CODE NINJAS®

Silver Belt Ninja Guide

Activity 08: World of Color

REPLAYABILITY

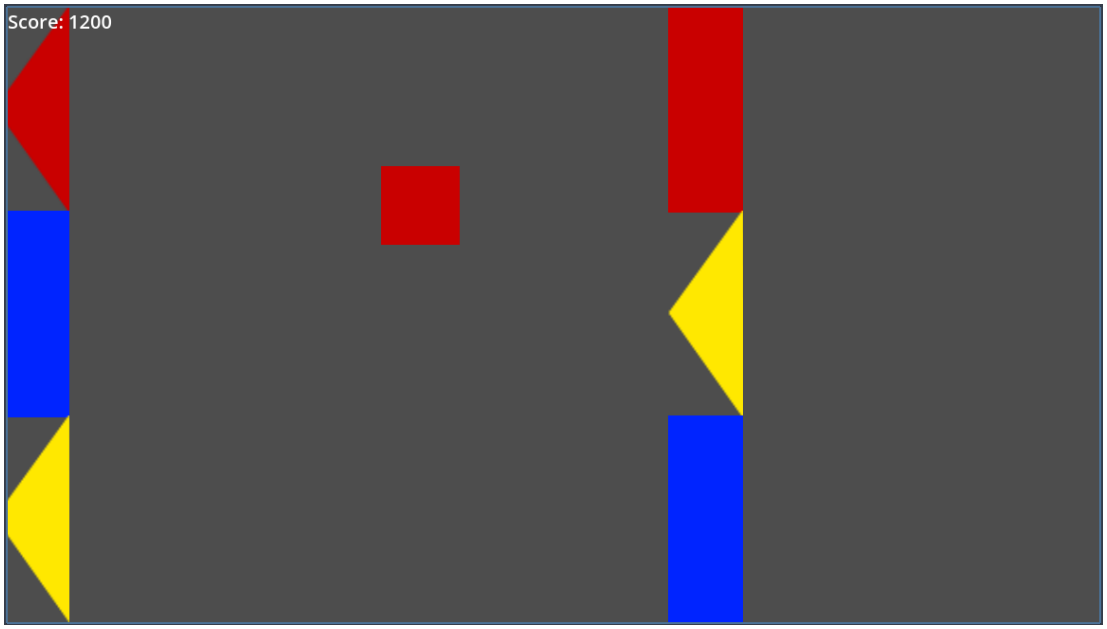
Replayability is the number of times a game can be played while still giving the player a new experience each time. Games are often judged on how replayable they are, and players are drawn to a game that they can enjoy over and over again. Sometimes replayability is represented by giving the player different choices each time they play. Another way it's used is to make the game a challenge for the player to overcome, and to try to improve each time – this is an approach that many classic games have taken.

When coming up with a way to get players to replay a game, consider these questions:

- What makes the player want to try a game that they've already played again?
- How might replayability be added to a game without the code becoming too complicated?
- What are the benefits of adding this idea to a game?

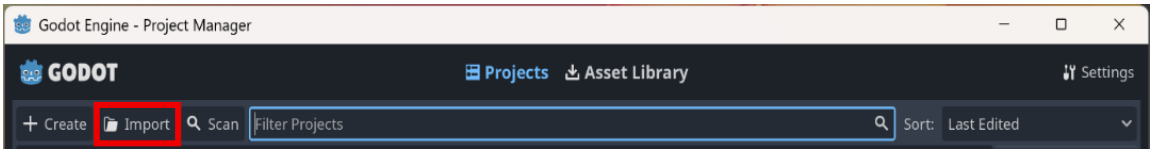
Activity 08: World of Color

In this project, you'll use arrays and combine random elements from them to make a never-ending series of changing obstacles that the player must navigate to reach as high a score as possible. You'll also use particle effects to make elements of the game look more exciting to players. By the end of this activity, you will have learned new ways to combine things you've used – arrays and randomness – to make a new type of game, and how to incorporate some new visual effects into your games.



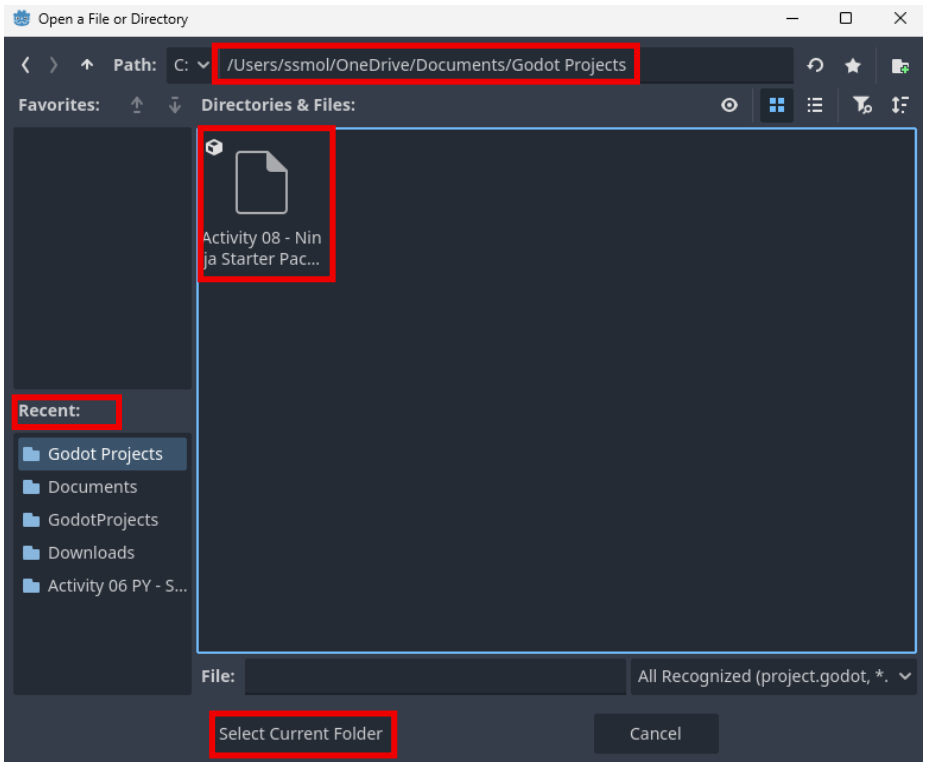
1 Begin the project by **importing** the starter code. This approach maintains the Input Map settings and other properties in the project file.

Open Godot and click **Import.**



2 In the File Directory, navigate to the file path using the **Path** text field or by finding it under **Recent**.

Select **SB Activity 08 – Ninja Starter Pack.zip** and click **Open**.

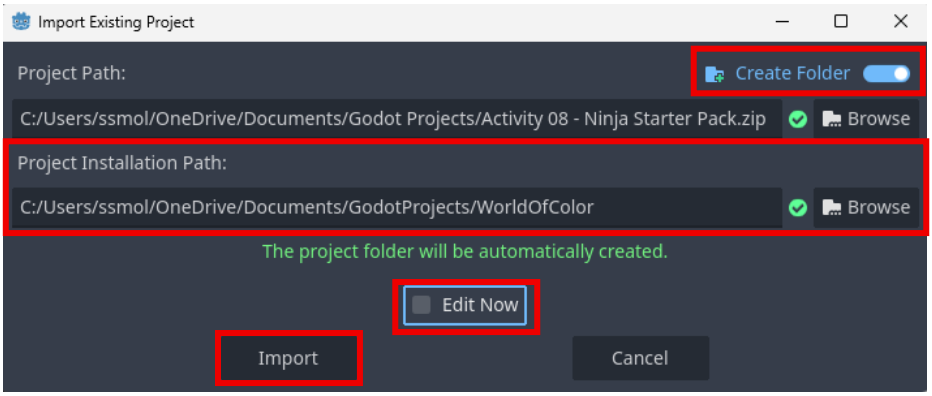


Pro Tip:

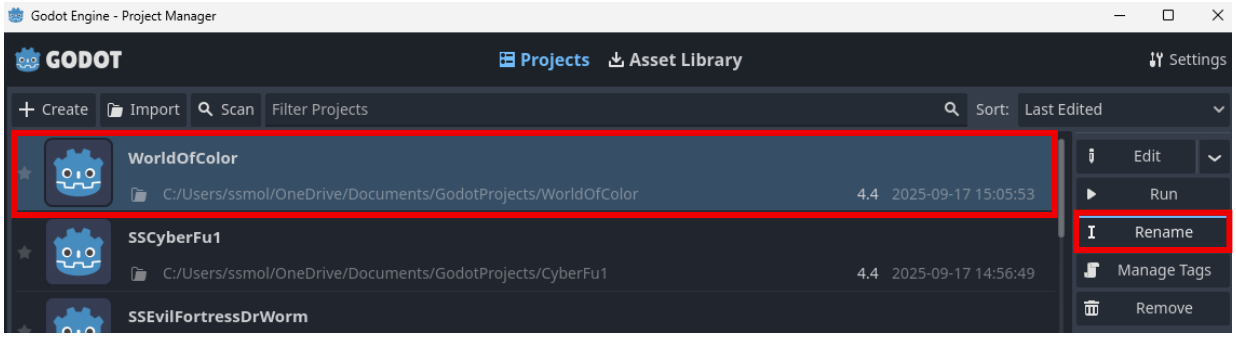
The folder should be **zipped** and must contain a **project.godot** file to be properly imported.

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

Uncheck Edit Now and click **Import**.

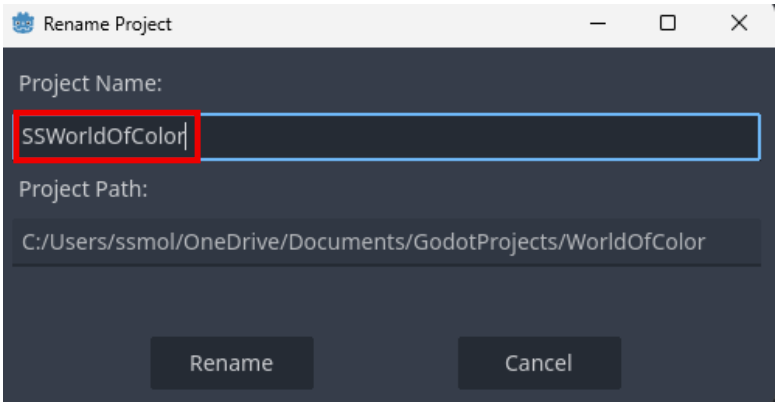


4 The project will appear at the top. Click on the project and select **Rename** on the right.



5 Update the Project Name to [YourInitials]WorldOfColor.

Do not click Rename until after the Sensei stop!

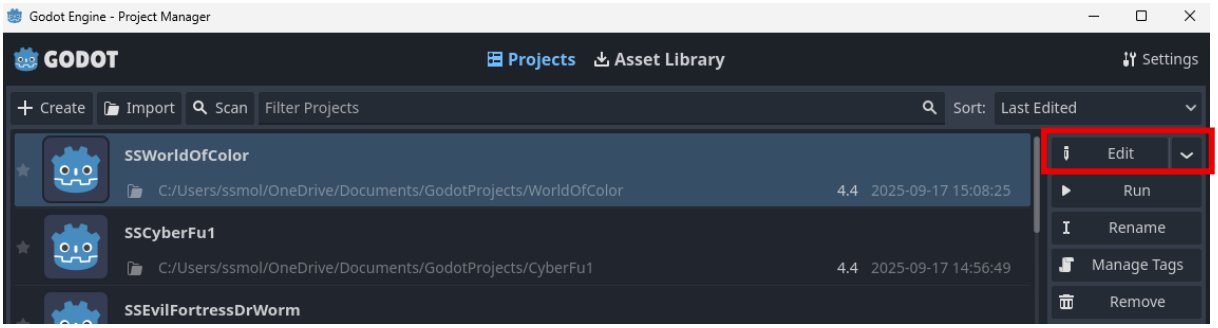


Pause for **Sensei Stop #1**!

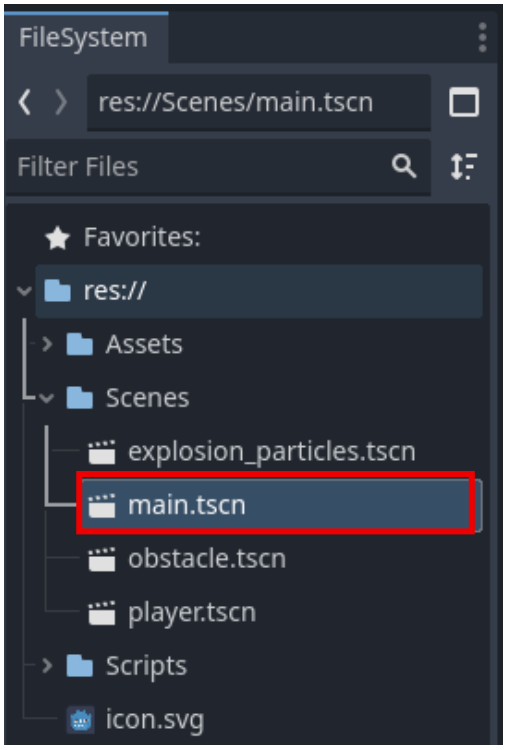
Check in with a Code Sensei before moving on. Make sure the **project import path** is correct, and the project **has been renamed.**

Click **Rename** when complete.

6 Select the project and click **Edit** to re-open the starter code.

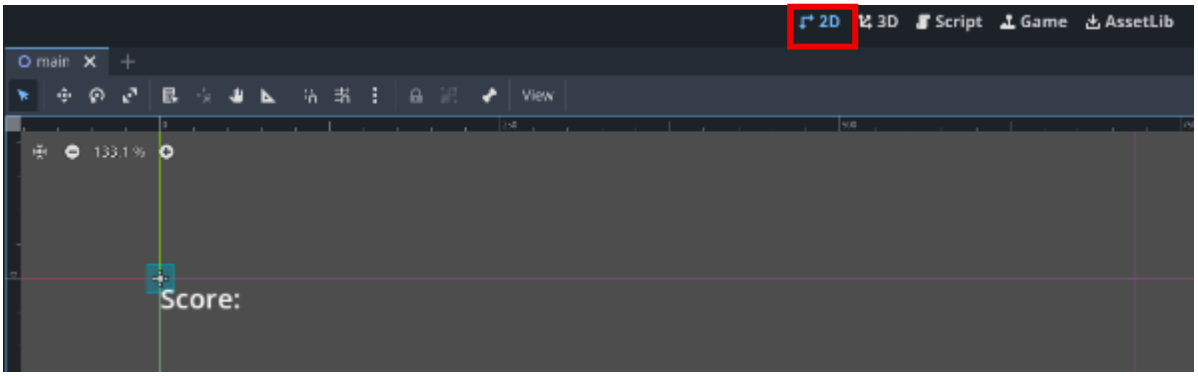


7 In **FileSystem**, navigate to **main.tscn** and double-click to open the scene.



Toggle to the **2D** workspace. Notice that only the **"Score:"** text is present so far.

Begin the project by spawning obstacles to appear from the right side of the screen.



8 In **FileSystem**, open the **game_manager.gd** script.

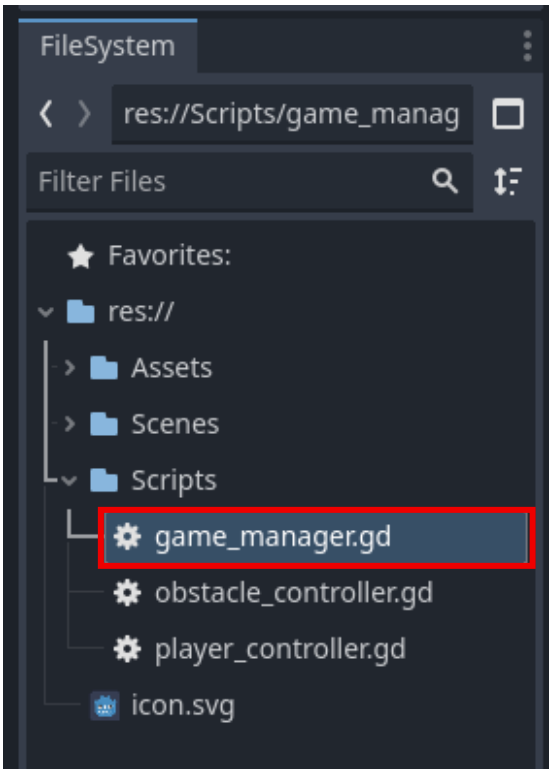
Notice the code that has already been set up.

Review the **obstacle variables** declared at the top of the script. These will be used to set the behavior of the game's obstacle shapes.

- `obstacle_scene`: stores a completed instance of an obstacle shape.
- `base_obstacle_speed`: sets the obstacles' speed at the start of the game.
- `obstacle_speed_increase`: the amount added to the current obstacle speed.
- `obstacle_count`: sets the number of obstacles that will appear at a time.
- `obstacle_speed`: stores the current speed of the obstacles.
- `obstacle_height`: stores the size of the obstacles.

Notice how `obstacle_speed` and `obstacle_height` are not exported and are of type float.

How might those two variables be used?




```
# obstacle vars
@export var obstacle_scene: PackedScene
@export var base_obstacle_speed: float = 300
@export var obstacle_speed_increase: float = 10
@export var obstacle_count: int = 3
var obstacle_speed: float
var obstacle_height: float
```

9 Set the height of the obstacles so that the player can't go around them.

Find **TODO 1** inside the `_ready()` method**.**

Reminder:

Before doing anything else, remove the **pass** command that is currently in the function.

10 Since there are three obstacles per column, the height of each obstacle should be one third of the height of the game screen. When they are arranged from top to bottom, there will be no gaps for the player to slip through.

Under **TODO 1**, set the `obstacle_height` variable to equal the length of the Y axis, divided by the number of obstacles.

Use the `get_viewport()` method and its `get_visible_rect()` property to retrieve the dimensions of the screen. The `size.y` property can then be used to specify the Y axis.

Hint: Which variable stores the number of obstacles in the game?

```
func _ready():
    # --------------------
    # TODO 1:
    # Figure out how tall each obstacle should be.
    # Hint: screen height divided by the number of obstacles
    # --------------------
    obstacle_height = (get_viewport().get_visible_rect().size.y / obstacle_count)
```

11

Spawn 3 obstacles per round!

Find **TODO 2** in the `spawn_obstacles()` function.

Add a `for` loop that runs from the range of `0` to the number of obstacles stored in `obstacle_count`.

```
func spawn_obstacles():
    # --------------------
    # TODO 9:
    # Shuffle the color and shape arrays so obstacles are different each time.
    # --------------------

    # --------------------
    # TODO 2:
    # Use a for loop to create all obstacles.
    # For each obstacle:
    #   - Instantiate it
    #   - Add it to the game
    #   - Give it a position (use obstacle_height * (i - 1))
    # --------------------
    # for ??
```

Reminder:

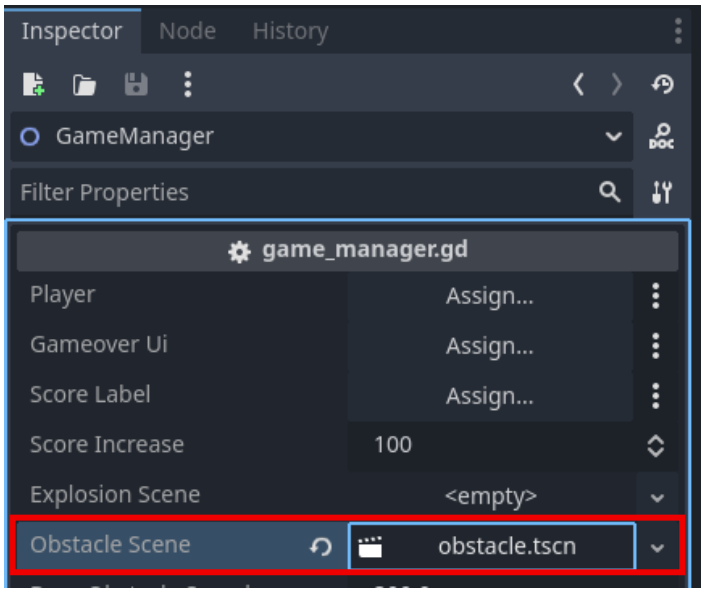
Remove **pass** from the `spawn_obstacles()` function.

12 Inside the loop, declare an `obstacle` variable.

Instantiate a copy of the `obstacle_scene` variable and assign it to `obstacle`.

```
# TODO 2:
# Use a for loop to create all obstacles.
# For each obstacle:
#   - Instantiate it
#   - Add it to the game
#   - Give it a position (use obstacle_height * (i - 1))
# --------------------
# for ??
    # var obstacle = ??
```

13 In the **Inspector** for **GameManager**, use **Quick Load** to assign the Obstacle packed scene to **obstacle.tscn**.



14 Inside the `for` loop under **TODO 2**, use the `get_parent()` and `add_child()` methods to add the instantiated scene stored in `obstacle` to the GameManager node.

```
# TODO 2:
# Use a for loop to create all obstacles.
# For each obstacle:
#   - Instantiate it
#   - Add it to the game
#   - Give it a position (use obstacle_height * (i - 1))
# --------------------
# for ??
    # var obstacle = ??
    # get_parent() ??
```

15 Inside the loop, set the **obstacle's** position to a `Vector2`.

Pass `1200` as the `x` parameter. Set the `y` parameter to the `obstacle_height` multiplied by the difference between `i` (the current index of the array) and `1`.

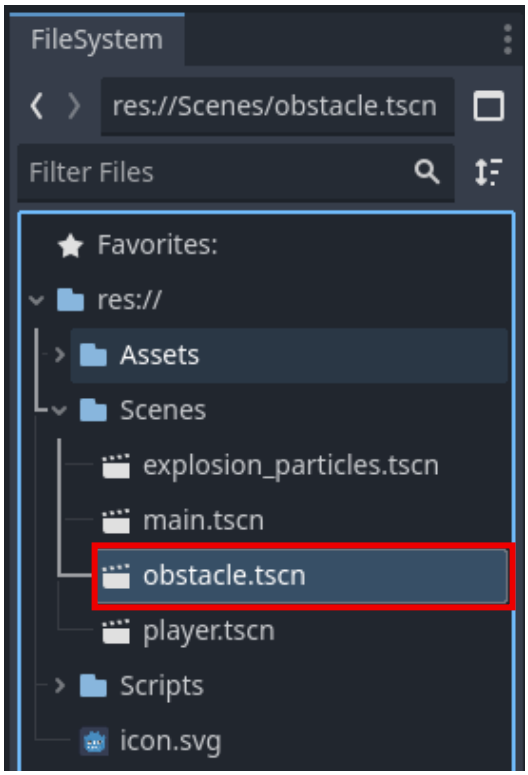
```
# TODO 2:
# Use a for loop to create all obstacles.
# For each obstacle:
#   - Instantiate it
#   - Add it to the game
#   - Give it a position (use obstacle_height * (i - 1))
# --------------------
# for ??
    # var obstacle = ??
    # get_parent() ??
    # obstacle.position = ??
```

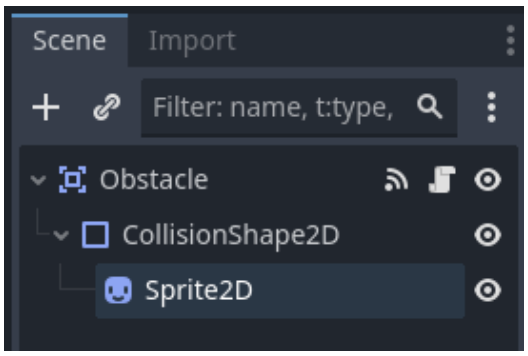
16 Check the code! Update the script as needed.

```
func spawn_obstacles():
    # --------------------
    # TODO 9:
    # Shuffle the color and shape arrays so obstacles are different each time.
    # --------------------

    # --------------------
    # TODO 2:
    # Use a for loop to create all obstacles.
    # For each obstacle:
    #   - Instantiate it
    #   - Add it to the game
    #   - Give it a position (use obstacle_height * (i - 1))
    # --------------------
    for i in range(0, obstacle_count):
        var obstacle = obstacle_scene.instantiate()
        get_parent().add_child(obstacle)

        obstacle.position = Vector2(1200, obstacle_height * (i - 1))
```

17 In **FileSystem**, open **obstacle.tscn**.

Notice that the **Obstacle** main root has two children nodes: CollisionShape2D and Sprite2D.





Reminder:

CollisionShape2D assigns a Shape2D node to the CollisionShape's parent. This is used to create physics behavior and make game objects interact with each other. A Sprite2D is a useful node for displaying a texture.

18 Use a variable to store the Obstacle main root's children nodes.

Inside the `for` loop under **TODO 2**, declare an `obstacle_sprite` variable. Use the `get_node()` method to store the **CollisionShape2D** and **Sprite2D** nodes in the variable.

```
for i in range(0, obstacle_count):
    var obstacle = obstacle_scene.instantiate()
    get_parent().add_child(obstacle)

    obstacle.position = Vector2(1200, obstacle_height * (i - 1))

    var obstacle_sprite = obstacle.get_node("CollisionShape2D/Sprite2D")
```

19 Return to the top of the **game_manager.gd** script and review the list of variables.

Notice the `base_obstacle_speed`, `obstacle_speed`, and `obstacle_speed_increase` variables. These will be used to set the speed of the obstacles during the game.

The `base_obstacle_speed`, is the starting speed of the obstacles. Its value will remain **constant**, so it can be assigned directly in the code. `obstacle_speed`, will change continuously as the game runs.

```
# obstacle vars
@export var obstacle_scene: PackedScene
@export var base_obstacle_speed: float = 300
@export var obstacle_speed_increase: float = 10
```

20 Under **TODO 3** in the `_ready()` method, assign the value of `base_obstacle_speed` to the `obstacle_speed` variable.

```
func _ready():
    # --------------------
    # TODO 3:
    # Set the starting speed of obstacles.
    # Hint: assign base_obstacle_speed to obstacle_speed
    # --------------------
    obstacle_speed = base_obstacle_speed
```

21 Find **TODO 4** in the `spawn_obstacles` function, inside the `for` loop that was created earlier.

Assign the value of `obstacle_speed` to the `speed` property of the `obstacle` variable. (This part can get tricky – remember that properties of scenes can be accessed using dot notation.)

```
    # --------------------
    # TODO 4:
    # Pass the current obstacle_speed to the obstacle.
    # --------------------
    obstacle.speed = obstacle_speed
```

Reminder:

Check the indentation so that this step is **inside** the for loop!

22 Find **TODO 5**. Add code *outside* of the `for` loop to increase the `obstacle_speed` variable.

Add the value of the `obstacle_speed_increase` variable to the current `obstacle_speed` value.

Underneath, add a `print` command that will send a message to the console that "Speed increased to: ", along with the new `obstacle_speed` value. As soon as the speed successfully increases, this message will show up in the console.

```
for i in range(0, obstacle_count):
    var obstacle = obstacle_scene.instantiate()
    get_parent().add_child(obstacle)

    obstacle.position = Vector2(1200, obstacle_height * (i - 1))

    var obstacle_sprite = obstacle.get_node("CollisionShape2D/Sprite2D")
    # --------------------
    # TODO 4:
    # Pass the current obstacle_speed to the obstacle.
    # --------------------
    obstacle.speed = obstacle_speed
    # --------------------
    # TODO 8:
    # Pick a random shape and a random color for the obstacle.
    # Assign the texture and the modulate color to the sprite.
    # Add selected shape and color to corresponding groups.
    # --------------------

# --------------------
# TODO 5:
# After spawning all obstacles, increase the obstacle_speed slightly.
# --------------------
obstacle_speed += obstacle_speed_increase
print("Speed increased to: ", obstacle_speed)
```

Pro Tip:

Printing console messages is a useful tool for checking what's happening in the code – or what's **not** happening, but should be.

23 Return to the list of variables at the top of the script. There are two variables that will be used next: **`distance_between_obstacles`** and **`distance_covered`**.

How might these variables be used as the game runs?

```
# track how far the obstacles have moved since the last round
@export var distance_between_obstacles: float = 750
var distance_covered: float = 0
```

24 Under **TODO 6** in the **`_process()`** method, increase the value of the **`distance_covered`** variable.

Multiply the **`delta`** parameter value by the **`obstacle_speed`** value.

Be sure to delete the **`pass`** command!

```
56 func _process(delta: float) -> void:
57     # --------------------
58     # TODO 14:
59     # If the game is over, stop here and do nothing.
60     # --------------------
61
62     # --------------------
63     # TODO 6:
64     # Keep track of how far the obstacles have moved left.
65     # If the obstacles moved far enough, reset distance_covered and call spawn_obstacles().
66     # --------------------
67     distance_covered += delta * obstacle_speed
```

Reminder:

`delta` is a value that represents the amount of time that passed between rendering frames.

25 Underneath, write an `if` statement.

Check if the value of `distance_covered` is greater than or equal to the value of `distance_between_obstacles`.

If so, reset the value of `distance_covered` to `0`, and call the `spawn_obstacles` function.

```
distance_covered += delta * obstacle_speed


if distance_covered >= distance_between_obstacles:
    distance_covered = 0
    spawn_obstacles()
```

26 Playtest the game – what do you notice?

Look at the **Output** console as the game runs. Notice the "speed increased" messages are being printed even though there are no obstacles visible.

The message tells that the obstacles are spawning; they just can't be seen because they don't have a **texture** yet.

Close the playtest window.

Pause for **Sensei Stop #2**!

Make sure that the code to set the obstacle's height and speed is correct so far.

Reminder: Save your work!

27 To make the obstacles visible, they'll each need a shape and a color.

Return to the top of the script and review the variables under **`# textures and colors`**. What might the purpose of each variable be?

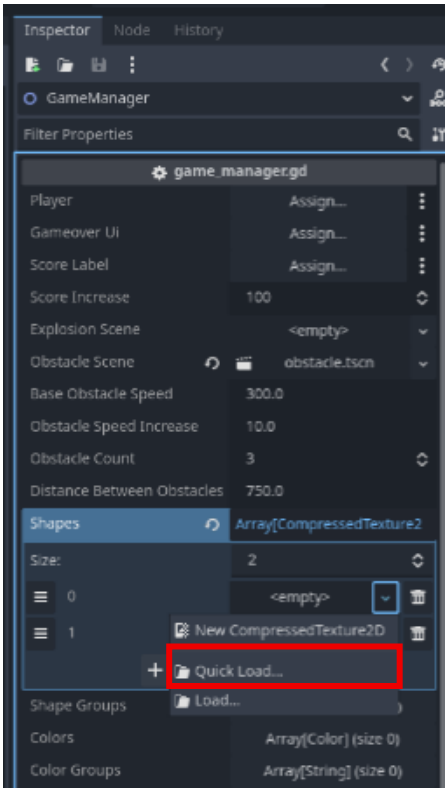
```
# textures and colors
@export var shapes: Array[CompressedTexture2D] = []
@export var shape_groups: Array[String] = []
@export var colors: Array[Color] = []
@export var color_groups: Array[String] = []

var color_array: Array[int]
var shape_array: Array[int]
```

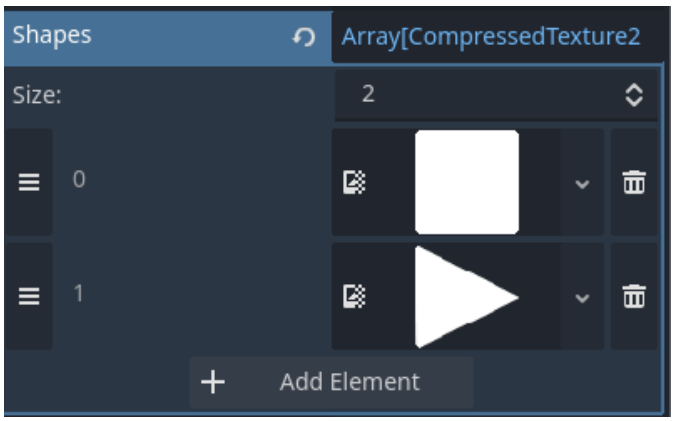
28 Create the array to store shapes.

In the **Inspector** for **GameManager**, find the **Shapes** property – this represents the **shapes** array from the script.

Click **Array[CompressedTexture2D]** to expand the box, then use the "**Add Element**" button to change the size of the array to 2.

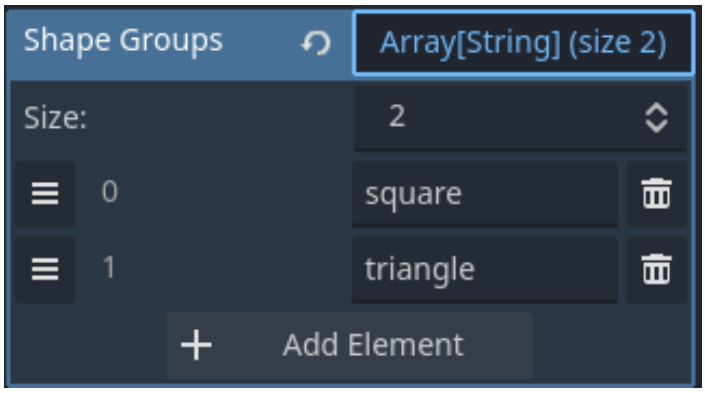
For each new element, click **<empty>**, then use **Quick Load** to add the square and the triangle textures to the array.





29 Underneath the **shapes** property, find the **Shape Group** property. This is an array of strings.

Add two elements to this array. For each element, type the names of the shapes that were used in the **Shapes** array: "square" and "triangle."



30 Return to the **game_manager.gd** script and find **TODO 7** in the `_ready()` method.

Set the `shape_array` variable to an empty array.

Then, `resize` that array to be the same `size` as the `shapes` array.

```
# --------------------
# TODO 7:
# Create arrays of indexes for colors and shapes.
# Hint: fill each array with numbers 0...size-1
# --------------------
shape_array = []
shape_array.resize(shapes.size())
```

Pro Tip:

Pay close attention to the names of the variables here – **shapes** is an array storing actual shape values, while **shape_array** stores an array of numbers that will be matched to those shapes.

31 Underneath, define a `for` loop that will iterate through a range that is equal to the `size` of the `colors` array.

Each time the loop runs, store the current number from that range in the current index of the `shape_array`.

```
# --------------------
# TODO 7:
# Create arrays of indexes for colors and shapes.
# Hint: fill each array with numbers 0...size-1
# --------------------
shape_array = []
shape_array.resize(shapes.size())

for i in range(shapes.size()):
    shape_array[i] = i
```

32

Assign a random shape to an obstacle when it spawns.

Find **TODO 8** in the `spawn_obstacles()` function.

Inside the `for` loop, declare a `random_shape` variable. Assign the variable a random index from the `shape_array` by using the modulo operator `%` to find the remainder of `i` divided by the `size` of the `shapes` array.

This is an example of **random integer generation –** the number must an integer to match an index in the array being chosen from.

```
for i in range(0, obstacle_count):
	
	var obstacle = obstacle_scene.instantiate()
	get_parent().add_child(obstacle)

	obstacle.position = Vector2(1200, obstacle_height * (i - 1))
	
	var obstacle_sprite = obstacle.get_node("CollisionShape2D/Sprite2D")
	obstacle_sprite.scale.y = obstacle_height / 64

	obstacle.speed = obstacle_speed
	
	# --------------------
	# TODO 8:
	# Pick a random shape and a random color for the obstacle.
	# Assign the texture and the modulate color to the sprite.
	# Add selected shape and color to corresponding groups.
	# --------------------
	var random_shape = shape_array[i % shapes.size()]
```

33 Once the random integer is assigned to the `random_shape` variable, use it to assign the matching texture to the `obstacle_sprite`.

Change the obstacle_sprite's `texture` to the element found at the `random_shape` value index of the `shapes` array.

```
for i in range(0, obstacle_count):

    var obstacle = obstacle_scene.instantiate()
    get_parent().add_child(obstacle)

    obstacle.position = Vector2(1200, obstacle_height * (i - 1))

    var obstacle_sprite = obstacle.get_node("CollisionShape2D/Sprite2D")
    obstacle_sprite.scale.y = obstacle_height / 64

    obstacle.speed = obstacle_speed

    # --------------------
    # TODO 8:
    # Pick a random shape and a random color for the obstacle.
    # Assign the texture and the modulate color to the sprite.
    # Add selected shape and color to corresponding groups.
    # --------------------
    var random_shape = shape_array[i % shapes.size()]
    obstacle_sprite.texture = shapes[random_shape]
```

34 Playtest the game! Notice the obstacles appearing on screen.

However, the shapes in every column of obstacles are in the same order every time they spawn. Add code to make the shapes appear in a random order.

Close the playtest window.

35 Find **TODO 9** in the `spawn_obstacles()` function.

Use the `shuffle()` array method to randomly reorder the values in the `shape_array`. This is a built-in method in Godot, made for randomizing arrays.

> `shuffle()`: Shuffles all elements of the array in a random order.
>
> **Parameters:** None
>
> **Returns (*void*):** This method is void; it doesn't return anything.

```
func spawn_obstacles():
    # --------------------
    # TODO 9:
    # Shuffle the color and shape arrays so obstacles are different each time.
    # --------------------
    shape_array.shuffle()
```

Pro Tip:

Notice how the `shuffle()` method is called before the `for` loop. This will update the order of the `shape_array` before the obstacles spawn.

36 Playtest the game again. Notice how the shapes now spawn in a more random order.

However, there's a new problem – the obstacle shapes are overlapping each other. They should each take up one-third of the screen height without overlapping.

Close the playtest window.

37 Return to **TODO 2**, where the `obstacle_sprite` variable was declared.

On the line below, use `scale.y` to re-scale the `obstacle_sprite`.

Divide the `obstacle_height` by `82` and assign that value to the `Y` axis of the `scale` property for the `obstacle_sprite`.

```
for i in range(0, obstacle_count):
    var obstacle = obstacle_scene.instantiate()
    get_parent().add_child(obstacle)

    obstacle.position = Vector2(1200, obstacle_height * (i - 1))

    var obstacle_sprite = obstacle.get_node("CollisionShape2D/Sprite2D")
    obstacle_sprite.scale.y = obstacle_height / 64
```

```
for i in range(0, obstacle_count):

    var obstacle = obstacle_scene.instantiate()
    get_parent().add_child(obstacle)

    obstacle.position = Vector2(1200, obstacle_height * (i - 1))

    var obstacle_sprite = obstacle.get_node("CollisionShape2D/Sprite2D")
    obstacle_sprite.scale.y = obstacle_height / 82
```

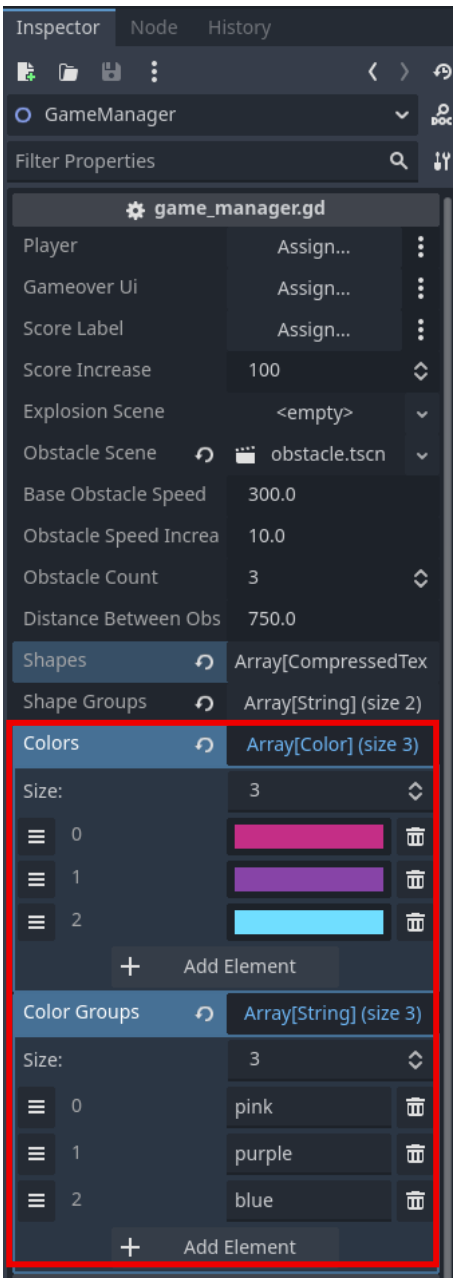
Playtest the game and see how much space is between the obstacle shapes now. Tinker with the divisor value until the shapes aren't overlapping, without space between them. All three shapes should fit exactly on the screen.

38 Store random colors for each obstacle!

In the **Inspector** for **GameManager**, add 3 elements to the **Colors** and **Color Groups** arrays.

Select three distinct colors in the **Colors** array, then type those names in the **Color Groups** array.

Refer back to steps **28-29** for additional guidance.



39 Update the code under each of the comments below to assign a random color to each obstacle:

- **TODO 7:** Define an empty `color_array`. Resize the array to be the same length as `colors`.
- **TODO 7**: Add a `for` loop to assign the current `color_array` index.
- **TODO 8**: Declare a `random_color` variable. Assign the obstacle_sprite's `modulate` property using `random_color`.
- **TODO 9:** Shuffle the values in the `color_array`.

Refer back to steps **30-35** for additional guidance.

40 Check the code! Update the script as needed.

```
# TODO 7:
# Create arrays of indexes for colors and shapes.
# Hint: fill each array with numbers 0...size-1
# --------------------
shape_array = []
shape_array.resize(shapes.size())

for i in range(shapes.size()):
    shape_array[i] = i

color_array = []
color_array.resize(colors.size())

for i in range(colors.size()):
    color_array[i] = i
```

```
    # --------------------
    # TODO 8:
    # Pick a random shape and a random color for the obstacle.
    # Assign the texture and the modulate color to the sprite.
    # Add selected shape and color to corresponding groups.
    # --------------------
    var random_shape = shape_array[i % shapes.size()]
    obstacle_sprite.texture = shapes[random_shape]

    var random_color = color_array[i % colors.size()]
    obstacle_sprite.modulate = colors[random_color]
```

```
func spawn_obstacles():
    # --------------------
    # TODO 9:
    # Shuffle the color and shape arrays so obstacles are different each time.
    # --------------------
    color_array.shuffle()
    shape_array.shuffle()
```

41

Playtest the game.

Do the obstacle shapes spawn in random colors?

Close the playtest window.

Pause for **Sensei Stop #3**!

Make sure the code to assign colors and shapes works correctly, and check that the arrays are set up in the Inspector as well.

Reminder: Save your work!

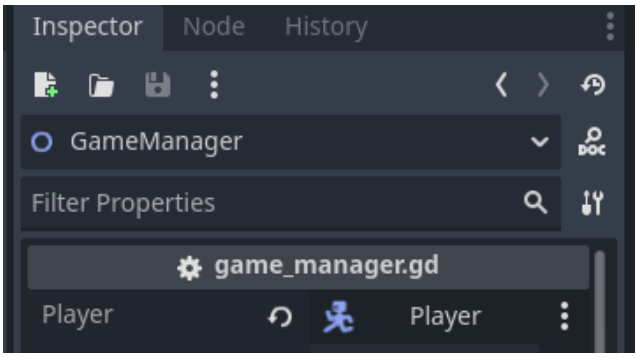
42

Add a player to the game!

At the top of **game_manager.gd**, notice the `player` variable of type `Node2D`.

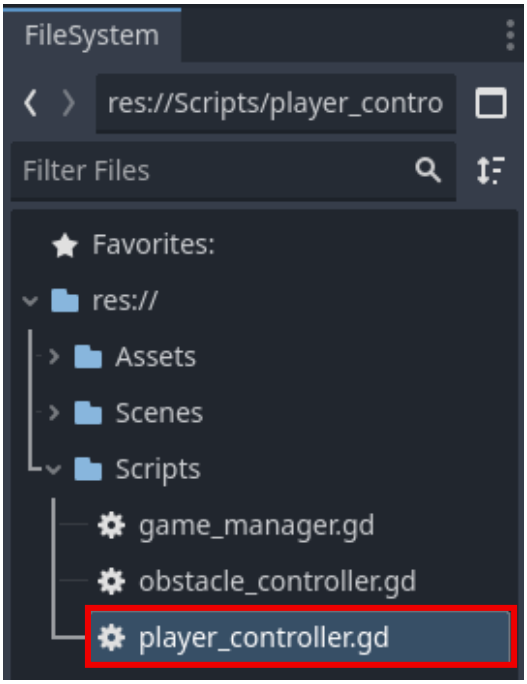
In the Inspector for **Game Manager**, find the Player property, and assign the **Player** node to it.

```
@export var player: Node2D
```



43 In **FileSystem**, open the **player_controller.gd** script, and review the code that's already been provided.

Notice the functions for moving the player and detecting when the player hits an obstacle.



44 Find the `randomize_player()` function. Remove the `pass` command.

Under **TODO 10**, use the `add_to_group()` method to add the player to the `"player"` group.

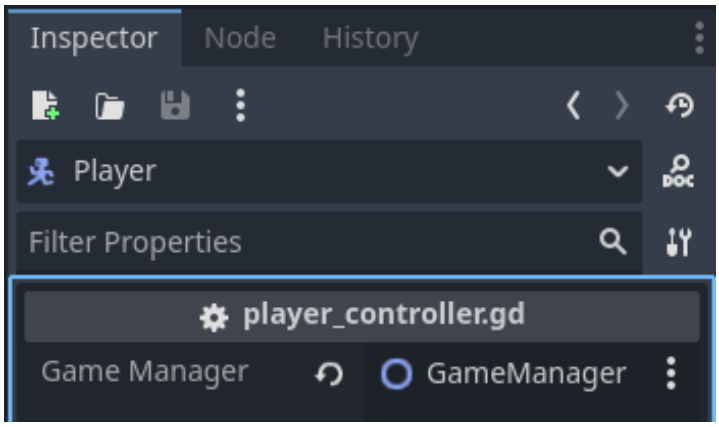
```
func randomize_player() -> void:
    # --------------------
    # TODO 10
    # Add the player to the "player" group so the game knows this node is the player.
    # --------------------
    add_to_group("player")
```

Reminder:

In Godot, groups act like tags that can be used to make changes to all nodes that have that tag at the same time.

45 In the Inspector for **Player** node, assign the **GameManager** node to the Game Manager property.

This will allow the Player node to access the shape & color array information stored in the Game Manager node.



46 Inside the `randomize_player()` function, find **TODO 11.**

Declare a `random_shape` variable and use the `randi_range()` method to assign it a random number value.

Set the range from `0` to the `size` of the `shapes` **array** from **game_manager.gd** script, minus `1`**.**

```
func randomize_player() -> void:
    add_to_group("player")
    # --------------------
    # TODO 11
    # Pick a random shape number then set the player sprite texture to that shape.
    # Pick a random color number then set the player sprite modulate to that color.
    # --------------------
    var random_shape = randi_range(0, game_manager.shapes.size() - 1)
```

47 Randomize the shape of the player sprite.

Underneath, update the player_sprite's **`texture`**. Assign it one of the values from **game_manager**'s **`shapes`** array. Use the **`random_shape`** variable as the array index.

```
# --------------------
# TODO 11
# Pick a random shape number then set the player sprite texture to that shape.
# Pick a random color number then set the player sprite modulate to that color.
# --------------------
var random_shape = randi_range(0, game_manager.shapes.size() - 1)
player_sprite.texture = game_manager.shapes[random_shape]
```

48 Inside the **`randomize_player()`** function, find **TODO 12.**

Update the value of the **`player_shape_name`** variable with one of the values from game_manager's **`shape_groups`** array. Use the **`random_shape`** variable as the index.

Now, the game will keep track of what the player's current shape is to determine if it's collided with the correct obstacle or not.

```
# --------------------
# TODO 12
# Save the chosen shape group and color group into player_shape_name
# and player_color_name so the game knows what the player currently is.
# --------------------
player_shape_name = game_manager.shape_groups[random_shape]
```

49 Playtest the game!

When the game starts, the player should either be a square or a triangle. Restart the game a few times – is the player always the same shape when the game starts?

Note: An error may occur when the score is supposed to change. Why might that be occurring?

50 Repeat the steps above to assign a random color to the player.

- **TODO 11**: Declare a `random_color` variable. Access the game_manager's `colors` array.
- **TODO 11**: Set the player_sprite's `modulate` property to the element at the `random_color` index of `colors`.
- **TODO 12:** Assign the `random_color` index as the `player_color_name`.

Refer to steps 46-48 for additional guidance.

51

Check the code! Update the script as needed.

```
var random_color = randi_range(0, game_manager.colors.size() - 1)
player_sprite.modulate = game_manager.colors[random_color]
```

```
# --------------------
# TODO 12
# Save the chosen shape group and color group into player_shape_name
# and player_color_name so the game knows what the player currently is.
# --------------------
player_shape_name = game_manager.shape_groups[random_shape]
player_color_name = game_manager.color_groups[random_color]
```

52

Add the values from the **shape_group** and **color_group** arrays to groups.

Go back to **game_manager.gd**, and find **TODO 11**.

Use the **add_to_group()** method to add the **obstacle** to both the **shape_group** and **color_group** arrays.

Now, the game will have the information to see if the player's shape or color match the obstacle.

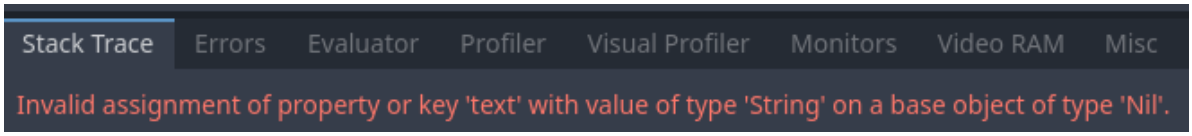
```
var random_shape = shape_array[i % shapes.size()]
obstacle_sprite.texture = shapes[random_shape]

var random_color = color_array[i % colors.size()]
obstacle_sprite.modulate = colors[random_color]

obstacle.add_to_group(shape_groups[random_shape])
obstacle.add_to_group(color_groups[random_color])
```

53 Playtest the game!

When the game starts, notice that the player is one of the three possible colors. Restart the game a few times and the player should be a different combination of a random shape and a random color each time.

What might happen if the player collides with an obstacle that's a different color? What about hitting one that's the same color?

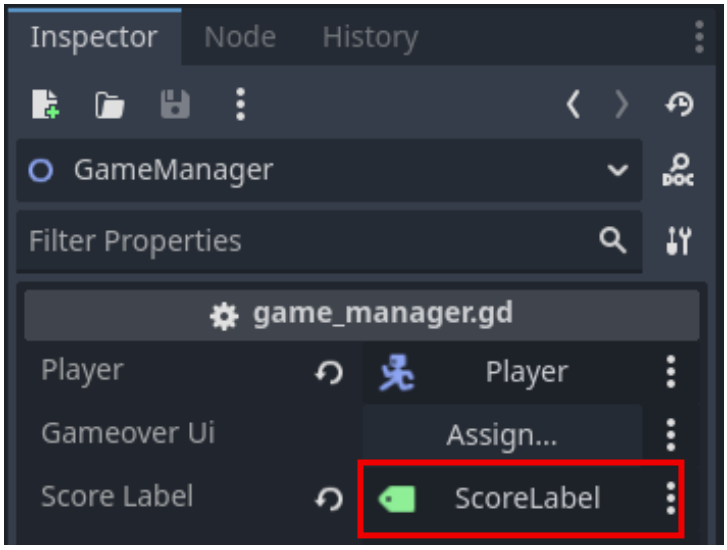
Note: An error may occur when the score is supposed to change. Why might that be occurring?



54 In the Inspector for **Game Manager**, assign the **ScoreLabel** node to the **Score Label** property.

When a square player collides with an obstacle that's the same color, or when a triangle player collides with an obstacle that's the same shape, score will increase.

Otherwise, nothing happens.



55 Make sure the player can keep up with increasingly faster obstacles.

In **player_controller.gd**, underneath **TODO 12**, increase the `move_speed` variable by the value of the `move_speed_increase` variable.

```
# --------------------
# TODO 12
# Save the chosen shape group and color group into player_shape_name
# and player_color_name so the game knows what the player currently is.
# --------------------
player_shape_name = game_manager.shape_groups[random_shape]
player_color_name = game_manager.color_groups[random_color]

move_speed += move_speed_increase
```

56 Playtest the game.

As the player passes more obstacles, it will be able to move up and down more quickly. This will give them a chance to keep up as the obstacles start moving faster.

Tinker with the `move_speed_increase` value in the Inspector until the game feels playable, but still challenging enough.

Pause for **Sensei Stop #4**!

Check that the player sprite's shape and color randomize correctly.

Reminder: Save your work!

57 In the **player_controller.gd** script, find the `on_obstacle_hit()` function and review the code in it. What is this function doing?

The goal for the game is to match a square player with an obstacle of the same color, or a triangle player with an obstacle of the same shape.

The game will end when a triangle player hits a square obstacle or when a square player hits an obstacle of a different color.

```
# when hit, if the object has a group that we don't match, lose
func on_obstacle_hit(obstacle: Node):
    var found_match = false

    # if we are a square, check if our color matches. otherwise, check if our shape matches
    if player_shape_name == game_manager.shape_groups[0]:
        if obstacle.is_in_group(player_color_name):
            found_match = true
            game_manager.increase_score()
    else:
        if obstacle.is_in_group(player_shape_name):
            found_match = true
            game_manager.increase_score()

    if not found_match:
        game_manager.end_game()
```

58

Add the game over functionality to the game.

In **game_manager.gd**, find **TODO 13** and the `end_game()` function.

First, delete the `pass` command inside the function.

Add a `print` statement to display the message `"Game Over!"` to check when the game has ended in the console.

Change the `game_over` variable to `true`.

```
func end_game():
    # --------------------
    # TODO 13:
    # End the game:
    # - Print "Game over!"
    # - Mark game_over as true
    # - Show the Game Over UI
    # --------------------
    print("Game over!")
    game_over = true
```

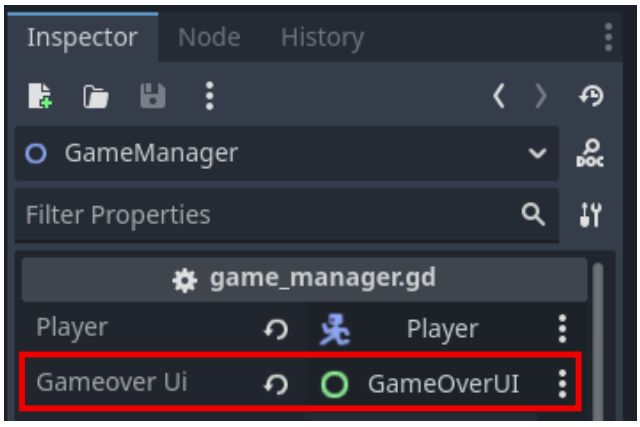
59

Display the Game Over message on the game screen.

In the `end_game()` function, switch the `gameover_UI` visibility to `true`.

In the Inspector for **GameManager**, assign the GameOverUI node to the **Gameover UI** property.

```
func end_game():
    # --------------------
    # TODO 13:
    # End the game:
    # - Print "Game over!"
    # - Mark game_over as true
    # - Show the Game Over UI
    # --------------------
    print("Game over!")
    game_over = true
    gameover_UI.visible = true
```



60 Playtest the project. What happens when a square hits a different color, or a triangle hits a square?

Does the game's appearance change?

Does the game end?

61 When the game ends, obstacles should stop spawning.

In **GameManger.gd**'s `_process()` function, under **TODO 14**, add a conditional statement that checks if the `game_over` variable is set to `true`. If it is, run the `return` command.

Playtest the game again. Once the game ends, new obstacles will stop spawning.

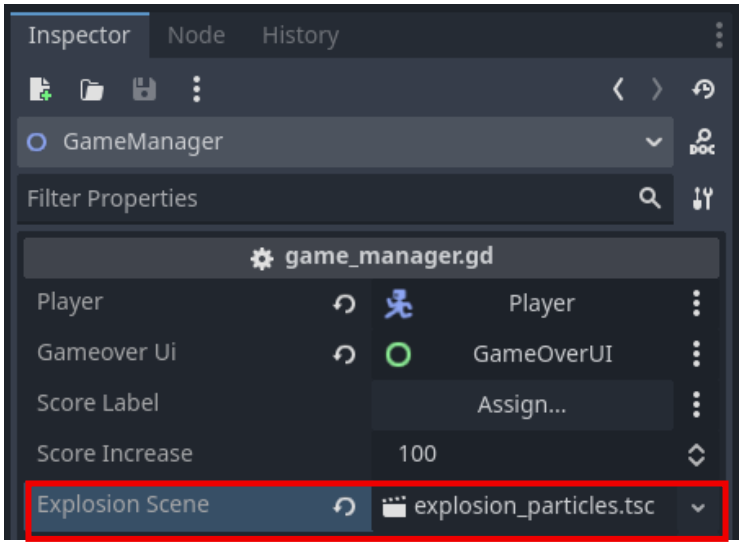
```
func _process(delta: float) -> void:
    # --------------------
    # TODO 14:
    # If the game is over, stop here and do nothing.
    # --------------------
    if game_over:
        return
```

62 Add some visual effects to make the game over screen more eye-catching! When the game ends by hitting the wrong obstacle, an explosion effect should go off.

At the top of **game_manager.gd**, notice the `explosion_scene` variable, which is of the packed scene type.

In the **Inspector** for **Game Manager**, assign the **explosion_particles** scene to the **Explosion Scene** property.

```
# gameover explosion scene
@export var explosion_scene: PackedScene
```



63 In the `end_game()` function, find **TODO 15**.

Declare an `explosion` variable and use it to instantiate an instance of `explosion_scene`.

```
# --------------------
# TODO 15:
# Play the explosion effect where the player is.
# move it to the player's position, and start it.
# --------------------
# var explosion ??
```

64 Add the `explosion` variable as a child of the **Main** node.

```
# --------------------
# TODO 15:
# Play the explosion effect where the player is.
# move it to the player's position, and start it.
# --------------------
# var explosion ??
# add_child ??
```

65 Set the **explosion's** `global position` to be the same as the **player's** `global position`.

```
# --------------------
# TODO 15:
# Play the explosion effect where the player is.
# move it to the player's position, and start it.
# --------------------
# var explosion ??
# add_child ??
# explosion position ??
```

66 Use the `restart()` method to play the effect and turn on the explosion's **emitting** property.

```
# --------------------
# TODO 15:
# Play the explosion effect where the player is.
# move it to the player's position, and start it.
# --------------------
# var explosion ??
# add_child ??
# explosion position ??
# restart ??
# emitting ??
```

67 Check the code! Update the script as needed.

```
# --------------------
# TODO 15:
# Play the explosion effect where the player is.
# move it to the player's position, and start it.
# --------------------
var explosion = explosion_scene.instantiate()
get_parent().add_child(explosion)
explosion.global_position = player.global_position
explosion.restart()
explosion.emitting = true
```

68 Playtest the game!

When the game ends, the player sprite will give off an explosion effect with star-shaped particles that fade away.

Close the playtest window.

69 When the game ends, the player sprite should be removed.

Find **TODO 16**. Remove the player from the game using the `queue_free()` method.

```
# --------------------
# TODO 16:
# Remove the player
# --------------------
player.queue_free()
```

70 Playtest the game one more time to make sure everything works: the player gets points for the correct obstacle, the game ends when they hit a wrong one, and the player is removed from the game with an explosion when the game ends.

Pause for **Sensei Stop #5**!

Great job! You now have a game that makes use of particle effects that's fun to play over and over!

- What did you learn while building this game?
- What problems did you run into?
- How can you use what you learned in new projects?

Reminder: Save your work!