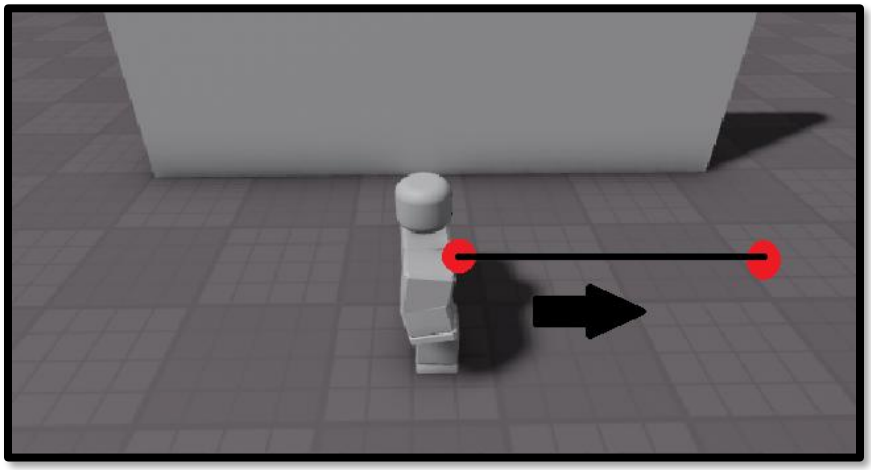
CODE NINJAS®

Silver Belt Ninja Guide
Activity 04: Jungle Escape

RAYCASTING

A common game design technique that is used to *gather information* about surrounding objects is called **Raycasting**. Rays must be given a **source** and a **destination**, so they act very similar to Vectors. The difference between Rays and Vectors is that Rays can gather information about *any* objects they collide with.

During gameplay, **Raycasting** has many uses! It can be used to detect what object is in front of the player:



VISUAL OF A RAYCAST FROM A ROBLOX AVATAR. SOURCE: DEVFORUM.ROBLOX.COM

Or to determine if instant-hit (hitscan) weapons like lasers hit a target:

LAS-98 LASER CANNON FROM HELLDIVERS 2. SOURCE: HELLDIVERS.WIKI.GG

In Jungle Escape, **Raycasting** will be used to detect different types of nearby platforms.

Enums & Match Statements

Often in programming, there are a set number of possible values that a variable should take on. Instead of setting a variable to 0, 1, 2, 3, ... for different "states", which can be hard to read, a better alternative would be to use **Enums**. **Enum**, short for enumeration, is a user-defined data type that can hold one of the user-defined values.

For example, a user could make an **Enum** named **BarnAnimal** which could have possible values **BarnAnimal.COW**, **BarnAnimal.SHEEP**, **BarnAnimal.CHICKEN**, and more! Then, a variable can be set to one of the possible values in the **Enum** and compared to the others.

Creating the Enum

```
enum BarnAnimal {
        COW,
        SHEEP,
        CHICKEN,
        [etc.]
}
```

Using the Enum

```
func speak(animalType: BarnAnimal):
        if animalType == BarnAnimal.COW:
                print("Moo")
        if animalType == BarnAnimal.SHEEP:
                print("Baaa")
        if animalType == BarnAnimal.CHICKEN:
                print("Bokbokbok")
        [etc.]

speak(BarnAnimal.CHICKEN) # prints "Bokbokbok"
```

A popular way to compare a variable to multiple possible values to run code is a **match** statement. It is most used with **Enums**, and is a more readable alternative to writing a series of if statements (as shown in the diagram below):

If statements	**Match statement**

if variable == value1:
    code1
if variable == value2:
    code2
if variable == value3:
    code3
...
	match variable:
	    value1:
	        code1
	    value2:
	        code2
	    value3:
	        code3

Activity 04: Jungle Escape

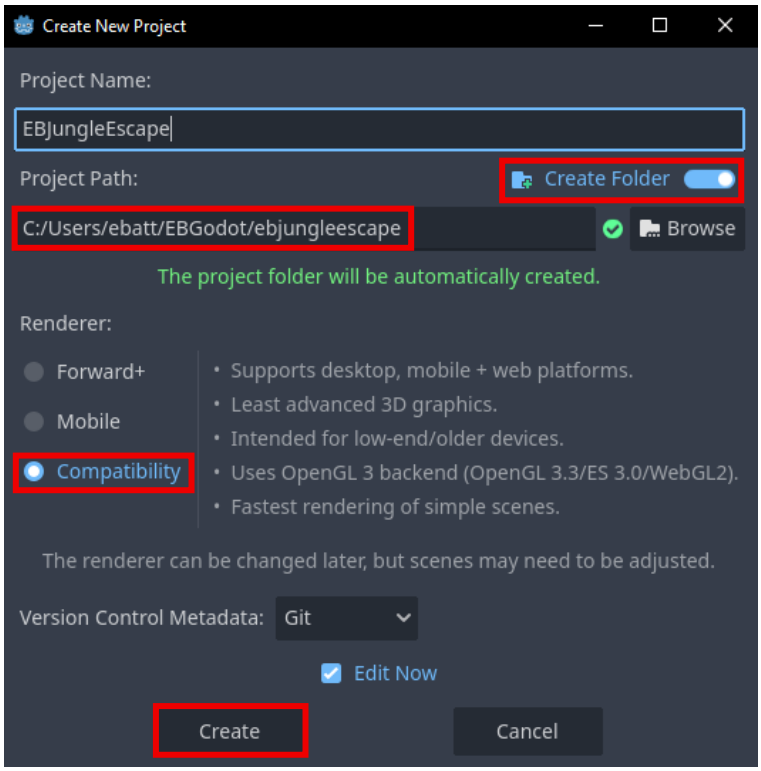
In this game, Codey must navigate the wooden boards toward the glowing portal that teleports Codey to the escape point: the beach! Be careful though – some boards may disappear after being stepped on and others will trick you and make you fall right through!

In this activity, you will learn about **Raycasting** and use it to check what kind of board is in front of Codey. You will also bugfix Codey's jumping and animation code!

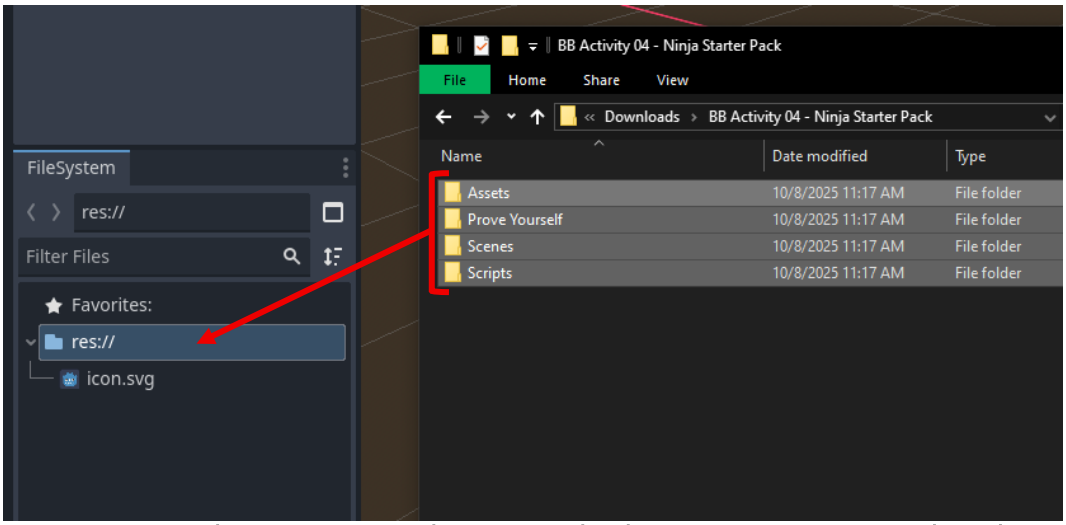
1 After opening Godot, in the top left corner select **+ Create**.

A **Create New Project** window will pop up. Name the project **[YourInitials]JungleEscape.**

Enable **Create Folder** if it is not already enabled. Make sure the **Project Path** is the same as previously set by a Code Sensei at your center. Ensure that the project is in **Compatibility** mode, then click **Create**.

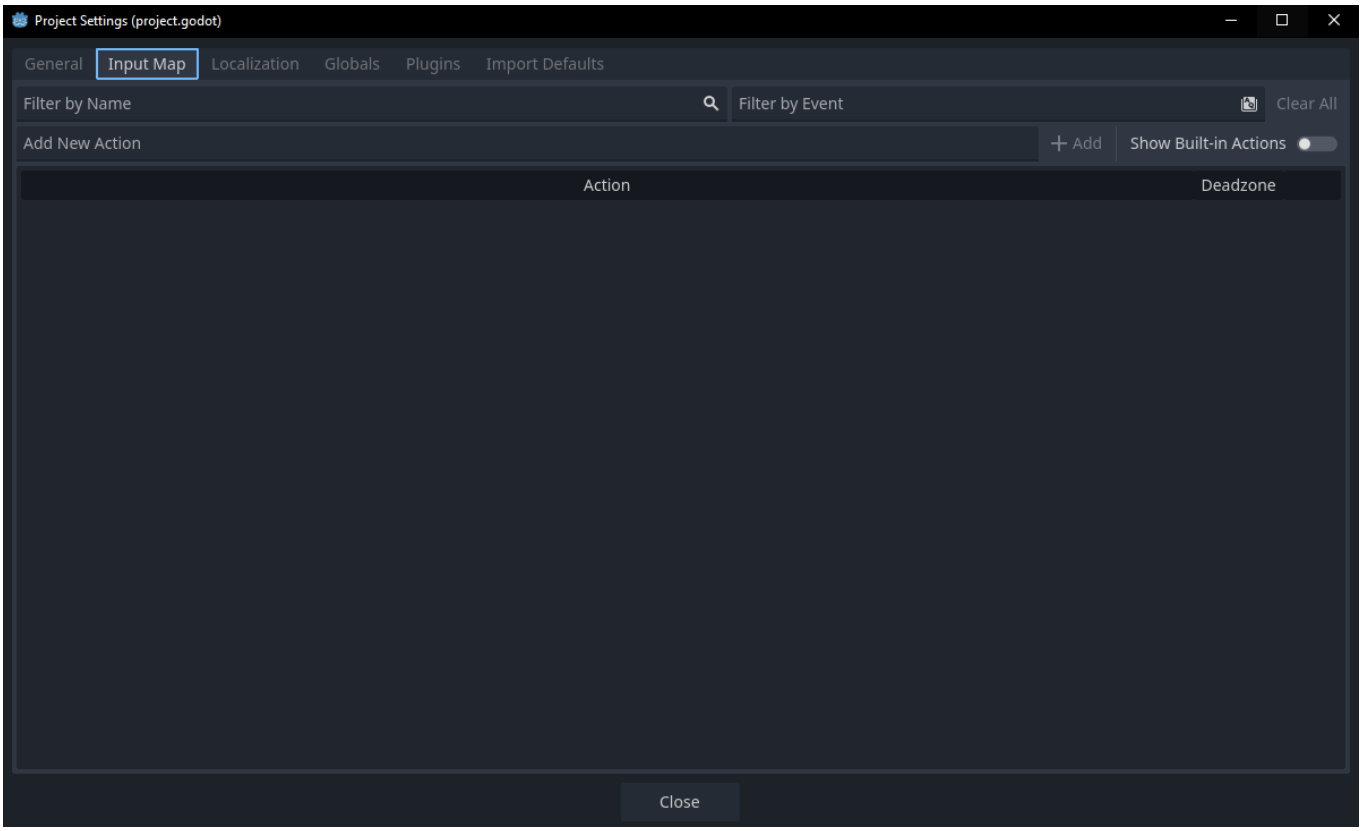


2 **Extract** the **SB Activity 04 - Ninja Starter Pack** and drag all the folders into res:// in **FileSystem**.



3 Add custom controls to move Codey around! The project's scripts already use specific actions in their code that must be created in the Input Mapping.

Navigate to **Project Settings > Input Map**.

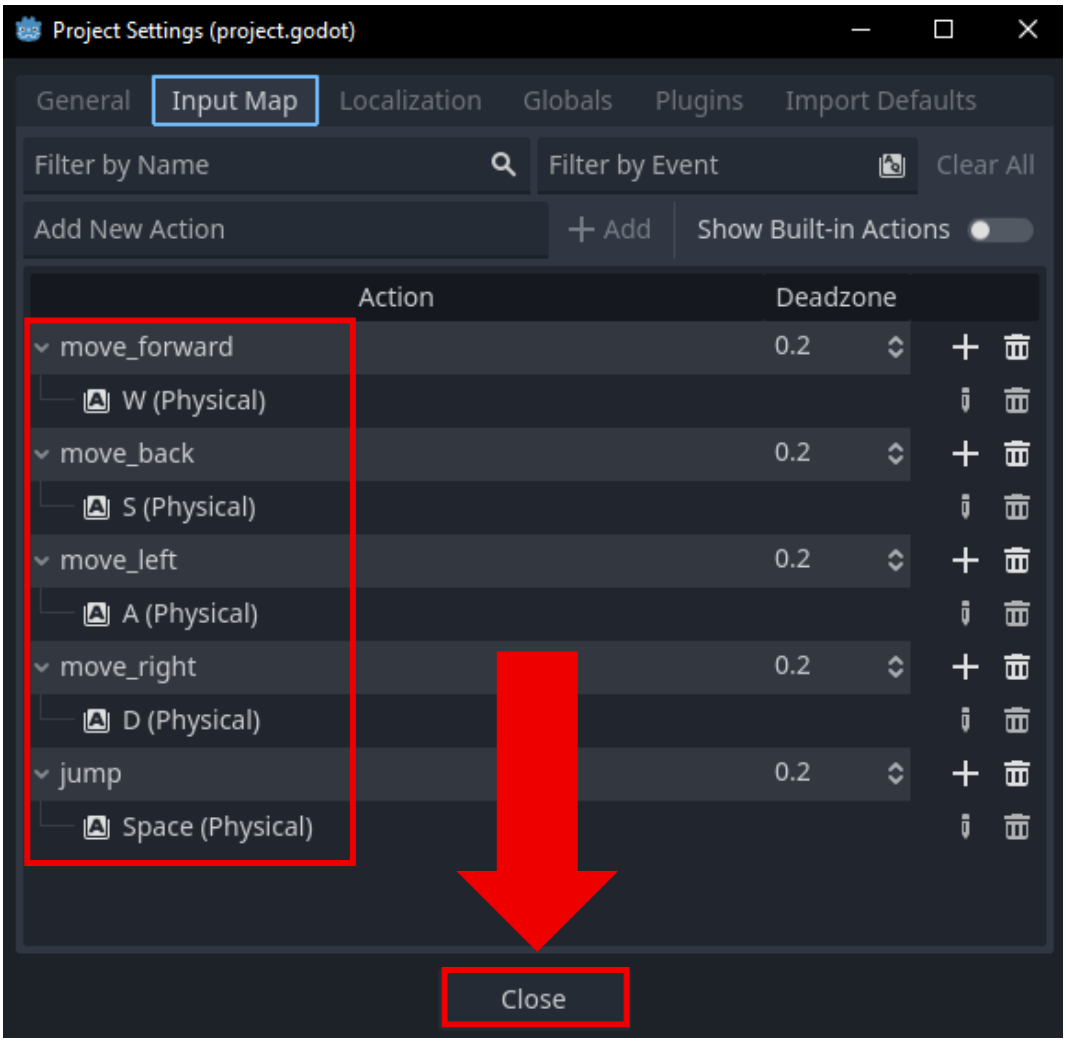


4 Add the following actions to the project:

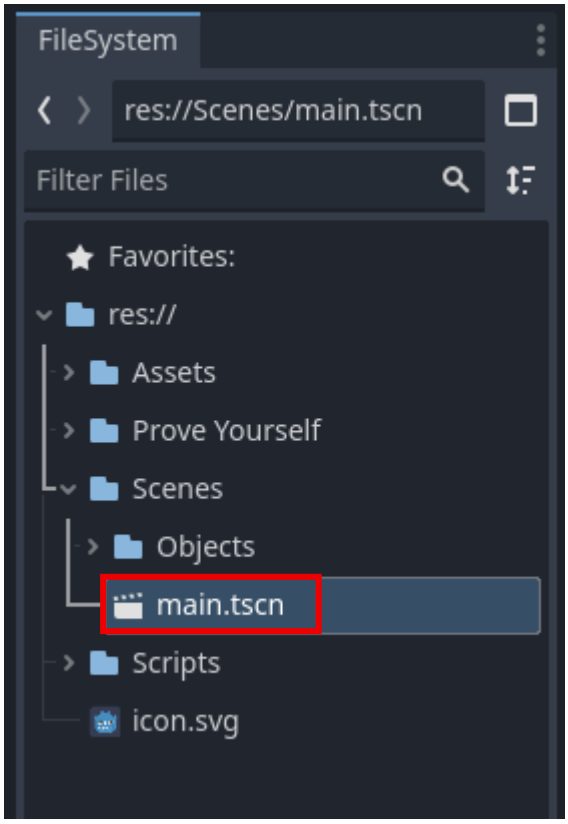
- W – **move_forward**
- S – **move_back**
- A – **move_left**
- D – **move_right**
- Space – **jump**

Make sure the spelling is exact, or the scripts will not work as intended.

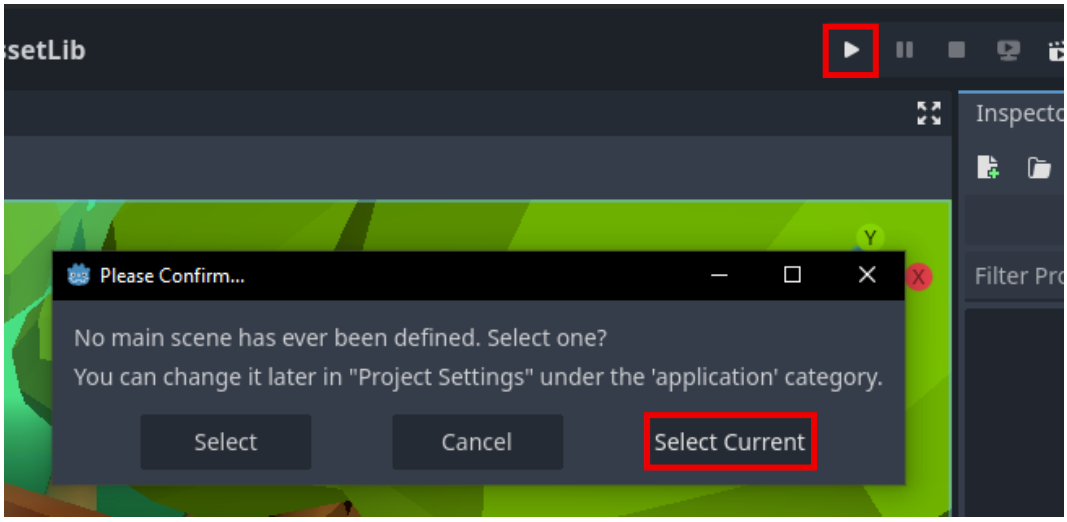
Click **Close**.



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



In the top right corner, start a playtest and **Select Current** as the main scene.

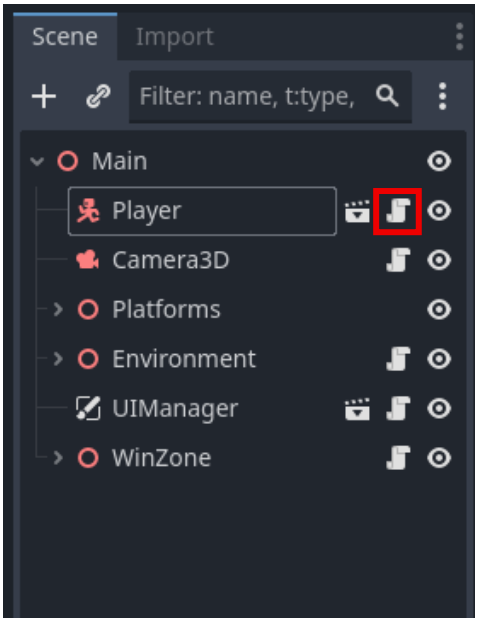


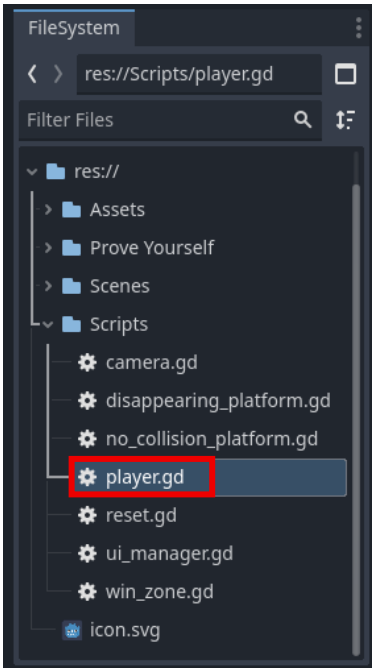
6 While playtesting, try using the space bar to jump, then again while in mid-air.

Oh, no! Codey can jump infinitely and skip the entire game!

7 In **Scene** or **FileSystem**, navigate to **player.gd**.

The code already in **player.gd** takes care of *player movement* (using the **InputMap** actions) and *animation* (using the `is_jumping` variable). There are also some export variables that may be tweaked in the **Inspector.**





8 Scroll down to **TODO 1**, which can be found inside of the `_physics_process()` method.

What's wrong with the code?

```
72      if not is_jumping:
73          handle_animation(AnimationState.IDLE)
74
75  # --------------------
76  # TODO 1
77  # Jumping and gravity bugfixing
78  # --------------------
79  if Input.is_action_just_pressed("jump"):
80      player_velocity.y = jump_velocity
81      is_jumping = true
82      handle_animation(AnimationState.JUMPING)
83
84  player_velocity.y -= gravity * delta
85
86  # Assign velocity and move
87  velocity = player_velocity
88  move_and_slide()
```

9 The code allows the player to jump whenever they want *instead of* only when they are on the ground!

Use CharacterBody's `is_on_floor()` method to add a condition to ensure that Codey is only able to jump when on the ground!

```
69    # --------------------
70    # TODO 1
71    # Jumping and gravity bugfixing
72    # --------------------
73    if is_on_floor():
74        if Input.is_action_just_pressed("jump"):
75            player_velocity.y = jump_velocity
76            isJumping = true
77            handle_animation(AnimationState.JUMPING)
```

Pro Tip:

Select multiple lines then press tab to indent them all!

10

Playtest the project.

Hooray! Codey cannot jump mid-air anymore. However, *something else* seems wrong.

When Codey lands, the animations don't seem right. Codey *freezes up*! The player also falls off platforms too fast since gravity is still applied *even* when Codey is on a platform.

11

Back in the **player.gd** script, fix the gravity so it is only applied while Codey is mid-air.

Add an `else`-statement that matches the `is_on_floor()` `if`-statement.

Notice the code that decreases the player's y-velocity by `gravity * delta`. Press TAB to move this line of code inside the else statement.

```
74  # --------------------
75  # TODO 1
76  # Jumping and gravity bugfixing
77  # --------------------
78  if is_on_floor():
79      if Input.is_action_just_pressed("jump"):
80          player_velocity.y = jump_velocity
81          is_jumping = true
82          handle_animation(AnimationState.JUMPING)
83  else:
84      player_velocity.y -= gravity * delta
85
```

12 Notice the function call to **`handle_animation()`** when the player jumps. What does this function do?

Press and hold **CTRL**, then click on the blue **`handle_animation`** text to navigate to the function definition.

```
74  # --------------------
75  # TODO 1
76  # Jumping and gravity bugfixing
77  # --------------------
78  if is_on_floor():
79      if Input.is_action_just_pressed("jump"):
80          player_velocity.y = jump_velocity
81          is_jumping = true
82          handle_animation(AnimationState.JUMPING)
83  else:
84      player_velocity.y -= gravity * delta
85
```

13 Before looking into the **handle_animation** function, it's important to talk about what an **enum** is. Above the function, an **AnimationState** enum is created which has three possible values: **IDLE**, **WALKING**, and **JUMPING**.

```
20  enum AnimationState
21  {
22      IDLE,
23      WALKING,
24      JUMPING,
25  }
26
27  func handle_animation(anim_state: AnimationState) -> void:
28      match anim_state:
29          AnimationState.IDLE:
30              if anim.current_animation != "PlayerLibrary/IdlePose":
31                  anim.play("PlayerLibrary/IdlePose")
32          AnimationState.JUMPING:
33              if anim.current_animation != "PlayerLibrary/Jump":
34                  anim.play("PlayerLibrary/Jump")
35          AnimationState.WALKING:
36              if anim.current_animation != "PlayerLibrary/Walk":
37                  anim.play("PlayerLibrary/Walk")
38
```

New Concept: Enum

Enum, short for enumeration, is a user-defined data type. Enums are very helpful for making code easier to read and for variables that can only be equal to a small set of values.

14 The parameter for the `handle_animation()` function is called `anim_state`, which is of type `AnimationState` (the enum). This means that `anim_state` must equal `AnimationState.IDLE`, `AnimationState.JUMPING`, or `AnimationState.WALKING`.

Inside the function, the first line reads `match anim_state:` which checks if `anim_state` is equal to any of the below values.

```
20  enum AnimationState
21  {
22      IDLE,
23      WALKING,
24      JUMPING,
25  }
26
27  func handle_animation(anim_state: AnimationState) -> void:
28      match anim_state:
29          AnimationState.IDLE:
30              if anim.current_animation != "PlayerLibrary/IdlePose":
31                  anim.play("PlayerLibrary/IdlePose")
32          AnimationState.JUMPING:
33              if anim.current_animation != "PlayerLibrary/Jump":
34                  anim.play("PlayerLibrary/Jump")
35          AnimationState.WALKING:
36              if anim.current_animation != "PlayerLibrary/Walk":
37                  anim.play("PlayerLibrary/Walk")
38
```

New Concept: Match Statements

Match statements compare a variable to multiple possible values. It executes the corresponding block of code when a match is found. It's a cleaner and more efficient way of writing a series of if-statements.

15 In the rest of the function, it simply plays the fitting animation that matches with `anim_state`.

```
20  enum AnimationState
21  {
22      IDLE,
23      WALKING,
24      JUMPING,
25  }
26
27  func handle_animation(anim_state: AnimationState) -> void:
28      match anim_state:
29          AnimationState.IDLE:
30              if anim.current_animation != "PlayerLibrary/IdlePose":
31                  anim.play("PlayerLibrary/IdlePose")
32          AnimationState.JUMPING:
33              if anim.current_animation != "PlayerLibrary/Jump":
34                  anim.play("PlayerLibrary/Jump")
35          AnimationState.WALKING:
36              if anim.current_animation != "PlayerLibrary/Walk":
37                  anim.play("PlayerLibrary/Walk")
38
```

16 Navigate directly above **TODO 1**.

The starter code uses the player's input to move Codey and handle animations. Notice how `is_jumping` is used to determine which animation is run when the player is not jumping.

```
50      # Move
51      if direction != Vector3.ZERO:
52          player_velocity.x = direction.x * speed
53          player_velocity.z = direction.z * speed
54
55          if not is_jumping:
56              handle_animation(AnimationState.WALKING)
57
58          # Determine facing direction (cardinal only)
59          if direction.x > 0:
60              target_rotation_y = deg_to_rad(90)
61          elif direction.x < 0:
62              target_rotation_y = deg_to_rad(-90)
63          elif direction.z > 0:
64              target_rotation_y = deg_to_rad(0)
65          elif direction.z < 0:
66              target_rotation_y = deg_to_rad(180)
67      else:
68          player_velocity.x = move_toward(player_velocity.x, 0.0, speed)
69          player_velocity.z = move_toward(player_velocity.z, 0.0, speed)
70
71          if not is_jumping:
72              handle_animation(AnimationState.IDLE)
73
```

17 If the `is_jumping` variable is removed and `is_on_floor()` was used instead, Codey's animation could not be changed mid-air!

```
48      # Move
49      if direction != Vector3.ZERO:
50          player_velocity.x = direction.x * speed
51          player_velocity.z = direction.z * speed
52
53          if is_on_floor():
54              handle_animation(AnimationState.WALKING)
55
56          # Determine facing direction (cardinal only)
57          if direction.x > 0:
58              target_rotation_y = d
59          elif direction.x < 0:
60              target_rotation_y = d
61          elif direction.z > 0:
62              target_rotation_y = deg_to_rad(0)
63          elif direction.z < 0:
64              target_rotation_y = deg_to_rad(180)
65      else:
66          player_velocity.x = move_toward(player_velocity.x, 0.0, speed)
67          player_velocity.z = move_toward(player_velocity.z, 0.0, speed)
68
69          if is_on_floor():
70              handle_animation(AnimationState.IDLE)
```

We can't change animations mid-air

If Codey walks off a platform, and while falling, the player releases all movement buttons, Codey will continue the walking animation instead of switching to idle. This is why the `is_jumping` variable is so important!

18 Fix the animations by setting `is_jumping` to `false` when Codey is on the floor.

Set the player's **y velocity** to **0** when Codey is on the ground.

```
74  # --------------------
75  # TODO 1
76  # Jumping and gravity bugfixing
77  # --------------------
78  if is_on_floor():
79      
80      if Input.is_action_just_pressed("jump"):
81          player_velocity.y = jump_velocity
82          is_jumping = true
83          handle_animation(AnimationState.JUMPING)
84  else:
85      player_velocity.y -= gravity * delta
86
```

19 Check the code! Update the script as needed.

```
74  # --------------------
75  # TODO 1
76  # Jumping and gravity bugfixing
77  # --------------------
78  if is_on_floor():
79      player_velocity.y = 0
80      is_jumping = false
81      if Input.is_action_just_pressed("jump"):
82          player_velocity.y = jump_velocity
83          is_jumping = true
84          handle_animation(AnimationState.JUMPING)
85  else:
86      player_velocity.y -= gravity * delta
87
```

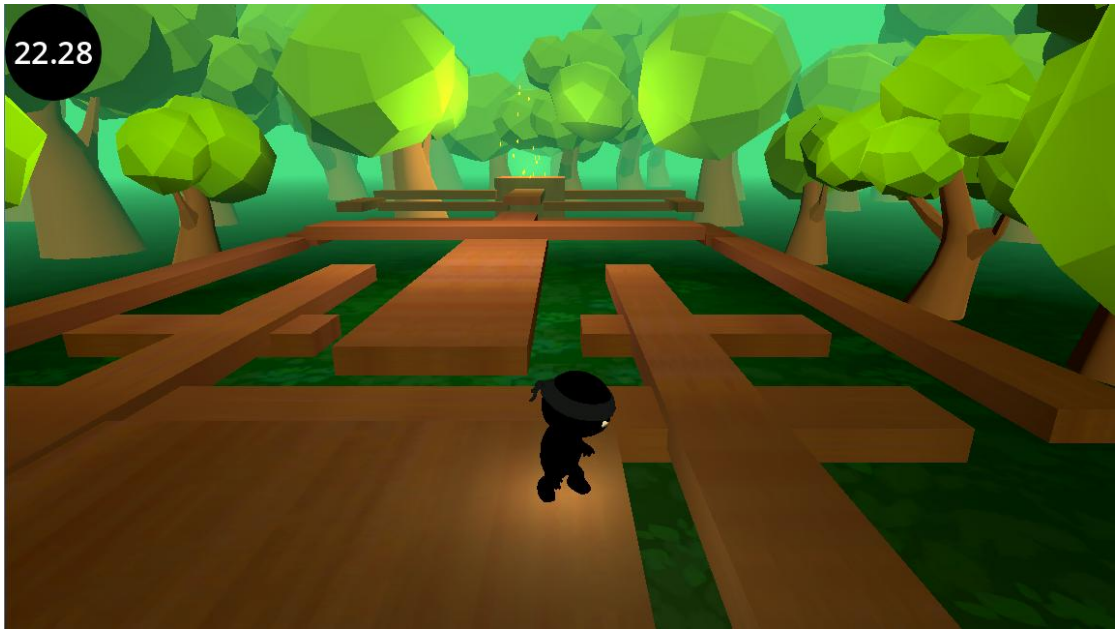
Pro Tip:

This code works because it is placed *before* the jumping code, so the values are overwritten when Codey jumps. The next frame after jumping, Codey will no longer be on the floor and this code won't run!

20

Playtest the project!

Codey should now return to normal animations after landing from a jump and should fall off platforms at the correct gravity.



Pause for **Sensei Stop #1**!

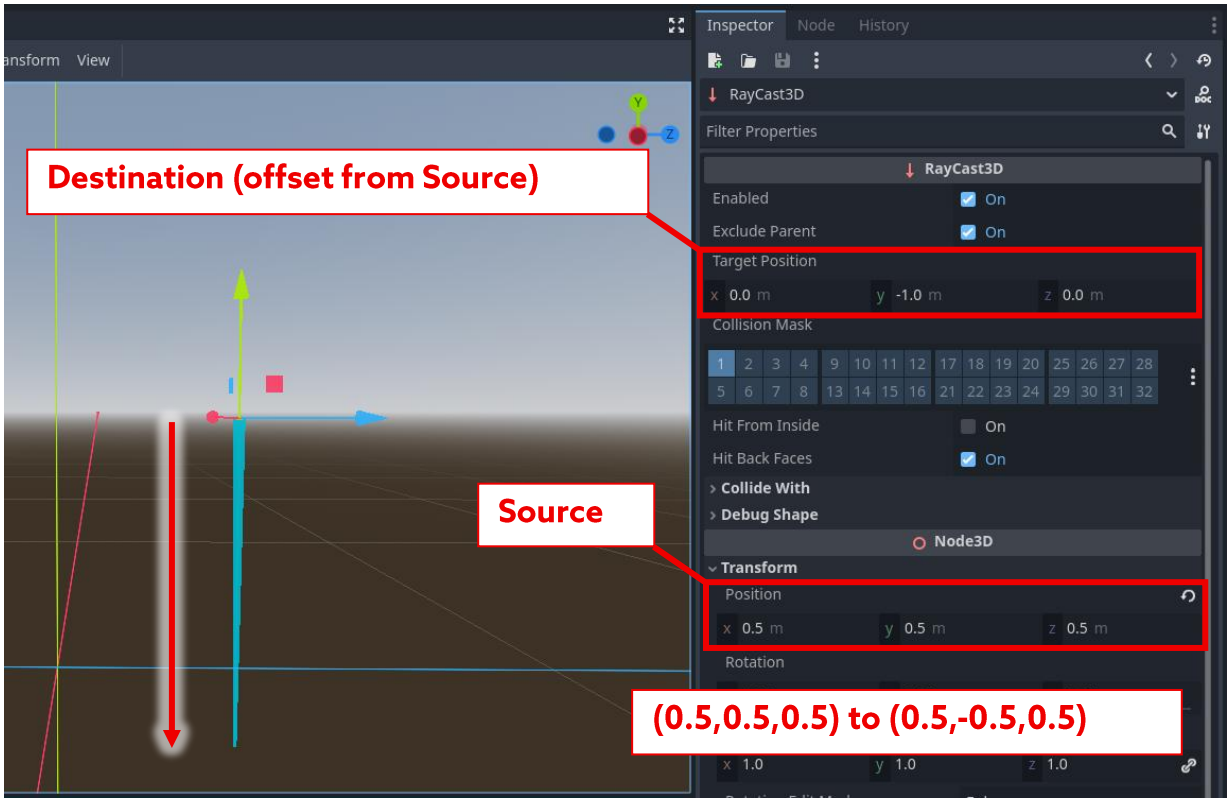
Check in with a Code Sensei before moving on. Ensure the Player cannot jump while mid-air and that Codey is animated properly.

Reminder: Save your work!

21 While playtesting the game, you may have realized that some platforms **disappear** when the player exits them and other platforms have **no collision** at all! Unfortunately, the player is never told which platforms have these functionalities. This is something that can be tackled with **Raycasting**!

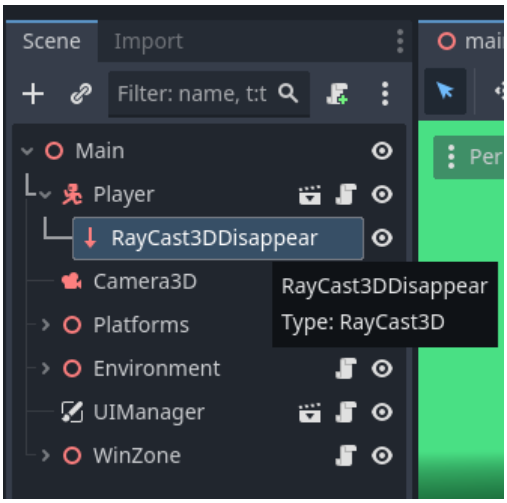
A common game design technique that is used to *gather information about surrounding objects* is called **Raycasting**. Godot's **RayCast3D** node creates a ray with a **source** and a **destination** (just like a Vector!).

The **source** is the node's position, and the **destination** is the offset from the node's position. Both may be tweaked in the Inspector or through code.



22

In **Scene**, add a **RayCast3D** node as a child to **Player** and rename it **RayCast3DDisappear**.

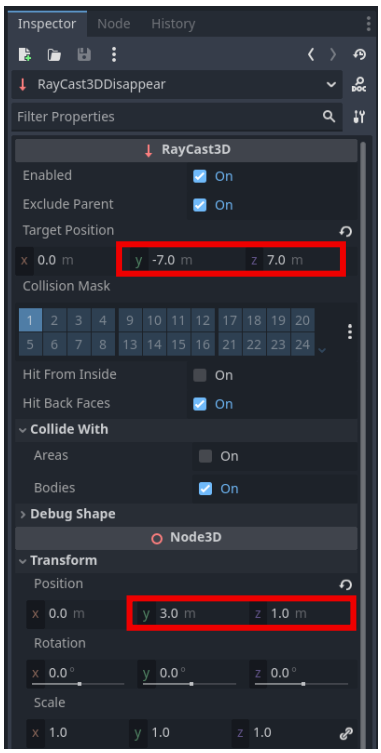


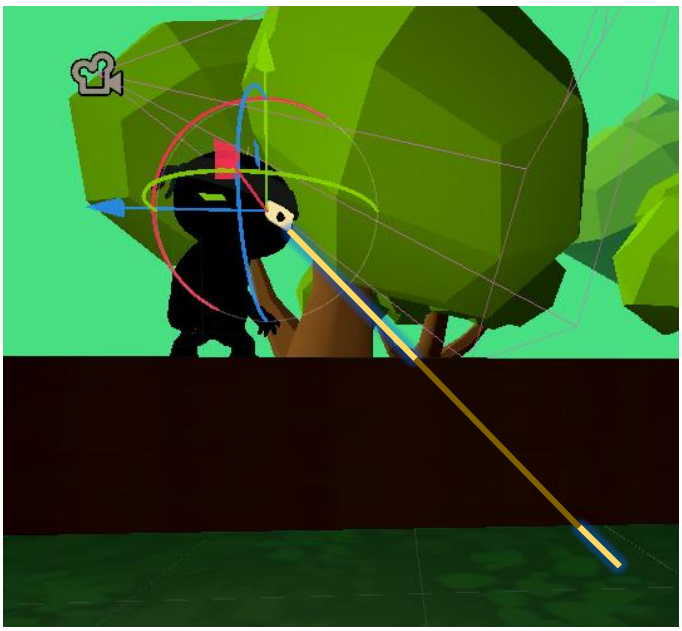
23

In the **Inspector**, set the **source** and the **destination** for the raycast!

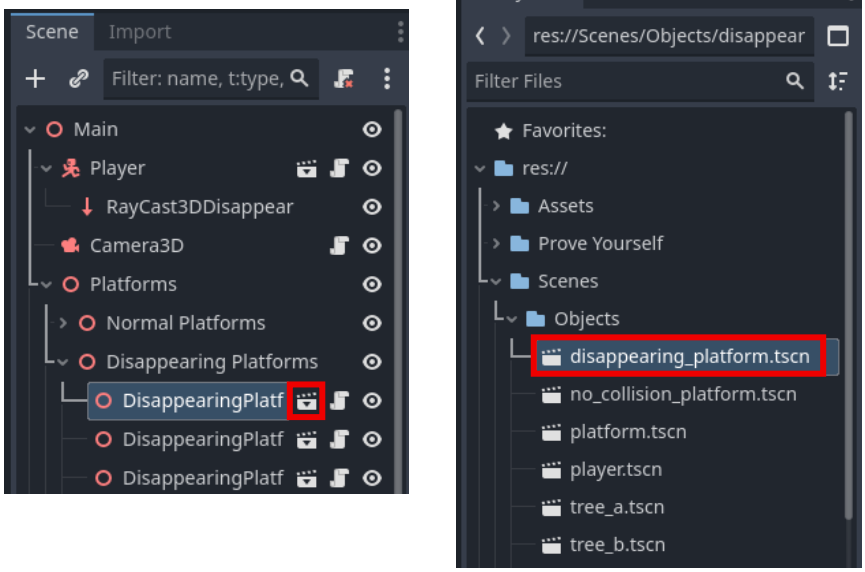
The values below will create a ray that starts at Codey's forehead and points down and forwards, through the ground.

Target Position (offset/destination): 0, -7.0, 7.0
Position (source): 0, 3.0, 1.0





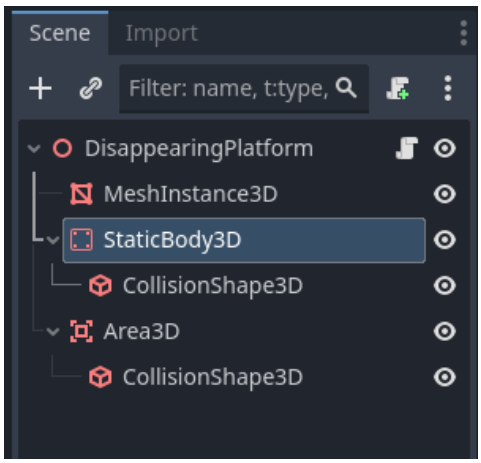
24 In **Scene** or **FileSystem**, navigate to **disappearing_platform.tscn**.

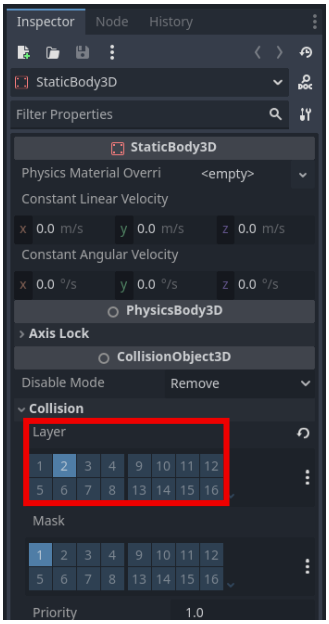


25 In **Robomania**, Collision Layers were used to ensure the **ProtonBlaster** can hit the **crushers**.

This time, Collision Layers will be used to detect different types of platforms. The **disappearing** platforms will be on a separate layer from the **no-collision** platforms.

Select **StaticBody3D** and in the **Inspector** under **CollisionObject3D**, click the **Collision** drop-down. Set the Layer to *only* **2**.





Pro Tip:

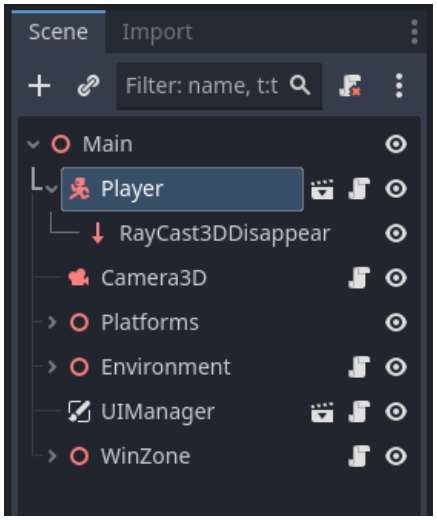
The Area3D is used in the DisappearingPlatform script to destroy the platform when the player exits it!

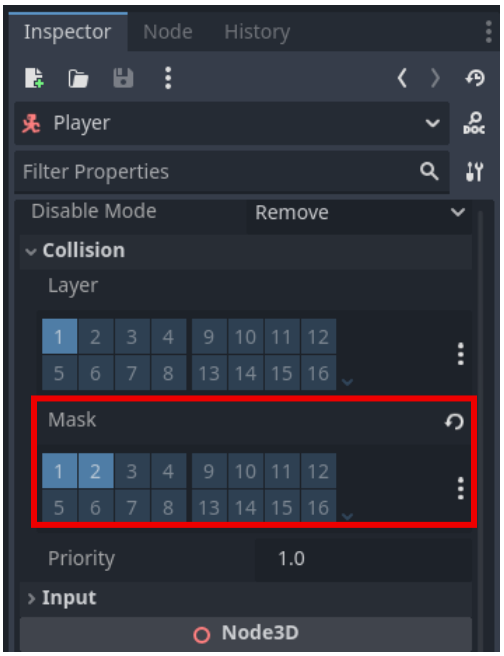
26

Return to the main scene. In **Scene**, select **Player**.

In the Inspector under **CollisionObject3D**, click the **Collision** drop-down. Set the Mask to **1** *and* **2**.

If **2** is not selected, then Codey will phase right through the disappearing platforms since Codey would *only* collide with objects on Layer **1**.





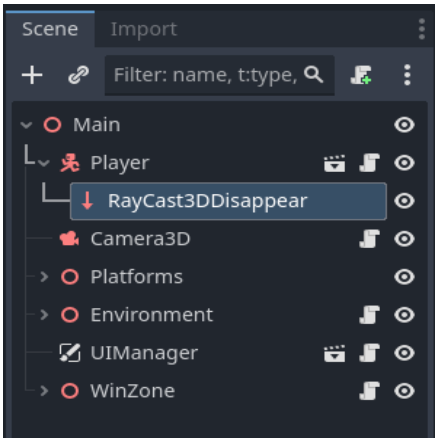
New Concept: Collision Mask

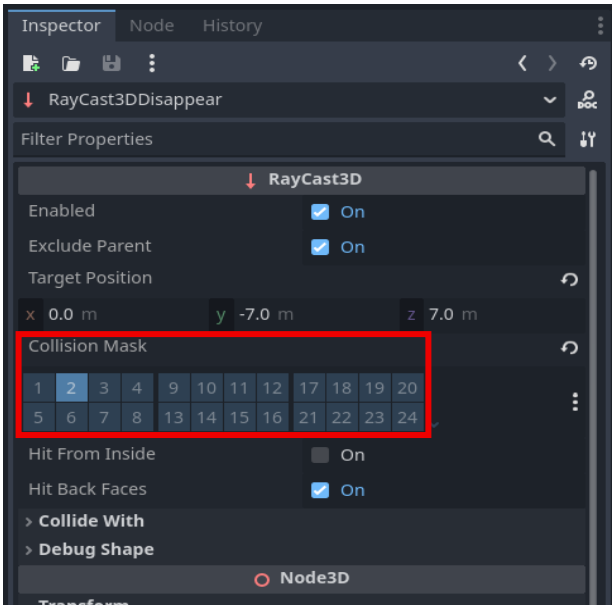
A body's **Collision Mask** describes which **Collision Layers** the current body will scan for to handle collisions. The body will ignore all objects that do not lie on any of the **Collision Mask's** layers.

27

In **Scene**, select **RayCast3DDisappear**.

In the Inspector, set its **Collision Mask** to *only* **2**. This Raycast now *only* collides with disappearing platforms!





28

The ray detects the disappearing platforms, but it doesn't report anything to the player yet!

Navigate back to **player.gd** and locate **TODO 2**.

Declare an `onready ray_disappear` variable equal to the **RayCast3DDisappear** child node.

```
8   var player_velocity: Vector3 = Vector3.ZERO
9   var target_rotation_y: float = rotation.y
10  var is_jumping: bool = false
11
12  # --------------------
13  # TODO 2
14  # Raycast onready
15  # --------------------
16
17
18  @onready var area = $Model/Area3D
19  @onready var anim = $Model/AnimationPlayer
20
21  enum AnimationState
22  {
```

29 Check the code! Update the script as needed.

```
12  # --------------------
13  # TODO 2
14  # Raycast onready
15  # --------------------
16  @onready var ray_disappear = $RayCast3DDisappear
17
```

30 To programmatically detect when a raycast collides with an object on its masked collision layers, **RayCast3D** has a handy method called `is_colliding()`.

> `is_colliding()`: part of the RayCast2D/3D classes, returns whether the given RayCast is colliding with any objects on its masked collision layers.
>
> **Parameters:** *none*
>
> **Returns (*boolean*):** whether the target is colliding with any objects on its masked collision layers

31 Navigate to **TODO 3** towards the bottom of the script.

Underneath, write an `if`-statement to check if `ray_disappear` is colliding with anything on its masked collision layers.

```
94      # Smooth rotation toward cardinal direction
95      rotation.y = lerp_angle(rotation.y, target_rotation_y, rotation_speed)
96
97      # --------------------
98      # TODO 3
99      # Raycast collisions
100     # --------------------
101     if
102
```

32 For debugging purposes, it's useful to display the name of the node the raycast is colliding with.

To do so, the node itself must be retrieved first. **RayCast3D** has another method called `get_collider()` that does just that!

> `get_collider()`: part of the RayCast2D/3D classes, returns the first object that the ray intersects with, starting from its source.
>
> **Parameters:** *none*
>
> **Returns (*Node*):** the first object that the ray intersects with, starting from its source

33 Inside the `if`-statement, write a line of code to declare an `other_object` variable and assign it to the first object that `ray_disappear` collides with.

```
97      # --------------------
98      # TODO 3
99      # Raycast collisions
100     # --------------------
101     #if ???:
102         
103
```

34 Since the collision from the raycast was with the platform's **StaticBody3D**, the **StaticBody3D**'s parent must be retrieved to get the platform's name.

The `get_parent()` method from Node can be used to access this information!

> `get_parent()`: part of the Node class, returns the parent of this node.
>
> **Parameters:** *none*
>
> **Returns (*Node*):** the parent of the node. If there is no parent, returns null instead

35 On the next line, declare a `platform_node` variable and assign it the value of `other_object`'s parent.

```
97  # --------------------
98  # TODO 3
99  # Raycast collisions
100 # --------------------
101 #if ???:
102     #other_object ???
103     
104 
```

36 Now, the player needs to be informed of the danger ahead of them! Print some warning text in quotes. After the quotes – but still inside the parentheses – type a `+` then access `platform_node.name`.

Write an `else`-statement to match the **if**, and inside of it, print `"Ray is not colliding"`.

```
97  # --------------------
98  # TODO 3
99  # Raycast collisions
100 # --------------------
101 #if ???:
102     #other_object ???
103     #platform_node ???
104     #print ???
105 #else:
106     #print ???
107 
```

New Concept: name

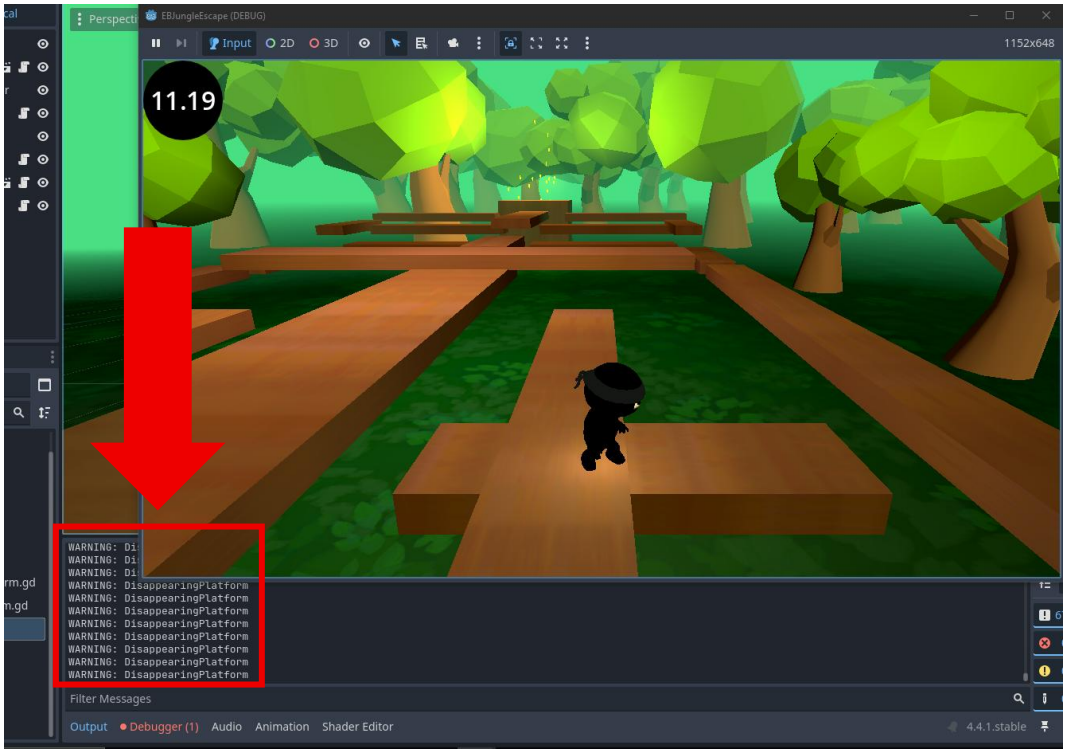
A property for every Node that stores its name as seen in the **Scene** panel.

37 Check the code! Update the script as needed.

```
97    # --------------------
98    # TODO 3
99    # Raycast collisions
100   # --------------------
101   if ray_disappear.is_colliding():
102       var other_object = ray_disappear.get_collider()
103       var platform_node = other_object.get_parent()
104       print("WARNING: " + platform_node.name)
105   else:
106       print("Ray is not colliding.")
107
```

38 Playtest the game. Use the Output panel to locate any disappearing platforms!

If the print statements are not working, refer back to the previous steps.



Pause for **Sensei Stop #2**!

Check in with a Code Sensei before moving on. Make sure that the raycasting for disappearing platforms is working properly and printing to **Output**.

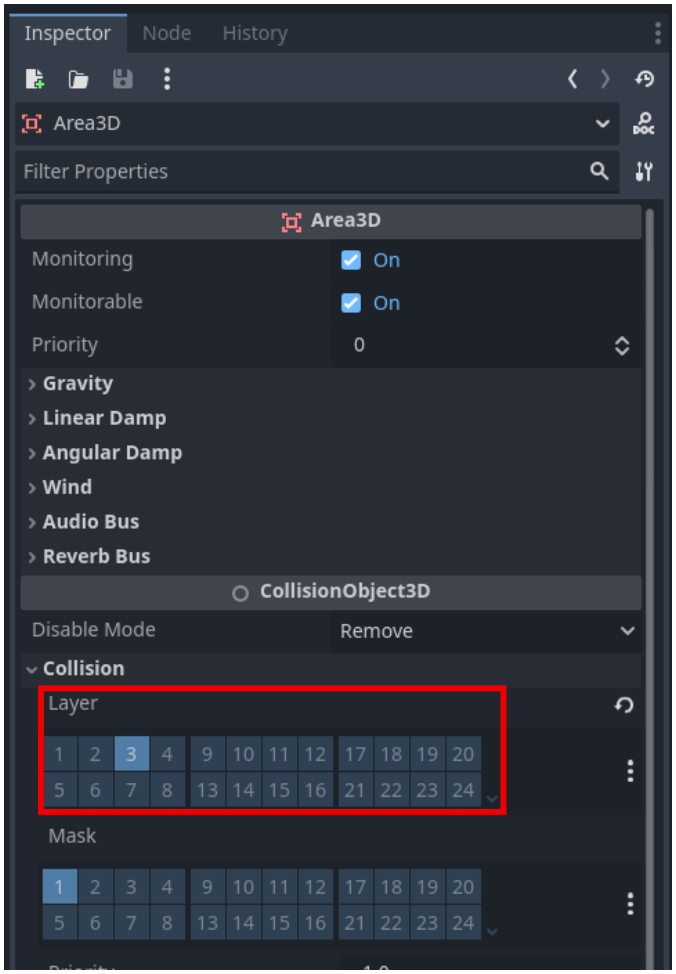
Reminder: Save your work!

39

Repeat the previous steps for another RayCast3D that checks for "no collision" platforms instead!

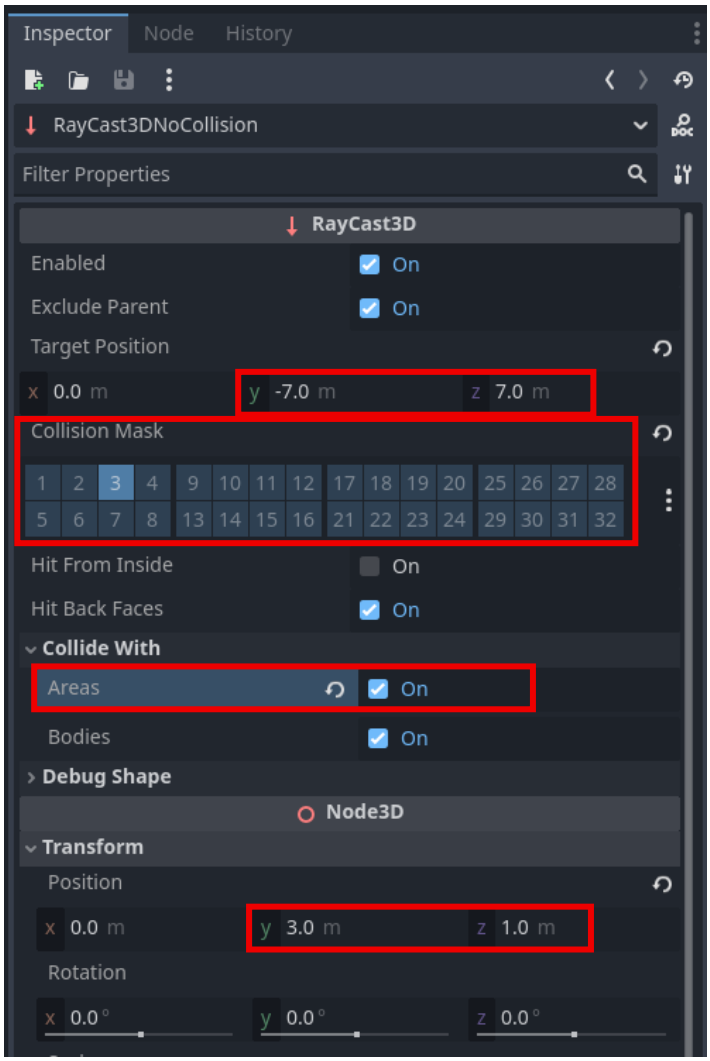
Start with the following:

- Add **RayCast3D** as a child to Player and rename it **RayCast3DNoCollision**. **NEW:** Inside of **no_collision_platform.tscn**, set the **Area3D**'s collision layer to *only* 3.



40 Set the properties for the **RayCast3D:**

- In **main.tscn**, set **RayCast3DNoCollision's** source and destination to be the same as **RayCast3DDisappear**.
  NEW: Set its collision mask to *only* 3 and **Collide With Areas** to **On**.



Pro Tip:

Codey doesn't need a Collision Mask with 3 enabled because Area3Ds are not solid colliders! Area3Ds merely keep track of the CollisionObjects that overlap them.

41 Make some changes in **player.gd:**

- In **player.gd** at **TODO 2**, declare an `onready ray_no_collision` variable.

```
12  # --------------------
13  # TODO 2
14  # Raycast onready
15  # --------------------
16  @onready var ray_disappear = $RayCast3DDisappear
17  #ray_no_collision ???
18
```

- At **TODO 3**, add an `elif`-statement with the same logic as the `if`-statement but this time for `ray_no_collision`. Use a slightly different print statement.
- Update the wording of the print call in the `else`-statement to say **`"Neither ray is colliding".`**

```
97      # --------------------
98      # TODO 3
99      # Raycast collisions
100     # --------------------
101     if ray_disappear.is_colliding():
102         var other_object = ray_disappear.get_collider()
103         var platform_node = other_object.get_parent()
104         print("WARNING: " + platform_node.name)
105     #elif ???:
106         #other_object ???
107         #platform_node ???
108         #print ???
109     #else:
110         #print ???
111
```

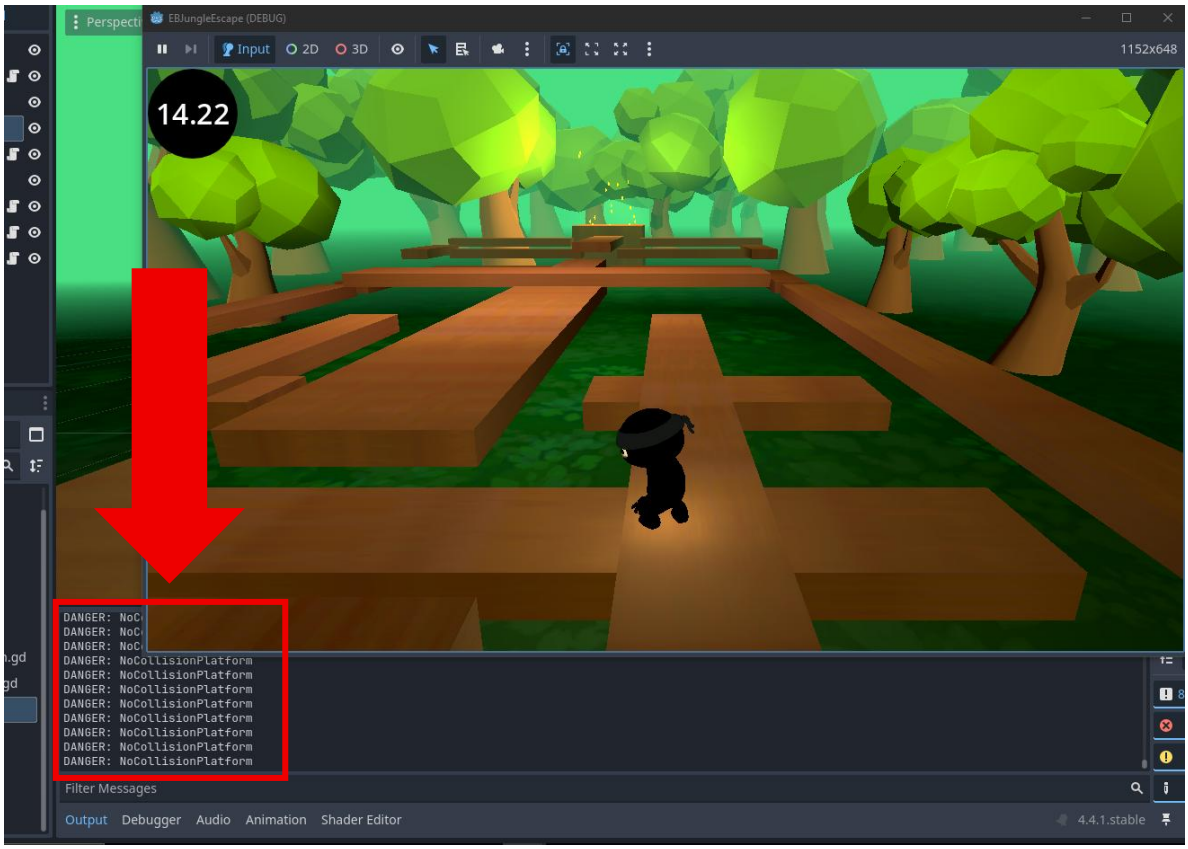
42 Check the code! Update the script as needed.

The "no collisions" print statement should have *a different* warning text than the "disappear" one to help distinguish the platforms.

```
12  # --------------------
13  # TODO 2
14  # Raycast onready
15  # --------------------
16  @onready var ray_disappear = $RayCast3DDisappear
17  @onready var ray_no_collision = $RayCast3DNoCollision
18
```

```
97      # --------------------
98      # TODO 3
99      # Raycast collisions
100     # --------------------
101     if ray_disappear.is_colliding():
102         var other_object = ray_disappear.get_collider()
103         var platform_node = other_object.get_parent()
104         print("WARNING: " + platform_node.name)
105     elif ray_no_collision.is_colliding():
106         var other_object = ray_no_collision.get_collider()
107         var platform_node = other_object.get_parent()
108         print("DANGER: " + platform_node.name)
109     else:
110         print("Neither ray is colliding.")
111
```

43 Playtest the game. Use the **Output** panel to navigate the jungle and escape to the beach! What's your best time?



Pause for **Sensei Stop #3**!

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

Before submitting, check in with a Code Sensei to check that the rays for NoCollision platforms work properly then reflect on the following:

- ☐ What did you learn about Enums, Match statements, and Raycasting?
- ☐ What did you enjoy most when creating this project?
- ☐ What was something you found difficult and why?

Reminder: Save your work!