are on Frame 0 (in the properties view)

Back in 2D mode, select the Gunship Root Node, and click the "group child nodes" to stop you accidently moving it around away from the ship.

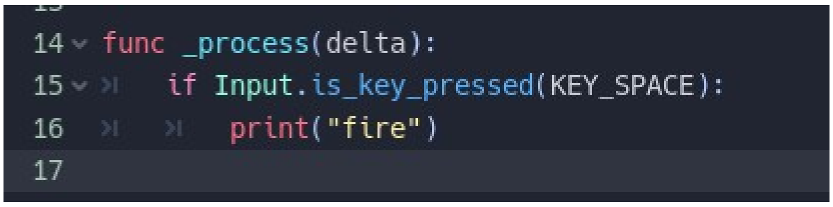


Now lets add a script: Call it Gunship.tscn. Do this by Clicking the “Attach Script” button on the toolbar:



(Accept the default name)

func _process is the main loop
Lets check here to see if we are pressing the fire button and fire:



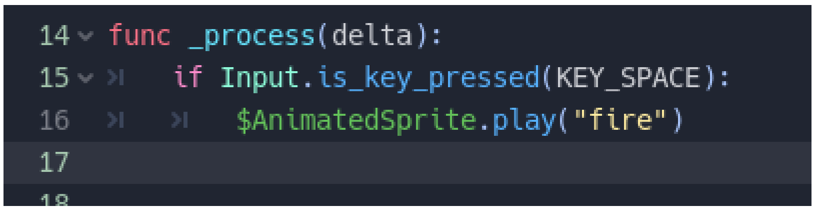
```
func _process(delta):
	if Input.is_key_pressed(KEY_SPACE):
		print("fire")

```

[Gunship.gd]

Run scene - see the word "fire appear" when you press space
Stop playing scene

Move the gunship to where you can see it (in the 2D view): the “viewport box” is the rectangle you see when you zoom out slightly

Make it really fire when you press space:



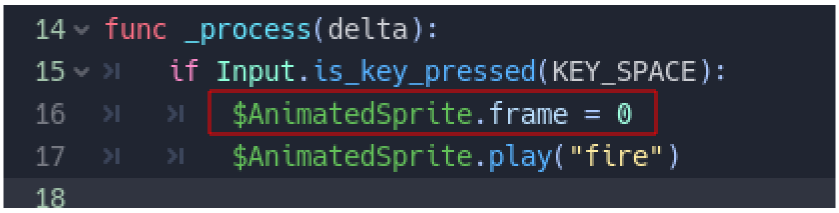
```
func _process(delta):
	if Input.is_key_pressed(KEY_SPACE):
		$AnimatedSprite.play("fire")

```

[Gunship.gd]

If nothing happens, go to “Properties” of AnimatedSprite, make sure frame is 0 and playing is "OFF" ...

Cool, but it only plays through once...

Change the gunship _process:

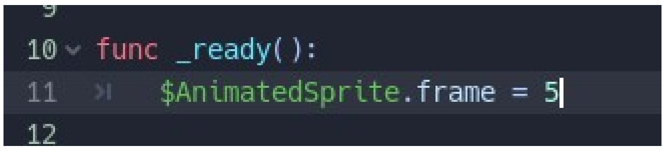


```
func _process(delta):
	if Input.is_key_pressed(KEY_SPACE):
		$AnimatedSprite.frame = 0
		$AnimatedSprite.play("fire")
```

[Gunship.gd]

Better, but we want the sprite to start on frame 5 when the game starts...

Change the _ready function in Gunship.gd, which occurs when the sprite is added to the scene (in this case when the game starts):



```
func _ready():
	$AnimatedSprite.frame = 5
```

[Gunship.gd]

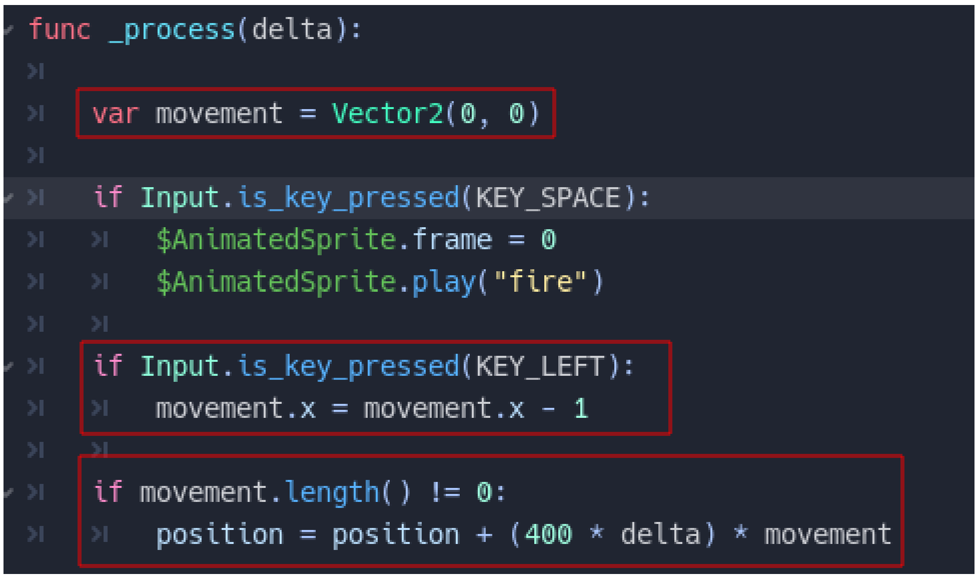
Note what happens when we hold the button down...

Lets make it move left and right.

But first: What does delta mean...

It's the fraction of a second between the process method being called and the last time it was called. If it last occurred 1 second ago, it would be 1, half a second ago, 0.5 etc. Ususally called 60 times per second, so delta is usually around (1/60), but it depends on how much the computer has to do. Don't rely on it being any particular value, but use it in your calculations.

So, when we press the left button, we want to move the gunship to the left at a speed of 400 pixels per second, so every frame we need to move it 400*delta pixels



```
func _process(delta):

	var movement = Vector2(0, 0)

	if Input.is_key_pressed(KEY_SPACE):
		$AnimatedSprite.frame = 0
		$AnimatedSprite.play("fire")

	if Input.is_key_pressed(KEY_LEFT):
		movement.x = movement.x - 1

	if movement.length() != 0:
		position = position + (400 * delta) * movement
```

[Gunship.gd]

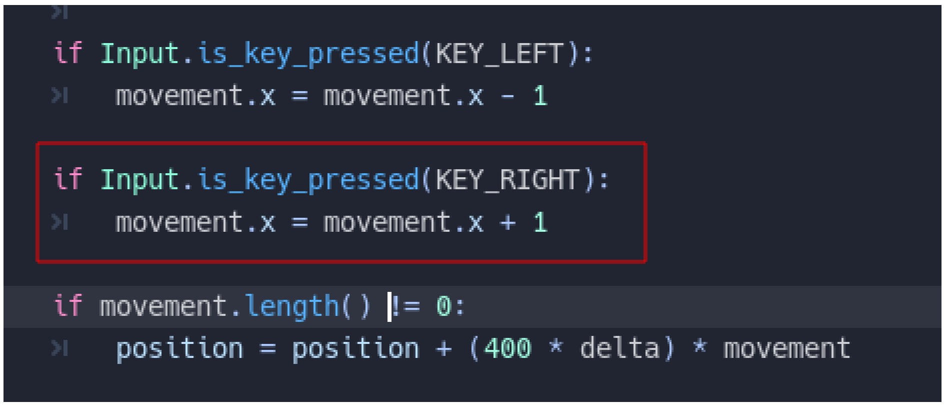
Here “movement” is a vector. A vector is like a pair of x & y coordinates on a graph. In our case, for the gunship the "y" will always be zero, and x will be -1 for a move to the left and +1 for a move to the right

At the end of the loop, we need to update the gunships position.

Checking the ‘length’ of “movement” being not zero means that the player has tried to move the gunship during this frame. If we have moved, we update the position with the current position, adding on the fraction of the movement of 400 pixels per second.

Run the scene - notice that when you press the left key, you now move left.

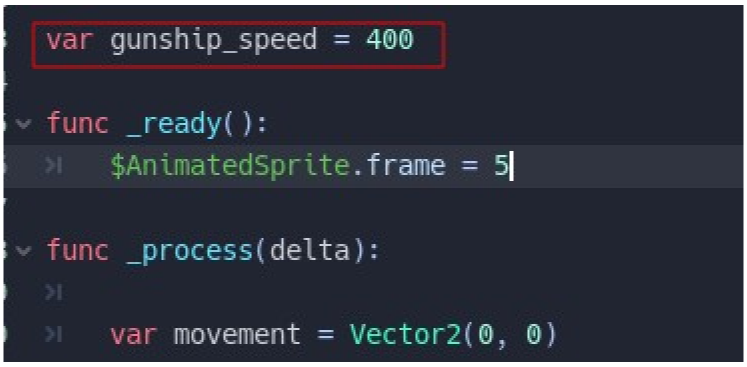
Challenge: Make the sprite move right!



```
	if Input.is_key_pressed(KEY_LEFT):
		movement.x = movement.x - 1

	if Input.is_key_pressed(KEY_RIGHT):
		movement.x = movement.x + 1

	if movement.length() != 0:
		position = position + (400 * delta) * movement
```

[Gunship.gd]

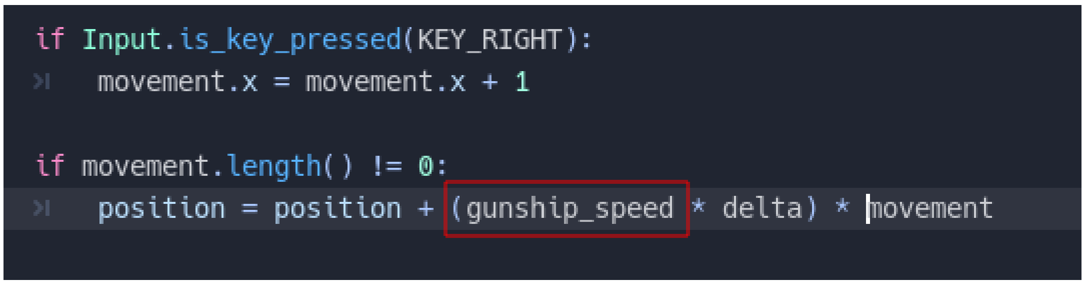
It's good practice to separate "magic numbers" out of the code and give them names. In our case "400" is the speed of the ship. Lets call this gunship_speed and refactor it:



```
var gunship_speed = 400

func _ready():
	$AnimatedSprite.frame = 5

func _process(delta):

	var movement = Vector2(0, 0)
```

[Gunship.gd]

change:



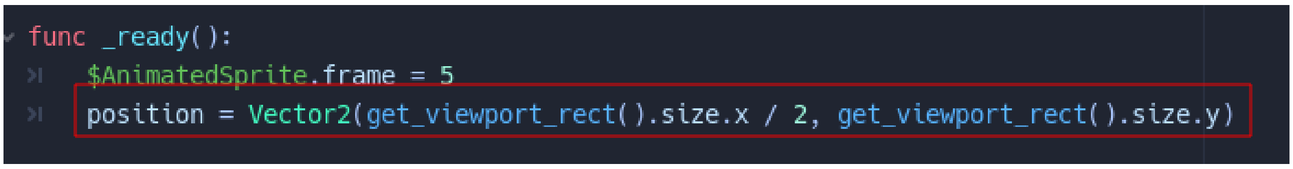
```
	if Input.is_key_pressed(KEY_RIGHT):
		movement.x = movement.x + 1

	if movement.length() != 0:
		position = position + (gunship_speed * delta) * movement
```

[Gunship.gd]

It's a bit easier to make sense of now when we are coming back to this is a few weeks time.

Review. Can move left and right, and when we press fire, the gun recoils. However, we move off the edges of the screen (should stop that) and holding the space bar down means that it seems to always play animation frame 0, so don't seem to be shooting. Let's fix the left/right first

Initial position: In _ready, we'll put the gunship at the bottom of the screen in the centre:

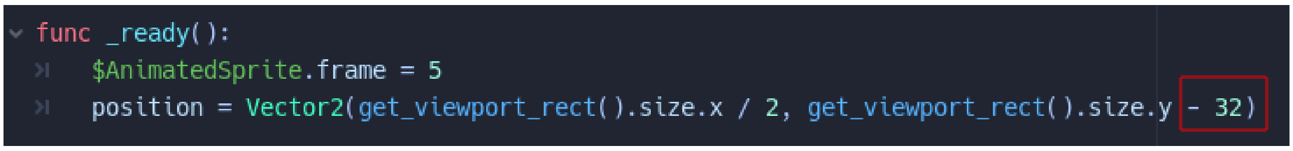


```
func _ready():
	$AnimatedSprite.frame = 5
	position = Vector2(get_viewport_rect().size.x / 2, get_viewport_rect().size.y)
```

[Gunship.gd]

Run it.

Slight problem - we need to move it up a bit...



```
func _ready():
	$AnimatedSprite.frame = 5
	position = Vector2(get_viewport_rect().size.x / 2, get_viewport_rect().size.y - 32)
```

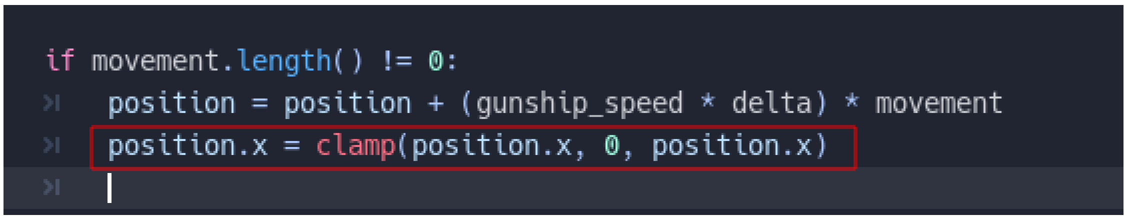
[Gunship.gd]

Better.

Now let's clamp:

Clamping, basically means, if a value gets smaller than a certain fixed value, make it the fixed value. If it gets bigger than a certain fixed value, make it the bigger one.

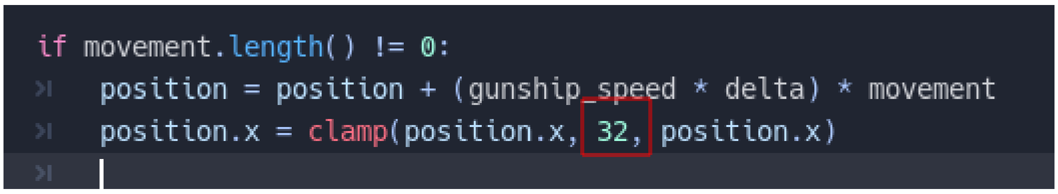
Let's try this:



```
if movement.length() != 0:
	position = position + (gunship_speed * delta) * movement
	position.x = clamp(position.x, 0, position.x)
	
```

[Gunship.gd]

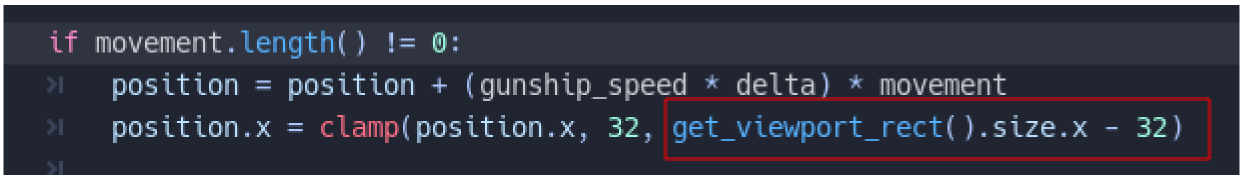
Cool, but when you try it out, you notice that we have let the minimum get too small:



```
if movement.length() != 0:
	position = position + (gunship_speed * delta) * movement
	position.x = clamp(position.x, 32, position.x)
	
```

[Gunship.gd]

Is better, but now we fall off the RHS of the screen:



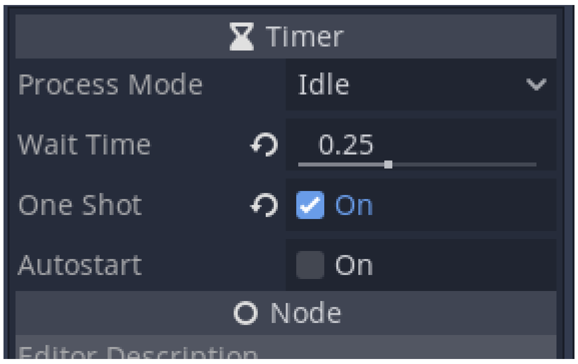
```
if movement.length() != 0:
    position = position + (gunship_speed * delta) * movement
    position.x = clamp(position.x, 32, get_viewport_rect().size.x - 32)
```

[Gunship.gd]

Is perfect - we now can move left and right and fire within the context of the screen.

One more thing, lets' slow the gun firing down a bit. We can do that with a timer.

Add a "Timer" child node to the Gunship, call it “FireTimer”. Make it a 'One Shot', and set it's wait time to 0.25 (by changing the properties in the RHS panel. So, we will be able to fire 4 shots per second.



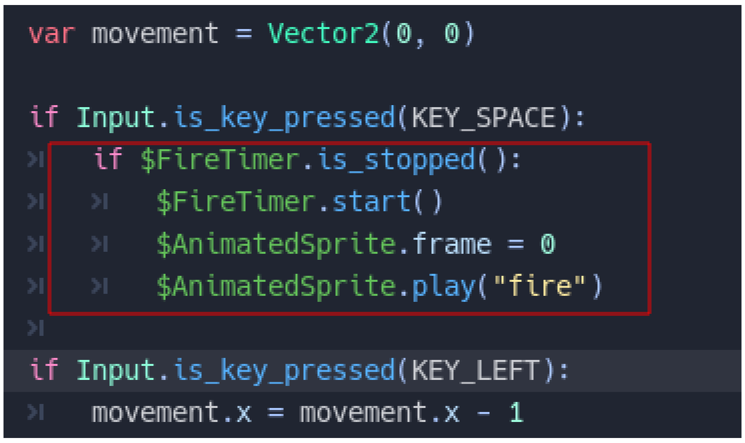
Timer

Process Mode	Idle
Wait Time	0.25
One Shot	✓ On
Autostart	On

Node

Editor Description

Now we'll change the Firing code, so that we see if the fire timer has been triggered, if it hasn't we can fire our shot and trigger the timer. If it has been triggered we can't fire another shot yet...



```
var movement = Vector2(0, 0)

if Input.is_key_pressed(KEY_SPACE):
    if $FireTimer.is_stopped():
        $FireTimer.start()
        $AnimatedSprite.frame = 0
        $AnimatedSprite.play("fire")

if Input.is_key_pressed(KEY_LEFT):
    movement.x = movement.x - 1
```

[Gunship.gd]

That's about it for the basics of the gunship until we want to make it collide with an alien or really fire bullets at them. In order to do that we are going to need a baddie (or misunderstood extra terrestrial to be more PC) ....

All the code up until this point is available in the project in the “Part1” directory

Now we are going to make a new Scene - called Alien. Create this by clicking the “+” along the top tab bar.

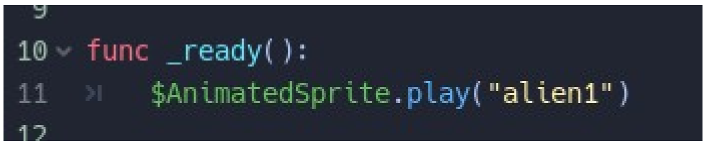
It's also an Area2D

Rename it to Alien and switch back to the 2D view
It needs a image, add a AnimatedSprite child node. Select sprite frames for 'Cthulhy'
Call the animation 'alien1'. 5 FPS is okay. This time we can loop.

Lock the child frames together

Add a script (called Alien.gd). Save it.

Make sure the alien is playing when it is added to a scene:



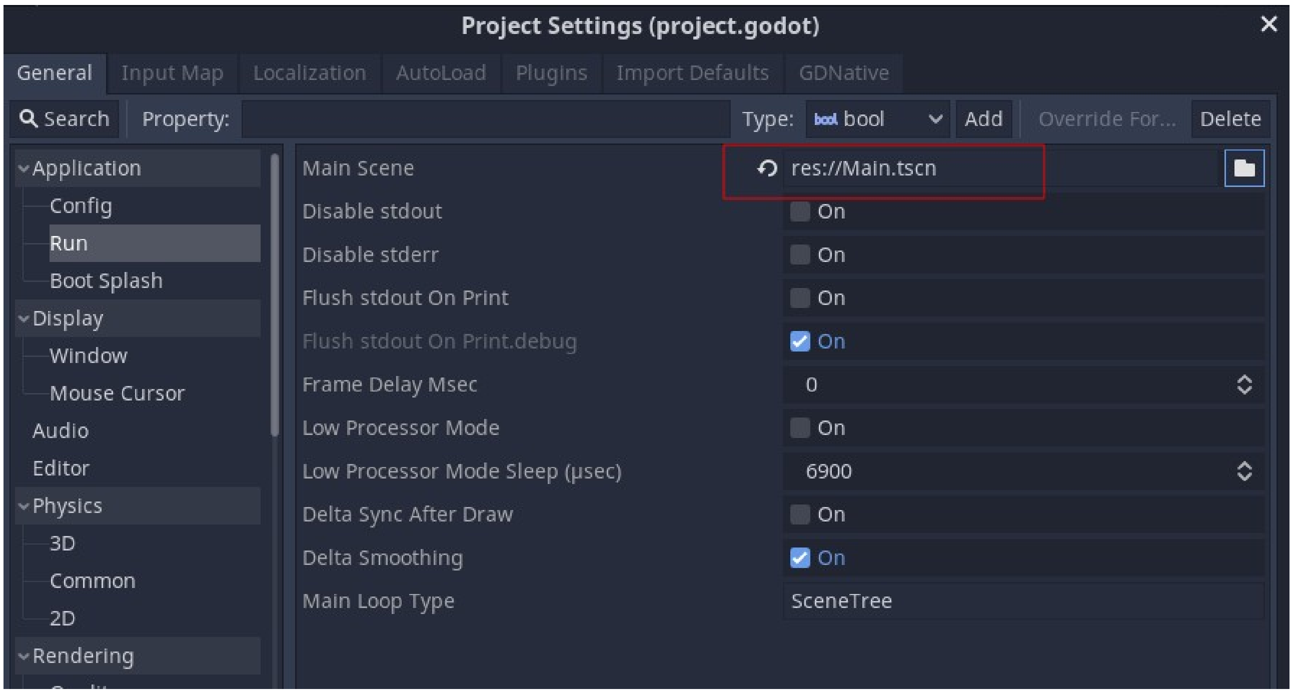
```
func _ready():
	$AnimatedSprite.play("alien1")
```

[Alien.gd]

Now, let's make a "Main" scene, where bring both the Alien and the Gunship together:

Make a new Scene from a simple "Node2D" and call it "Main".

We can make this the main project node that is run when we press the "Play" button. Select project settings, and under the Application/Run section, choose "Main.tscn" as the main scene.



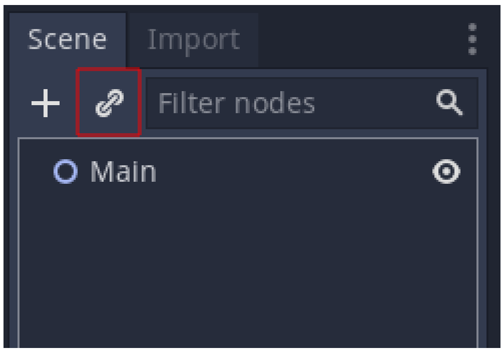
Project Settings (project.godot)

General | Input Map | Localization | AutoLoad | Plugins | Import Defaults | GDNative

Search | Property: | Type: bool | Add | Override For... | Delete

Main Scene	res://Main.tscn
Disable stdout	On
Disable stderr	On
Flush stdout On Print	On
Flush stdout On Print.debug	✓ On
Frame Delay Msec	0
Low Processor Mode	On
Low Processor Mode Sleep (µsec)	6900
Delta Sync After Draw	On
Delta Smoothing	✓ On
Main Loop Type	SceneTree

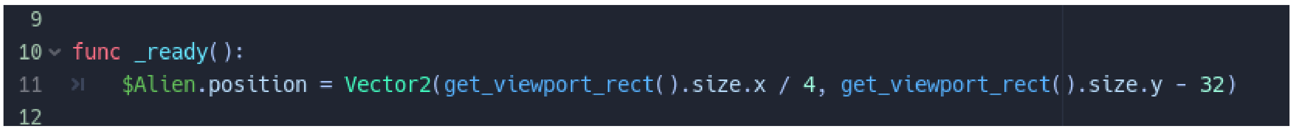
Pressing play now will launch the main scene. Not much happens. Lets link in the Gunship and Alien. Using the "Link" chain icon.



You can play around with the parenting here to see what happens....

Let’s move the alien down to the same level as the gunship:

To do this Add a Script to the Main scene (called Main.gd), and put some code into ready:



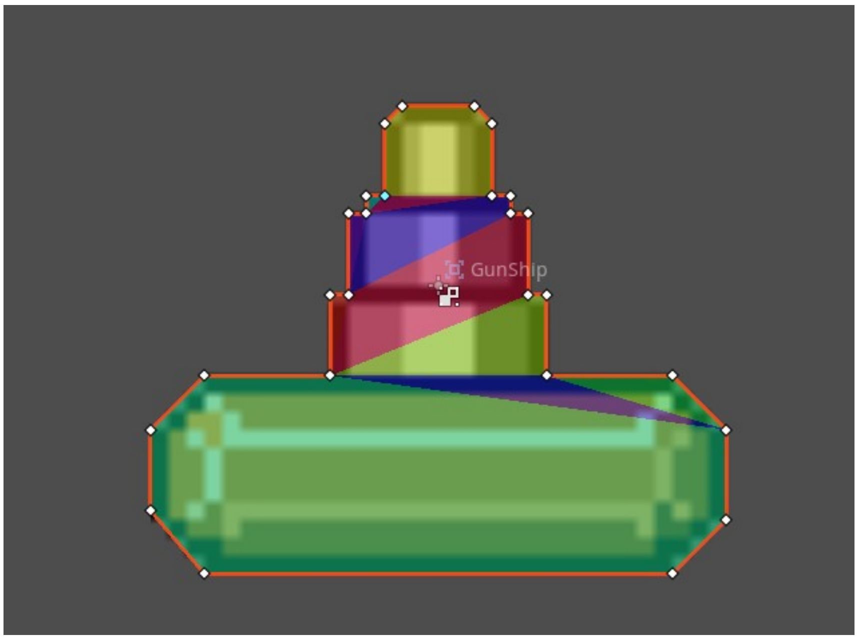
```
func _ready():
	$Alien.position = Vector2(get_viewport_rect().size.x / 4, get_viewport_rect().size.y - 32)
```

[Main.gd]

When we run this we see that we can touch the alien without damage. We need to learn about collisions and get rid of those warning signs on Alien and Gunship

Go back to the Gunship, switch to 2D view and, set the AnimatedSprite to frame 5. Zoom in so you can see it properly.

Add a CollisionPolygon2D as a child node of the Gunship root node. We are going to draw around the outline of the turret. Click around the shape, each click adds a point. Draw the outline. Can click and move to tidy them up. Right click deletes them.



You can Hide the rather confusing colors by clicking the "eye" next to the collision polygon in the scene view. Save the scene.

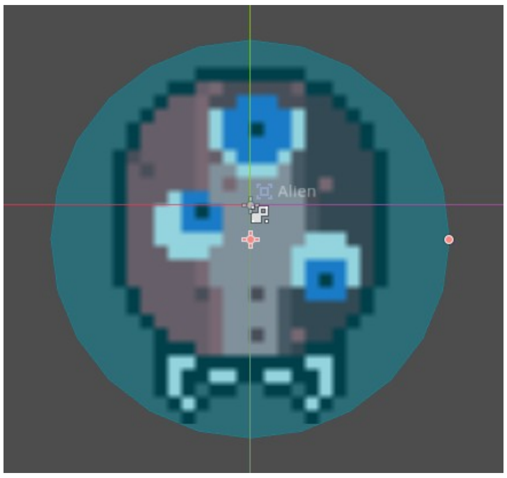
Select the Alien.

Before we give this one a collision shape, lets add a few more animations so we can have some variety in the game.

Click the AnimatedSprite, then the Frames/SpriteFrames property and add a 2 new animations (call them alien2 and alien3 and make them the skully and medusy shapes. They can all be 5 FPS and loopy.

You can play with the animations by clicking "Playing" and changing the "Animation" drop down.

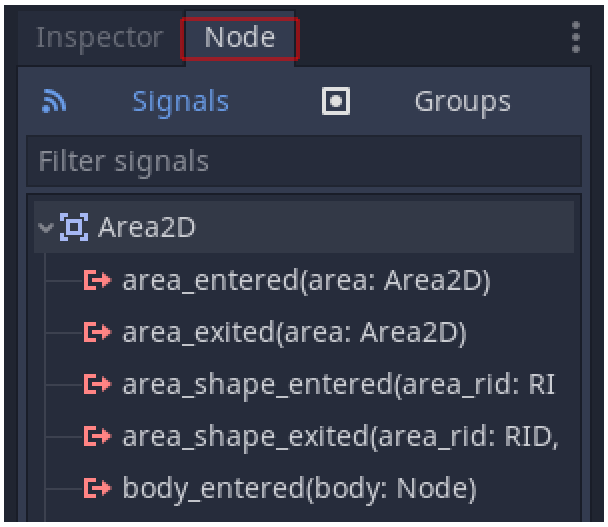
Add a "CollisionShape2D" to the Alien, Choose a shape of "CircleShape2D" and resize/position it so that it covers the alien reasonably well for each of the frames. You may need to "make the children selectable" to do this. If you do, lock them all together again afterwards.



Ok, so now each shape has a "Collision" area, so we can detect when they touch each other. All the warning triangles have gone off.

Go back to the Gunship.

Select the Top node, then over on RHS, click Node. These are "signals"


Inspector Node

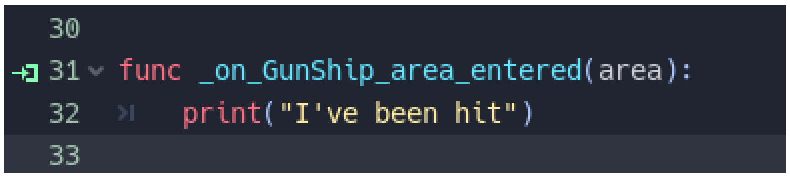
Signals Groups

Filter signals

Area2D
- area_entered(area: Area2D)
- area_exited(area: Area2D)
- area_shape_entered(area_rid: RI
- area_shape_exited(area_rid: RID,
- body_entered(body: Node)

What are signals?
Signals are also known as events - they tell you when something happens. area_entered will "happen" when something collides with you. You can make a method and connect it to a signal, so that method runs when the signal occurs.

Click area_entered() and click connect. Select the defaults (Make in Gunship)



```
func _on_GunShip_area_entered(area):
	print("I've been hit")

```

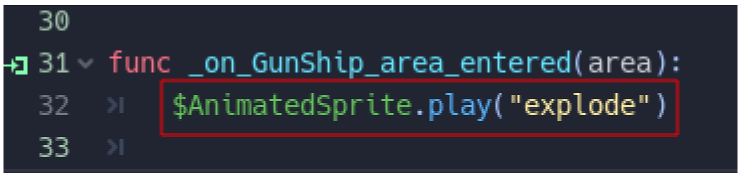
[Gunship.gd]

Try driving your gunship over the alien. You'll see the message "I've been hit!" every time you first touch the alien.

Let's make the gunship explode when it hits the alien

Add an animation to the Gunships AnimatedSprite (art/explosion1.png). Call it ‘explode’. It's 50 frames in total (10x5). Set FPS to 50 and turn loop off. Set the animation back to "fire"

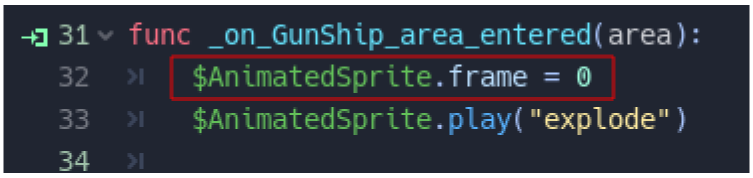
Now let’s change the collision code:



```
func _on_GunShip_area_entered(area):
	$AnimatedSprite.play("explode")

```

[Gunship.gd]

Looks good, but something interesting is happening. The sprite is still there and the game is still running. To see this set the frame back to 0:

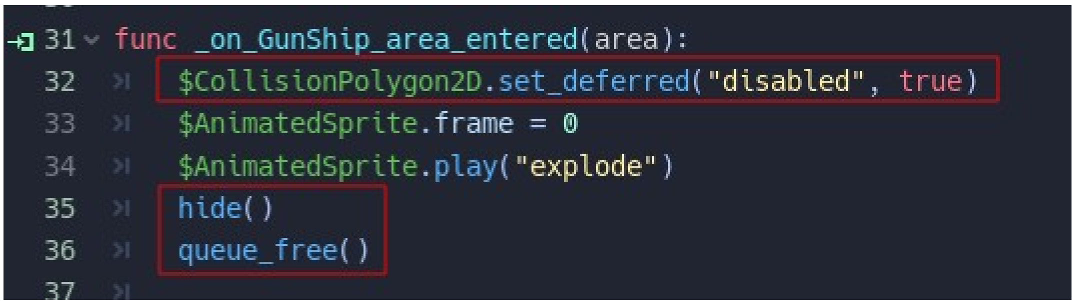


```
func _on_GunShip_area_entered(area):
	$AnimatedSprite.frame = 0
	$AnimatedSprite.play("explode")

```

[Gunship.gd]

Now try "driving" your gunship back and forth over the alien! Explosions from nowhere!

To fix this, we'll hide and really destroy the gunship when it touches the alien:



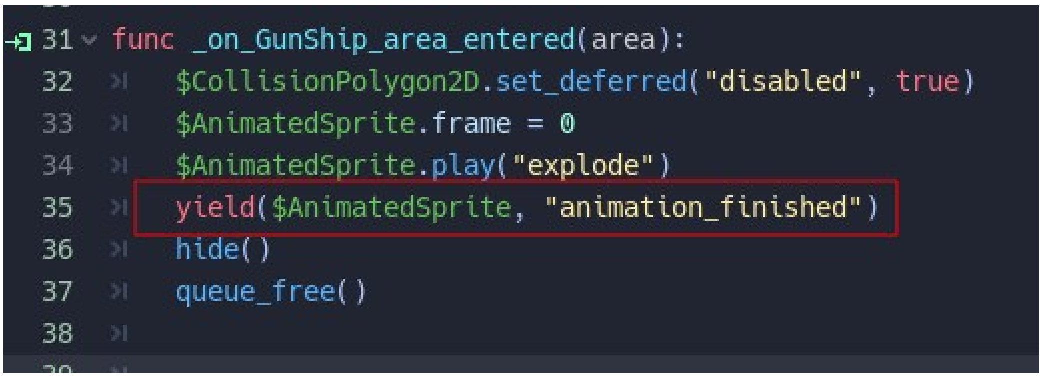
```
func _on_GunShip_area_entered(area):
	$CollisionPolygon2D.set_deferred("disabled", true)
	$AnimatedSprite.frame = 0
	$AnimatedSprite.play("explode")
	hide()
	queue_free()
```

[Gunship.gd]

Here we are disabling collision detection (so that the Gunship does not interact with other object anymore), hiding the it, and asking it politely to “free” (delete) itself when the current frame finishes.

What? Now it disappears when it touches the alien. This is because we dont wait for the explode animation to finish before we hide it, disable it's collision area (to make it stop interacting with other objects) (and delete it).

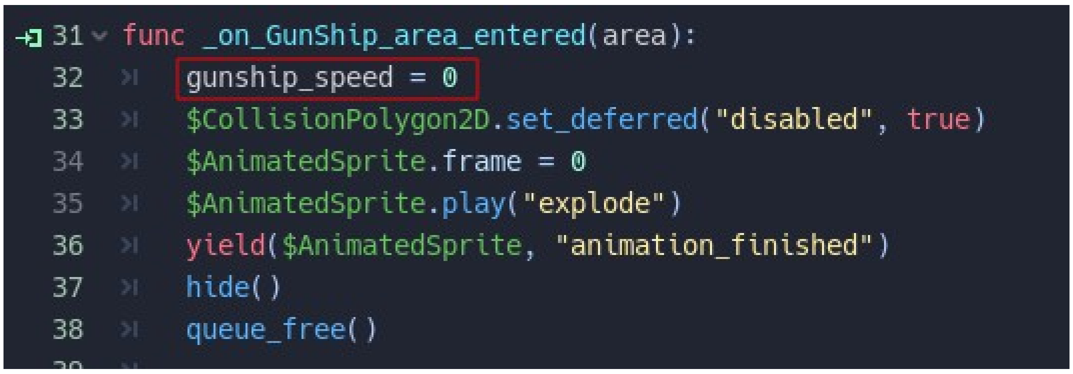
An "AnimatedSprite" has a signal too - called "animation_finished". We can wait for this before we hid and delete the sprite:



```
func _on_GunShip_area_entered(area):
	$CollisionPolygon2D.set_deferred("disabled", true)
	$AnimatedSprite.frame = 0
	$AnimatedSprite.play("explode")
	yield($AnimatedSprite, "animation_finished")
	hide()
	queue_free()
```

[Gunship.gd]

Better, but notice that the exloding gunship keeps moving until it finally disappears... We can change the gunspeed to zero to fix this - was worthwhile refactoring this out earlier:



```
func _on_GunShip_area_entered(area):
	gunship_speed = 0
	$CollisionPolygon2D.set_deferred("disabled", true)
	$AnimatedSprite.frame = 0
	$AnimatedSprite.play("explode")
	yield($AnimatedSprite, "animation_finished")
	hide()
	queue_free()
```

[Gunship.gd]

So, now we have a situation where we can move our gunship, have it explode when it hits an alien and fire it's gun, although no bullets come out...

All the code up until this point is available in the project in the “Part2” directory

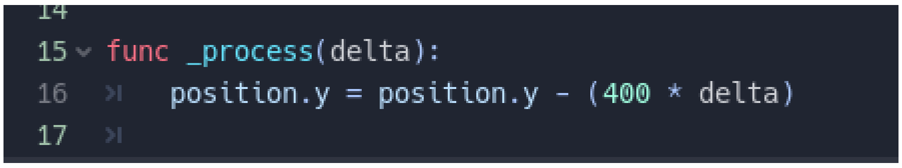
Lets make bullets work!

Make a new scene, an Area2D, called Missile and save it.
Make a new AnimatedSprite child, and load up the missile shape (bolt_64.png).
Also, give it a collisionshape of a simple rectangle. (New RectangleShape2D). Then lock all the children together to prevent you from accidently moving parts

Move it to the bottom middle of the scene for testing. Click play scene. Not very exciting yet....

We need it to move up the screen. Lets move it at 400 pixels per second...

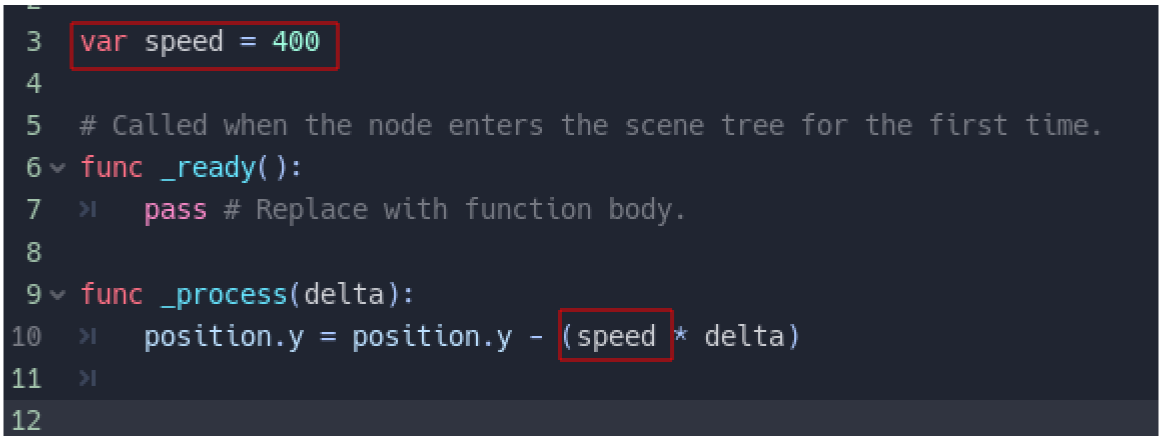
Add a script, called Missile.gd:



```
func _process(delta):
    position.y = position.y - (400 * delta)
    
```

[Missile.gd]

While we are at it, lets factor out the bullet speed:

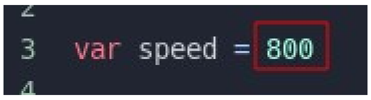


```
var speed = 400

# Called when the node enters the scene tree for the first time.
func _ready():
    pass # Replace with function body.

func _process(delta):
    position.y = position.y - (speed * delta)
    

```

[Missile.gd]

Cool, now we can make the bullet faster if we want..

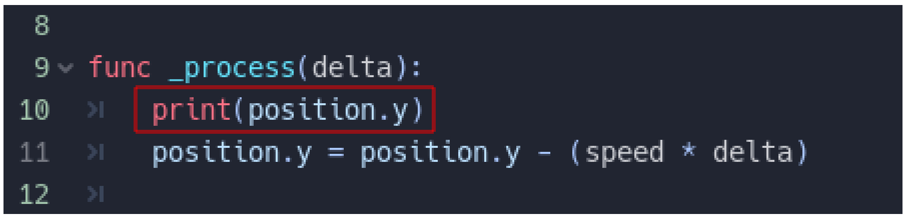
800 is better :)



```
var speed = 800
```

[Missile.gd]

The bullet flies on for ever - to see this:

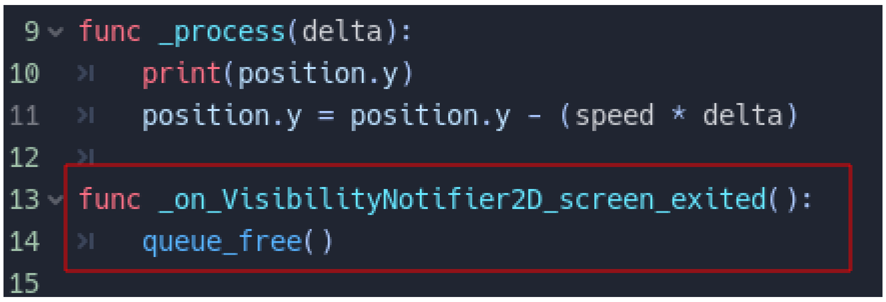


```

func _process(delta):
    print(position.y)
    position.y = position.y - (speed * delta)
    
```

[Missile.gd]

And run again... You'll see that the position in the debug window keeps printing an ever decreasing Y position for the non-visible missile. Not good - we kind of want the bullet to stop when it has left the screen. Fortunately, there is a way to do this, using a 'VisibilityNotifier2D' node. Add one as a child of the missile.

Then connect the signal 'screen_exited' to the Missile, and make sure it's deleted:



```
func _process(delta):
	print(position.y)
	position.y = position.y - (speed * delta)

func _on_VisibilityNotifier2D_screen_exited():
	queue_free()

```

[Missile.gd]

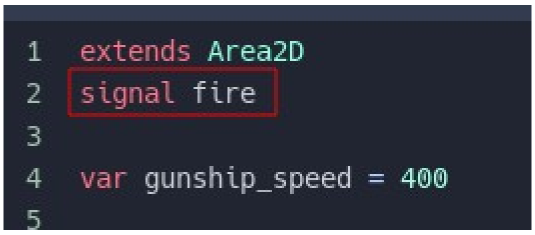
Now you can see that the debug output stops when the missile leaves the screen. This may not seem important, but in a while we are going to have loads of missiles on screen at the same time. If they all go on travelling forever, and the game loop has to perform calculations for them, we may eventually slow down and run out of memory (unlikely on a simple game like this on a modern computer, but this is very important when dealing with far more complex games with thousands or tens of thousands of complex objects in play at a time)

We can now remove the "print() from the _process as we can see that all is well.

So, let's make the gun really fire that missile.

We will create our own signal called "fire" on the Gunship. When we fire the gun, the gunship will raise the "fire" signal, which we can then connect to the main game and launch a new missile

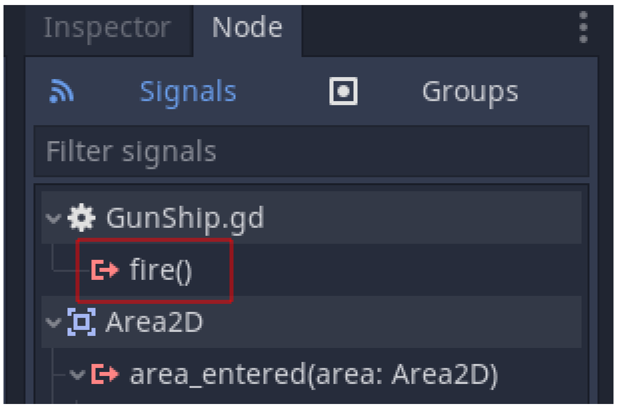
In the Gunship.gd:



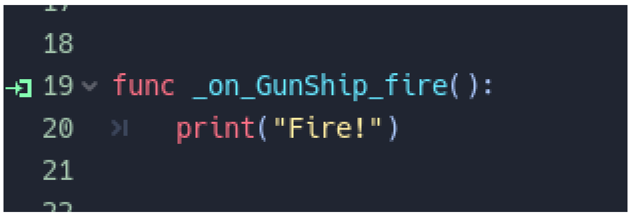
```
extends Area2D
signal fire

var gunship_speed = 400

```

[Gunship.gd]

Now, if you click on the "Gunship" main node, you will see a "fire()" signal :)



Go to the Main scene, and click on the Gunship Node, then Look at it's signals. Connect the fire() signal to the Main scene.

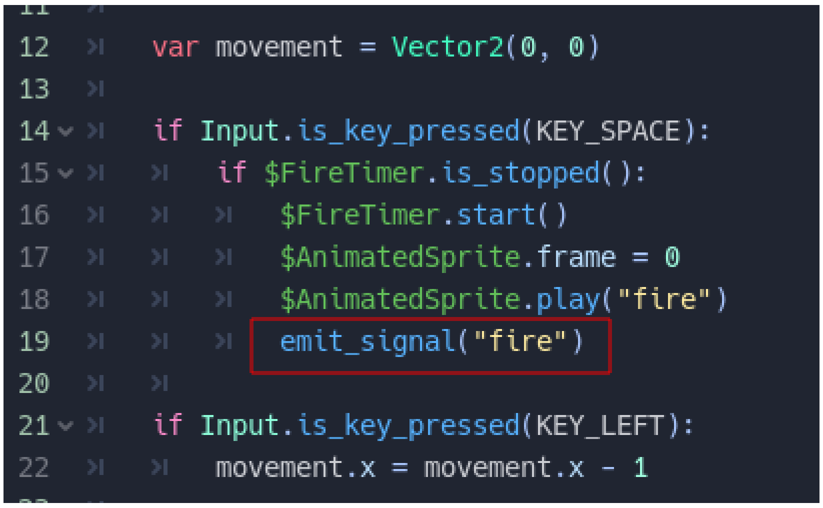


```
func _on_GunShip_fire():
	print("Fire!")
```

[Main.gd]

Run the main program and press "space" to fire.... Hmm, nothing happens
Ah - we need to tell the Gunship when to tell everybody when that fire signal should happen

In the Gunship, add an "emit_signal":

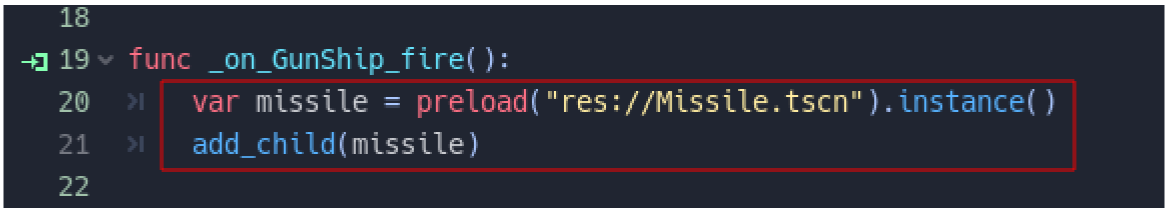


```
	var movement = Vector2(0, 0)

	if Input.is_key_pressed(KEY_SPACE):
		if $FireTimer.is_stopped():
			$FireTimer.start()
			$AnimatedSprite.frame = 0
			$AnimatedSprite.play("fire")
			emit_signal("fire")

	if Input.is_key_pressed(KEY_LEFT):
		movement.x = movement.x - 1
```

[Gunship.gd]

Now, when running the game you'll see the message "Fire!” happening when we try and fire bullets

In the Main scene, in the _on_Gunship_fire() code:



```
func _on_GunShip_fire():
	var missile = preload("res://Missile.tscn").instance()
	add_child(missile)
```

[Main.gd]

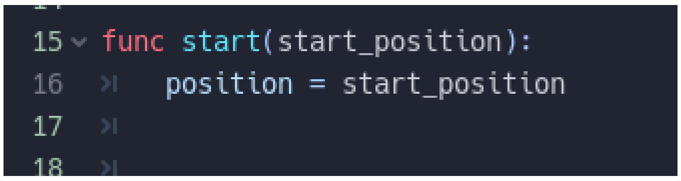
That will make a new missle instance and add it to the scene. The missile will immediately start running and fly off the screen.

There are a few problems. Firstly, the missile is not firing out of the gun, and secondly, it doesn't move around with the gun. We will need to set the position of the missile when it is fired to the barrel of the gun.

To do this - we will make a new function in the missile called start() and pass it a position (Vector2)

Open Missile.gd

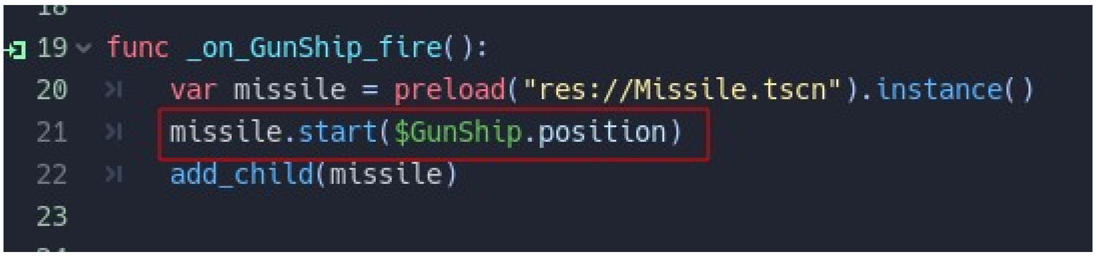
add new function



```
func start(start_position):
	position = start_position
	
	
```

[Missile.gd]

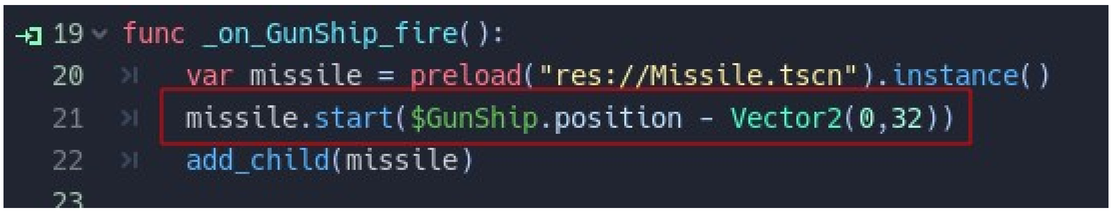
And change the Main.gd:



```
func _on_GunShip_fire():
	var missile = preload("res://Missile.tscn").instance()
	missile.start($GunShip.position)
	add_child(missile)
```

[Main.gd]

Try that out! BANG!!! The Gunship explodes! I guess those missiles hurt us as much as the aliens. Let's fix that by making the missile come from just above the gunship turret:



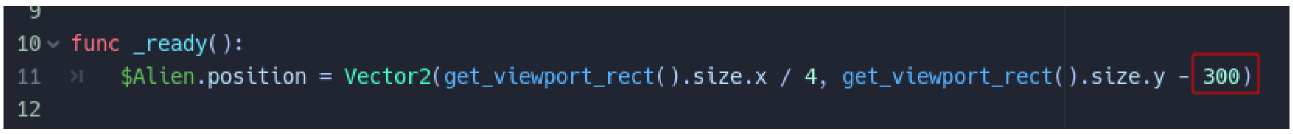
```
func _on_GunShip_fire():
	var missile = preload("res://Missile.tscn").instance()
	missile.start($GunShip.position - Vector2(0,32))
	add_child(missile)
```

[Main.gd]

Better.

Now lets move the alien away from the bottom of the screen so we can shoot him/her/it

Change _ready in Main

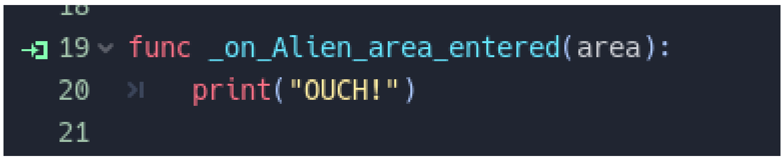


```
func _ready():
	$Alien.position = Vector2(get_viewport_rect().size.x / 4, get_viewport_rect().size.y - 300)
```

[Main.gd]

Can try shooting it now, but it doesn't die. We are going to need to set up a signal on the alien this time...

On Alien scene (select the Alien root node), set up an area_entered signal to run a method:

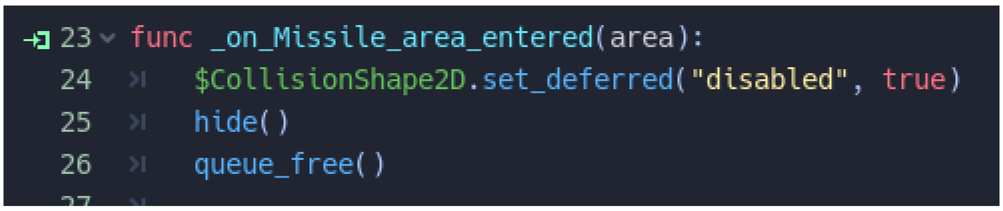


```
func _on_Alien_area_entered(area):
	print("OUCH!")
```

[Alien.gd]

Cool, the alien notices it, but the missile goes straight through it.. Let's stop the missile:

In the Missile scene, select the root node and connect the area_entered signal to a new method (in missile):



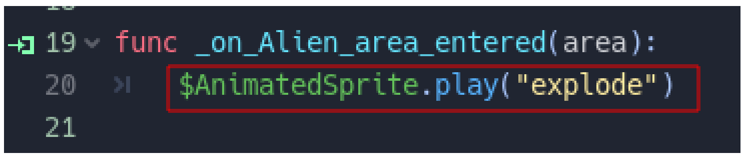
```
func _on_Missile_area_entered(area):
	$CollisionShape2D.set_deferred("disabled", true)
	hide()
	queue_free()
```

[Missile.gd]

Much better - now, lets make the alien pop!

In the Alien scene, add a new explosion animation to the AnimatedSprite, called explode (explosion_64.png, 12x1), non-looping, 25 FPS

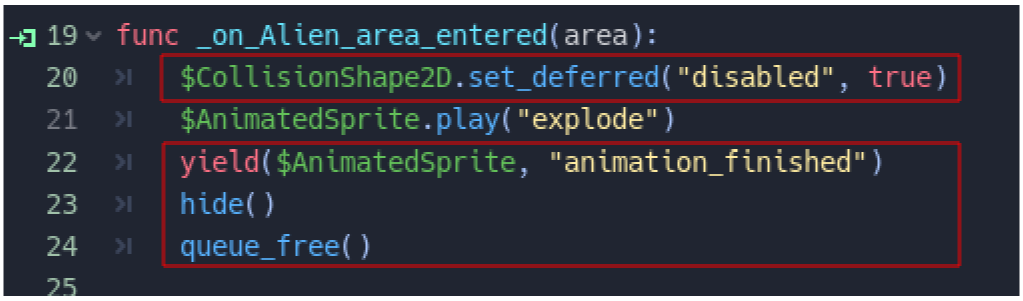
Change the alien entered:



```
func _on_Alien_area_entered(area):
	$AnimatedSprite.play("explode")
```

[Alien.gd]

You can shoot it, but it is still there :) So, like with the Gunship, we need to clean it up....



```
func _on_Alien_area_entered(area):
	$CollisionShape2D.set_deferred("disabled", true)
	$AnimatedSprite.play("explode")
	yield($AnimatedSprite, "animation_finished")
	hide()
	queue_free()
```

[Alien.gd]

And there we go - we can shoot the alien!

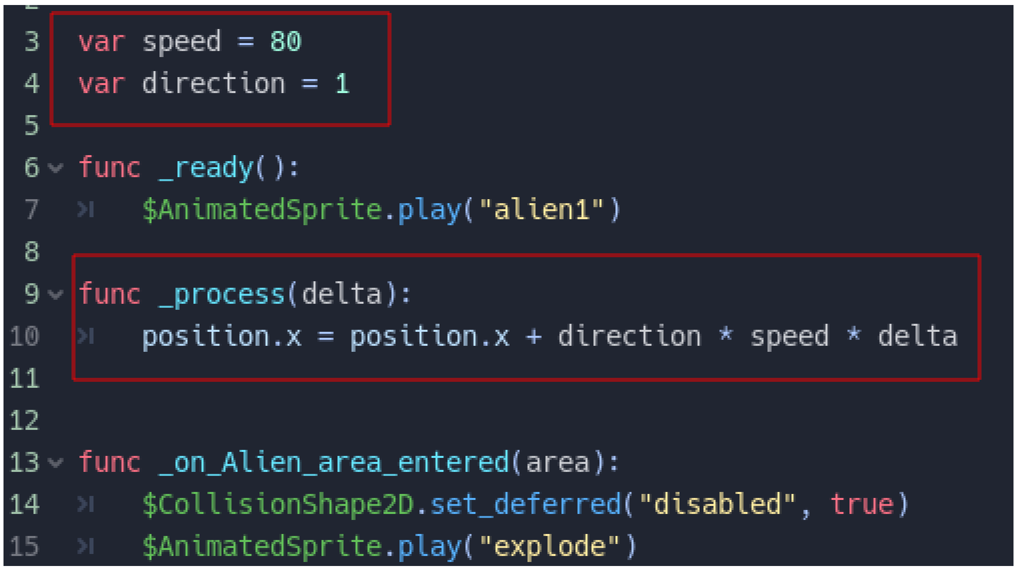
It's a bit unsporting, just having one alien that doesn't move.... We need to make it move, and also it would be good for it to
have some friends - a real swarm of aliens.

All the code up until this point is available in the project in the “Part3” directory

Let's figure out how to get it to move first...

In the Alien scene 2D, move the alien so you can see it nicely into the play area

Alien.gd, add some speed variable and movement code:

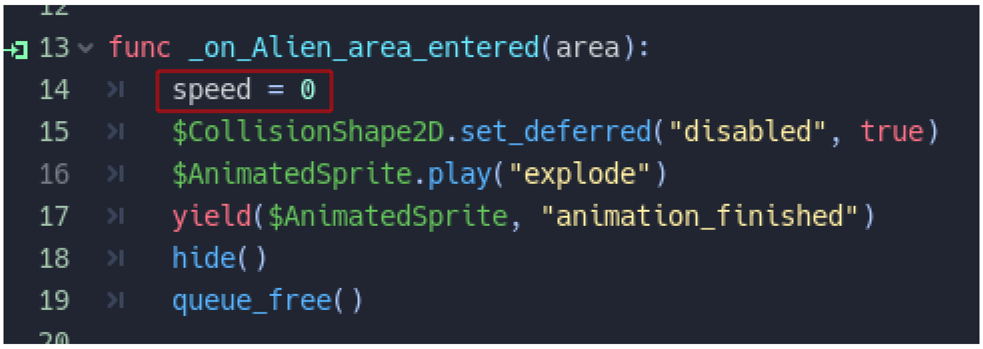


```
var speed = 80
var direction = 1

func _ready():
	$AnimatedSprite.play("alien1")

func _process(delta):
	position.x = position.x + direction * speed * delta


func _on_Alien_area_entered(area):
	$CollisionShape2D.set_deferred("disabled", true)
	$AnimatedSprite.play("explode")
```

[Alien.gd]

Now you can play the scene and see the alien moving.

Try playing the whole game, and see if you can catch the alien before it leaves the screen. Notice, it does continue moving even after it has been shot, so we should change the speed to 0 when it is hit:



```
func _on_Alien_area_entered(area):
	speed = 0
	$CollisionShape2D.set_deferred("disabled", true)
	$AnimatedSprite.play("explode")
	yield($AnimatedSprite, "animation_finished")
	hide()
	queue_free()
```

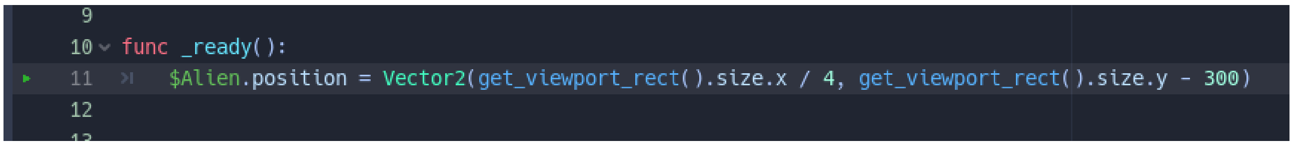
[Alien.gd]

Very nearly all good now. So, lets get it to bounce off the side of the screen and move down. The easiest way do do this is to make a new Scene - we'll call it swarm and put our Alien into that, rather than having it directly on the main scene.

Create a new Scene called Swarm - it can just be a Node. Now Link the Alien node into this new Swarm scene and save it.

Remove the Alien from the Main scene
Link the swarm scene to the main Scene.

If you try running the code now, it will fail, as we have some test code still in the Main function referring to the Alien:



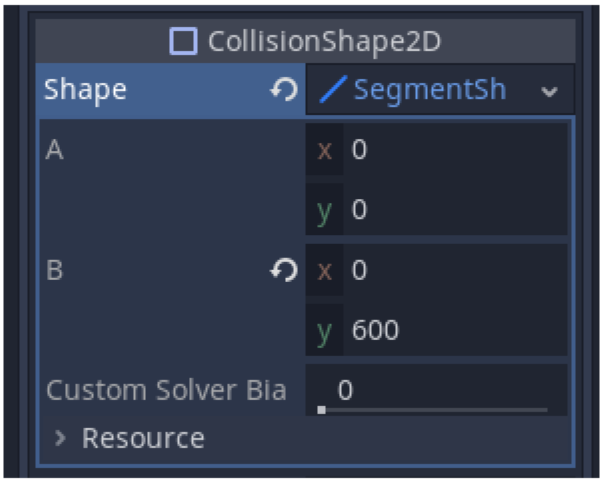
```
func _ready():
	$Alien.position = Vector2(get_viewport_rect().size.x / 4, get_viewport_rect().size.y - 300)
```

[in Main.gd]

Delete the entire _ready function. Things should work just like they used to. This is one of the really cool parts of Godot - refactoring and restructuring
the game.

Create a Limits scene – Area2D, call it Limits and save it as Limits.tscn

Now, in the Limits, add a CollisionShape2D, calling it Left. Set it's shape to SegmentShape2D, then click it. You can see that you can move the two points around. Make them cover the left side of the screen, either manually, or by setting the properties directly:


CollisionShape2D

Shape: SegmentSh

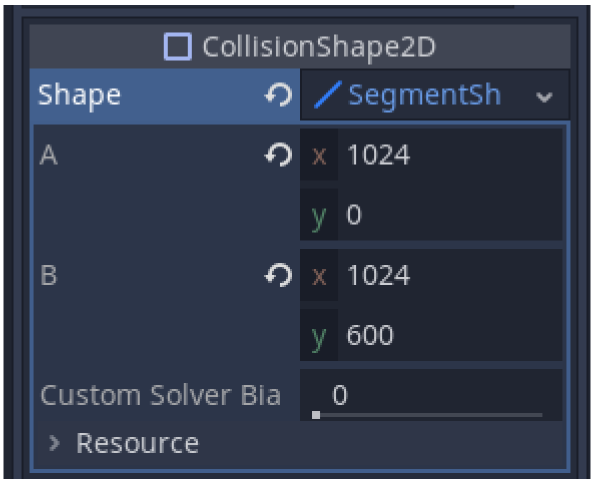
A: x 0, y 0

B: x 0, y 600

Custom Solver Bia: 0

Resource

Then do the same thing to the right side of the screen, creating a CollisionShape2D called right and setting it run down the right hand side of the screen:


CollisionShape2D

Shape: SegmentSh

A: x 1024, y 0

B: x 1024, y 600

Custom Solver Bia: 0

Resource

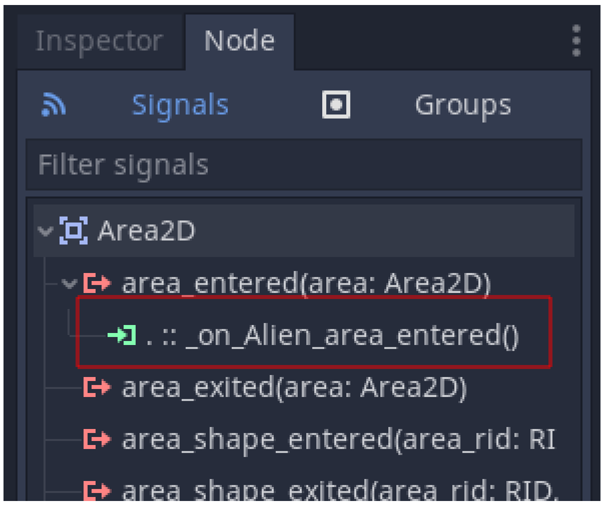
(1024x600)

Link the Limits scene to the Swarm scene.

So, these trigger specific collision detection for the Swarm - the aliens will hit it when they reach the edge

Run the game. Hmm, the alien explodes when it hits the side of the container. This is not what we want. That's because the alien will explode when it hits ANY other collision object. Lets refactor the _on_Alien_area_entered method - renaming it to destroy.

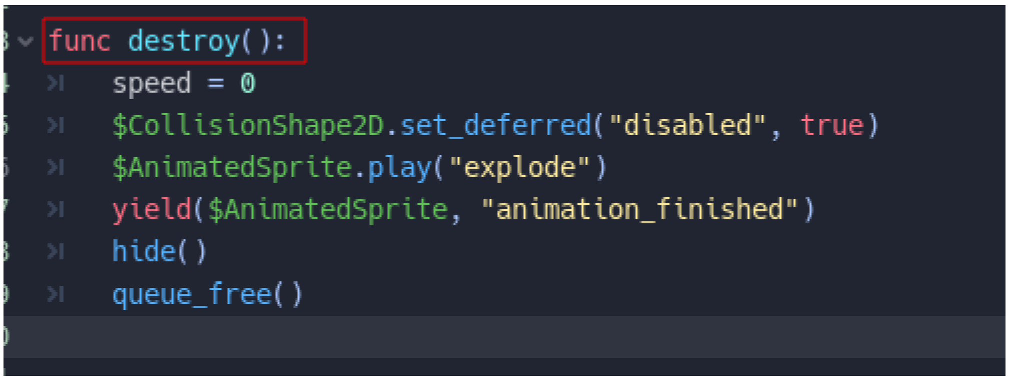
First, you will need to disconnect the signal (remember the little icon next to the function name. To do this, in the Alien.tscn, click on the Alien root node, the choose “Node” from the RHS panel so you can see the signals. Click on the “::_on_alien_area_entered()” signal identifier, and click “Disconnect” at the bottom of the screen:



```
Area2D
  area_entered(area: Area2D)
    . :: _on_Alien_area_entered()
  area_exited(area: Area2D)
  area_shape_entered(area_rid: RI
  area_shape_exited(area_rid: RID,
```

When you save the file, the green “signal indicator” will disappear.

Now rename the function:

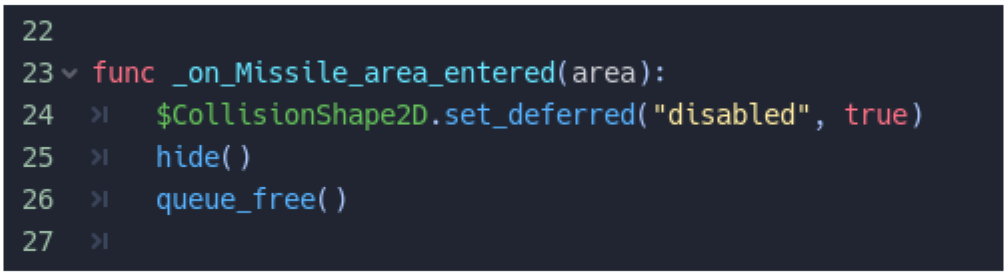


```
func destroy():
	speed = 0
	$CollisionShape2D.set_deferred("disabled", true)
	$AnimatedSprite.play("explode")
	yield($AnimatedSprite, "animation_finished")
	hide()
	queue_free()
```

[in Alien.gd]

(note, we also remove the ‘area’ parameter)

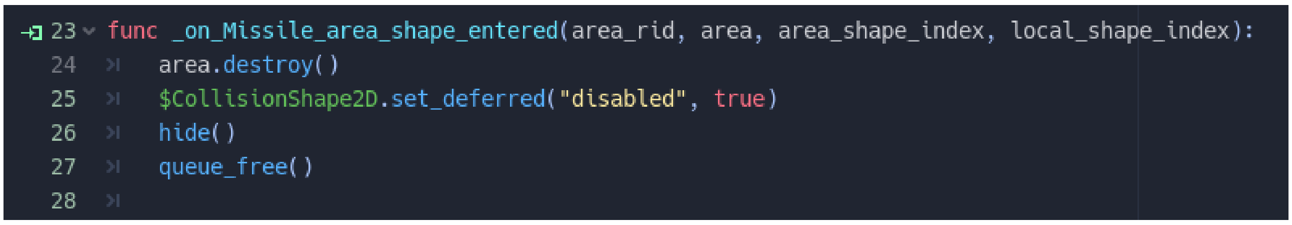
And now let's change the Missile around too, so that it is able to call this “destroy” method on whatever it hits. Do do this, disconnect the on_Missile_area_entered method in Missile.gd and change the method from this:



```

func _on_Missile_area_entered(area):
	$CollisionShape2D.set_deferred("disabled", true)
	hide()
	queue_free()
	
```

[in Missile.gd]

to this:



```
func _on_Missile_area_shape_entered(area_rid, area, area_shape_index, local_shape_index):
	area.destroy()
	$CollisionShape2D.set_deferred("disabled", true)
	hide()
	queue_free()

```

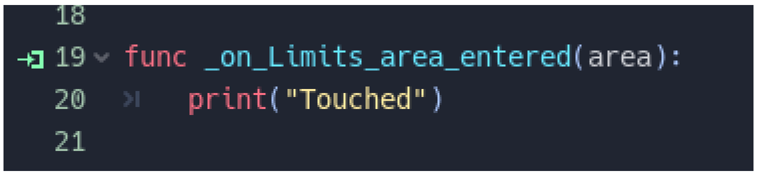
[in Missile.gd]

.. by connecting the area_shaped_entered of the Missile root node to the Missile scene.

Move most of the original body to here, and note the new area destroy - this will kill the alien when the missile hits it. The area_shape_entered() function lets you get a handle on the object you are colliding with, therefore you can interact with the alien (in this case, asking it to destroy itself)

We can now make it change direction:

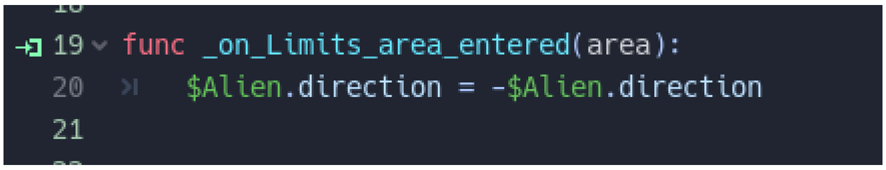
We'll do this is the Swarm
Create a new Script for the Swarm (called Swarm.gd), connect the limits#area_entered to the swarm



```

func _on_Limits_area_entered(area):
	print("Touched")

```

[in Swarm.gd]

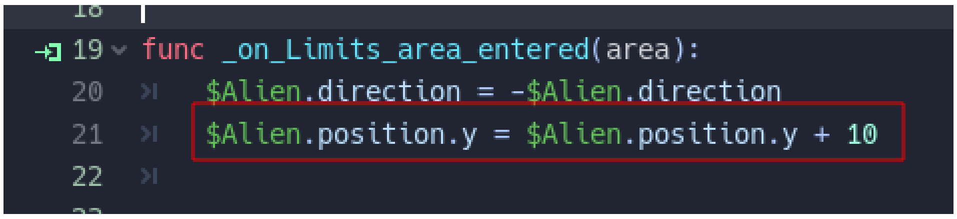
If you run the game now, the alien will detect it has touched the side of the screen and tell you. Cool, let's make the alien change direction:



```
func _on_Limits_area_entered(area):
	$Alien.direction = -$Alien.direction

```

[in Swarm.gd]

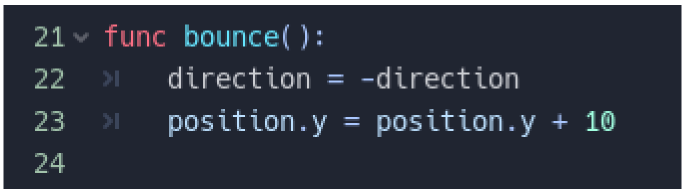
And we can also move it down a line:



```

func _on_Limits_area_entered(area):
	$Alien.direction = -$Alien.direction
	$Alien.position.y = $Alien.position.y + 10
	
```

[in Swarm.gd]

It’s probably better to move this code to the Alien.gd itself, since we are referring to the $Alien quite a lot. So, let’s move the code here to a new function in the Alien called bounce:

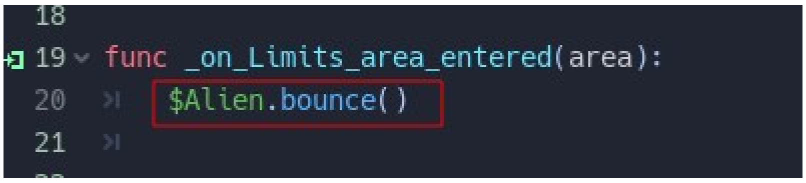


```
func bounce():
	direction = -direction
	position.y = position.y + 10
```

[in Alien.gd]

(note, we don’t need to keep referring to $Alien now – that was a sort of ‘code smell’ that perhaps something was not in the correct file.

And now we can change the Swarm to refer to this new ‘bounce’ method on the Alien:



```
func _on_Limits_area_entered(area):
	$Alien.bounce()
	
```

[in Swarm.gd]

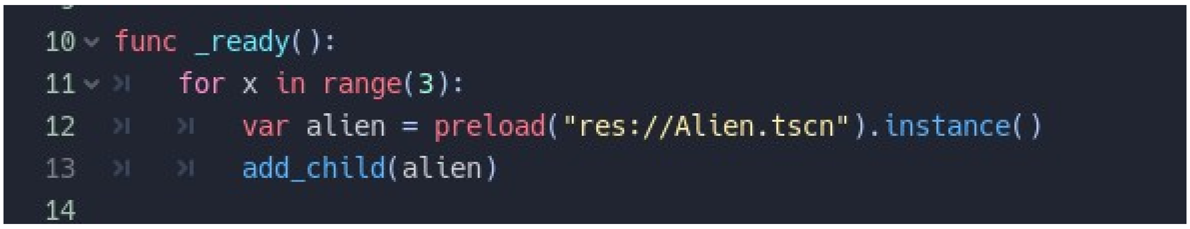
Notice that we also explode when we move our Gunship to the sides of the screen! We'll fix that with collision layers later.

All the code up until this point is available in the project in the “Part4” directory

It's time to make more aliens.

Let's make 3 to start with.

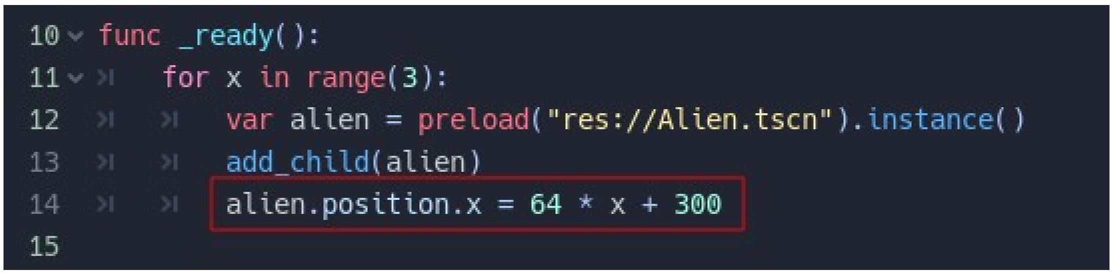
Working in the Swarm.gd file:



```
func _ready():
	for x in range(3):
		var alien = preload("res://Alien.tscn").instance()
		add_child(alien)
```

[in Swarm.gd]

Hmm, can't see them, (until they touch the side of the screen momentarily)
Probably because they are all on top of each other! Let’s fix that:



```
func _ready():
	for x in range(3):
		var alien = preload("res://Alien.tscn").instance()
		add_child(alien)
		alien.position.x = 64 * x + 300
```

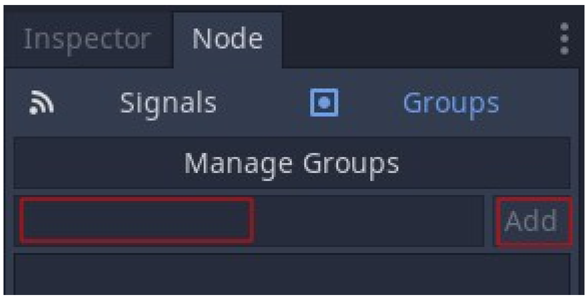
[in Swarm.gd]

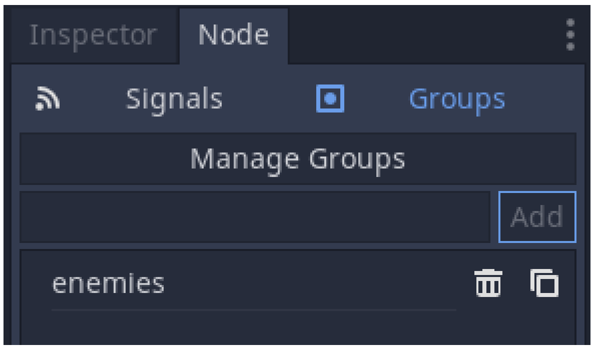
Run it! Hey - there's an extra one! And also, as each alien touches the wall, only one (the original) reacts

So, when an alien hits the wall, we trigger the _on_Limit_area_entered and it raises a signal - which is connected to just one alien....

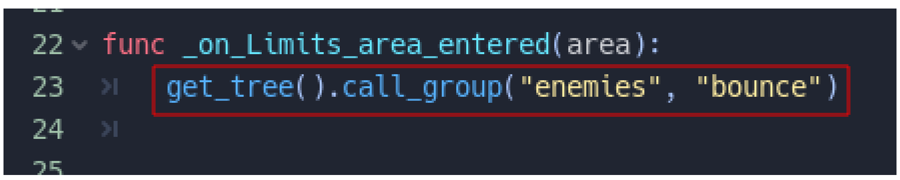
We need that all aliens be treated the same... That is, make a group..

Now, select the Alien scene, and the root Node (Alien) - Click 'Groups' and add a new group called 'enemies'





In the Swarm scene, replace the _on_Limits_area_entered code:

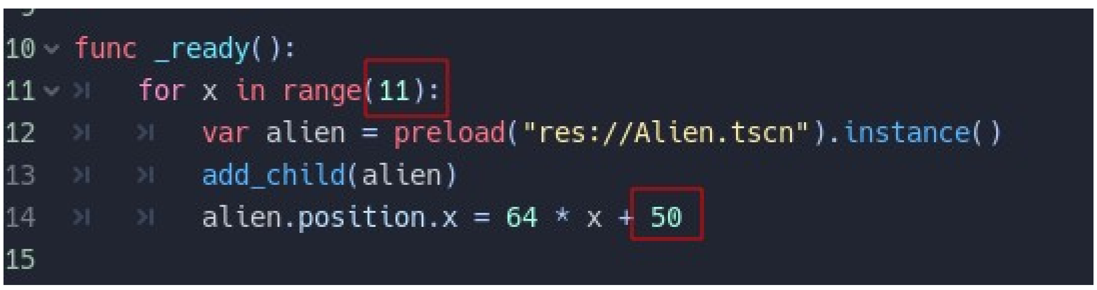


```
func _on_Limits_area_entered(area):
	get_tree().call_group("enemies", "bounce")
	
```

[in Swarm.gd]

Looks good! Remove the original Alien from the Swarm scene (by using ‘delete node’):

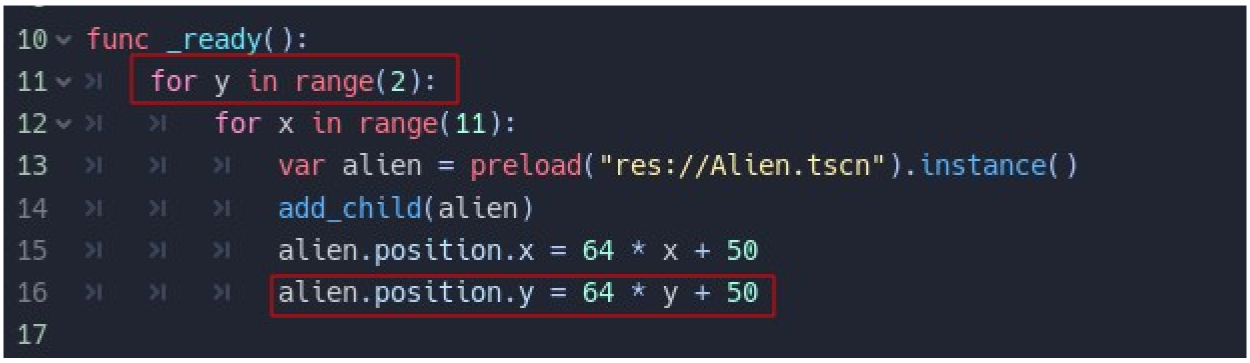
Lets up the number of aliens:



```
func _ready():
	for x in range(11):
		var alien = preload("res://Alien.tscn").instance()
		add_child(alien)
		alien.position.x = 64 * x + 50

```

[in Swarm.gd]

And then make 2 rows

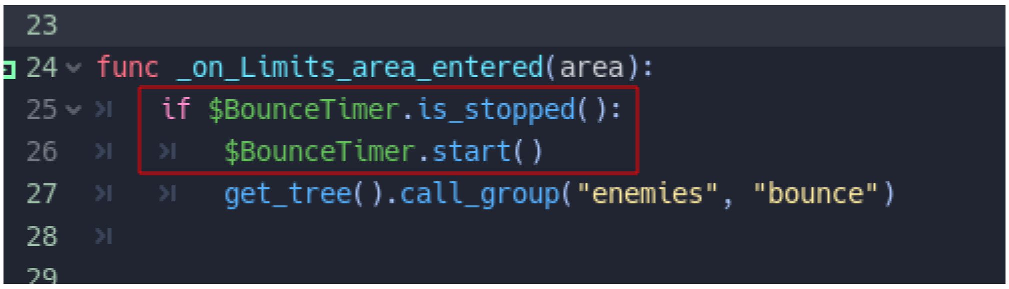


```
func _ready():
	for y in range(2):
		for x in range(11):
			var alien = preload("res://Alien.tscn").instance()
			add_child(alien)
			alien.position.x = 64 * x + 50
			alien.position.y = 64 * y + 50
```

[in Swarm.gd]

Run again - but we've got a bug.... Anyone guess? The _on_Limits_area_entered is getting called twice - one for each row. We only need to get that run once. Do do this we can use a little timer (a hack, but it'll work for us)

Add a 'Timer' called 'BounceTimer' to Swarm. Make it fire after .5 sec as a oneshot. Change the _on_Limits_area_entered to:



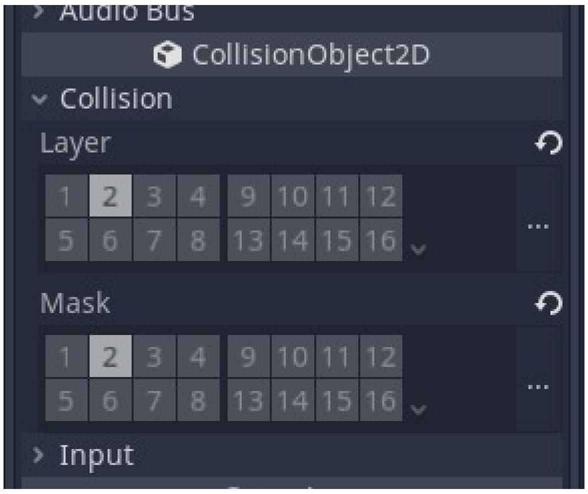
```
func _on_Limits_area_entered(area):
	if $BounceTimer.is_stopped():
		$BounceTimer.start()
		get_tree().call_group("enemies", "bounce")
	
```

[in Swarm.gd]

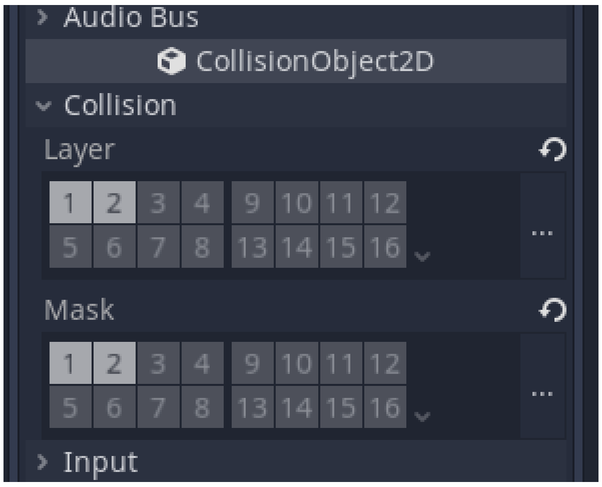
Fixing Bugs - crashing into the side of the screen makes us explode

We will change the collision layer.

Click Limits in Swarm Scene. Change Layer to 2 and mask to 2



Go to Alien scene, change Collision Layer to 1 and 2, and mask to 1 and 2 on the Root Node:



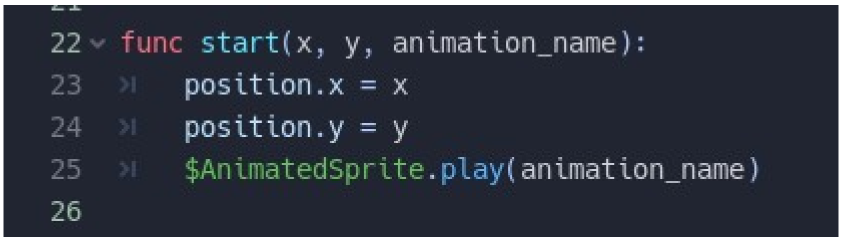
The Aliens are now the only things that can interact with the walls. The Gunship (on Layer one) will only interact with the Missle and the Aliens, not the walls.

All the code up until this point is available in the project in the “Part5” directory

We can now add a few more alien types

First, let's do a little bit of refactoring:

Let’s make a method in the Alien, called start, that sets the position of the alien, and specifies the ‘shape’ of the alien (by changing the animation):

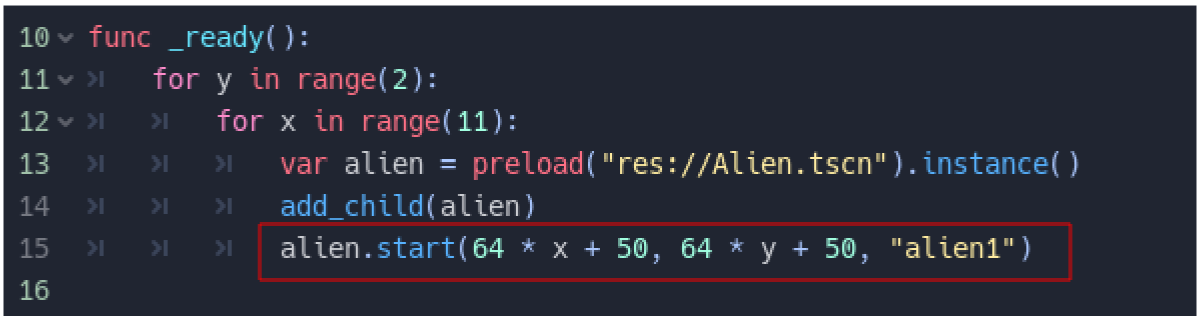


```
func start(x, y, animation_name):
	position.x = x
	position.y = y
	$AnimatedSprite.play(animation_name)

```

[in Alien.gd]

Remove the ‘_ready()’ function from the Alien, since we’ll set the animation running in the start function instead.

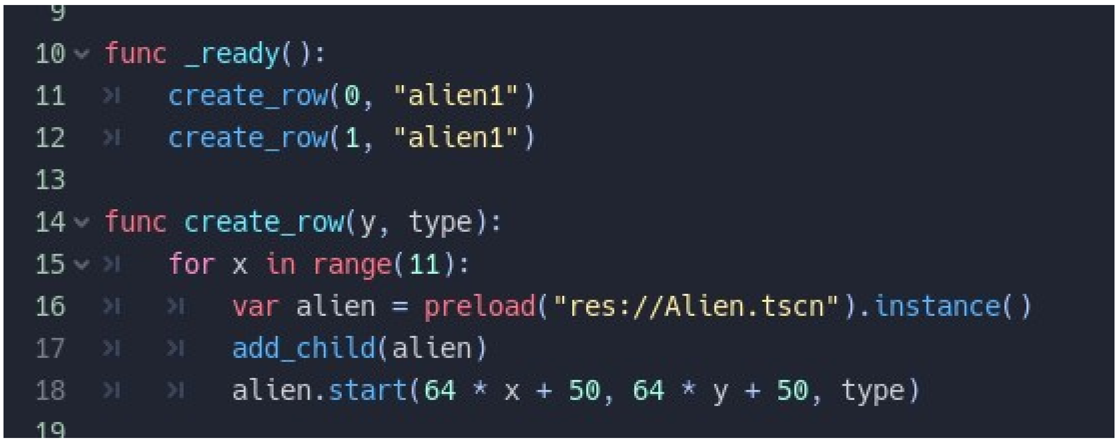
Then change, the _ready() of the swarm to use this new code:



```
func _ready():
	for y in range(2):
		for x in range(11):
			var alien = preload("res://Alien.tscn").instance()
			add_child(alien)
			alien.start(64 * x + 50, 64 * y + 50, "alien1")

```

[in Swarm.gd]

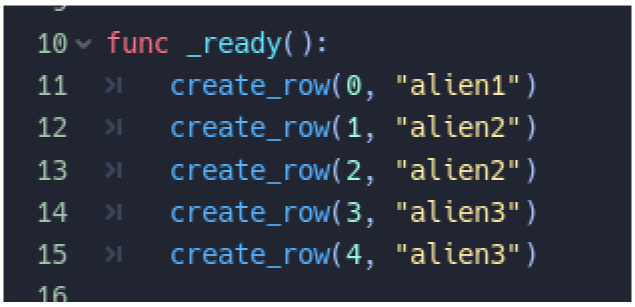
It would also be nice to split the creation of the rows out into their own function and simplify the _ready() function:



```
func _ready():
    create_row(0, "alien1")
    create_row(1, "alien1")

func create_row(y, type):
    for x in range(11):
        var alien = preload("res://Alien.tscn").instance()
        add_child(alien)
        alien.start(64 * x + 50, 64 * y + 50, type)
```

[in Swarm.gd]

Now let's go wild with the alien swarm!:

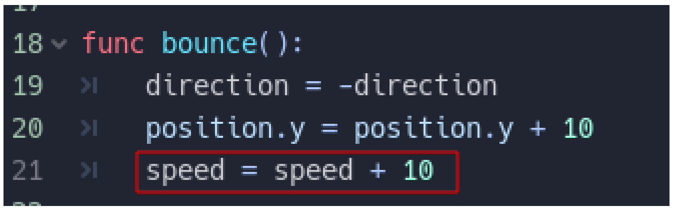


```
func _ready():
    create_row(0, "alien1")
    create_row(1, "alien2")
    create_row(2, "alien2")
    create_row(3, "alien3")
    create_row(4, "alien3")
```

[in Swarm.gd]

That makes 5 rows of different types of aliens... Just like the original arcade

We can make them faster as they go down lines:

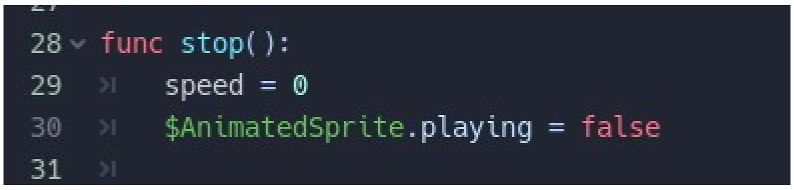


```
func bounce():
    direction = -direction
    position.y = position.y + 10
    speed = speed + 10
```

[in Alien.gd]

Also, let's make them all stop when they hit the turret:

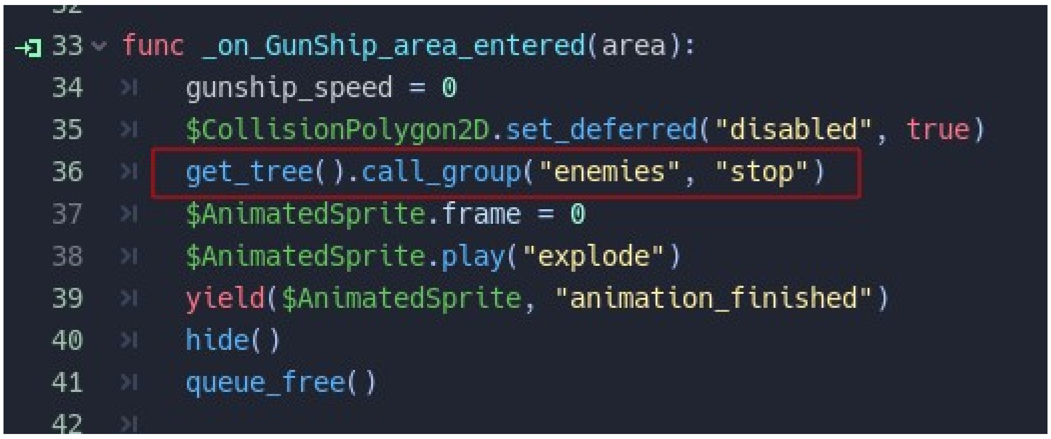
In Alien.gd:



```
func stop():
    speed = 0
    $AnimatedSprite.playing = false
```

[in Alien.gd]

In Gunship, _on_Gunship_area_entered()



```
func _on_GunShip_area_entered(area):
	gunship_speed = 0
	$CollisionPolygon2D.set_deferred("disabled", true)
	get_tree().call_group("enemies", "stop")
	$AnimatedSprite.frame = 0
	$AnimatedSprite.play("explode")
	yield($AnimatedSprite, "animation_finished")
	hide()
	queue_free()
```

[in Gunship/gd]

That's pretty much it.

All the code up until this point is available in the project in the “Part6” directory

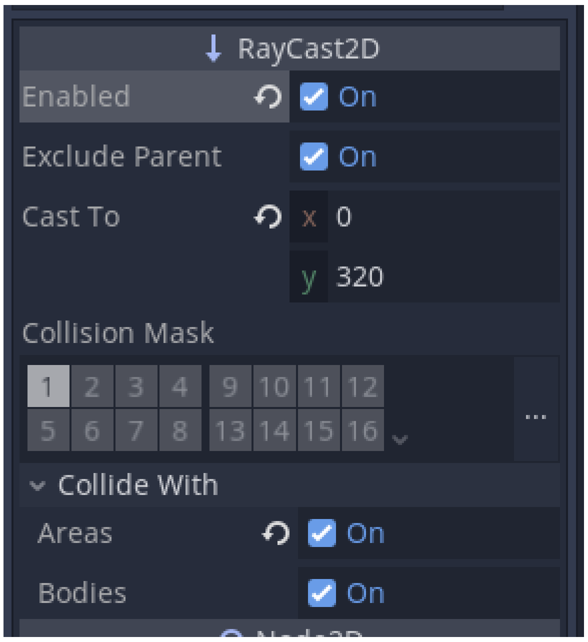
Lets look at improvements.

Letting the aliens fire back.

We need to equip the aliens with a missle system of their own..

First, only the bottom row of aliens can shoot - the easiest way to figure that out is to use RayCast2D.

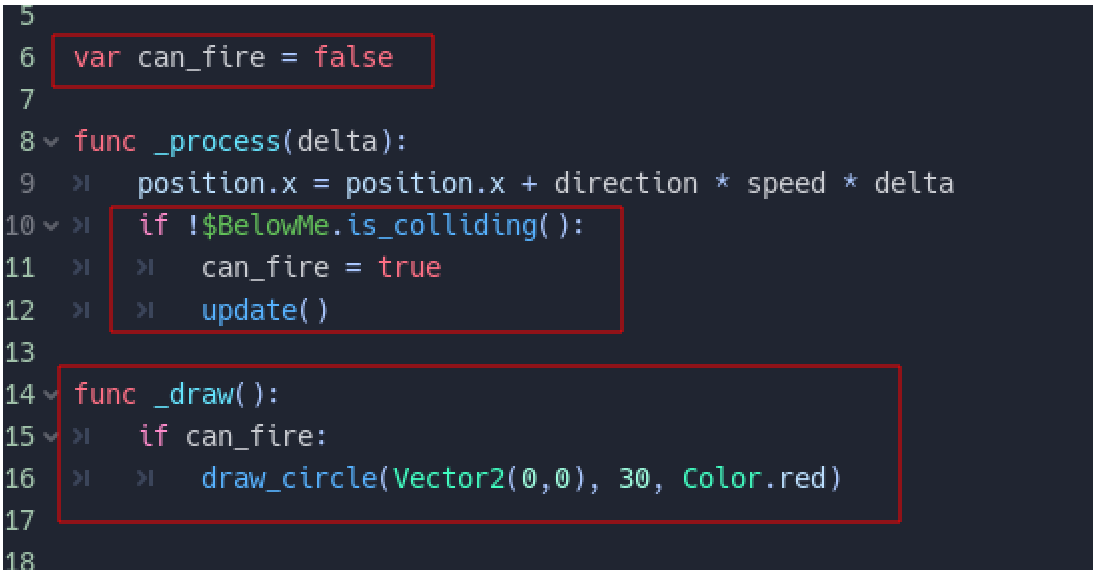
Add a RayCast2D node to the Alien scene, call it BelowMe, set enabled, Collide with "areas" and "Cast To" x:0, y:320 This makes a line-of-sight area below the alien that will be checked for collisions. If there is an alien below, we should not try firing the gun.



RayCast2D

Enabled	✓ On
Exclude Parent	✓ On
Cast To	x 0
	y 320
Collision Mask	1
Collide With	
Areas	✓ On
Bodies	✓ On

To see which aliens are able to shoot, lets do the following:



```
var can_fire = false

func _process(delta):
	position.x = position.x + direction * speed * delta
	if !$BelowMe.is_colliding():
		can_fire = true
		update()

func _draw():
	if can_fire:
		draw_circle(Vector2(0,0), 30, Color.red)
```

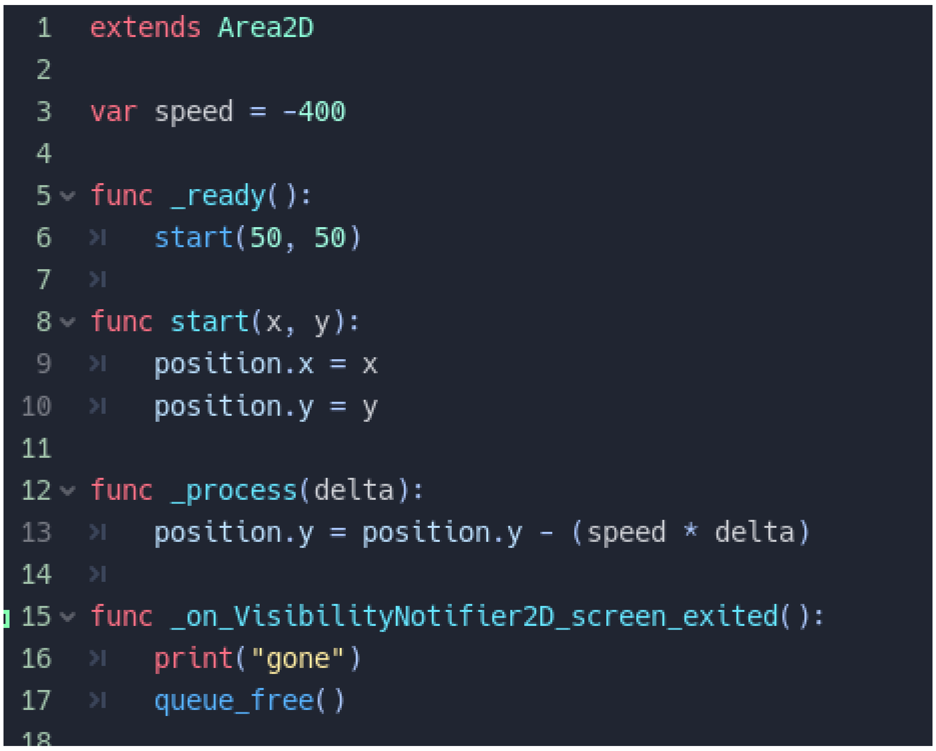
[in Alien.gd]

Now, when running, we can see red circles around the aliens that can fire - and have a handy variable called "can_fire" which we can check to see if we can shoot.

Make a new Scene (Area2D) called "AlienBomb", with an AnimatedSprite of the bomb (bolt_b_64.png). Add a collision shape.

Add a VisibilityNotifier2D, to free up the Bomb when it leaves the screen. Make sure you connect the signal (use some print debugging to confirm)

Add a simple script, just like the Missile, but going in the opposite direction:



```
extends Area2D

var speed = -400

func _ready():
	start(50, 50)
	
func start(x, y):
	position.x = x
	position.y = y

func _process(delta):
	position.y = position.y - (speed * delta)
	
func _on_VisibilityNotifier2D_screen_exited():
	print("gone")
	queue_free()
```

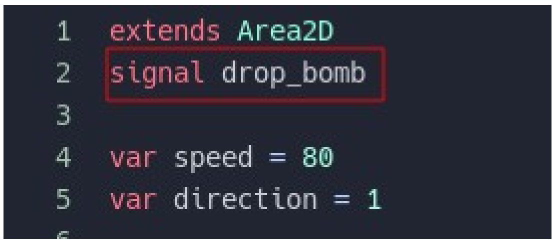
[in AlienBomb.gd]

Running the scene will allow you to see if it works okay.

To make it fire, lets' add a timer to to the aliens to make them fire a missile every second, but only if they can fire:

[Remove the _ready() from the AlienBomb – it was only used for testing]

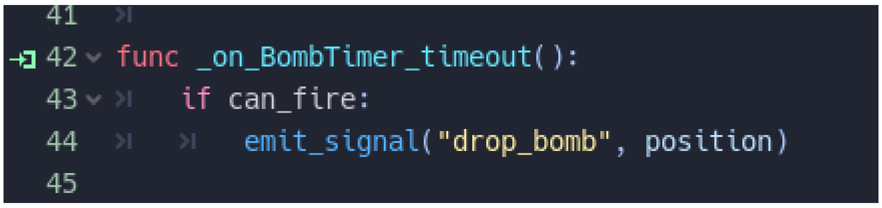
add a new Signal to the Alien:



```
extends Area2D
signal drop_bomb

var speed = 80
var direction = 1
```

[in Alien.gd]

Add a BombTimer to the Alien. Set WaitTime to 1, AutoStart to On, Link the timeout() signal to a method in the Alien

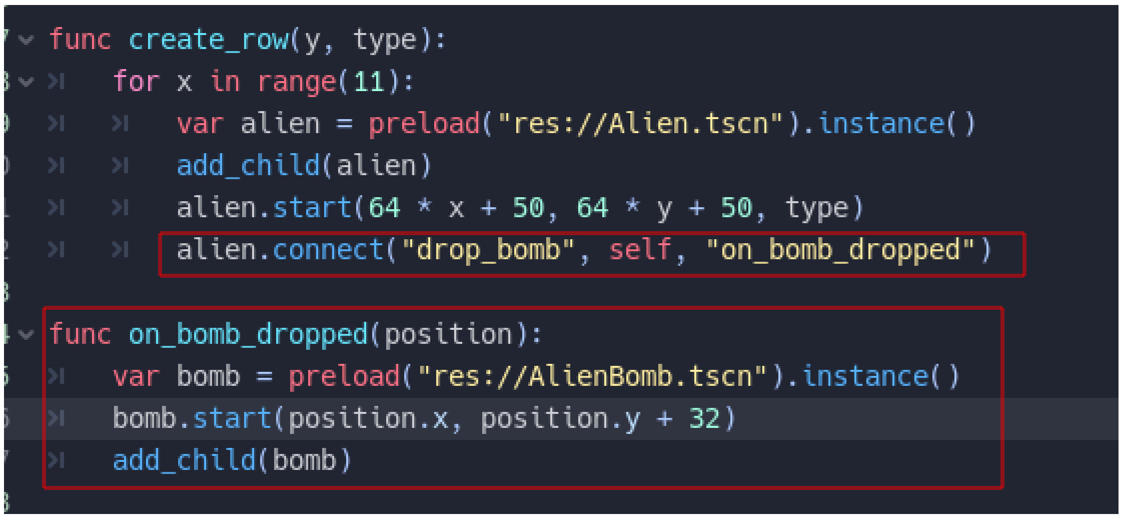


```

func _on_BombTimer_timeout():
	if can_fire:
		emit_signal("drop_bomb", position)

```

[in Alien.gd]

```
func _on_BombTimer_timeout():
    if can_fire:
        emit_signal("drop_bomb", position)
```

In the Swarm scene, connect the aliens we create to a on_bomb_dropped method, and respond to the "drop_bomb", this time using code:



```
func create_row(y, type):
	for x in range(11):
		var alien = preload("res://Alien.tscn").instance()
		add_child(alien)
		alien.start(64 * x + 50, 64 * y + 50, type)
		alien.connect("drop_bomb", self, "on_bomb_dropped")

func on_bomb_dropped(position):
	var bomb = preload("res://AlienBomb.tscn").instance()
	bomb.start(position.x, position.y + 32)
	add_child(bomb)

```

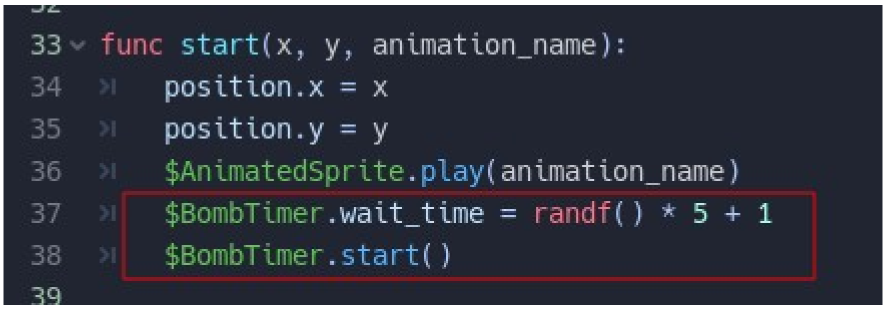
[in Swarm.gd]

Running the game is a bit hard! Also, the aliens are all firing at the same time.

We can drop the red circles now, by removing the _draw function, and the update() in _process of the Alien.gd

Turn the autostart off on the BombTimer, and switch it to One-Shot

In the Alien Start:

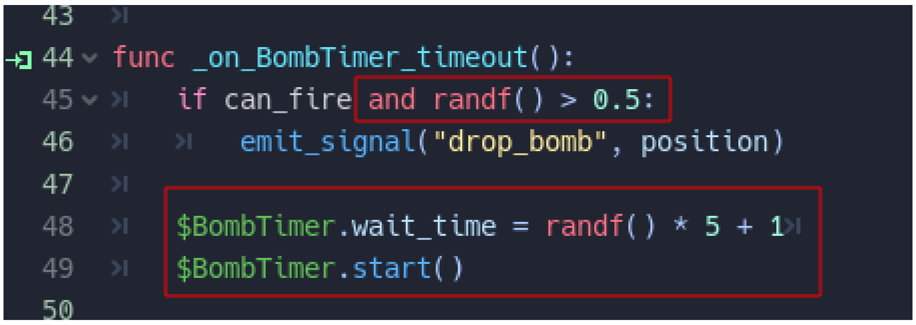


```
func start(x, y, animation_name):
	position.x = x
	position.y = y
	$AnimatedSprite.play(animation_name)
	$BombTimer.wait_time = randf() * 5 + 1
	$BombTimer.start()
```

[in Alien.gd]

This will trigger the timer a random interval between 1 and 5 seconds

Now need to re-queue on the timer timeout:



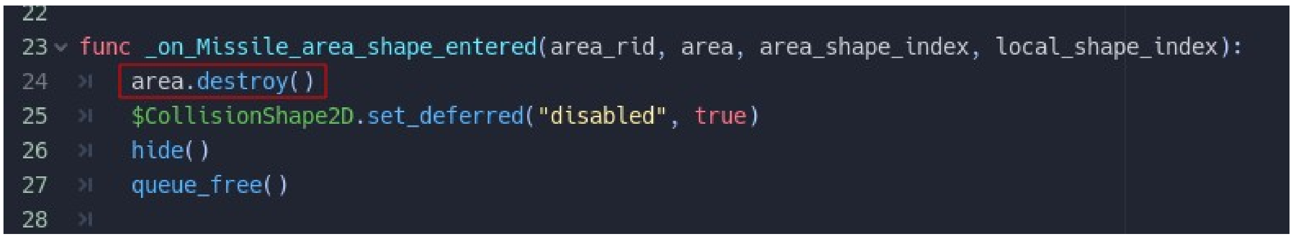
```
func _on_BombTimer_timeout():
	if can_fire and randf() > 0.5:
		emit_signal("drop_bomb", position)

	$BombTimer.wait_time = randf() * 5 + 1
	$BombTimer.start()
```

[in Alien.gd]

Note, we make the game a little easier by only 50:50 times the timer times-out do we actually drop a bomb

Shooting an alien bomb will cause the game to crash:

This is caused by:

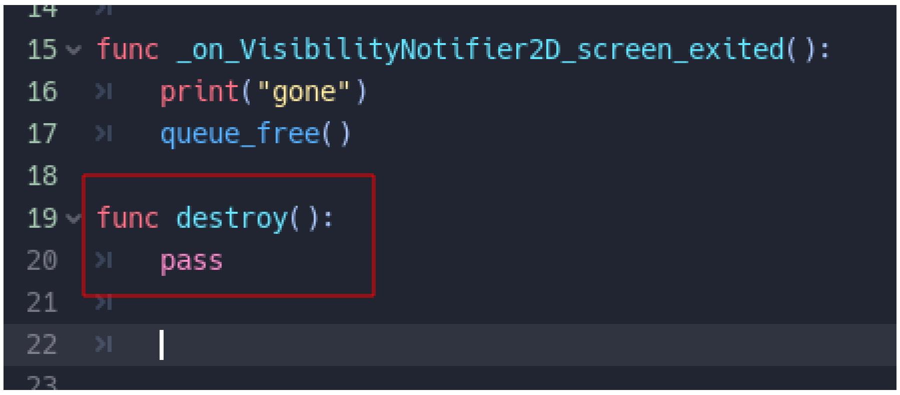


```
func _on_Missile_area_shape_entered(area_rid, area, area_shape_index, local_shape_index):
	area.destroy()
	$CollisionShape2D.set_deferred("disabled", true)
	hide()
	queue_free()
```

[in Missile.gd]

The area in this case will be an alien bomb - which cannot be destroyed...

Fix it by adding a destroy method in the AlienBomb:

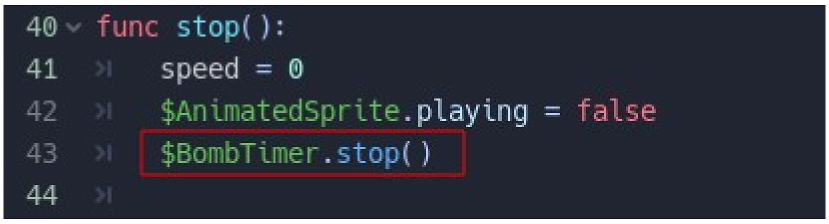


```
func _on_VisibilityNotifier2D_screen_exited():
	print("gone")
	queue_free()

func destroy():
	pass


```

[in AlienBomb.gd]

(Maybe later we'll look at how to destroy them)

Finally, to make sure the Aliens stop bombing you when you are destroyed:



```
func stop():
	speed = 0
	$AnimatedSprite.playing = false
	$BombTimer.stop()

```