CODE NINJAS®

Silver Belt Ninja Guide

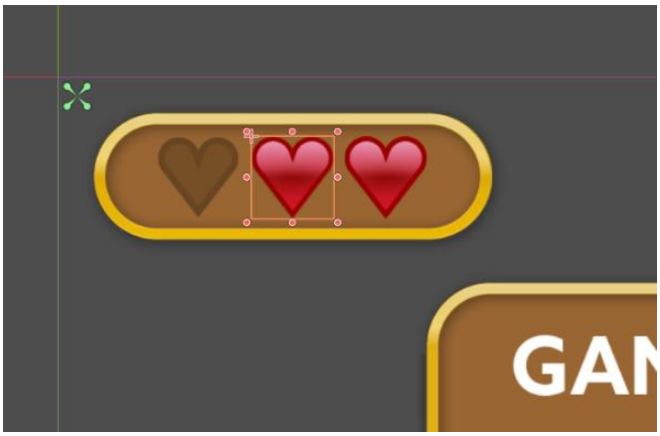
Activity 12: Amazing Ninja Worlds Part 2

Health Bars

Games are more interesting when the player has limited resources. One frequent example of a limited resource is player health. With limited health, if a player makes the wrong decision, they will lose the level and have to restart. But how does the user know how much health the player has left?

This is the purpose of health UI. From the simple hearts seen in many games, to dynamically shifting health bars, to red flashing screens, games must communicate the player's health status so that the user can make appropriate risk assessments during their gameplay. What are some of the ways that you have seen games portraying the amount of health a player has left?

To create a health UI, the amount of health that the player has must be tracked throughout the game. This can easily be done with a variable.

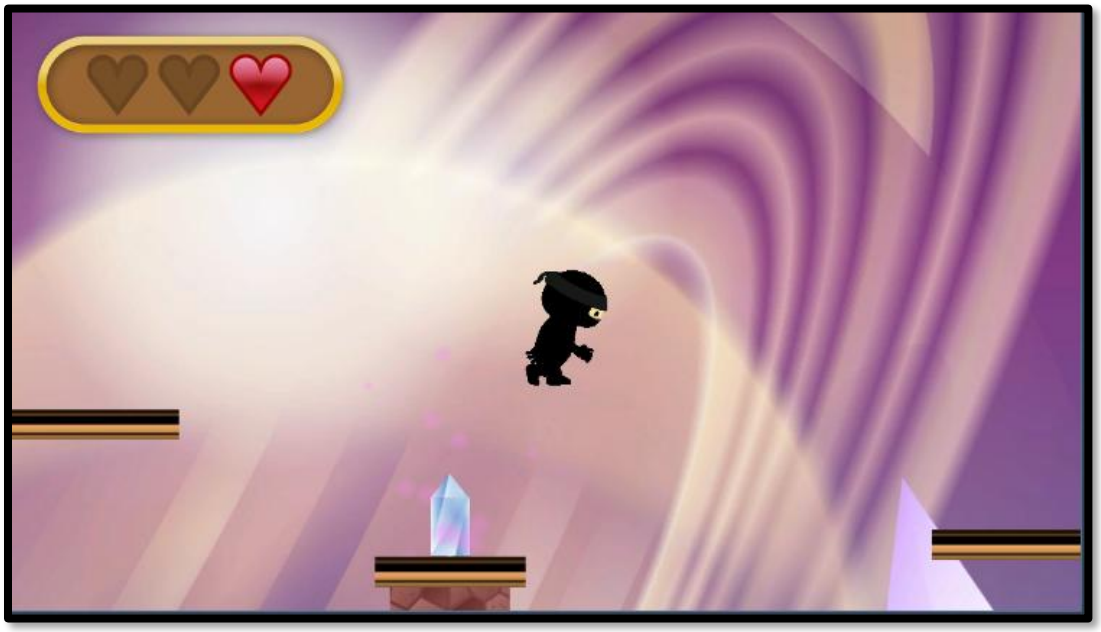


The health then needs to be displayed inside the game. This functionality can be accomplished with **TextureRect** nodes that are set to appear or disappear according to how much health the player has.

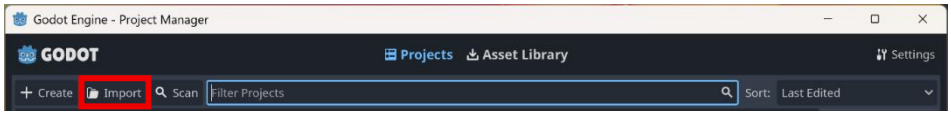
When an event occurs that should hurt the player, the script must first calculate how much health the player has left. The health UI must then update to reflect the player's health status. If the player's health reaches or goes under zero, then the game must update to either restart the level or the game in its entirety before resetting the player's health back to its full value.

Activity 12: Amazing Ninja Worlds Part 2

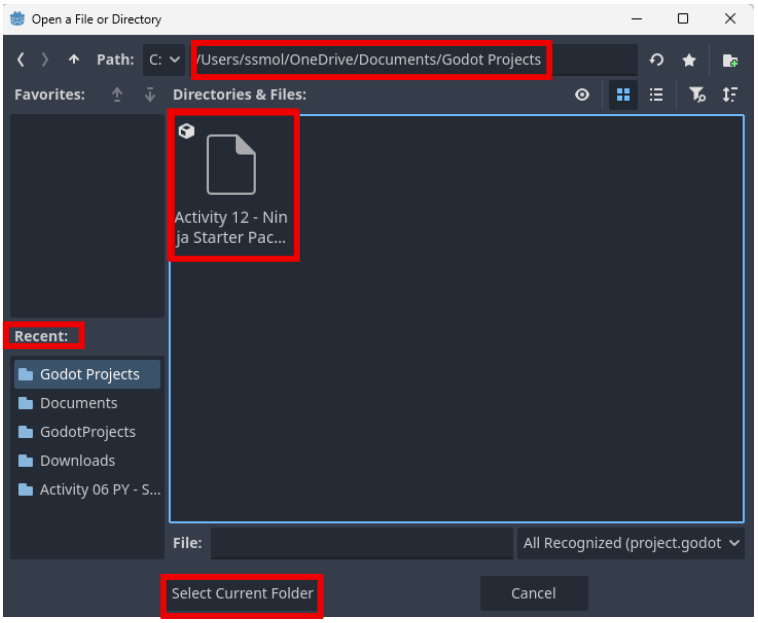
In this project, you will create health bar UI and functionality. You will script the health bar UI to update when a life is lost, and you will script the logic that will reset the health bar UI when a level is failed.



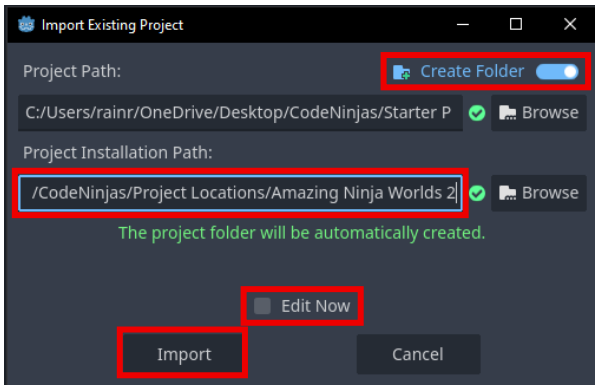
1 Open Godot and click **Import.**



2 In the File Directory, navigate to the correct file path. Select **SB Activity 12 – Ninja Starter Pack.zip** and click **Open**.



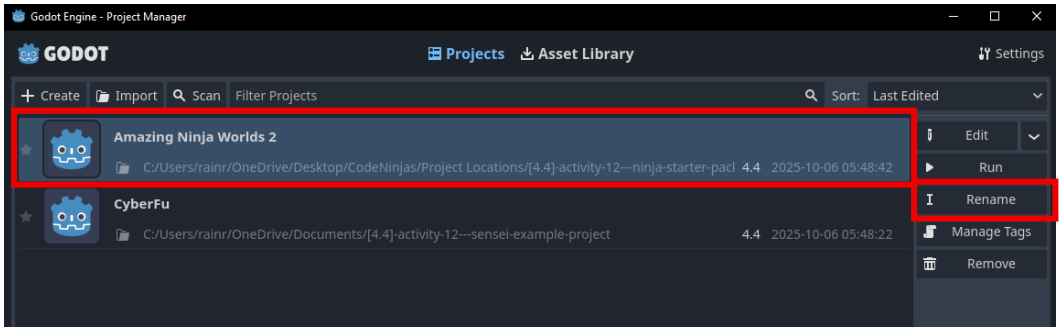
3 Update the **Project Installation Path** and make sure **Create Folder** is enabled. **Uncheck Edit Now** and click **Import**.



4 The project will appear at the top. Click on the project and select **Rename** on the right.

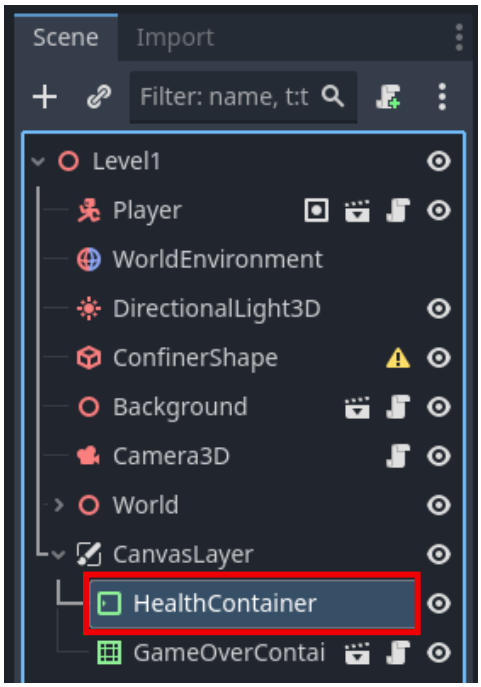
Update the Project Name to **[YourInitials]AmazingNinjaWorlds2**.

Once renamed, select the project and click **Edit** to open the starter code.



5 Once loaded, the **level_1** scene should open. If the scene does not open automatically, find **level_1.tscn** in the **Scenes > Levels** folder and open the scene.

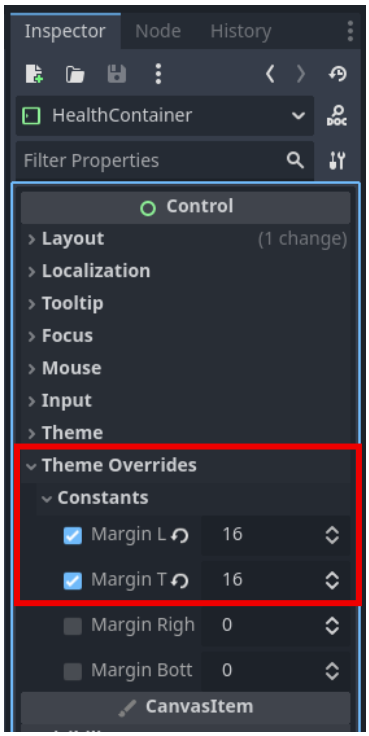
In **Scene**, add a **MarginContainer** as a child to **CanvasLayer** and rename it to **HealthContainer**. This will become the UI healthbar.



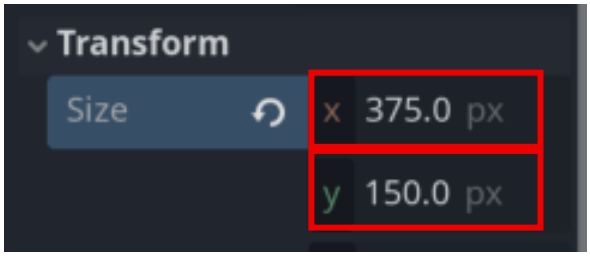
New Concept: MarginContainer

It can be difficult to adjust exact spacing between individual items by hand. **MarginContainer** is made specifically to evenly adjust spacing around its child nodes to help align elements nicely.

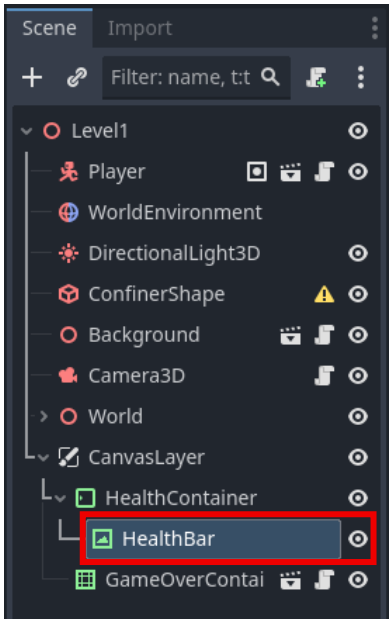
6 In the **Inspector** for **HealthContainer**, under **Theme Overrides > Constants**, set both **Margin Left** and **Margin Top** to **16**.



Under **Transform**, set **Size X** to **375** and **Size Y** to **150**.



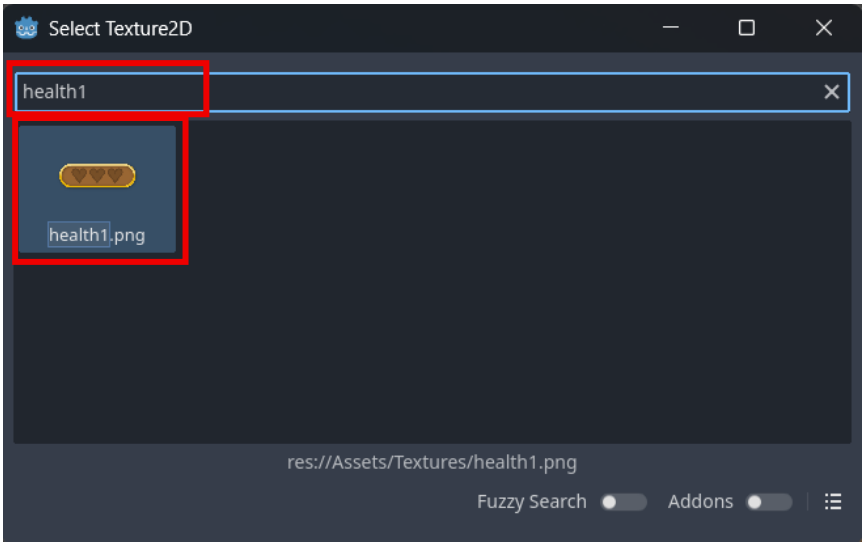
7 In **Scene**, add a **TextureRect** as a child to **HealthContainer** and rename the node **HealthBar**.



Reminder:

A **TextureRect** is a control node that displays a texture inside a UI. The minimum size for the node's bounding rectangle is equal to the texture size.

8 In the **Inspector** for **HealthBar**, use **Quick Load** to search for and set the **Texture** to **health1.png**.

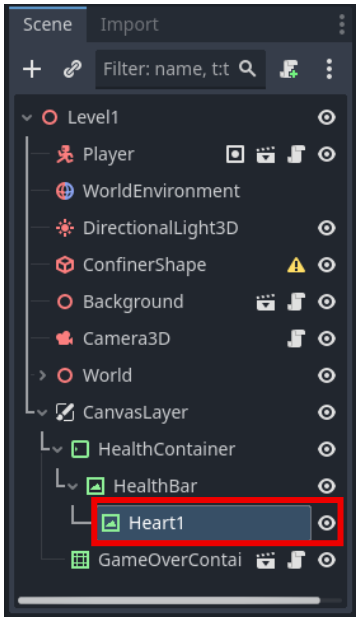


Pro Tip:

Notice that while the **HealthBar** is selected, it has a small 16-pixel margin from the top and left of the screen. The **HealthBar** inherits these margins from its parent **HealthContainer.**

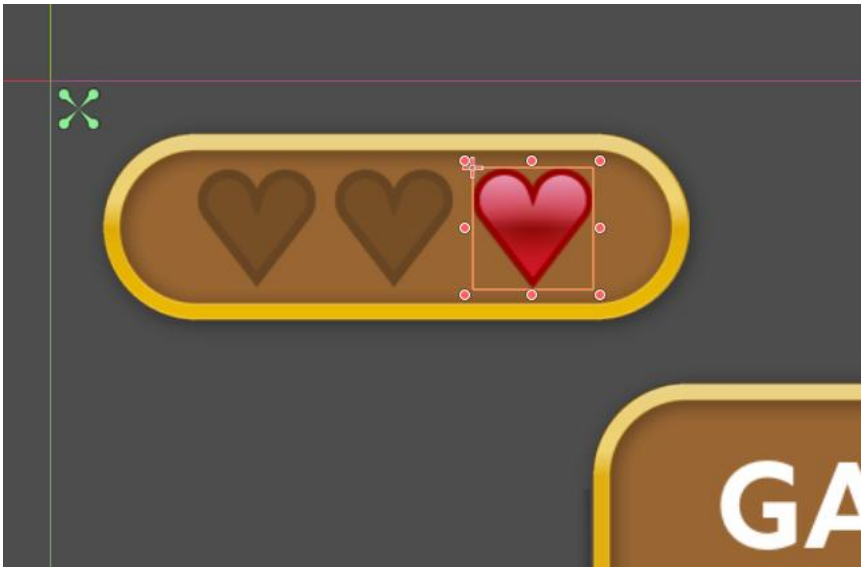
9 Toggle to the **2D workspace** to see the UI as it is made.

10 In **Scene**, add a **TextureRect** as a child to **HealthBar** and rename the node **Heart1**.

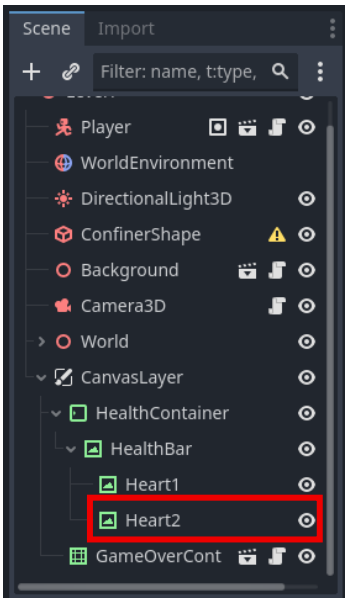


11 In the **Inspector** for **Heart1**, use **Quick Load** to search for and set the **Texture** to **healthHeart.png**.

12 In the **2D workspace**, with **Heart1** selected, drag the heart so that it is in the rightmost slot of the health bar UI.



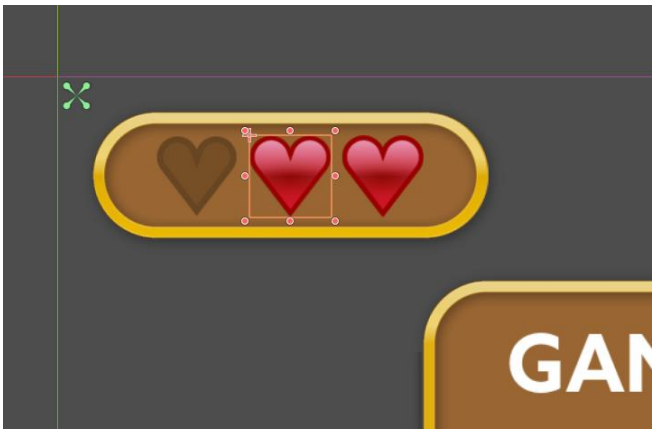
13 In **Scene**, duplicate the **Heart1** node and name it **Heart2**.



Pro Tip:

Press **CTRL+ D** with Heart1 selected to duplicate the node. Godot will automatically duplicate the node and iterate the number next to it, naming the new node

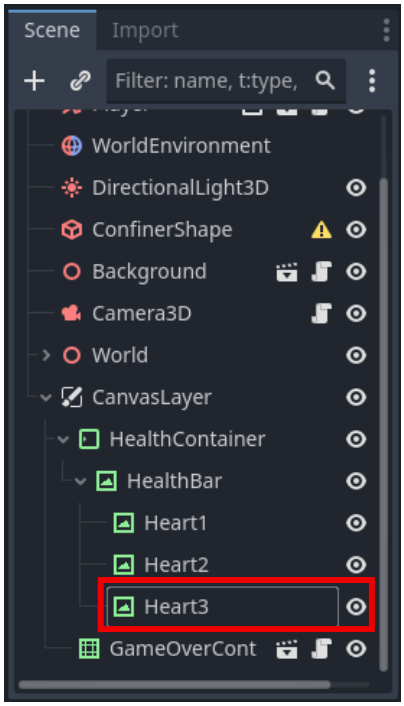
14 In the **2D workspace**, select **Heart2**. Drag **Heart2** to the left and place it over the middle slot in the health bar UI.

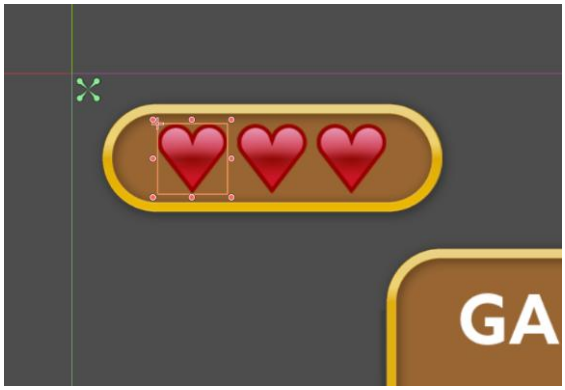


Pro Tip:

After beginning to drag Heart2, hold down **Shift + CTRL** to lock the heart and move it directly horizontal to Heart1.

15 Repeat steps 13-14 to create the **Heart3** node and place its heart texture in the final empty slot in the UI.





Pause for **Sensei Stop #1**!

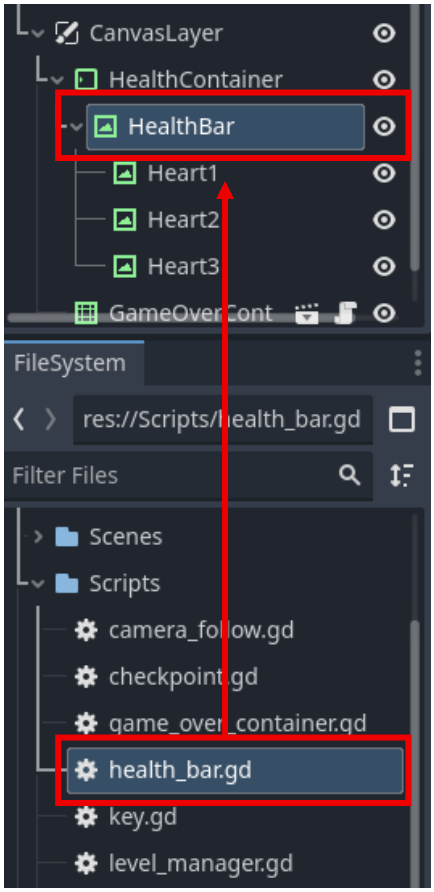
Check with a Code Sensei and confirm the health bar UI nodes are properly set up before continuing.

Reminder: Save your work!

16

Code the health bar UI.

In **Scene**, locate the **HealthBar** node. From **FileSystem**, attach the **health_bar.gd** script to the **HealthBar** node.



17 In **FileSystem**, open the **health_bar.gd** script. Notice the starter code included in the script.

```
1   class_name HealthBar
2   extends TextureRect
3
4   var hearts: Array[TextureRect] = []
5
6   func _ready() -> void:
7       for child in get_children():
8           if child is TextureRect:
9               hearts.append(child)
10
11  func set_health(value: int) -> void:
12      for i in range(hearts.size()):
13          # This is the simple way of doing it
14          #if i < value:
15              #hearts[i].visible = true
16          #else:
17              #hearts[i].visible = false
18
19          # This is the one line, better way of doing it
20          hearts[i].visible = (i < value)
21
```

The script first uses the `class_name` keyword to define the script as a globally accessible class with the name `HealthBar`. Classes will be explored further in the next activity, Food Frenzy Part 1.

An empty `array` of `TextureRect` nodes is then defined, named `hearts`.

The `_ready()` method searches through the child nodes of the node the script is attached to (the **HealthBar** node) and adds a reference to any child `TextureRect` node to the `hearts` array.

Finally, the `set_health()` function is defined. It takes in an `integer` parameter `value`, then **loops** through the `hearts` array, setting `value` amount of hearts to be `visible` in the UI.

18 In **FileSystem**, under **Scripts**, open the **player.gd** script.

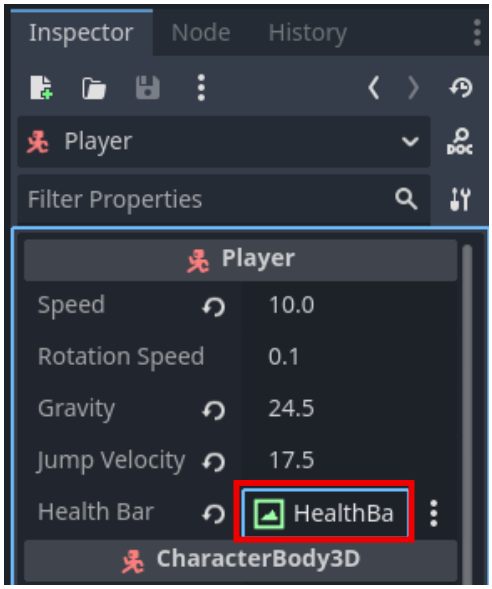
The variables and functions in this script, namely **`lives`,** make up the bulk of the functionality of tracking health and its UI, so **player.gd** should be edited first.

```
14
15  # --------------------
16    # TODO  1
17    # Create lives and health bar variables
18    # --------------------
19    @onready var lives: int = 3
20    @export var health_bar: HealthBar
21
```

Under **TODO 1**, use the **`@onready`** keyword to define a **`lives`** variable of type **`int`**, and initialize it to **`3`**.

On the next line, use the **`@export`** keyword to define a **`health_bar`** variable of type **`HealthBar`**.

19 In the **Inspector** for the **Player** node, assign **Health Bar** to the **HealthBar** node.



20

Code what happens when the player loses a life.

Scroll to the bottom of the **player.gd** script to find **TODO 2** in the `_lose_life()` function. Inside the function, `decrement lives by 1`.

```
93
94  func _lose_life():
95      # --------------------
96      # TODO 2
97      # Program the _lose_life function
98      # --------------------
99      lives -= 1
100
```

Reminder:

Decrement means to decrease or reduce by a number. If a different number value is not specified, then subtract 1 from the variable.

21 On the next line, chain together a reference to the `health_bar` variable and a call to the function `set_health()` given the value `lives`.

Ensure the call to `pass` is **removed** from the function.

The call to `health_bar.set_health(lives)` updates the UI to display the amount of hearts equal to the number of lives the player has left.

```
93
94  func _lose_life():
95      # --------------------
96      # TODO 2
97      # Program the _lose_life function
98      # --------------------
99      lives -= 1
100     health_bar.set_health(lives)
101
```

Reminder:

pass is a keyword that prevents parsing errors from being thrown for empty methods and functions, or other code that has an indented body. This keyword is commonly used while outlining code that has yet to be implemented in languages that require indentation, like Python and GDScript.

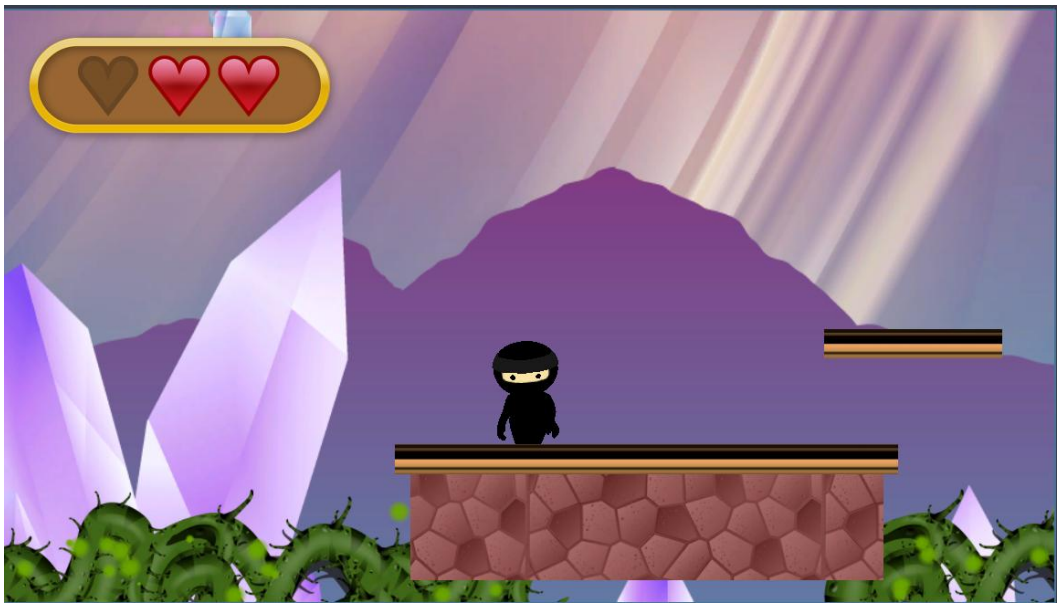
22 On the next line, chain together a reference to `LevelManager` and a call to the `reset_player()` function.

```
93
94  func _lose_life():
95      # --------------------
96      # TODO 2
97      # Program the _lose_life function
98      # --------------------
99      lives -= 1
100     health_bar.set_health(lives)
101     LevelManager.reset_player()
102
```

23 Playtest the game.

The player should now lose hearts on the health bar when it hits the vines.

What happens when the player runs out of lives?



24

Code the game over UI to display when the player runs out of lives!

On the next line, create an `if` statement that checks if `lives` is less than or equal to `0`.

```
93
94  func _lose_life():
95      # --------------------
96      # TODO 2
97      # Program the _lose_life function
98      # --------------------
99      lives -= 1
100     health_bar.set_health(lives)
101     LevelManager.reset_player()
102
103     if lives <= 0:
104
```

25

Inside the if statement, set `frozen` to `true`. On the next line, chain together a reference to `LevelManager` and a call to the `game_over()` function.

```
93
94  func _lose_life():
95      # --------------------
96      # TODO 2
97      # Program the _lose_life function
98      # --------------------
99      lives -= 1
100     health_bar.set_health(lives)
101     LevelManager.reset_player()
102
103     if lives <= 0:
104
105
106
```

26 Check the code and update the script as needed.

```
93
94  func _lose_life():
95      # --------------------
96      # TODO 2
97      # Program the _lose_life function
98      # --------------------
99      lives -= 1
100     health_bar.set_health(lives)
101     LevelManager.reset_player()
102
103     if lives <= 0:
104         frozen = true
105         LevelManager.game_over()
106
```

In the `_lose_life()` function, the `lives` variable that tracks the number of lives the player currently has is decremented.

The call to `health_bar.set_health()` updates the UI so that the correct number of hearts are visible.

The call to `LevelManager.reset_player()` resets the player back to its last checkpoint.

The `if` statement checks if the player has lost all lives. If it has, then the player is frozen, and the game is over.

27

Set what happens when the level is reset.

In **FileSystem**, under **Scripts**, open the **level_manager.gd** script.

```
1   extends Node
2
3   var current_checkpoint : Vector3 = Vector3.ZERO
4   var player : Player
5   var keys_left : int = 0
6
7   var gameOverCanvas
8
9   func reset_player() -> void:
10      player.global_position = current_checkpoint
11
12  func change_level(scene_path : String) -> void:
13      keys_left = 0
14      get_tree().call_deferred("change_scene_to_file", scene_path)
15
16  func game_over() -> void:
17      gameOverCanvas.visible = true
18
```

Notice the starter code already included in the level_manager.gd script.

There are three variables: `current_checkpoint`, `player`, and `keys_left`. These are the same variables from Amazing Ninja Worlds Part 1.

The `reset_player()` function sets the player's position back to the position of the last checkpoint that it interacted with.

The `change_level(scene_path)` function resets the `keys_left` variable back to `0`, then defers a call to change the scene to the scene specified by `scene_path`.

The `game_over()` function makes the Game Over UI visible.

28 Find **TODO 3**. On the next line, define a function named `restart_level()` that takes no parameters and returns void.

```
18
19  # --------------------
20    # TODO 3
21    # Create the restart_level function
22    # --------------------
23  func restart_level() -> void:
24    
```

29 In the `restart_level()` function, set `keys_left` to `0`.

On the next line, chain together calls to the methods `get_tree()` and `reload_current_scene()`.

```
18
19  # --------------------
20    # TODO 3
21    # Create the restart_level function
22    # --------------------
23  func restart_level() -> void:
24    
25    
26    
```

30 Check that the code matches the screenshot.

```
18
19  # --------------------
20  # TODO 3
21  # Create the restart_level function
22  # --------------------
23  func restart_level() -> void:
24      keys_left = 0
25      get_tree().reload_current_scene()
26
```

When the function `restart_level()` is called, `keys_left` is reset to `0`. The scene is then reloaded to start the level over.

31 In **FileSystem**, under **Scripts**, open **game_over_container.gd**.

Find **TODO 4**. On the next line, chain together a reference to `Button` using the `$` symbol, the `pressed` property, and the `connect()` method with the argument `LevelManager.restart_level`.

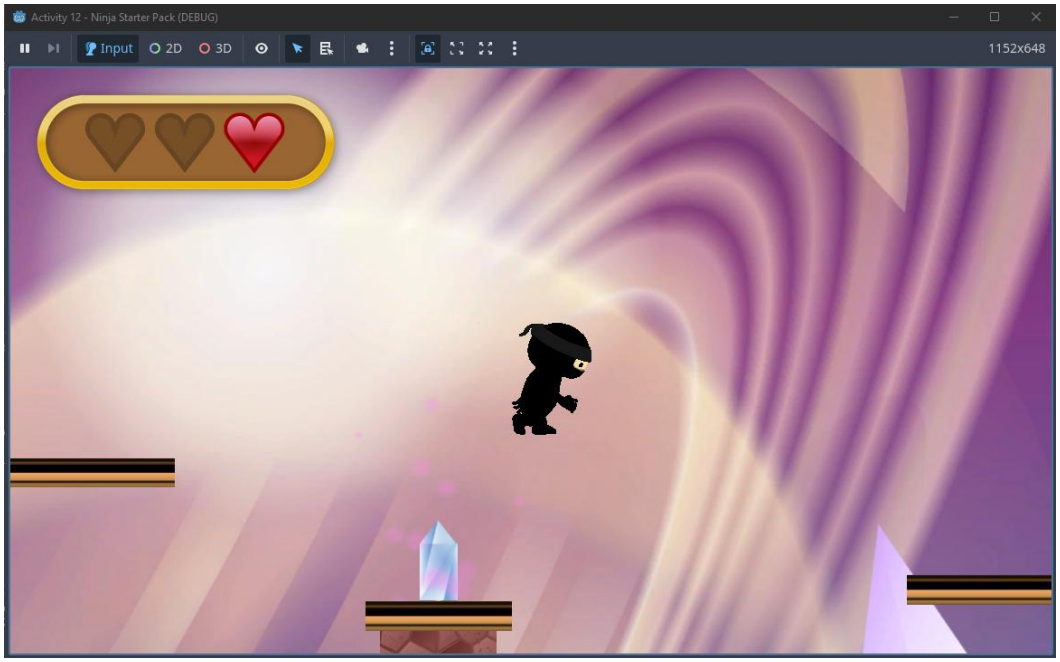
```
5   # --------------------
6   # TODO 4
7   # Connect button press to level_restart function
8   # --------------------
9   $Button.pressed.connect(LevelManager.restart_level)
```

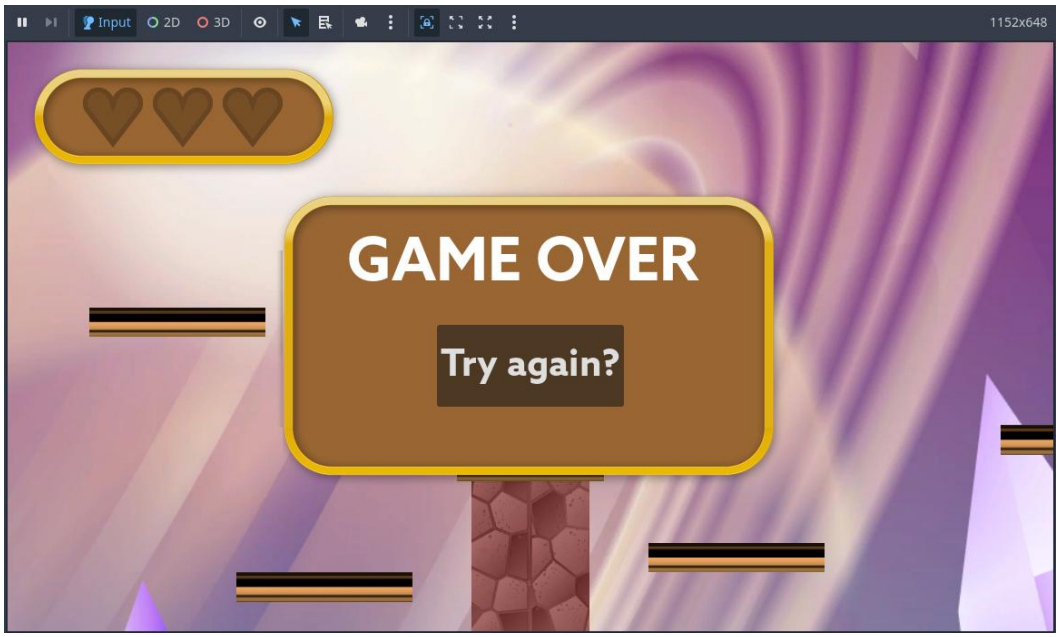
This sets the button in the **GameOver UI** to call the `restart_level()` function when pressed.

32 Playtest the game.

When the player loses all lives, the Game Over UI should appear.

When the **Try Again?** Button is pressed, it should reload the current level.





Pause for **Sensei Stop #2**!

Before submitting, check in with a Code Sensei to make sure the health bar UI is set up and working correctly. Then, reflect on the following:

- What did you learn about scripting health? Resetting levels?
- What did you enjoy most when creating this project?
- What was something you found difficult and why?

Reminder: Save your work!

Congratulations on completing **SB Activity 12: Amazing Ninja Worlds Part 2** in Godot – **You Rock**! You are now ready to save this project and submit it.

Continue your exploration with Godot by opening the **SB Activity 13: Food Frenzy Part 1** Ninja Guide.