CODE NINJAS®

Bronze Belt Ninja Guide

Activity 11: Dropping Bombs Part 5

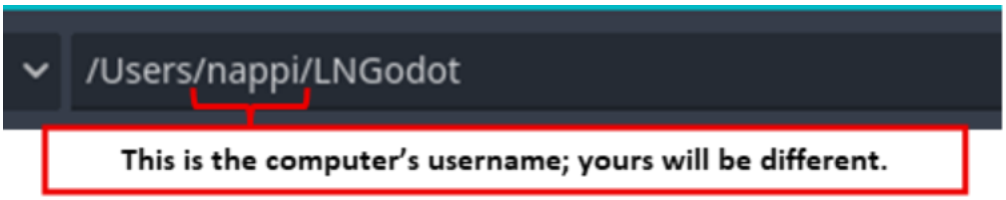
Activity 11: Dropping Bombs Part 5

In this project, you will learn more about creating a useful User Interface (UI) by adding important elements to Dropping Bombs – the score counter, and the high score. These are important pieces of information for the player that show how well they are playing the game.

In this activity, you will explore more applications of label nodes such as updating them during a game, and appropriate use of global variables vs. local variables. By the end of the activity, the project will keep track of the player's current score while playing and show the highest score a player has reached during a session.

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

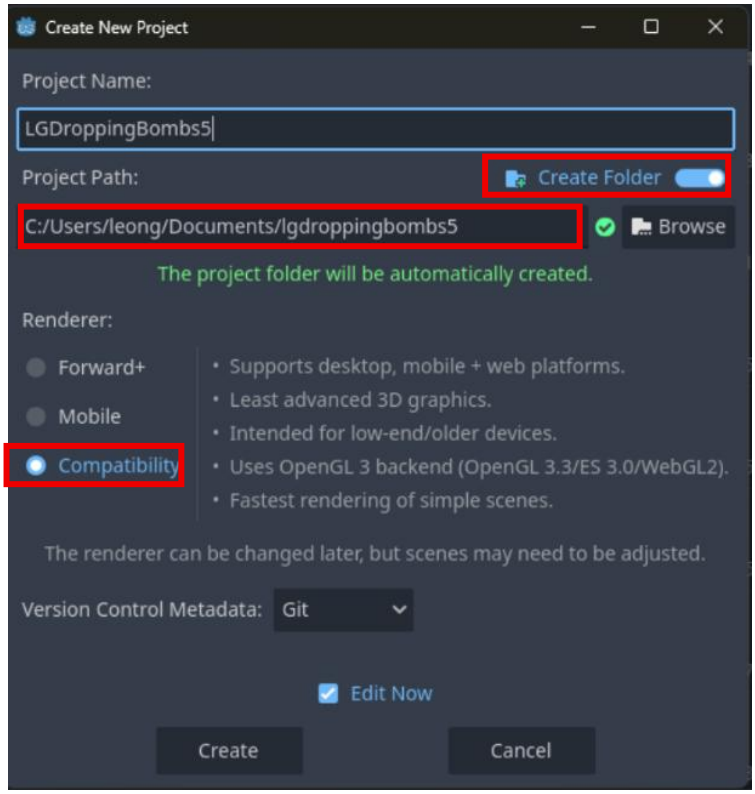
Don't worry if your path looks slightly different from the image shown! All computers have their own username.



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

A Create New Project window will pop up. Name the project **[MyInitials]DroppingBombs5**.

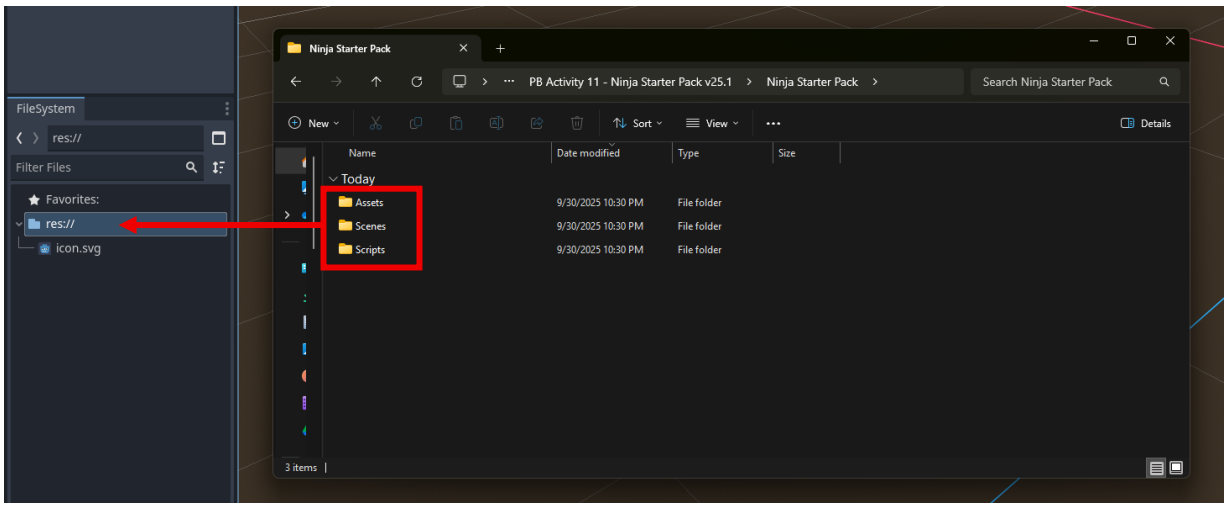
Check that **Create Folder** is turned on, and that the **Compatibility** mode for the renderer is being used. Then click **Create**.



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

Extract **BB Activity 11 – 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.

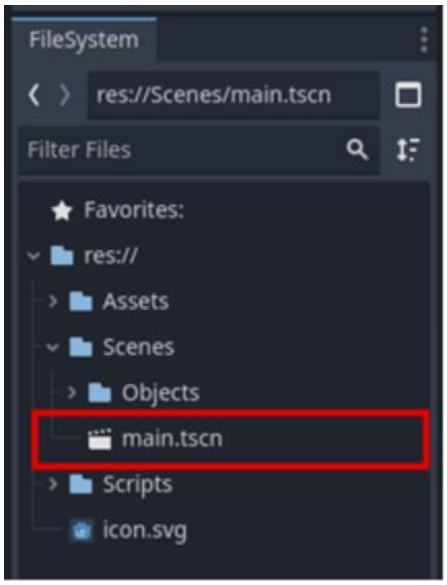


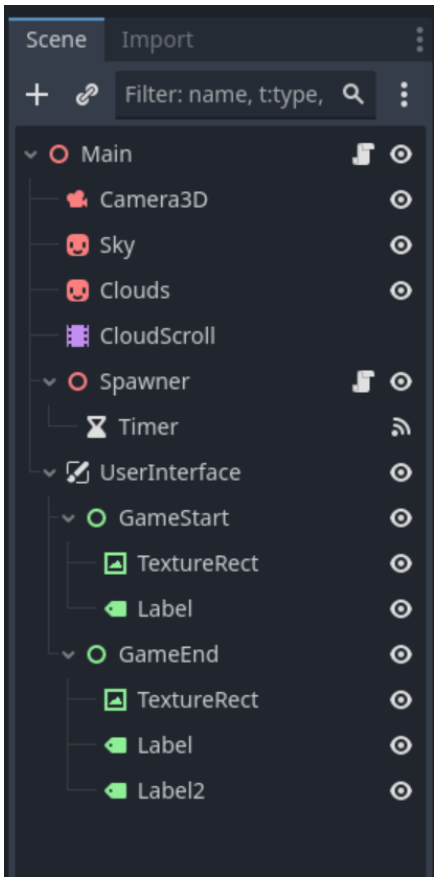
Reminder:

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 that the main scene already has the camera, sky, and clouds created in Part 3, and the **GameStart** and **GameEnd** nodes created in Part 4.

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.



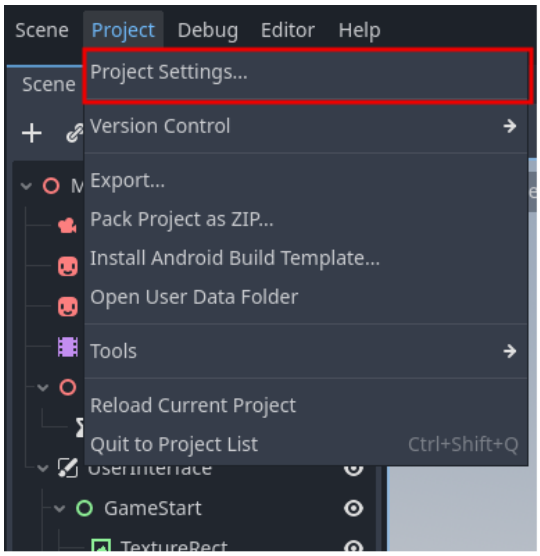


Reminder:

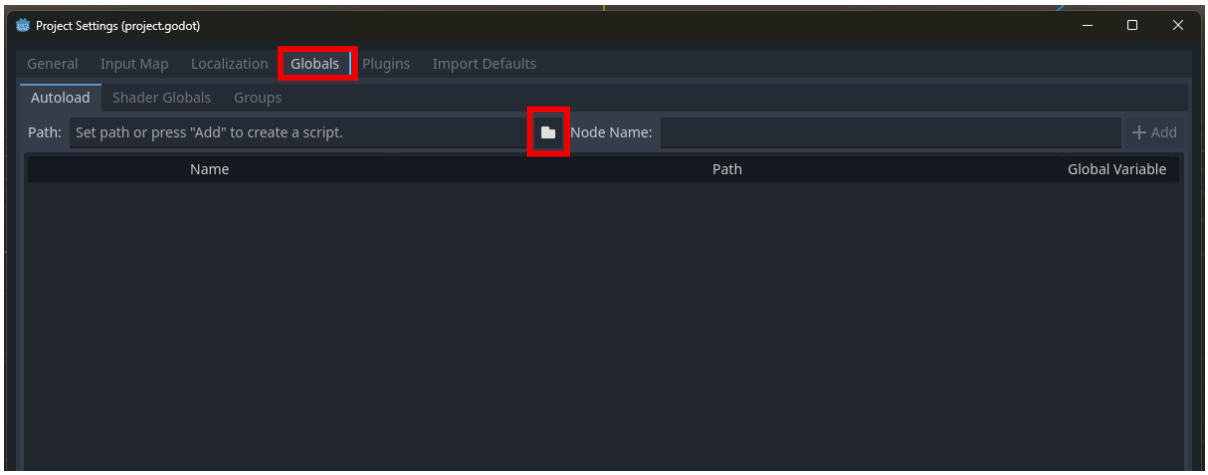
Click the arrows next to the folders to open them.

5 Add the **Globals** script to the project.

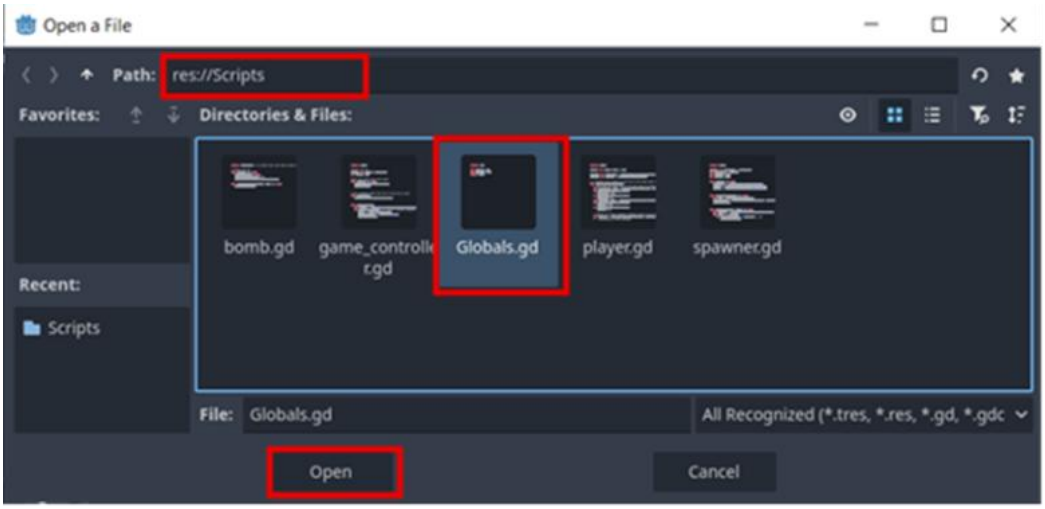
In the top left corner of the editor, select **Project**. Select **Project Settings** from the dropdown menu.



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

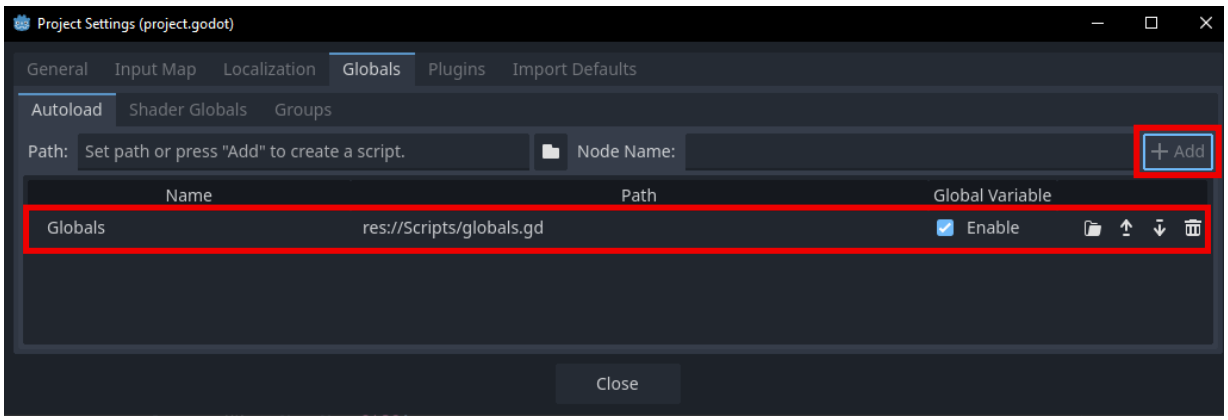


7 In the new popup window, open the **Scripts** folder. Select the **Globals.gd** file and click Open.



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

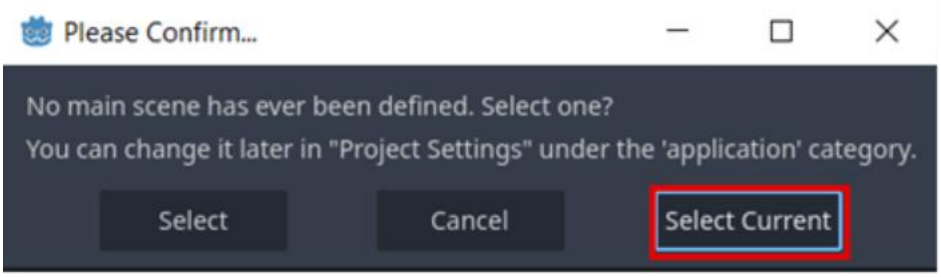
Close the **Project Settings** window.



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

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

Notice that the game is the same as where Part 4 left off. Close the playtest window.



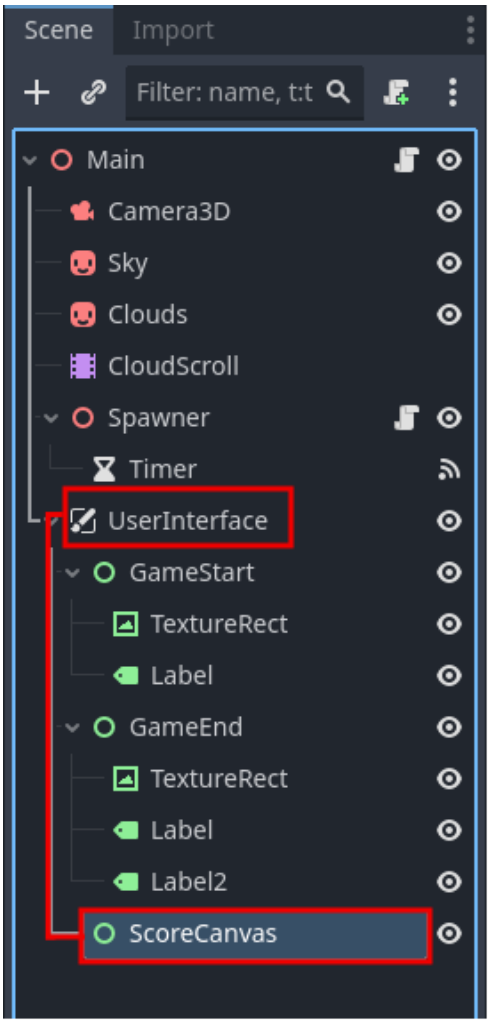
Pause for **Sensei Stop #1**!

Check with a Code Sensei to make sure the game files have been imported correctly, and that the **global variables** have been set.

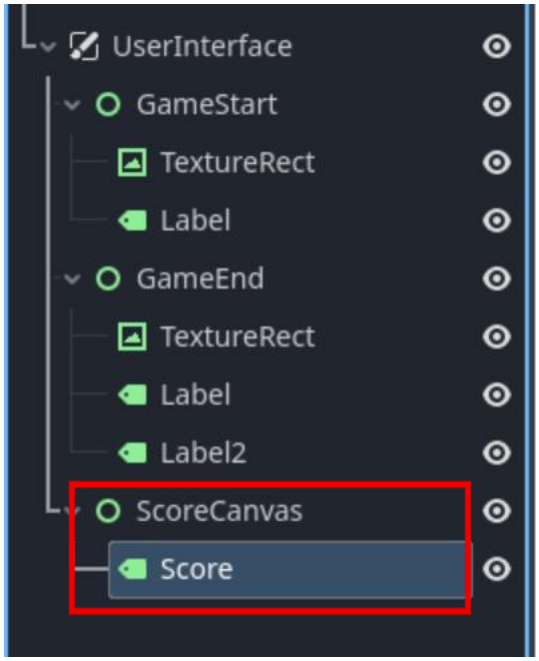
Reminder: Save your work!

10 Add labels for the player's current score and highest score.

Add a Control node as a child to the **UserInterface CanvasLayer** node and rename it **ScoreCanvas**.

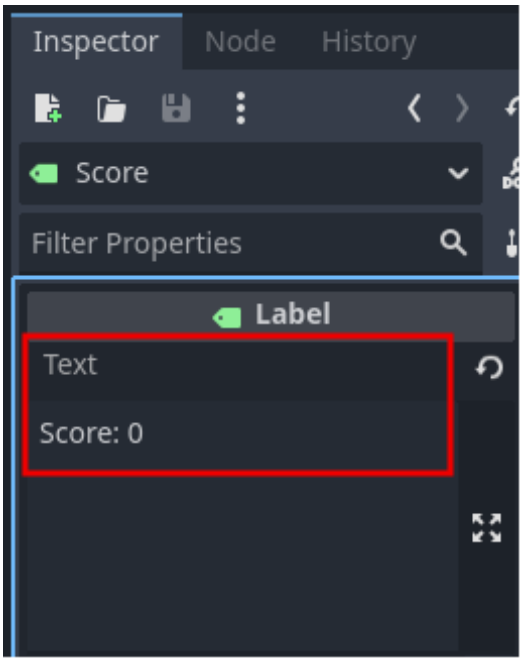


11 Add a **Label** node as a child of **ScoreCanvas**, and rename it **Score**.



12 Update the visual display of the score text on screen.

In the **Inspector** for **Score**, set the text property to "Score: 0".



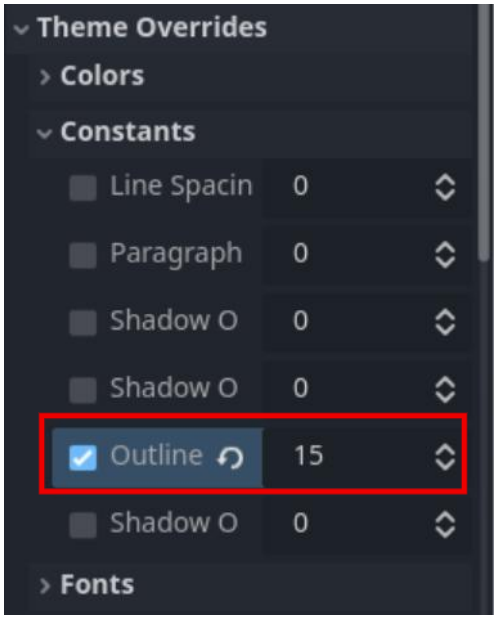
13 Open the **Theme Overrides** section of the Inspector, then find the **Font Sizes** submenu.

Check the box next to **Font Size** and set the font size to 36 px.

Pro Tip:

Changing the font size this way will also automatically resize the label container to fit the text.

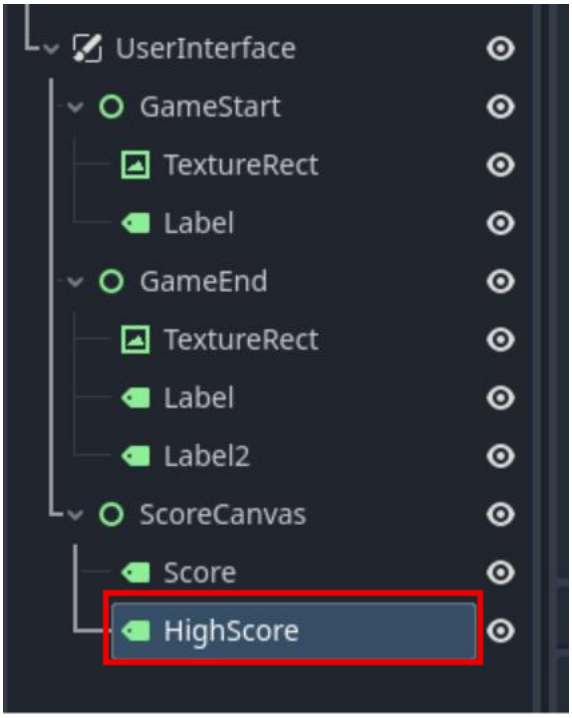
14 Under **Theme Overrides**, find the **Constants** submenu. Check the box next to **Outline** and set the text outline to **15**.



This will make the white text of the **Score** label easier to read against the game's light blue sky background.

15 Add a new label that shows the player's high score.

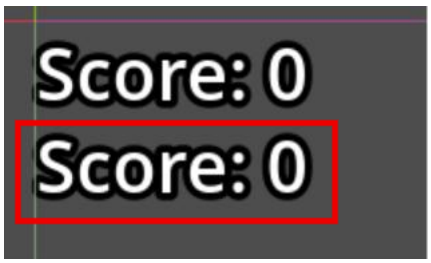
Duplicate the **Score** label and rename it **HighScore**.



Reminder:

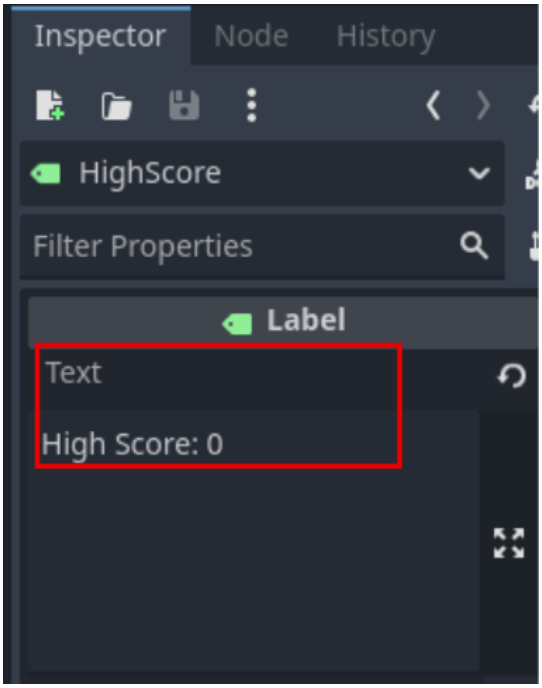
The shortcut to duplicate a node, **Ctrl+D**, will automatically create the new node at the same level in the hierarchy as the original.

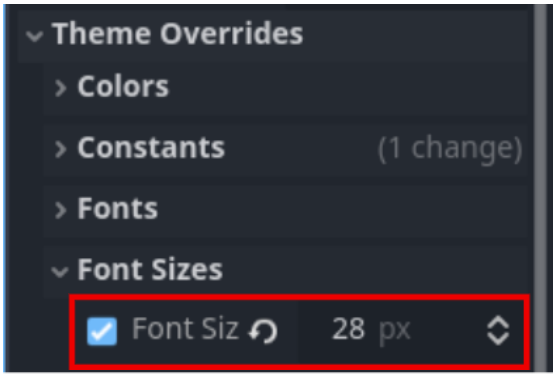
16 In the **2D workspace**, reposition the **HighScore** label directly below the **Score** label.

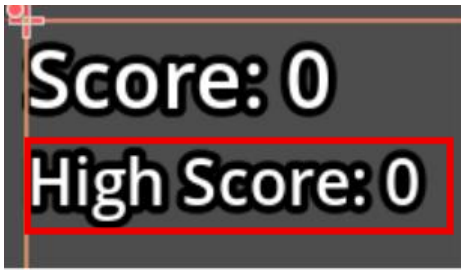


In the Inspector for the HighScore label node, repeat steps 12-14.

Update the text to "High Score: 0" and the font size to 28.





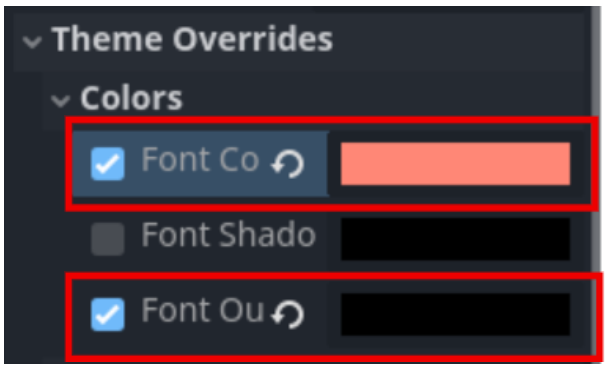


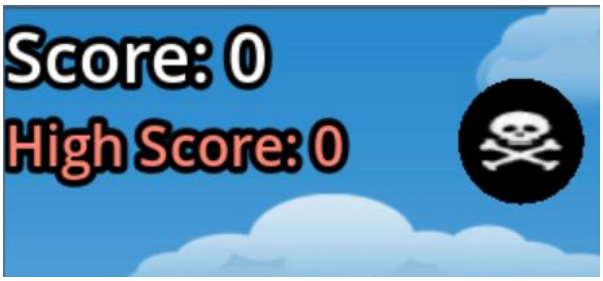
17 Under **Theme Overrides**, find the **Colors** submenu.

Tinker with the color of the **HighScore** label to make it more distinctive.

A color like red or orange would stand out well on the blue background but feel free to try some different options.

Make sure the **Font Outline** box is checked as well.





Pro Tip:

Playtest the project to see how the text appears on top of the background, then adjust the font color as needed.

Pause for **Sensei Stop #2**!

Check with a Code Sensei that the **Score** and **HighScore** labels are in the correct places in the hierarchy, and that they are readable while the game is running.

Reminder: Save your work!

18

Create a variable for the **Score** label.

Open the **game_controller** script attached to Main.

Under **TODO #1**, declare an `@onready` `score_label` variable. Assign it the value of the **Score** label node.

```
1   extends Node3D
2
3   @export var rocket: PackedScene
4   @onready var game_start = get_node("UserInterface/GameStart")
5   @onready var game_end = get_node("UserInterface/GameEnd")
6   var player: Node
7   # TODO #1: Create variable for score label.
8   @onready var score_label = get_node("UserInterface/ScoreCanvas/Score")
9   #TODO #9: Create variable for high score and best score label.
10
```

19

Create a variable that will store the player's score throughout the game.

In FileSystem, open **globals.gd**.

Under **TODO #2**, declare a `score` variable of type integer.

```
1   extends Node
2
3   var started: bool
4   var spawning: bool
5
6   # TODO #2: Create global score variable.
7   var score: int
8
```

Why is this variable being declared in **globals** instead of in **game_controller**?

20 Add a signal to trigger the score being changed.

Under **TODO #3**, declare a `score_changed` signal. Pass a `score` parameter inside the parentheses, of type `integer`.

```
8
9   signal hit()
10  # TODO #3: Create signal to signal when score has been changed.
11  signal score_changed(score: int)
12
```

21 Create a function to update the player's score and send the `score_changed` signal.

Under **TODO #4**, define a `change_score` function with a `value` parameter of type integer.

```
13
14  # TODO #4: Create global function to increase score
15  func change_score(value: int):
16      
17
```

22 Inside the `change_score` function, add the `value` parameter to the `score` variable.

```
13
14  # TODO #4: Create global function to increase score
15  func change_score(value: int):
16      score += value
17      
18
```

23 On the next line, call the emit method to send the `score_changed` signal, using the **score** variable as the parameter.

When this function is called, the number passed by the `value` parameter will be added to the score, and the `score_changed` signal will be sent out.

```
13
14    # TODO #4: Create global function to increase score
15    func change_score(value: int):
16        score += value
17        score_changed.emit(score)
18
```

24 Set the score when the game launches and connect the `score_changed` signal to a function that will update the score.

In **game_controller.gd**, find **TODO #5**. Underneath, set the global `score` variable to `0`.

```
11
12    func _ready() -> void:
13        Globals.hit.connect(game_over)
14        game_start.visible = true
15        Globals.spawning = false
16        game_end.visible = false
17        # TODO #5: Set score to 0 at start of game and connect signal.
18        Globals.score = 0
19
```

25 After the `score` variable is set, connect the `score_changed` signal from **Globals** to an `update_score` function.

```
11
12 func _ready() -> void:
13     Globals.hit.connect(game_over)
14     game_start.visible = true
15     Globals.spawning = false
16     game_end.visible = false
17     # TODO #5: Set score to 0 at start of game and connect signal.
18     Globals.score = 0
19     Globals.score_changed.connect(update_score)
```

Pause for **Sensei Stop #3**!

Check with a Code Sensei before moving on. Make sure the code to update the **score** label on screen was created correctly.

Reminder: Save your work!

26

Create the `update_score` function.

In **game_controller.gd**, find **TODO #6**. Underneath, declare an `update_score` function that takes a `score` parameter, of type `integer`. This function should **not** return a value.

```
60
61    # TODO #6: Update score label.
62    func update_score(score: int) -> void:
63        pass
64
```

27

Inside the function, update the `score_label` text to read `"Score: "` concatenated with the current amount of points the player has.

```
60
61    # TODO #6: Update score label.
62    func update_score(score: int) -> void:
63        score_label.text = "Score: " + str(score)
64
```

Reminder:

Strings can only be added to other strings, and variables can be recast to other types.

28

Add to the current score when a bomb is destroyed.

In **spawner.gd**, find the **TODO #7** comment.

Notice that the `_process` function's `if` statement checks if the global variable `spawning` is `false`, so that bombs don't spawn when the player hasn't started the game. The player should only get points if the game is still running, and bombs are still spawning.

```
20
21  func _process(_delta: float) -> void:
22      var all_bombs = get_tree().get_nodes_in_group("Bomb")
23      for bombs in all_bombs:
24          if (bombs.position.y < -spawn_y) or (Globals.spawning == false):
25              bombs.queue_free()
26              # TODO #7: Change score only when bombs are destroyed during gameplay.
```

Inside the `if` statement and after the bomb node is destroyed, add another `if` statement that checks if `spawning` is set to **true**.

```
21  func _process(_delta: float) -> void:
22      var all_bombs = get_tree().get_nodes_in_group("Bomb")
23      for bombs in all_bombs:
24          if (bombs.position.y < -spawn_y) or (Globals.spawning == false):
25              bombs.queue_free()
26              # TODO #7: Change score only when bombs are destroyed during gameplay.
27              if Globals.spawning == true:
28
```

29

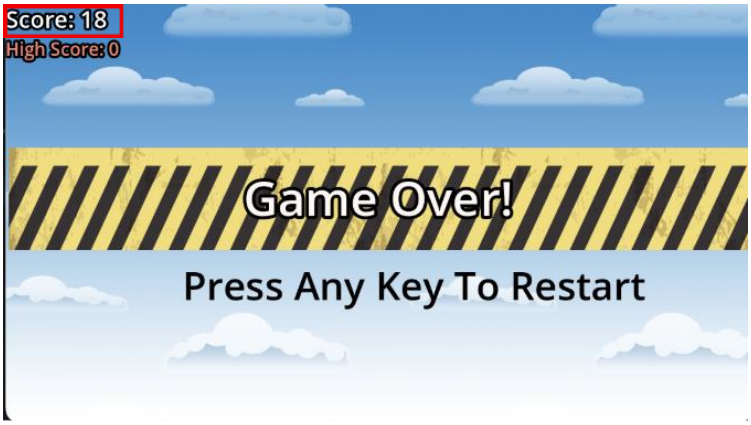
Inside the `if` statement, call the global **change_score** function, and pass `1` as the parameter – the player should get one point for each bomb they avoid.

```
20
21  func _process(_delta: float) -> void:
22      var all_bombs = get_tree().get_nodes_in_group("Bomb")
23      for bombs in all_bombs:
24          if (bombs.position.y < -spawn_y) or (Globals.spawning == false):
25              bombs.queue_free()
26              # TODO #7: Change score only when bombs are destroyed during gameplay.
27              if Globals.spawning == true:
28                  Globals.change_score(1)
29
```

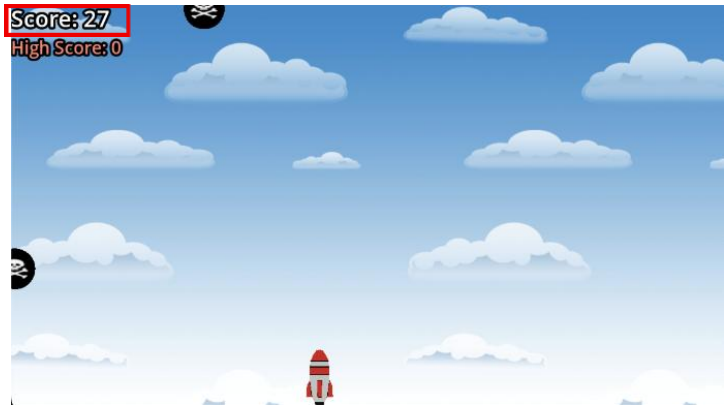
When are the bombs destroyed? When does the player earn a point?

30 Playtest the game.

Notice how the score increases by 1 every time the player dodges a bomb.



What happens to the score when the game is restarted after the player is hit by a bomb?



Pause for **Sensei Stop #4**!

Check with a Code Sensei to make sure the `update_score()` function was created correctly, and that the `_process` function in the **spawner** script was updated.

Reminder: Save your work!

31

Update the score label so it resets to 0 when the game ends and the player starts over.

Find the **TODO 8** comment in **game_controller.gd**. Set the global `score` variable to `0`, then call the `update_score()` function using the global `score` as the parameter.

```
35 func game_over():
36     game_end.visible = true
37     Globals.started = true
38     Globals.spawning = false
39     player.queue_free()
40     # TODO #12: Only update high schore label if score is greate
41
42     # TODO #8: Reset the score when the game ends.
43     Globals.score = 0
44     update_score(Globals.score)
45
```

Pro Tip:

Both lines of code are needed to make the score display correctly. The first line changes the actual value of the **score** variable back to 0, and the second line calls the function which will update what the Score label node will show the player.

32

Playtest the game again.

Notice that when the player gets hit by a bomb, the score now resets to 0!

33

Add code to display the player's highest score!

Create a variable for the HighScore label, and a local variable to store the value of the player's highest score.

In **game_controller.gd**, find the **TODO #9** comment, and declare an `@onready best_score` variable underneath. Use the `get_node` method to assign it the value of the **HighScore** label.

```
1   extends Node3D
2
3   @export var rocket: PackedScene
4   @onready var game_start = get_node("UserInterface/GameStart")
5   @onready var game_end = get_node("UserInterface/GameEnd")
6   var player: Node
7   # TODO #1: Create variable for score label.
8   @onready var score_label = get_node("UserInterface/ScoreCanvas/Score")
9   #TODO #9: Create variable for high score and best score label.
10  @onready var best_score = get_node("UserInterface/ScoreCanvas/HighScore")
11
12
```

34 Underneath, declare a local `high_score` variable of type `int`.

```
1   extends Node3D
2
3   @export var rocket: PackedScene
4   @onready var game_start = get_node("UserInterface/GameStart")
5   @onready var game_end = get_node("UserInterface/GameEnd")
6   var player: Node
7   # TODO #1: Create variable for score label.
8   @onready var score_label = get_node("UserInterface/ScoreCanvas/Score")
9   # TODO #9: Create variable for high score and best score label.
10  @onready var best_score = get_node("UserInterface/ScoreCanvas/HighScore")
11  var high_score: int
12
```

This will store the value of the highest score the player has reached so far.

35 Reset the high score when the game is first launched.

Find the **TODO #10** comment in the `_ready()` function.

Set the value of the `high_score` variable to `0`.

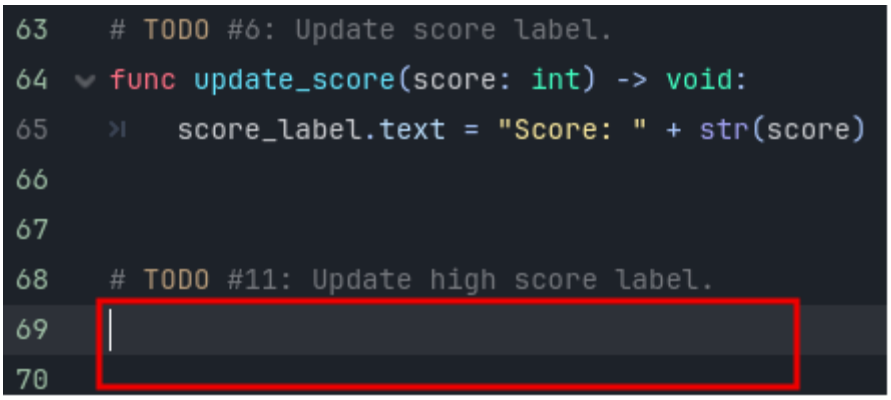
```
14  func _ready() -> void:
15      Globals.hit.connect(game_over)
16      game_start.visible = true
17      Globals.spawning = false
18      game_end.visible = false
19      # TODO #5: Set score to 0 at start of game and connect signal.
20      Globals.score = 0
21      Globals.score_changed.connect(update_score)
22      # TODO #10: Set high score to 0 at start of game.
23      high_score = 0
24
```

36 Update the **HighScore** label to show the player's best score so far.

Find the **TODO #11** comment.

Underneath, add the following code:

- Define an `update_high_score()` function with a `score` integer parameter.
- Update the `text` property of `best_score`.
- Use concatenation to display `"High Text: "` with the `score` parameter.



Refer back to steps 26-27 for guidance.

37 Check that the code matches the screenshot.

```
64    # TODO #6: Update score label.
65    func update_score(score: int) -> void:
66        score_label.text = "Score: " + str(score)
67
68
69    # TODO #11: Update high score label.
70    func update_high_score(score:int) -> void:
71        best_score.text = "High Score: " + str(score)
72
```

38 Update the high score if the player's current score is greater than the existing high score.

Find the **TODO #12** comment inside the `game_over()` function. Create an if statement that checks if the global **score** variable is greater than the value of `high_score`.

```
37  func game_over():
38      game_end.visible = true
39      Globals.started = true
40      Globals.spawning = false
41      player.queue_free()
42      # TODO #12: Only update high schore label if score is greater than current high score.
43      if Globals.score > high_score:
44      # TODO #8: Reset the score when the game ends.
45      Globals.score = 0
46      update_score(Globals.score)
47
```

39 Inside the `if`-statement, update the `high_score` variable to equal the current **score**.

```
37  func game_over():
38      game_end.visible = true
39      Globals.started = true
40      Globals.spawning = false
41      player.queue_free()
42      # TODO #12: Only update high schore label if score is greater than current high score.
43      if Globals.score > high_score:
44          high_score = Globals.score
45      # TODO #8: Reset the score when the game ends.
46      Globals.score = 0
47      update_score(Globals.score)
48
```

40 Call the `update_high_score` function to update the **HighScore** label with the global `score` variable (which is also the new high score).

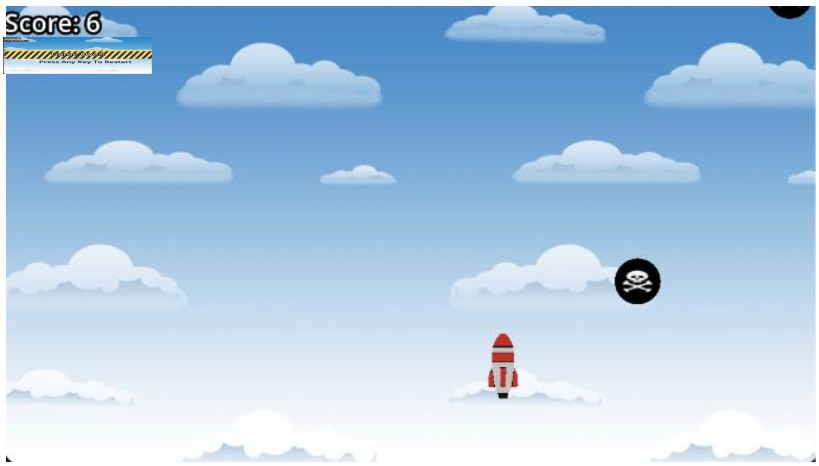
```
37  func game_over():
38      game_end.visible = true
39      Globals.started = true
40      Globals.spawning = false
41      player.queue_free()
42      # TODO #12: Only update high schore label if score is greater than current high score.
43      if Globals.score > high_score:
44          high_score = Globals.score
45          update_high_score(Globals.score)
46      # TODO #8: Reset the score when the game ends.
47      Globals.score = 0
48      update_score(Globals.score)
49
```

Reminder:

The code for **TODO #12** needs to come *before* the code for **TODO #6**, where the score gets reset to 0. Otherwise, the `update_high_score()` function will always update with 0, and not the real high score.

41 Playtest the game again.

Notice that when the player gets hit by a bomb and the game ends, the high score shows the highest score the player has reached since they started playing.



Pause for **Sensei Stop #5**!

You finished another part of the Dropping Bombs project and created an important part of many games! Players will want to play this game repeatedly to reach new high scores.

- ☐ What did you learn about how to store and update information in variables?
- ☐ What did you learn about what the player will see as they play your game?
- ☐ 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 11: Dropping Bombs Part 5** 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 12: Dropping Bombs Part 6** Ninja Guide.