CODE NINJAS®

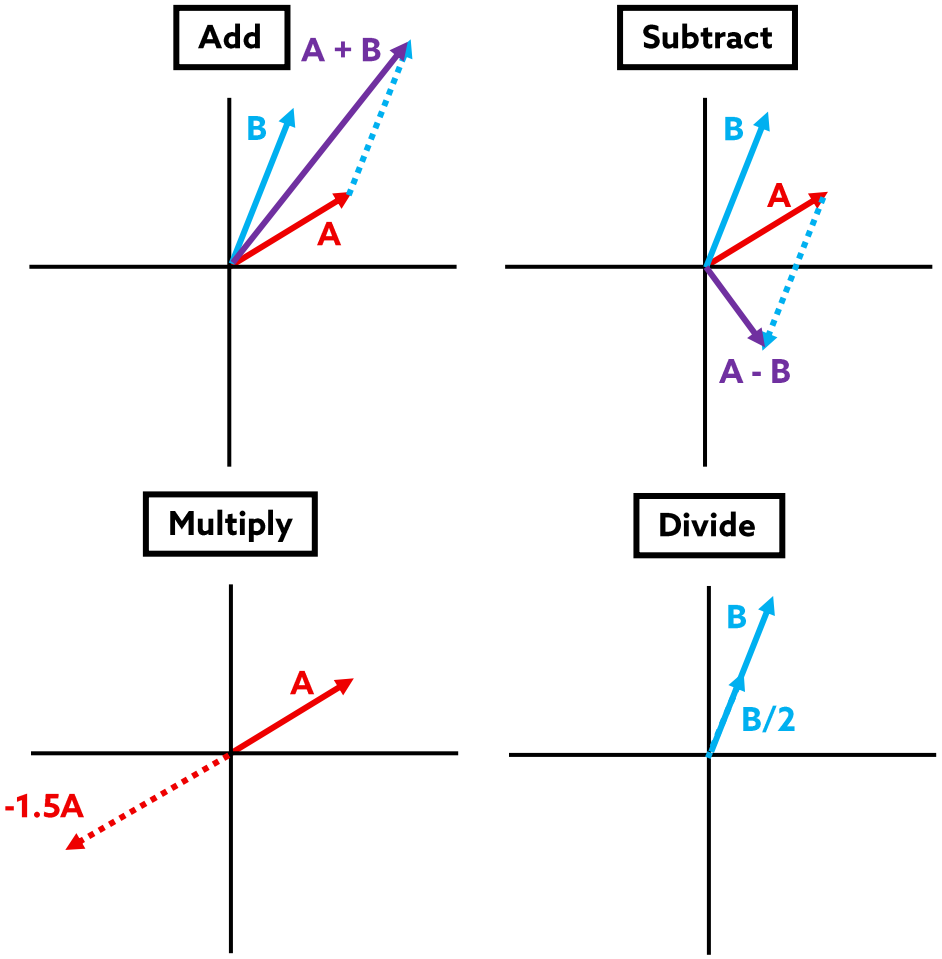
Silver Belt Ninja Guide

Activity 03: Shape Jam

Vector Math

Vectors can be added to and subtracted from **each other** by lining them up end-to-end and drawing the resulting vector from the origin. They can also be multiplied or divided by a **scalar** to change the length (magnitude) and, if the scalar is negative, can even flip their direction.

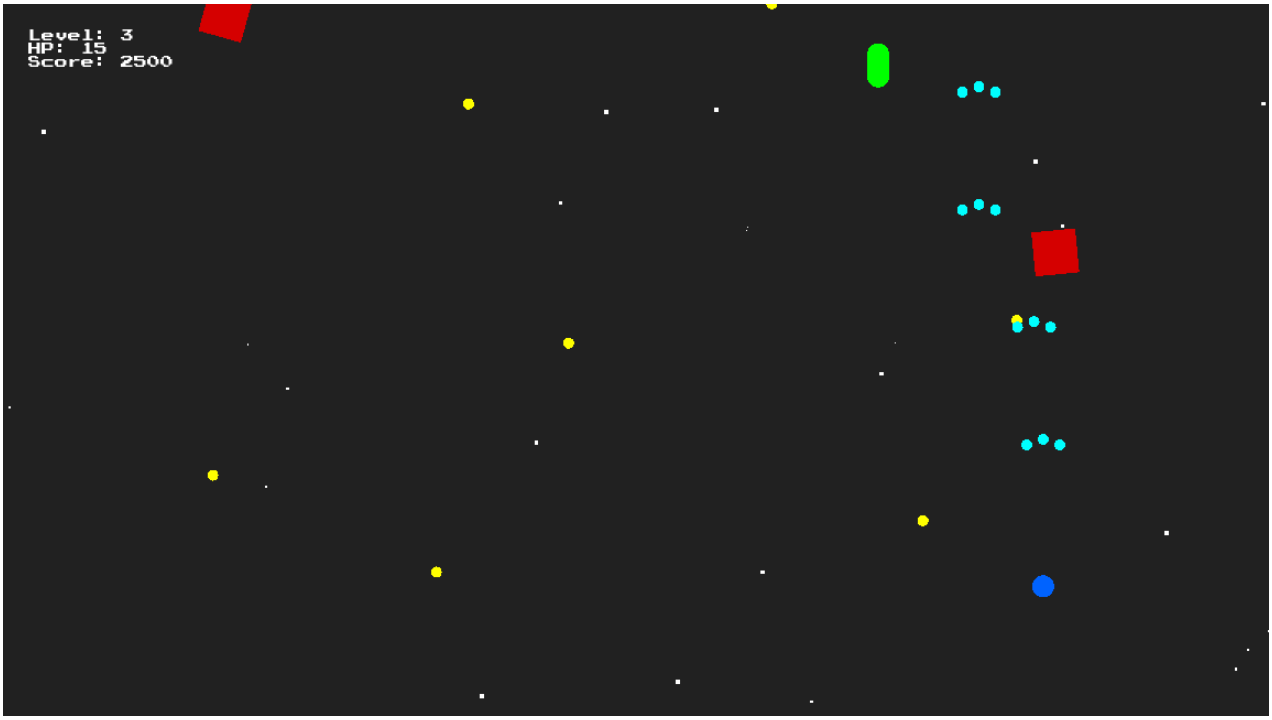
Below are some visual examples with Vector2, using vectors A and B. The dashed lines show the steps that lead to the result:



Activity 03: Shape Jam

Your mission: A nearly complete game is given to you, but the most essential functions are missing – those that create new objects in the scene. In this activity, you will add code that creates these new objects and elements within the game.

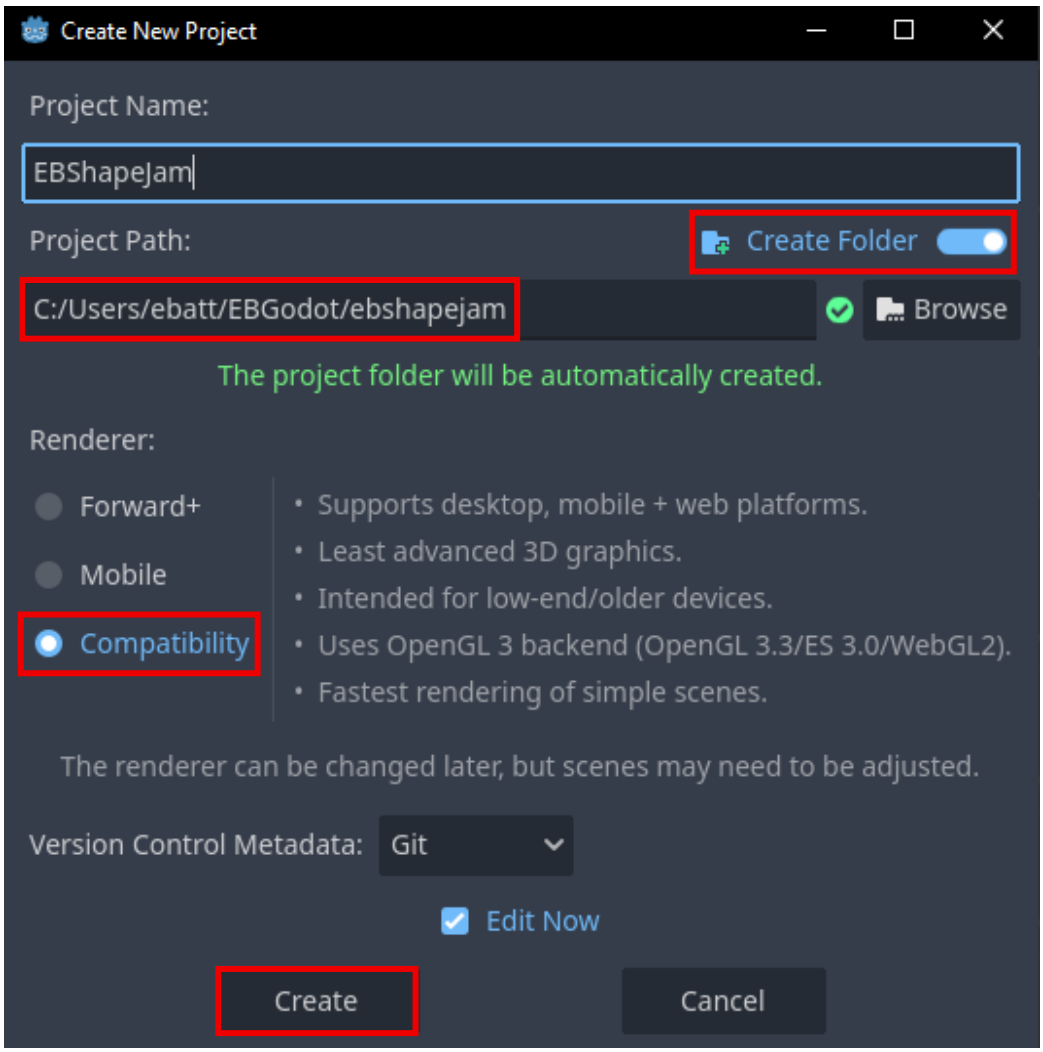
By the end of the project, the player will be able to fire projectiles, enemies and powerups will spawn at regular intervals, and enemies will constantly fire hazardous objects at the player. You will learn an advanced pattern to instantiate new objects in the scene!



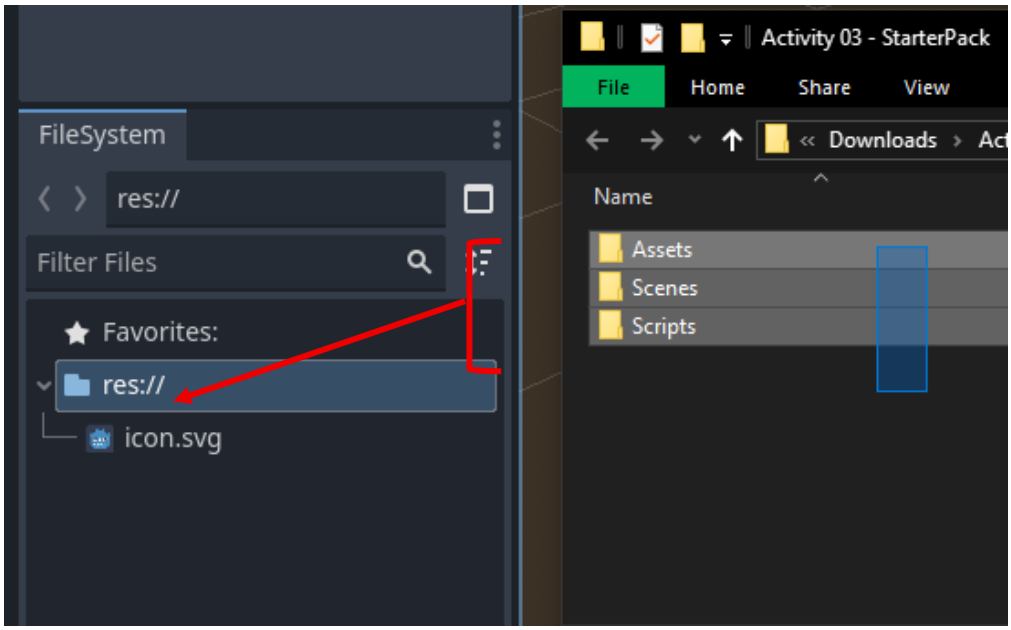
1 After opening Godot, in the top left corner select **+ Create**.

A **Create New Project** window will pop up. Name the project **[YourInitials]ShapeJam.**

Enable **Create Folder** if it is not already enabled. Make sure the **Project Path** is the same as previously set by a Code Sensei at your center. Ensure that the project is in **Compatibility** mode, then click **Create**.

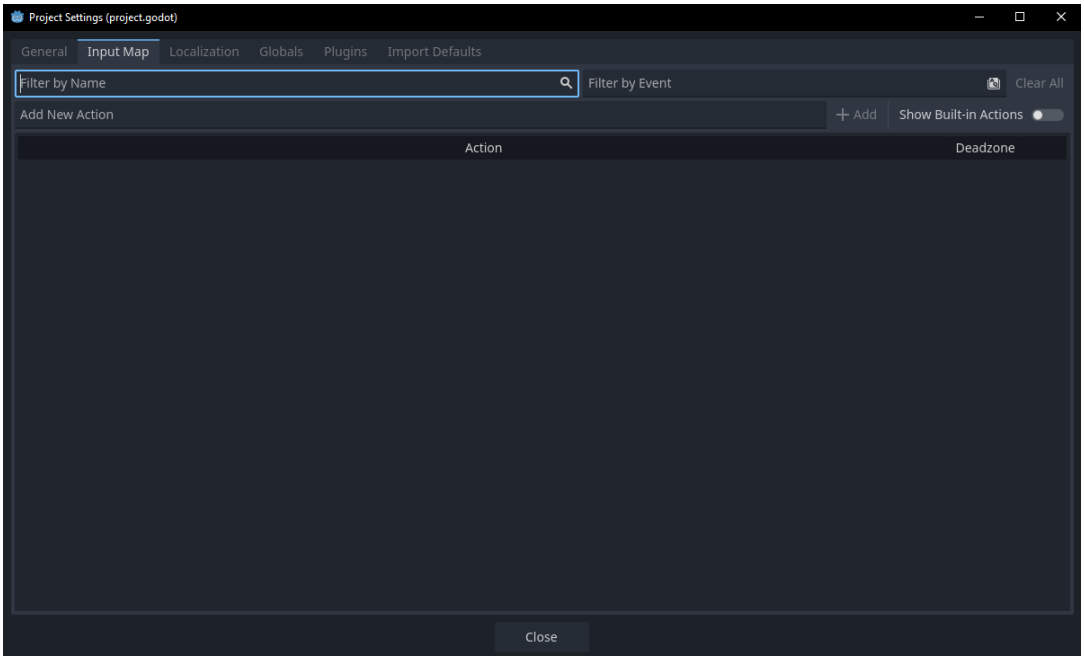


2 Extract **SB Activity 03 – Ninja Starter Pack.zip** and drag all the folders into res:// in **FileSystem**.



3 The project scripts already use specific actions in their code that must be created in the Input Mapping.

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



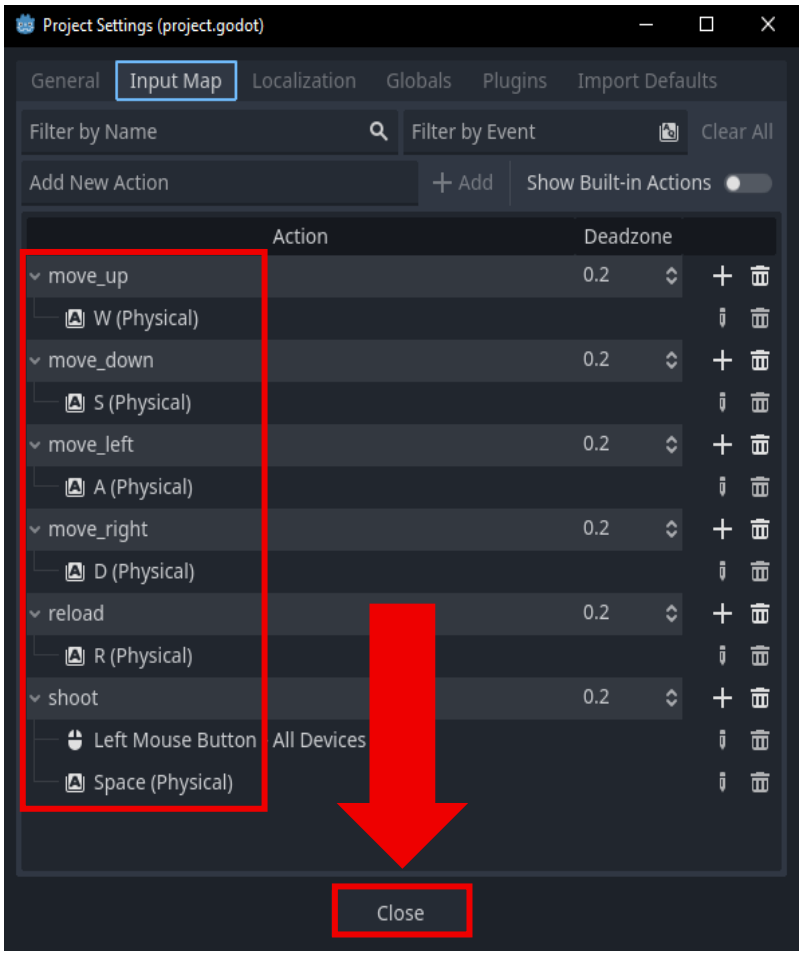
4

Add the following actions to the project:

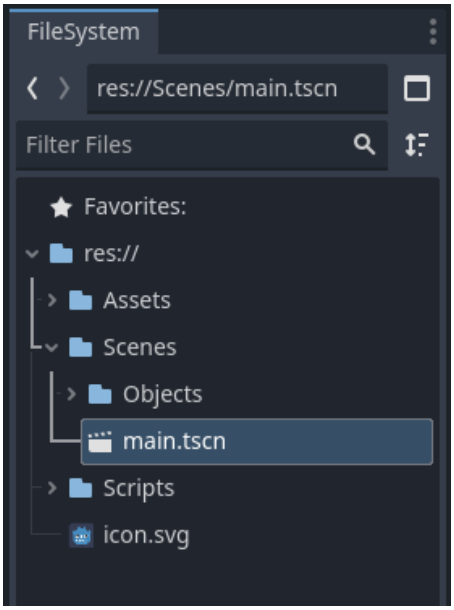
- W – **move_up**
- S – **move_down**
- A – **move_left**
- D – **move_right**
- R – **reload**
- Left Click/Space – **shoot**

Make sure the spelling is exact, or the scripts will not work as intended. The left click event must be clicked inside of the "Listening for Input..." text box for it to be recorded.

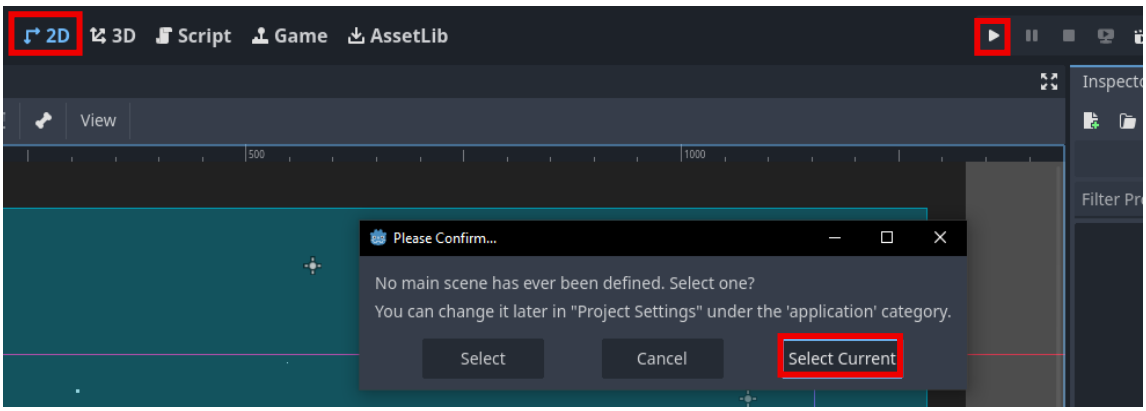
Click **Close**.



5 In **FileSystem**, navigate to **main.tscn** and double-click to open the scene.

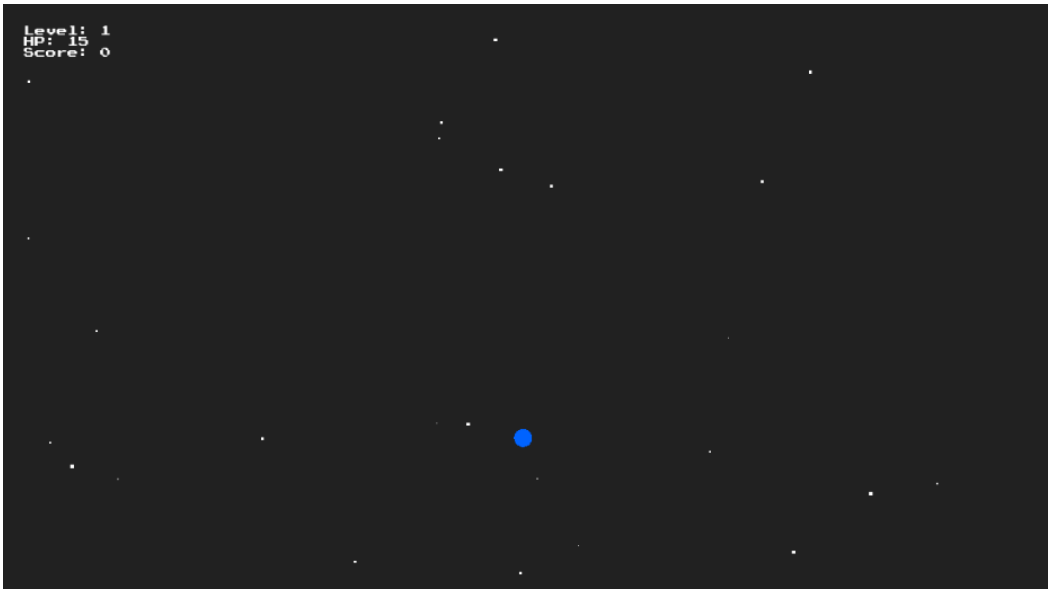


Since this is a fully 2D game, set the workspace to be **2D**. In the top right corner, start a playtest and **Select Current** as the main scene.



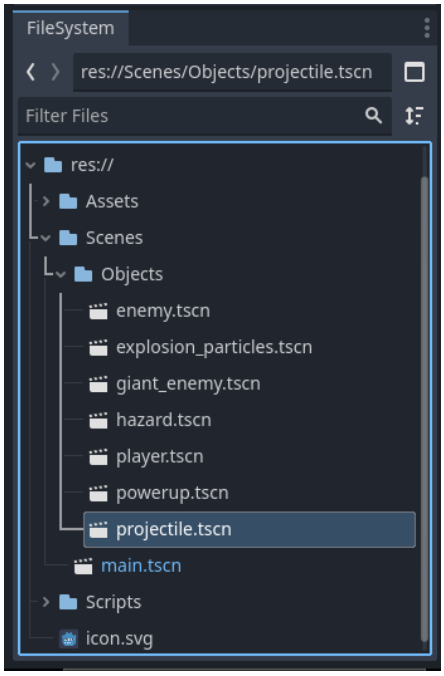
6 What happens when the WASD keys are pressed? What about the R key and the space bar?

Notice that the player can move around the game window but cannot fire projectiles and the enemies are not spawning!

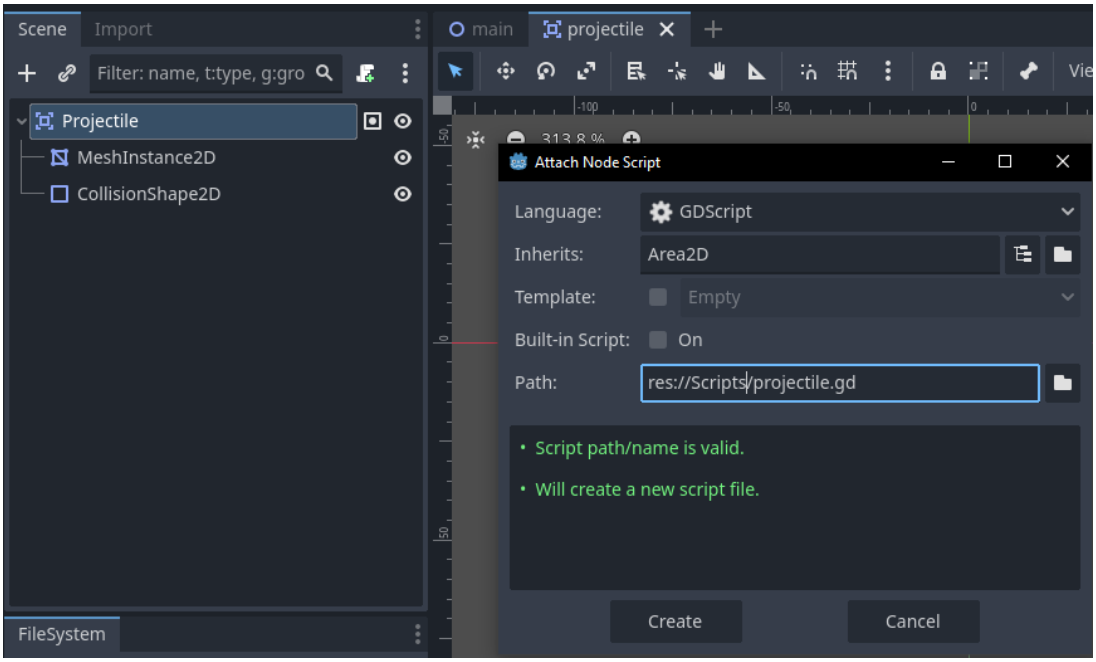


7 Since this game will have several different projectiles, it will be handy to create a generic projectile script that constantly moves a node given a speed and direction.

Navigate to **projectile.tscn** in **FileSystem** and double-click to open the scene.



Select **Projectile** and attach a new script named projectile.gd, then click **Create**.



8 Create a **`speed`** variable of type **`float`** and a **`direction`** variable of type **`Vector2`**, without initial values.

Underneath define the **`_process()`** function with the parameter **`delta.`**

```
1   extends Area2D
2
3   # speed ???
4   # direction ???
5
6   # _process ???
```

9 Check the code! Update the script as needed.

```
1  extends Area2D
2
3  var speed: float
4  var direction: Vector2
5
6  func _process(delta):
7      
```

10 One approach to make something move over time is by calling `translate()` by small amounts every game update.

Translate is a simple method that exists for all Node2D/3D that allows node's position to be offset by a given Vector2D/3D.

> `translate()`: part of the Node2D/3D classes, changes the node's position by the given Vector2D/3D offset.
>
> **Parameters:**
>
> 1. **offset *(Vector2)*:** the direction and magnitude of the desired translation
>
> **Returns *(void)*:** No return value but updates the position of the node by adding the provided Vector2D/Vector3D offset to it.

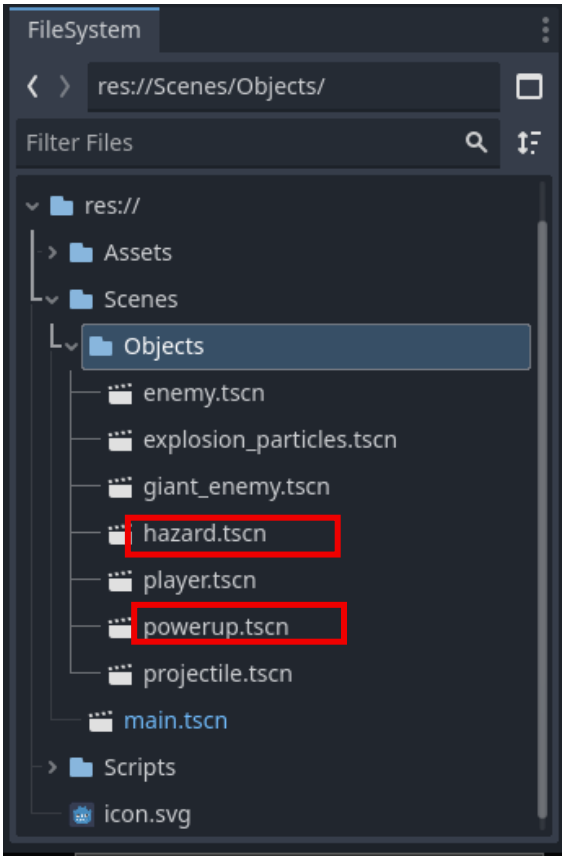
11 Inside the `_process()` function, call translate with the parameter as the three values multiplied together: `direction, speed, delta.`

```
1   extends Area2D
2
3   var speed: float
4   var direction: Vector2
5
6   func _process(delta):
7       # translate ???
```

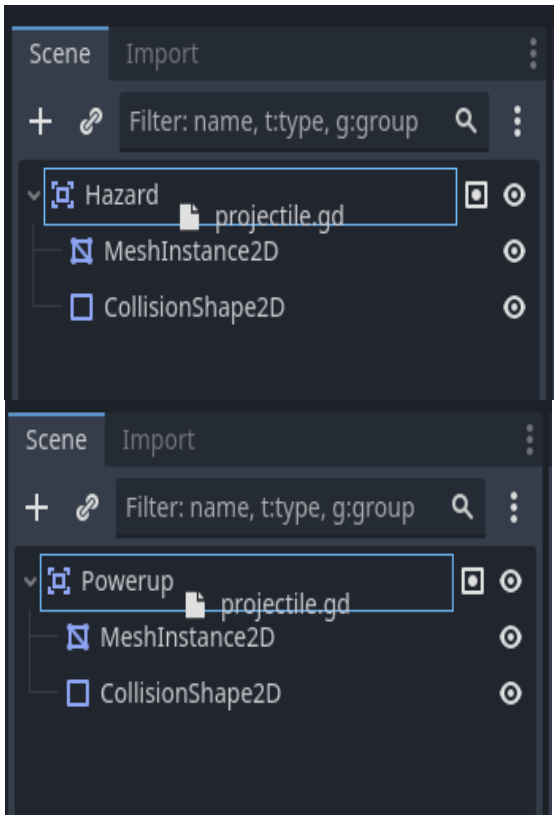
12 Check the code! Update the script as needed.

```
1   extends Area2D
2
3   var speed: float
4   var direction: Vector2
5
6   func _process(delta):
7       translate(direction * speed * delta)
8
```

13 Now that the Player's projectiles can be given a direction and speed, the same should be done for the Enemies' **hazardous** projectiles and the **powerups** which will help the player get stronger!

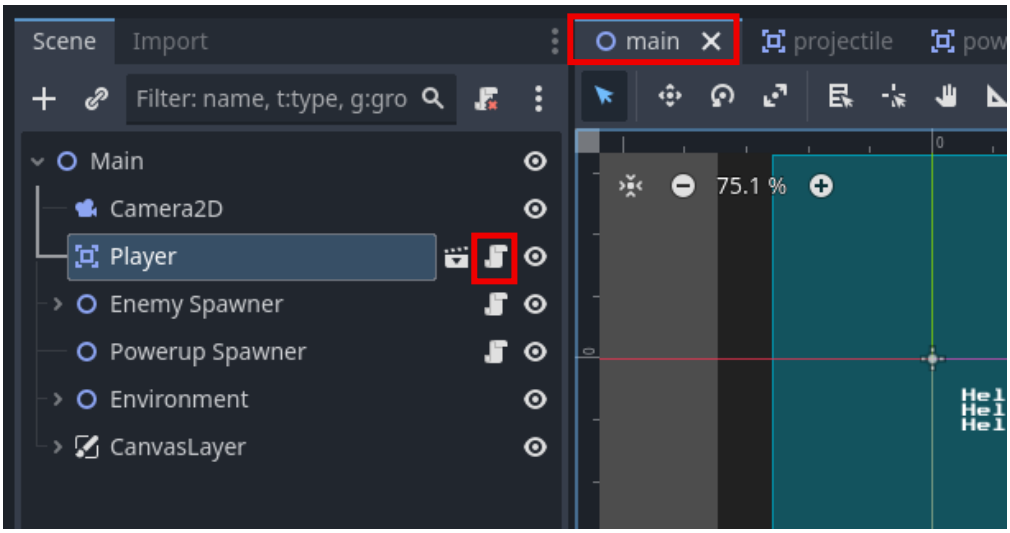


In **FileSystem**, open the **hazard.tscn** and **powerup.tscn** scenes. Attach **projectile.gd** to the root nodes of both scenes by dragging it from **FileSystem**.



14 Navigate back to **main.tscn** and enter the Player's attached script, **player.gd.**

The existing code in **player.gd** handles collision with other projectiles and handles the player's movement.



Pro Tip:

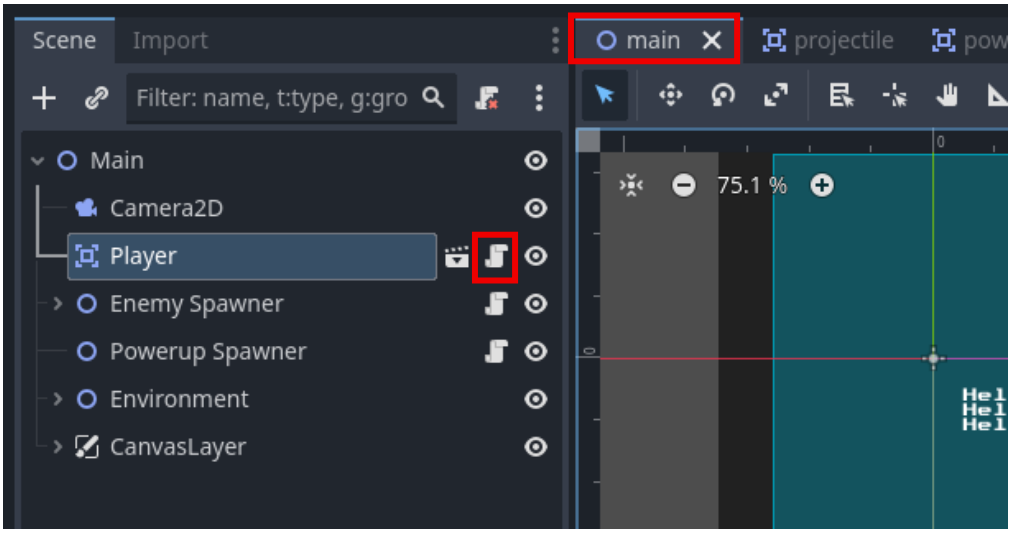
This game uses `Input.get_vector()` instead of `Input.get_axis()` to get the player controls. How does it work?

15

Navigate to **TODO 1**.

The code underneath can only run when the player is pressing the **shoot** action and is limited in frequency by **fire_cooldown**.

In other words, it's the perfect place to create the player's projectiles!



16

The export variable **projectile_scene** is already linked to projectile.tscn. Create a new variable named `projectile` of type `Area2D` and set it equal to `projectile_scene.instantiate()`.

On the next two lines, set the projectile's `direction` and `speed` variables that were created in **projectile.gd**!

Set `direction` to `Vector2.UP` and `speed` to the export variable `projectile_speed`. This will kick off the projectile's movement!

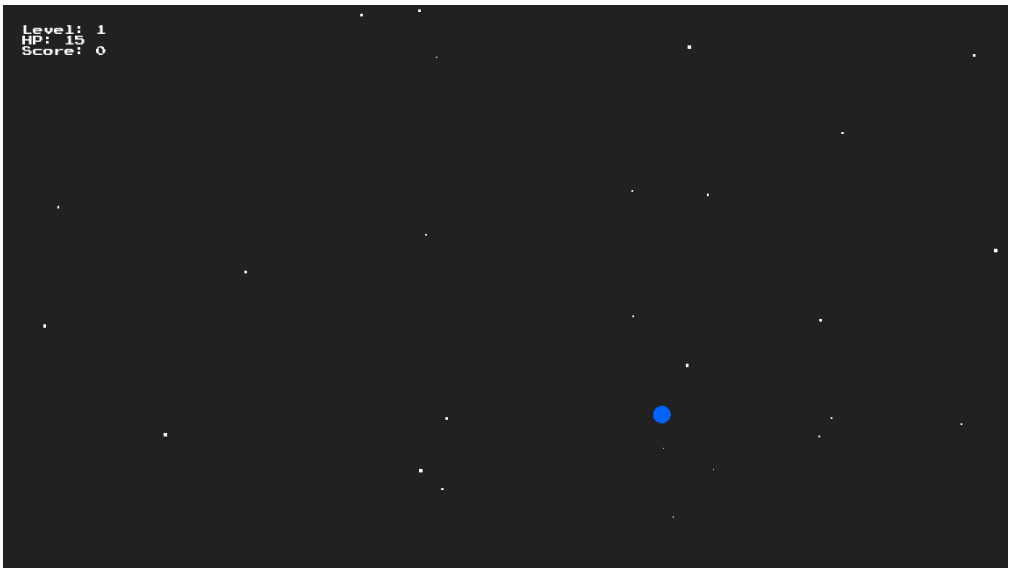
```
# --------------------
# TODO 1 & 5
# Player projectiles
# --------------------
# projectile ???
# projectile.direction ???
# projectile.speed ???
```

17 Check the code! Update the script as needed.

```
51  # --------------------
52  # TODO 1 & 5
53  # Player projectiles
54  # --------------------
55  var projectile: Area2D = projectile_scene.instantiate()
56  projectile.direction = Vector2.UP
57  projectile.speed = projectile_speed
58
```

18 Playtest the project. Try firing projectiles with Left Click or Spacebar.

Uh oh! The projectiles aren't appearing! This is because they haven't been added to the scene tree yet, and instead, they are merely being added to the computer's **memory**.



19 When adding projectiles to the scene tree, they shouldn't be added as children to the player because then they would move *with* the player. Instead, they can be set as children to the main root of the project (or a container node).

SceneTree (which is returned when calling `get_tree()`) has a `current_scene` property, which is the root node of the currently active scene. In this case, it would be the **Main** node.

Use the `get_tree()` method to access the `current_scene` property, then call `add_child(projectile)` to add the projectile as a child to the main root. Then, set the projectile's `global_position` property to the `global_position` of the player.

```
51  # --------------------
52  # TODO 1 & 5
53  # Player projectiles
54  # --------------------
55  var projectile: Area2D = projectile_scene.instantiate()
56  projectile.direction = Vector2.UP
57  projectile.speed = projectile_speed
58  # get_tree() ???
59  # projectile.global_position ???
60
```

Reminder:

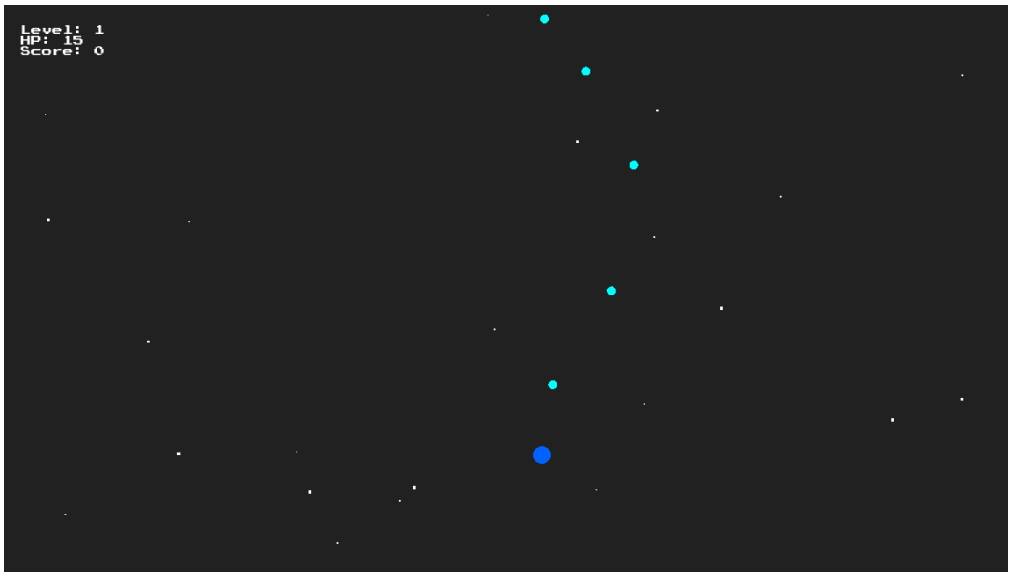
When accessing a property of the node that the script is attached to, write only the name of the property.

20 Check the code! Update the script as needed.

```
51  # --------------------
52  # TODO 1 & 5
53  # Player projectiles
54  # --------------------
55  var projectile: Area2D = projectile_scene.instantiate()
56  projectile.direction = Vector2.UP
57  projectile.speed = projectile_speed
58  get_tree().current_scene.add_child(projectile)
59  projectile.global_position = global_position
60
```

21 Playtest the project. Try firing projectiles with Left Click or Spacebar.

If projectiles are not firing properly, refer back to the previous steps.



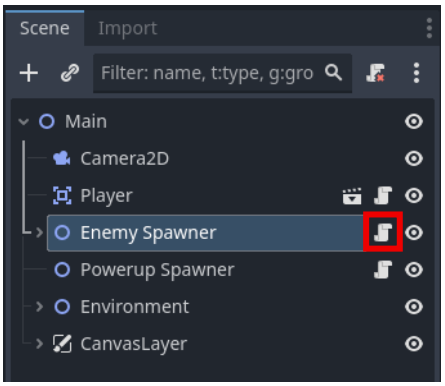
Pause for **Sensei Stop #1!**

Check in with a Code Sensei before moving on. Ensure the player direction and firing controls work properly.

Reminder: Save your work!

22 Currently, there is nothing for the player to shoot at. Time to spawn enemies!

Navigate to the Enemy Spawner's attached script, **enemy_spawner.gd.**



The existing code in **enemy_spawner.gd** prepares some useful variables and sets up a Timer to automatically spawn new enemies in an interval defined by the `spawn_cooldown` export variable.

```
1   extends Node2D
2
3   @export var enemy_scene: PackedScene
4   @export var spawn_cooldown: float
5
6   @onready var player: Node2D = get_tree().current_scene.get_node("Player")
7   @onready var waypoints: Array[Node2D] = [
8       $"Waypoints/Waypoint 1",
9       $"Waypoints/Waypoint 2",
10      $"Waypoints/Waypoint 3",
11      $"Waypoints/Waypoint 4",
12      $"Waypoints/Waypoint 5",
13  ]
14  @onready var spawn_timer: Timer = Timer.new()
15
16  func _ready():
17      add_child(spawn_timer)
18      spawn_timer.wait_time = spawn_cooldown
19      spawn_timer.timeout.connect(_spawn_enemy)
20      spawn_timer.start()
21
22  func _spawn_enemy():
23      if player.game_over:
24          return
25
```

23 Navigate to **TODO 2** in the `_spawn_enemy()` function which is called during every timeout from `spawn_timer.`

```
22  func _spawn_enemy():
23      if player.game_over:
24          return
25
26      # --------------------
27      # TODO 2
28      # Enemy spawning
29      # --------------------
30
31
```

Pro Tip:

User-defined functions that are not meant for other scripts to call will typically start with an underscore and are referred to as “private” functions.

24 The export variable **enemy_scene** is already linked to **enemy.tscn**, so create a new `enemy` variable of type `Area2D` and set it equal to `enemy_scene.instantiate()`.

The **enemy.gd** script uses a variable called **waypoints** to determine which points to randomly float across during gameplay. On the next line, set `enemy.waypoints` equal to this script's `waypoints` @onready variable so the enemy knows which points to randomly float across.

Then, add enemy as a child to this node and set its **global_position** to this node's **global_position**.

```
26  # ---------------------
27  # TODO 2
28  # Enemy spawning
29  # ---------------------
30  # enemy ???
31  # enemy.waypoints ???
32  # add_child() ???
33  # enemy.global_position ???
34
```

Pro Tip:

Instead of adding enemies as children to the main root of the scene by using `get_tree().current_scene`, it makes more sense to add them as children to the **Enemy Spawner** node so it acts as a container.

25 Check the code! Update the script as needed.

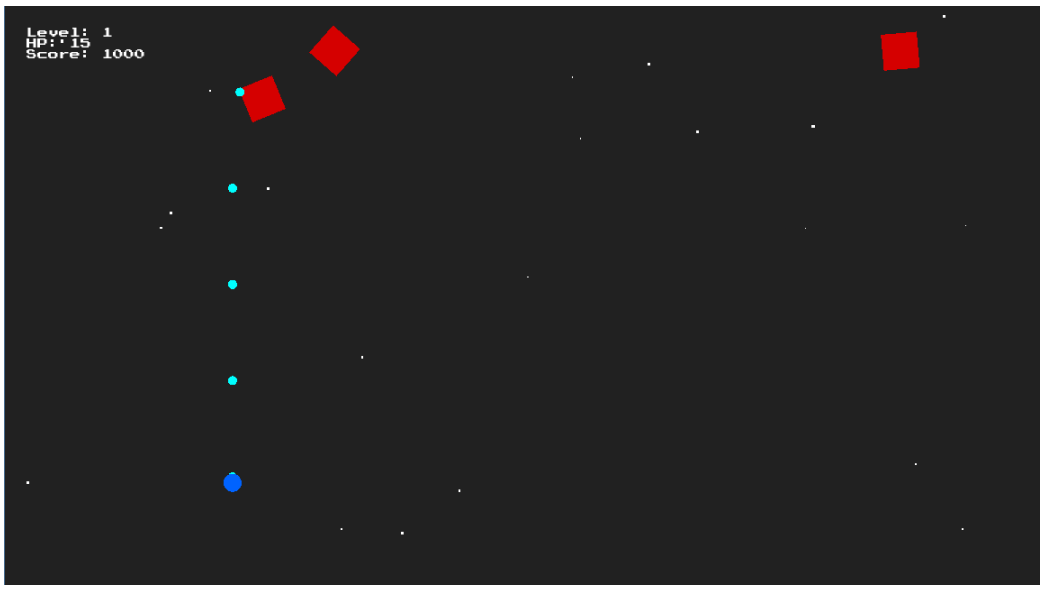
```
26  # --------------------
27  # TODO 2
28  # Enemy spawning
29  # --------------------
30  var enemy: Area2D = enemy_scene.instantiate()
31  enemy.waypoints = waypoints
32  add_child(enemy)
33  enemy.global_position = global_position
34
```

Pro Tip:

You may have noticed so far that instantiating objects follows a certain pattern. The pattern is: 1. Instantiate to memory, 2. Handle object-specific tasks, 3. Add to the scene tree, then 4. Handle scene-specific tasks.

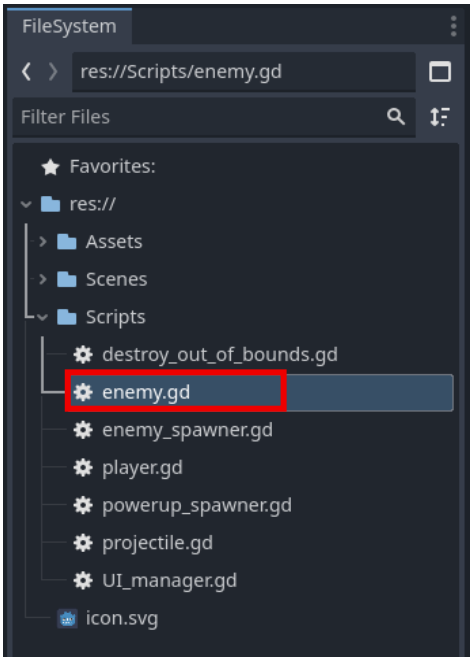
26 Playtest the project. Do enemies spawn in as expected?

There's a problem though. The enemies don't fight back against the player!



27 Navigate to **enemy.gd** in **FileSystem**. This script is already attached to the root node in **enemy.tscn**.

The existing code in **enemy.gd** handles collisions with projectiles, movement between the waypoints, and enemy death.



Navigate to **TODO 3** towards the bottom of the script. The soon-to-be added code will allow the enemies to fire hazardous projectiles at the player at the frequency described by the `attack_cooldown` export variable.

```
57    # move the enemy towards the target
58    translate(direction.normalized() * speed * delta)
59    rotate(rot_speed * delta)
60
61    attack_timer -= delta
62    if attack_timer <= 0:
63        attack_timer = attack_cooldown
64
65        # --------------------
66        # TODO 3
67        # Enemy hazard projectiles
68        # --------------------
69
70
```

28 Under **TODO 3**, write code to do the following:

1. Instantiate `hazard_scene` to a `hazard` variable of type `Area2D`.
2. Set the hazard's direction to the normalized version of the player's `global_position` minus this node's `global_position.`
3. Set the hazard's speed to the `hazard_speed` export variable.
4. Add hazard as a child to the main root of the scene.
5. Set the hazard's `global_position` to this node's `global_position`.

This code is very similar to what was written in **player.gd**; refer back to that script for help.

```
65      # --------------------
66      # TODO 3
67      # Enemy hazard projectiles
68      # --------------------
69      # hazard ???
70      # hazard.direction ???
71      # hazard.speed ???
72      # get_tree() ???
73      # hazard.global_position ???
74
```

29

Check the code! Update the script as needed.

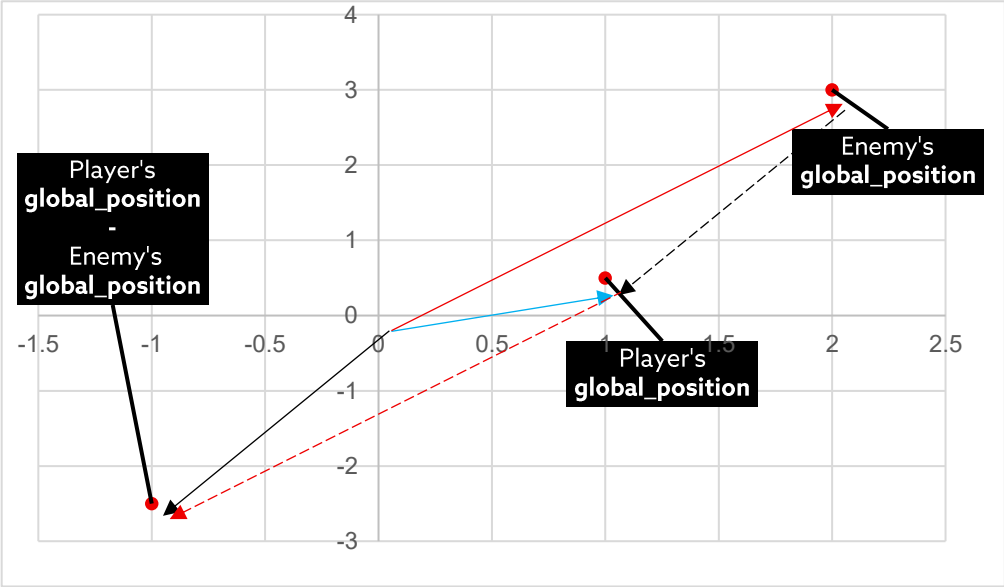
```
65  # --------------------
66  # TODO 3
67  # Enemy hazard projectiles
68  # --------------------
69  var hazard: Area2D = hazard_scene.instantiate()
70  hazard.direction = (player.global_position - global_position).normalized()
71  hazard.speed = hazard_speed
72  get_tree().current_scene.add_child(hazard)
73  hazard.global_position = global_position
74
```

30

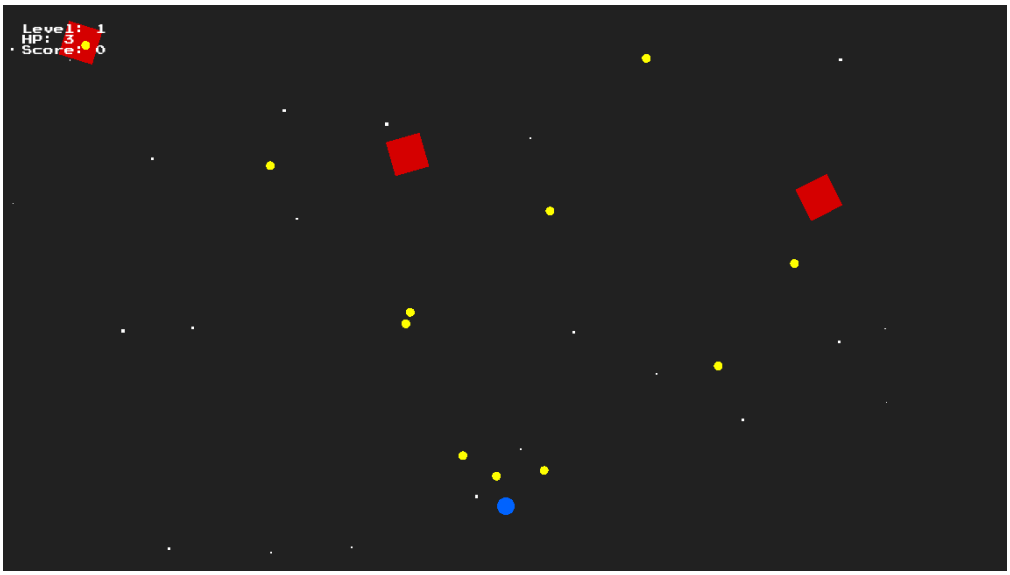
The hazard's direction may be a little confusing, but it ensures that the hazard targets the player.

This diagram shows the Player's, Enemy's, and the Player minus the Enemy's **global_positions** as vectors in solid lines. The dashed red line shows what happens when the Enemy's **global_position** is subtracted from the Player's. The dashed black line shows how the subtraction accurately describes the direction towards the Player's **global_position** from the Enemy.

In the code, the Vector is then normalized to ensure that all projectiles have the same speed.



31 Playtest the project. Notice how the enemies accurately fire hazards at the player now!



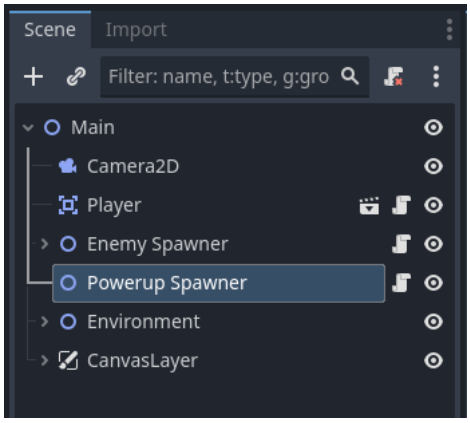
Pause for **Sensei Stop #2!**

Check in with a Code Sensei before moving on. Make sure that the enemies are spawning in and firing at the player as intended.

Reminder: Save your work!

32 The next thing to add is... Powerups! These will level-up the Player and make them stronger.

In **main.tscn**, navigate to the Powerup Spawner's attached script, **powerup_spawner.gd.**



The existing code in **powerup_spawner.gd** prepares some useful variables and sets up a Timer to automatically spawn new powerups in an interval defined by the `spawn_cooldown` export variable.

Navigate to **TODO 4** in the `_spawn_powerup()` function which is called during every timeout from `spawn_timer.`

```gdscript
extends Node2D

@export var powerup_scene: PackedScene
@export var powerup_speed: float
@export var spawn_cooldown: float

@onready var player: Node2D = get_tree().current_scene.get_node("Player")
@onready var spawn_timer: Timer = Timer.new()

func _ready():
    add_child(spawn_timer)
    spawn_timer.wait_time = spawn_cooldown
    spawn_timer.timeout.connect(_spawn_powerup)
    spawn_timer.start()

func _spawn_powerup():
    if player.game_over:
        return

    # --------------------
    # TODO 4
    # Power-up spawning
    # --------------------

```

33 To randomize the spawning of powerups across the top of the screen, the width of the screen is needed. CanvasItem has a convenient method for this, `get_viewport_rect()`.

> `get_viewport_rect()`: part of the CanvasItem class, returns the boundaries of the viewport as a Rect2.
>
> **Parameters:** none
>
> **Returns *(Rect2)*:** the boundaries of the viewport. Includes Vector2 properties **position, end,** and **size.**

New Concept: Rect2

A class that stores bounds of a 2D rectangle with the Vector2 properties **position** (top-left coordinate), **end** (bottom-right coordinate), and **size** (width, height).

34 Declare a `random_x` variable of type float and set it equal to the result of `randf_range()` with the first parameter `0` and the second parameter `get_viewport_rect().size.x`.

```
20    # --------------------
21    # TODO 4
22    # Power-up spawning
23    # --------------------
24    # random_x ???
25
```

Pro Tip:

Since **size** is a Vector2, it has a property **x** which describes its width.

35

Check the code! Update the script as needed.

```
20    # --------------------
21    # TODO 4
22    # Power-up spawning
23    # --------------------
24    var random_x: float = randf_range(0, get_viewport_rect().size.x)
25
```

36

Under the `random_x` variable declaration, write code to do the following:

1. Instantiate `powerup_scene` to a `powerup` variable of type `Area2D`.
2. Set the powerup's direction to `Vector2.DOWN`
3. Set the powerup's speed to the `powerup_speed` export variable.
4. Add the powerup as a child to this node.

Set the powerup's `global_position` to a new Vector2. Pass `random_x` as the x value and -100 as the y value (so powerups spawn slightly above the top of the screen).

```
20    # --------------------
21    # TODO 4
22    # Power-up spawning
23    # --------------------
24    var random_x: float = randf_range(0, get_viewport_rect().size.x)
25
26    # powerup ???
27    # powerup.direction ???
28    # powerup.speed ???
29    # add_child() ???
30    # powerup.global_position ???
31
```

37 Check the code! Update the script as needed.

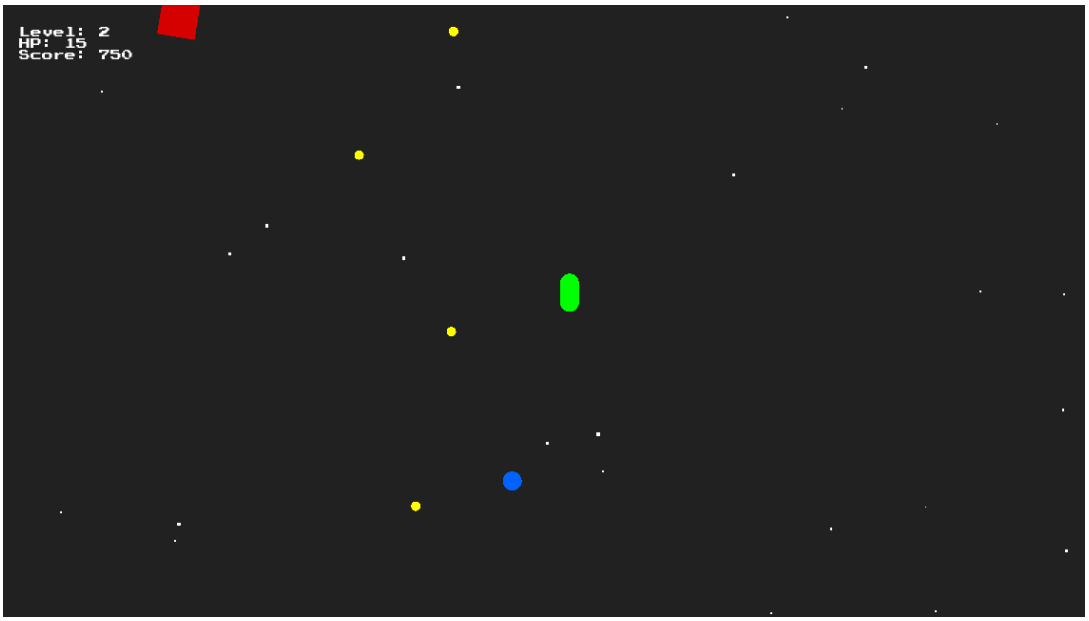
```
20  # --------------------
21  # TODO 4
22  # Power-up spawning
23  # --------------------
24  var random_x: float = randf_range(0, get_viewport_rect().size.x)
25
26  var powerup: Area2D = powerup_scene.instantiate()
27  powerup.direction = Vector2.DOWN
28  powerup.speed = powerup_speed
29  add_child(powerup)
30  powerup.global_position = Vector2(random_x, -100)
31
```

Reminder:

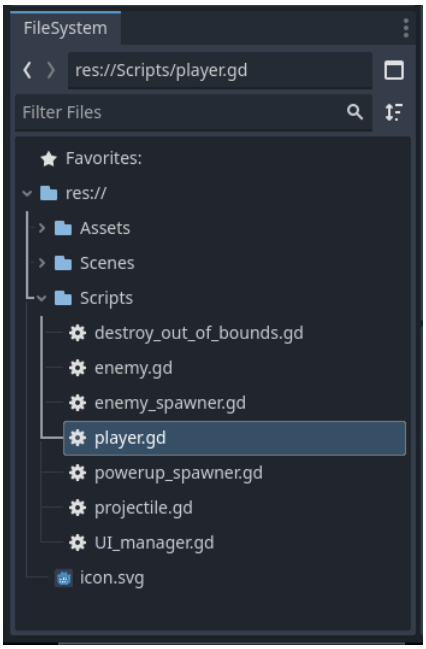
Just like when spawning enemies, instead of adding powerups as children to the main root of the scene by using **`get_tree().current_scene`**, it makes more sense to add them as children to the **Powerup Spawner** node so it acts as a container.

38 Playtest the project. Powerups should now rain from the sky and collecting them will increase the Level in the top-left corner. The higher the level, the greater the score that enemies give when defeated.

However, the player isn't powering up yet! New logic needs to be added in **player.gd** to create more projectiles when the player levels up.



39 Navigate back to **player.gd** in **FileSystem** and scroll down to **TODO 5**.

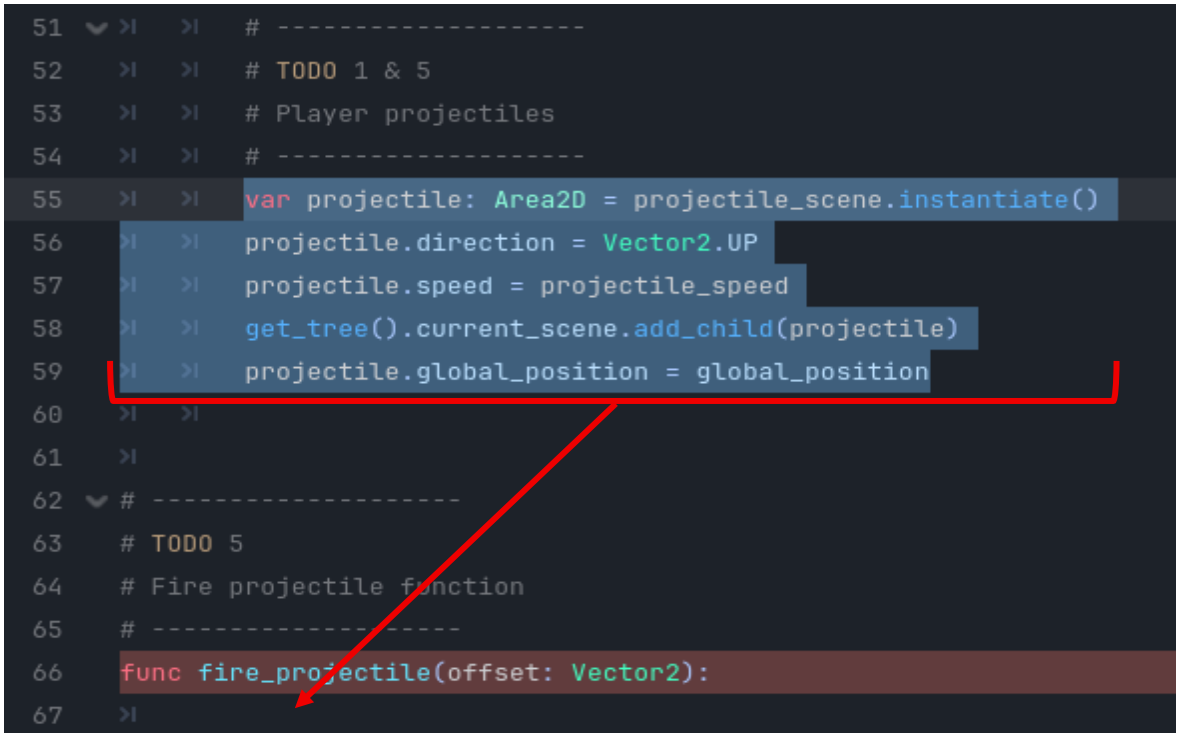


To make it easier to spawn projectiles, define a `fire_projectile()` function which takes an `offset` parameter of type `Vector2`.

```
47      fire_timer -= delta
48      if Input.is_action_pressed("shoot") and fire_timer <= 0:
49          fire_timer = fire_cooldown
50
51          # --------------------
52          # TODO 1 & 5
53          # Player projectiles
54          # --------------------
55          var projectile: Area2D = projectile_scene.instantiate()
56          projectile.direction = Vector2.UP
57          projectile.speed = projectile_speed
58          get_tree().current_scene.add_child(projectile)
59          projectile.global_position = global_position
60
61
62  # --------------------
63  # TODO 5
64  # Fire projectile function
65  # --------------------
66
```

40 Cut **(Ctrl + X)** the code from **TODO 1** and paste it **(Ctrl + V)** into the new function.

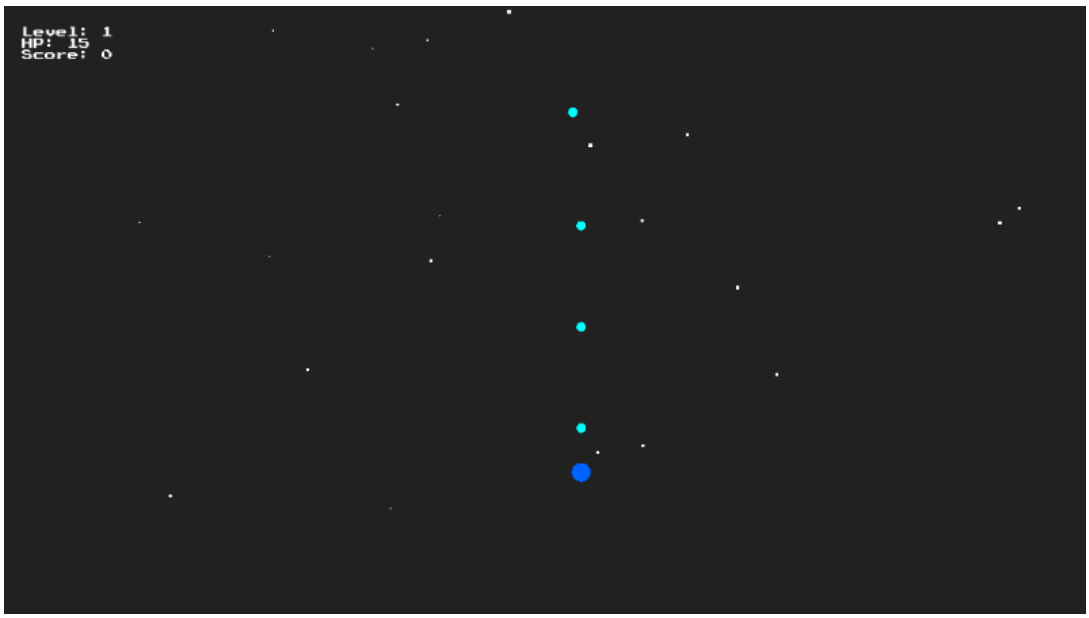
In the final line inside the function, add the **`offset`** parameter to **`global_position`**.



41 Check the code! Update the script as needed.

```
51      # --------------------
52      # TODO 1 & 5
53      # Player projectiles
54      # --------------------
55
56
57
58  # --------------------
59  # TODO 5
60  # Fire projectile function
61  # --------------------
62  func fire_projectile(offset: Vector2):
63      var projectile: Area2D = projectile_scene.instantiate()
64      projectile.direction = Vector2.UP
65      projectile.speed = projectile_speed
66      get_tree().current_scene.add_child(projectile)
67      projectile.global_position = global_position + offset
68
```

42 To test if the function works, call `fire_projectile()` with the offset of `Vector2.ZERO` under **TODO 1** and playtest the project. Is the player able to fire projectiles normally?




```
51    # --------------------
52    # TODO 1 & 5
53    # Player projectiles
54    # --------------------
55    fire_projectile(Vector2.ZERO)
56
```

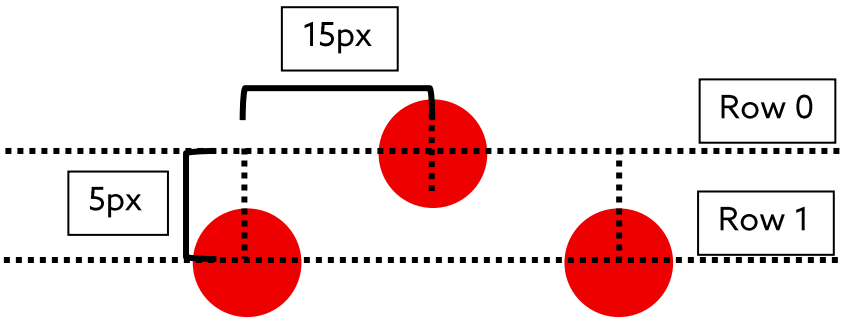
43

Great! Code the player to shoot more projectiles as they level up.

What if the extra projectiles could form an upside-down V shape as more get added? It's good practice to plan how to implement things before diving in, so take a moment and try to understand the thought process and diagram below.

In **Row 0**, there is no offset. In **Row 1**, there is a 5-pixel vertical offset and a 15-pixel horizontal offset in both directions.

In **Row 2**, there would be a 10-pixel vertical offset and 30-pixel horizontal offset in both directions, and so on. The **Row** must be kept track of in the code to scale the offsets correctly.



44

Under **TODO 1**, delete the line of code that calls `fire_projectile()`.

Create two variables called `horizontal_offset` and `vertical_offset` of type float, with values `15.0` and `5.0` respectively.

```
51  # --------------------
52  # TODO 1 & 5
53  # Player projectiles
54  # --------------------
55  var horizontal_offset: float = 15.0
56  var vertical_offset: float = 5.0
57
```

45 To create as many projectiles as the current level of the player, **`a for i in range()` loop** is optimal. This is because **`range()`** takes an integer parameter and returns an array of integers from zero to the parameter minus one.

```
    for i in range(5):
        pass
```

```
for (let value of list) {

}
```

On the next line, write **`for i in range(level):.`** The code inside the loop will run for each level of the player, starting at **`0`** and ending at **`level-1`**. In other words, it will run **`level`** times.

> **`range()`:** generates an array of integers starting at 0 and incrementing by 1 up to the given end.
>
> Parameters:
>
> 1. **end *(int)*:** the maximum value, excluded from the array
>
> **Returns *(Array)*:** An array of integers from 0 to end, excluding end

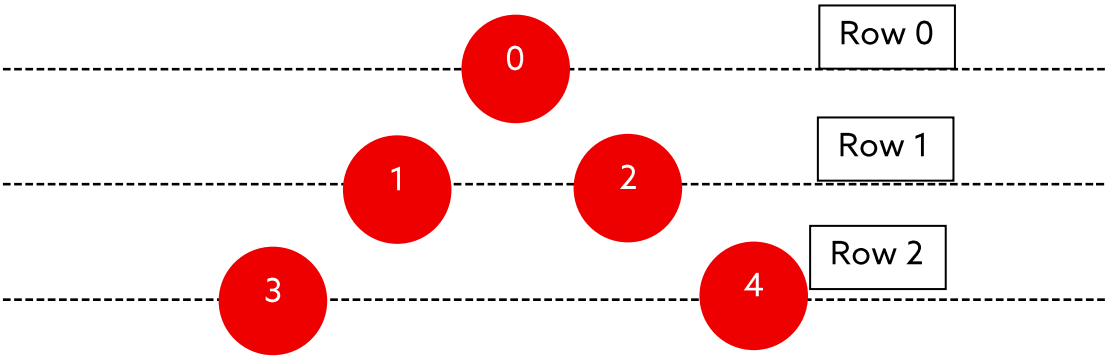
```
51        # --------------------
52        # TODO 1 & 5
53        # Player projectiles
54        # --------------------
55        var horizontal_offset: float = 15.0
56        var vertical_offset: float = 5.0
57        for i in range(level):
58
```

46 On the next line inside the loop, declare a `row` variable and set its value to the integer cutoff of `(i + 1) / 2.`

Use the values and diagram below to explore how it works:

i = 0: row = (0 + 1) / 2 = 0
i = 1: row = (1 + 1) / 2 = 1
i = 2: row = (2 + 1) / 2 = 1.5 -> 1
i = 3: row = (3 + 1) / 2 = 2
i = 4: row = (4 + 1) / 2 = 2.5 -> 2
And so on...

```
51  # --------------------
52  # TODO 1 & 5
53  # Player projectiles
54  # --------------------
55  var horizontal_offset: float = 15.0
56  var vertical_offset: float = 5.0
57  for i in range(level):
58      var row = int((i + 1) / 2)
59
```



47 Remember how each row will push the projectiles further down and away? Now, the `horizontal_offset` and `vertical_offset` need to be scaled according to the row.

Write two lines to declare `x_offset` and `y_offset` variables which are set equal to `row * horizontal_offset` and `row * vertical_offset` respectively.

```
51      # --------------------
52      # TODO 1 & 5
53      # Player projectiles
54      # --------------------
55      var horizontal_offset: float = 15.0
56      var vertical_offset: float = 5.0
57      for i in range(level):
58          var row = int((i + 1) / 2)
59          var x_offset = row * horizontal_offset
60          var y_offset = row * vertical_offset
61
```

48 All the *even* indices place projectiles on the right, which means they should have a **positive x_offset**. Meanwhile, the *odd* indices place projectiles on the left, so they will have a **negative x_offset**.

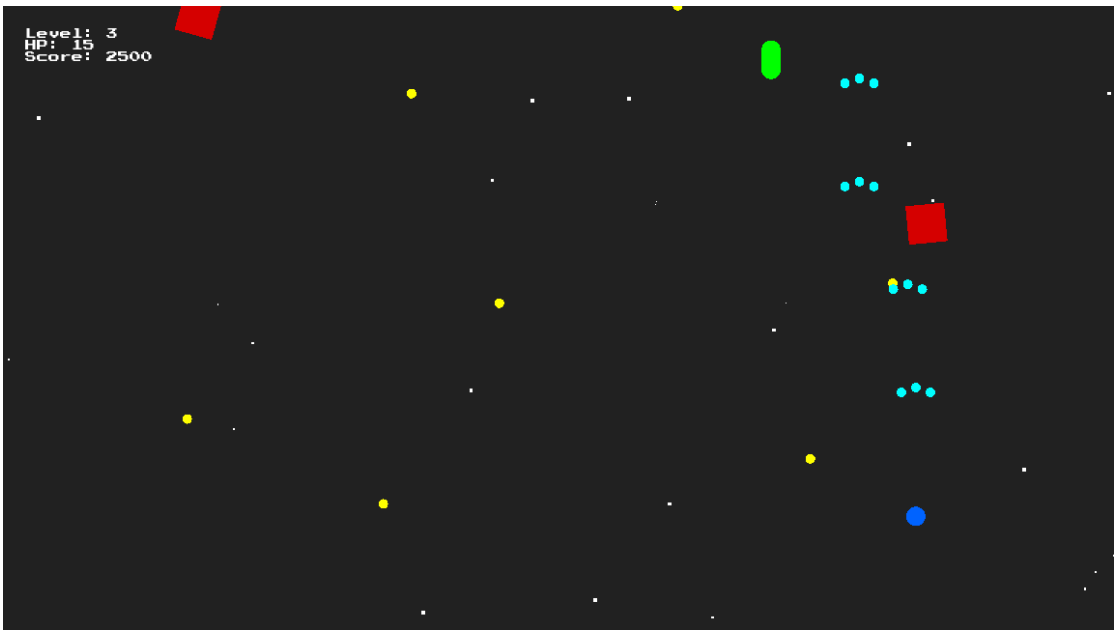
On the next few lines, use the `modulo operator %` to determine if the current index is even. Then, call `fire_projectile()` with `x_offset` and `y_offset` as values in a new Vector2, setting **x_offset** to be negative when the index is odd.

```
51  # --------------------
52  # TODO 1 & 5
53  # Player projectiles
54  # --------------------
55  var horizontal_offset: float = 15.0
56  var vertical_offset: float = 5.0
57  for i in range(level):
58      var row = int((i + 1) / 2)
59      var x_offset = row * horizontal_offset
60      var y_offset = row * vertical_offset
61      if i % 2 == 0:
62          fire_projectile(Vector2(x_offset, y_offset))
63      else:
64          fire_projectile(Vector2(-x_offset, y_offset))
65
```

New Concept: Modulo %

This operator computes the remainder of integer division. For example: **7 % 4 = 3, 12 % 6 = 0, 8 % 2 = 0.** It is mostly used to check if a number is even by comparing the output of the number modulo 2 to 0 or 1. If it is 0, that means there is no remainder, so the number is even. Otherwise, it's odd.

49 Playtest the project. Level up, get stronger, and beat the bad guys!



Pause for Sensei Stop #3

Congratulations on creating your first fixed-shooter game in Godot! Great job!

Before submitting, check in with a Code Sensei to check that the powerups are spawning properly and the player fires more projectiles as they level up.

Then, reflect on the following:

- What did you learn about instantiating scenes?
- What did you enjoy most when creating this project?
- What was something you found difficult and why?