CODE NINJAS®

Bronze Belt Ninja Guide

Activity 10: Dropping Bombs Part 4

User Interface

Games by themselves are fun to play, but there is a lot of information about a game that needs to be communicated to the user to understand what's going on. This communication occurs through what is referred to as "User Interface," or UI for short. What are some things that UI can communicate? Think of popular games that you've created or played; common UI elements include the game's title, information to tell the user how to get started, game mechanics such as a score or timer, and information to play again.

To be effective, UI needs to be visible to the user and not obstructed by game elements. To achieve this first requirement of UI, a **CanvasLayer** node can be used. A **CanvasLayer** node holds 2D elements that will be visible in the viewport that the camera node creates. Additionally, the **CanvasLayer** can be set to supersede all z layers in relation to the camera, so that it will always be visible no matter the position of any other objects in the scene. This makes it especially useful for informational overlays like health bars or title screens. Child nodes to **CanvasLayer** nodes must be edited in the **2D** editor window. **Control** nodes child to **CanvasLayer** nodes make up the objects seen within the UI, such as **Textures** and **Labels**. **Textures** are used to hold images, and **Labels** are used to hold text for the player to read.

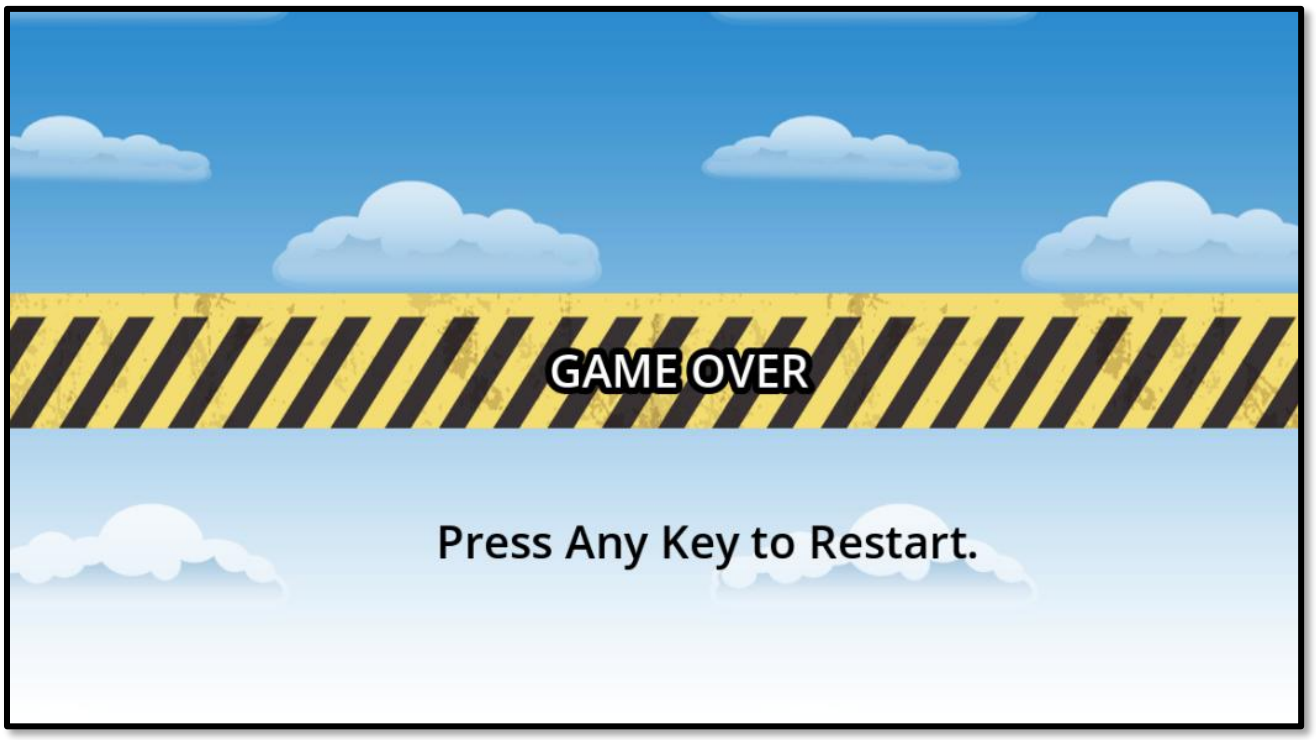
Second, the UI must appear at the appropriate times during the game. If the UI cannot be seen, it cannot provide value to the player. If UI appears at the wrong time, it can mislead the player and create a confusing experience. To achieve this second requirement of UI, scripting the visibility of UI elements is vital. This allows the appearance of UI to coincide with game events that are dealt with in the code. Setting the visibility property of UI to either true or false will make the UI appear or disappear at the correct time, keeping the right information available to the user when needed.

```
38  func restart():
39      game_end.visible = false
40      Globals.started = false
41      Globals.spawning = true
42      spawn_player()
43
```

Activity 10: Dropping Bombs Part 4

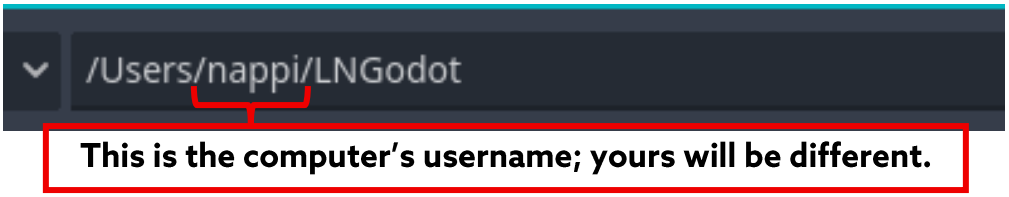
In this project, you will expand on Dropping Bombs by adding UI elements for a Game Start and Game End screen. Additionally, you will script these UI elements to appear and disappear at the appropriate time.

By the end of this activity, you will have explored how to create, edit, and script UI.



1 Remember all projects will be stored in a path like:
/Users/[MyComputerUsername]/[MyInitials]Godot

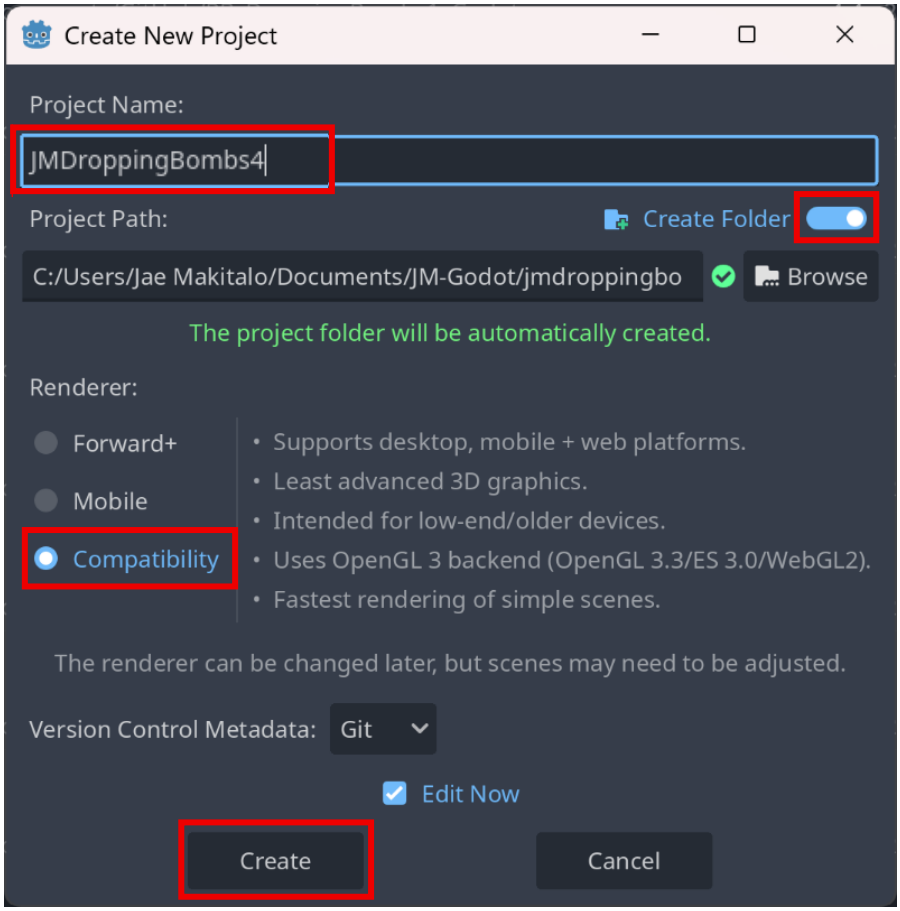
Don't worry if your path looks slightly different from the image shown! All computers have their own username.



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

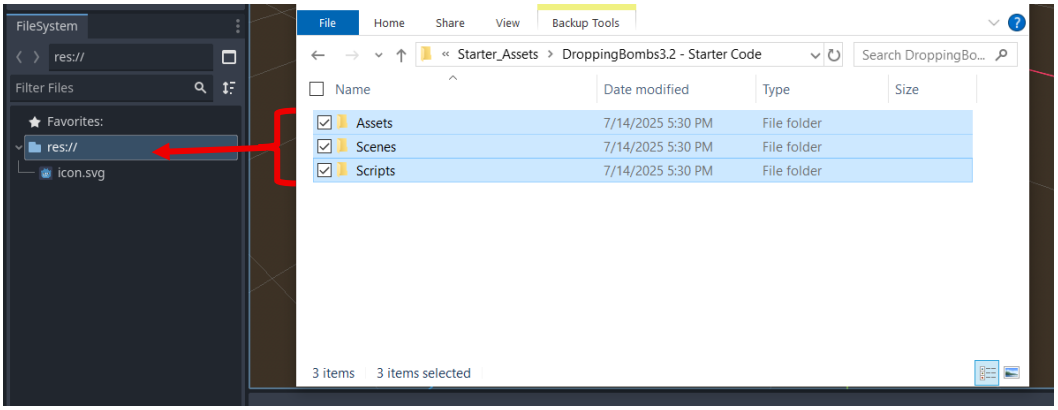
A **Create New Project** window will pop up. Name the project **[MyInitials]DroppingBombs4.**

Check that **Create Folder** is turned on, and that the **Compatability** mode for the renderer is being used. Then click **Create**.



3 Don't create the **main scene** and **Main root** node just yet!

Extract **BB Activity 10 – Ninja Starter Pack.zip** and select all folders inside. Drag them into the **res://** folder in **FileSystem**.



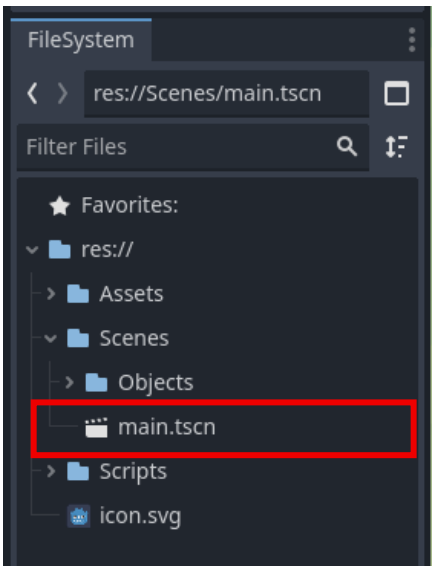
At the top center of the editor, check that **3D** is selected.

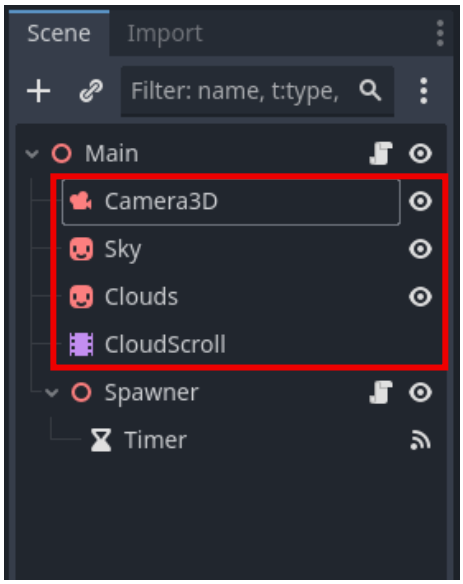
Reminder:

Double-click on the folder icon from **Downloads**. Then **right-click** on the zip file and select **Extract All.** Press **CTRL + J** on the keyboard to reopen the **File Explorer**.

4 In **FileSystem**, navigate to **main.tscn** and double click to open it. Notice the main scene already has the camera, sky, and clouds created from part 3.

The Player and Animation nodes from Part 3 are not visible in the **main scene**. They are found in a separate scene, under the **Scenes > Objects** folder.



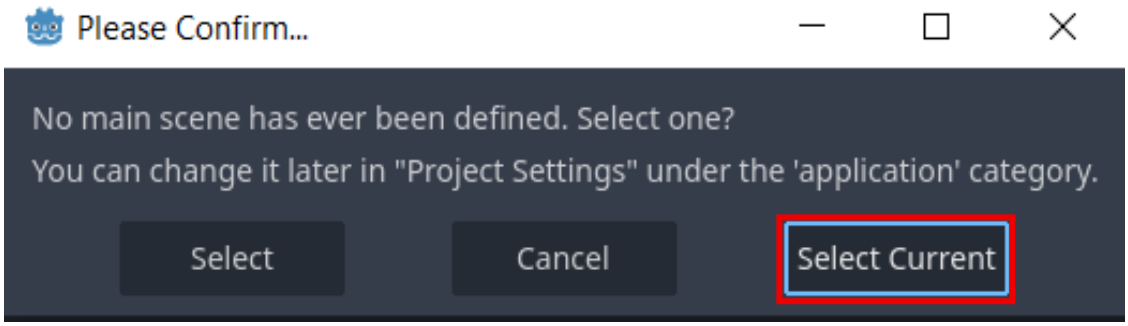


Reminder:

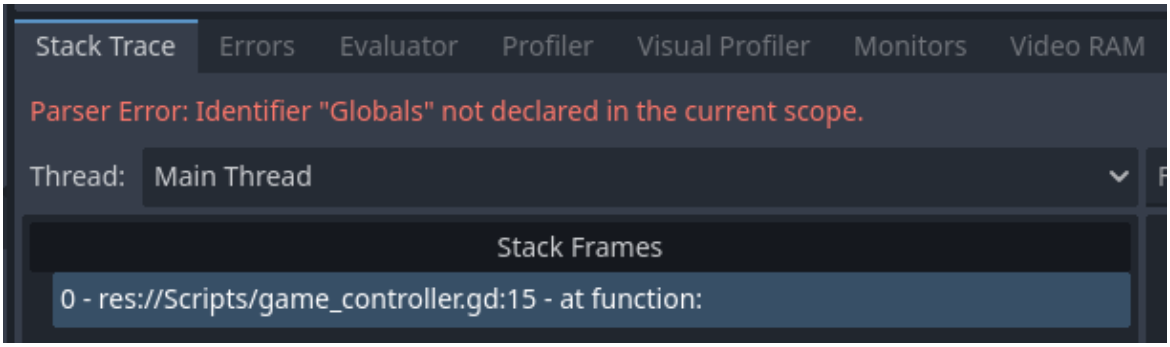
Click the arrows next to the folders to open them.

5 In the top right corner, click the **play** button to run the game.

Click **Select Current** to define the main scene.

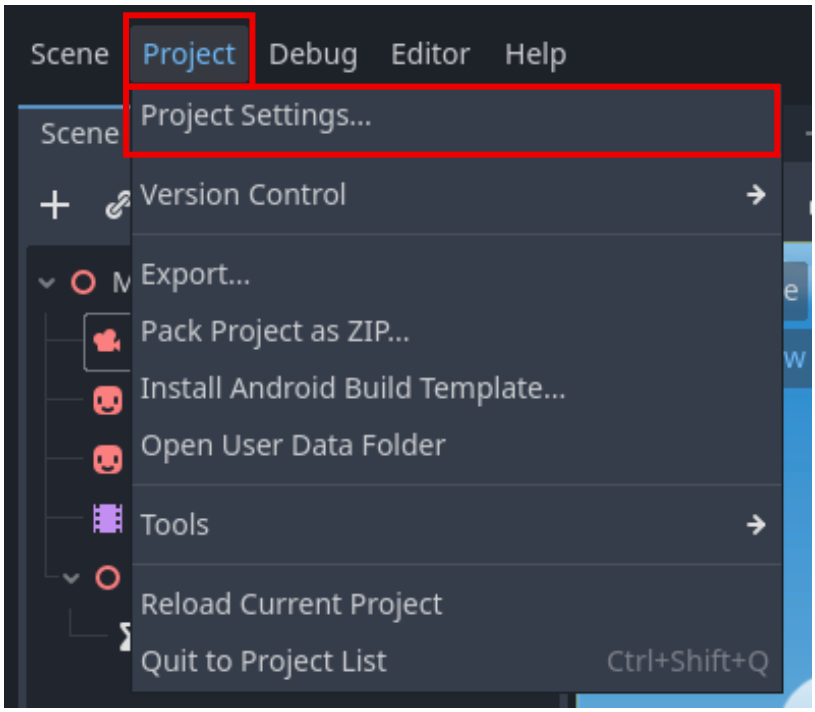


6 The game will not load on the first playtest and will crash immediately. There will be a parser error stating that **Globals** is not declared in the current scope. Ignore this; it will get fixed in the next steps.

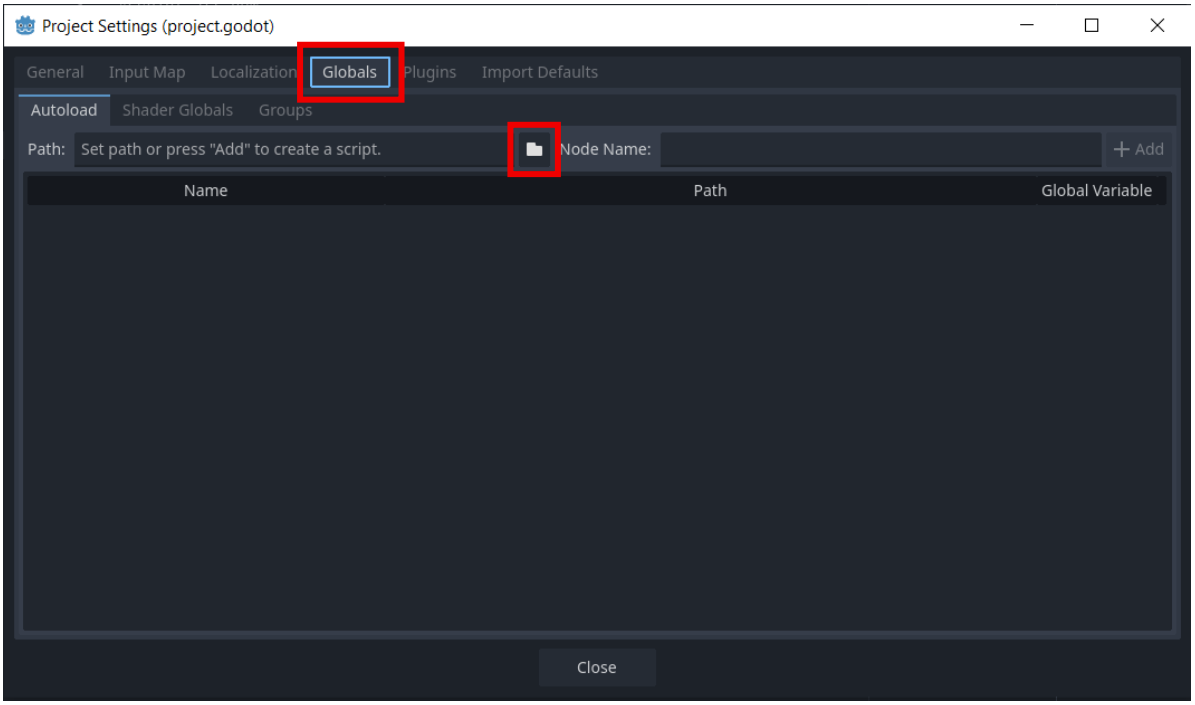


7 Add a Globals script to the project.

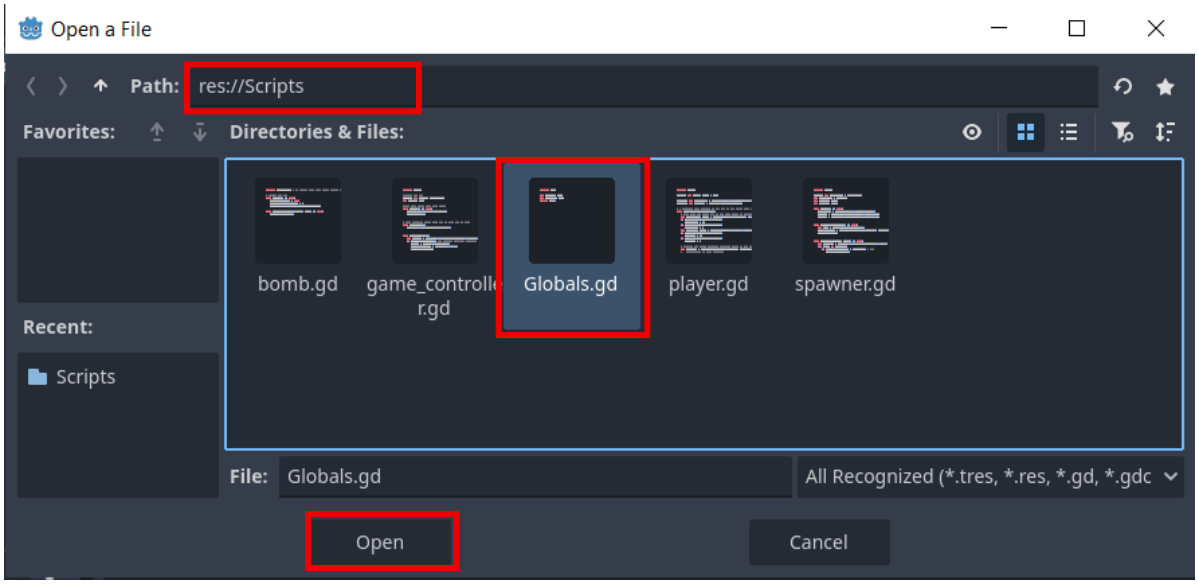
In the top left corner of the editor, select **Project**. Select **Project Settings** from the dropdown menu.



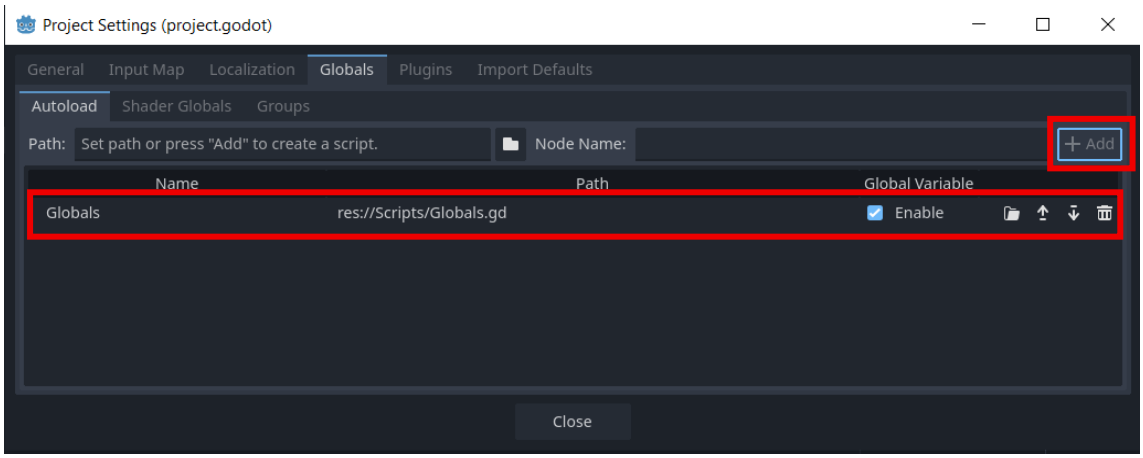
8 In the **Project Settings** window, select the **Globals** tab. Then, click the **folder icon.**



9 In the new popup window, open the **Scripts** folder. Select the **Globals.gd** file and click **Open**.

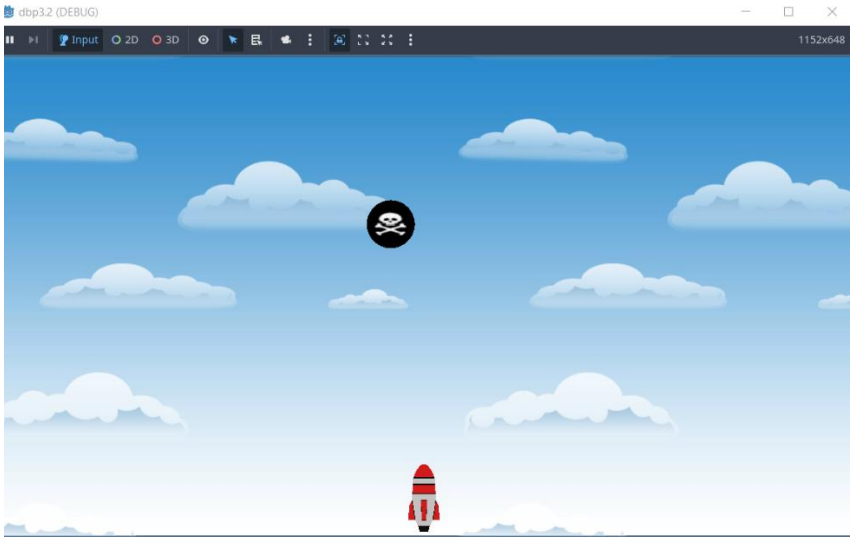


10 Click **Add** and double check that the node has appeared in the list below.



11 Playtest the game. Now that the Globals script is set, the playtest window will open.

Notice that the game is the same as where **Dropping Bombs Part 3** left off.



Pause for **Sensei Stop #1**!

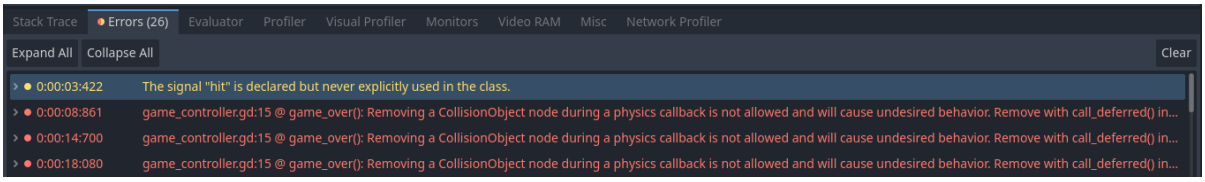
Before continuing, check in with a Code Sensei and make sure the **Globals** node, **Ninja Starter Pack,** and **main scene** are set up properly.

Reminder: Save your work!

12

Close the **Project Settings** window.

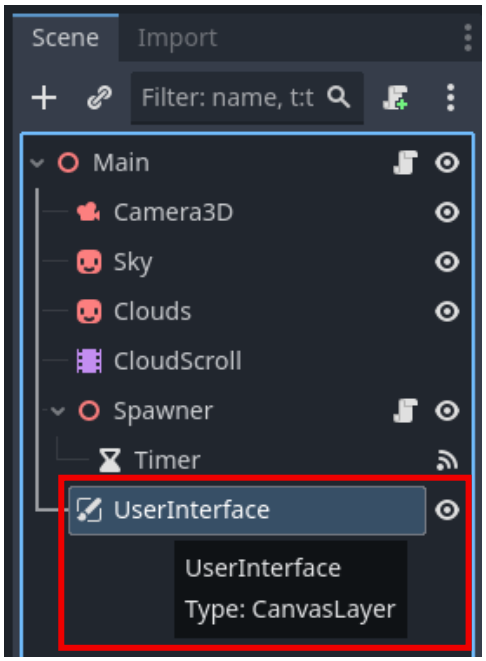
Notice that there are errors that appear when the Rocket gets hit by a Bomb. Ignore this; it will get fixed later.



13

Create the game introduction UI.

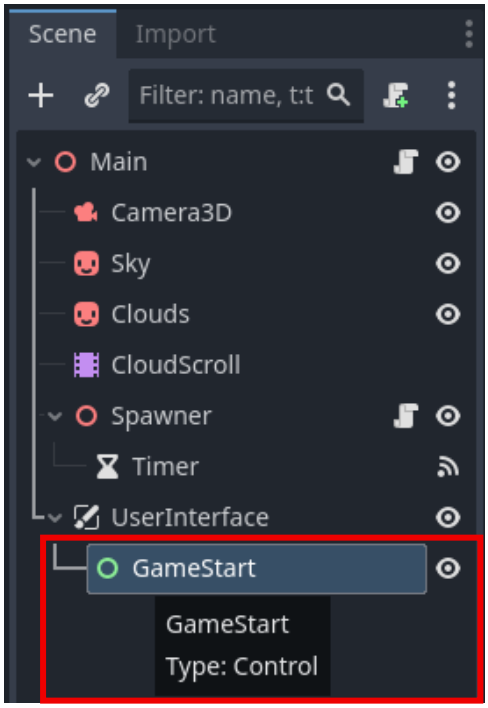
In **Scene**, add a new **CanvasLayer** node as a child to the **Main root**. Rename it **UserInterface**.



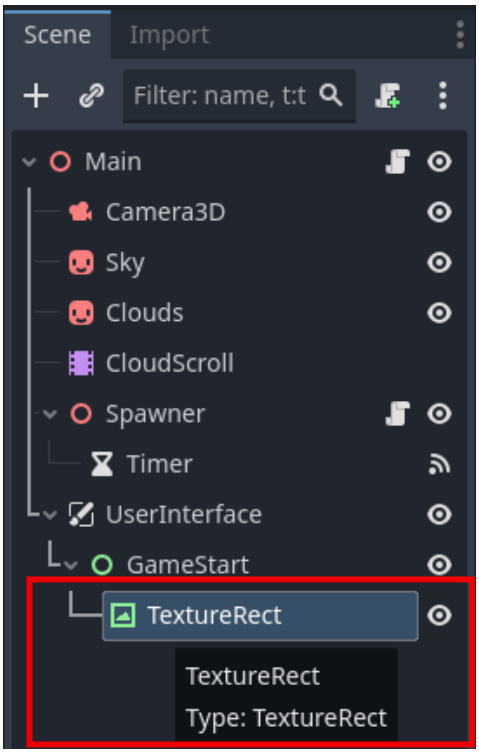
Reminder:

CanvasLayer nodes will override all other layers in relation to the camera so that they will appear at the front when the game runs. They must be edited in the **2D** workspace.

14 In **Scene**, add a new **Control** node as a child to the **UserInterface.** Rename it **GameStart**.



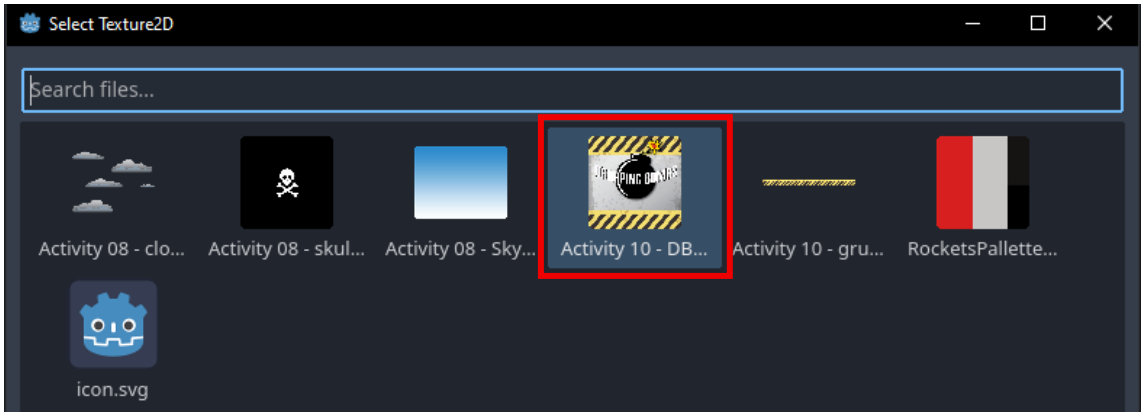
15 In **Scene**, add a new **TextureRect** node as a child to the **GameStart** node.

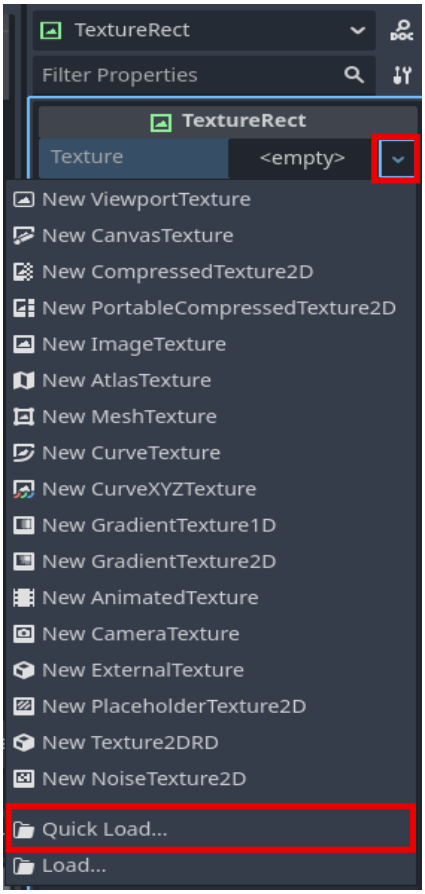


16 In **Scene**, select the **TextureRect** node.

In **Inspector**, click the dropdown to the right of **Texture**. From the dropdown menu, select **Quick Load**.

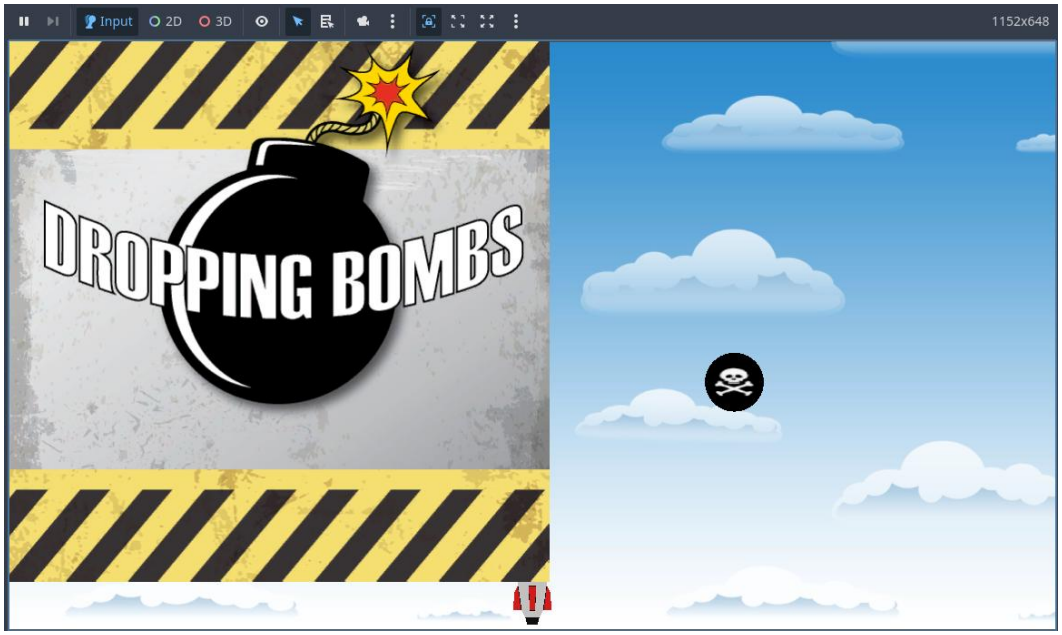
Select the **BB Activity 09 – DBTitle.png** to add the texture to the **TextureRect**.





17 Playtest the game.

Notice that in the **3D** editor, the **TextureRect** node is not visible, but can be seen in the playtest.



18 Select the **2D** workspace button at the top of the editor.

In **Scene**, select the **TextureRect** node.

In **Inspector,** under the **Layout** section, under **Transform,** set the **scale** to **1.2**.

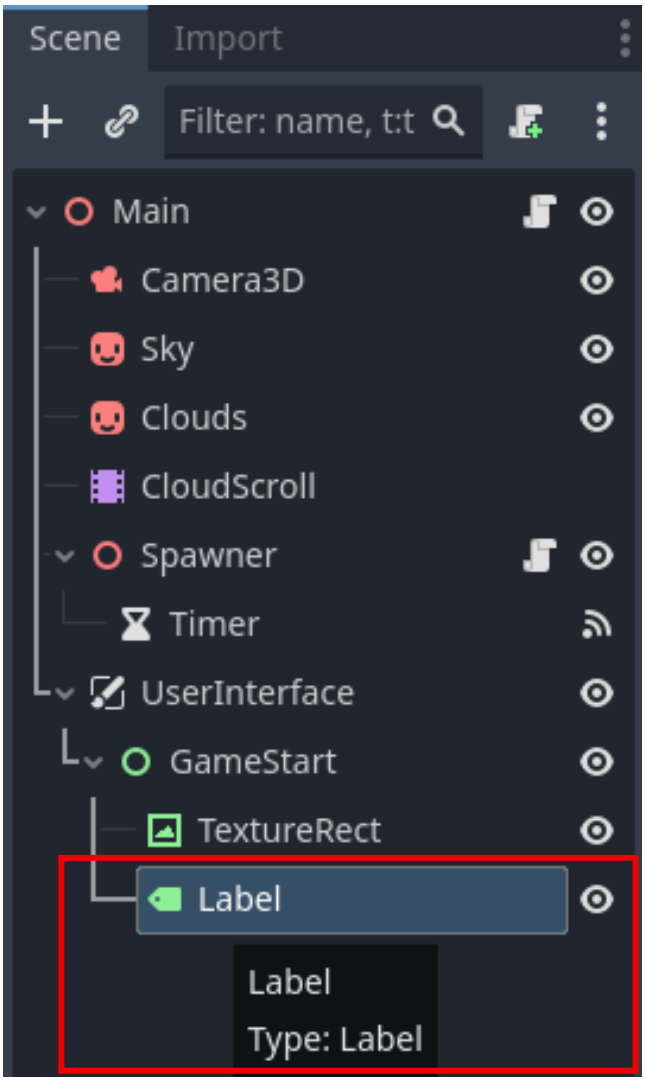
19 In the **2D** workspace, drag the Dropping Bombs title card into the center of the screen.

Playtest the game during this process to make sure the image shows up as intended!

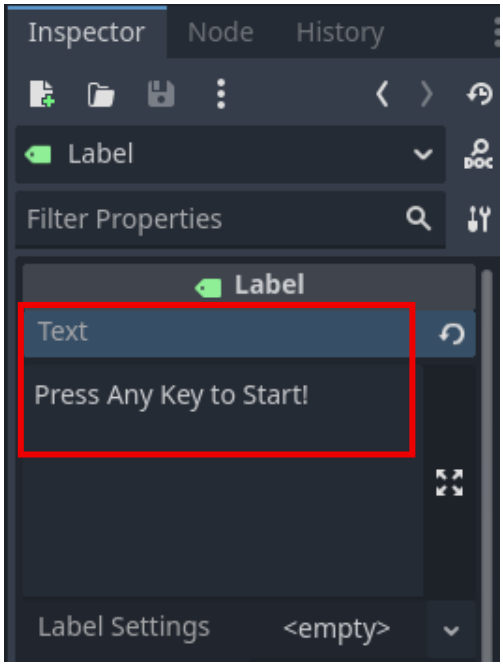
Reminder:

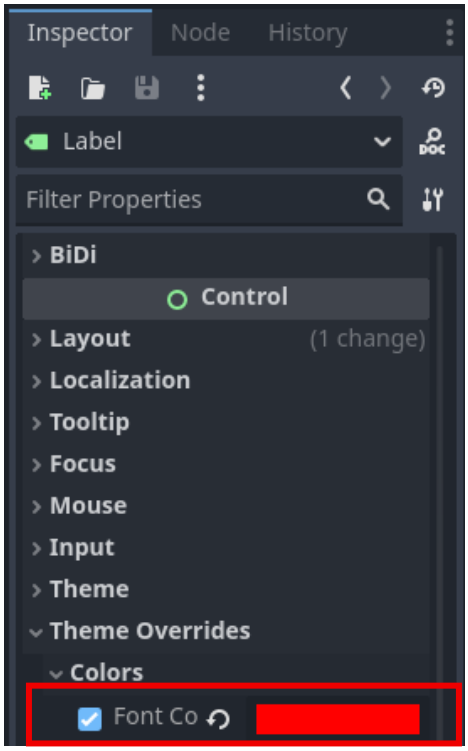
The blue outline in the 2D workspace indicates the edges of the screen.

20 In **Scene**, add a **Label** node as a child to **GameStart**.



21 In **Inspector**, set the **Text** property to: "**Press Any Key to Start!"**

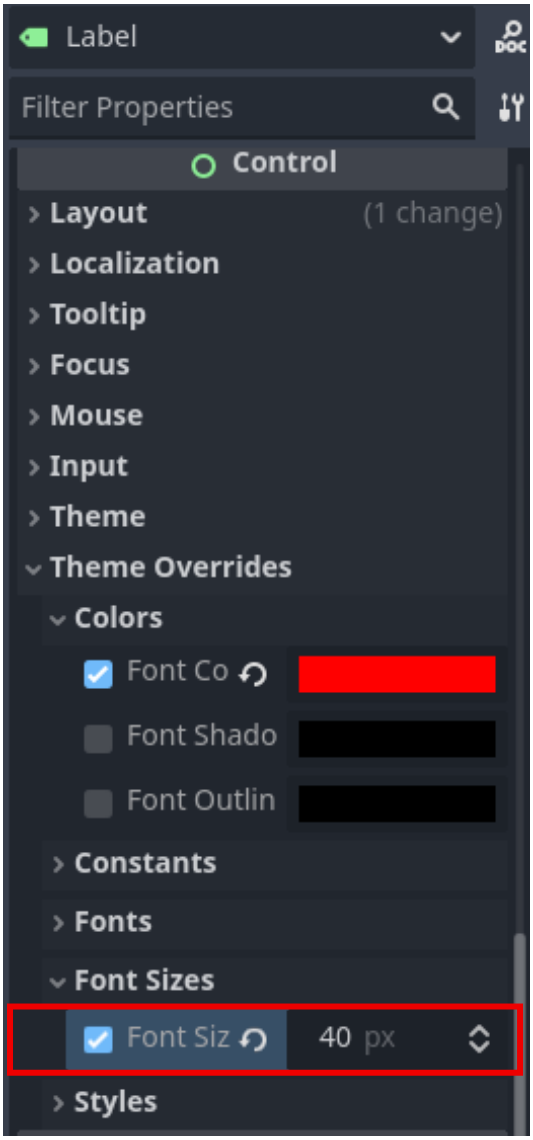




22 In **Inspector**, under **Control > Theme Overrides > Colors**, set the **Font Color**. Choose a color that will be visible against the Dropping Bombs title card, such as red.

23 In **Inspector**, set the **Font Size** to **40**. Ensure that the check box is also **checked on.**

In the **2D** workspace, drag the **Label** to be centered in the screen on top of the Dropping Bombs title card.



24 Playtest the game.

What happens when different keys are pressed - does the title card disappear? Are the bombs impacted by the title card?

Pause for **Sensei Stop #2**!

Before continuing, check with a Code Sensei and make sure the **GameStart** UI is set up properly.

Reminder: Save your work!

25

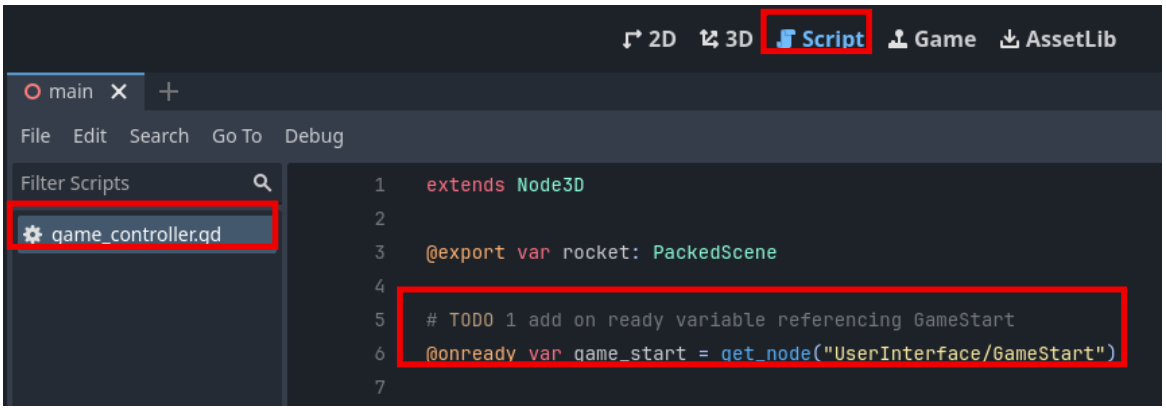
Edit the script so that the title card disappears when a key is pressed!

In the **Script** workspace, open the **game_controller** script.

Find the **TODO 1** comment in the script. On the next line, use the `@onready` keyword to declare a new variable named `game_start`.

Assign the variable the value returned from a call to `get_node()`. Set the parameter of `get_node()` to **`"UserInterface/GameStart"`**.

This line of code sets the variable `game_start` to be a reference to the **GameStart** node from the Scene.



26

Find the **TODO 2** comment in the script. On the next line, set `game_start.visible` to `true`.

This sets the Dropping Bombs Title Card to be visible when the game begins.

```
10 func _ready() -> void:
11     Globals.hit.connect(game_over)
12     #TODO 2 set visible to true
13     game_start.visible = true
```

27 Find the **TODO 3** comment in the script. On the next line, set `Globals.spawning` to `false`.

This prevents Bombs from spawning when the game begins.

```
10  func _ready() -> void:
11      Globals.hit.connect(game_over)
12      #TODO 2 set visible to true
13      game_start.visible = true
14      # TODO 3 set spawning to false
15      Globals.spawning = false
16      spawn_player()
```

28 Find the **TODO 4** comment in the script. On the next line, use the code completion to define a `_process` function.

Change the name of the parameter to `_delta`. This will prevent it from throwing a warning error later since this parameter will not be used, but is important to the function.

This function will run every frame.

```
17
18  # TODO 4 process function, if anything pressed visible = false
19  func _process(_delta: float) -> void:
```

29 Within the `_process` function, write an `if`-statement that checks if `Input.is_anything_pressed()` and `Globals.spawning` is `false`.

This if statement checks to see if any key is currently being pressed and if the Bombs are not yet spawning.

```
22  # TODO 4 process function, if anything pressed visible = false
23  func _process(_delta: float) -> void:
24      if Input.is_anything_pressed() and Globals.spawning == false:
25
```

30 Within the **if** statement from the previous step, set `game_start.visible` to `false`.

On the next line, set `Globals.spawning` to `true`.

```
21
22    # TODO 4 process function, if anything pressed visible = false
23    func _process(_delta: float) -> void:
24        if Input.is_anything_pressed() and Globals.spawning == false:
25
26
```

Pro Tip:

It is best practice to use **and** to check for two conditions in GDScript, instead of **&&** like in JavaScript. Use an assignment operator **==** to check for equality in GDScript too!

31 Check that the code matches the screenshot.

The first new line will set the Dropping Bombs Title Card to be invisible, allowing the game to be visible.

The second new line toggles the `Globals` variable that decides whether the Bombs should be spawning.

```
21
22  # TODO 4 process function, if anything pressed visible = false
23  func _process(_delta: float) -> void:
24      if Input.is_anything_pressed() and Globals.spawning == false:
25          game_start.visible = false
26          Globals.spawning = true
27
```

32 Find the `_ready()` function. Cut the `spawn_player()` statement. The completed function should look as pictured.

Find the `_process()` function. Paste the `spawn_player()` function at the end of the **if** statement.

This will spawn the player once the game begins, preventing accidental loss before gameplay begins.

```
14  func _ready() -> void:
15      Globals.hit.connect(game_over)
16      #TODO 2 set visible to true
17      game_start.visible = true
18      # TODO 3 set spawning to false
19      Globals.spawning = false
```

```
21
22  # TODO 4 process function, if anything pressed visible = false
23  func _process(_delta: float) -> void:
24      if Input.is_anything_pressed() and Globals.spawning == false:
25          game_start.visible = false
26          Globals.spawning = true
27          spawn_player()
28
```

33 Playtest the game.

Notice that the Dropping Bombs Title Card does disappear when keys are pressed, but Bombs still fall in the background. How does this impact the user's experience?

34 In the **Script** workspace, open the **spawner.gd** script. Find **TODO 5** in the script.

Underneath the comment, write an `if` statement that checks if `Globals.spawning` is true. Place the existing contents of the `_on_timer_timeout()` function inside. Ensure that the `if` statement is indented.

This will prevent any Bombs from spawning before the setting from Globals is toggled.

35 Playtest the game.

The game should not have any visible Bombs falling in the background.

Pause for **Sensei Stop #3**!

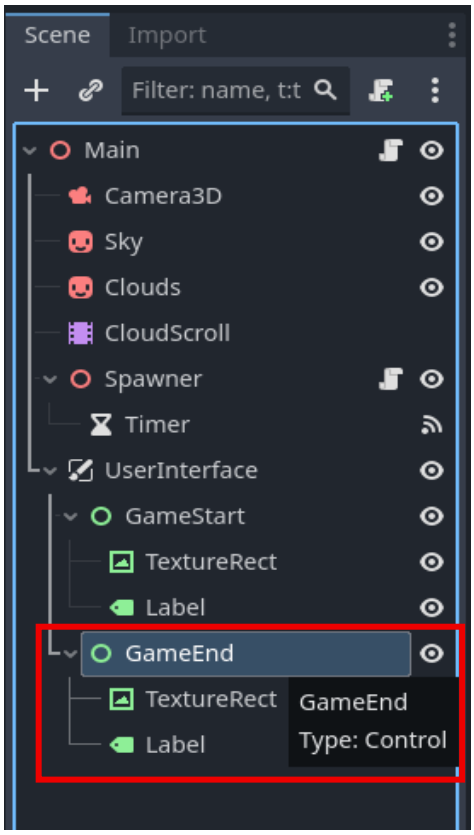
Before continuing, check with a Code Sensei and make sure the **game_controller** and **spawner** scripts are set up properly.

Reminder: Save your work!

36

Create the game end UI.

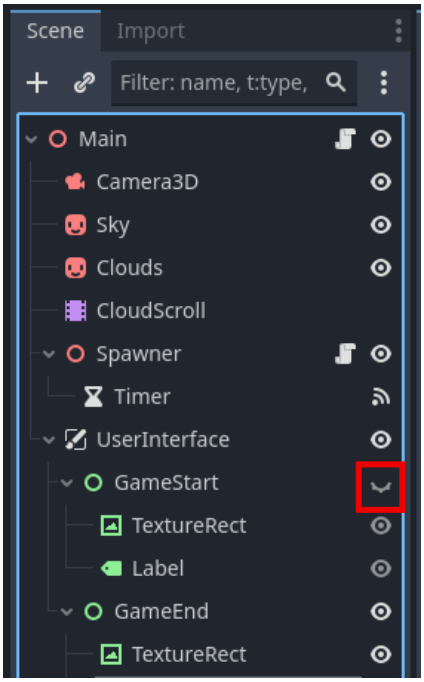
In **Scene**, right click on the **GameStart** node and select **Duplicate** from the dropdown menu. Name the duplicated node **GameEnd**.



Pro Tip:

Press **CTRL + D** on the keyboard while a node is selected to quickly duplicate it!

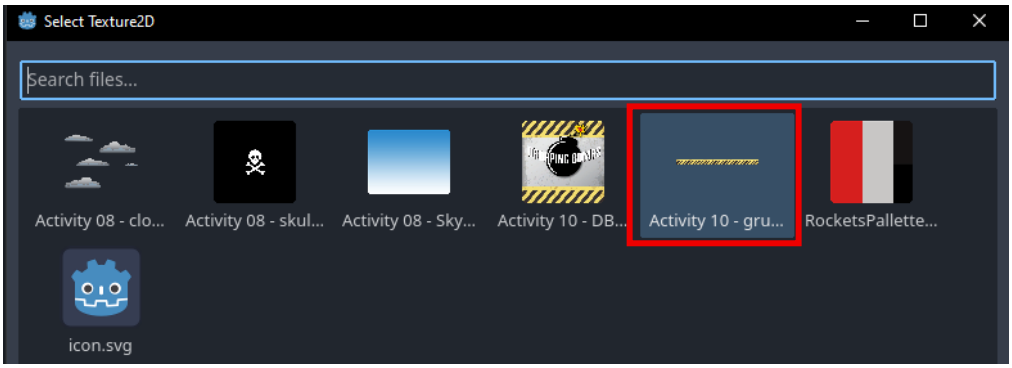
37 In **Scene**, click the **Open Eye** next to the **GameStart** node to toggle its visibility.



38 In **Scene**, select the **TextureRect** child node of the **GameEnd** node.

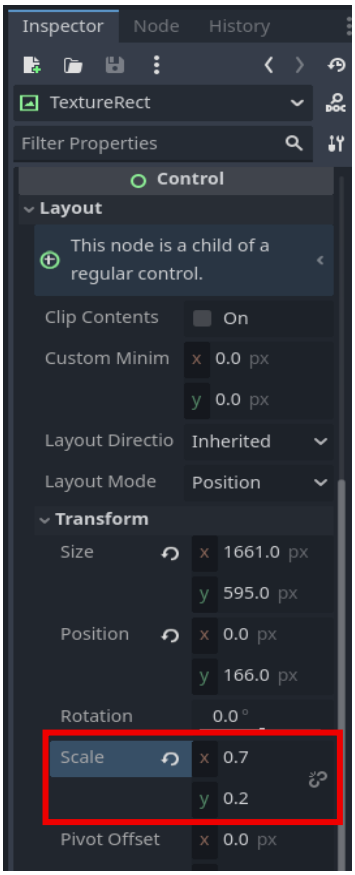
In **Inspector**, click the dropdown to the right of **Texture** and select **Quick Load**.

Select the **BB Activity 10 – grungeHazard.png** to add the texture to the **TextureRect**.

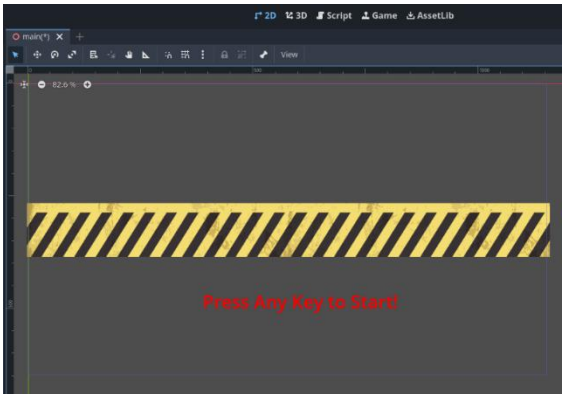


39 In **Inspector,** under the **Layout** section, under **Transform,** click the **Chain Icon** to the right of **scale** to unlock the component ratio.

Set the **x scale** to **0.7**, and the **y scale** to **0.2**.



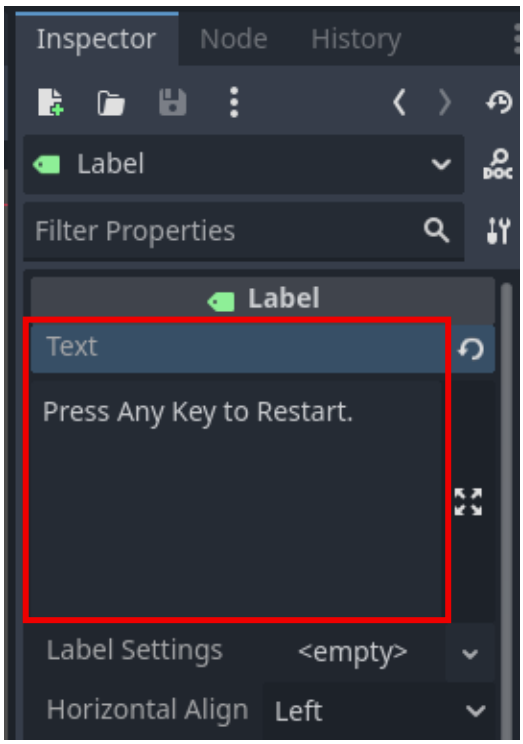
40 Select the **2D** workspace button at the top of the editor.



Drag the Hazard texture around in the workspace so that it is centered.

41 In **Scene**, select the **Label** child node of **GameEnd**.

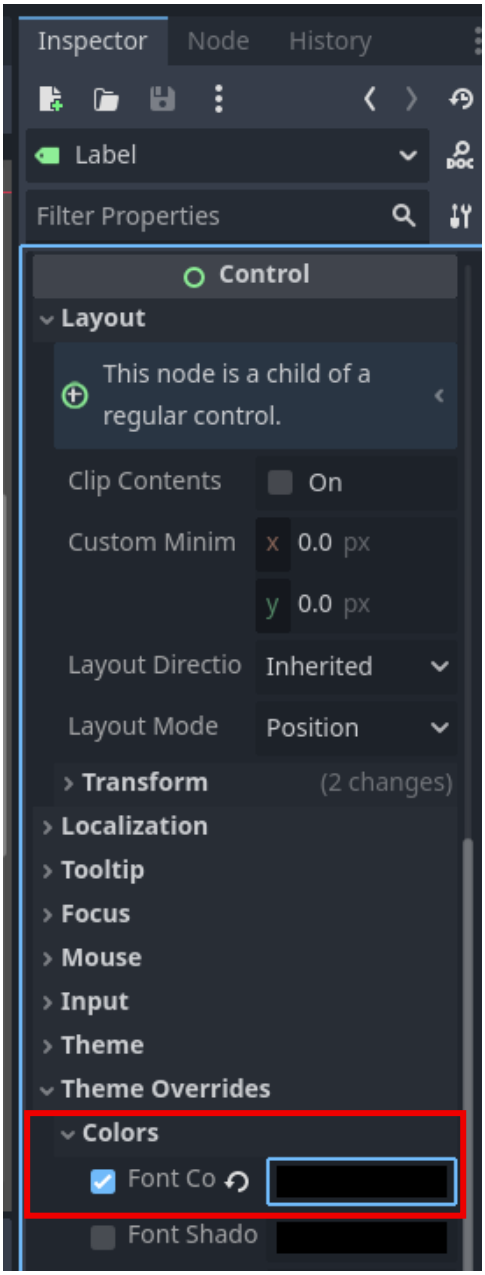
In **Inspector**, set the **Text** property to "**Press Any Key to Restart.**"



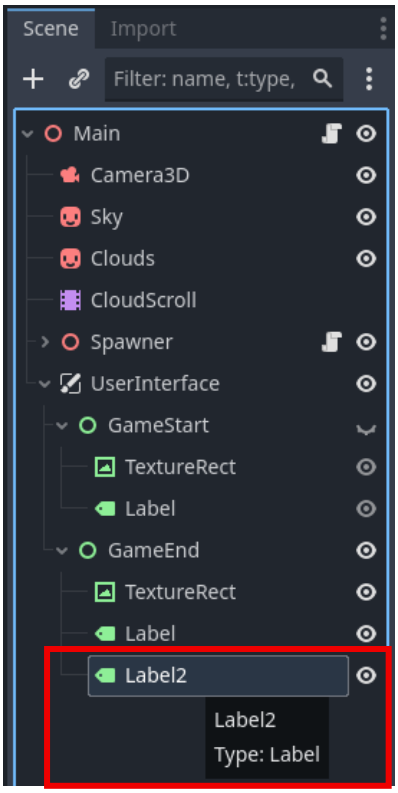
Pro Tip:

Selecting the label in the 2D workspace will also bring up its inspector!

42 In **Inspector**, under **Control > Theme Overrides > Colors**, set the **Font Color** to **black** so it shows up against the background.

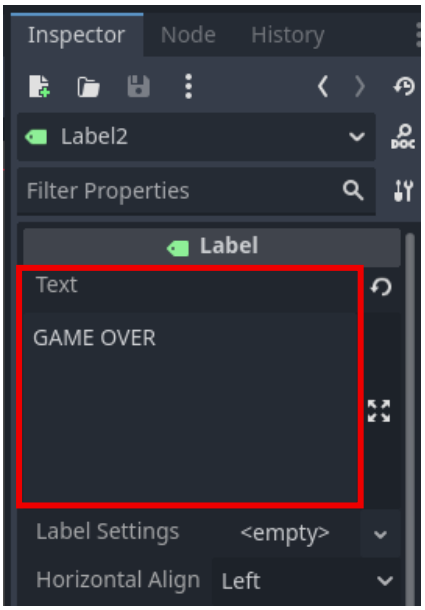


43 In **Scene**, duplicate the **Label** node child to **GameEnd**.

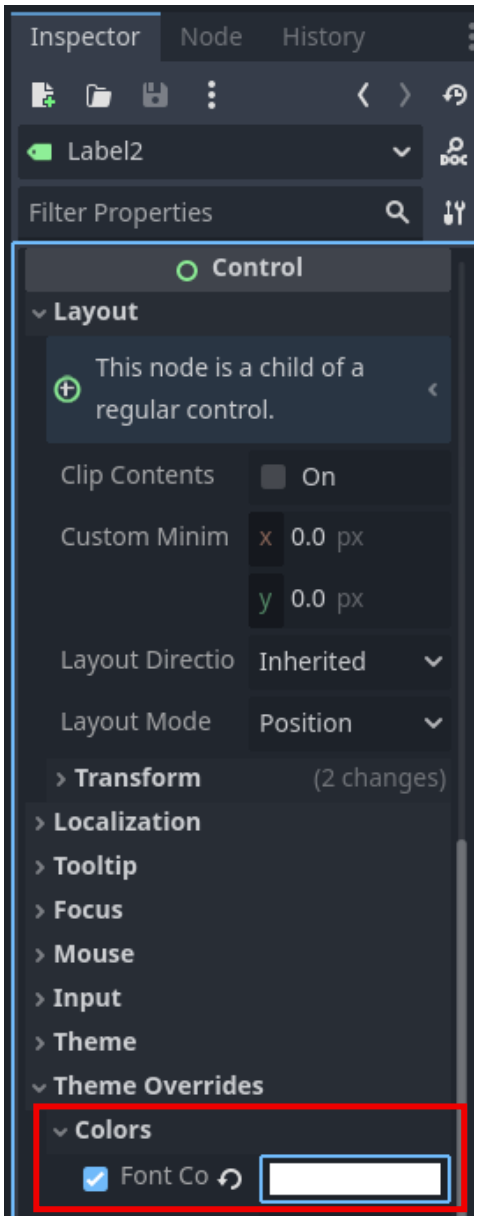


44 In **Scene**, select **Label2**.

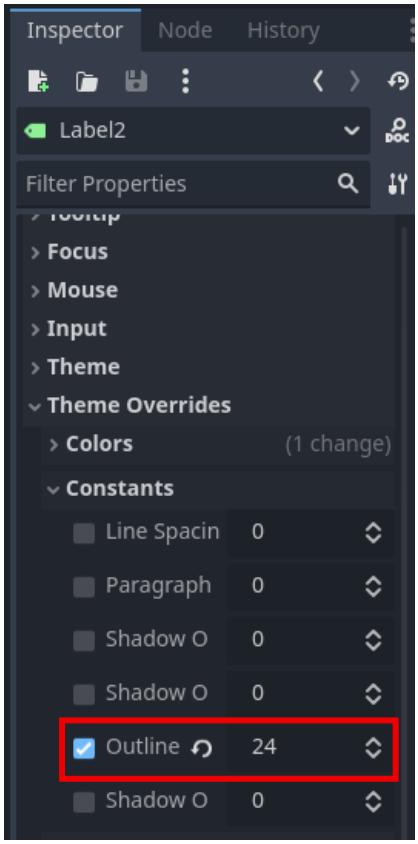
In **Inspector**, set the **Text** property to "**GAME OVER.**"



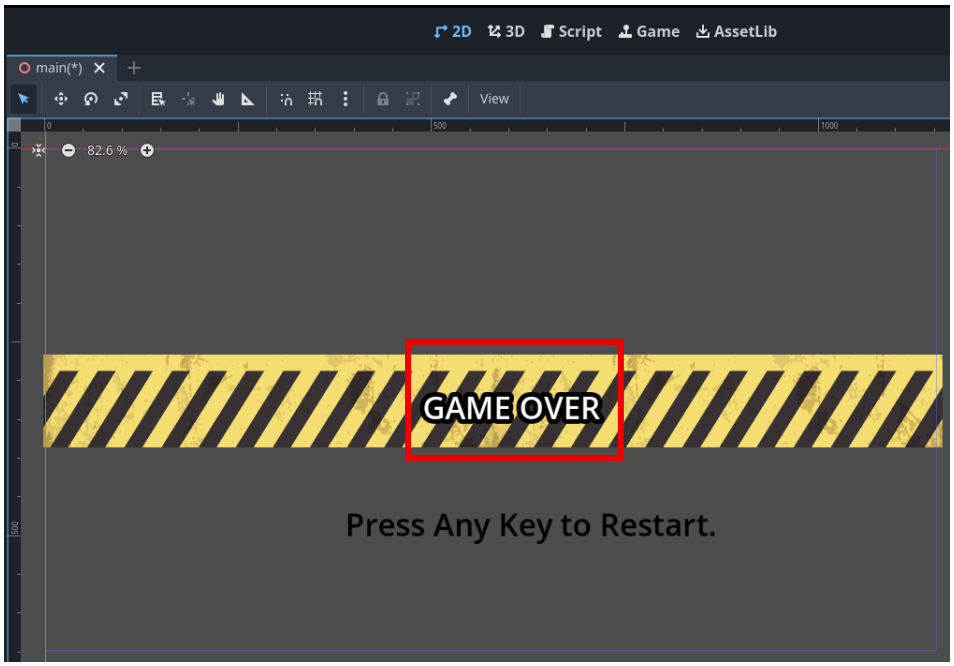
45 In **Inspector**, under **Control > Theme Overrides > Colors**, set the **Font Color** to **white.**



46 In **Inspector**, under **Control > Theme Overrides > Constants**, set the **Outline** to **24.**

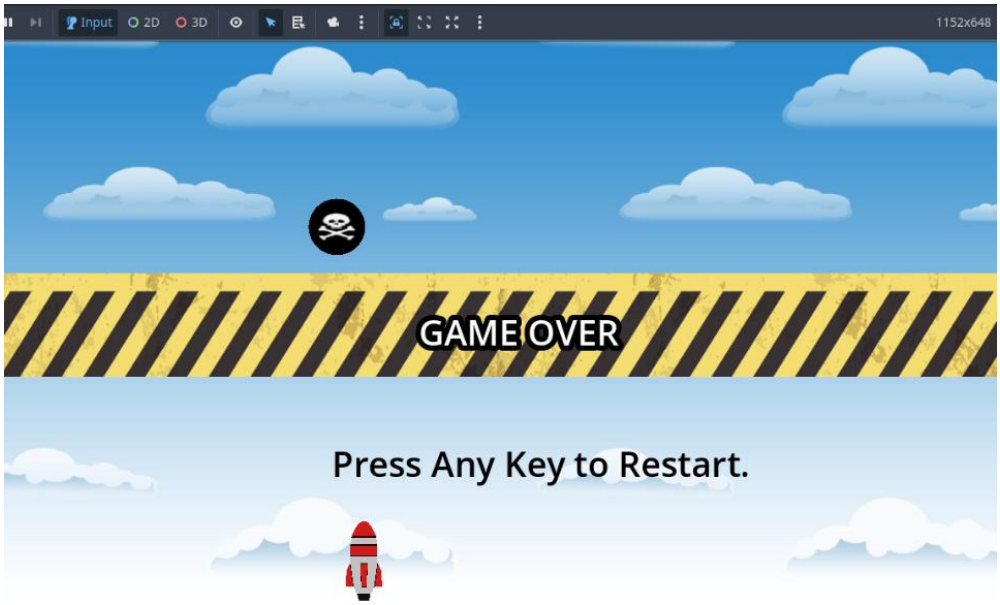


47 In the **2D** workspace, drag the "GAME OVER" text to be centered over the Hazard texture.



48 Playtest the game.

What happens to **GameStart** UI when a key is pressed? What about the **GameEnd** UI?



Pause for **Sensei Stop #4**!

Before continuing, check with a Code Sensei and make sure the **GameEnd** UI is set up properly.

Reminder: Save your work!

49

Edit the script so the GameEnd UI appears at the end of the game.

In the **Script** workspace, open the `game_controller.gd` script.

Find **TODO 6** in the script. On the next line, use the `@onready` keyword to declare a `game_end` variable.

Use the `game_start` variable declaration as a guide to assign `game_end` the value of the **GameEnd** node.

This line of code sets the variable `game_end` to be a reference to the **GameEnd** node from the Scene.

50

At the end of the `_ready()` function, set `game_end.visible` to be `false`.

```
15 func _ready() -> void:
16     Globals.hit.connect(game_over)
17     # TODO 2 set visible to true
18     game_start.visible = true
19     # TODO 3 set spawning to false
20     Globals.spawning = false
21
```

51 Check that the code matches the screenshot.

This will make the Game End UI invisible when the game starts.

```
15  func _ready() -> void:
16      Globals.hit.connect(game_over)
17      # TODO 2 set visible to true
18      game_start.visible = true
19      # TODO 3 set spawning to false
20      Globals.spawning = false
21      game_end.visible = false
```

52 Find **TODO 7** in the script. Delete the line above it that reloads the current scene.

```
48
49  func game_over():
50
51      # TODO 7 stop spawning bombs, show game end UI
52
```

53 On the lines after the **TODO 7** comment, add the following code:

- Set `game_end.visible` to `true`.
- Set `Globals.started` to `true`.
- Set `Globals.spawning` to `false`.

```
30  func game_over():
31      # TODO 7 stop spawning bombs, show game end UI
32
33
34
```

54 Check that the code matches the screenshot.

Now, on game over, the Game End UI will be displayed, the Global variable will track that the game end sequence has begun, and the Bombs will stop spawning.

```
30 func game_over():
31     # TODO 7 stop spawning bombs, show game end UI
32     game_end.visible = true
33     Globals.started = true
34     Globals.spawning = false
35
```

55 On the next line, queue the **player** to be deleted at the end of this frame with the `queue_free()` method.

This will put the Rocket node into a queue to be deleted when the frame finishes processing.

```
30 func game_over():
31     # TODO 7 stop spawning bombs, show game end UI
32     game_end.visible = true
33     Globals.started = true
34     Globals.spawning = false
35     player.queue_free()
36
```

56 Find **TODO 8** in the script. On the next line, define a custom `restart()` function that takes no parameters.

```
36
37 # TODO 8 restart the game
38 func restart():
39
```

57 On the lines below **TODO 8**, add the following code:

- Set `game_end.visible` to `false`.
- Set `Globals.started` to `false`.
- Set `Globals.spawning` to `true`.
- Call the `spawn_player()` function.

Read the code that was just written. What does it do?

```
27
28    # TODO 8 restart the game
29    func restart():
30
31
32
33
34
```

58 Check that the code matches the screenshot.

This will make the Game End UI invisible. This toggles the Global variable that tracks whether the game end sequence has begun. This will allow Bombs to begin dropping again.

```
36
37    # TODO 8 restart the game
38    func restart():
39        game_end.visible = false
40        Globals.started = false
41        Globals.spawning = true
42        spawn_player()
43
```

59 Find the `_process()` function.

Inside the `if` statement, nest another `if` statement that checks whether `Globals.started` is true.

Place the rest of the statements into an `else` statement. Ensure that there is a colon after the `if` and `else` statements, and that both are properly indented.

Notice that the **if** statement is red in the image below. This is because there is no content in the **if** statement; this will be fixed in the next step.

```
22
23    # TODO 4 process function, if anything pressed visible = false
24    func _process(_delta: float) -> void:
25        if Input.is_anything_pressed() and Globals.spawning == false:
26            if (Globals.started):
27                
28            else:
29                game_start.visible = false
30                Globals.spawning = true
31                spawn_player()
32
```

Pro Tip:

When checking if a Boolean is true, instead of checking equality like "`if (variable == true)`," conditional statements can be written as "`if (variable)`." Since a Boolean can only be true or false, the **if** statement simply checks what the value of the Boolean variable is!

60 Inside `if (Globals.started)`, add an `await` keyword. Then, chain together a call to the method `get_tree()`, a call to the method `.create_timer()` with a value of `1`, and a signal to `.timeout`.

This code forces the game to wait a second before proceeding.

```
22
23    # TODO 4 process function, if anything pressed visible = false
24    func _process(_delta: float) -> void:
25        if Input.is_anything_pressed() and Globals.spawning == false:
26            if Globals.started:
27                await get_tree().create_timer(1).timeout
28            else:
29                game_start.visible = false
30                Globals.spawning = true
31                spawn_player()
32
```

61 On the next line, call the `restart()` function.

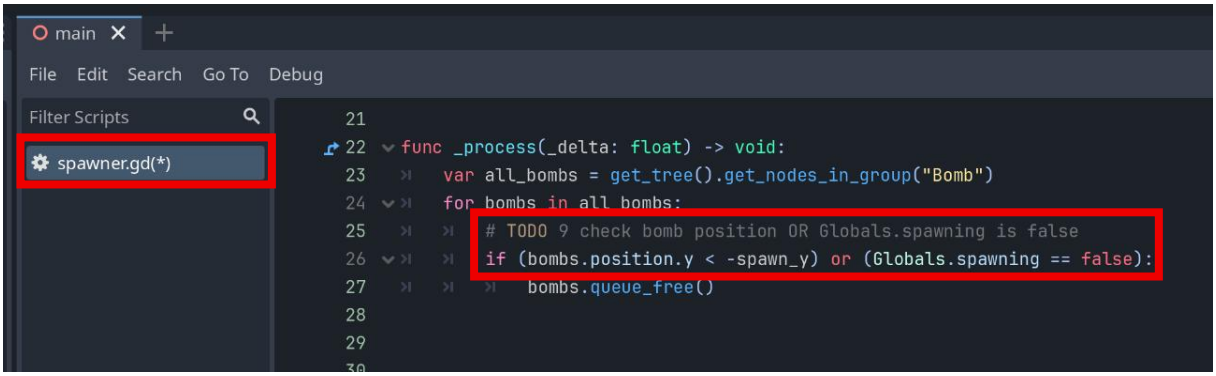
The game will restart if the game has ended and half a second has elapsed.

```
22
23    # TODO 4 process function, if anything pressed visible = false
24    func _process(_delta: float) -> void:
25        if Input.is_anything_pressed() and Globals.spawning == false:
26            if Globals.started:
27                await get_tree().create_timer(1).timeout
28                restart()
29            else:
30                game_start.visible = false
31                Globals.spawning = true
32                spawn_player()
33
```

62 In the **Script** workspace, open the **spawner.gd** script.

Find **TODO 9**. On the next line, edit the `if` statement to check if the `bomb's y position` is less than `-spawn_y` **OR** if `Globals.spawning` is `false`.

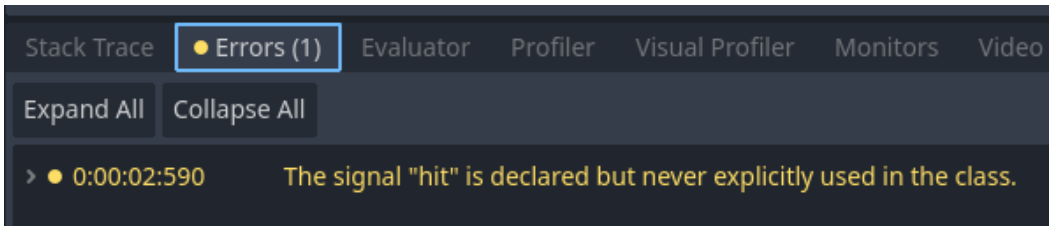
This will delete bombs actively in the scene if the Globals variable that decides whether bombs spawn is set to false.



Pro Tip:

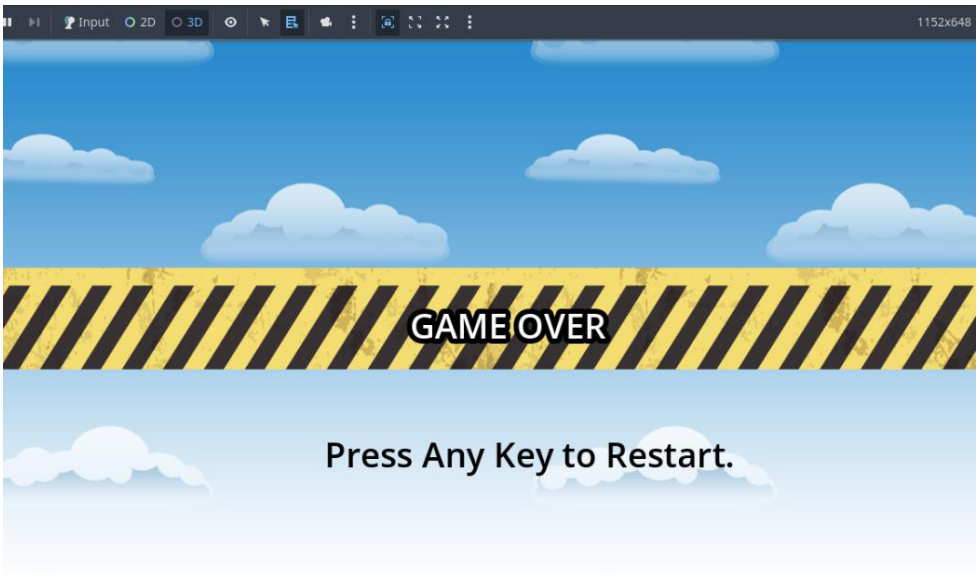
It is best practice to use **or** to check for two conditions in GDScript, instead of **||** like in JS.

63 **Note**: There will be a warning that signal **hit** is declared but never explicitly used. That is alright and can be ignored.



64 Playtest the game.

UI should appear at the beginning and end of the project, bombs should only spawn while the game is being played, and the game should restart without showing the title card again after losing.



Pause for **Sensei Stop #5**!

Congratulations on creating your first game with UI in Godot! Great job!

Before submitting, check in with a Code Sensei to make sure the UI and cycling game statuses work then reflect on the following:

- What did you learn about UI? Globals variables?
- What did you enjoy most when creating this project?
- What was something you found difficult and why?

Reminder: Save your work!

Congratulations on completing **BB Activity 10: Dropping Bombs Part 4** and in Godot – **You Rock**! You are now ready to save this project and submit it.

Continue your exploration with Godot by opening the **BB Activity 11: Dropping Bombs Part 5** Ninja Guide.