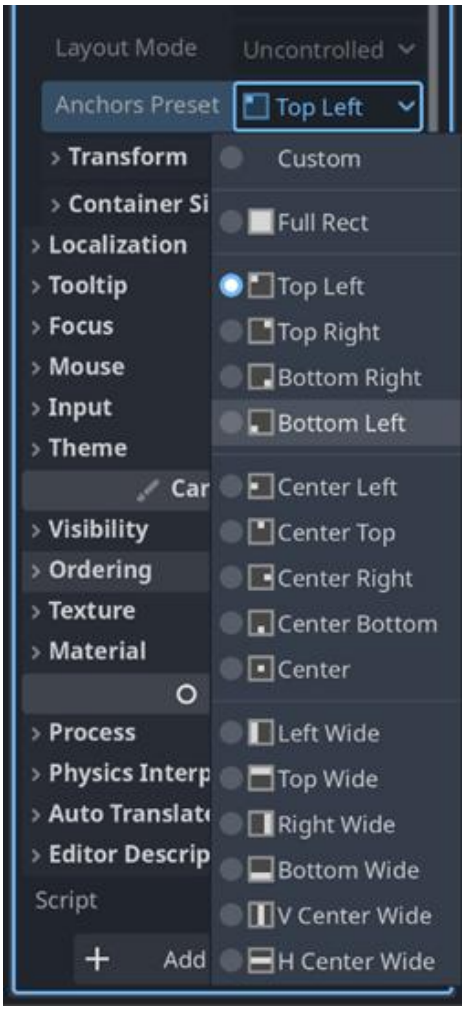
CODE NINJAS®

Silver Belt Ninja Guide

Activity 13: Food Frenzy Part 1

Control Nodes and HUD

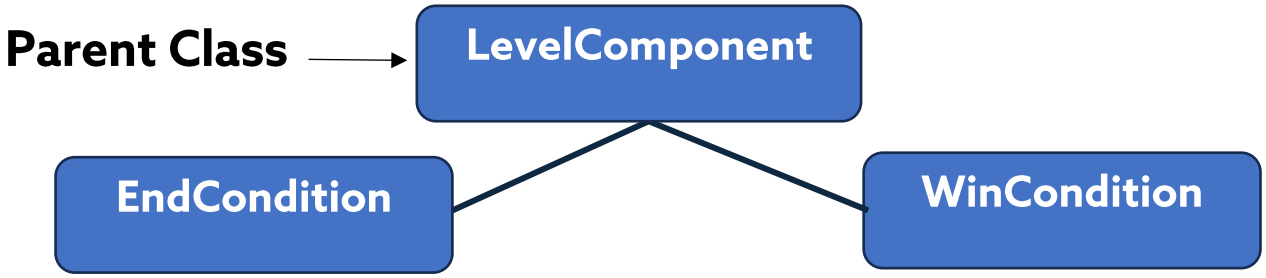
Control nodes are the base class for all UI (user interface) related nodes, such as Label, NinePatchRect and TextureRect. The Control class has a bounding rectangle, which defines its size, and an anchor position relative to the parent control (usually the parent node) or viewport. These properties can be helpful when information, such as a health or status bar, are needed to provide important information to the player during gameplay without disrupting the play experience. This type of user interface is called an HUD (heads up display).



A CanvasLayer root node is often used over a control root node when creating an HUD, or UI elements that overlap game play. CanvasLayers can be rendered above or below a 2D scene, hidden or follow the viewport. When creating strictly UI-based scenes, such as level select game screens, a control node is preferred.

Classes and Inheritance

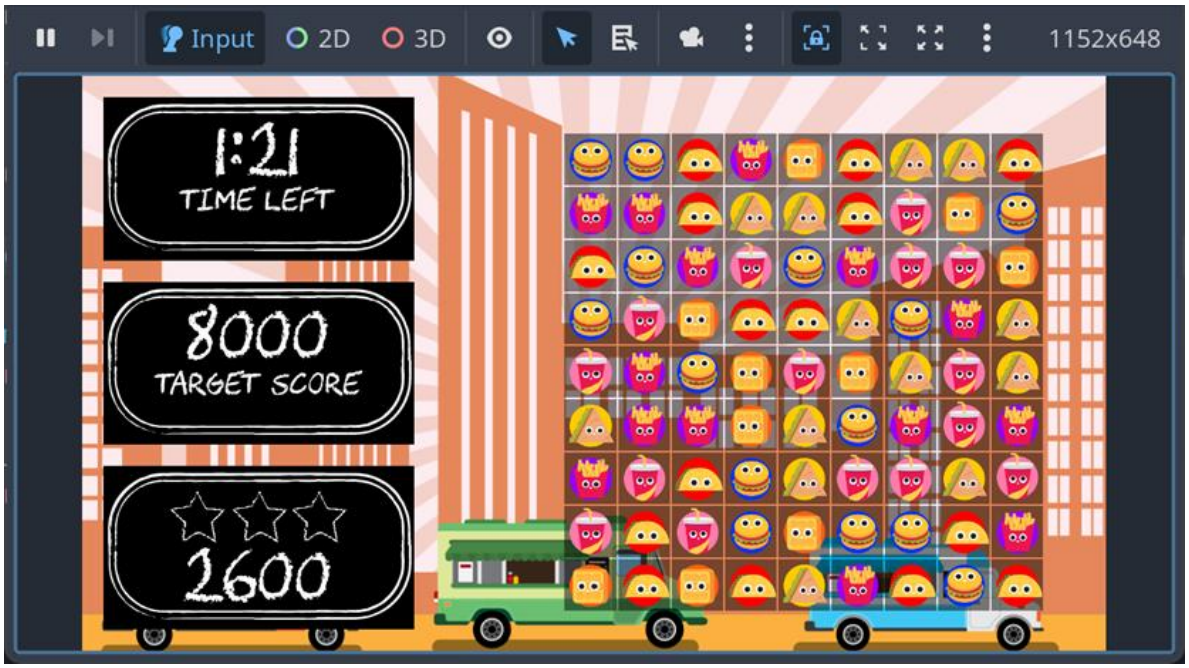
Inheritance is a concept that should sound familiar as this is what Godot's entire structure is based off. Inheritance allows for properties and methods to be passed down through Godot's node classes. Inheritance is extremely useful when creating objects that may require some, but not all the same functionalities. A game with three levels that require the same basic operations, such as moving and clearing pieces, but different goals to complete each level, can use a custom inheritance structure where the Parent Class contains the core functionality and is inherited by classes that provide level-specific functionality.



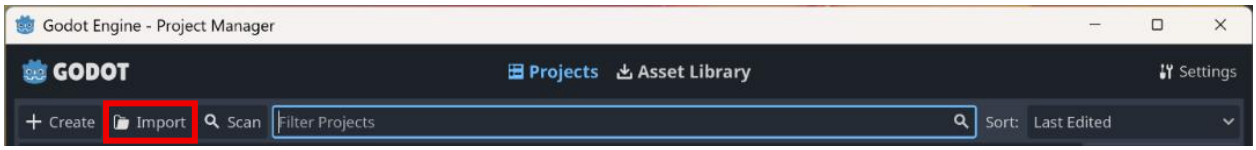
Project Description

In this project, you will build a **HUD** (heads up display) for the game's user interface. You will code all the necessary functions to connect the HUD to the game while using inheritance and custom classes to provide different UI functionality to the different game levels based on each level's objective.

By the end of this activity, you will have further explored user interface design, inheritance and classes.

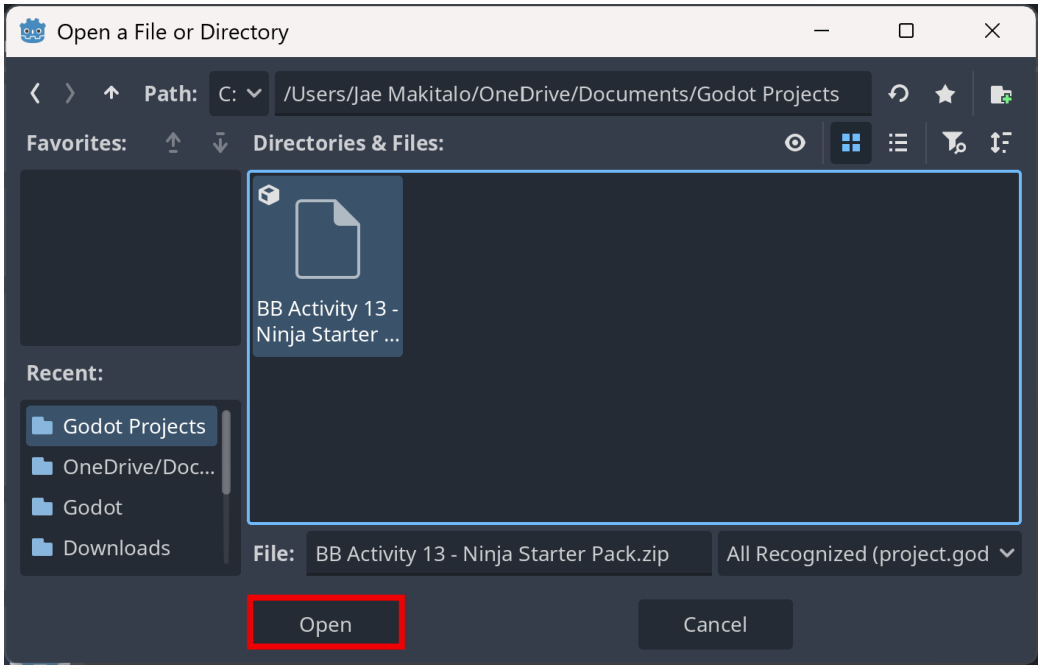


1 Open Godot and click **Import.**



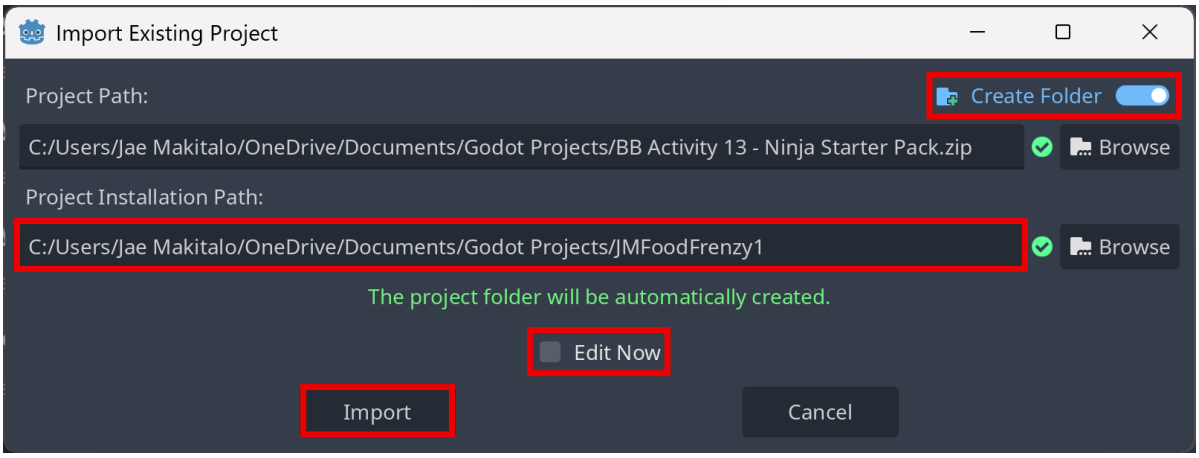
2 In the File Directory, navigate to the correct file path.

Select **SB Activity 13 – Ninja Starter Pack.zip** and click **Open**.



3 Update the **Project Installation Path** and make sure **Create Folder** is enabled.

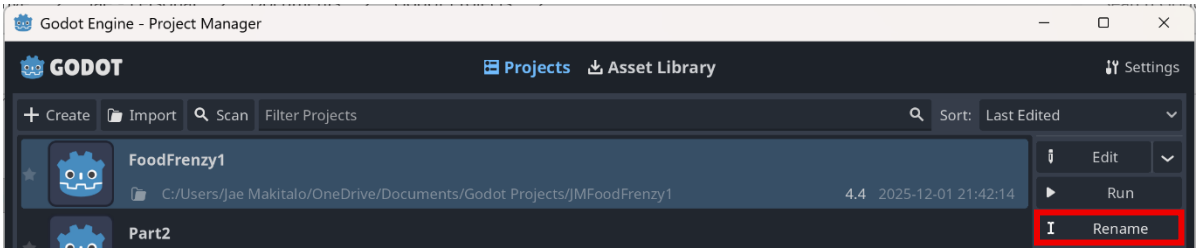
Uncheck Edit Now and click **Import**.



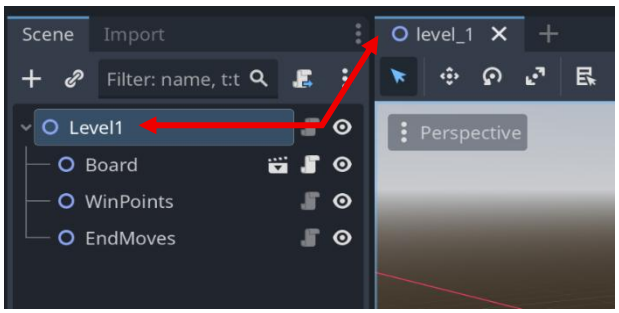
4 The project will appear at the top. Click on the project and select **Rename** on the right.

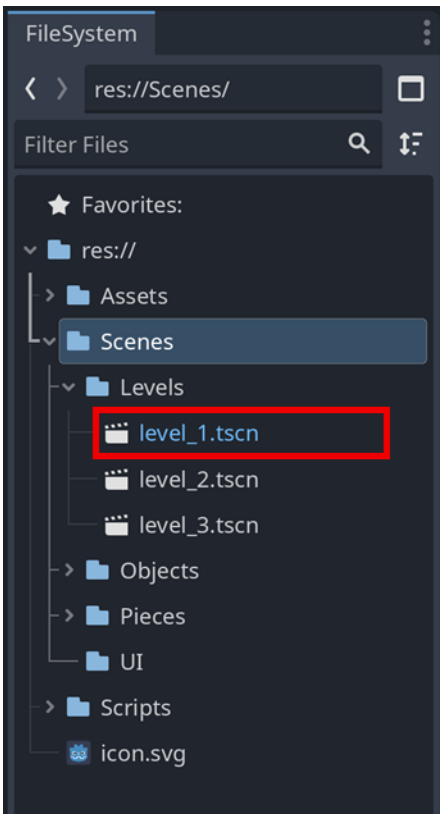
Update the Project Name to **[YourInitials]FoodFrenzy1**.

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



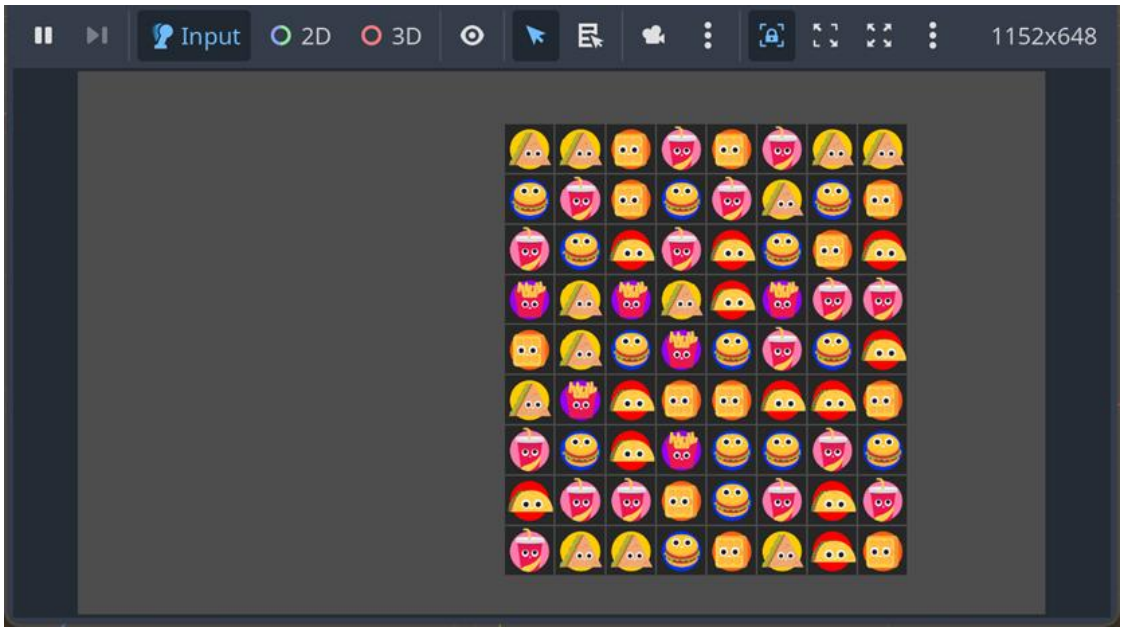
5 Once loaded, the **level_1** scene (which is the project's main scene) should open. If the scene does not open automatically, find **level_1.tscn** in the **Scene > Levels** folder and open the scene.





Toggle to the **2D workspace** when the correct scene is open.

6 Playtest the scene!

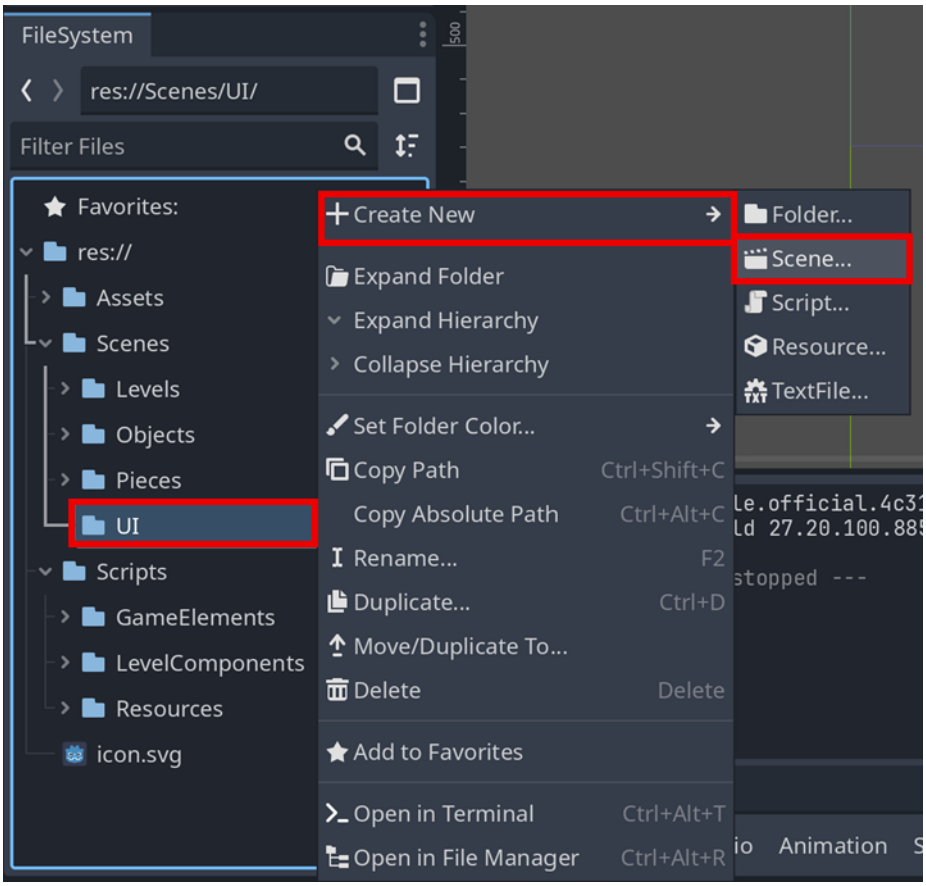


Drag a food sprite and swap it with a neighbor to create three or more in a row. If no moves can be made, the board will reshuffle itself.

How many moves can be made before the food sprites can't be swapped anymore? The game board will disappear when no more moves can be made and the game is over.

The player can only make a set number of moves before the game stops working, but the player can't see how many moves are left without a user interface!

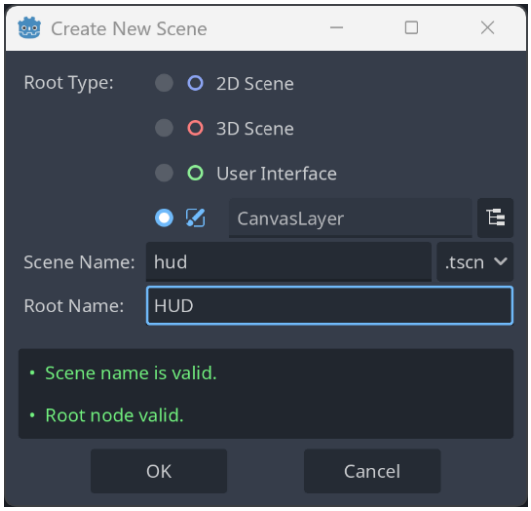
7 In **FileSystem**, right click on the UI folder in Scenes, select **+Create New** and **Scene** to create a new scene.

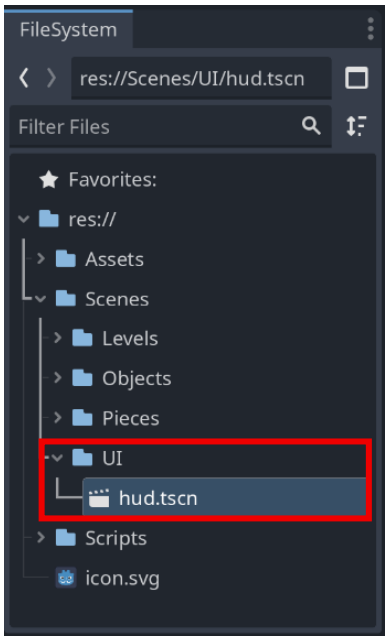


8 In the Create New Scene window, use the button to search for and select a **CanvasLayer** as the root node.

Update the Scene Name to **hud** and Root Name to **HUD**, then click **OK**.

Confirm the **hud.tscn** scene was created inside the UI folder.



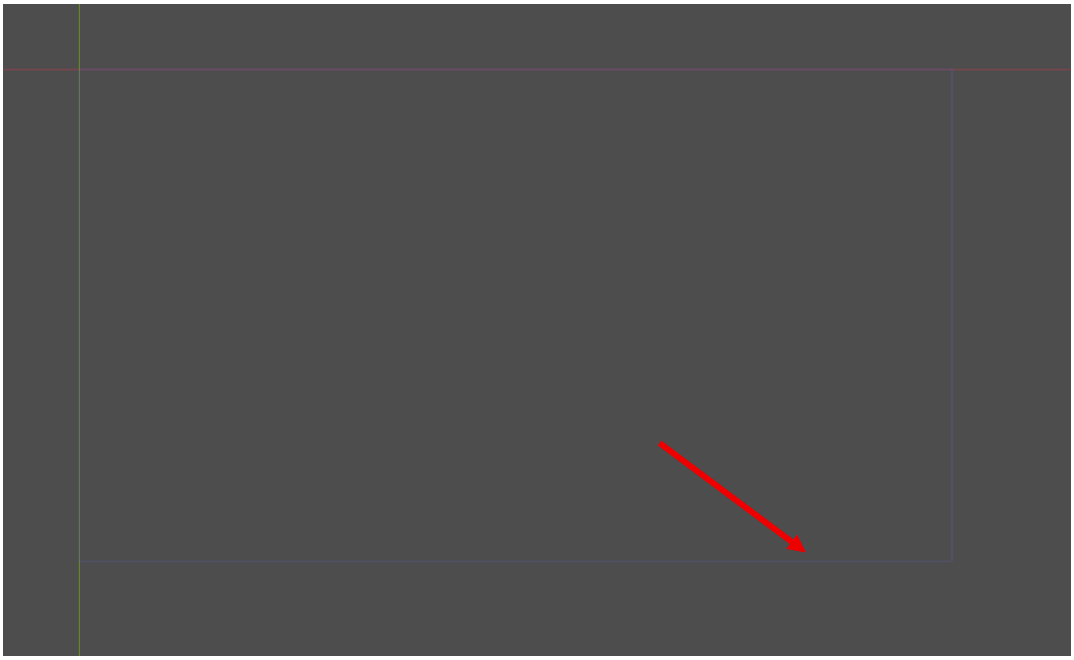


Reminder:

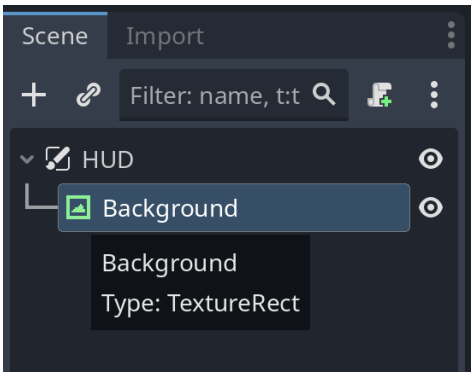
A **CanvasLayer**, not Control node, is used as the root node since the HUD will overlay and interact with gameplay.

9 The 2D workspace will show the outline of a rectangle. Use the mouse wheel to zoom out if needed to view the outline.

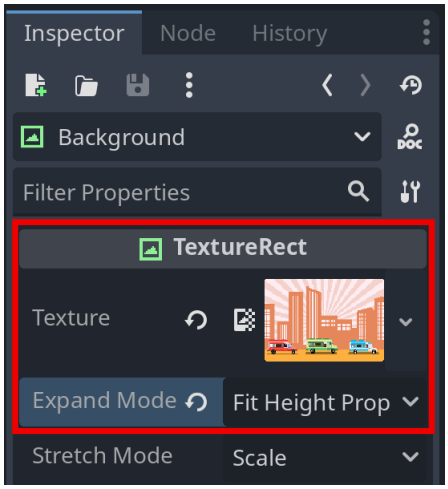
This represents the dimensions of the **CanvasLayer**, which match the viewport dimensions



10 Add a **TextureRect** as a child to **HUD** and rename the node **Background**.

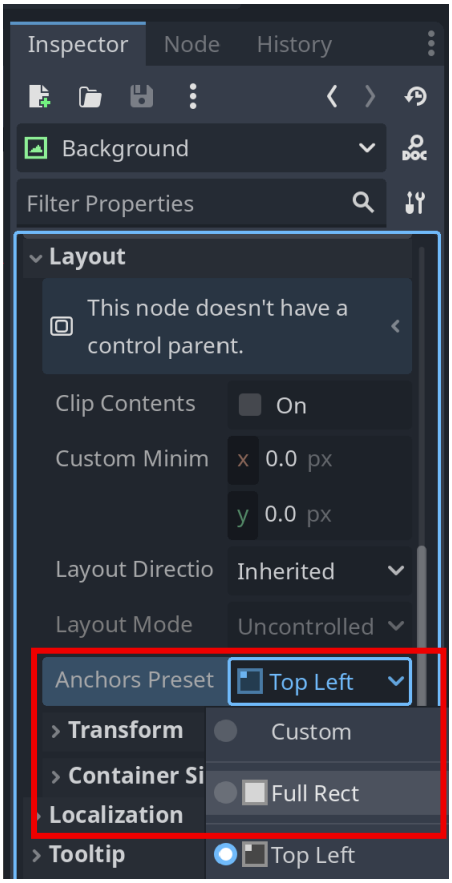


11 In the **Inspector** for **Background**, use **Quick Load** to set the **Texture** to **background_1.png** and set **Expand Mode** to **Fit Height Proportional**.



The texture will shrink down in the 2D workspace.

12 In the **Inspector** under **Layout**, set **Anchors Preset** to **Full Rect**.

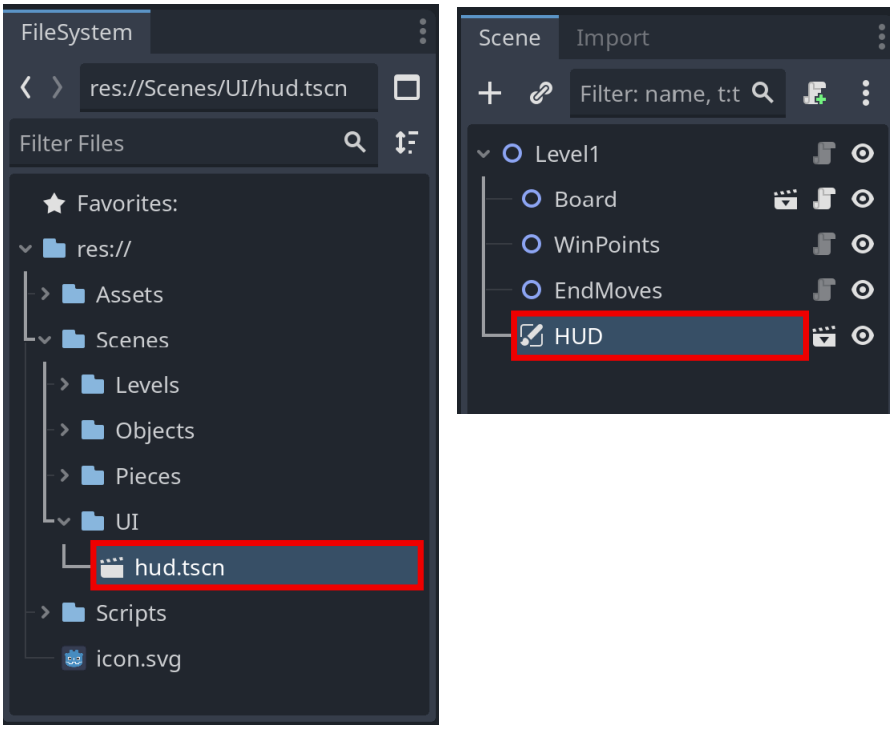


The texture should now fill the CanvasLayer in the 2D workspace.

13

Return to the **level_1.tscn** scene.

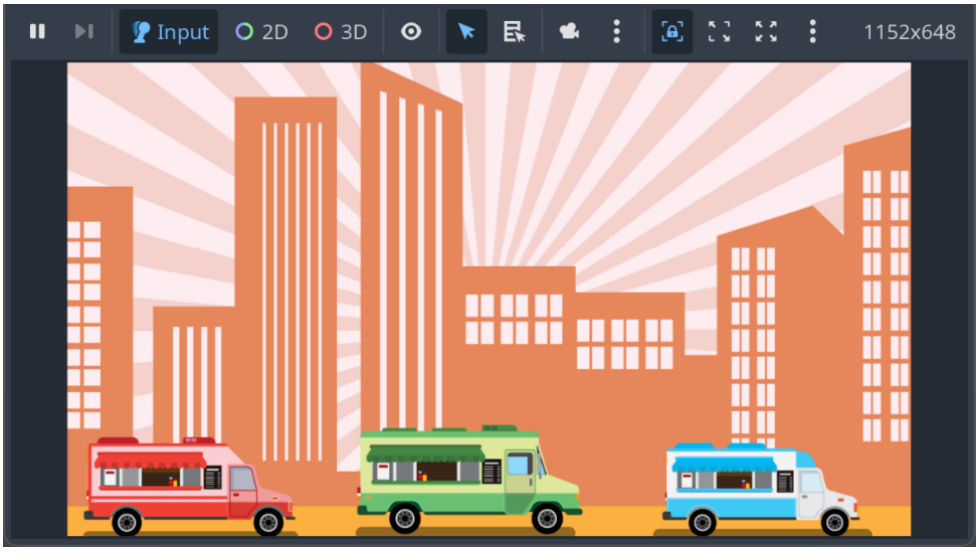
Find the **hud.tscn** scene under **Scenes > UI** and add the scene to Level1.



14

Playtest the **level_1.tscn** scene. What do you notice about the background?

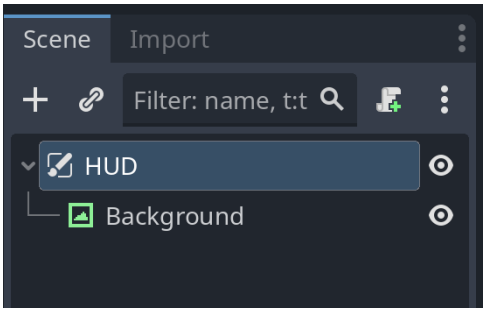
The background image is currently in front of the game board. This is because by default, a 2D scene renders with **index 0** but a CanvasLayer renders with **index 1**. This draws the CanvasLayer, which includes the background texture, above the game.

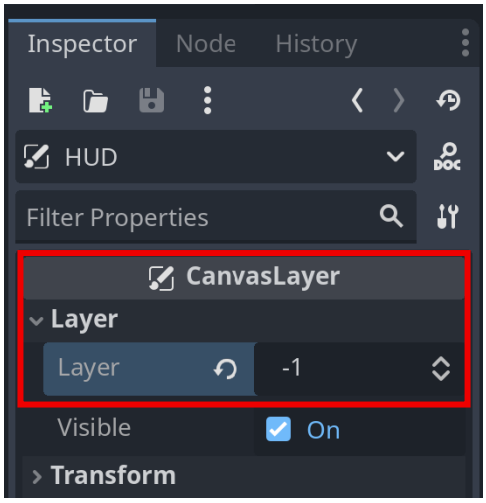


15 Return to the **hud.tscn** scene.

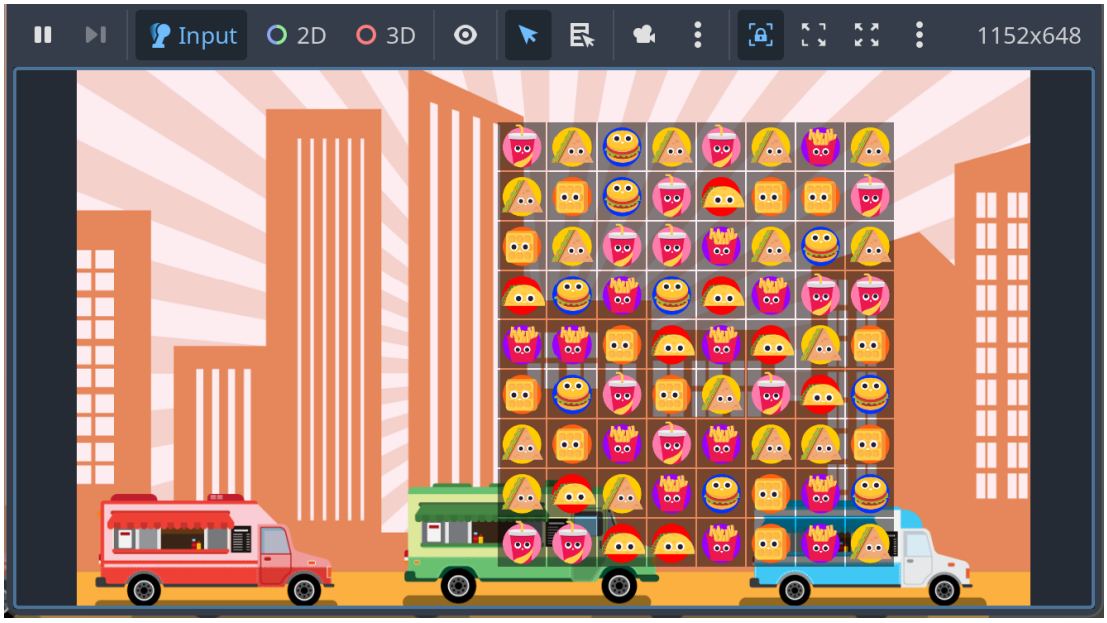
In the **Inspector** for **HUD** change **Layer** from **1** to **-1.**

This will draw the CanvasLayer, which includes the background, behind the 2D workspace.

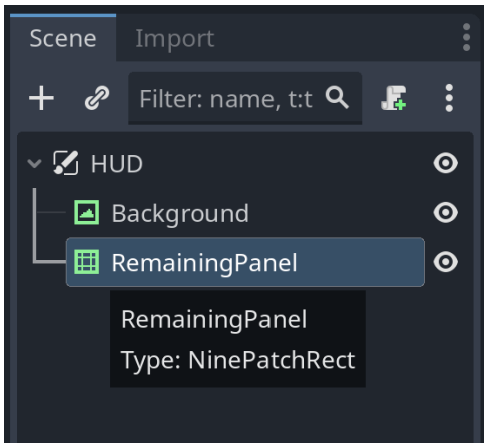




16 Playtest the level. The background image now appears behind the game board.



17 Add a **NinePatchRect** as a child to **HUD** and rename the node **RemainingPanel**.



New Concept: NinePatchRect

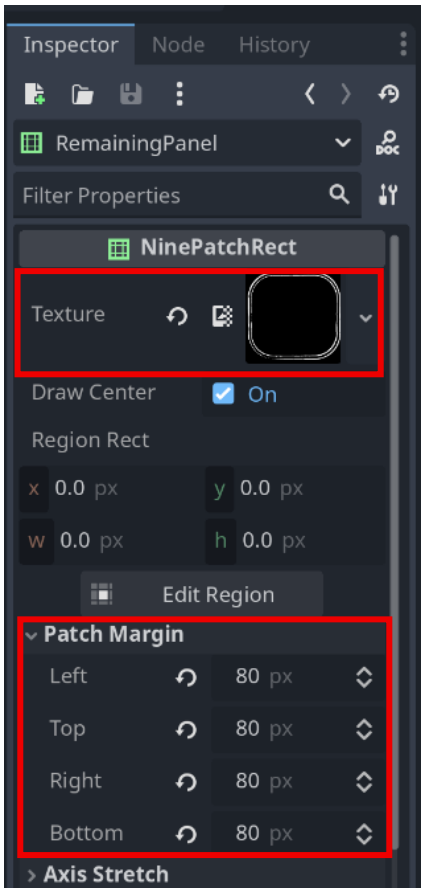
A **NinePatchRect** is a control node that displays a texture by keeping its corners intact but tiling the edges and center. The texture is split into a 3x3 grid. When scaling the node, the texture is tiled along center, vertical and horizonal axis. This node is useful when creating panels from a small texture.

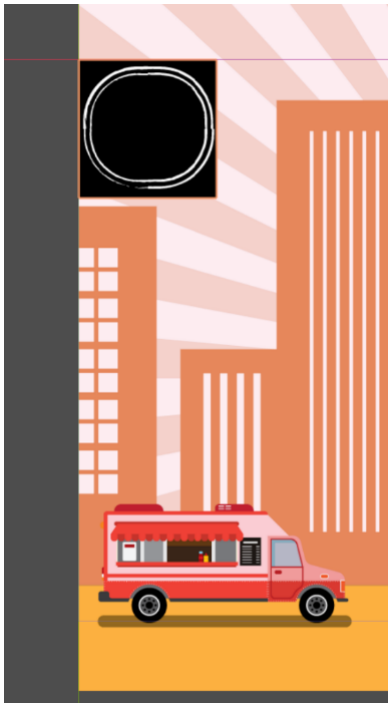
18 In the **Inspector** for **RemainingPanel**, use **Quick Load** to set the **Texture** to **UI_square.png**.

Then, under **Patch Margin**, set the **Left**, **Top**, **Right** and **Bottom** patch margins to **80**.

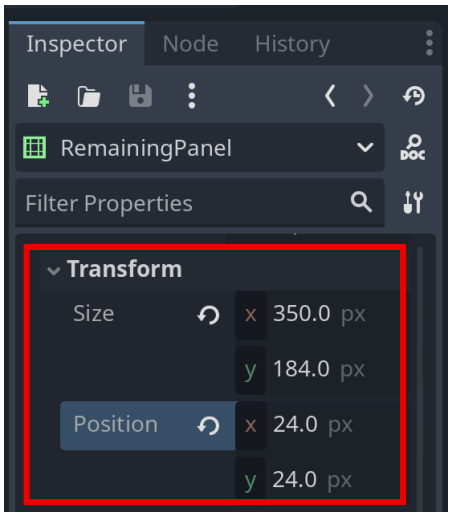
The Patch Margin is the width/heigh of the respective column/row.

The texture will begin to appear larger as the patch margins are updated.





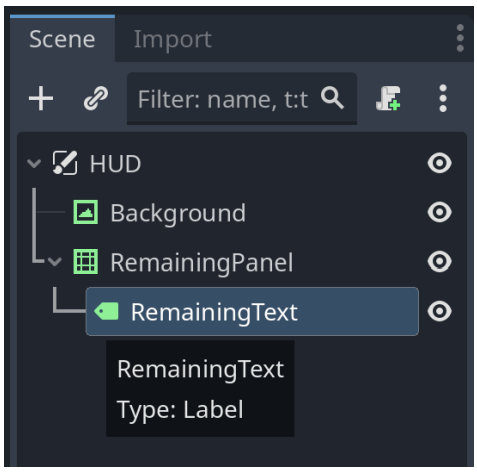
19 In Layout under **Transform**, set **Size** to **x: 350**, **y: 184** and **Position** to **x: 24, y: 24.**



The panel should appear larger with a small buffer to the edge of the canvas layer.

20 Add a **Label** as a child to **RemainingPanel** and rename the node to **RemainingText**.

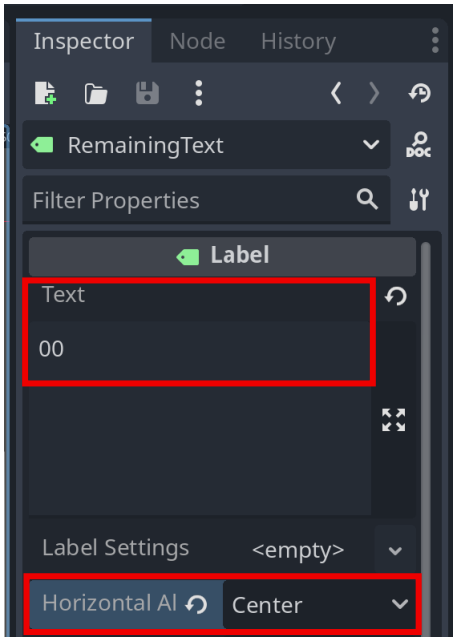
This node will show how many moves the player has left in the level.



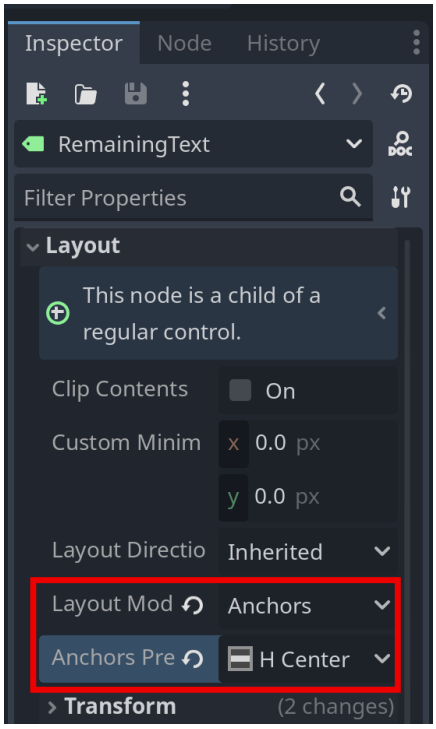
Reminder:

A **Label** node is a control for displaying plain text. Label nodes also have a bounding rectangle.

21 In the **Inspector** for **RemainingText**, set **Text** to **00** and **Horizontal Alignment** to **Center**.

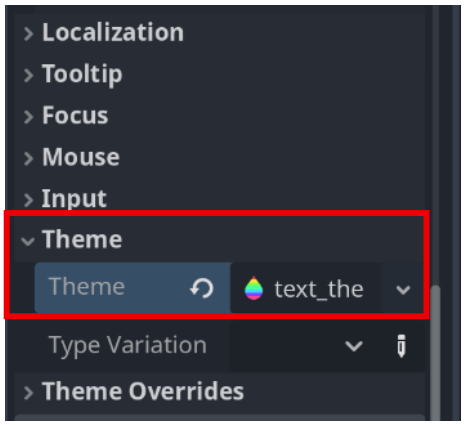


Then, under **Layout**, set **Layout Mode** to **Anchors** and **Anchors Preset** to **H Center Wide.**

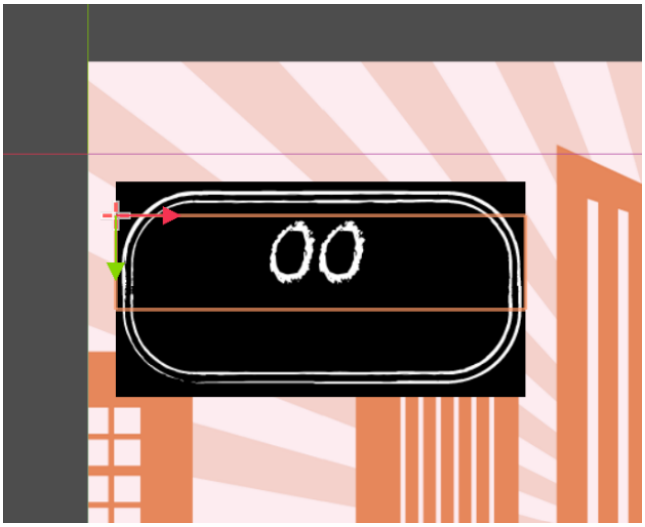


22 In the **Inspector** under **Theme**, use **Quick Load** to set **Theme** to **text_theme.tres**.

The **text_theme.tres** file contains the font, font size and font color that will be applied to the label text.

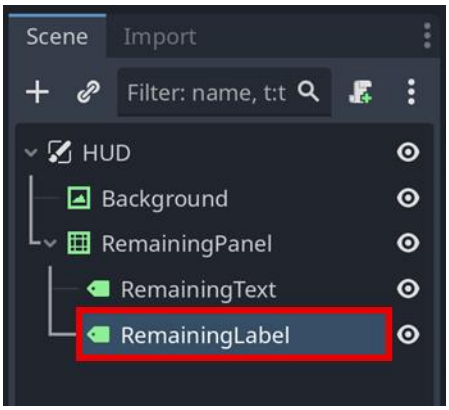


23 Adjust the label's position within the panel by holding down the **shift** key on the keyboard to lock the y axis, and use **Move Mode** to move the label towards the top of the panel.



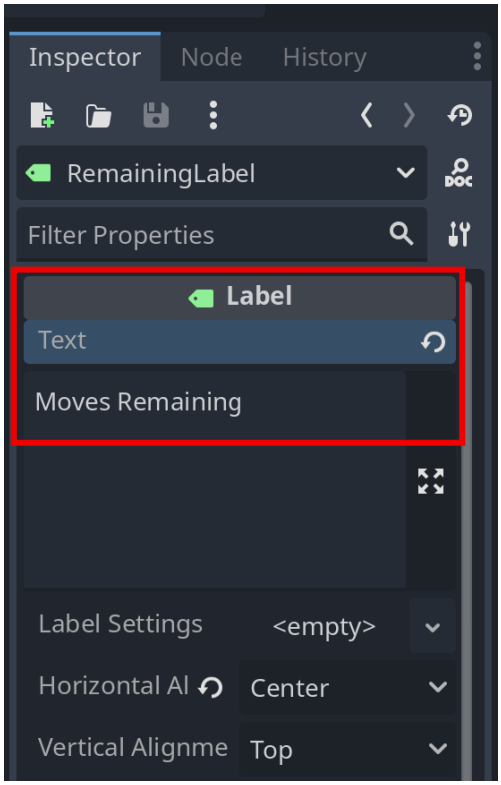
24 Duplicate RemainingText and rename the new node to RemainingLabel.

This node will label this component of the user interface.



25 In the **Inspector** for **RemainingLabel**, update **Text** to **Moves Remaining**.

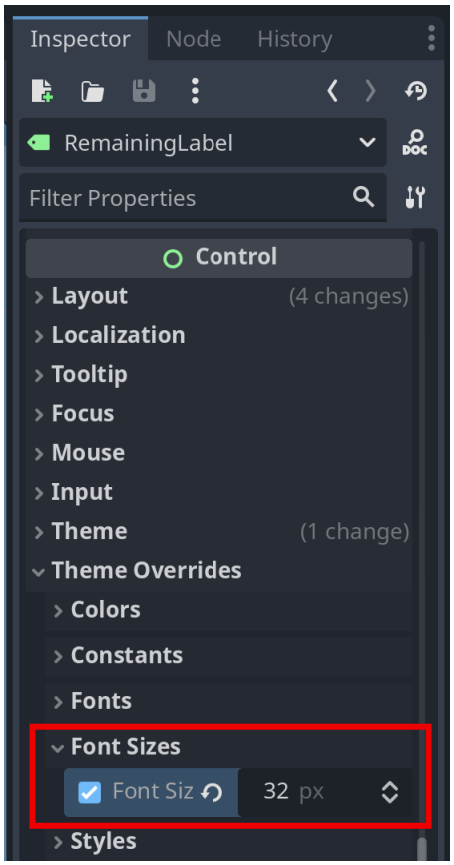
Initially, the text will appear too large for the panel.



26

In the Inspector under **Theme Overrides**, set **Font Size** to **32**.

In the 2D workspace, use Move Mode to adjust the label's position and move the text below the 00.



Pause for **Sensei Stop #1**!

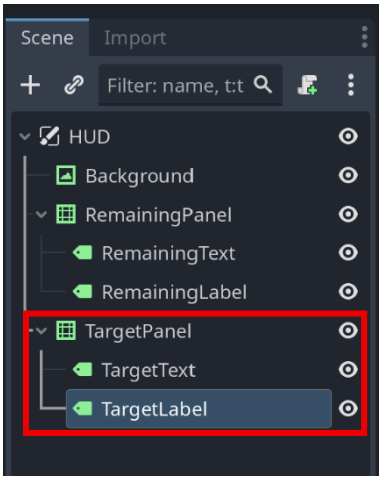
Check with a Code Sensei and confirm the **Background**, **RemainingPanel**, **RemainingText** and **RemainingLabel** are properly set up before continuing.

Reminder: Save your work!

27 Duplicate **RemainingPanel** and rename the new node **TargetPanel**.

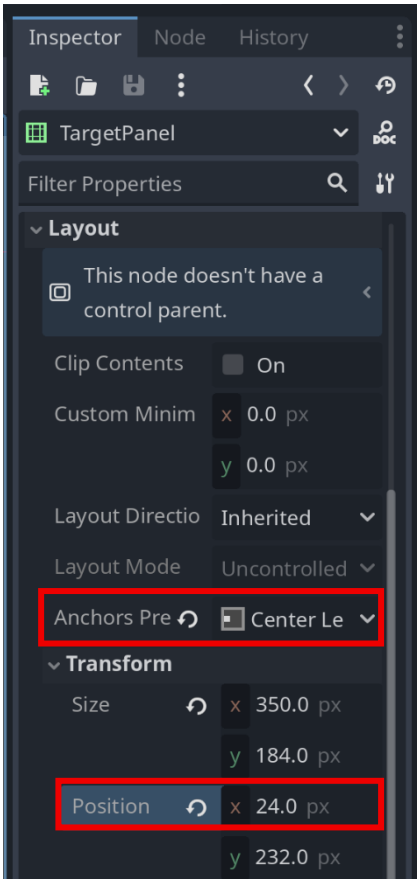
Then, rename **RemainingText** to **TargetText** and **RemainingLabel** to **TargetLabel**.

This panel will display a **target score** for the player to reach.



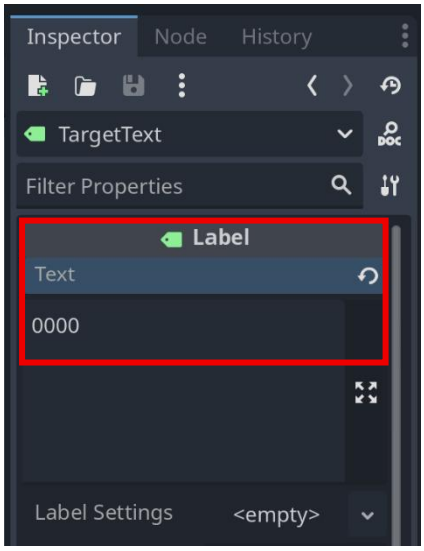
28 In the **Inspector** for **TargetPanel**, set **Anchors Preset** to **Center Left**.

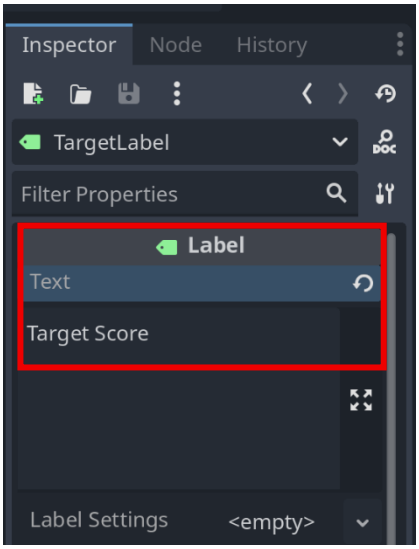
Then, set **Position x** to **24**.



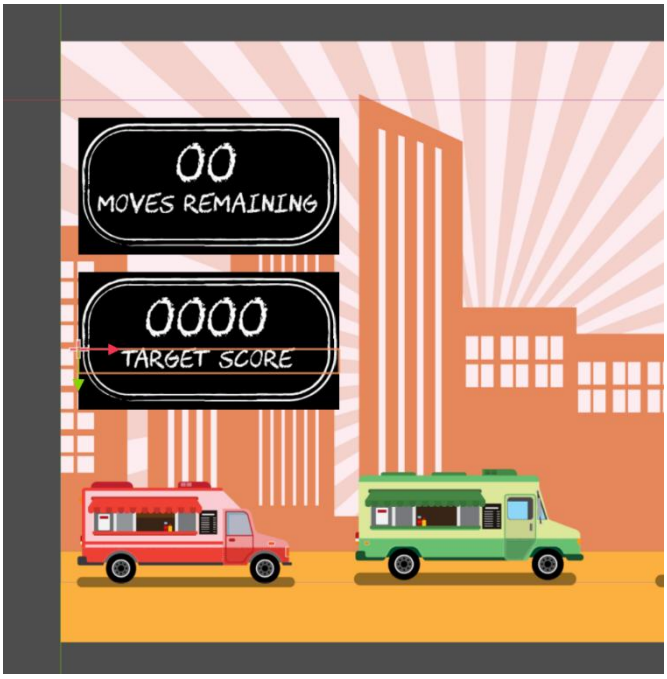
29 In the **Inspector** for **TargetLabel**, update **Text** to **Target Score**.

The **target score** displayed by the **TargetText** label will vary for different levels. The text will be updated later in the code, but some additional zeros can be added to the **TargetText Text** if desired.





30 The user interface should now display a panel for the remaining moves and a target score.

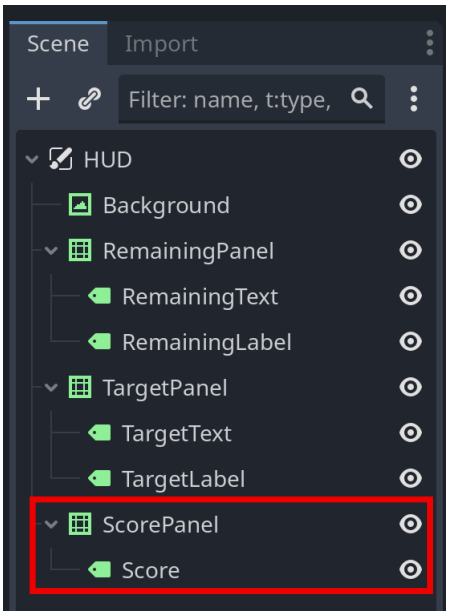


31 Duplicate **TargetPanel** and **rename** the new node **ScorePanel**.

Then, rename **TargetText** to **Score**.

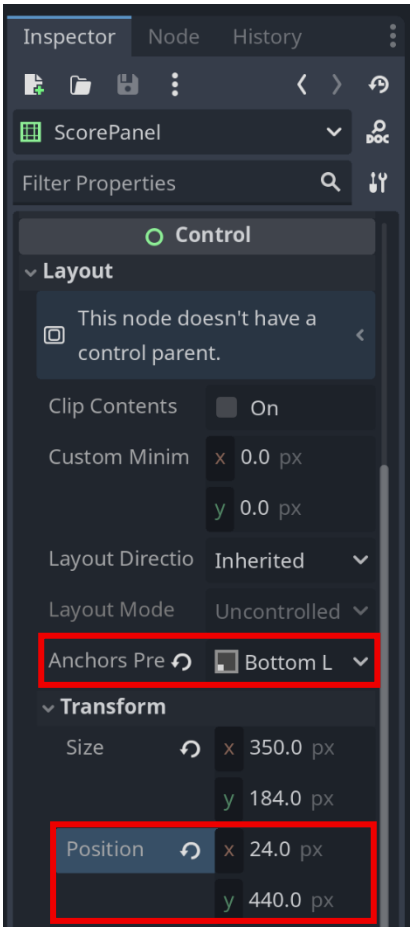
This panel will display the player's score and award stars based on that score.

ScorePanel only needs one Label node, Score, so the second Label node, **TargetLabel** can be **deleted**.

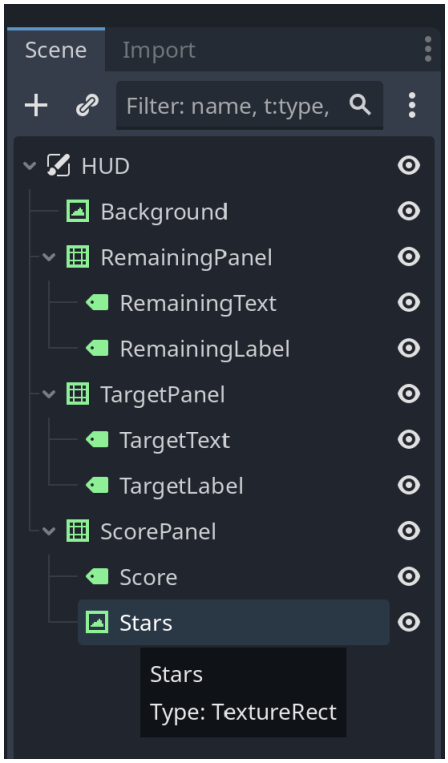


32 In the **Inspector** for **ScorePanel**, set **Anchors Preset** to **Bottom Left** and update **Position** to **x: 24, y: 440**.

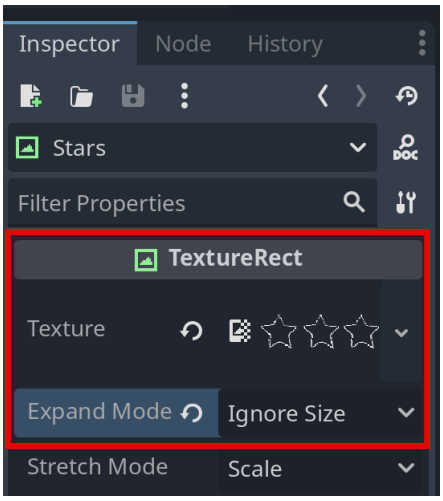
There should now be three panels equally spaced in the 2D workspace.

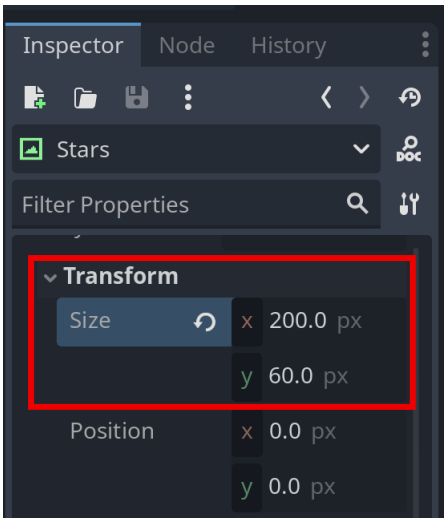


33 Add a **TextureRect** as a child to **ScorePanel** and rename the node to **Stars**.

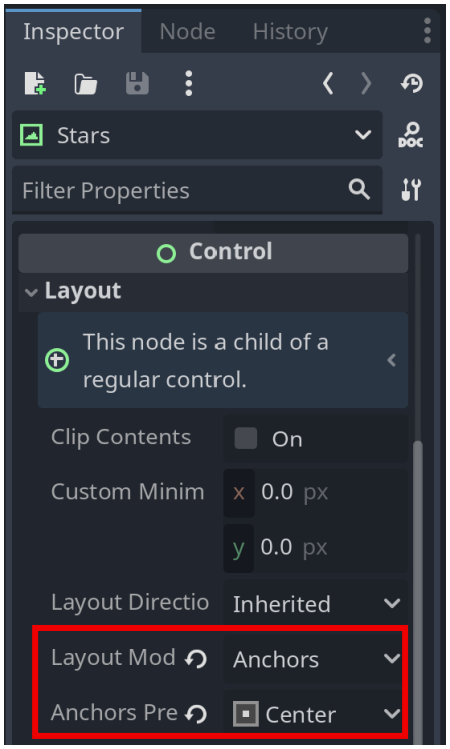


In the **Inspector** for **Stars**, set **Texture** to **stars_0.png** and **Expand Mode** to **Ignore Size,** then change **Size x** to **200** and **y** to **60**.

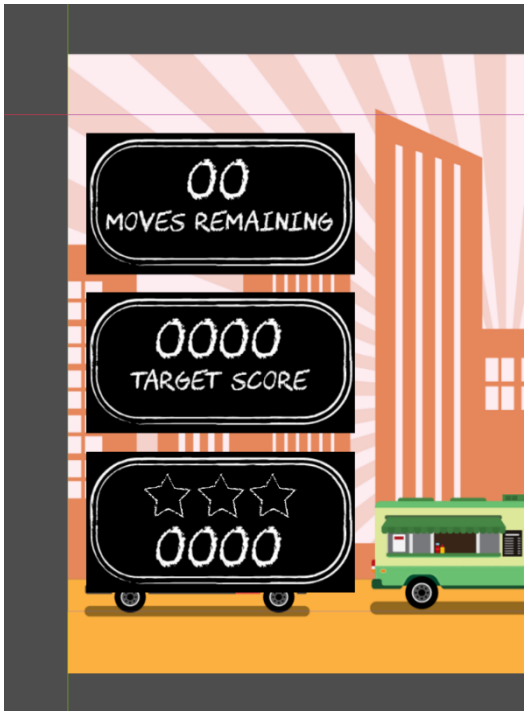




34 Set **Layout Mode** to **Anchors** and **Anchors Preset** to **Center**.



Then, holding the **shift** key on the keyboard to lock the y axis, use **Move Mode** to space out the star texture and score label.



35 Playtest the project to view how the HUD appears alongside the game board.

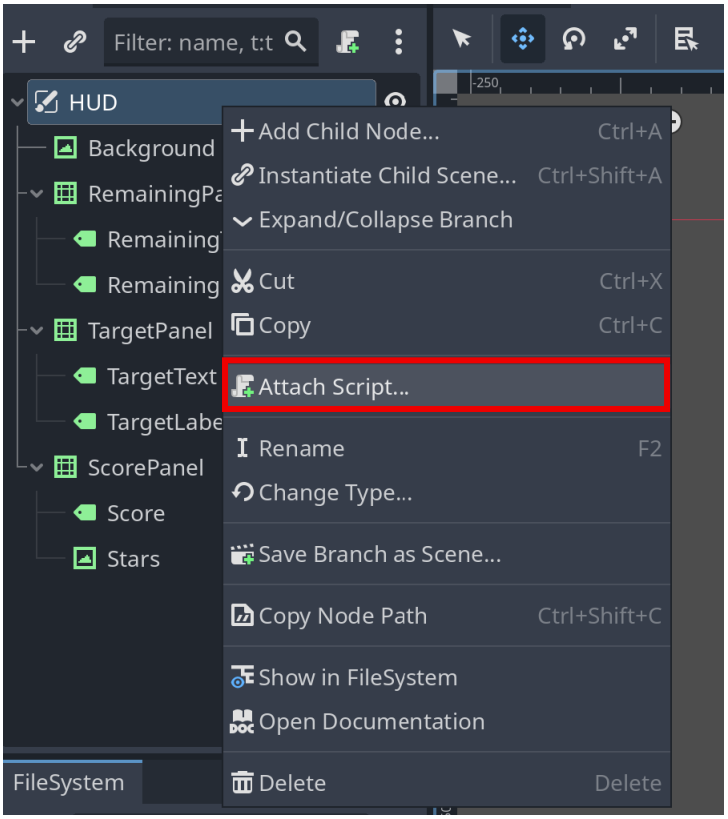
Code still needs to be added to update the HUD during gameplay.

Pause for **Sensei Stop #2**!

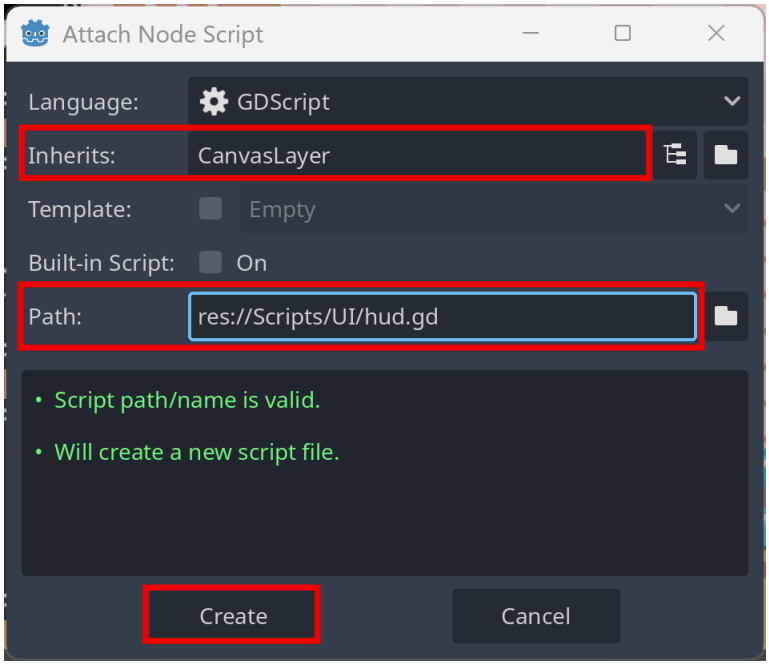
Check with a Code Sensei and confirm the **TargetPanel** and **ScorePanel** are properly set up before continuing.

Reminder: Save your work!

36 In **hud.tscn**, **right click** on the Root Node, **HUD**, and select **Attach Script**.

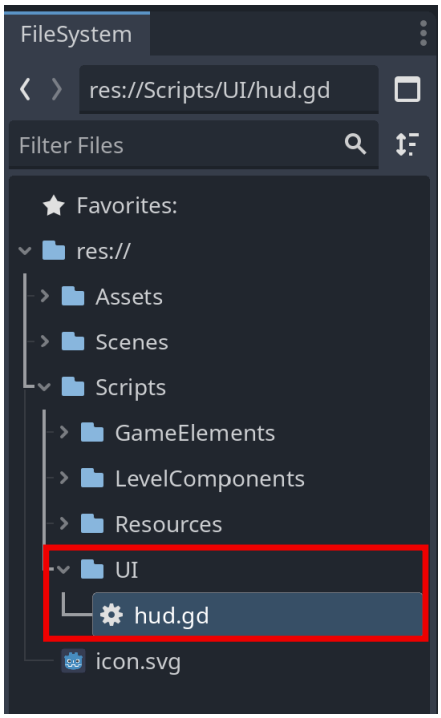


37 In the Attach Node Script window, check that **Inherits** is set to **CanvasLayer**. Update the **Path** to **res://Scripts/UI/hud.gd** and click **Create**.



Check that **hud.gd** script is in the **Scripts > UI folder**.

The script should attach itself to the HUD node automatically.



38 Underneath extends CanvasLayer, write a new line of code `class_name HUD`.

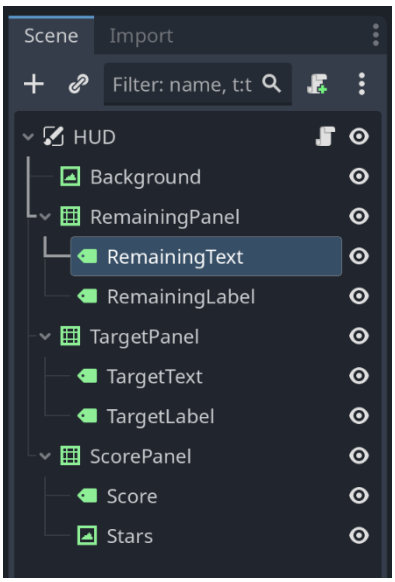
```
1   extends CanvasLayer
2   class_name HUD
3
4
```

New Concept: `class_name`

The keyword **`class_name`** defines the script as a globally accessible class with the specified name, **`HUD`**. This allows other scripts to extend the **`HUD`** class and access the functions and variables within it.

39 Drag the **RemainingText** node into the script editor, the hold **CTRL** on the keyboard and release the node to create its **`@onready`** variable.

Repeat for the remaining 4 label nodes, then update the score variable name to **score_text**.




```
1   extends CanvasLayer
2   class_name HUD
3
4
5
6     RemainingText
7
```

40

Then, underneath the variables, define a `set_remaining()` function, which updates the text in the remaining panel to show how many moves the player has left.

`set_remaining()`, needs one **parameter**, **remaining**, of **type String** and returns **void**. Inside the function, set **remaining_text** to **remaining**.

The **text property** will be helpful here.

```
3
4   @onready var remaining_text: Label = $RemainingPanel/RemainingText
5   @onready var remaining_label: Label = $RemainingPanel/RemainingLabel
6   @onready var target_text: Label = $TargetPanel/TargetText
7   @onready var target_label: Label = $TargetPanel/TargetLabel
8   @onready var score_text: Label = $ScorePanel/Score
9
10  # function: set_remaining() | parameters[1]: remaining (String) | return: void
11      # remaining_text.text = ???
```

41

Underneath the `set_remaining()` function, declare 2 more functions: `set_target()`, which updates the text in the target panel with a target score and `set_score()`, which updates the score panel with the players' score.

`set_target()` needs one parameter, `target`, of type String. Inside the function, set `target_text` to `target`.

`set_score()` needs one parameter, `score`, of type String. Inside the function, set `score_text` to `score`.

Both functions return void. The **text** property will be helpful here as well.

```
10  func set_remaining(remaining: String) -> void:
11      remaining_text.text = remaining
12
13  # function: set_target() | parameters[1]: target (String) | return: void
14      # ???
15
16  # function: set_score() | parameters[1]: score (String) | return: void
17      # ???
18
```

42 Check the code for all three functions and update as needed.

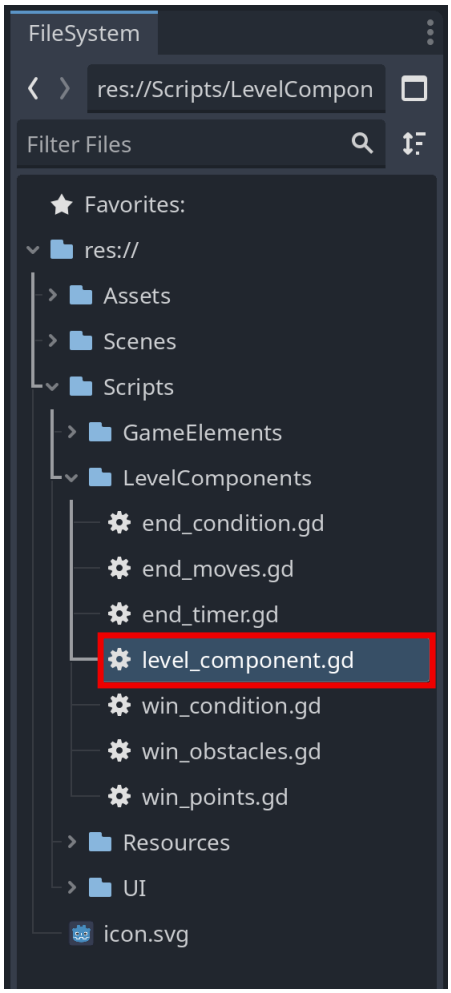
```
 9
10 func set_remaining(remaining: String) -> void:
11     remaining_text.text = remaining
12
13 func set_target(target: String) -> void:
14     target_text.text = target
15
16 func set_score(score: String) -> void:
17     score_text.text = score
```

Pause for **Sensei Stop #3**!

Check with a Code Sensei and confirm the label variables, and `set_remaining()`, `set_target()` and `set_score()` functions are properly set up before continuing.

Reminder: Save your work!

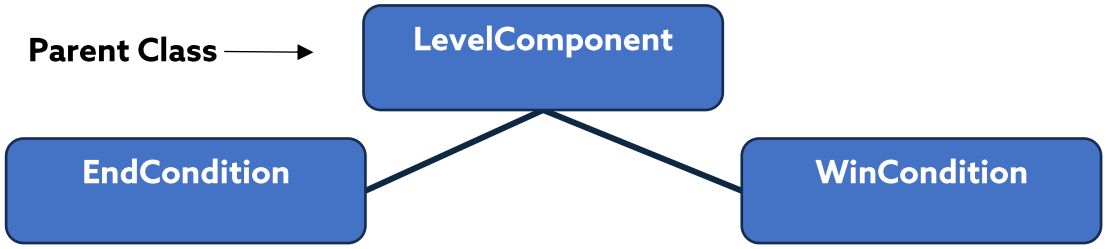
43 In **FileSystem**, find the **level_component.gd** script in the **Scripts > LevelComponents** folder and open the script.



44 Notice this script is a globally accessible script, like the hud.gd script. This is the parent class for the inheritance structure used in this project.

The `LevelComponent` class is directly inherited by two classes, `WinConditon` and `EndConditon`.

```
1   extends Node2D
2   class_name LevelComponent
3
4   var level: Level
5   var board: GameBoard
6   #------
7   # TODO 1: create hud variable
8   #------
9
10  func setup(_level: Level) -> void:
11      level = _level
12      board = _level.board
13      #------
14      # TODO 2: set hud variable
15      #------
```



45

Underneath **TODO 1**, create a variable `hud` of type **HUD**.

```
1   extends Node2D
2   class_name LevelComponent
3
4   var level: Level
5   var board: GameBoard
6   #------
7   # TODO 1: create hud variable
8   #------
9
```

The HUD type exists and can be used in the script because it was created in the hud.gd script.

```
1   extends CanvasLayer
2   class_name HUD
3
4
```

hud.gd script

46

Find **TODO 2** inside the `setup()` function and set `hud` to `_level.hud`.

This sets the `hud` variable inside the `LevelComponent` class to the `hud` **property** in the `Level` class when the `setup` function is called with an argument of type `Level`.

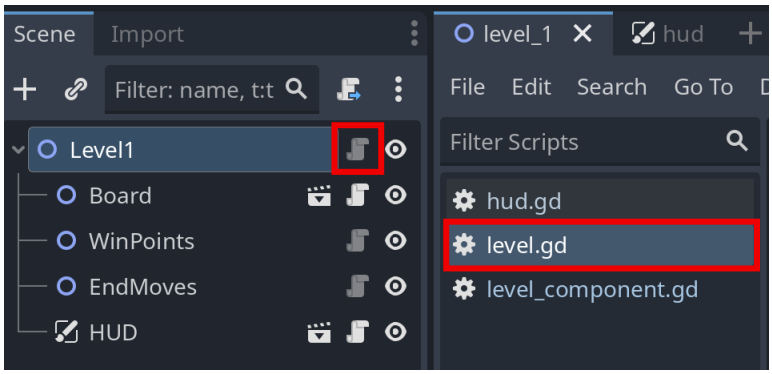
```
10
11  func setup(_level: Level) -> void:
12      level = _level
13      board = _level.board
14      #------
15      # TODO 2: set hud variable
16      #------
17
18
```

47 Check the code and update as needed.

```
4   var level: Level
5   var board: GameBoard
6   #------
7   # TODO 1: create hud variable
8   #------
9   var hud: HUD
10
11  func setup(_level: Level) -> void:
12      level = _level
13      board = _level.board
14      #------
15      # TODO 2: set hud variable
16      #------
17      hud = _level.hud
18
```

48 In **level_1.tscn**, click the script icon beside Level1 to open the **level.gd** script.

Notice the faded script icon. When a class is declared using `class_name`, Godot registers the class as a node type that can be created like any other built-in nodes. The script icon is faded because the Level1 node type is from a class defined in a user script.



49 Under **TODO 3**, use `@onready` to create the `hud` variable of type **HUD,** then use `$` and the **node path** to assign it to the HUD node in the Level1 scene.

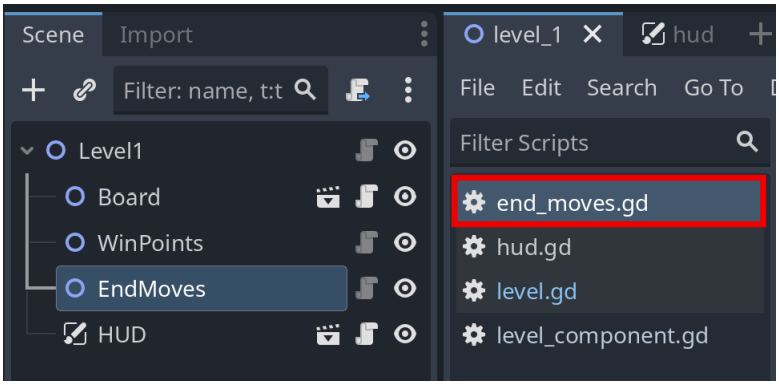
This creates the **hud** property used in the `setup()` function in the level_component.gd script. Notice the board variable and how its similarly used in the script.

```
1  extends Node2D
2  class_name Level
3
4  @onready var board: GameBoard = $Board
5  #------
6  # TODO 3: use @onready to create the hud variable
7  #------
8
```

What else can be noticed about the level_component.gd script?

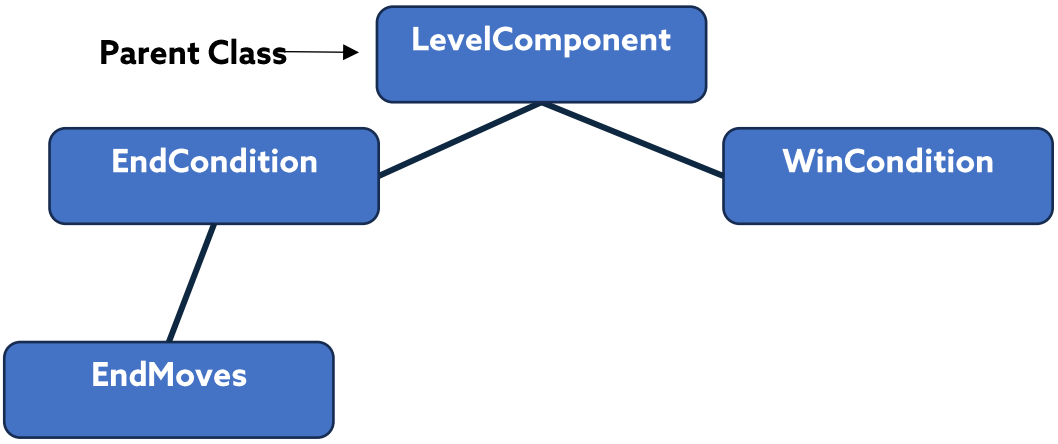
```
4  var level: Level
5  var board: GameBoard
6  #------
7  # TODO 1: create hud variable
8  #------
9  var hud: HUD
10
11 func setup(_level: Level) -> void:
12     level = _level
13     board = _level.board
14     #------
15     # TODO 2: set hud variable
16     #------
17     hud = _level.hud
18
```

50 In the **level_1.tscn** scene, click the script icon beside **EndMoves** to open the **end_moves.gd** script.



51 Notice this script extends EndCondition, which extends LevelComponent.

```
1  extends EndCondition
2  class_name EndMoves
3
4  @export var moves_left: int
5
6  func setup(_level: Level) -> void:
7      super(_level)
8      #------
```



This script, similarly to the level_compontent.gd script, contains a `setup()` function. The `super()` method with the argument `_level`, is called at the top of the function. Godot loads child scripts before parent scripts, which means the `setup()` function in the end_moves.gd script will run before the `setup()` function in the level_component.gd script.

```
4   var level: Level
5   var board: GameBoard
6   #------
7   # TODO 1: create hud variable
8   #------
9   var hud: HUD
10
11  func setup(_level: Level) -> void:
12      level = _level
13      board = _level.board
14      #------
15      # TODO 2: set hud variable
16      #------
17      hud = _level.hud
18
```

Calling `super()` at the start of the `setup()` function in the end_moves.gd script will call the `setup()` function in the parent script before executing the rest of the code in the `setup()` function of the end_moves.gd script.

New Concept: `super()`

The **`super()`** keyword is used to call a function or method in the parent class

52 Under TODO 4, call **`hud.set_remaining()`** and pass **`moves_left`** through as the functions' argument. The variable, **`moves_left`**, is of type **`int`** and needs to be converted to a String using the **`str()`** method before it's passed through the **`set_remaining()`** function.

This updates the RemainingText text to show the number of moves the player has using the **`set_remaining()`** function inside the hud.gd script at the start of the game.

The **`str()`** method converts one or more arguments to a String.

```
1   extends EndCondition
2   class_name EndMoves
3
4   @export var moves_left: int
5
6   func setup(_level: Level) -> void:
7       super(_level)
8       #------
9       # TODO 4: set remaining moves
10      #------
11      hud.set_remaining(str(moves_left))
12
```

53 Find TODO 5 inside the `on_move()` function and use the `set_remaining()` function to update the user interface to display the moves remaining after a move is made.

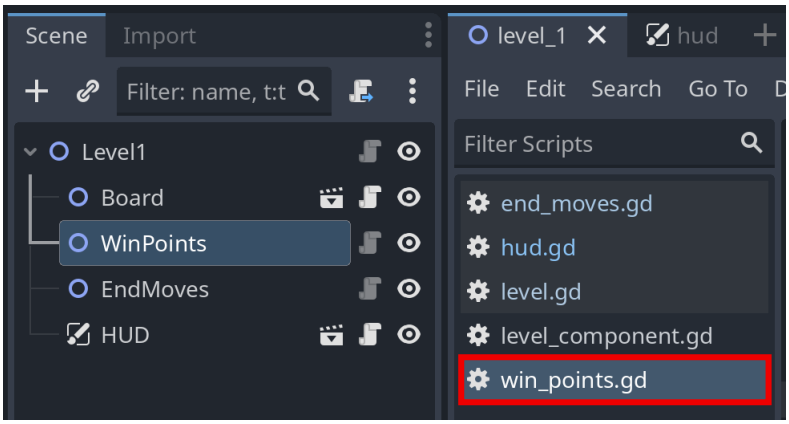
```
12
13 func on_move() -> void:
14     moves_left -= 1
15     #------
16     # TODO 5: update moves remaining
17     #------
18
19
20
```

54 Playtest the project. Does the moves remaining panel update?

55 Check the code for TODO 5 and update the script as needed.

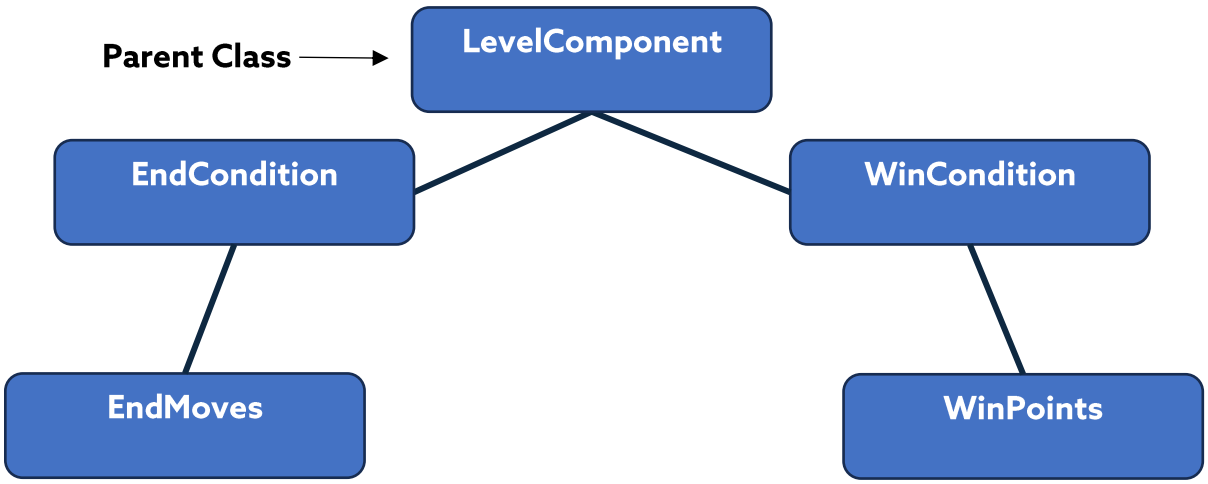
```
12
13 func on_move() -> void:
14     moves_left -= 1
15     #------
16     # TODO 5: update moves remaining
17     #------
18     hud.set_remaining(str(moves_left))
19
```

56 In **level_1.tscn**, click the script icon beside **WinPoints** to open the **win_points.gd** script.



Notice this script **extends WinCondition**, which **extends LevelComponent**, and also contains a `setup()` function that calls `super()`.

```
1   extends WinCondition
2   class_name WinPoints
3
4   @export var score_1_star: int
5   @export var score_2_star: int
6   @export var score_3_star: int
7
8   func setup(_level: Level) -> void:
9       super(_level)
10      #------
```



57 Find **TODO 6** inside the `setup()` function and use the `set_target()` function inside the **hud.gd** script to set the target score.

```
7
8  func setup(_level: Level) -> void:
9      super(_level)
10     #------
11     # TODO 6: set_target score
12     #------
13
```

58 Look at the score variables at the top of the script. How might one be passed through the function as an argument? Is the `str()` method helpful here? Playtest the project.

Does target score update at the start of the level? Does the score update as pieces are cleared?

59 Check the code for TODO 6 and update as needed.

```
7
8  func setup(_level: Level) -> void:
9      super(_level)
10     #------
11     # TODO 6: set_target score
12     #------
13     hud.set_target(str(score_1_star))
14
```

60 The score does not update as game pieces are cleared.

Return to the **level.gd** script and find **TODO 7** inside the `on_piece_cleared()` function.

Use the `set_score()` function inside the hud.gd script to update the score when a game piece is cleared.

What variable in the `on_piece_cleared()` function might be passed through `set_score()` as an argument?

ZIs the `str()` method helpful here?

```
38
39 func on_piece_cleared(piece: GamePiece):
40     win_condition.on_piece_cleared(piece)
41     end_condition.on_piece_cleared(piece)
42
43     score += piece.score_value
44     #------
45     # TODO 7: update score
46     #------
47
48
49     star_count = win_condition.get_star_count()
50
```

61 Playtest the project. Does the score update as pieces are cleared? Does anything happen when the target score is reached?

62 Check the code for TODO 7 and update as needed.

```
38 func on_piece_cleared(piece: GamePiece):
39     win_condition.on_piece_cleared(piece)
40     end_condition.on_piece_cleared(piece)
41
42     score += piece.score_value
43     #------
44     # TODO 7: update score
45     #------
46     hud.set_score(str(score))
47
```

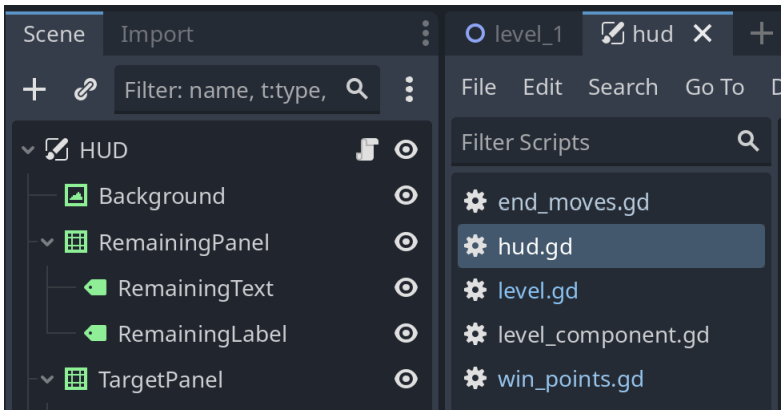
Pause for **Sensei Stop #4**!

Check with a Code Sensei and confirm the **Moves Remaining, Target Score** and **score** are updating during gameplay before continuing.

Reminder: Save your work!

63 The score now updates when game pieces are cleared but the star images do not change when their respective target scores are reached.

Return to the **hud.tscn** scene and open the **hud.gd** script.



64 In the hud.gd script, create 2 new variables:

- `star_image` of type `TextureRect` using `@onready` (can be dragged and dropped into the script editor like the previous `@onready` variables)
- `star_textures` of type `Array[Texture2D]` using `@export`. Assign the variable to an empty array.

```
4   @onready var remaining_text: Label = $RemainingPanel/RemainingText
5   @onready var remaining_label: Label = $RemainingPanel/RemainingLabel
6   @onready var target_text: Label = $TargetPanel/TargetText
7   @onready var target_label: Label = $TargetPanel/TargetLabel
8   @onready var score_text: Label = $ScorePanel/Score
9   # star_image
10
11  # star_textures
12
```

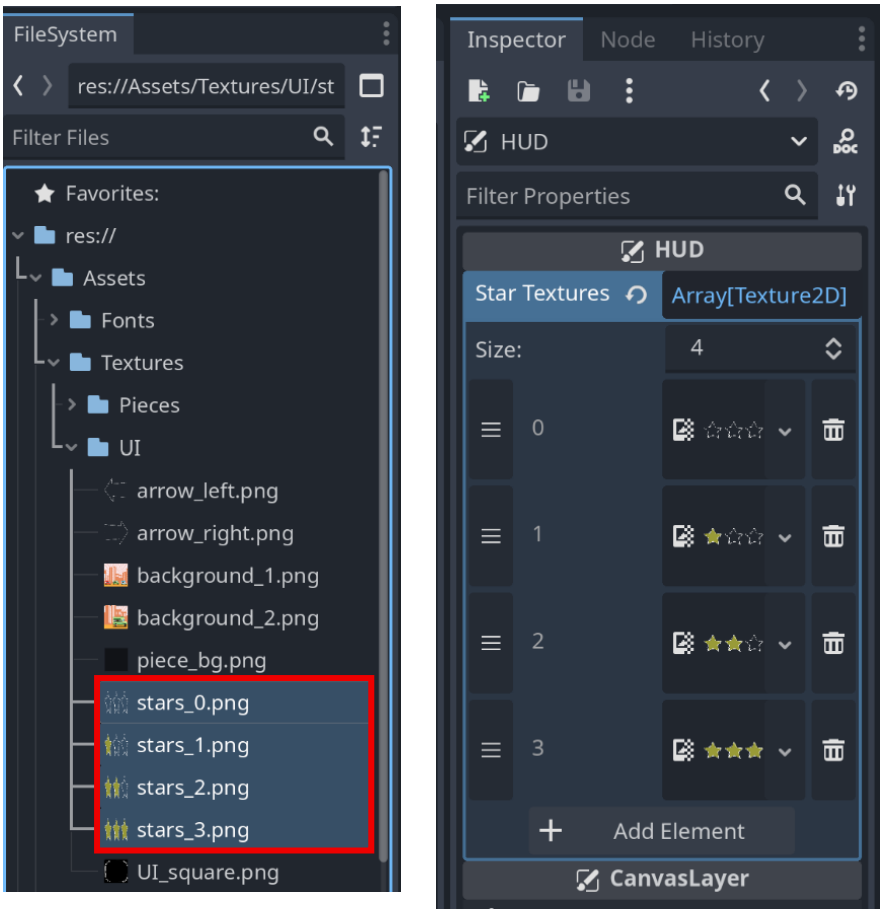
65 Check the code for the 2 new variables, update as needed, and save the script.

```
4   @onready var remaining_text: Label = $RemainingPanel/RemainingText
5   @onready var remaining_label: Label = $RemainingPanel/RemainingLabel
6   @onready var target_text: Label = $TargetPanel/TargetText
7   @onready var target_label: Label = $TargetPanel/TargetLabel
8   @onready var score_text: Label = $ScorePanel/Score
9   @onready var star_image: TextureRect = $ScorePanel/Stars
10
11  @export var star_textures: Array[Texture2D] = []
12
```

66 In **FileSystem**, find the 4 **star textures** in the **Assets > Textures > UI folder**. Hold down the shift key on the keyboard and select the 4 textures.

Click, drag then drop the 4 textures onto **Star Texture** in the **Inspector** for **HUD**.

Click **Array[Texture2D]** to expand the array and view the textures.



67 In the hud.gd script, underneath the `set_score()` function, declare a function, `set_stars()`, which updates the `star_image` texture with elements from the `star_textures` array.

`set_stars()`, needs one **parameter**, `star_index`, of type `int`, and returns `void`. Inside the function, set `star_image` to `star_index`, then save the script.

The `texture` property and `star_textures` array will be helpful here.

```
20      score_text.text = score
21
22  # function: set_stars() | parameters[1]: star_index (int) | return: void
23      # star_image.??? = star_textures[???]
24
```

68 Check the code for the `set_stars()` function and update as needed.

```
22 func set_stars(star_index: int) -> void:
23     star_image.texture = star_textures[star_index]
24
```

69 Return to the **level.gd** script and find **TODO 8** inside the `on_piece_cleared()` function.

Use the `set_stars()` function to update the stars image after the score has been updated.

What variable inside the `on_piece_cleared()` function might be passed through `set_stars()` as an argument?

```
38 func on_piece_cleared(piece: GamePiece):
39     win_condition.on_piece_cleared(piece)
40     end_condition.on_piece_cleared(piece)
41
42     score += piece.score_value
43     #------
44     # TODO 7: update score
45     #------
46     hud.set_score(str(score))
47
48     star_count = win_condition.get_star_count()
49
50     #------
51     # TODO 8: update stars
52     #------
53
54
```

70 Playtest the **level_one** scene.

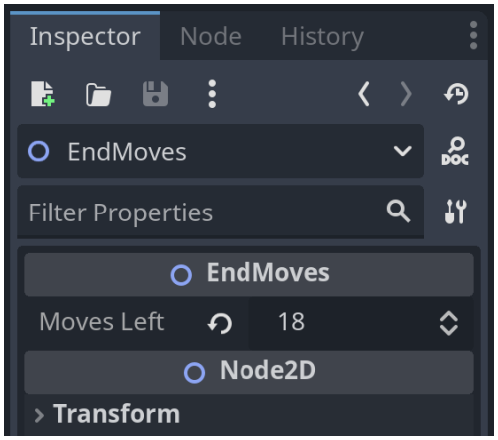
Does 1 star appear when the target score of 5000 is reached? Do 2 stars appear at 7000? Do all 3 stars appear when the score becomes larger than 10000?

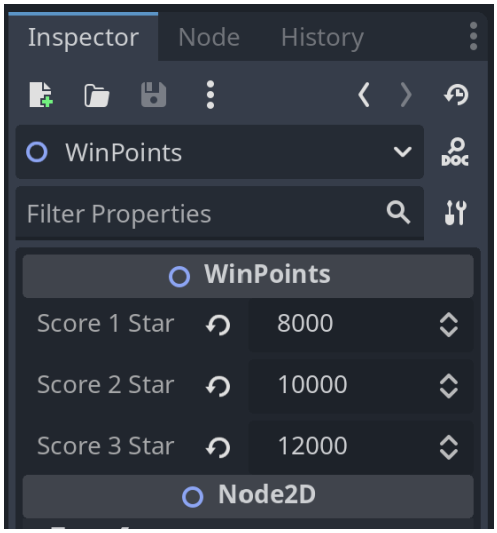
71 Check the code for TODO 8 and update as needed.

```
38  func on_piece_cleared(piece: GamePiece):
39      win_condition.on_piece_cleared(piece)
40      end_condition.on_piece_cleared(piece)
41
42      score += piece.score_value
43      #------
44      # TODO 7: update score
45      #------
46      hud.set_score(str(score))
47
48      star_count = win_condition.get_star_count()
49
50      #------
51      # TODO 8: update stars
52      #------
53      hud.set_stars(star_count)
54
```

72 The level can be customized by adjusting the **Moves Left** variable in the Inspector for **EndMoves**, and the **Score** variables in the Inspector for **WinPoints**.

When adjusting the values, be sure to keep the level challenging, but not impossible.





Pause for **Sensei Stop #5**!

Check with a Code Sensei and confirm the star images are updating during gameplay before continuing.

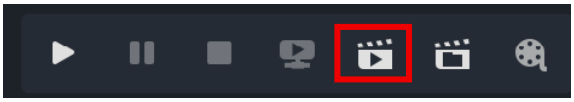
Reminder: Save your work!

73 In **FileSystem**, find the **level_2.tscn** scene in the Scenes > Levels folder and open the scene.

Drag the **hud.tscn** scene into the **Level2** scene.

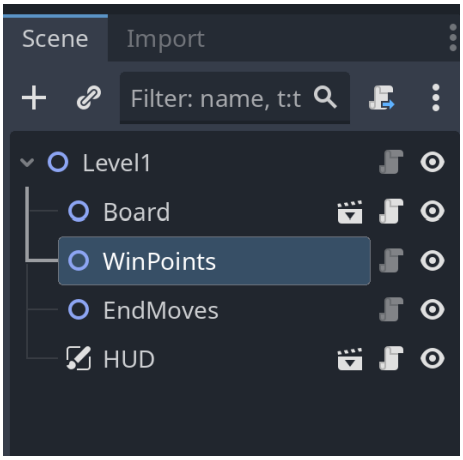
74 Use the **Run Current Scene** button to run the level_2.tscn scene.

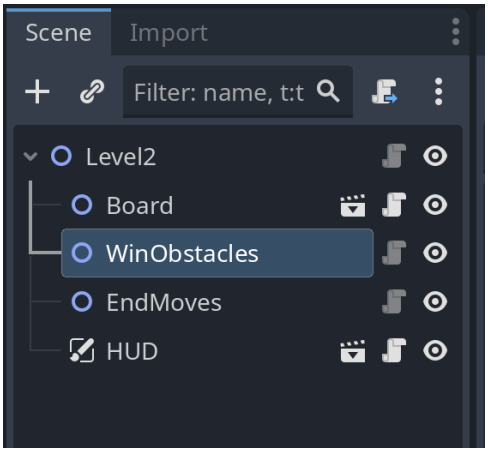
What can be noticed about this level? What can be noticed about the HUD?



75 The level 1 objective is to reach the target score within a set number of moves. However, the objective for level 2 is to clear all the obstacles within a set number of moves.

Notice the two levels have different win conditions.





76 The panel labels need to update depending on the win and end condition for each level.

Return to the **hud.gd** script and create a new function **`set_label_titles()`.**

`set_label_titles()` needs one **parameter**, **`level`**, of **type** **`Level`** and returns **`void`**.

```
22 func set_stars(star_index: int) -> void:
23     star_image.texture = star_textures[star_index]
24
25 # function: set_label_titles() | parameters[1]: level (Level) | return: void
26     #
27
28
29
30
31
```

77 Inside the **`set_label_titles()`** function, check if **`level.win_condition`** is the **`WinPoints`** class using the keyword **`is`**, then update the **`target_label`** text as needed.

```
25 func set_label_titles(level: Level) -> void:
26     if level.win_condition is WinPoints:
27         # target_label.text = " ??? "
28
```

Reminder: `is` (keyword)

Tests whether a variable extends a given class or is of a given built-in type.

78 Check the code and update as needed.

Underneath the if statement, use **`elif`** to check if **`level.win_condition`** is the **`WinObstacles`** class then update the **`target_label`** text as needed.

```
24
25  func set_label_titles(level: Level) -> void:
26      if level.win_condition is WinPoints:
27          target_label.text = "Target Score"
28      # elif ??? is ???:
29          # ??? = ???
30
```

79 Check the code and update as needed.

Inside the function, create a new **`if-elif`** statement that checks if **`level.end_condition`** is **`EndMoves`** or **`EndTimer`** (used in level 3) and update the **`remaining_label`** text as needed.

```
25  func set_label_titles(level: Level) -> void:
26      if level.win_condition is WinPoints:
27          target_label.text = "Target Score"
28      elif level.win_condition is WinObstacles:
29          target_label.text = "Dishes Remaining"
30
31      # if level.end_condition is EndMoves:
32          # ??? = "Moves Remaining"
33      # elif ??? is EndTimer:
34          # ??? is "Time Left"
35
```

80

Check the code and update as needed.

```
25 func set_label_titles(level: Level) -> void:
26     if level.win_condition is WinPoints:
27         target_label.text = "Target Score"
28     elif level.win_condition is WinObstacles:
29         target_label.text = "Dishes Remaining"
30
31     if level.end_condition is EndMoves:
32         remaining_label.text = "Moves Remaining"
33     elif level.end_condition is EndTimer:
34         remaining_label.text = "Time Left"
35
```

81

Return to the **level.gd** script and find **TODO 9** inside the `_ready()` method.

Call the `set_label_titles()` function with the argument `self` to set the label titles based on the level's `win_condition` and `end_condition`.

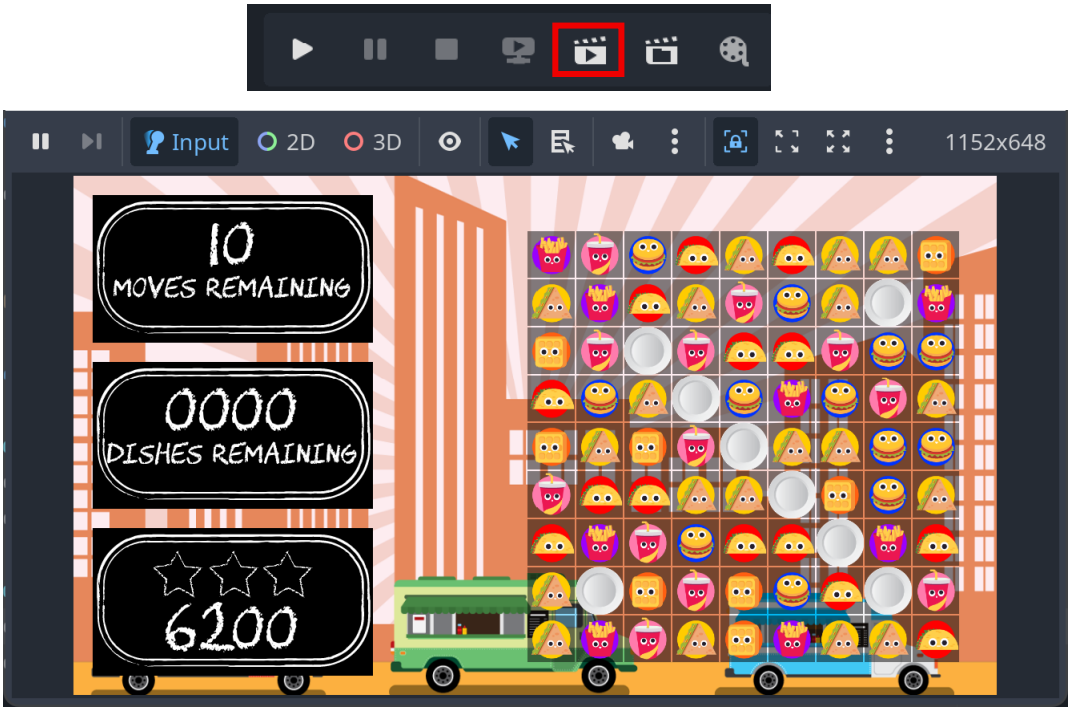
```
22
23 func _ready() -> void:
24     for child in get_children():
25         if child is WinCondition:
26             win_condition = child
27         elif child is EndCondition:
28             end_condition = child
29     if not win_condition or not end_condition:
30         push_error("Level needs both win condition and end condition children to function.")
31
32     win_condition.setup(self)
33     end_condition.setup(self)
34     #------
35     # TODO 9: set labels
36     #------
37     hud.set_label_titles(self)
38
```

New Concept: `self`

The `self` keyword is used to refer to the current instance of the class the script is attached to.

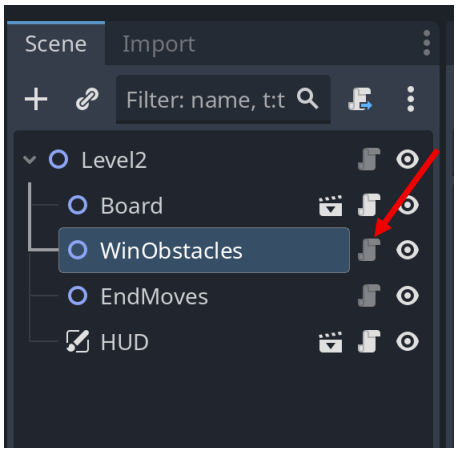
82 Playtest the level_2.tscn scene. Do the labels update?

Are there other parts of the UI that are not updating?



83 The number of obstacles to clear needs to be set and updated in the HUD.

Click on the script icon beside **WinObstacles** to open the **win_obstacles.gd** script.



84 Notice the script **extends WinConditon**, which **extends LevelComponent**.

The script only contains 2 score variables: `score_2_star` and `score_3_star`. This is because a star is rewarded if all the obstacles are cleared within the set number of moves.

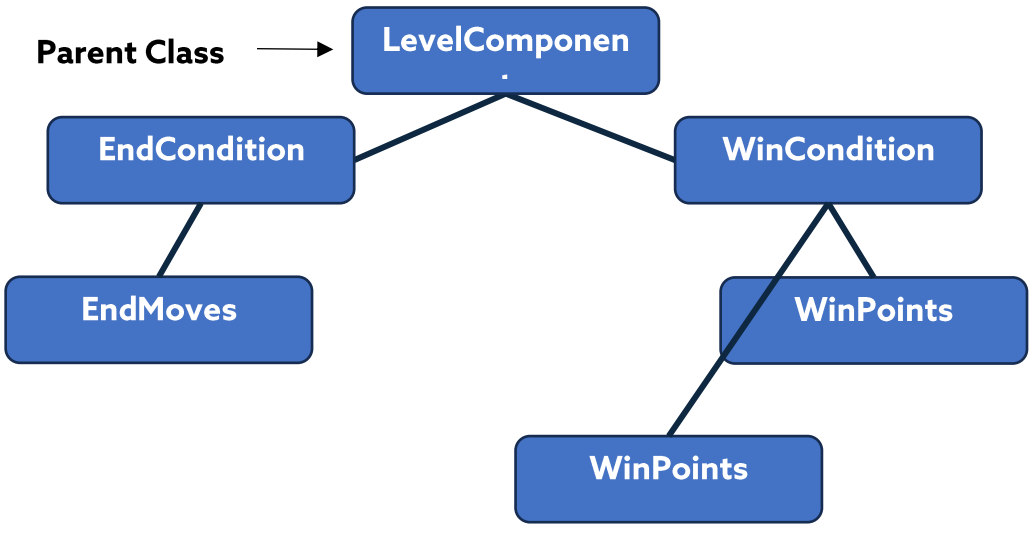
```
1  extends WinCondition
2  class_name WinObstacles
3
4  @export var score_2_star: int
5  @export var score_3_star: int
6
7  var obstacles_left: int
8
```

85 Notice the `super()` method called at the top of the `setup()` function.

Inside the `setup()` function find **TODO 10** and use the `set_target()` function to set the number of obstacles to be cleared.

What variable might be passed though as an argument? Is the `str()` method helpful here?

```
 8
 9  func setup(_level: Level) -> void:
10      super(_level)
11
12      var all_pieces: Array[GamePiece] = board.get_all_pieces()
13      obstacles_left = 0
14      for piece in all_pieces:
15          if piece.is_obstacle():
16              obstacles_left += 1
17      #------
18      # TODO 10: set obstacles left
19      #------
20
```

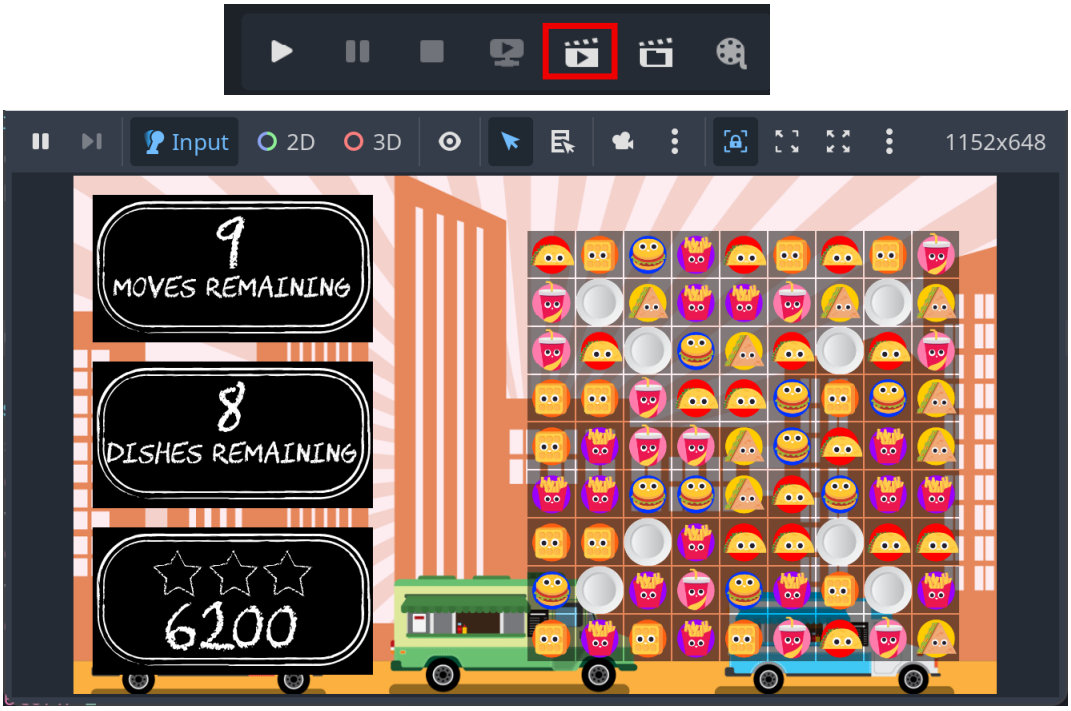


86 Inside the `on_piece_cleared()` function, find **TODO 11** and update the number of obstacles remaining after one has been cleared.

```
17      #------
18      # TODO 10: set obstacles left
19      #------
20      hud.set_target(str(obstacles_left))
21
22  func on_piece_cleared(piece: GamePiece) -> void:
23      if piece.is_obstacle():
24          obstacles_left -= 1
25          #------
26          # TODO 11: update obstacles left
27          #------
28
29
```

87 Playtest the **level_2.tscn** scene.

Does the number of dishes remaining decrease as the obstacles are cleared?



88

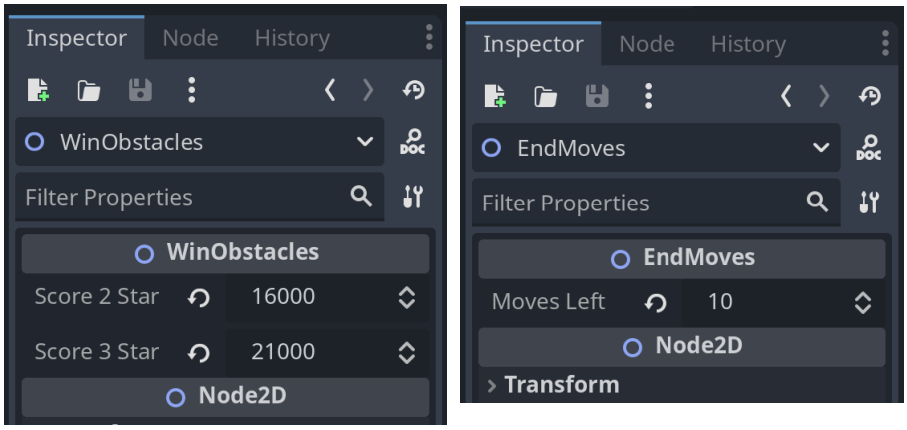
Check the code for TODO 10 and 11 and update as needed.

```
 9 func setup(_level: Level) -> void:
10     super(_level)
11
12     var all_pieces: Array[GamePiece] = board.get_all_pieces()
13     obstacles_left = 0
14     for piece in all_pieces:
15         if piece.is_obstacle():
16             obstacles_left += 1
17     #------
18     # TODO 10: set obstacles left
19     #------
20     hud.set_target(str(obstacles_left))
21
22 func on_piece_cleared(piece: GamePiece) -> void:
23     if piece.is_obstacle():
24         obstacles_left -= 1
25         #------
26         # TODO 11: update obstacles left
27         #------
28         hud.set_target(str(obstacles_left))
29
```

89

The level can be customized by adjusting the **Moves Left** variable in the Inspector for **EndMoves**, and the **Score** variables in the Inspector for **WinObstacles**.

When adjusting the values, be sure to keep level challenging, but not impossible.



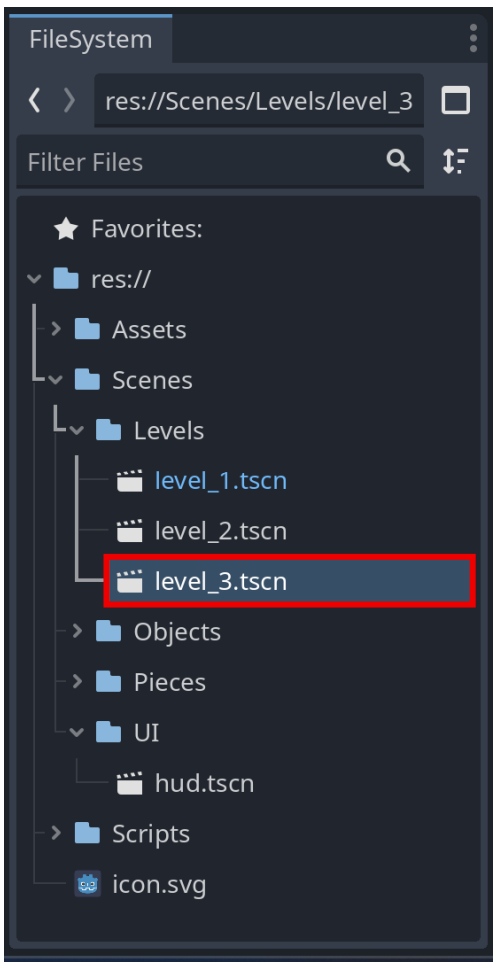
Pause for **Sensei Stop #6**!

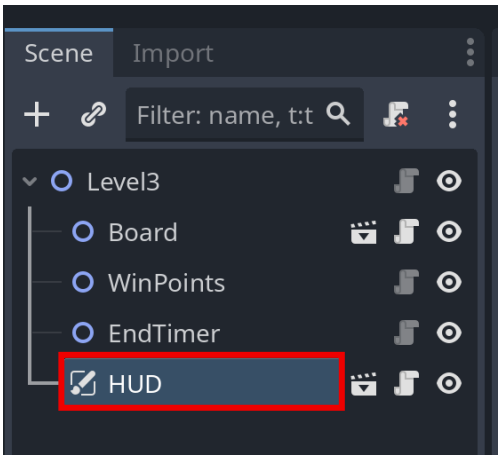
Check with a Code Sensei and confirm the **level labels** and **dishes remaining** are updating during gameplay before continuing.

Reminder: Save your work!

90 In **FileSystem**, find the **level_3.tscn** scene in the **Scenes > Levels** folder and open the scene.

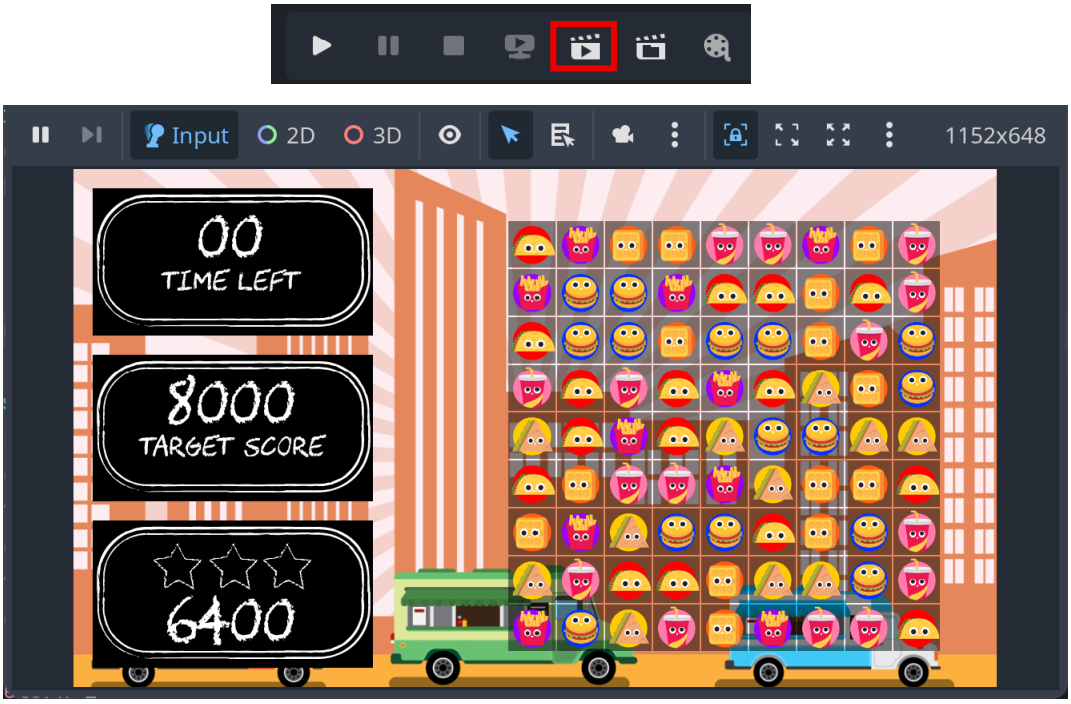
Drag the **hud.tscn** scene into the **Level3** scene.





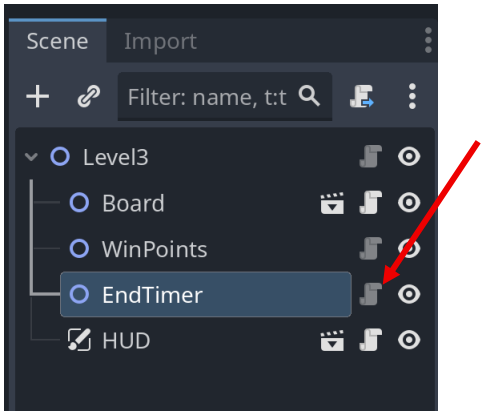
91 Playtest the level_3.tscn scene. What can be noticed about the level?

What can be noticed about the UI?



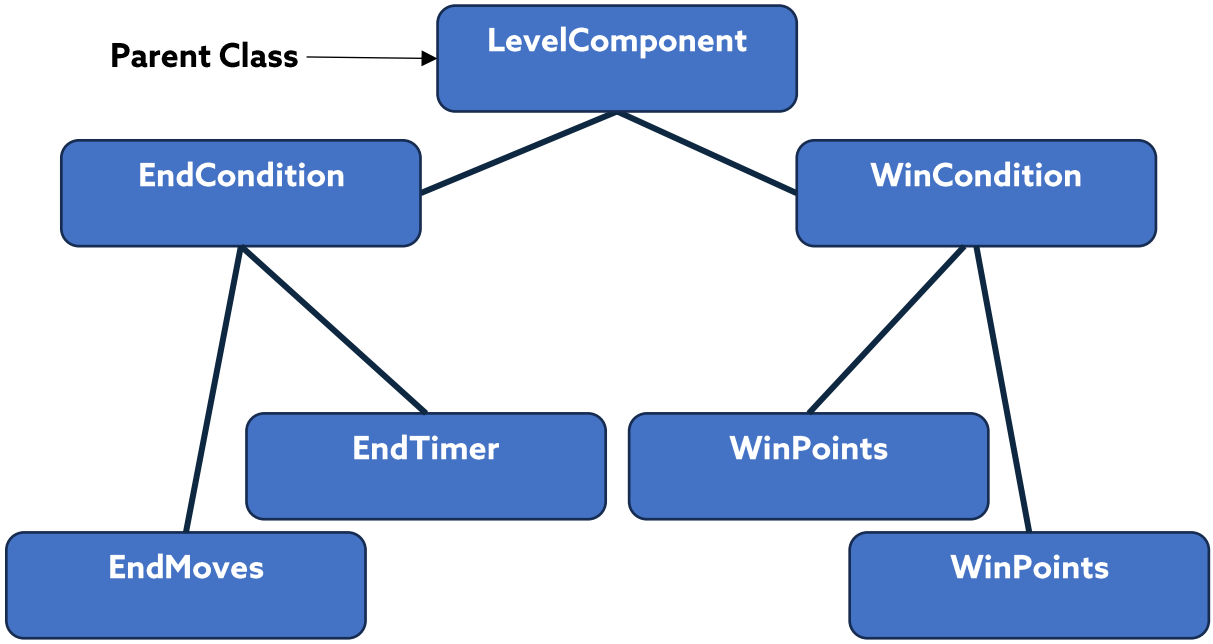
92 Unlike levels 1 and 2, which had an end condition built around the number of moves, level 3 has an end condition which uses a time limit.

Click on the script icon beside **EndTimer** to open the **end_timer.gd** script.



Notice the script **extends EndConditon**, which **extends LevelComponent**.

```
1  extends EndCondition
2  class_name EndTimer
3
4  @export var time_remaining: float
5
```

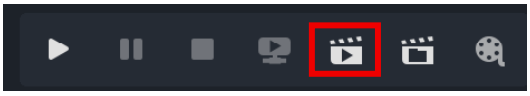


93 Scroll down to **TODO 12** and use the `set_remaining()` function to update the time remaining.

Notice there is a function `_format_time()` in the script, which returns the time left as a **String**. How might this function be used to update the time under TODO 12? Will the `str()` method be needed here?

```
2   class_name EndTimer
3
4   @export var time_remaining: float
5
6   func _process(delta: float) -> void:
7       time_remaining -= delta
8       if time_remaining < 0:
9           time_remaining = 0
10
11      #------
12      # TODO 12: update time left
13      #------
14
15
16  func _format_time(time: float) -> String:
17      return "%d:%02d" % [time / 60, int(time) % 60]
18
```

94 Playtest the level. Do all parts of the UI update?

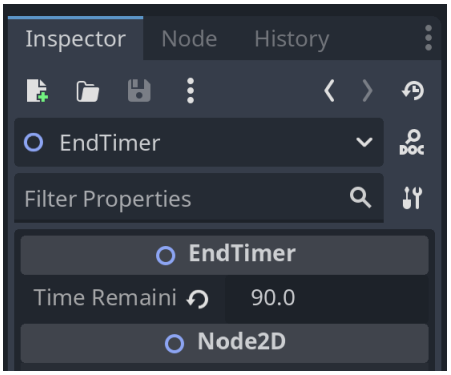


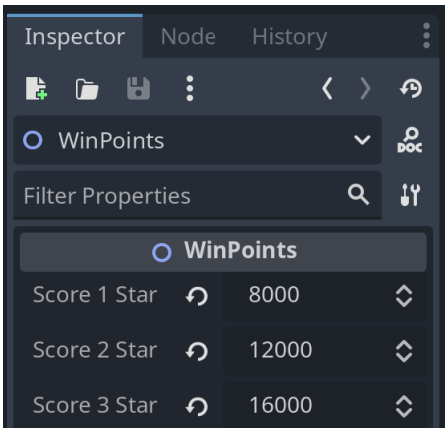
95 Check the code for TODO 12 and update as needed.

```
6  func _process(delta: float) -> void:
7      time_remaining -= delta
8      if time_remaining < 0:
9          time_remaining = 0
10
11     #------
12     # TODO 12: update time left
13     #------
14     hud.set_remaining(_format_time(time_remaining))
15
16 func _format_time(time: float) -> String:
17     return "%d:%02d" % [time / 60, int(time) % 60]
18
```

96 The level can be customized by adjusting the **Time Remaining** variable in the Inspector for **EndTimer**, and the **Score** variables in the Inspector for **WinPoints**.

When adjusting the values, be sure to keep the level challenging, but not impossible.





97 The game over UI will be built out later in Food Frenzy Part 2, but a game over message can be printed in the console to show whether the game has been won or lost.

Return to the **hud.gd** script and create two new functions at the bottom of the script: `on_game_win()` and `on_game_lose()`.

Both functions take **no parameters** and **return void.** Inside each function, use the `print()` method to let the player know if the game was won or lost, then save the script.

```
31    if level.end_condition is EndMoves:
32        remaining_label.text = "Moves Remaining"
33    elif level.end_condition is EndTimer:
34        remaining_label.text = "Time Left"
35
36# function: on_game_win() | parameters: none | return: void
37    # print("???")
38
39# function: on_game_lose() | parameters: none | return: void
40    # print("???")
41
```

98 In the **level.gd** script, find **TODO 13** inside the `_end_game()` function.

Underneath TODO 13, write an if-else statement to check if the win condition was reached and the game was won using the `game_won()` function in the **WinCondition** class, which can be accessed using the `win_condition` variable.

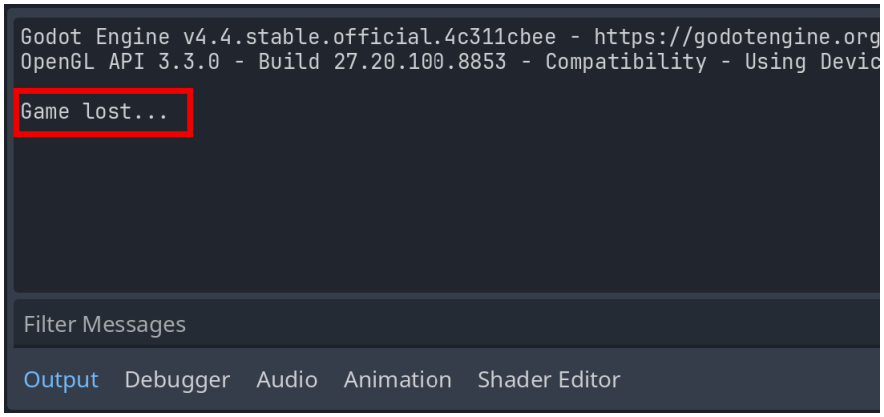
Call the `on_game_win()` and `on_game_lose()` functions in the hud.gd script as needed.

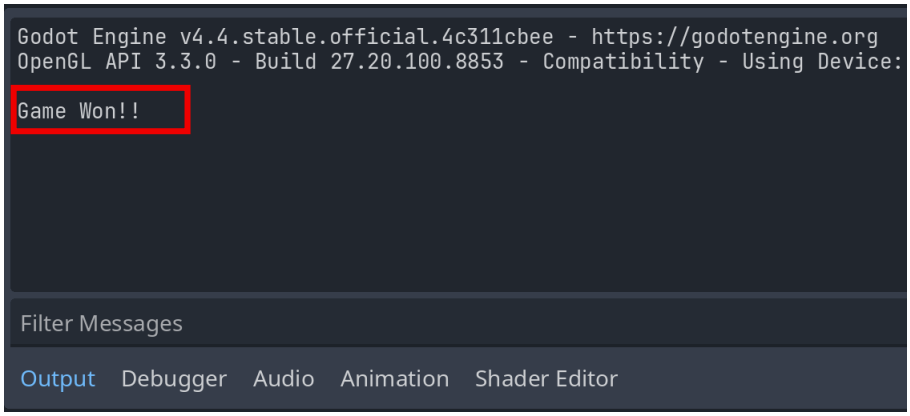
```
71 func _end_game():
72     game_over = true
73     board.game_over = true
74     board.visible = false
75
76     #------
77     # TODO 13: check if the game is won or lost
78     #------
79     # if win_condition.game_won():
80     #     # ???
81     # else:
82     #     # ???
83
```

99 Playtest a level to check that the `on_game_win()` and `on_game_lose()` functions are called.

It may be helpful to adjust the values for the win condition (**WinPoints**) or end condition (**EndMoves** or **EndTimer**) to make winning or losing the game easier.

When the game is won or lost, a message should appear in the Output console.





100 Check the code and update as needed.

hud.gd script

```
35
36 func on_game_win() -> void:
37     print("Game Won!!")
38
39 func on_game_lose() -> void:
40     print("Game lost...")
41
```

level.gd script

```
75
76     #------
77     # TODO 13: check if the game is won or lost
78     #------
79     if win_condition.game_won():
80         hud.on_game_win()
81     else:
82         hud.on_game_lose()
83
```

Pause for **Sensei Stop #7**!

Check with a Code Sensei and confirm the **time left** is decreasing during gameplay and the **game over functions** are called when the game ends.

Reminder: Save your work!

Discuss the following with a Code Sensei:

- Why might an HUD (heads up display) provide useful during gameplay?
- When might a Control node be used as the root node of a UI scene instead of a CanvasLayer?
- Why might inheritance and custom classes be useful when creating something like enemies or powerups?