Platinum Belt Ninja Guide

PB Activity 00: Catch the Stars

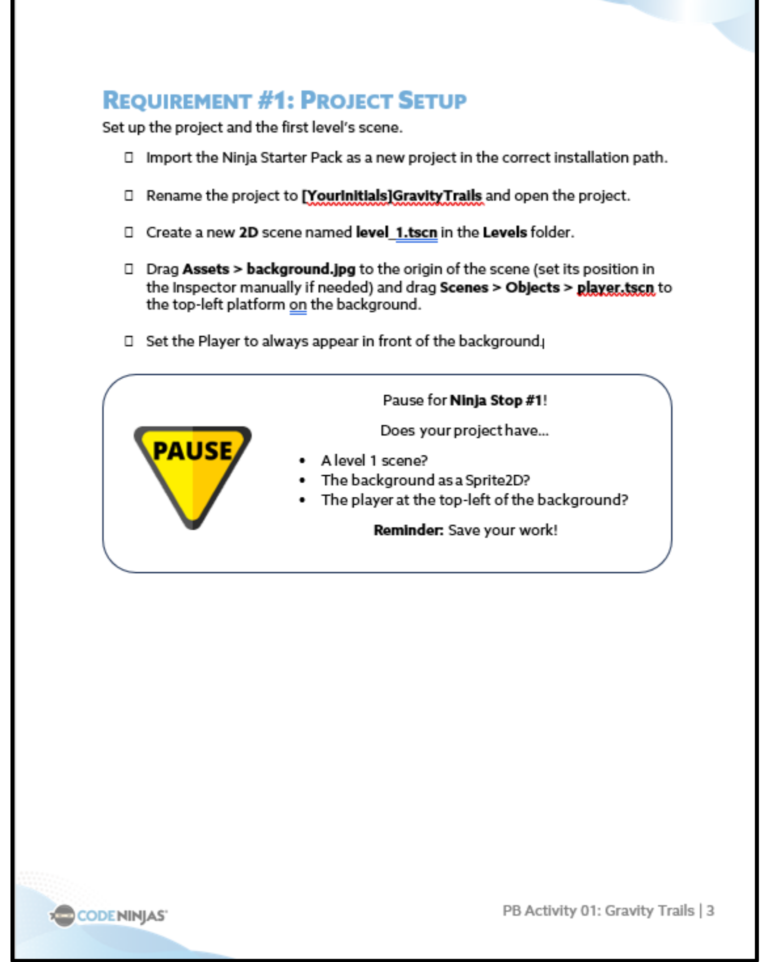
Contents

Requirements 2
Asset Creation 4
PB Activity 00: Catch the Stars 5
Requirement #1: Project Setup 6
Requirement #2 **(Instructions)**: Custom Assets 8
Requirement #3: Player Movement 19
Requirement #4 **(Instructions)**: Collectable Scene....................22
Requirement #5 **(Instructions)**: Spawner 32
Requirement #6: User Interface....................35
Requirement #7 **(Instructions)**: Countdown 38
Requirement #8: Game Over UI.................... 43

REQUIREMENTS

Now that you are comfortable creating projects in Godot, Platinum Belt will prepare you for Gold Belt by easing off the training wheels! Now, the Ninja Guides will cover a series of **Requirements**.

Some Requirements are instructional, and act similarly to the steps found in Bronze-Silver Belts. Others are more open-ended and will ask you to implement a feature with a checklist of tasks to guide you. At the end of these open-ended Requirements, there will be a **Ninja Stop** that prompts you to playtest the game to determine if the feature has been implemented properly.



REQUIREMENT #1: PROJECT SETUP

Set up the project and the first level's scene.

- Import the Ninja Starter Pack as a new project in the correct installation path.
- Rename the project to **[YourInitials]GravityTrails** and open the project.
- Create a new **2D** scene named **level_1.tscn** in the **Levels** folder.
- Drag **Assets > background.jpg** to the origin of the scene (set its position in the Inspector manually if needed) and drag **Scenes > Objects > player.tscn** to the top-left platform on the background.
- Set the Player to always appear in front of the background.

PAUSE

Pause for **Ninja Stop #1**!

Does your project have...

- A level 1 scene?
- The background as a Sprite2D?
- The player at the top-left of the background?

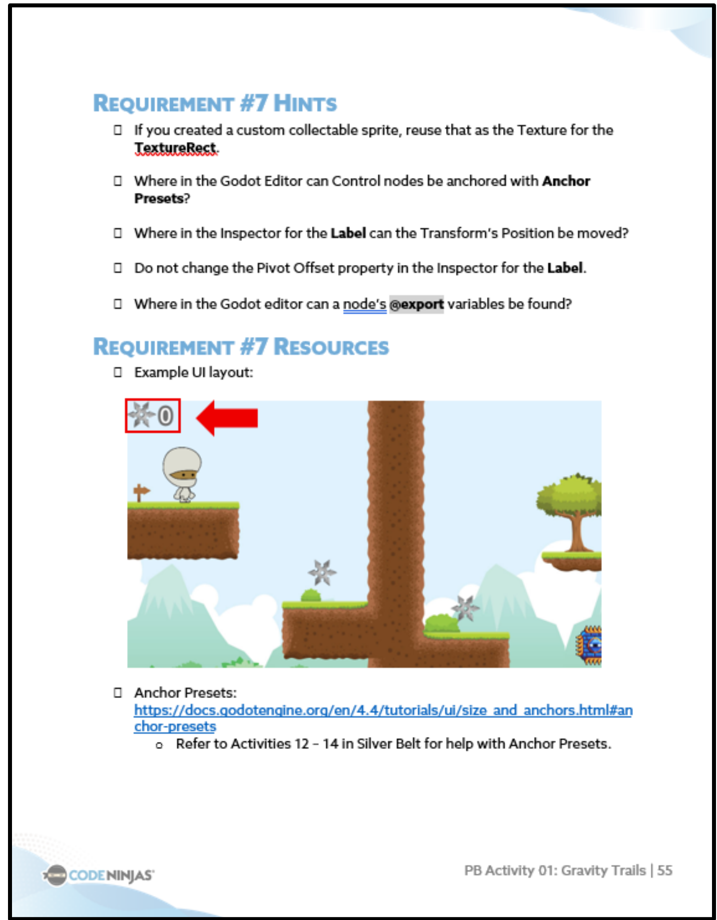
Reminder: Save your work!

CODE NINJAS

PB Activity 01: Gravity Trails | 3

EXAMPLE OF AN OPEN-ENDED REQUIREMENT

Below each open-ended Requirement's Ninja Stop, there will be two additional sections: **Hints** and **Resources**. **Hints** will provide additional details to assist with the task checklist. **Resources** will provide example images, previous Godot activities to refer to, and documentation links to follow for additional information.



Requirement #7 Hints

- If you created a custom collectable sprite, reuse that as the Texture for the **TextureRect**.
- Where in the Godot Editor can Control nodes be anchored with **Anchor Presets**?
- Where in the Inspector for the **Label** can the Transform's Position be moved?
- Do not change the Pivot Offset property in the Inspector for the **Label**.
- Where in the Godot editor can a node's **@export** variables be found?

Requirement #7 Resources

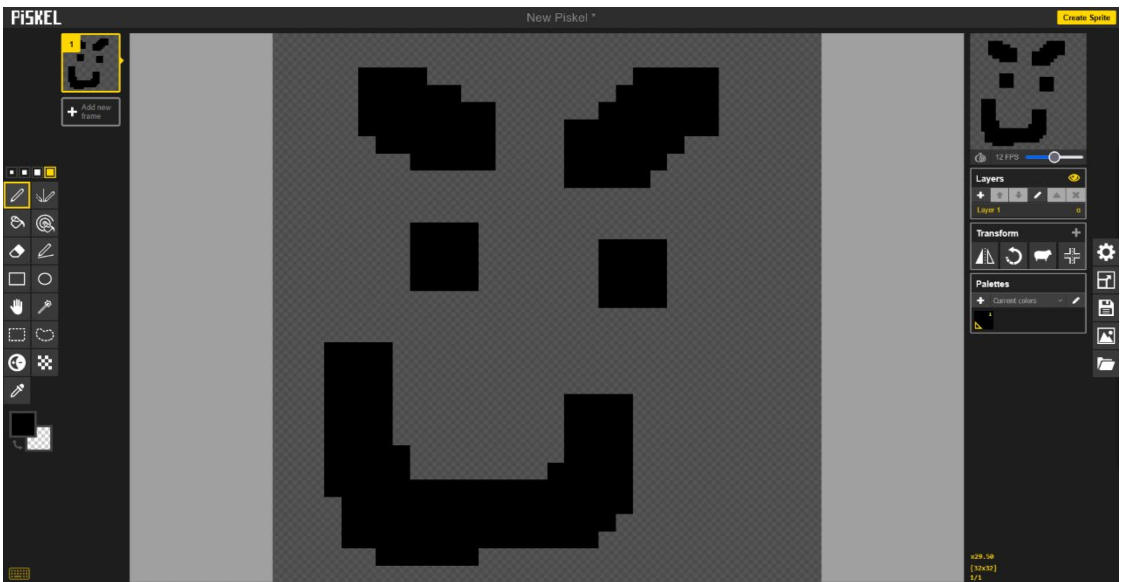
- Example UI layout:
- Anchor Presets: https://docs.godotengine.org/en/4.4/tutorials/ui/size_and_anchors.html#anchor-presets
  - Refer to Activities 12 – 14 in Silver Belt for help with Anchor Presets.

PB Activity 01: Gravity Trails | 55

EXAMPLE OF HINTS AND RESOURCES FOR AN OPEN-ENDED REQUIREMENT

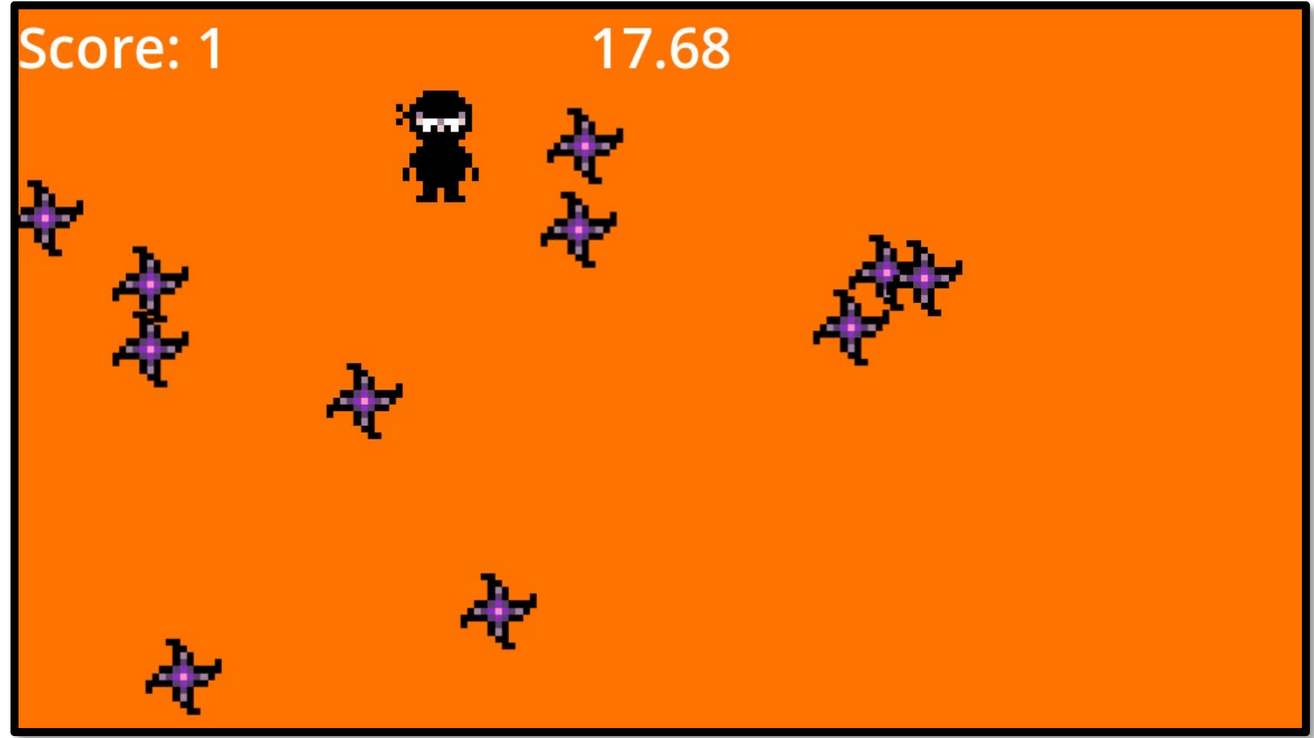
Asset Creation

So far in Godot, you have used art assets provided to you in the Starter Pack. Now, in addition to pre-provided assets, you will also be creating your own assets using a program called **Piskel**! Piskel has a web version and a downloadable version, so ask your Code Sensei which one will be used.



PB Activity 00: Catch the Stars

This project serves as a light introduction to Platinum Belt by introducing you to **Requirements**, **Ninja Stops**, and **Asset Creation**. You will build a Catch the Stars project where stars will randomly appear across the screen, and the Player must collect as many as possible before time runs out!



Requirement #1: Project Setup

Set up the project and the main scene.

- ☐ Import the Ninja Starter Pack as a new project in the correct installation path.
- ☐ Rename the project to **[YourInitials]CatchTheStars** and open the project.
- ☐ Create a new **2D** scene named **main.tscn** in the **Scenes** folder.
- ☐ Create a new **CharacterBody2D** as a child to Main, rename it to **Player**, and add a Sprite2D and a CollisionShape2D as children to Player. **Do not** set the **Texture** or the **Shape** yet.
- ☐ In **Project Settings > Environment**, set the **Default Clear Color** to any color of choice.

Pause for **Ninja Stop #1**!

Test your project! Does it have...

- A main scene?
- A Player that has no texture or collision?
- A customized background color?

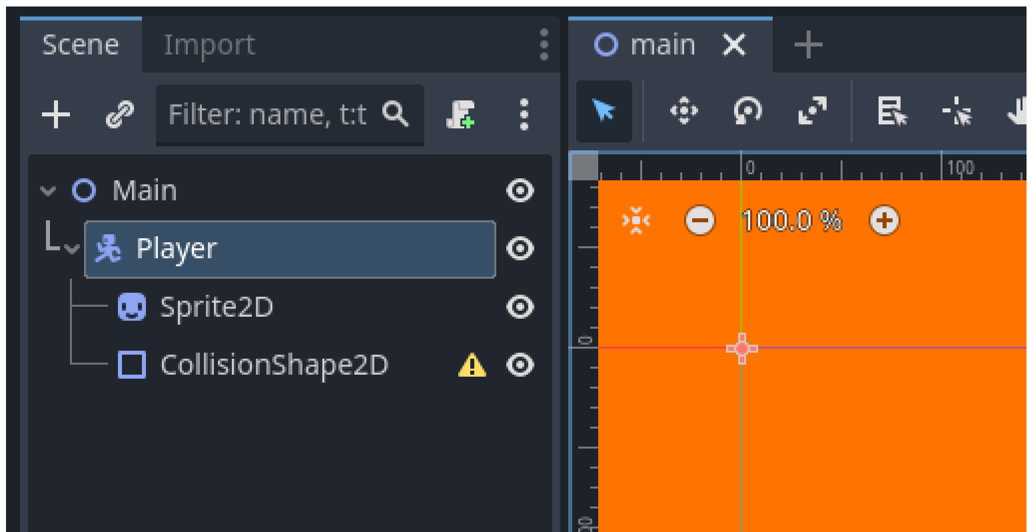
Reminder: Save your work!

Requirement #1 Hints

- Which button imports files as a new project in the Project Manager?
- Where is the starter code stored? What file type is the starter pack?
- The Default Clear Color can be set in **Project Settings > General > Rendering > Environment**.

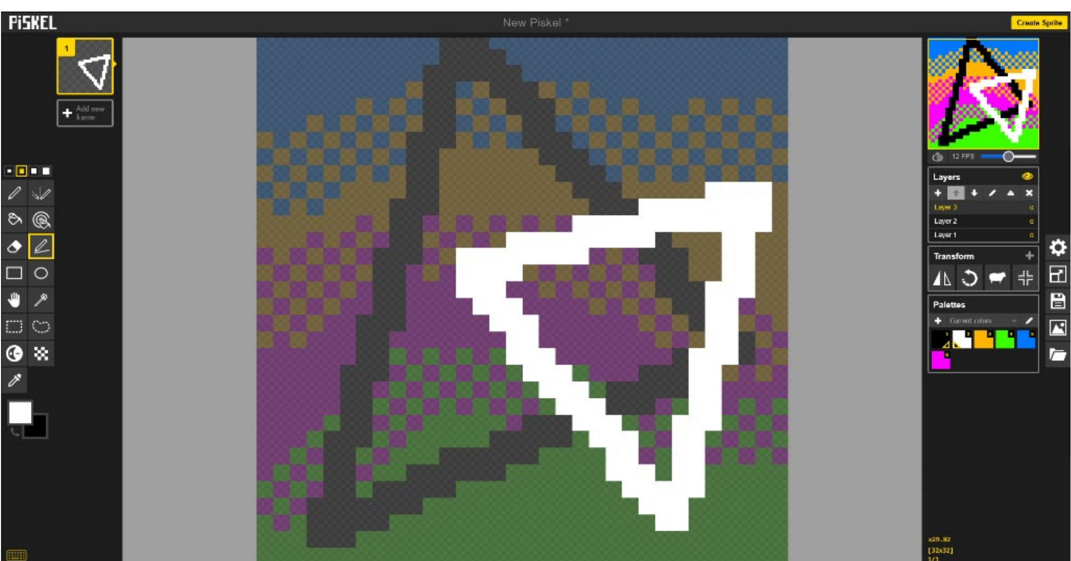
Requirement #1 Resources

- Refer back to PB Activities 06 – 17 in Silver Belt for help with importing a project.
- Example Scene hierarchy:

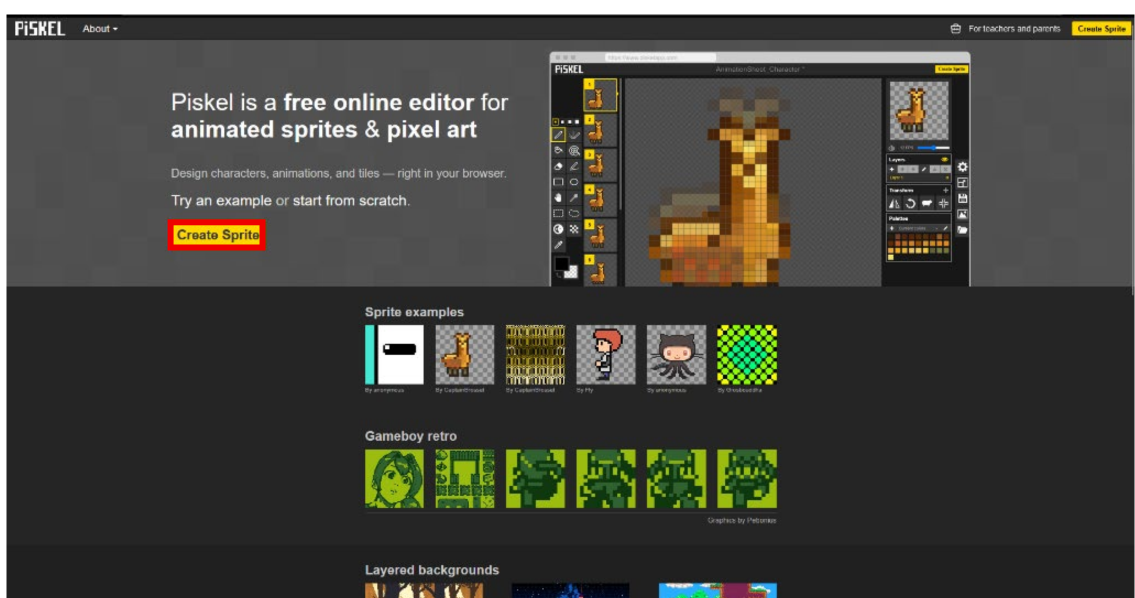


Requirement #2 (Instructions): Custom Assets

1 Piskel is a free art software that enables users to quickly draw simple pixel or high-definition art and animations. In Platinum Belt, you will be using it to create assets from scratch instead of only using starter assets.



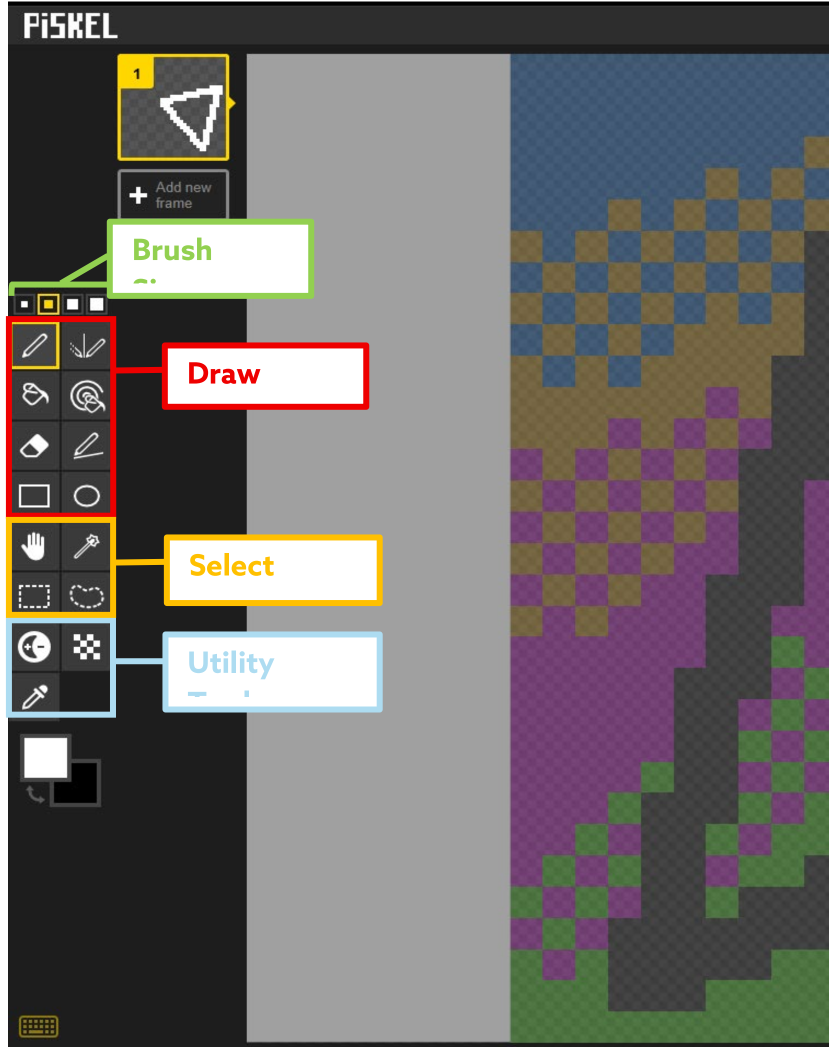
Open **Piskel** and click **Create Sprite** to follow along.



If you don't have access to Piskel, use the assets provided in the Starter Pack.

2 Many of the **tools** in Piskel are like those found in MakeCode's paint editor. Hover over each tool to find additional key binds that change the functionality of the tool.

For example, holding SHIFT when using the Vertical Mirror pen will make it mirror both horizontally and vertically. Try drawing a heart using the **Vertical Mirror** pen!

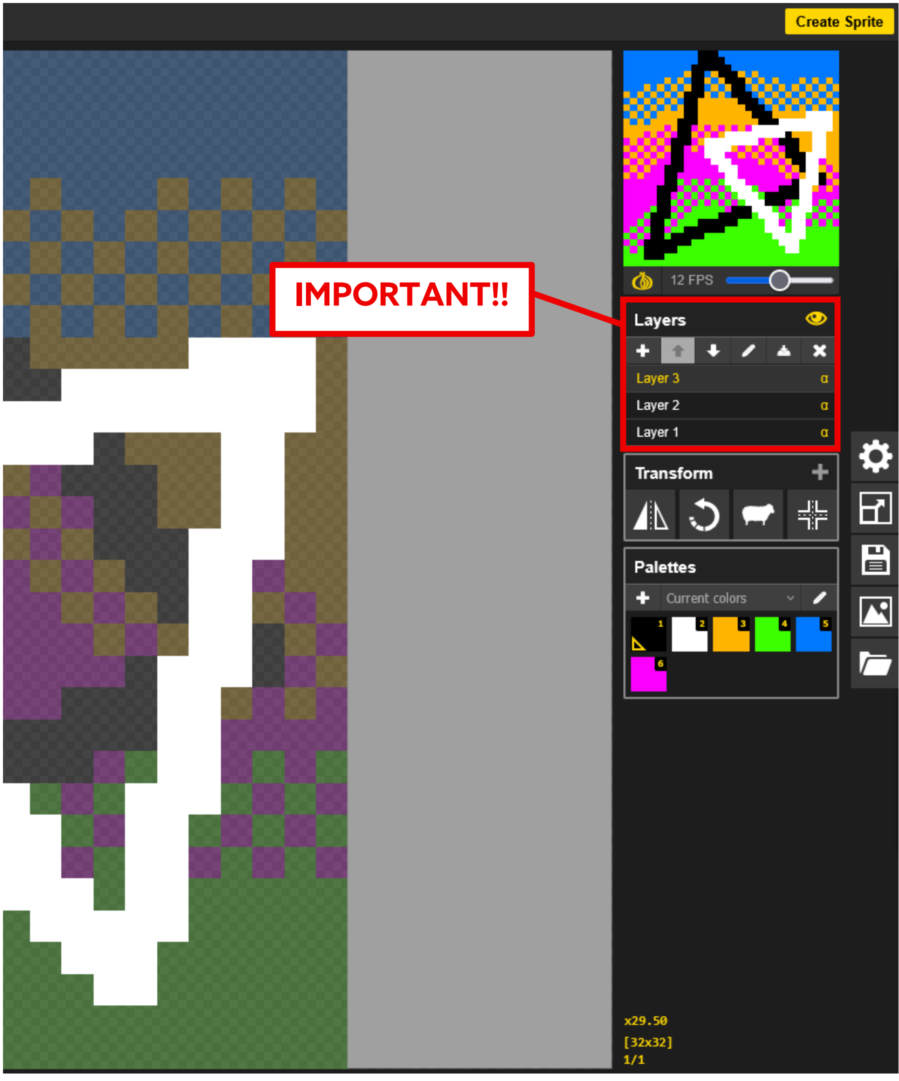


Pro Tip:

Click the Keyboard Shortcuts icon in the bottom left corner for a list of all shortcuts!

3 On the **right side** of the UI, notice the preview window and additional utilities.

The most important section is **Layers**. It allows artists to separate important parts of their drawing, so they can easily be added and removed without the artist needing to re-draw certain parts.

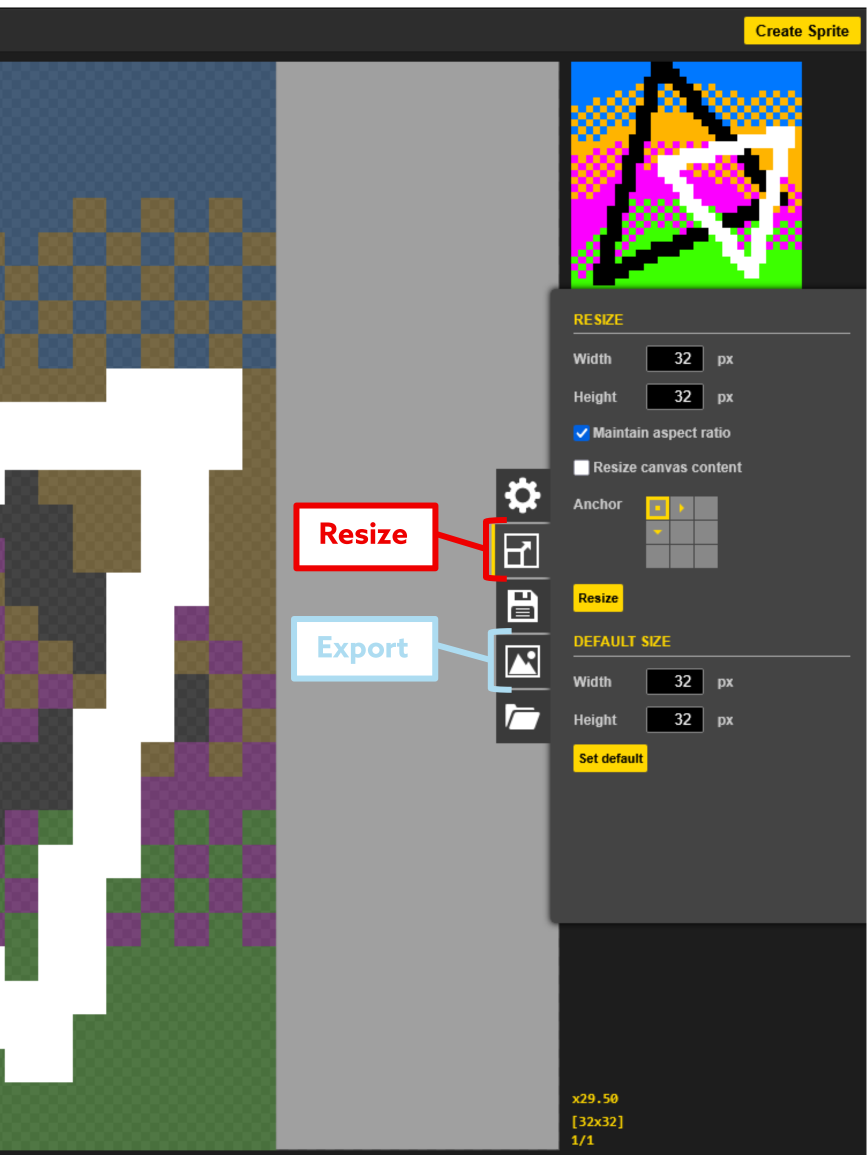


In this image, **Layer 3** contains a white triangle. Since it is at the top of the **Layers** stack, it is *drawn over* all the other layers. Can you guess which Layer the black triangle is in?

Try drawing a **rectangle** on a **new layer** so it is drawn *below* the heart!

4 To change the pixel resolution of the image, select **Resize** on the right panel.

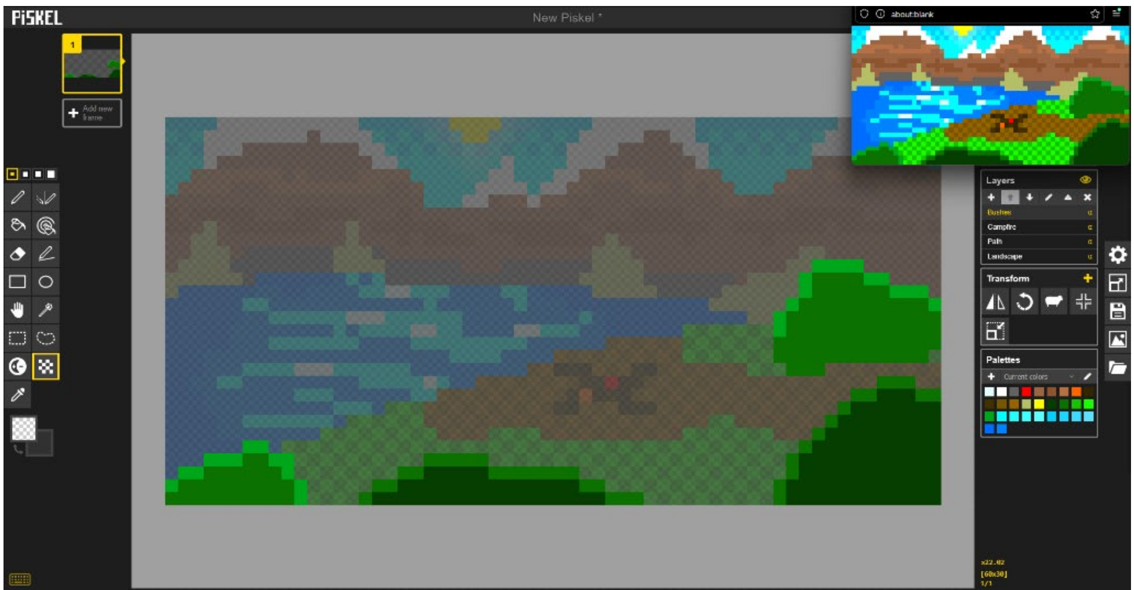
The art you make with Piskel doesn't have to be pixel art – crank up the Width and Height to as much as you want (within reason)! For assets that are not perfectly square, uncheck **Maintain aspect ratio**. Try **resizing the canvas** so it is wider than it is tall!



Pro Tip:

When navigating the Canvas, make sure to use these simple key binds:
Zoom – Scroll Wheel, +, -, 0
Move – Scroll Click & Drag, Shift + Direction Keys

5 Practice using **Piskel** to make an art piece! It can be a simple object, like a ball, or a landscape! If your art includes multiple components, draw them on separate layers so they can be adjusted more easily.



Think back to the pixel art strategies you explored in MakeCode. Use the **dithering tool** to transition between multiple colors, fill with **highlights/shadows** to make colors pop, and **remove jaggies** to create clean lines in your art.

6 Once you're done with the artwork open **Save** and update the text under **Title** to reflect the artwork.

Pro Tip:

Piskel artwork can be saved as a **.piskel** file which preserves project settings and Layers. Meanwhile, importing a **.png** file into Piskel will "flatten" it into just one Layer.

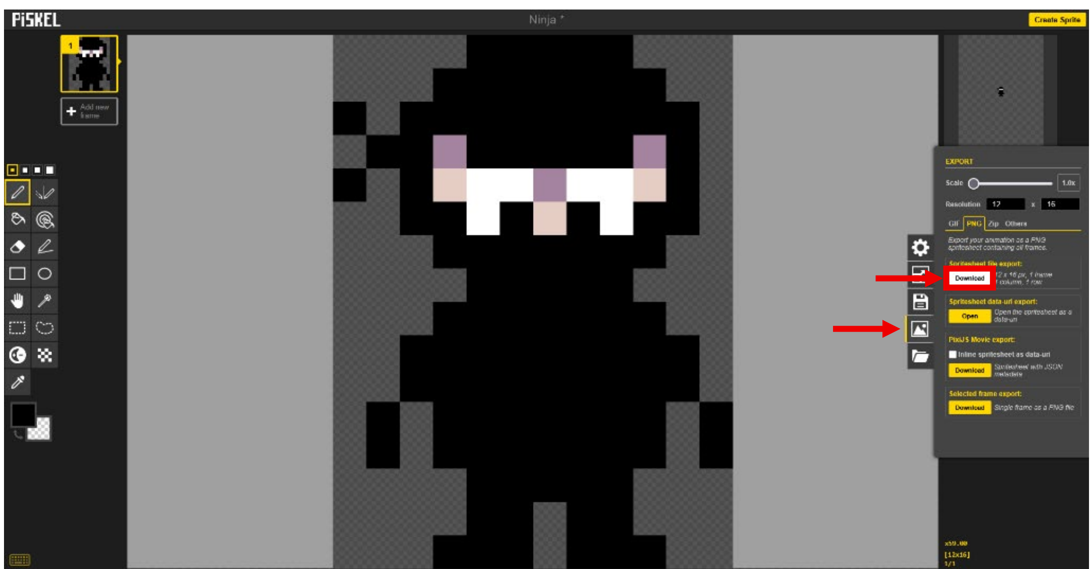
7 To export the project, open **Export**, keep the **Scale** and **Resolution** the same, ensure **PNG** is selected, then click **Download** under **Spritesheet file export**.

The download will appear in the computer's **Downloads** folder by default.

8 Click **Create Sprite** or **+** to make a new project.

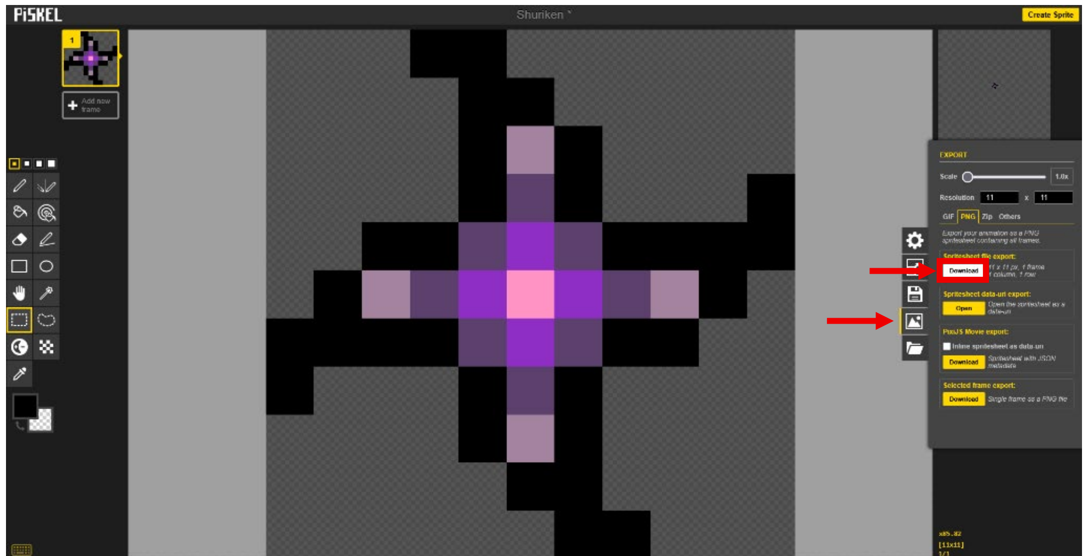
Draw the player sprite using **Piskel**! Refer to previous steps for help. Remember, this project will feature a player sprite collecting objects around the screen.

Once you're done with the sprite, go to **Export**, keep the **Scale** and **Resolution** the same, ensure **PNG** is selected, then click **Download**. Leave it in the Downloads folder for now – it will be imported into Godot after the Collectable sprite is made!

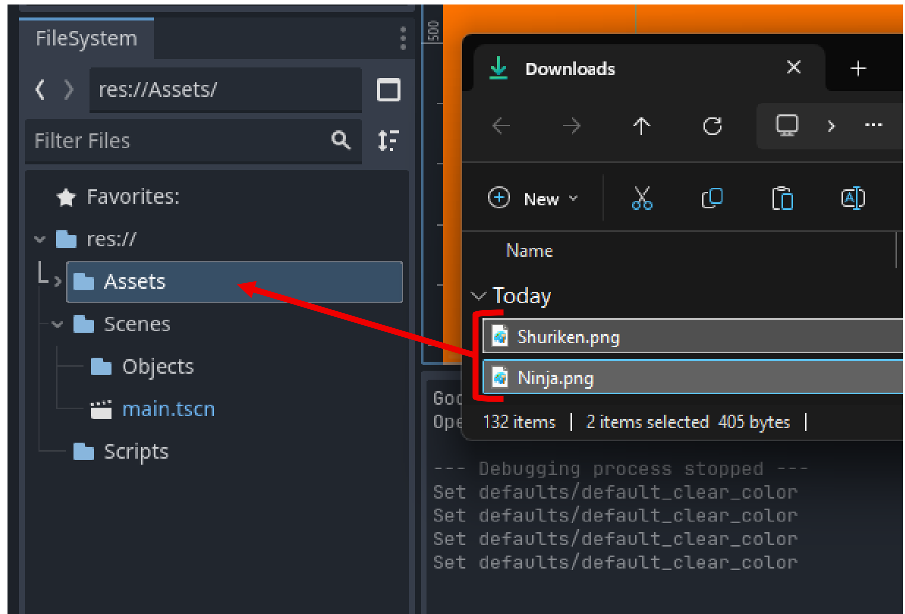


9 Draw the collectable sprite using **Piskel**! Remember, these are the sprites that will appear around the screen for the player sprite to collect.

Once you're done with the sprite, go to **Export**, keep the **Scale** and **Resolution** the same, ensure **PNG** is selected, then click **Download**.

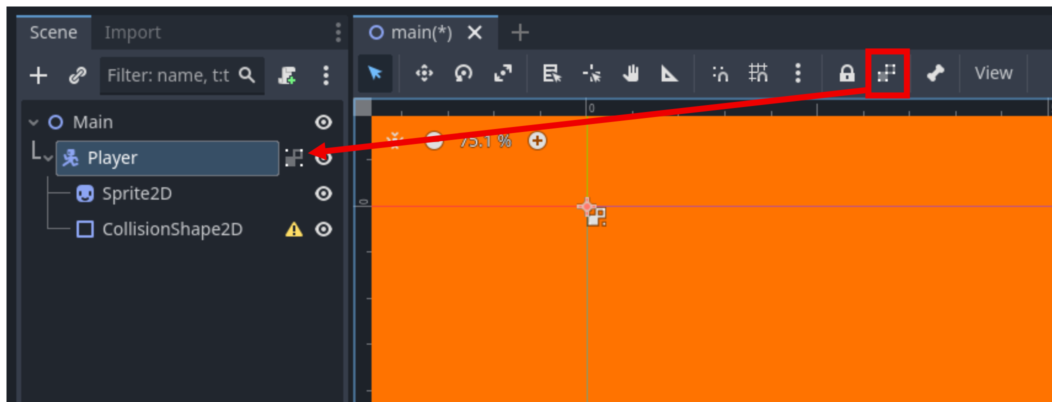


10 Navigate to the Downloads folder on your computer. Select the two newly created **.png** files and drag them into the **Assets** folder in Godot.



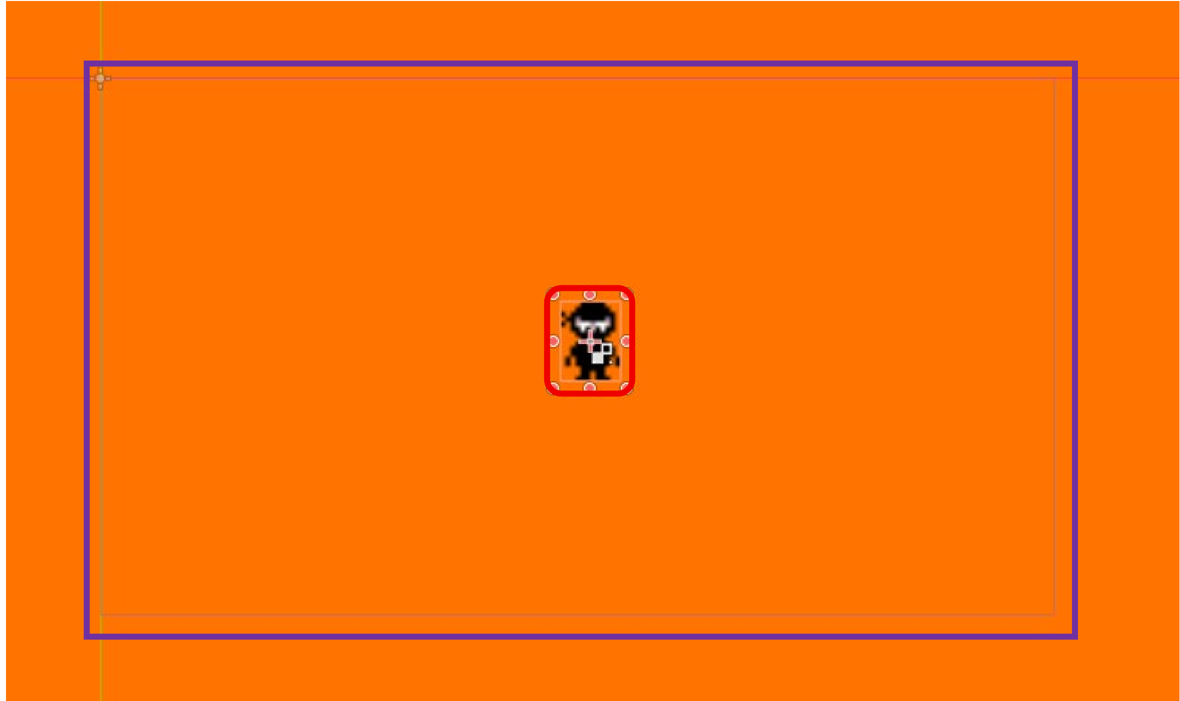
11

Group the Sprite2D and the CollisionShape2D to the player using **Group Selected Nodes**.

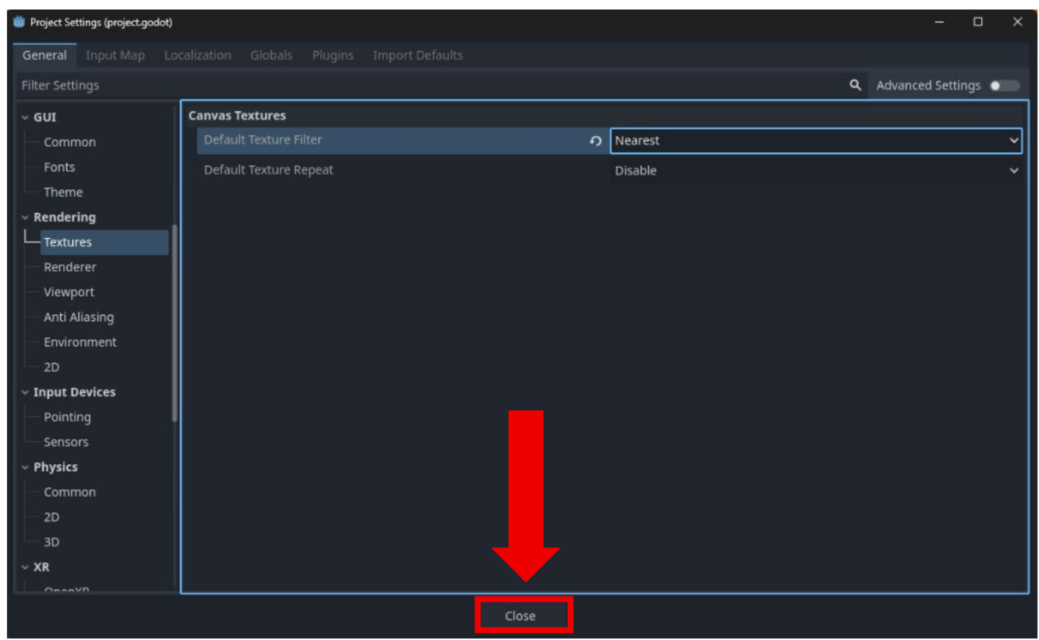


Then, set the **Texture** property of the **Sprite2D** to your custom player asset. Drag **Player** to the center of the viewport and scale its **Sprite2D** so it takes roughly the same size on the screen as the provided image shows.

Before setting the collider's **Shape**, notice how the **Sprite2D** is blurry for pixelated assets. Do you recall how to fix this issue?



12 To fix the blurriness on pixel art, go to **Project Settings > General > Rendering > Textures** to set the Default Texture Filter to **Nearest**, instead of Linear. Click **Close**.



13 Set the **Shape** of the **CollisionShape2D** to match the shape of the Sprite2D and **scale** it so it fits.

Pause for **Sensei Stop #1**!

Check in with a Code Sensei before moving on.
Confirm that Requirements #1-2 have been completed.

Reminder: Save your work!

Requirement #3: Player Movement

Write code to allow the player to move in any direction.

- ☐ Attach a new script to **Player** and set its **Path** so it is created inside of the **Scripts** folder.
- ☐ Inside the new script, create an `@export speed` variable of type `float` with the default value of `500` and a `direction` variable of type `Vector2`.
- ☐ Define the `_process()` method. Inside, set the **direction x** and **y** positions to the user's input by using the `“ui_up”`, `“ui_right”`, `“ui_down”`, and `“ui_left”` actions. After, set `direction` to its normalized version.
- ☐ Set the `velocity` property to `direction * speed` then call `move_and_slide()`.
- ☐ Create a **StaticBody2D** named **Walls** as a child to **Main**, with 4 **CollisionShape2D's** as children. Set the **Shape** and **Position** of each CollisionShape2D to stop the player from moving outside of the screen.

Pause for **Ninja Stop #2**!

Test your project! Does it have...

- Player movement with direction buttons?
- Walls on each side of the viewport?

Reminder: Save your work!

Requirement #3 Hints

- To get the user's input, use `Input.get_axis()` or `Input.get_vector()`.
- To normalize a vector, use `Vector.normalized()`.
- For the walls, use **RectangleShape2D's** or rotated **WorldBoundaryShape2D's**. WorldBoundaryShape2D is preferred when exact pixel amounts can be measured.
- The viewport is **1152px** wide and **648px** tall. Half of those values are **576px** wide and **324px** tall!

Requirement #3 Resources

- Input:

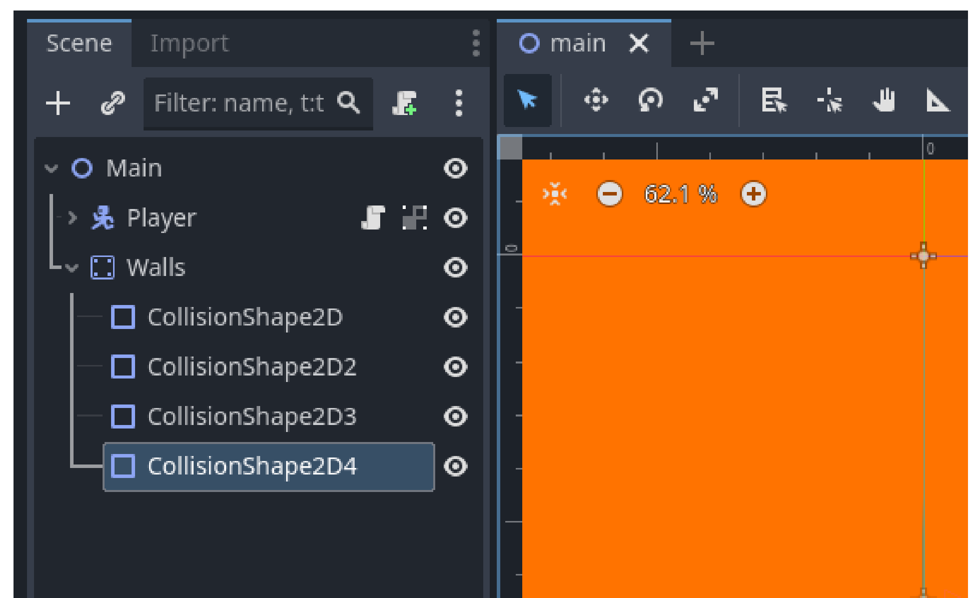
`Input.get_axis()`: checks if two inputs are being pressed and returns a number to show the direction of that input.

Parameters:

1. `negative_action` ***(String):*** the name of the input action for negative direction **("ui_left")**
2. `positive_action` ***(String):*** the name of the input action for positive direction **("ui_right")**

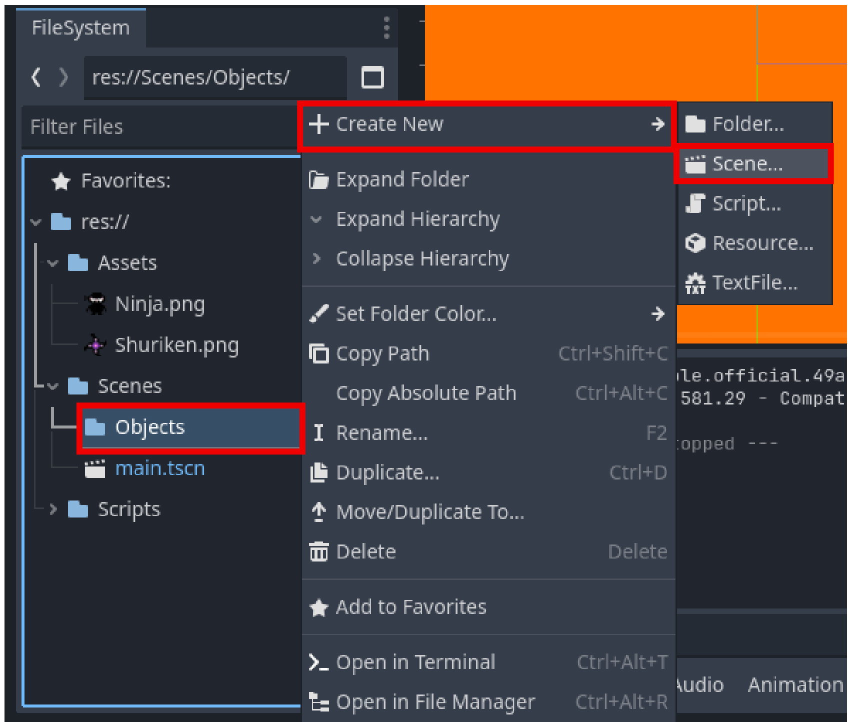
Returns *(float):* number between -1 and 1

- [ ] Example Scene hierarchy:

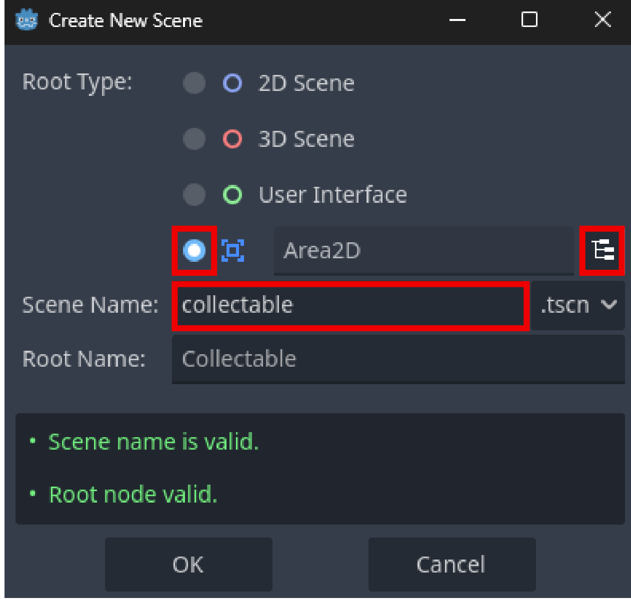


Requirement #4 (Instructions): Collectable Scene

14 Now that the player can move around, time to make the collectables! Create a new scene inside the **Scenes > Objects** folder to represent a collectable.

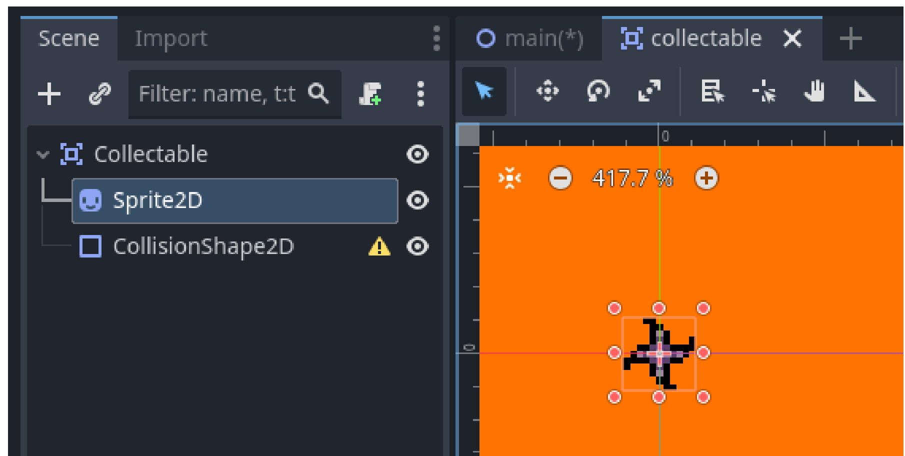


Set the **Root Type** to **Node** and set the type to **Area2D**. Then, set the **Scene Name** to **collectable** and click **OK**.

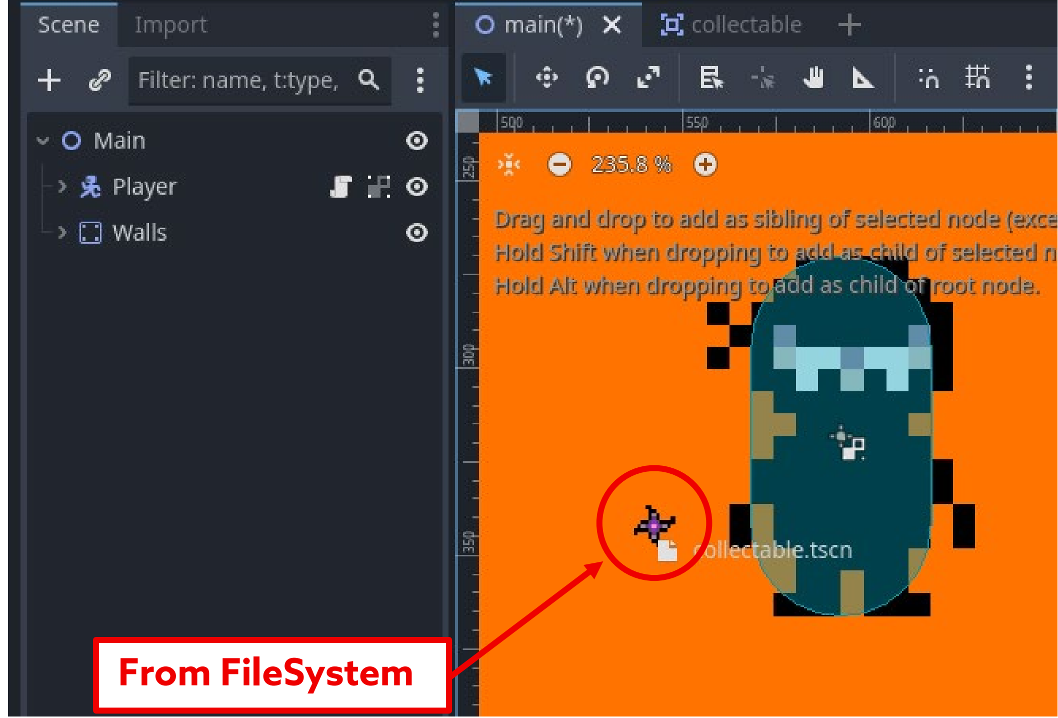


15

Add a **Sprite2D** and a **CollisionShape2D** as child nodes to Collectable. Set the **Texture** property of the **Sprite2D** to your custom collectable asset and press **CTRL+S** to save the Collectable scene.

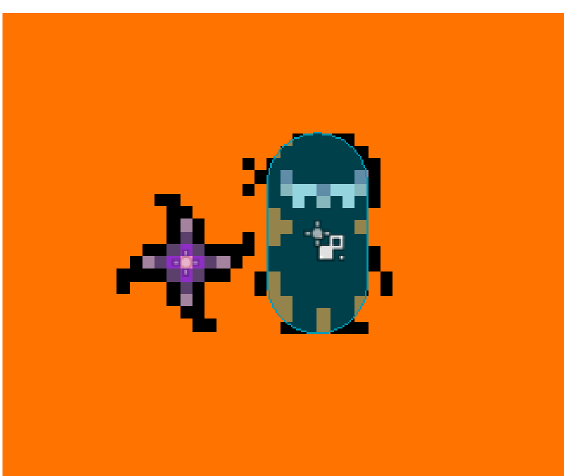


Return to the **main** scene and drag **collectable.tscn** from **FileSystem** and drop it onto the viewport near the player.

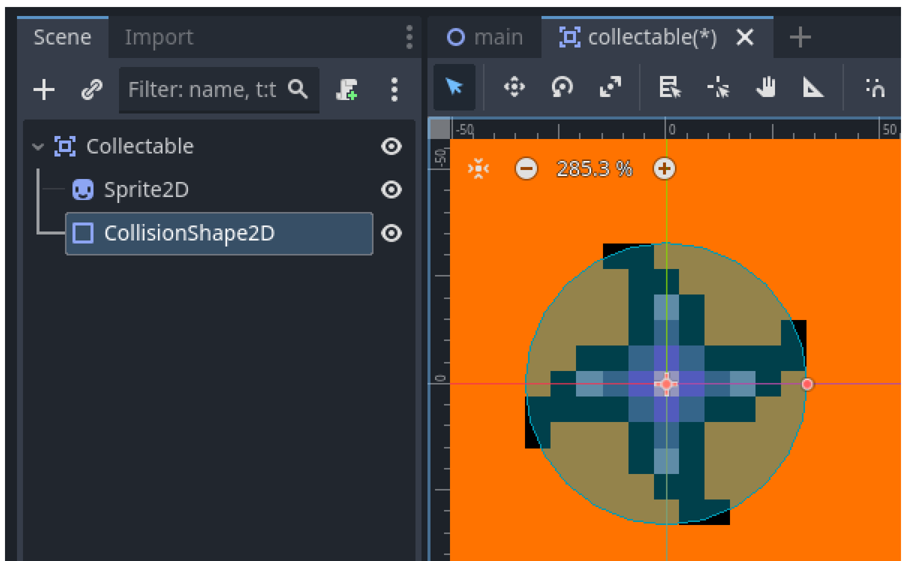


16 Return to the **collectable** scene. Adjust the Sprite2D's **Scale** property in the Inspector, press **CTRL+S** to save the scene, then return to the **main** scene.

Repeat until the collectable is the desired size, then return to the **collectable** scene.

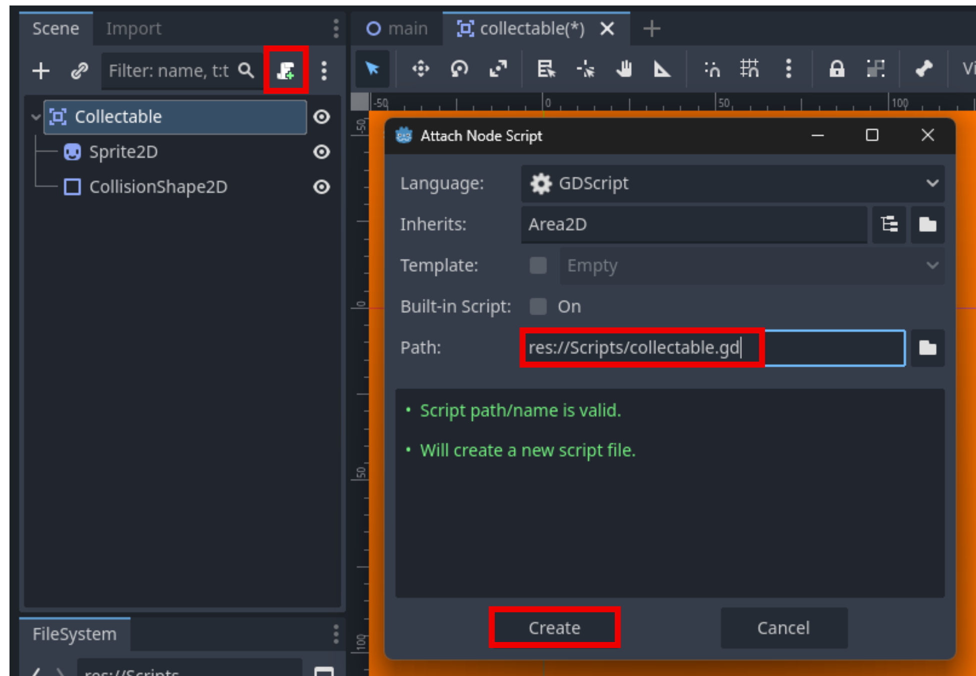


17 Set the **Shape** of the **CollisionShape2D** to match the shape of the Sprite2D and **scale** it so it fits.



18

The collectable needs to randomly generate across the screen! Create a new **collectable.gd** script attached to the **Area2D**.



Inside the script, define the `_ready()` method. Then, set a `rand_x` variable to any value between `0` and `get_viewport().get_visible_rect().size.x` using `randf_range()`.

```
1   extends Area2D
2
3   # _ready() ???
4       # rand_x ???
```

19 Repeat this for a `rand_y` variable; replace `size.x` with `size.y`.

Then, set the `position` property to `Vector2(rand_x, rand_y)` to combine the variables.

```
1   extends Area2D
2
3   # _ready() ???
4       # rand_x ???
5       # rand_y ???
6       # position ???
```

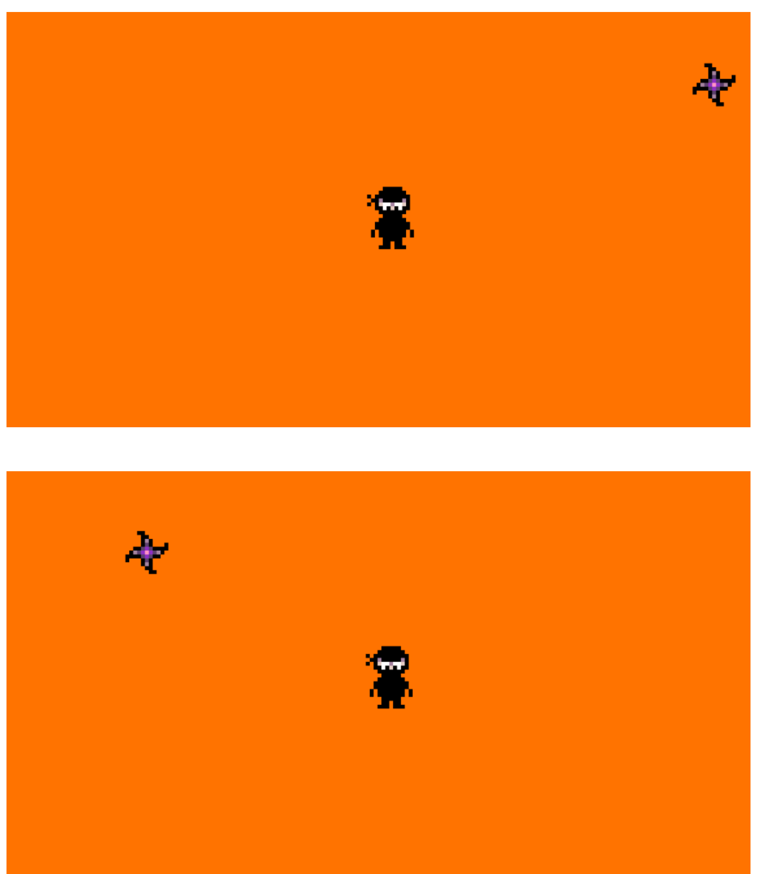
20

Check the code! Update the script as needed.

```
1  extends Area2D
2
3  func _ready() -> void:
4      var rand_x = randf_range(0, get_viewport().get_visible_rect().size.x)
5      var rand_y = randf_range(0, get_viewport().get_visible_rect().size.y)
6      position = Vector2(rand_x, rand_y)
```

21

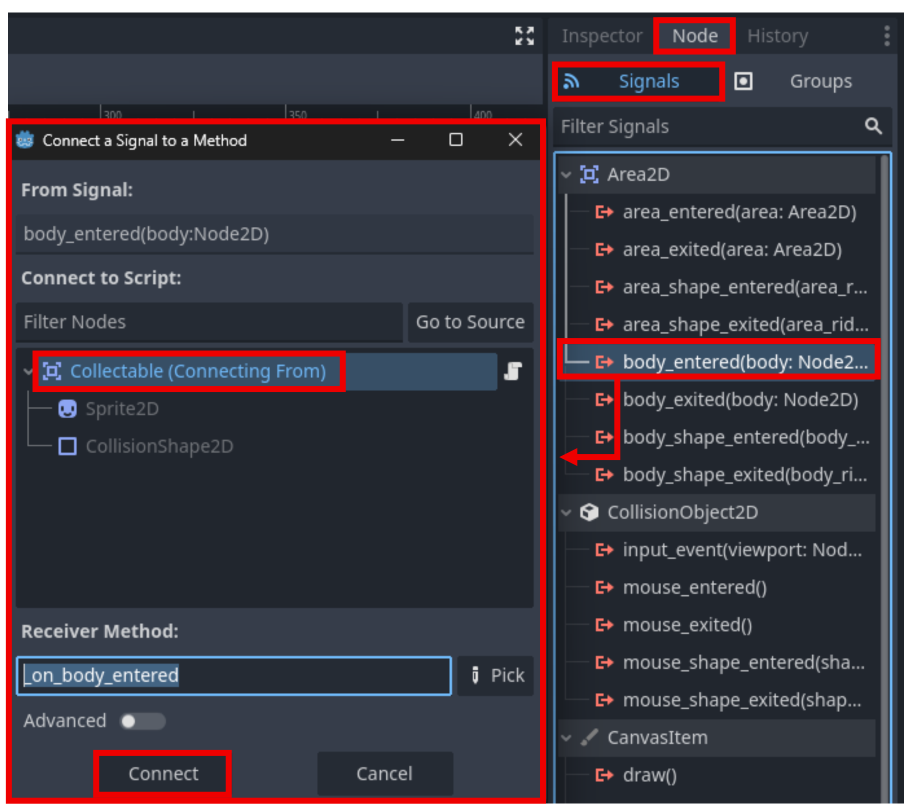
Playtest the project. Does the collectable teleport to a different location with each playtest?



22

In **Scene**, ensure the collectable scene's root node is selected. Then, toggle **Inspector** to **Node** and open the **Signals** section.

Connect the **body_entered()** signal to a new `_on_body_entered()` receiver method in Collectable's attached script.



23

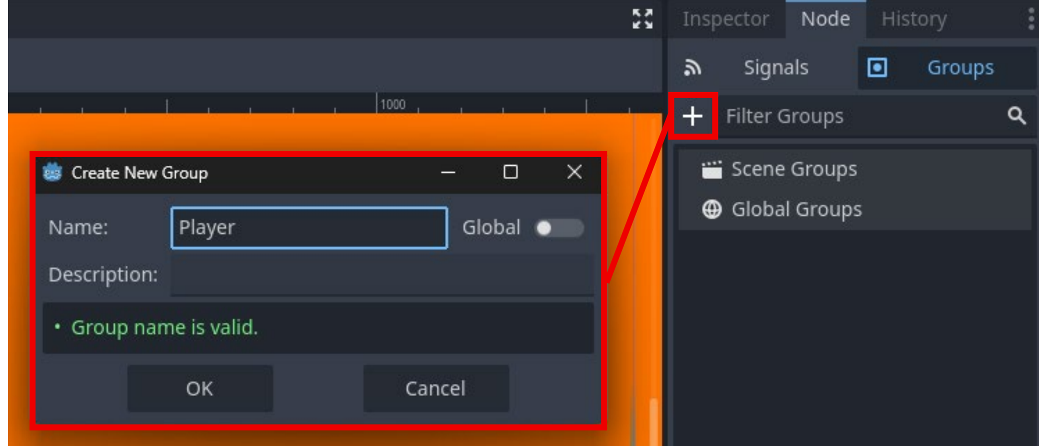
In the new `_on_body_entered()` method, write an `if`-statement that checks whether the `body` is in group `"Player"`. Then, call the `body.add_score()` method and call `queue_free()` to delete the collectable.

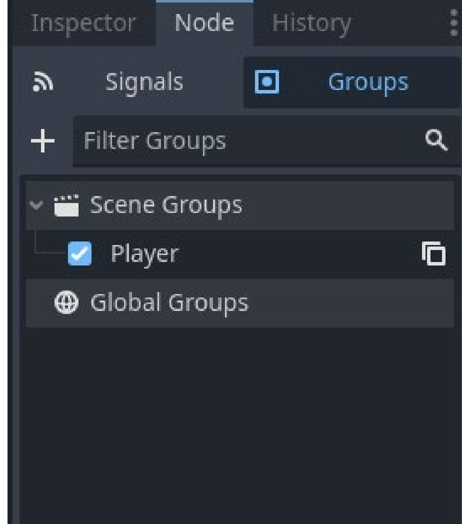
```
 9  func _on_body_entered(body: Node2D) -> void:
10      # if ???
11          # body ???
12          # queue_free ???
13
```

24 Check the code! Update the script as needed.

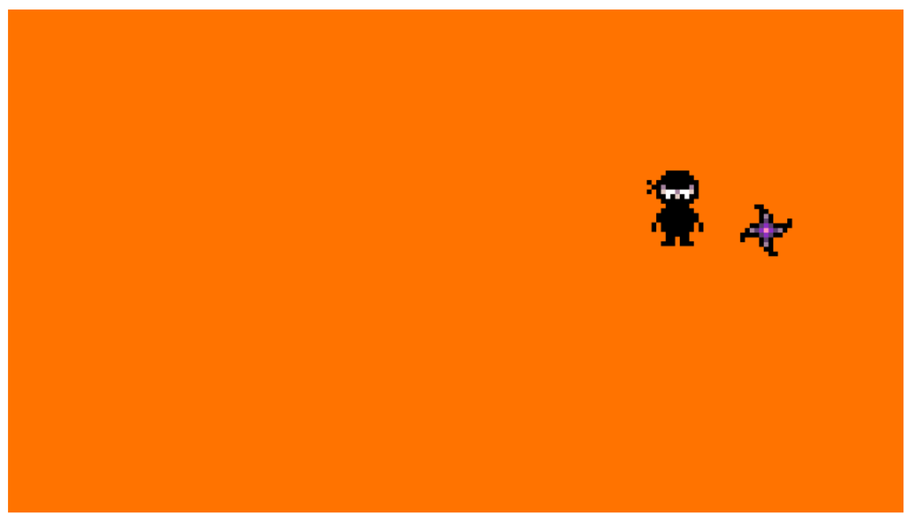
```
9   func _on_body_entered(body: Node2D) -> void:
10      if body.is_in_group("Player"):
11          body.add_score()
12          queue_free()
13
```

25 Return to the **main** scene. In Scene, select **Player** and toggle Inspector to **Node > Groups.** Use the **+** icon to create the new Player group and click **OK**.

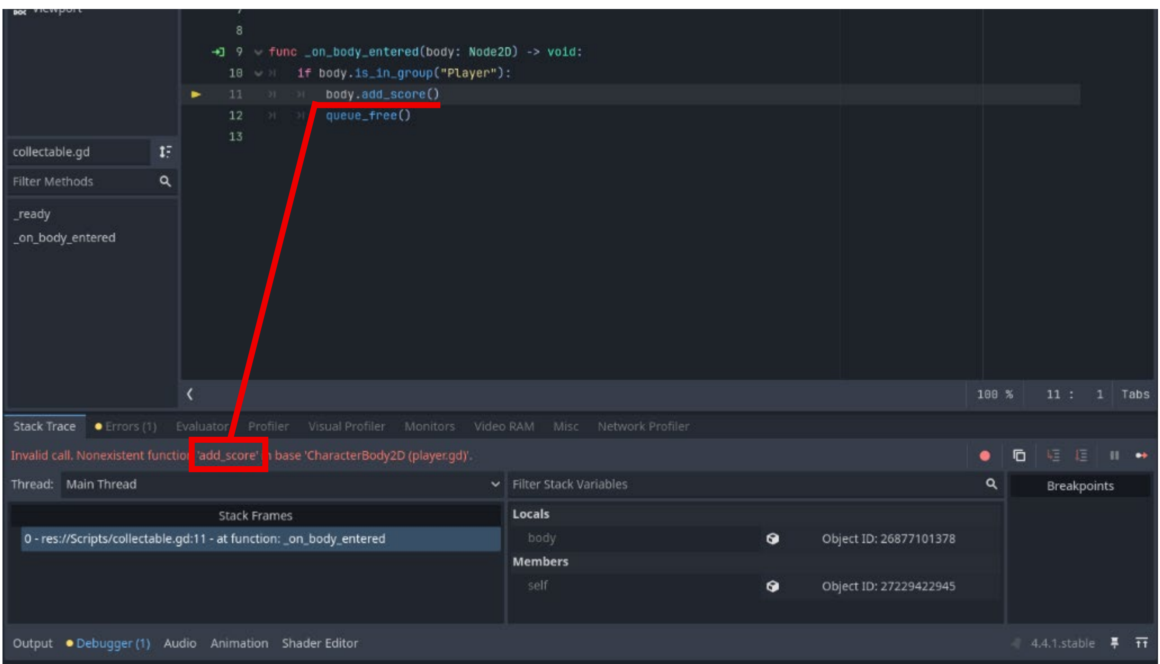




26 Playtest the project. What happens when the player collides with a collectable?



27 Oh no! There seems to be an error. Godot couldn't find the `add_score()` method in **player.gd** because it hasn't been implemented yet!



Navigate to **player.gd**.

28 At the top of the script, declare a `score` variable of type `int` and set it to `0`.

```
1    extends CharacterBody2D
2
3    @export var speed: float = 500
4    var direction: Vector2
5    var score: int = 0
6
```

29 Define a new `add_score()` function that returns `void`. Inside, increase the `score` variable by `1`.

```
15  func add_score() -> void:
16      score += 1
```

Playtest the project. Does the game throw any more errors when the player collides with the collectable?

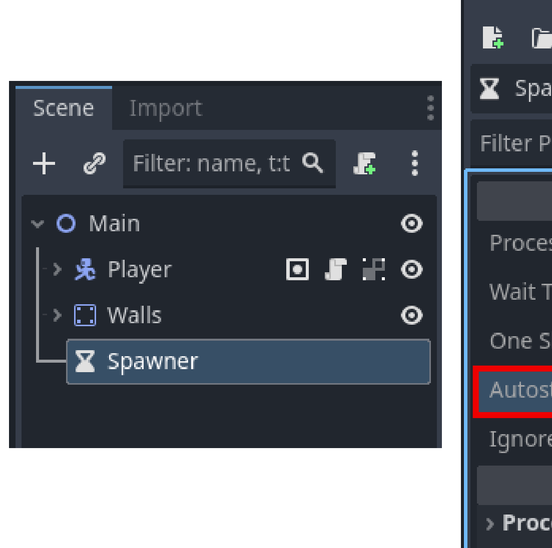
Pause for **Sensei Stop #2**!

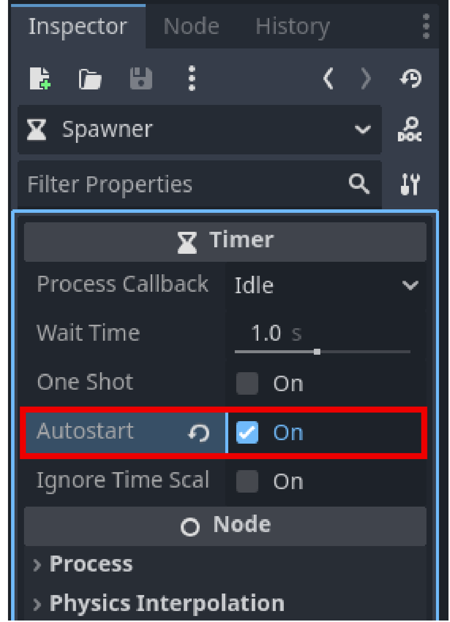
Check in with a Code Sensei before moving on. Confirm that Requirements #3-4 have been completed.

Reminder: Save your work!

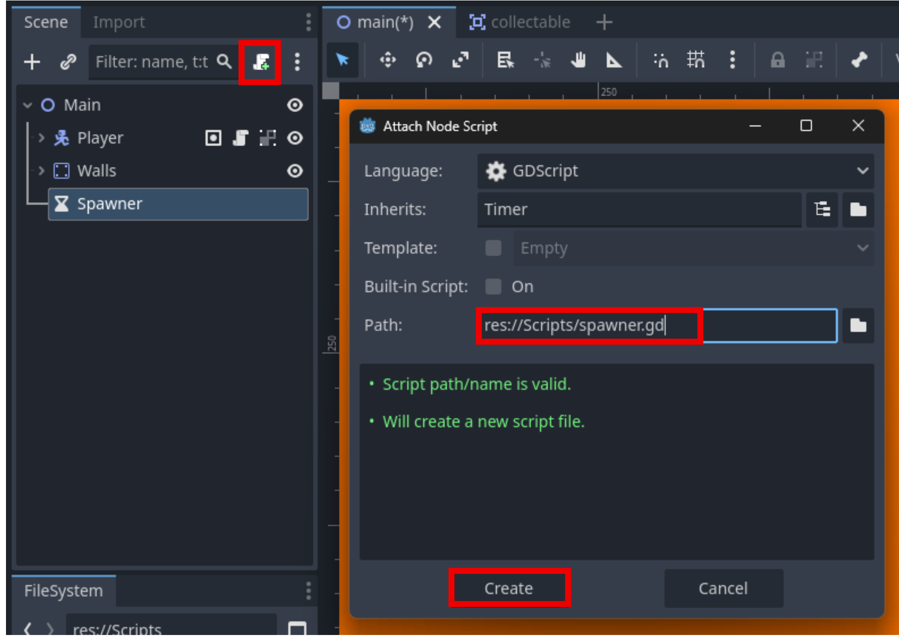
REQUIREMENT #5 (INSTRUCTIONS): SPAWNER

30 Delete the collectable node in Scene and add a **Timer** as a child to the root node. Rename the Timer to **Spawner** and enable **Autostart** in its Inspector.



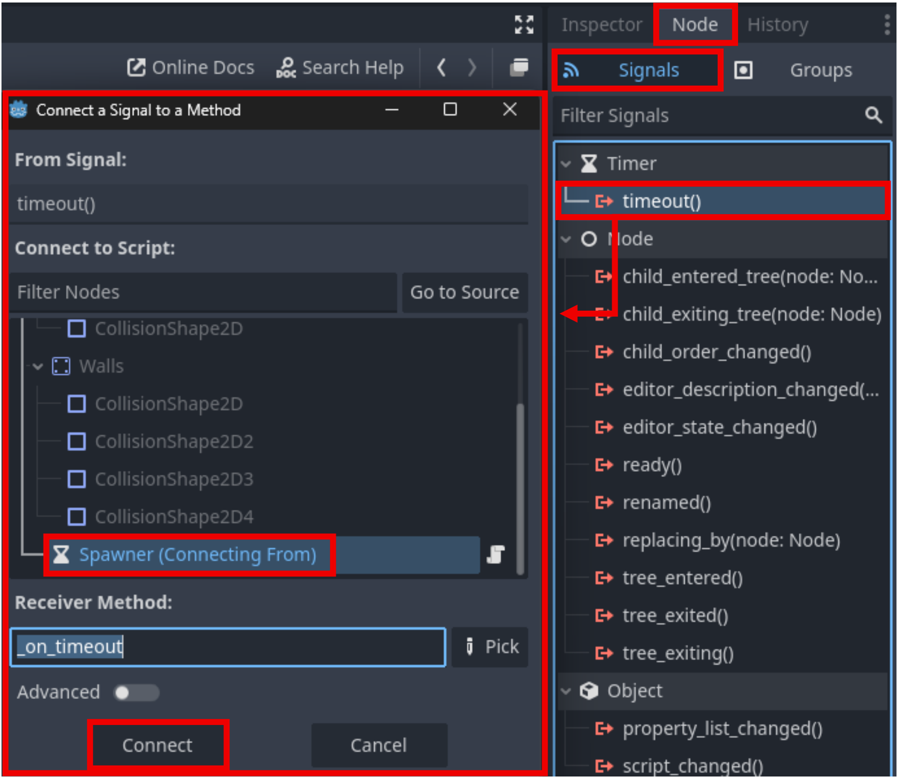


31 The Spawner will handle instantiating the collectables! Create a new **spawner.gd** script attached to the **Timer**.



32 In **Scene**, ensure that Spawner is selected. Then, toggle **Inspector** to **Node** and open the **Signals** section.

Connect the **timeout()** signal to a new `_on_timeout()` receiver method in Spawner's attached script.



33 Add code to spawner.gd so it preloads collectable.tscn into a `const` and whenever the timer reaches 0 it adds the instantiated scene as a child.

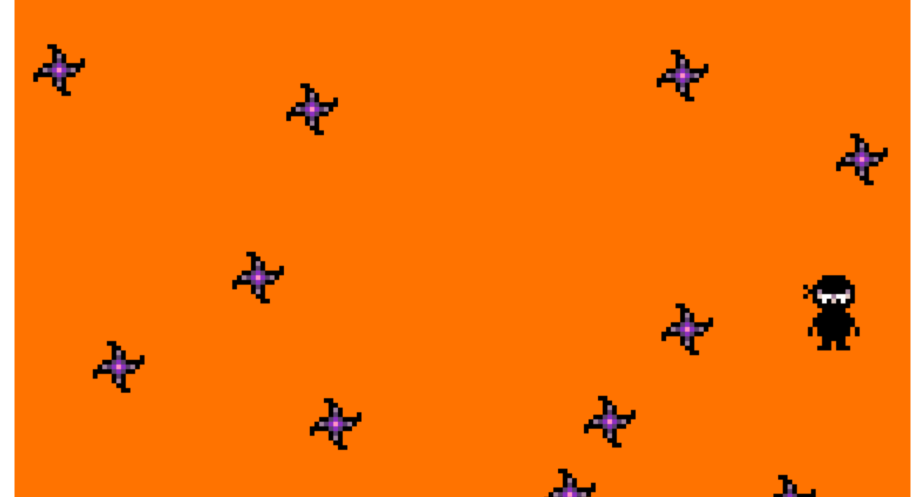
Drag **collectable.tscn** from **FileSystem** into the code editor then hold **CTRL** and release the drag to create the `COLLECTABLE` constant.

```
1   extends Timer
2
3   # COLLECTABLE
4
5   func _on_timeout() -> void:
6       # add_child ???
7
```

34 Check the code! Update the script as needed.

```
1   extends Timer
2
3   const COLLECTABLE = preload("res://Scenes/Objects/collectable.tscn")
4
5   func _on_timeout() -> void:
6       add_child(COLLECTABLE.instantiate())
7
```

35 Playtest the project. Are more collectables appearing over time?



REQUIREMENT #6: USER INTERFACE

Build the UI and update a label whenever the score updates!

- ☐ Add a **Canvaslayer** as a child to the root node and add a **Label** as a child node to the CanvasLayer.
- ☐ Rename both nodes to something that makes sense, like **PlayUI** and **ScoreText**.
- ☐ Set the default **Text** for the Label to **Score: 0** and set its **Font Size** anywhere within **40-60px**.
- ☐ In player.gd, create an `@onready` or an `@export` variable that stores **ScoreText**.
- ☐ In the `add_score()` method, set the `text` property of **ScoreText** to `“Score: “ + str(score)`.
- ☐ Add another **Label** as a child node to the CanvasLayer and rename it to something like **CountdownText**. Set its **Text** in the Inspector to **00.00** and set its **Font Size** to be slightly bigger than the score text.
- ☐ **Anchor** the new label anywhere that isn’t where the score text is placed.

Pause for **Ninja Stop #3**!

Test your project! Does it have...

- A score label that updates in real time?
- A countdown label?

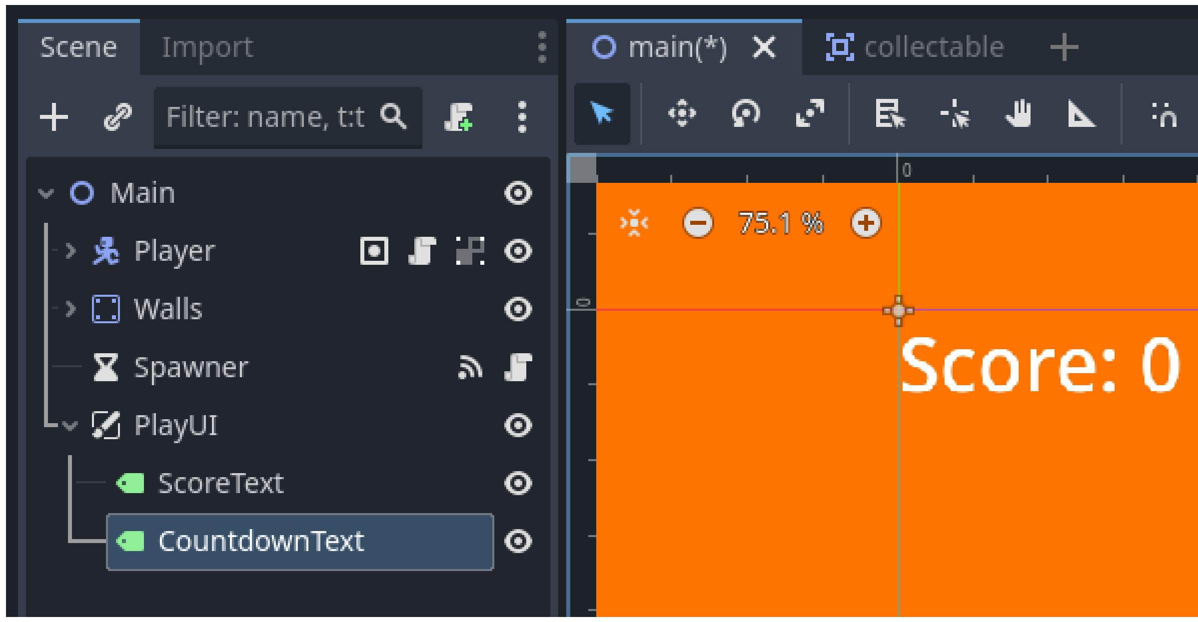
Reminder: Save your work!

Requirement #6 Hints

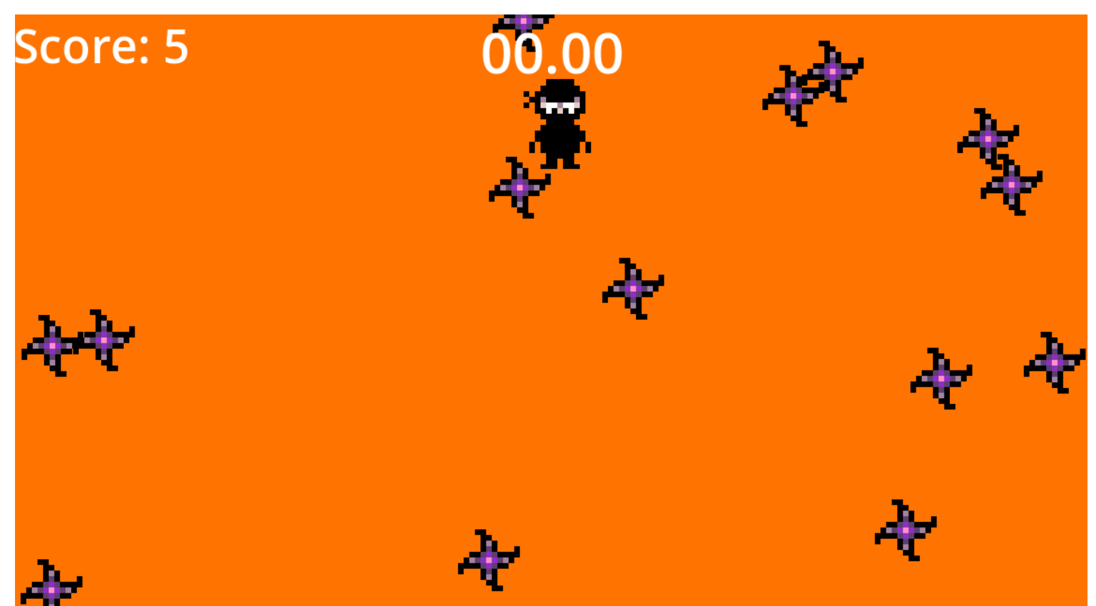
- To set the font size of a Label node, a **new LabelSettings** resource must be created in the Inspector first.
- To easily create and `@onready` variable, drag a node from Scene into the code editor and hold **CTRL** while releasing.
- Use the **Anchor Presets** button in the toolbar to easily change the anchor of any Control node.

Requirement #6 Resources

- Anchor Presets: https://docs.godotengine.org/en/4.4/tutorials/ui/size_and_anchors.html#anchor-presets
    - Refer back to Activities 12 – 14 in Silver Belt for help with Anchor Presets.
- Example Scene hierarchy:



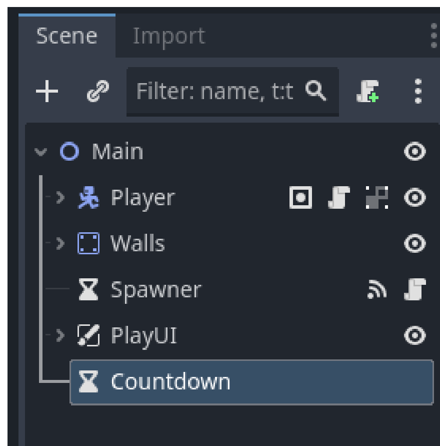
- Example UI layout:

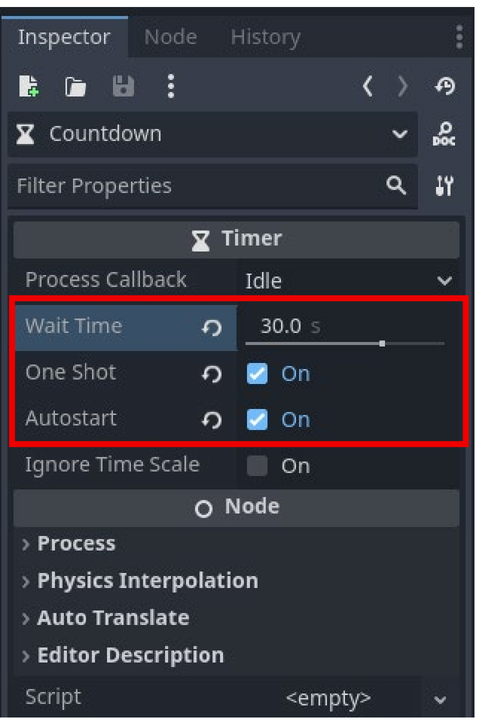


Requirement #7 (Instructions): Countdown

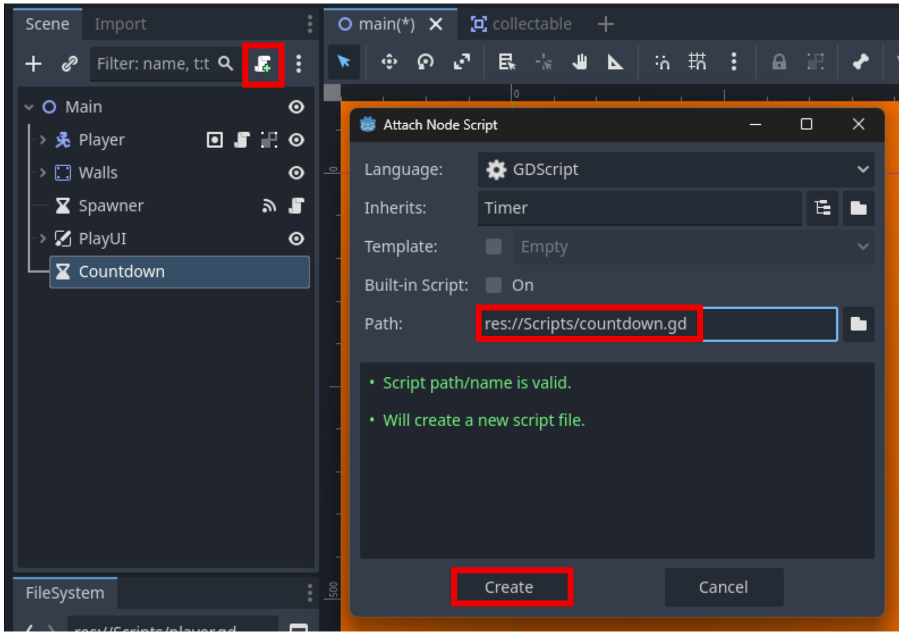
36 Now that the score works, it's time to work on the countdown timer. Add a new **Timer** as a child node to Main and rename it to **Countdown**.

In the Inspector, set the **Wait Time**, **One Shot**, and **Autostart** properties to **30.0s**, **On**, and **On** respectively.

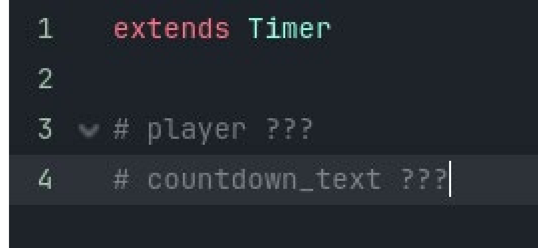




37 The countdown needs to update the UI label! Create a new **countdown.gd** script attached to the **Timer**.



Inside the script, create two `@onready` or `@export` variables to store nodes: `player` and `countdown_text`. They should store the Player and CountdownText nodes respectively.



```
1   extends Timer
2
3   # player ???
4   # countdown_text ???
```

Reminder:

To easily create an **@onready** variable, drag a node from Scene into the code editor and hold **CTRL** while releasing.

38 Define the `_process()` method. Inside, update `countdown_text`'s `text` property to `"%.2f" % time_left` using Countdown's `time_left` property.

```
1    extends Timer
2
3    # player ???
4    # countdown_text ???
5
6    # _process() ???
7        # countdown_text.text ???
```

Pro Tip:

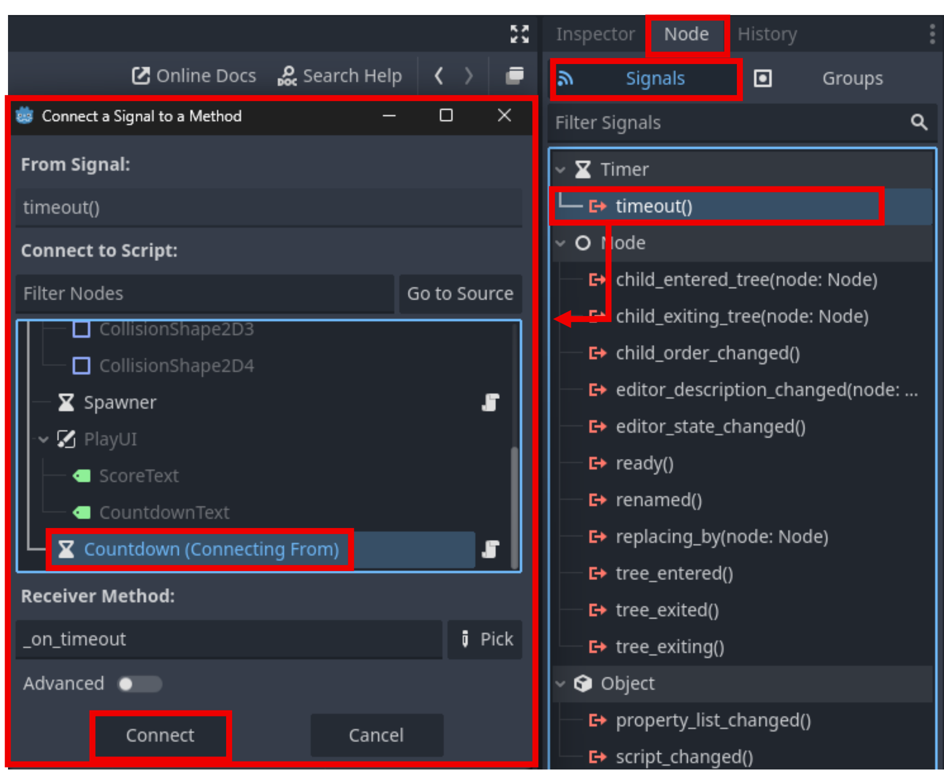
This is a usage of Godot's string formatting. **%** is not only a **modulo** operator when used with two **ints**, but it is also the **string formatting** operator when used with a string and some other data type. When the string on the left contains a **%**, that tells the operator *where* to insert the right operand. The **f** means it is expecting a float and the **.2** ensures there are two digits after the decimal point.

39 Check the code! Update the script as needed.

```
1  extends Timer
2
3  @onready var player: CharacterBody2D = $"../Player"
4  @onready var countdown_text: Label = $"../PlayUI/CountdownText"
5
6  func _process(delta: float) -> void:
7      countdown_text.text = "%.2f" % time_left
```

40 In **Scene**, ensure that Countdown is selected. Then, toggle **Inspector** to **Node** and open the **Signals** section.

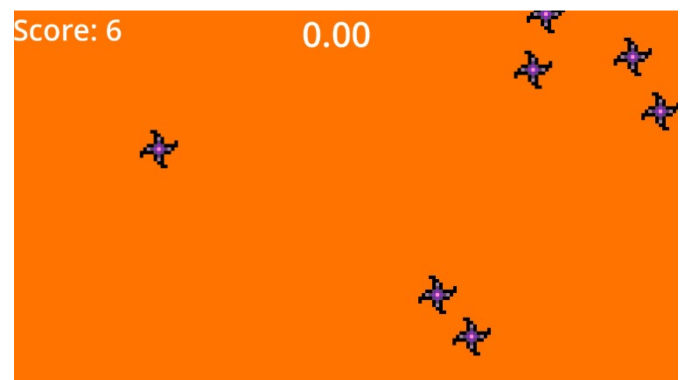
Connect the **timeout()** signal to a new `_on_timeout()` receiver method in Countdown's attached script.



41 In the new `_on_timeout()` method, call `queue_free()` on the player to end the game.

```
9   func _on_timeout() -> void:
10      player.queue_free()
11
```

42 Playtest the project. Does the player disappear when the timer reaches 0?



Pause for **Sensei Stop #3**!

Check in with a Code Sensei before moving on. Confirm that Requirements #5-7 have been completed.

Reminder: Save your work!

Requirement #8: Game Over UI

Build the Game Over UI so it appears when the countdown timer runs out!

- ☐ Add another **Canvaslayer** as a child to the root node and add a **Label** and a **Button** as children to it. **Anchor** both the Label and Button to the **center** of the screen using Anchor Presets.

- ☐ Rename the nodes to something that makes sense, like **GameOverUI**, **GameOverText**, and **RestartButton**.

- ☐ Set the **Text** for the Label to **Time's Up!** and set its **Font Size** anywhere within **40-60px**. Set the **Text** for the Button to **Restart** and set its **Font Size** anywhere within **30-40px**.

- ☐ Adjust the positions of the Label and Button using **Move Mode (W)** so they are not overlapping.

- ☐ Attach a new script to the Button. Then, connect the Button's `_pressed()` signal to the new script. Use `get_tree().call_deferred()` to call `reload_current_scene()` at the end of the current frame.

- ☐ In countdown.gd, create an `@onready` or an `@export` variable that stores the **Canvaslayer** for the game over UI. Then, in the `_on_timeout()` method, set the `visible` property of the Canvaslayer to `true`.

- ☐ In spawner.gd, create an `@onready` or an `@export` variable that stores the **Canvaslayer** for the game over UI. Then, in the `_on_timeout()` method, add an `if`-statement before instantiating the `COLLECTABLE` const that checks if the game over UI is `not visible`. This ensures that collectables only spawn while the game is still in play.

- ☐ Set the Canvaslayer to be **invisible** in **Scene** so it only appears when the timer runs out.

Pause for **Ninja Stop #4**!

Test your project! Does it have...

- A game over UI that only appears at the end?
- Collectables stop spawning when the game ends?
- A restart button that reloads the scene?

Reminder: Save your work!

Requirement #8 Hints

- To set the font size of a Label node, a **new LabelSettings** resource must be created in the Inspector first. For Button nodes, font size can be set in **Theme Overrides > Font Sizes**.

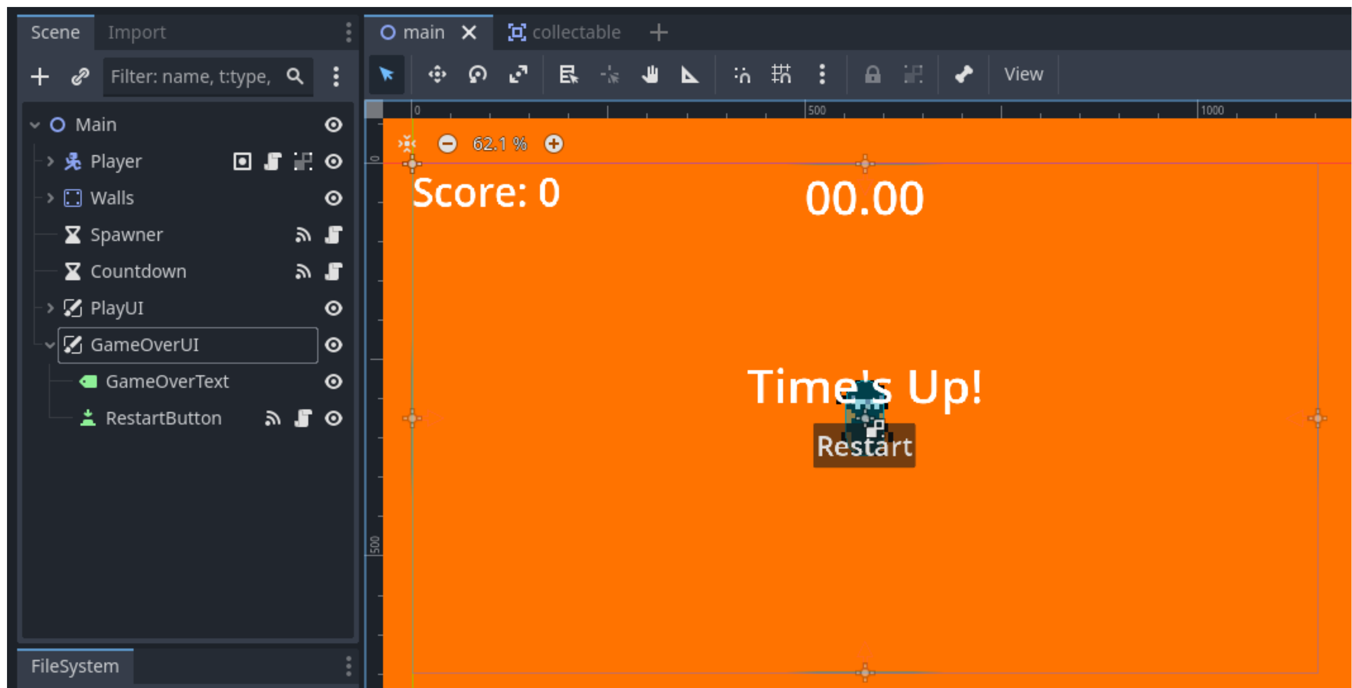
- To easily create and `@onready` variable, drag a node from Scene into the code editor and hold **CTRL** while releasing.

- Use the **Anchor Presets** button in the toolbar to easily change the anchor of any Control node.

Requirement #8 Resources

- Anchor Presets: https://docs.godotengine.org/en/4.4/tutorials/ui/size_and_anchors.html#anchor-presets
  - Refer back to Activities 12 – 14 in Silver Belt for help with Anchor Presets.

- Example Scene hierarchy and UI layout:



Pause for **Sensei Stop #4**!

Congratulations on learning the Platinum Belt project process in Godot! Great job!

Before submitting, check in with a Code Sensei to confirm that Requirement #8 has been completed, then reflect on the following:

- What did you learn about asset creation in Piskel?
- What did you enjoy most when creating this project?
- What was something you found difficult and why?

Reminder: Save your work!