CODE NINJAS®

Silver Belt Ninja Guide

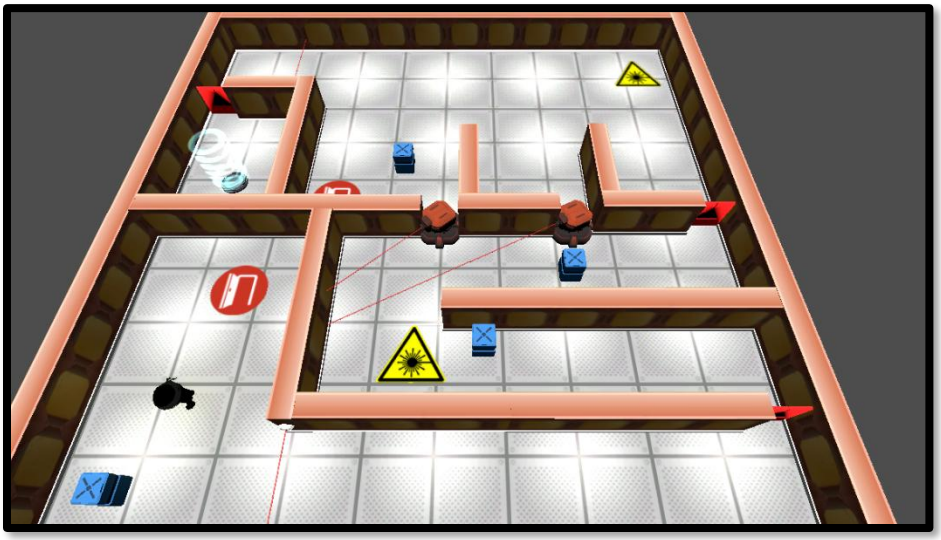
Activity 06: Evil Fortress of Doctor Worm

Level Design

Level Design is the practice of planning and building spaces for video games. *How* a level is designed can shape player behavior, and how the level is interacted with. When designing a level, it is important to provide the player with the necessary resources to win, while providing a level of difficulty that keeps the game challenging, but enjoyable.

When designing a level, consider the following:

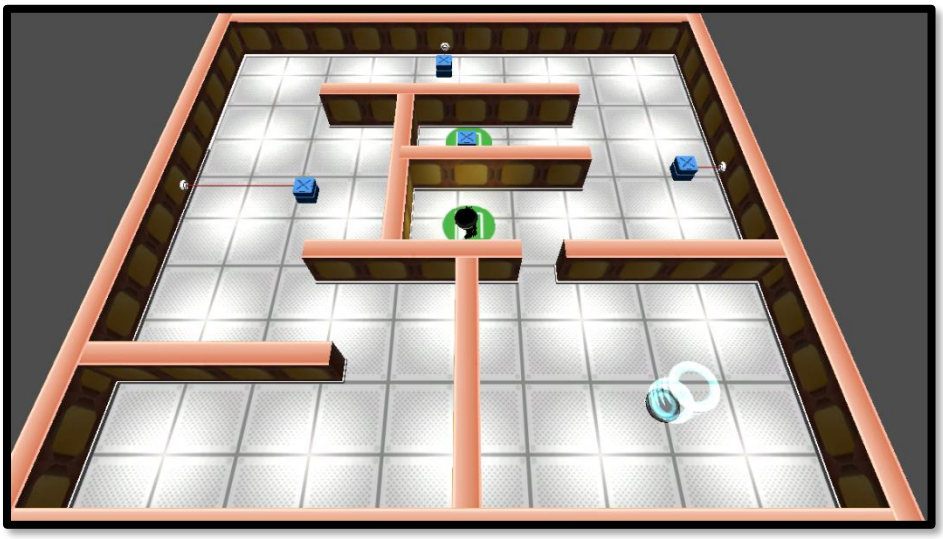
- What are the different ways the player can interact with the level?
- Can the level be completed in multiple ways?
- What tools might the player need to complete the level?
- Where will the player encounter these tools?
- What challenges will the Player encounter in the level?



Activity 06: Evil Fortress of Doctor Worm

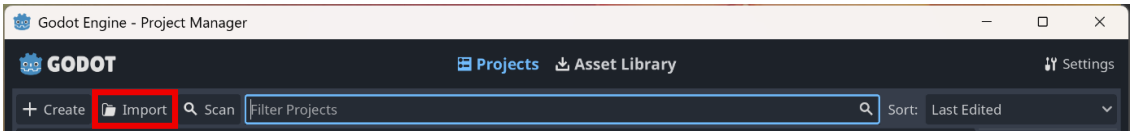
In this project, you will write scripts and connect signals with code to help Codey escape Doctor Worms' Evil Fortress. You'll design levels with the right amount of difficulty, using objects like switches and doors.

By the end of this activity, you will have explored level design and connecting signals through code to create triggers.



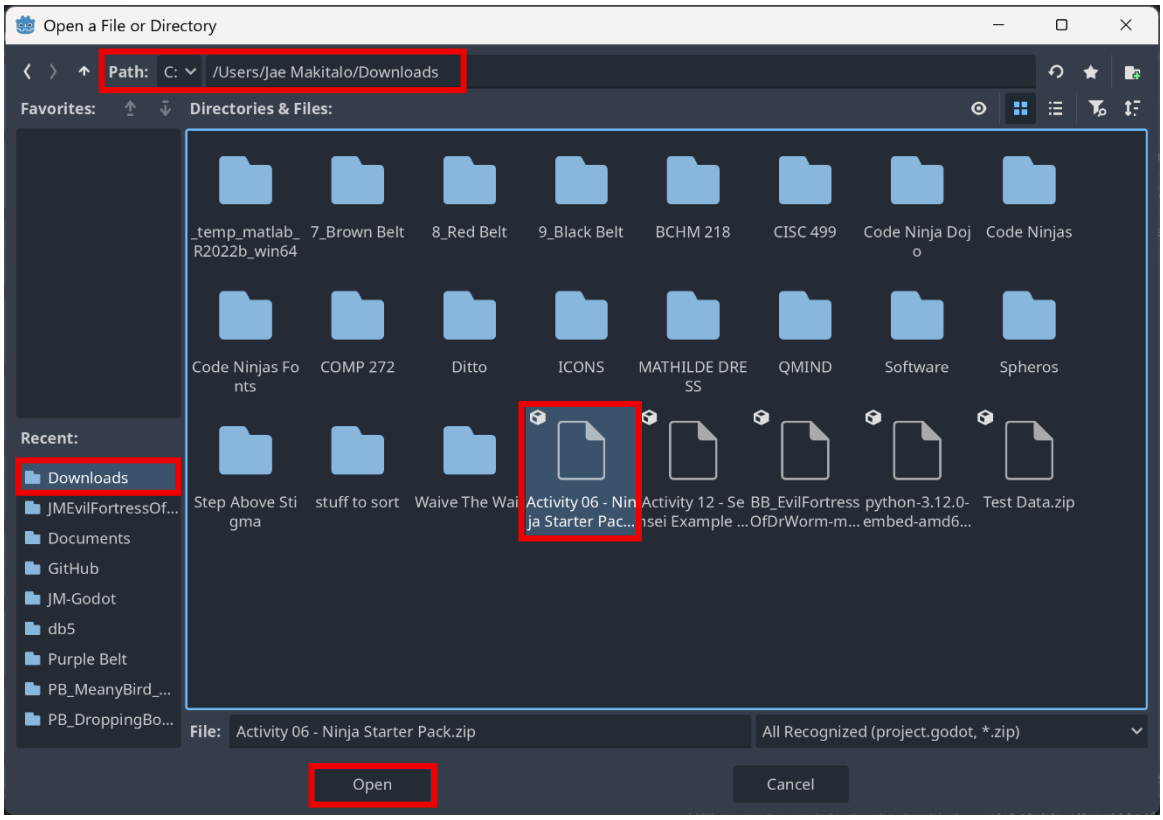
1 Begin the project by **importing** the starter code. This approach maintains the Input Map settings and other properties in the project file.

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 06 – Ninja Starter Pack.zip** and click **Open**.

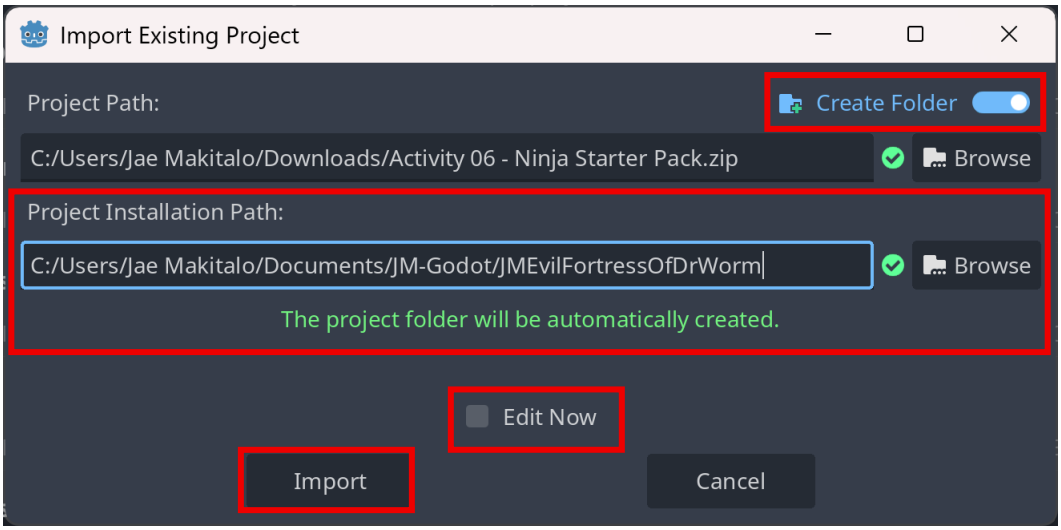


Pro Tip:

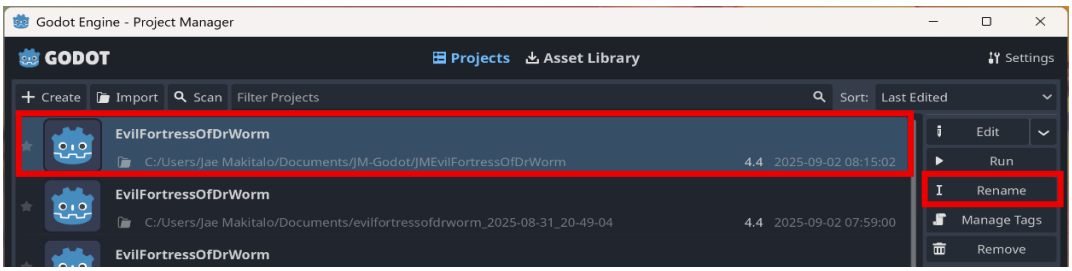
The folder should be **zipped** and must contain a **project.godot** file to be properly imported.

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

Uncheck Edit Now and click **Import**.

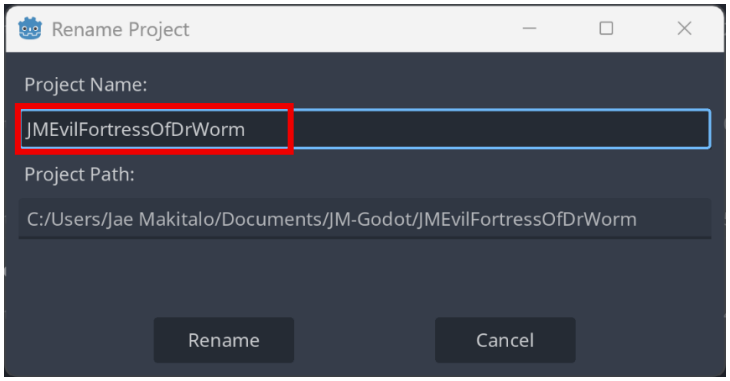


4 The project will appear at the top. Click on the project and select **Rename** on the right.



5 Update the Project Name to [YourInitials]EvilFortressofDrWorm.

Do not click Rename until after the Sensei stop!



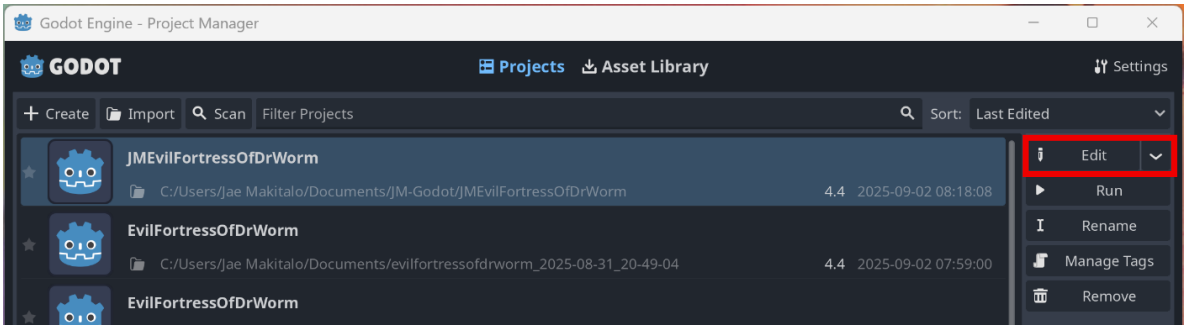
Pause for **Sensei Stop #1**!

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

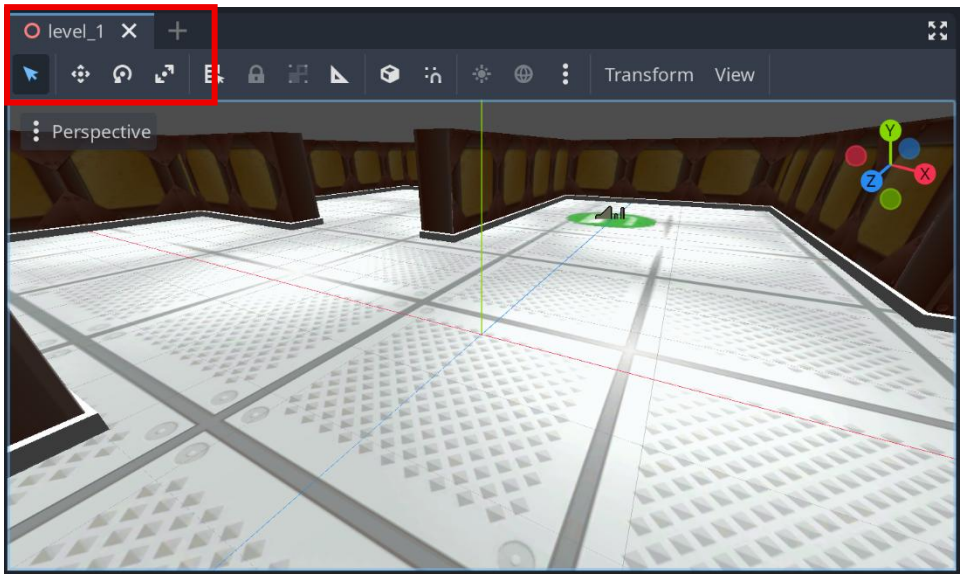
Click **Rename** when complete.

Reminder: Save your work!

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



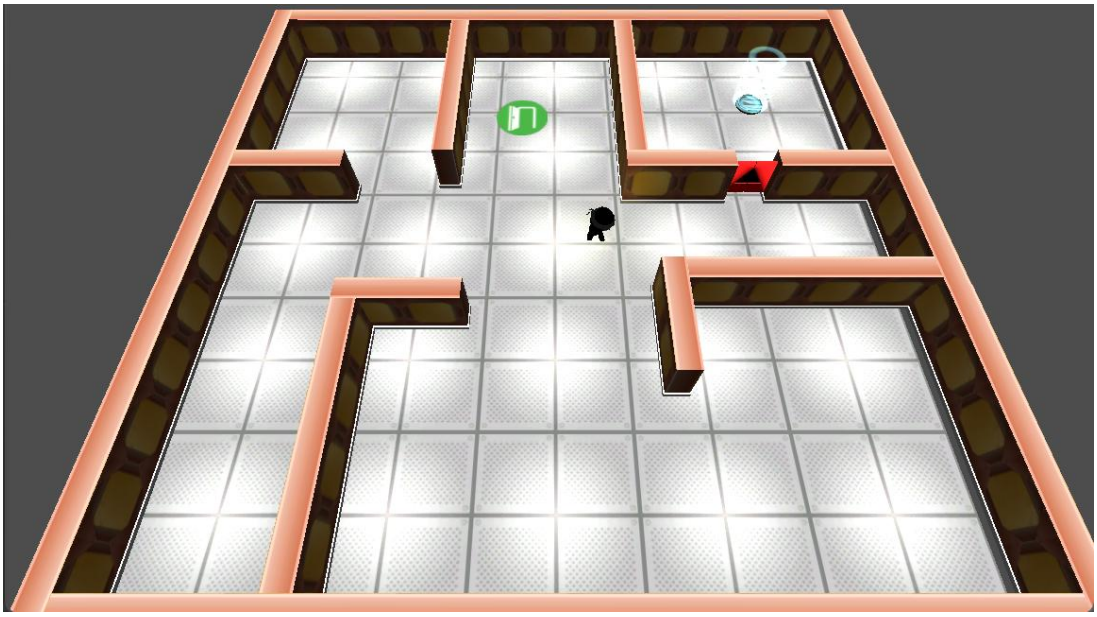
7 After the project has finished loading, the **level_1** scene will open. This is the game's main scene.



8 Playtest the project.

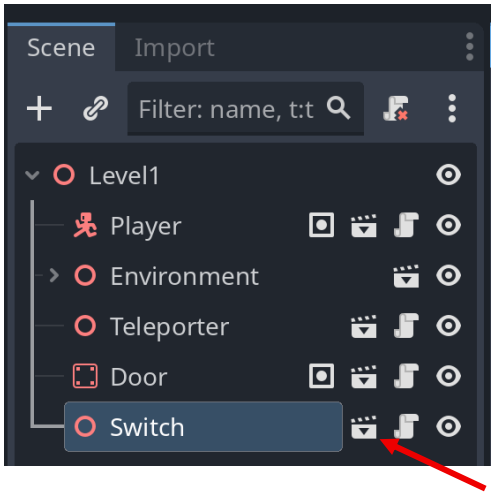
Because the project was imported, the main scene has already been set and does not need to be set again before playtesting.

Codey can be moved by using the direction buttons or the WASD keys on the keyboard.



The current scene has a switch, door, and teleporter. What happens when Codey stands on the switch?

9 When Codey stands on the switch, the door does not open.

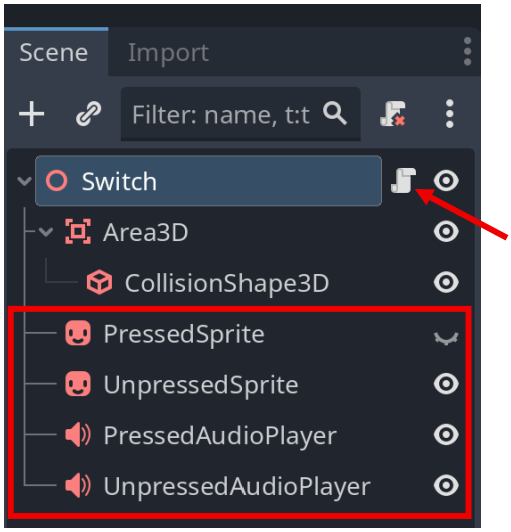


In Scene, click on the **scene icon** beside **Switch** to open the switch.tscn scene.

Pro Tip:

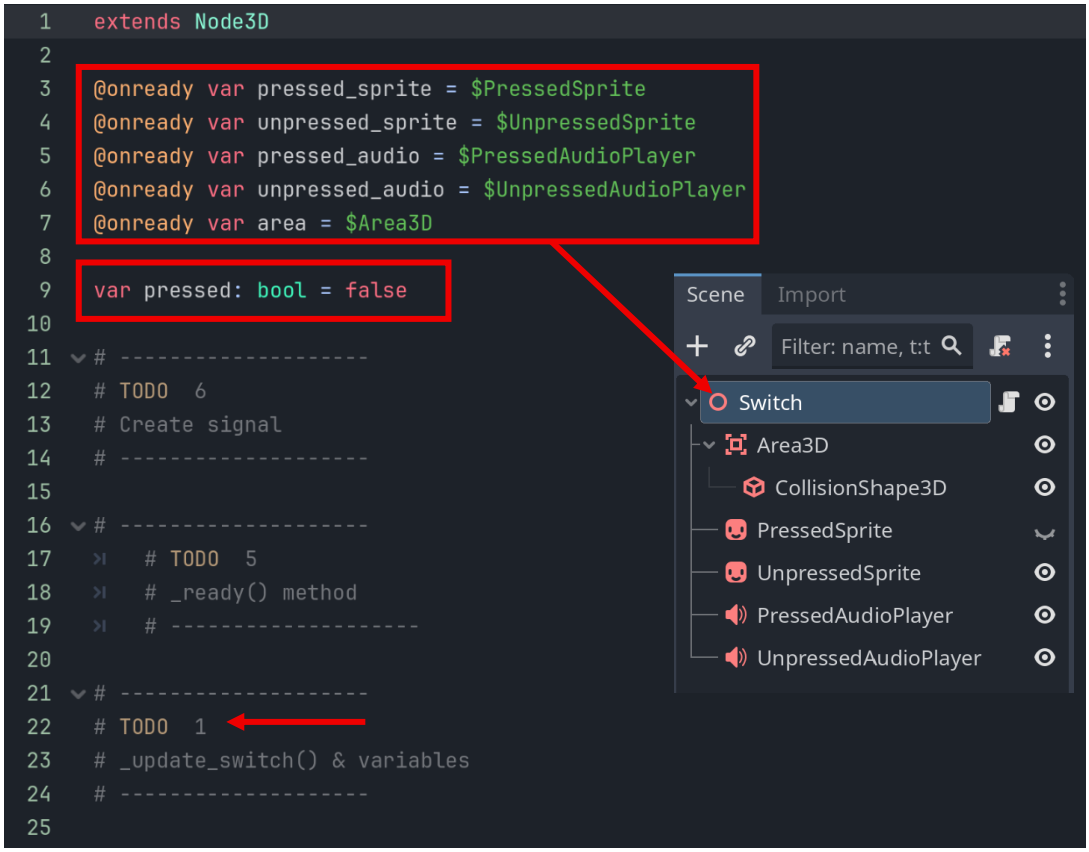
Click the **arrow** beside **Environment** to collapse the Environment and Wall nodes.

10 The Switch scene contains two **Sprite3D** nodes and two **AudioStreamPlayer3D** nodes to show when the switch is toggled on and off.



Click the script icon beside **Switch** to open the **switch.gd** script.

11 The script contains variables to access nodes in the Switch scene and a **`pressed`** variable to check if the switch has been pressed.



The **switch.gd** script needs to be coded to do the following:

- Toggle the switch on when overlapped.
- Toggle the switch off when it's no longer overlapped.
- Update the switch's sprite image.
- Send a signal to the door to open and close.

Scroll down to **TODO 1**.

12 Underneath TODO 1, create the `_update_switch()` function. The `_update_switch()` function takes one parameter, `_body`.

```
21  # --------------------
22  # TODO  1
23  # _update_switch() & variables
24  # --------------------
25  # _update_switch()
26      # var bodies
27      # var touching_switch_presser
28
```

Declare these two variables inside the `_update_switch()` function:

- `bodies`, which uses the method `get_overlapping_bodies(),` to get an array of PhysicsBody3Ds and GridMaps overlapping the Area3D node in the Switch node tree.
- `touching_switch_presser` set to `false`, which will be used to determine if the Area3D is being overlapped by an object or item that can trigger the switch.

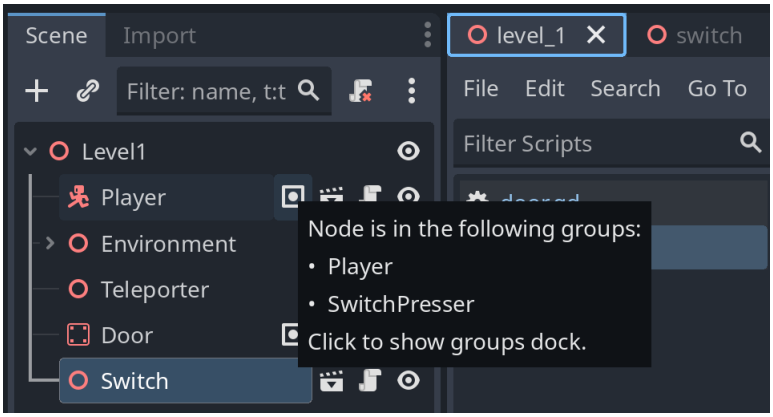
> `get_overlapping_bodies():` Returns a list of intersecting PhysicsBody3Ds and GridMaps.
>
> For performance reasons (collisions are all processed at the same time) this list is modified once during the physics step, not immediately after objects are moved.
>
> **Parameters:** *None*
>
> **Returns:** An array of overlapping bodies.

13 Check the code and update the script as needed.

```
20
21  # --------------------
22  # TODO  1
23  # _update_switch() & variables
24  # --------------------
25  func _update_switch(_body):
26      var bodies = area.get_overlapping_bodies()
27      var touching_switch_presser = false
28
```

14 Toggle back to the **level_1** scene and hover over the **group icon** beside the **Player node** to see what groups the player is part of.

The Player is in two groups, **Player** and **SwitchPresser**.



The **SwitchPresser** group can be used to tag objects that can turn switches on and off.

15 Underneath **TODO 2**, write a for-loop to check if the array of **bodies** contains a **body** that is in the **group** **SwitchPresser**. If a SwitchPresser is found, update the **touching_switch_presser** variable. **break** out of the for loop once a SwitchPresser is found.

```
29    # --------------------
30    # TODO  2
31    # Check SwitchPresser
32    # --------------------
33    # for ???:
34    #     # if ???:
35    #     #     touching_switch_presser ???
36    #     #     break
37
```

Pro Tip:

The method **is_in_group()** will be helpful here.

16 Check the code and update the script as needed.

```
29  # --------------------
30  # TODO  2
31  # Check SwitchPresser
32  # --------------------
33  for body in bodies:
34      if body.is_in_group("SwitchPresser"):
35          touching_switch_presser = true
36          break
37
```

17 Under **TODO 3**, use the `not` operator and the `pressed` and `touching_switch_presser` variables to write some code to do the following:

- If touching a SwitchPresser and the switch is not pressed, press the switch
- Else, if not touching a SwitchPresser and the switch is pressed, unpress the switch.

```
38  # --------------------
39  # TODO  3
40  # Toggle switch
41  # --------------------
42  # if ??? and ???:
43      # pressed ???
44  # elif ??? and ???:
45      # pressed ???
```

The not operator:

if **true** (returns **true**) if **false** (returns **false**) if **not true** (returns **false**)	if **true** and **not true** → if **true** and **false** (returns **false**)

18 Check the code and update the script as needed.

```
38    # --------------------
39    # TODO  3
40    # Toggle switch
41    # --------------------
42    if touching_switch_presser and not pressed:
43        pressed = true
44    elif not touching_switch_presser and pressed:
45        pressed = false
46
```

19 Under **TODO 4**, update which button sprite is visible depending on whether the switch is on or off.

```
47    # --------------------
48    # TODO  4
49    # update switch texture
50    # --------------------
51    # pressed_sprite ???
52    # unpressed_sprite ???
53
```

Notice how the code here is *outside* of the if statements.

Update the **pressed** and **unpressed** sprites' visibility using the **pressed** variable and the **not** operator. Think about what value the pressed variable returns when the switch is pressed.

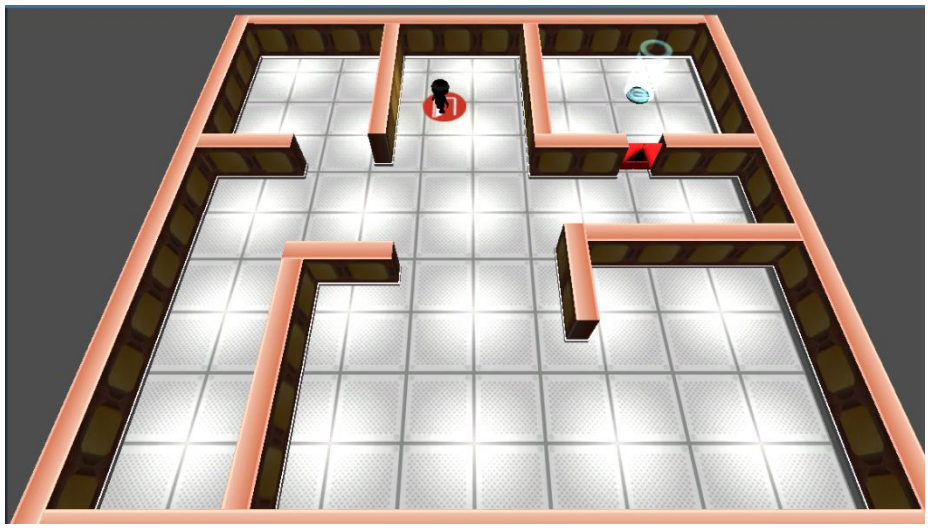
20 Check the code and think about *how* this code works.

```
47    # --------------------
48    # TODO  4
49    # update switch texture
50    # --------------------
51    pressed_sprite.visible = pressed
52    unpressed_sprite.visible = not pressed
53
```

Why might the sprite visibility be updated underneath the if-statement instead of inside of it?

21 Playtest the project.

What happens when Codey overlaps the switch? Does it change color? Does the door open?



In the **switch.gd** script, where is the `_update_switch()` function called?

22 The `_update_switch()` function is not called in the **switch.gd** script or connected to any signals.

Signals can be connected inside a script using code. Return to the **switch.gd** script and find **TODO 5**.

The signals will need to be connected right at the start of the game. This can be done using the `_ready()` method.

Underneath the TODO, declare the `_ready()` method.

```
15
16  # --------------------
17  # TODO  5
18  # _ready() method
19  # --------------------
20
```

23

The Area3D node in the Switch node tree will be used to check if a body has overlapped the switch. This *could* be done by connecting the `body_entered()` signal in the interface, but it can also be done with code using the same signal.

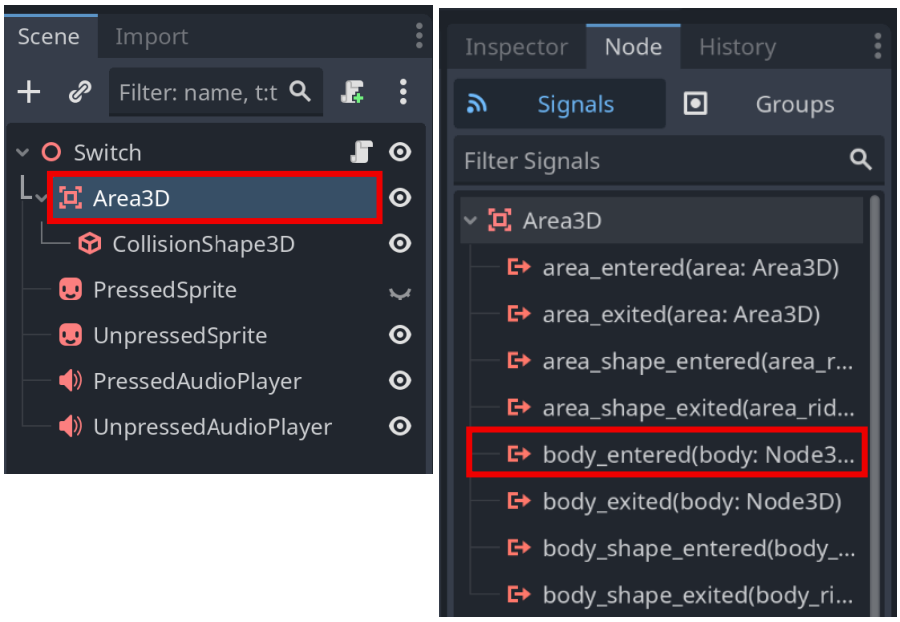
When connecting a signal with code, 4 things are needed:

- The node with the signal (**Area3D**)
- The signal (**body_entered**)
- The **connect()** method
- The signals receiver method (**_update_switch**)

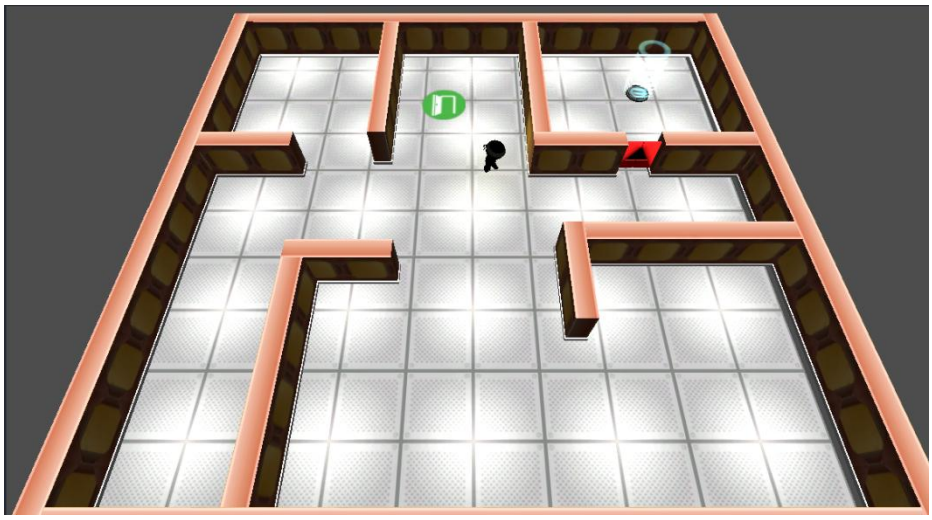
The **Area3D** node is accessed through the **area** variable which is part if the starter code variables at the top of the script.

```
16  # --------------------
17  # TODO  5
18  # _ready() method
19  # --------------------
20  func _ready() -> void:
21      area.body_entered.connect(_update_switch)
22
```

Notice that the receiver method inside of `connect()` is not a string and **does not have round brackets** after the name.



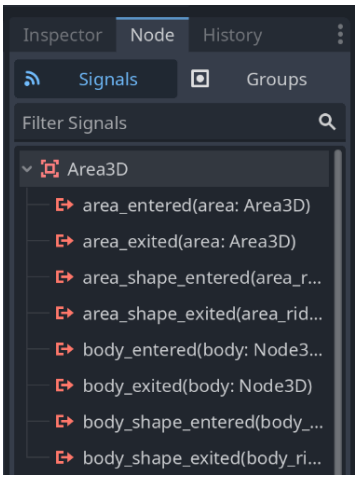
24 Save the script and playtest the project. What happens when Codey overlaps the button? What happens when Codey leaves the button?



25 When Codey exits the switch, it remains on. Another signal needs to be connected to turn the switch off when a SwitchPresser is no longer on it.

```
16  # --------------------
17  # TODO  5
18  # _ready() method
19  # --------------------
20  func _ready() -> void:
21      area.body_entered.connect(_update_switch)
22
```

Looking at the different signals for an Area3D node, what signal might be used?



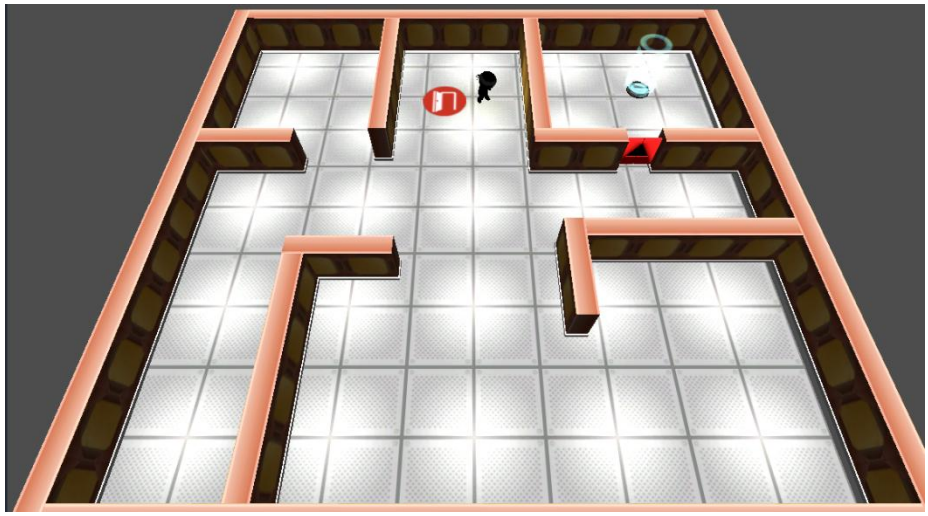
Try and code the signal.

26 Check the code and update the script as needed.

```
16  # --------------------
17  # TODO  5
18  # _ready() method
19  # --------------------
20  func _ready() -> void:
21      area.body_entered.connect(_update_switch)
22      area.body_exited.connect(_update_switch)
23
```

27 Playtest the project. Does the switch toggle on and off when Codey overlaps it?

The door will remain closed – this will be fixed in upcoming steps!



28

The graphics for the switch look good, but they can be improved by adding audio!

Return to the **switch.gd** script and find the **`if/elif`** statement under **TODO 3**.

Add code to play the pressed and unpressed audio when the switch is toggled on and off.

```
41    # --------------------
42    # TODO  3
43    # Toggle switch
44    # --------------------
45    if touching_switch_presser and not pressed:
46        pressed = true
47        pressed_audio.play()
48    elif not touching_switch_presser and pressed:
49        pressed = false
50        unpressed_audio.play()
51
```

Pause for **Sensei Stop #2**!

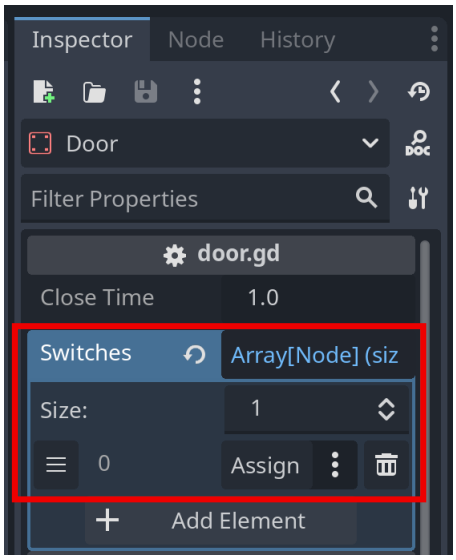
Check in with a Code Sensei before moving on. Make sure the code in the **switch.gd script** is correct.

The switch does not open the door yet – this part comes next.

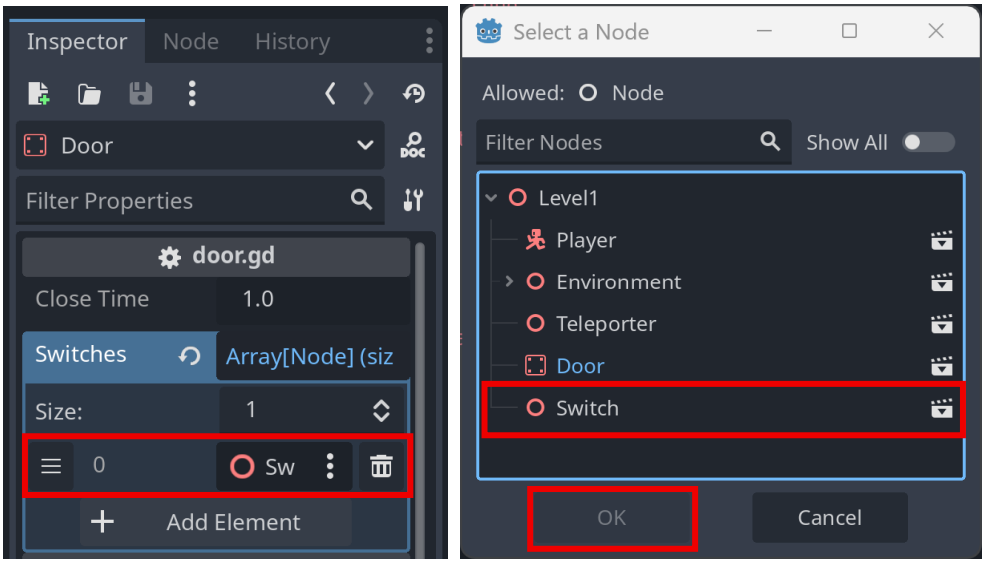
Reminder: Save your work!

29 The door does not open when the switch is triggered.

In the **Inspector** for **Door**, click on **Array[Node] (size)** and increase the size of the array from 0 to 1.



30 Click **Assign**, select **Switch** and click **OK** to set **element 0** in the Switches array to **Switch**.



31 Return to the **switch.gd** script and find **TODO 6**.

Create a signal `toggled`. This signal will emit a `bool` value to tell the door if the switch has been toggled on or off.

```
11  # --------------------
12  # TODO  6
13  # Create signal
14  # --------------------
15  signal toggled(pressed: bool)
16
```

32 The `toggled` signal needs to be emitted when the switch is toggled on or off.

Scroll down to **TODO 3** and emit the signal. The method `emit_signal()` emits a signal by name. Signals do not need to emit a value, such as a bool or int. In this case, toggled emits a bool value so the argument must also be passed into the `emit_signal()` method after the signal name.

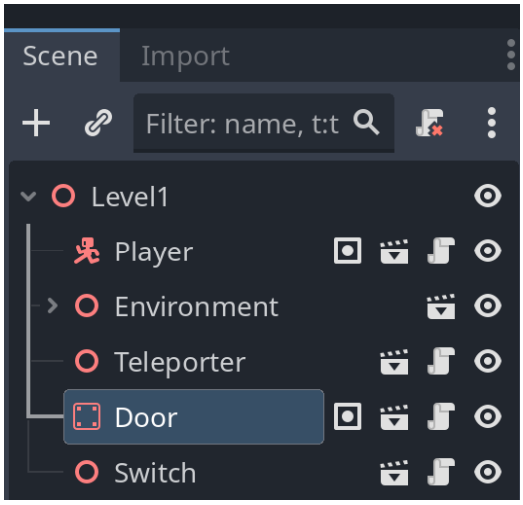
Notice how the name of the signal is passed into the **emit_signal()** method as a string.

```
42  # --------------------
43  # TODO  3
44  # Toggle switch
45  # --------------------
46  if touching_switch_presser and not pressed:
47      pressed = true
48      pressed_audio.play()
49      emit_signal("toggled", true)
50  elif not touching_switch_presser and pressed:
51      pressed = false
52      unpressed_audio.play()
53      emit_signal("toggled", false)
54
```

Save the script.

> `emit_signal():` part of the Object class, emits the given signal by name. The signal must exist as a built-in signal within this class or one of its inherited classes, or a user-defined signal.
>
> **Parameters:** *This method supports a variable number arguments, so parameters can be passed as a comma separated list.*
>
> **Returns:** Nothing, unless the signal does not exist or parameters are invalid, then an error is returned.

33 The signal needs to be received by the **door.gd** script. Open the **door.gd** script.



34 Notice the door.gd script contains quite a bit of code.

There are two variables, **`close_time`** and **`switches`**, which can be updated in the Inspector for Door.

There is also a function **`_switch_check()`** which checks if enough switches have been triggered to open the door.

Some code needs to be added to call the **`_switch_check()`** function when the signal toggled is emitted by a switch.

```
1   extends Node3D
2
3   @export var close_time: float = 1.0
4   @export var switches: Array[Node]
5
6   @onready var anim = $AnimationPlayer
7   @onready var audio = $AudioStreamPlayer3D
8   @onready var close_timer = $Timer
9
10  var _opened := false
11  var _switches_pressed: int = 0
12
13
14  func _ready():
15      close_timer.timeout.connect(_close_door)
16      close_timer.wait_time = close_time
17
18      # --------------------
19      # TODO  7
20      # connect switch to door
21      # --------------------
22
23
24  func _switch_check(is_pressed: bool):
25      if is_pressed:
26          _switches_pressed += 1
```

35 Since the `switches` variable is an array, multiple switches can be connected to a door. Each switch in the switches array will need to be connected to the `_switch_check()` function.

Under **TODO 7**, create a for-loop to check each switch in the array of switches.

```
18    # --------------------
19    # TODO  7
20    # connect switch to door
21    # --------------------
22    # for ???
23
```

36 Inside the `for-loop`, each switch needs to be connected to the `_switch_check()` function. This can be done using the `connect()` method, but the code will look different from before, since the `connect()` method is being called on a **node with a signal** instead of **directly on a signal**.

`connect()` method called on a node with a signal:

```
20    # connect switch to door
21    # --------------------
22    for switch in switches:
23        switch.connect()
24            Error connect(signal: StringName, callable: Callable, flags: int = 0)
25  func _switch_check(is_pressed: bool):
26    if is_pressed:
```

`connect()` method called directly on a signal:

```
16  # --------------------
17    # TODO  5
18    # _ready() method
19    # --------------------
20  func _ready() -> void:
21      area.body_entered.connect(_update_switch)
22      area.body_exited.connect(_update_switch)
23
```

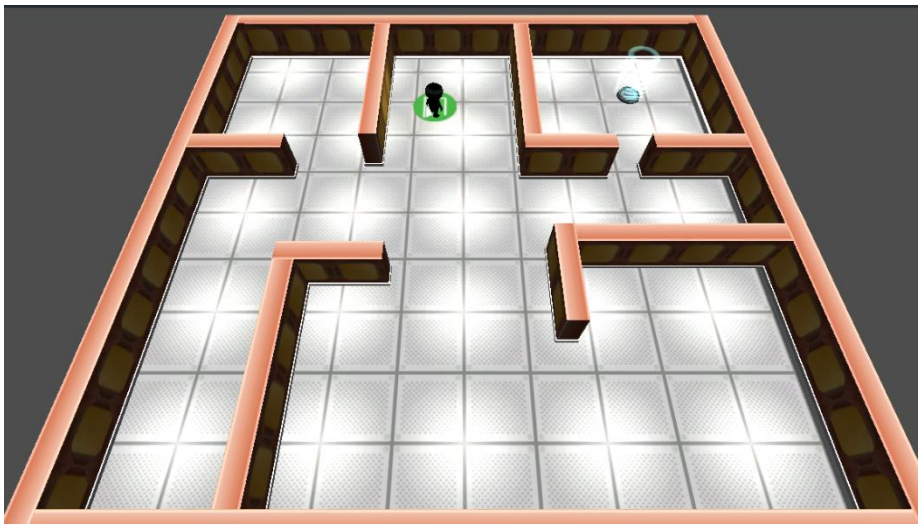
37 In this scenario, the `connect()` method needs two parameters:

- The **signal emitted** from the node that is going to be connected, as a **string.**
- The signals **receiver method**.

```
18    # --------------------
19    # TODO  7
20    # connect switch to door
21    # --------------------
22    for switch in switches:
23        switch.connect("toggled", _switch_check)
24
```

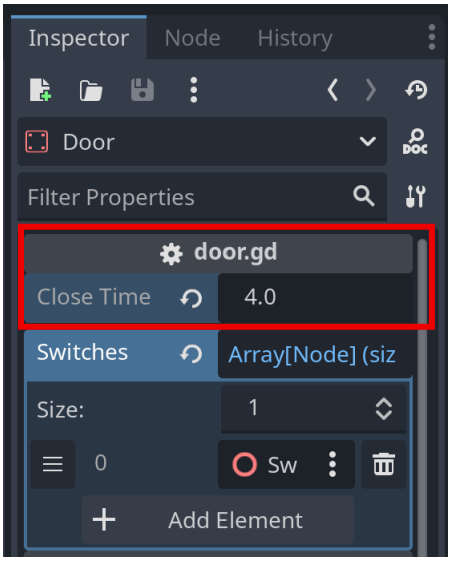
38 Playtest the project.

What happens when Codey stands on the switch? What happens when Codey leaves the switch?



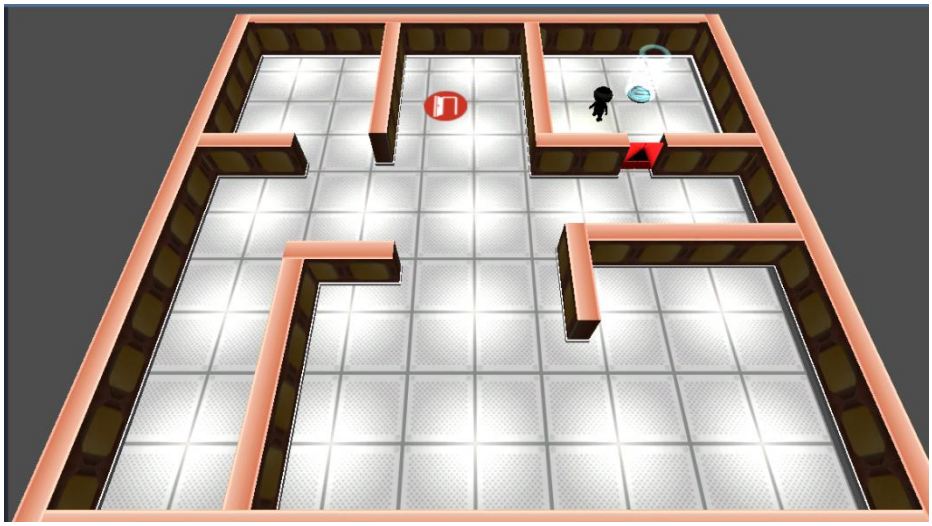
39 The door now opens when Codey stands on the switch, but there is not enough time to pass through the door before it closes again!

In the **Inspector** for **Door**, increase the value of **Close Time** to give Codey enough time to escape.



40 Playtest the project to make sure Codey can reach the teleporter.

Adjust the value of **close time** if needed. When teleported to the next level, close the playtest window and save the scene.



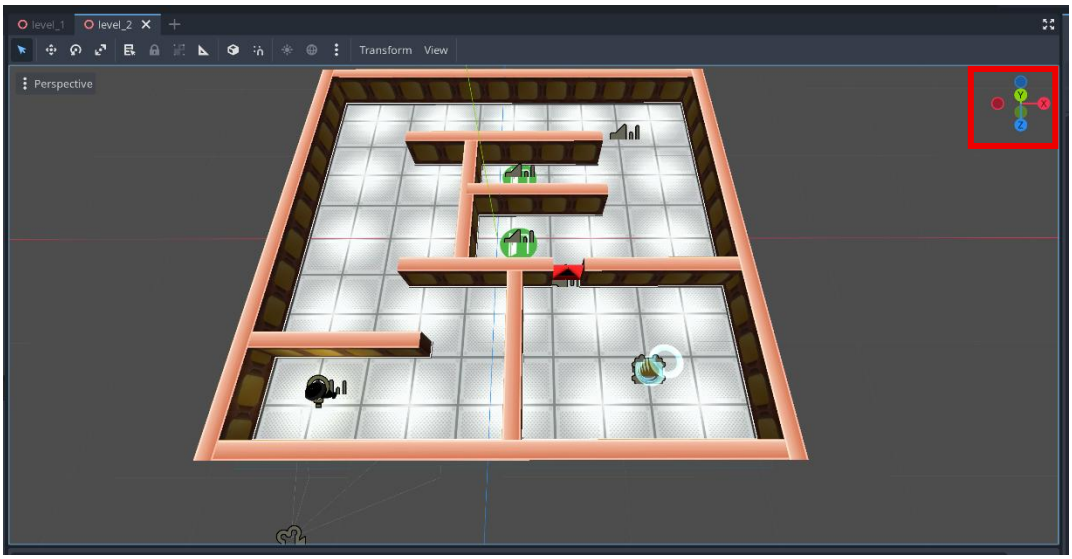
Pause for **Sensei Stop #3**!

Check in with a Code Sensei before moving on. Make sure the code in the **switch.gd** and **door.gd** scripts is correct, and that the switch and door are working properly.

Reminder: Save your work!

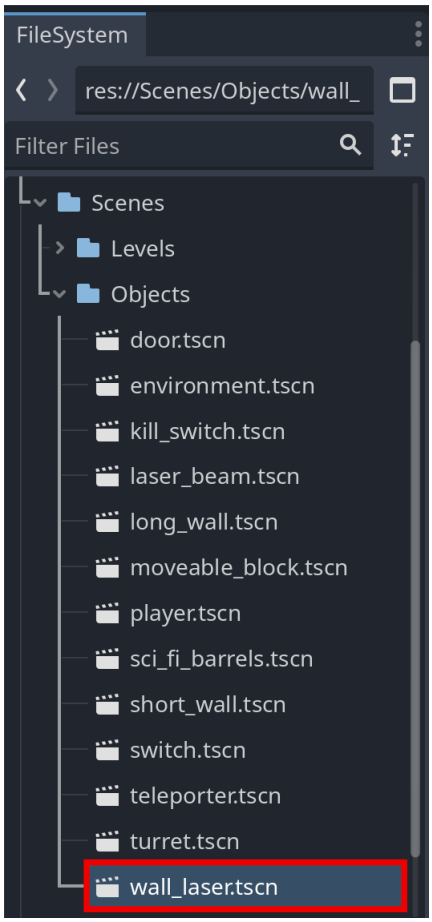
41 In **FileSystem**, find **level_2.tscn** scene inside the **Levels** folder in **Scenes**. Double click **level_2.tscn** to open it.

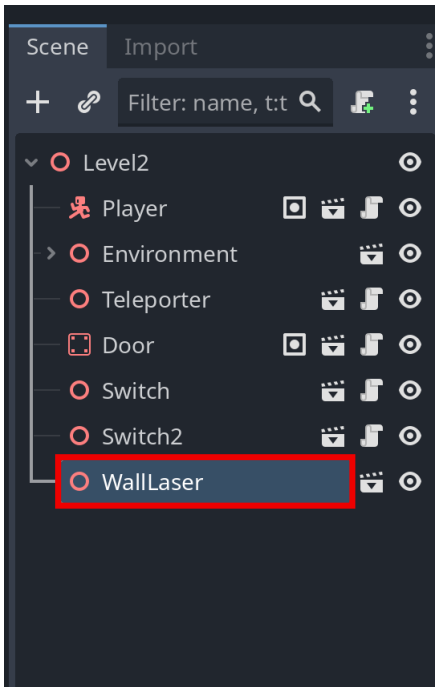
42 In the **3D workspace**, use the mouse or the manipulator gizmo to adjust the view so the whole level can be seen.



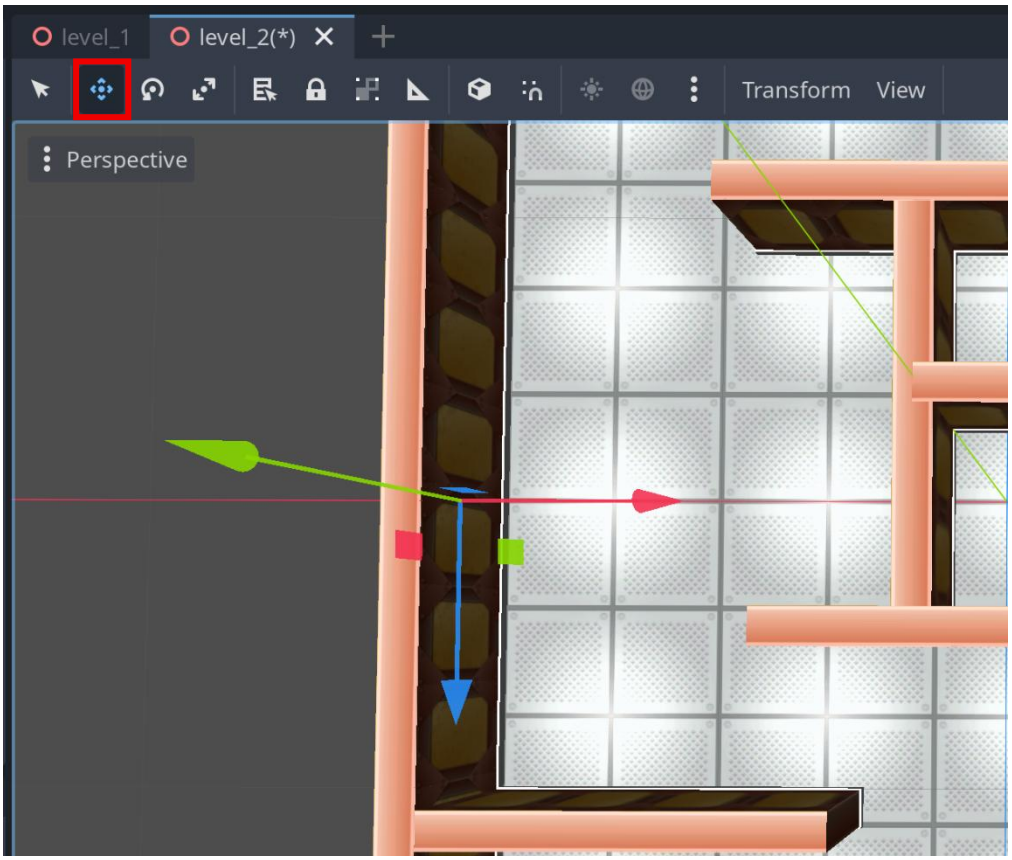
Level 2 isn't very different from level 1. Obstacles can be added to make the game more interesting!

43 In **FileSystem**, find the **wall_laser.tscn** scene inside the **Objects** folder in **Scenes**. Drag the scene onto the **Level2** root node to add a laser to the game.

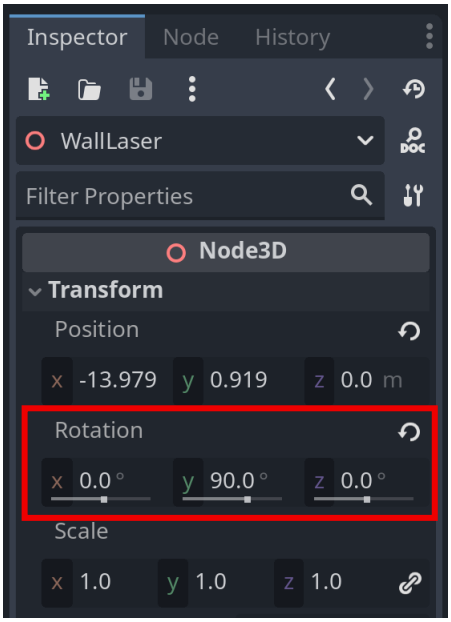




44 The laser will appear in the middle of the fortress. Use **Move Mode** to drag the **WallLaser** over to the **left wall**.

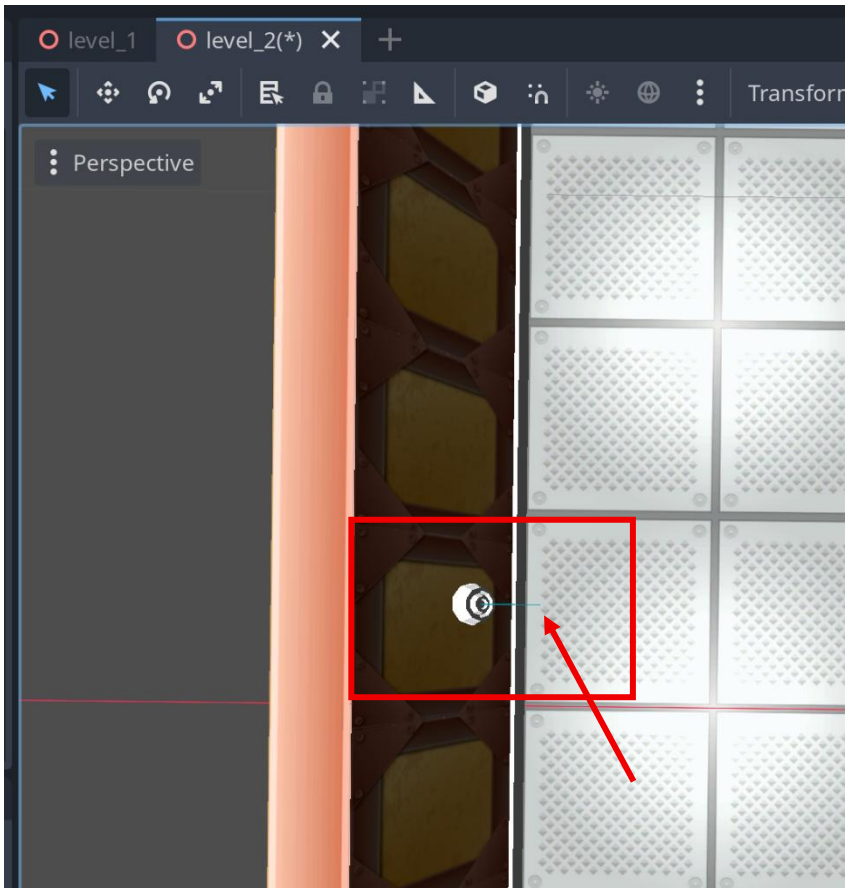


45 In the **Inspector** for **WallLaser**, set the **y rotation** to **90**.

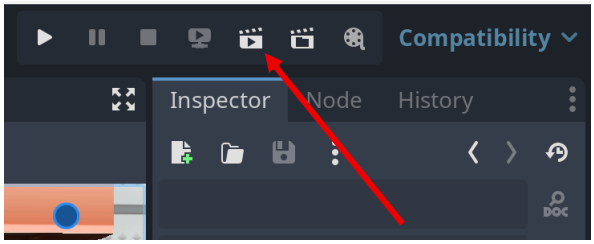


46 Further adjust the lasers position on the wall as needed until the laser is visible.

A thin blue line can be seen in the center of the laser. This shows the laser's direction.

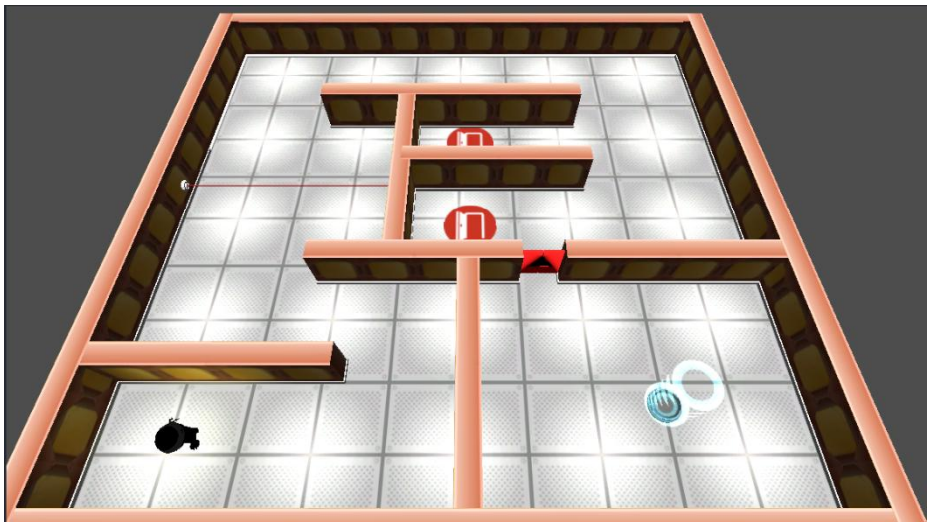


47 Use the **Run Current Scene** button to playtest the project.

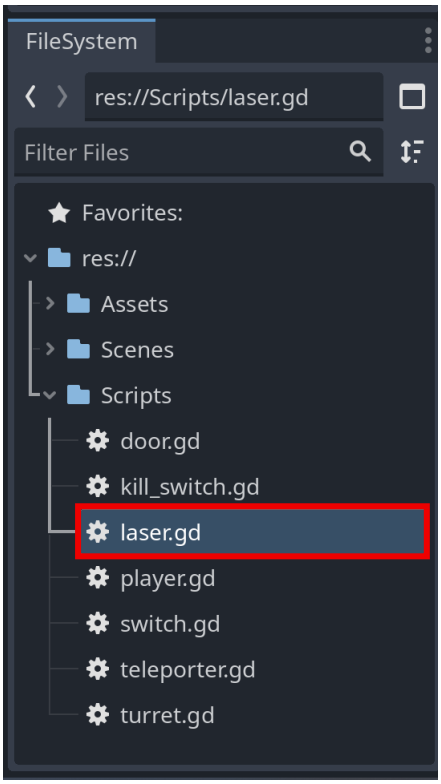


Playtesting with the Play button will load the project's main scene (Level 1).

48 When playtesting, the laser can be seen in the fortress. What happens if Codey tries to pass through the laser?



49 Codey can pass through the laser without consequence! In **FileSystem**, open the **laser.gd** script.



50 Notice that the **laser.gd** script contains variables and a fair bit of code.

The `raycast` variable, which accesses the laser's RayCast3D node, and the **active** variable, which is used to activate and deactivate the laser, will be used in the next bit of code.

```
1   extends Node3D
2
3   @export var beam_width: float = 0.05
4   @export var beam_range: float = 100.0
5
6   @onready var beam: MeshInstance3D = $BeamMesh
7   @onready var raycast: RayCast3D = $RayCast3D
8
9   var active: bool = true
10
```

51 Find **TODO 8** and create a variable `collider` to find which object the laser's RayCast3D node is colliding with.

```
28      # --------------------
29      # TODO  8
30      # Player death
31      # --------------------
32      var collider = raycast.get_collider()
33
```

Reminder:

The `get_collider()` method returns the first object the ray is intersecting with and null if no object is intersecting the ray.

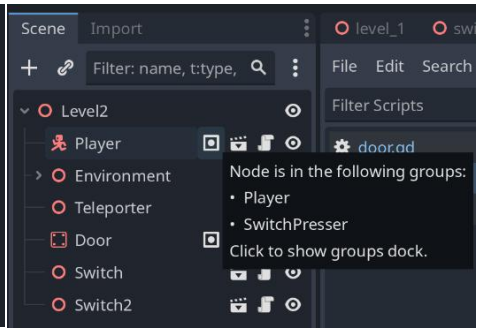
52 The laser might collide with different objects.

However, Codey's death animation should only run when the laser collides with the Player.

Write some code to call the `player_death()` function (which is in the script attached to the Player node) when the RayCast3D collides with Codey while the laser is active.

Think about what groups the Player node belongs to and how the `collider` and `active` variables might be used.

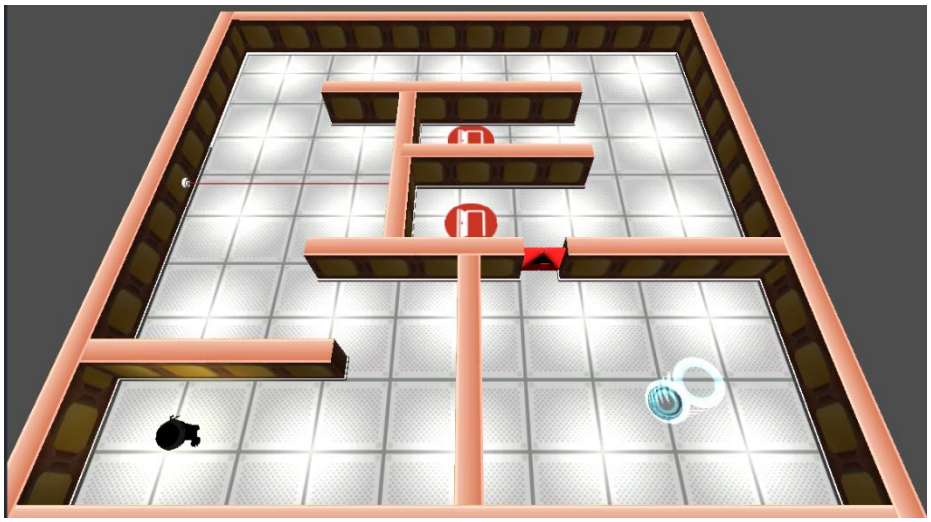
```
28    # --------------------
29    # TODO  8
30    # Player death
31    # --------------------
32    var collider = raycast.get_collider()
33    # if ??? and ???:
34        # if ???
35        # ??? player_death
```



53 Check the code and update the script as needed.

```
28    # --------------------
29    # TODO  8
30    # Player death
31    # --------------------
32    var collider = raycast.get_collider()
33    if collider and active:
34        if collider.is_in_group("Player"):
35            collider.player_death()
36
```

54 Playtest the project. What happens when Codey tries to pass through the laser?



Pause for **Sensei Stop #4**!

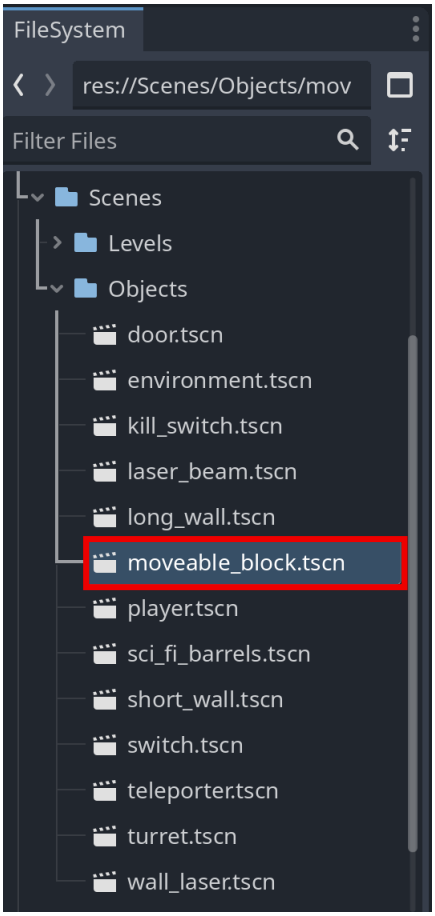
Check in with a Code Sensei before moving on. Make sure the code in the **laser.gd** script is correct.

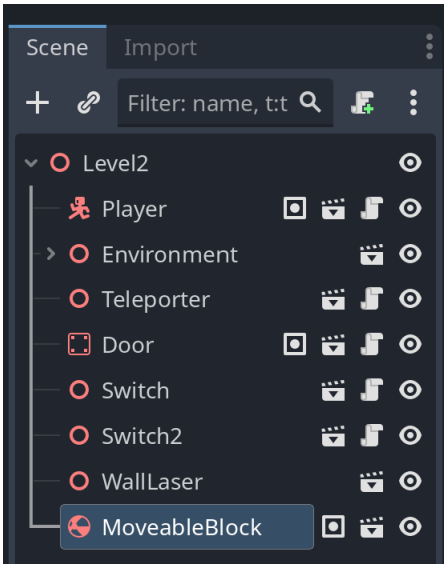
Reminder: Save your work!

55 When Codey tries to pass through the laser, the level resets. Something is needed to block the laser.

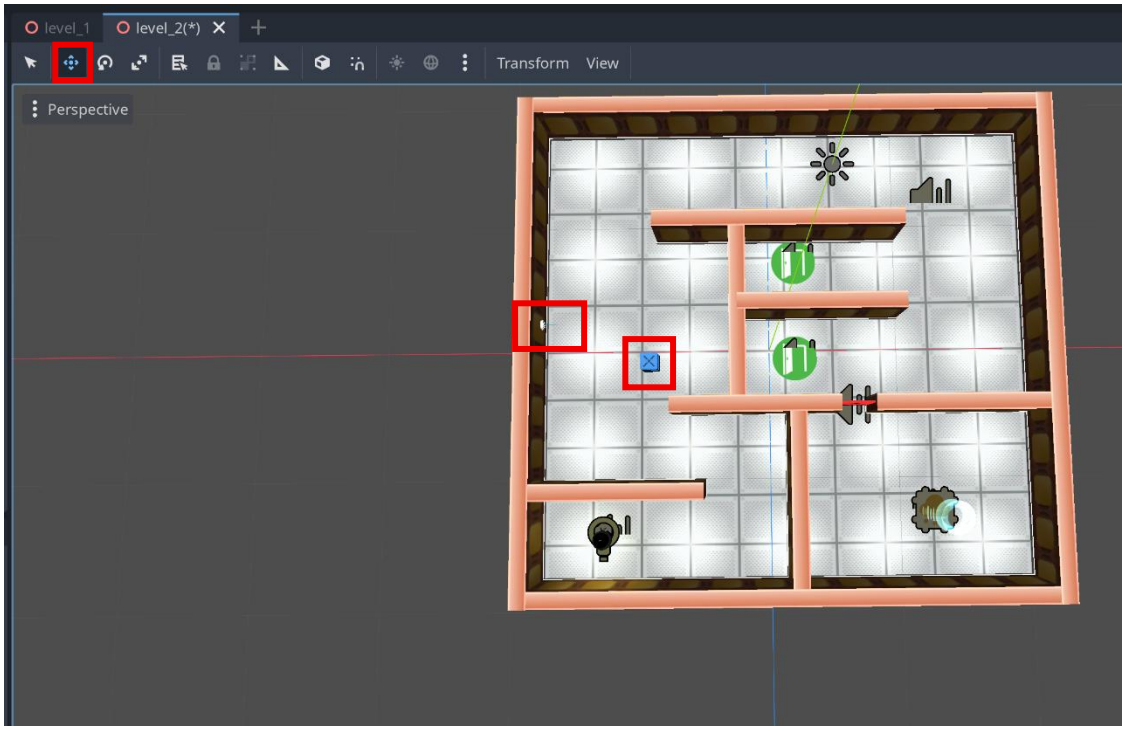
In **FileSystem**, find the **moveable_block.tscn** scene inside the **Objects** folder in **Scenes**.

Drag the scene onto the **Level2** root node to add a block to the game.

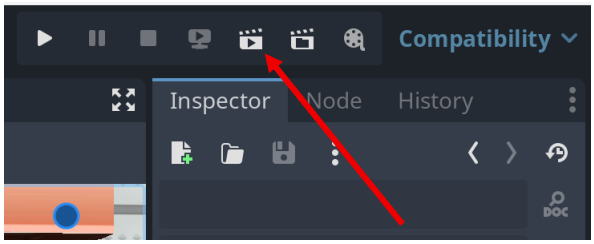




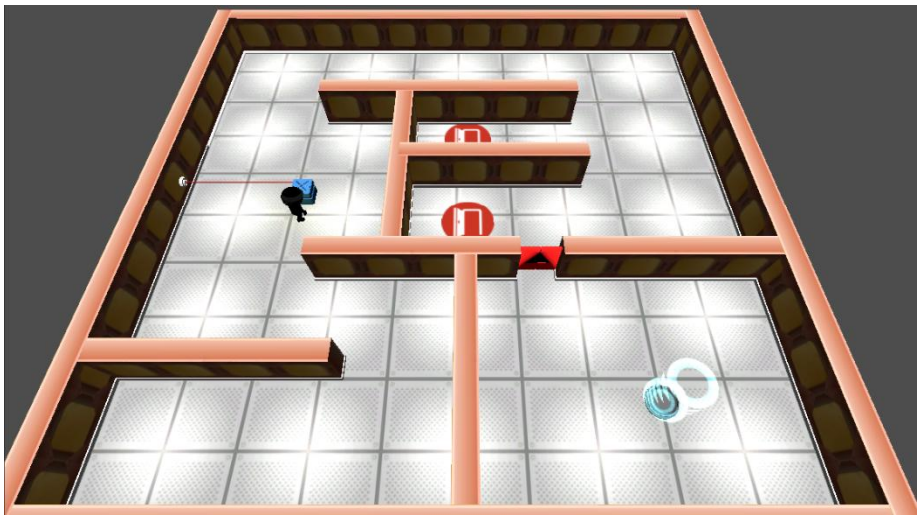
56 Return to the **3D** workspace. Use **Move Mode** to place the block before of the laser in the fortress.



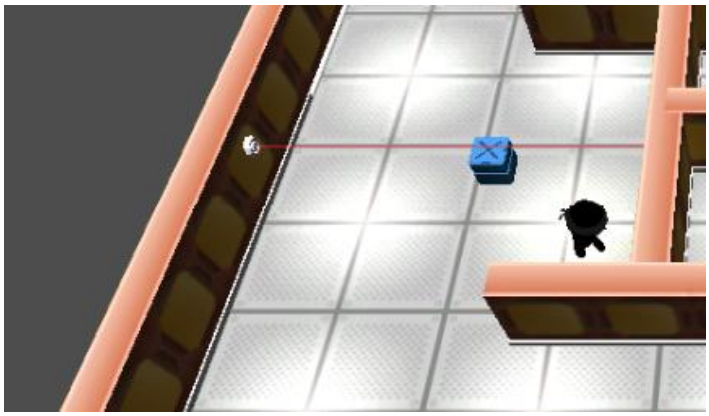
57 Playtest the project by running the current scene.



58 Codey can push the box in the fortress. What happens when the box intercepts the path of the laser?



If the laser can pass over the box, it is placed too high on the wall and needs to be adjusted.

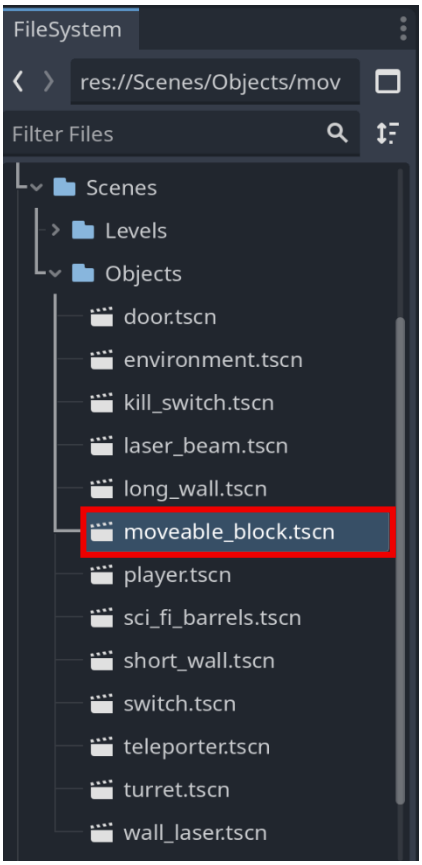


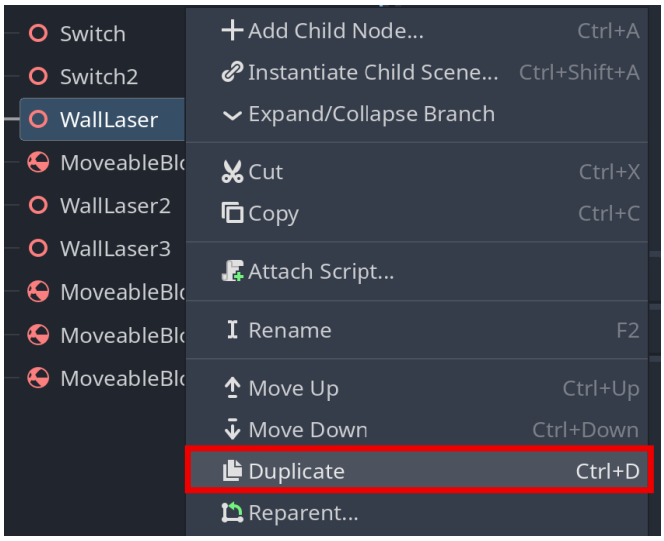
59 Add more **WallLasers** and **MoveableBoxes** to the fortress by **dragging** them in from the **FileSystem** or **duplicating** the nodes already in **Scene**.

Refer to steps 43 – 47 and 55 – 58 as needed.

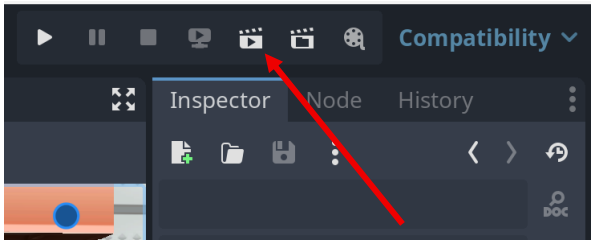
When adding additional obstacles, keep in mind...

- There should be one box per laser in the fortress.
- Boxes should be set so Codey can reach them before hitting the laser.
- Adjust the laser's rotation in the Inspector as needed.
- Check that the laser beams do not pass over the boxes.

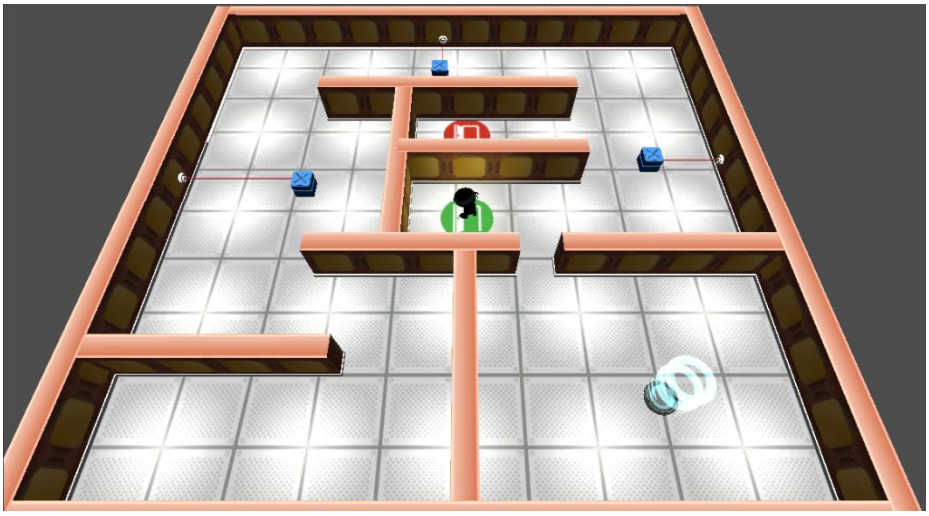




60 Playtest the project by running the current scene.

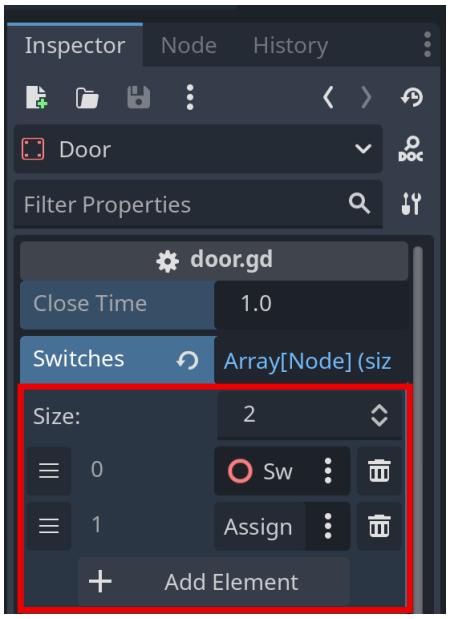


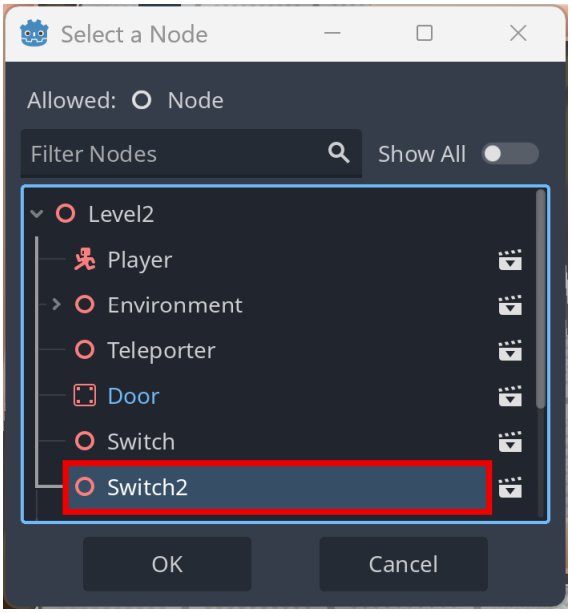
61 Can Codey reach the switches without touching the lasers? What happens when Codey overlaps the switches?



62 There are two switches in the fortress, but the door can be opened if Codey stands on one of the switches!

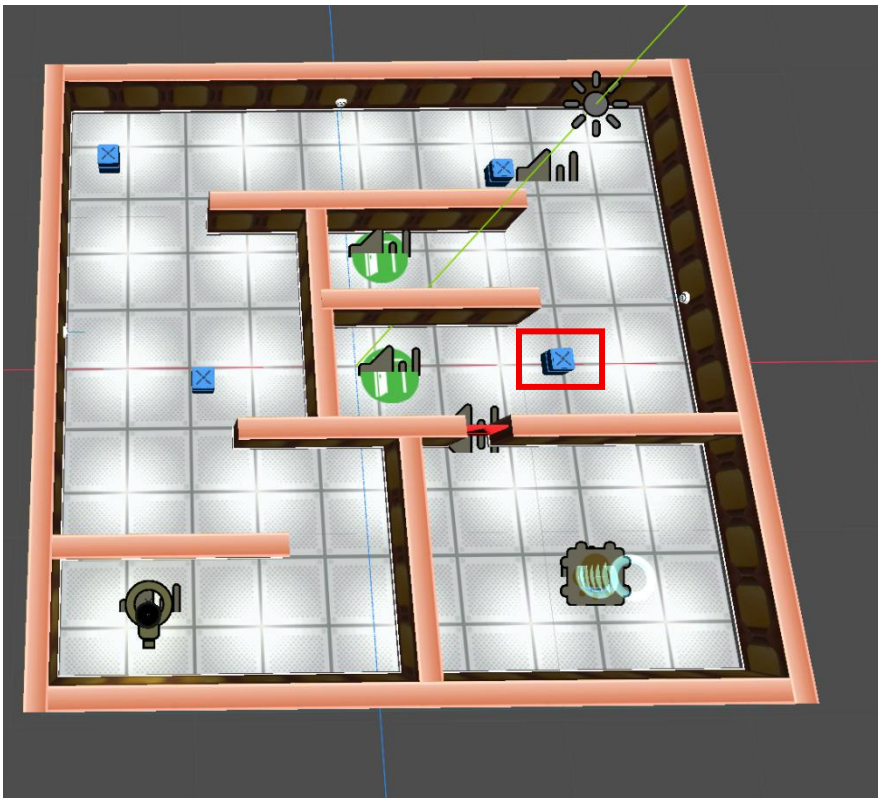
In the **Inspector** for **Door**, **add another element** to the array and set **element 1** to **Switch2**.



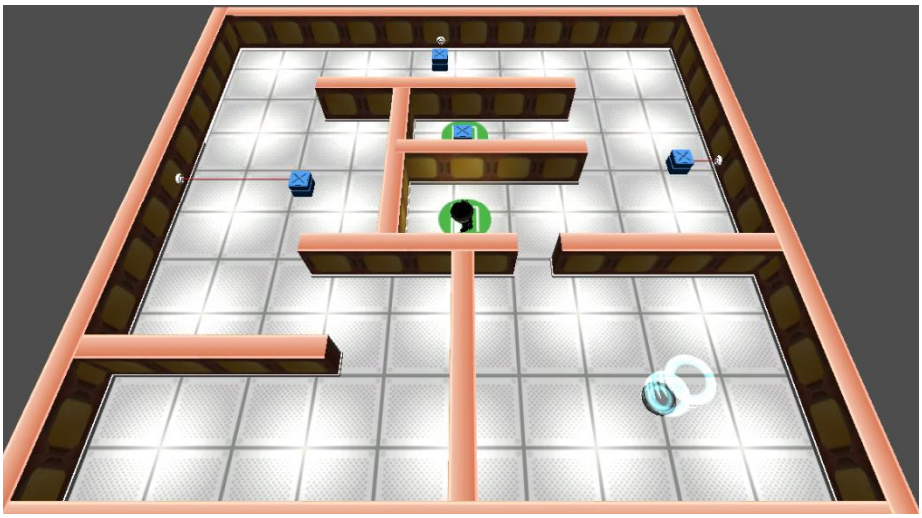


63 Codey needs something to help trigger the two switches at the same time.

Add another **MoveableBox** to the scene and position it near the switches.

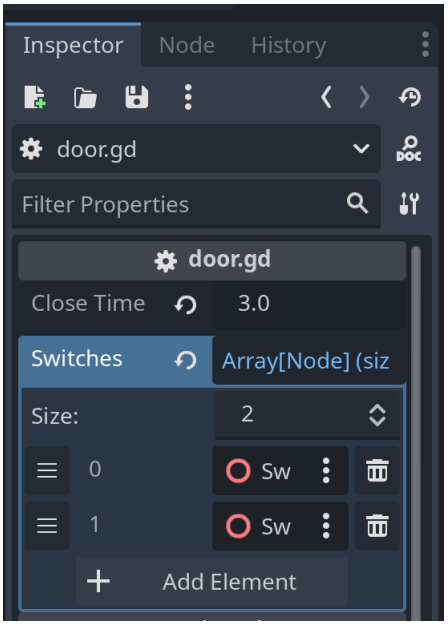


64 Playtest the project by running the current scene. What happens when both switches are triggered? Is there enough time for Codey to make it through the door before it closes?

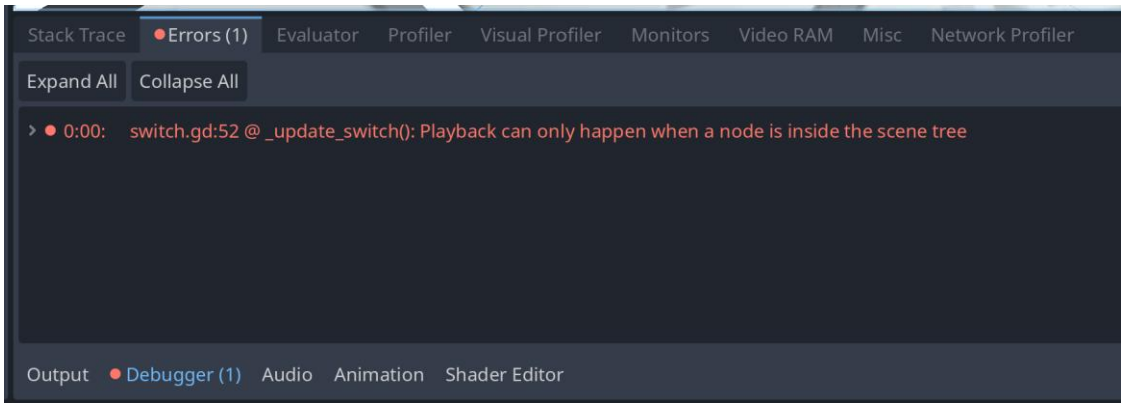


65 In the **Inspector** for **Door**, adjust the value of **Close Time** as needed so Codey has enough time to pass through the door.

Save the scene.



66 Levels requiring multiple switches to open a door may throw an error as seen below. These errors can be ignored and will not affect the game play.



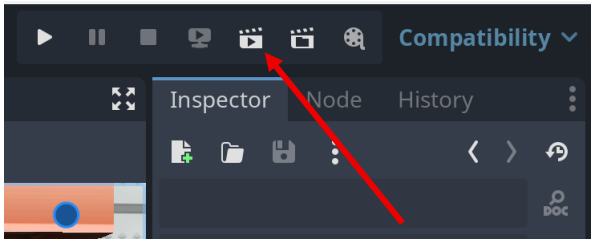
Pause for **Sensei Stop #5**!

Check in with a Code Sensei before moving on. Make sure the door is set to open with both switches, and enough obstacles have been added to the game!

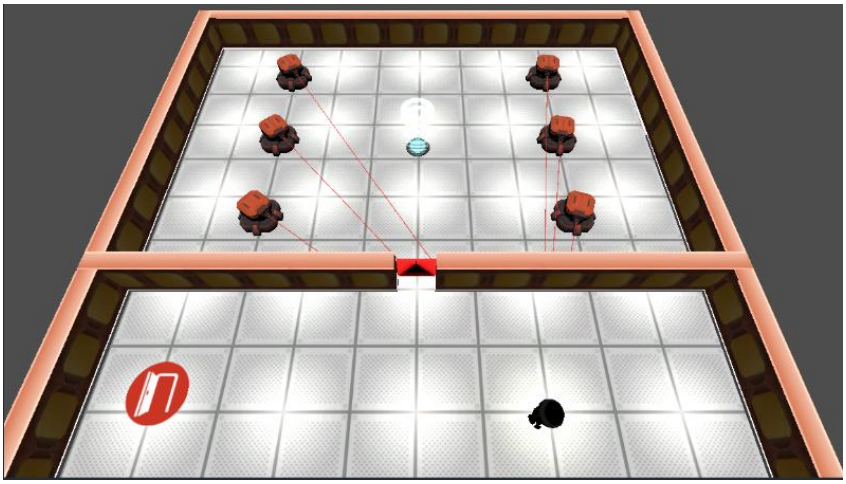
Reminder: Save your work!

67 In **FileSystem**, find **level_3.tscn** scene inside the **Levels** folder in **Scenes**. Double click the **level_3.tscn** scene to open it.

68 Playtest the project by running the current scene.

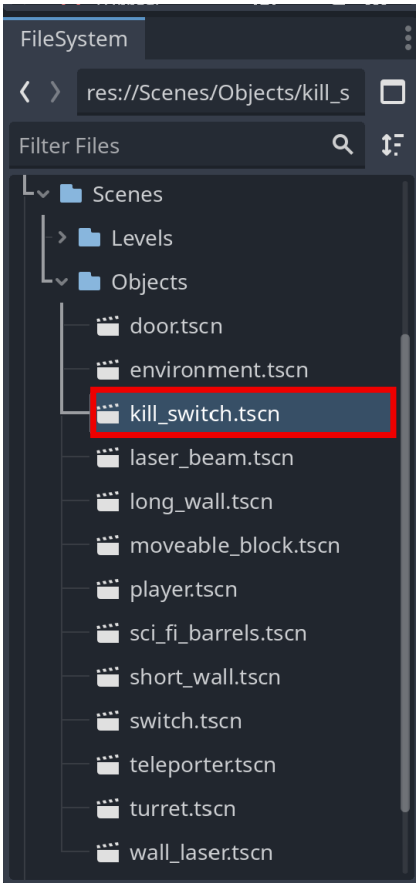


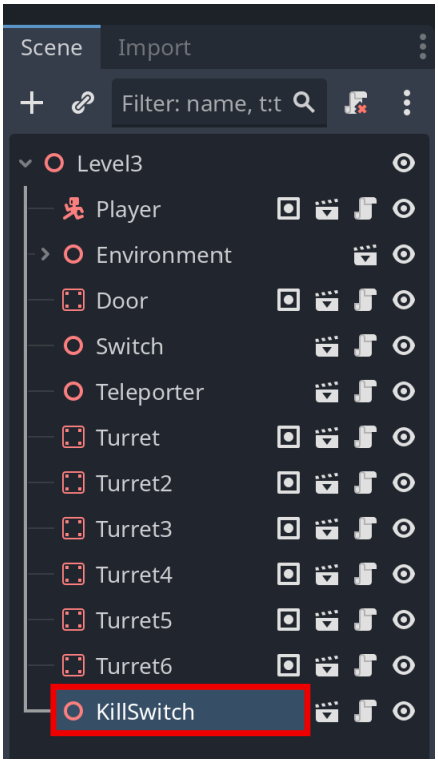
69 What do the turret lasers do when Codey moves? What happens when the door is opened, and Codey tries to escape?



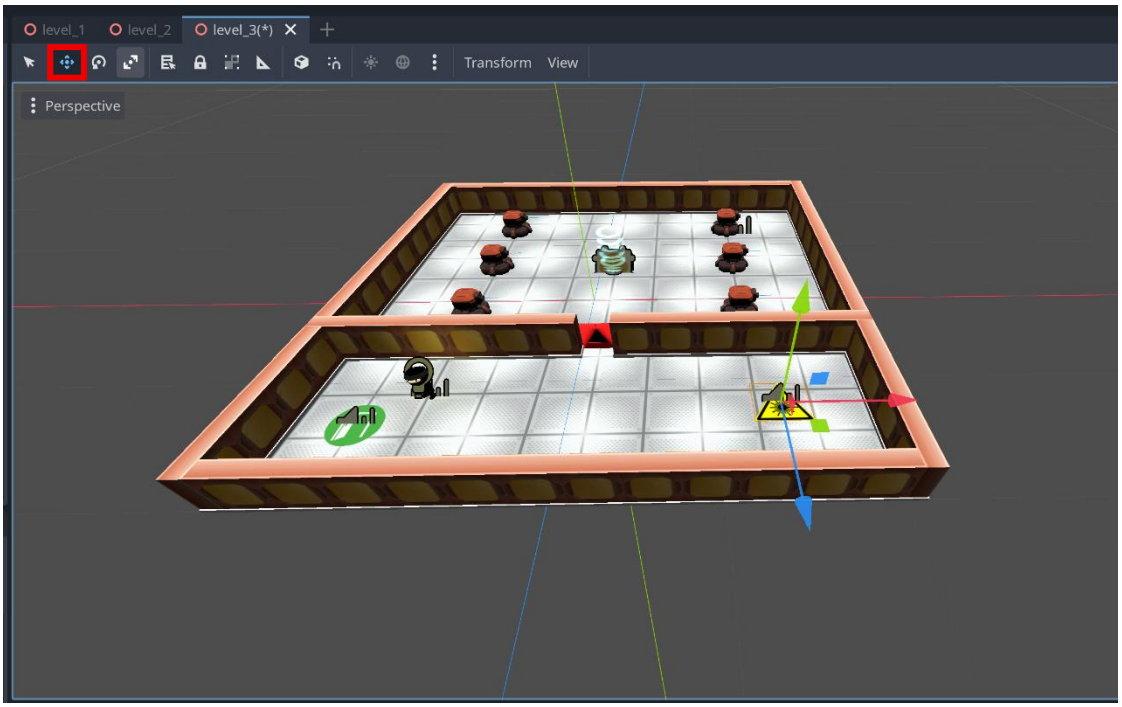
70 The turret lasers follow Codey around. A switch can be added to disable the lasers.

In **FileSystem**, find the **kill_switch.tscn** scene inside the **Objects** folder in **Scenes**. Drag the scene onto the **Level3** root node to add a kill switch to the game.





71 Using **Move Mode**, move the Kill Switch away from Codey.



72 The switch needs to be coded to turn off the turrets so Codey can escape through the door.

Open the **kill_switch.gd** script attached to the **KillSwitch** node.

Looking at the TODOs and the variables in the code, what can be observed?

The kill_switch.gd script will contain very similar code to the switch.gd script.

```
1   extends Node3D
2
3   @onready var pressed_sprite = $PressedSprite
4   @onready var unpressed_sprite = $UnpressedSprite
5   @onready var pressed_audio = $PressedAudioPlayer
6   @onready var unpressed_audio = $UnpressedAudioPlayer
7   @onready var area = $Area3D
8
9   # --------------------
10  # TODO  12
11  # find turrets in scene
12  # --------------------
13
14  var pressed: bool = false
15
16  # --------------------
17  # TODO  11
18  # _ready()
19  # --------------------
20
21  # --------------------
22  # TODO  9
23  # _update_sprite(), variables & touching_switch_presser
24  # --------------------
25
```

73 Underneath **TODO 9**, create the `_update_sprite()` function, which takes a single parameter, `_body`.

Inside the function, use the `get_overlapping_bodies()` method to get a list of `bodies`, and create the `touching_switch_presser` variable and set it to `false`.

```
20
21  # --------------------
22  # TODO  9
23  # _update_sprite(), variables & touching_switch_presser
24  # --------------------
25  # _update_sprite()
26      # var bodies
27      # var touching_switch_presser
28
```

74 Underneath the variables, create a `for-loop` to check `if` any of the nodes in the array of `bodies` are part of the `SwitchPresser group`.

If so, update the `touching_switch_presser` and `break` the loop.

Refer to the code in the **switch.gd** script as needed.

```
21  # --------------------
22  # TODO  9
23  # _update_sprite(), variables & touching_switch_presser
24  # --------------------
25  # _update_sprite()
26      # var bodies
27      # var touching_switch_presser
28
29      # for ???:
30          # if ???:
31              # touching_switch_presser ???
32              # break
33
```

75 Find **TODO 10** and write the `if` statements using the `not` operator to check:

- if touching a SwitchPresser and the switch is not pressed, press the switch and play the pressed sound.
- if not touching a SwitchPresser and the switch is pressed, unpress the switch and play the unpressed sound.

Underneath the `if` statements, update the sprite image to reflect if the kill switch is being pressed or not.

```
33
34      # --------------------
35      # TODO  10
36      # toggle switch & update sprite
37      # --------------------
38      # if ??? and ???:
39          # pressed ???
40          # audio ???
41          # --------------------
42          # TODO  13
43          # deactivate turrets
44          # --------------------
45      # elif ??? and ???:
46          # pressed ???
47          # audio ???
48          # --------------------
49          # TODO  14
50          # activate turrets
51          # --------------------
52
53      # pressed_sprite ???
54      # unpressed_sprite ???
55
```

Refer to the code in switch.gd script as needed.

76 Near the top of the script, find **TODO 11**.

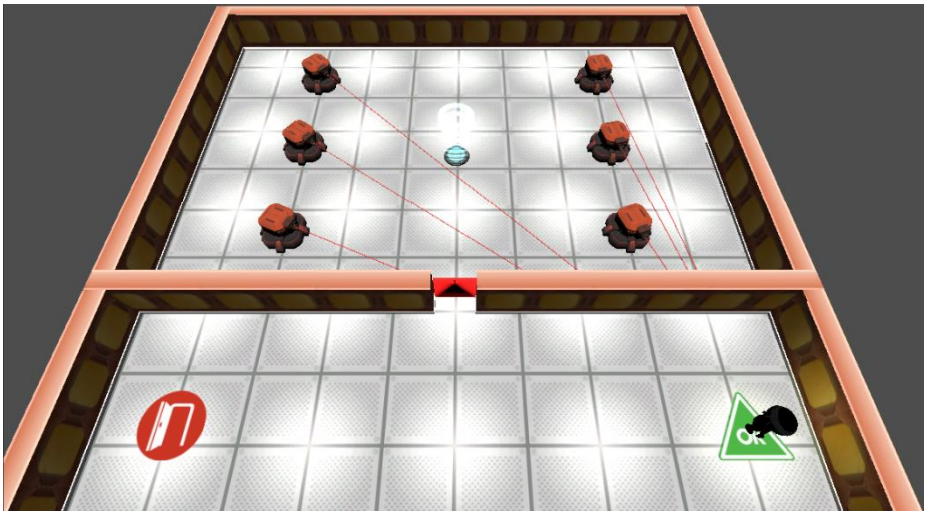
Create the `_ready()` method and `connect()` the Area3D nodes `body_entered` and `body_exited` signals to the `_update_sprite()` function.

```
16  # --------------------
17  # TODO  11
18  # _ready()
19  # --------------------
20  # _ready()
21      # body_entered
22      # body_exited
23
```

Pro Tip:

Check the variables at the top of the script to see how the Area3D node is referenced in the code.

77 Playtest the project. Does the kill switch turn on and off when Codey overlaps it? Do the turrets turn on and off?



78 Check the code and update the script as needed.

```
15
16  # --------------------
17  # TODO  11
18  # _ready()
19  # --------------------
20  func _ready() -> void:
21      area.body_entered.connect(_update_sprite)
22      area.body_exited.connect(_update_sprite)
23
24  # --------------------
25  # TODO  9
26  # _update_sprite(), variables & touching_switch_presser
27  # --------------------
28  func _update_sprite(_body):
29      var bodies = area.get_overlapping_bodies()
30      var touching_switch_presser = false
31
32      for body in bodies:
33          if body.is_in_group("SwitchPresser"):
34              touching_switch_presser = true
35              break
36
37      # --------------------
38      # TODO  10
39      # toggle switch & update sprite
40      # --------------------
41      if touching_switch_presser and not pressed:
42          pressed = true
43          pressed_audio.play()
44          # --------------------
45          # TODO  13
46          # deactivate turrets
47          # --------------------
48      elif not touching_switch_presser and pressed:
49          pressed = false
50          unpressed_audio.play()
51          # --------------------
52          # TODO  14
53          # activate turrets
54          # --------------------
55
56      pressed_sprite.visible = pressed
57      unpressed_sprite.visible = not pressed
58
```

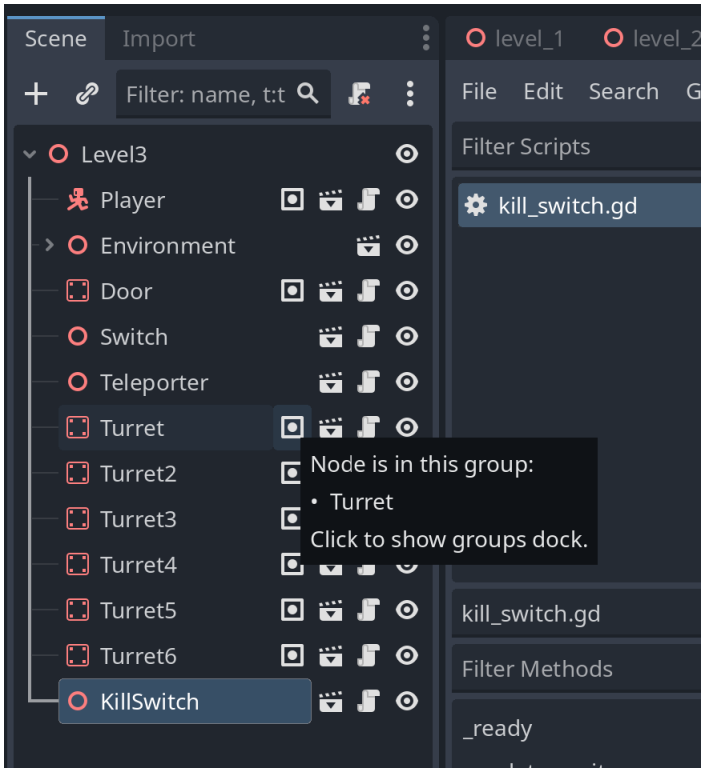
Pause for **Sensei Stop #6**!

Check in with a Code Sensei before moving on. Make sure the code in the **kill_switch.gd** script is correct

Reminder: Save your work!

79 In the **switch.gd** script, the switch uses signals to open and close the door, and the switch is assigned to a door in the Inspector. This allows for a level design where switch A opens door A and switch B opens door B.

In the **kill_switch.gd** script, the turrets will all be turned on and off with the same switch. Looking at what groups the turret nodes are part of; how might this be done?



80 The **kill_switch.gd** script will use groups to find all the turrets and activate or deactivate them one by one.

Underneath **TODO 12**, create an `@onready` variable `turrets`.

Use the `get_tree()` method to get the scene tree of the node the script is attached to.

The `get_nodes_in_group()` method can be used to find all the nodes within a group inside of the scene tree.

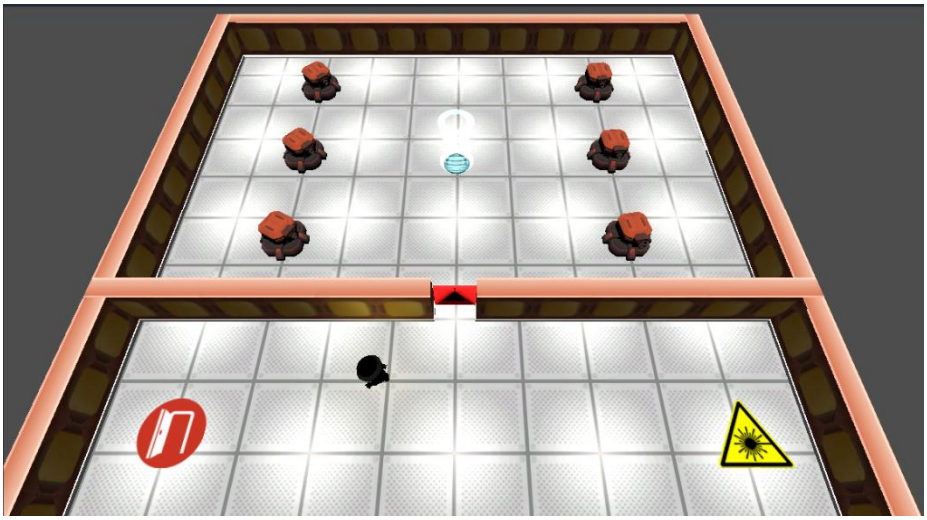
The variable turrets is now an **array** of nodes in the group Turret.

```
 9  # --------------------
10  # TODO  12
11  # find turrets in scene
12  # --------------------
13  @onready var turrets = get_tree().get_nodes_in_group("Turret")
14
15  var pressed: bool = false
```

81 Scroll down to **TODO 13** and create a for-loop inside the if-statement to deactivate each turret in the array of turrets using the `deactivate()` function.

```
38      # --------------------
39      # TODO  10
40      # toggle switch & update sprite
41      # --------------------
42      if touching_switch_presser and not pressed:
43          pressed = true
44          pressed_audio.play()
45          # --------------------
46          # TODO  13
47          # deactivate turrets
48          # --------------------
49          # for ??? in ???:
50          # ??? deactivate()
51      elif not touching_switch_presser and pressed:
```

82 Playtest the project. What happens to the turrets when Codey overlaps the kill switch? What happens to the turrets when Codey leaves the kill switch?

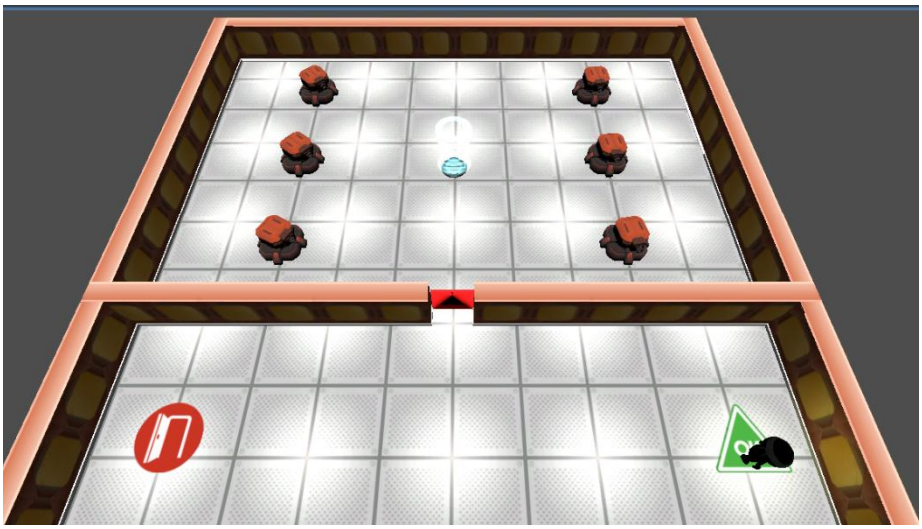


83 Return to the **kill_switch.gd** script and scroll down to **TODO 14**

Create a `for-loop` inside the `elif` statements to activate each turret in the array of turrets using the `activate()` function.

```
50          # ??? deactivate()
51      elif not touching_switch_presser and pressed:
52          pressed = false
53          unpressed_audio.play()
54          # --------------------
55          # TODO  14
56          # activate turrets
57          # --------------------
58          # for ??? in ???:
59          # ??? activate()
60
```

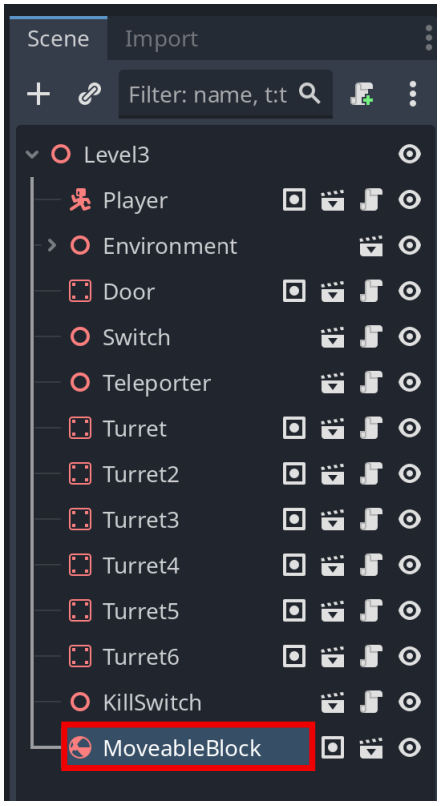
84 Playtest the project. Do the turrets turn on and off?



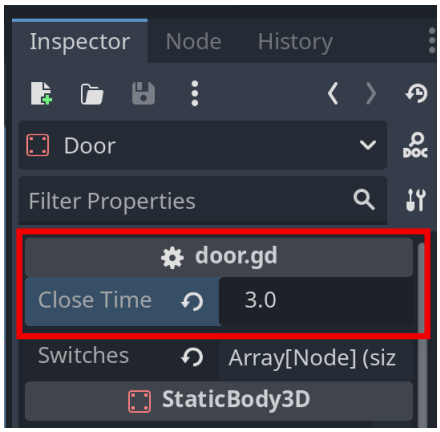
85 Check the code and update the script as needed.

```
38    # --------------------
39    # TODO  10
40    # toggle switch & update sprite
41    # --------------------
42    if touching_switch_presser and not pressed:
43        pressed = true
44        pressed_audio.play()
45        # --------------------
46        # TODO  13
47        # deactivate turrets
48        # --------------------
49        for turret in turrets:
50            turret.deactivate()
51    elif not touching_switch_presser and pressed:
52        pressed = false
53        unpressed_audio.play()
54        # --------------------
55        # TODO  14
56        # activate turrets
57        # --------------------
58        for turret in turrets:
59            turret.activate()
60
```

86 Add a **MoveableBlock** to the level so Codey can trigger both switches to escape.



In the **Inspector** for **Door**, adjust the value of **Close Time** if needed.



Pause for **Sensei Stop #7**!

Check in with a Code Sensei before moving on and reflect on the following:

- Why might signals be connected through code instead of the interface?
- How can groups and signals work together when creating trigger-specific events?

How was the door and switch set up with level design in mind?

Reminder: Save your work!