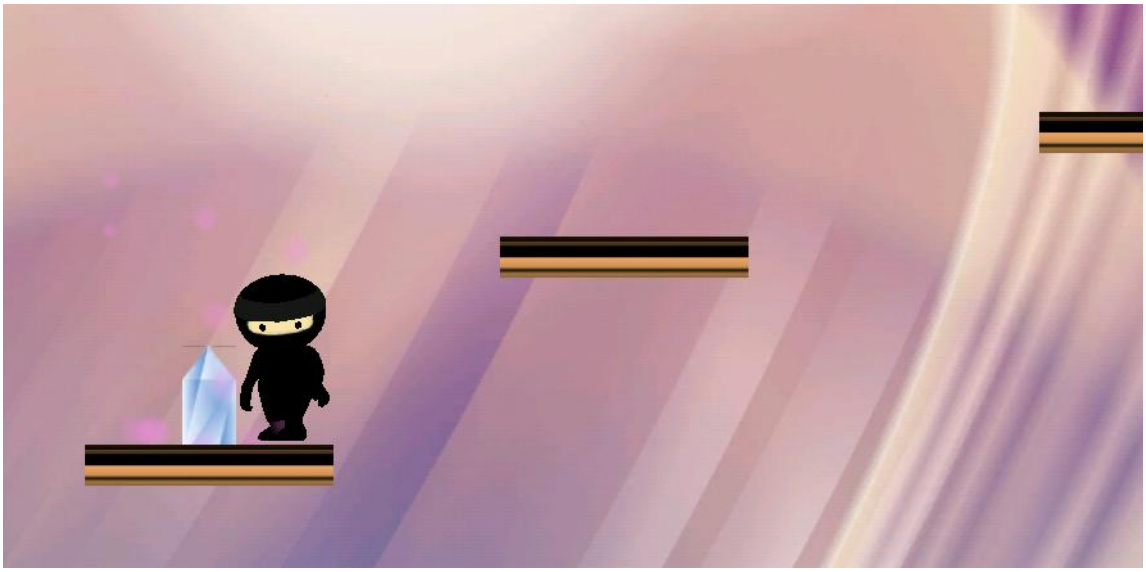
CODE NINJAS®

Silver Belt Ninja Guide

Activity 09: Amazing Ninja Worlds Part 1

Checkpoints, Objectives, and Loose Coupling



Have you ever worked for hours on something – whether it be building a project with code, writing an essay, or creating a physical structure – only for your work to be deleted or fall apart? When the fruits of our efforts get deleted, we all wish we could save our progress along the way. In games, checkpoints serve to "save" a player's progress. When a checkpoint is reached, the game saves the location of the checkpoint as the place to respawn the player, should they fail to proceed to the next checkpoint.

Games are often designed with an objective to complete before progressing to the next portion of the game. Often, this goal is to collect all of a given object within the area before proceeding to the exit. To force players to engage with this mechanic so that every object is collected, progress is frequently blocked for players who have not picked up all the collectibles. This requires the game to keep track of the total number of objects, as well as how many objects remain in the scene.

But where does the player go next? Think of games that allow the player to traverse between regions and levels. How do games handle the logic that loads different stages, and how is each traversal consistent?

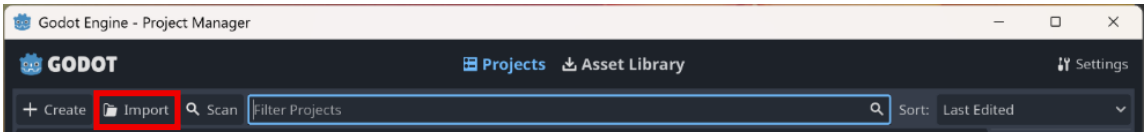
```
15  func _on_body_entered(body: Node3D) -> void:
16      if body.is_in_group("Player"):
17          LevelManager.keys_left -= 1
18          queue_free()
```

In these cases, the practice of **loose coupling** allows for a more generalized approach to programming. This method uses signals and groups instead of references to specific paths to allow for more flexible code. With **tight coupling**, which uses these more hard-coded specific paths, any changes in the Scene's node hierarchy can break the game. With loose coupling, individual nodes like the teleporters, checkpoints, and keys can be moved around and reused freely without worrying about resetting the path each time.

Activity 09: Amazing Ninja Worlds Pt 1

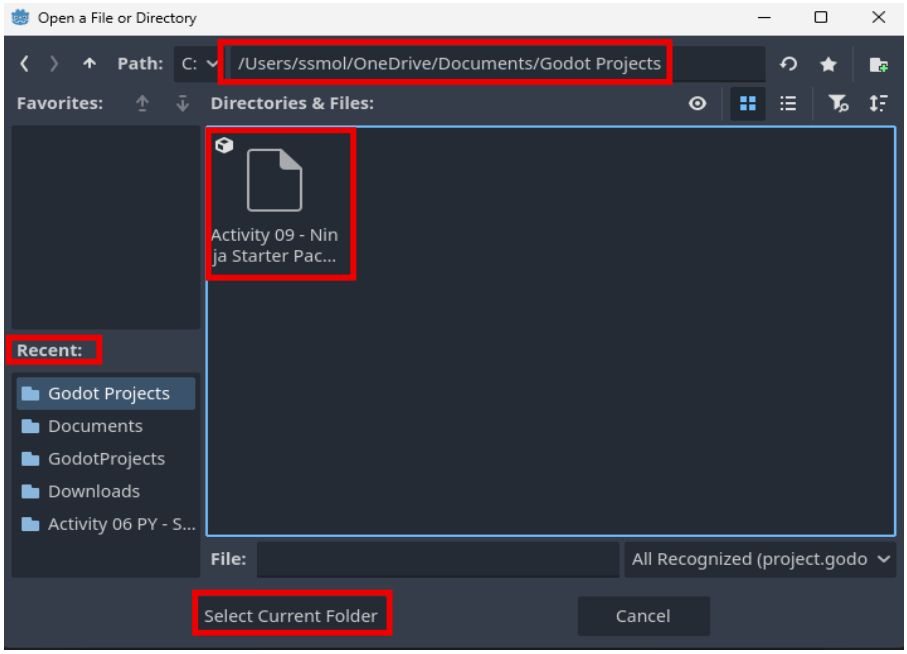
In this project, you will create checkpoints, collectible keys, and a teleporter in a platforming game. Additionally, you will script these objects to appropriately respawn the player to the correct location, and disappear keys that are collected, and teleport the player once all keys are collected.

1 Open Godot and click **Import.**



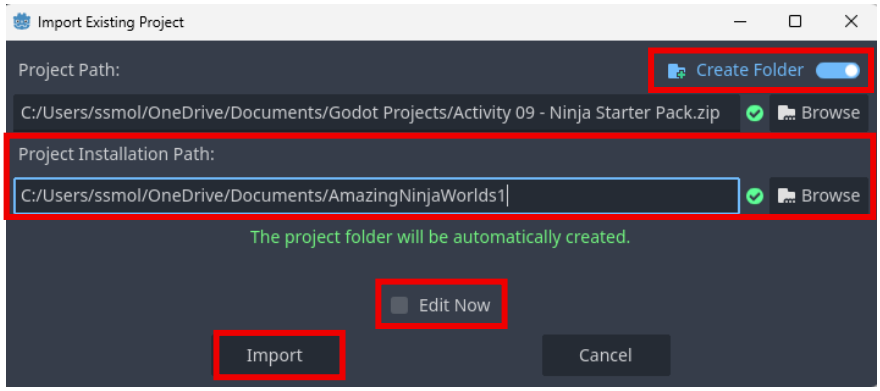
2 In the File Directory, navigate to the correct file path.

Select **Activity 09 – Ninja Starter Pack.zip** and click **Open**.



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

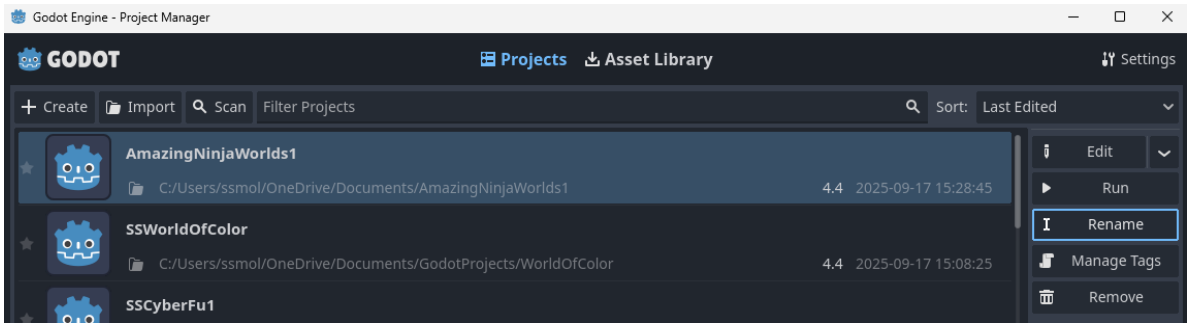
Uncheck Edit Now and click **Import**.



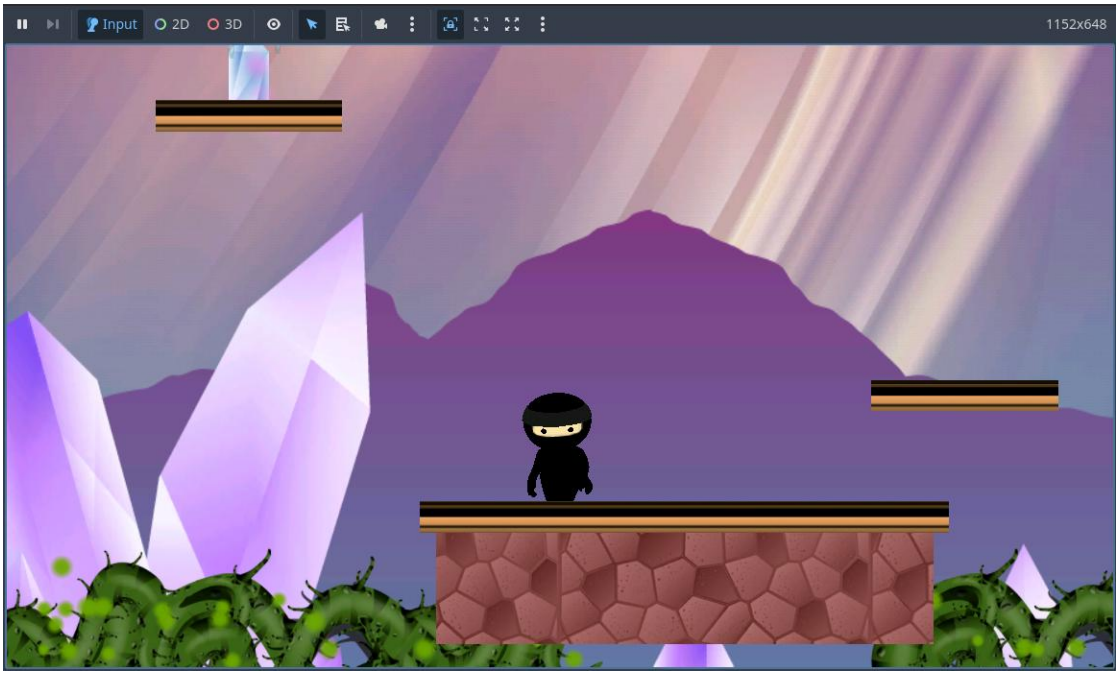
4 The project will appear at the top. Click on the project and select **Rename** on the right.

Update the Project Name to **[YourInitials]AmazingNinjaWorlds1**.

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

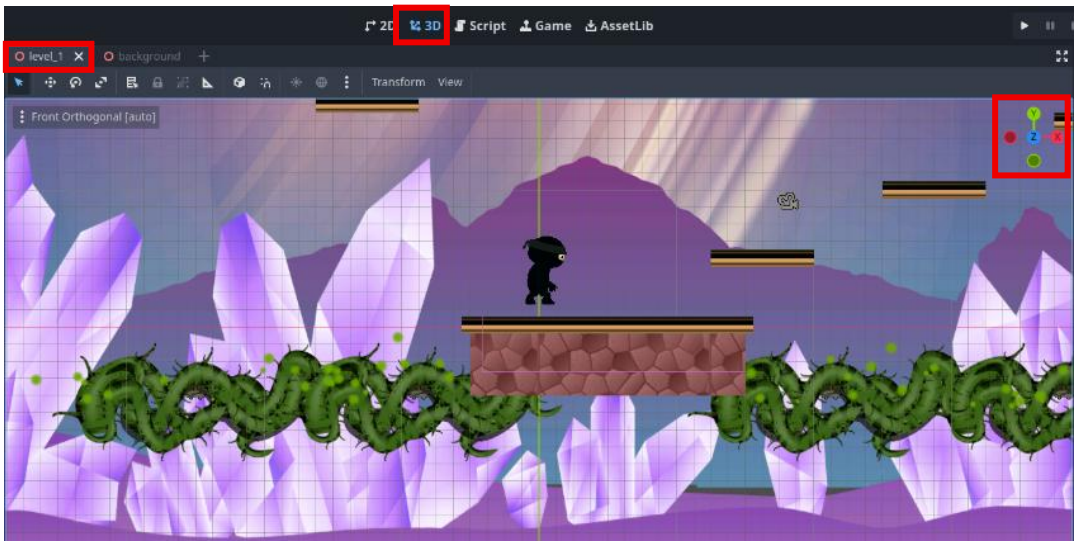


5 Playtest the game. The crystals should now show up in the near background, and the Ninja should be able to move and jump around.

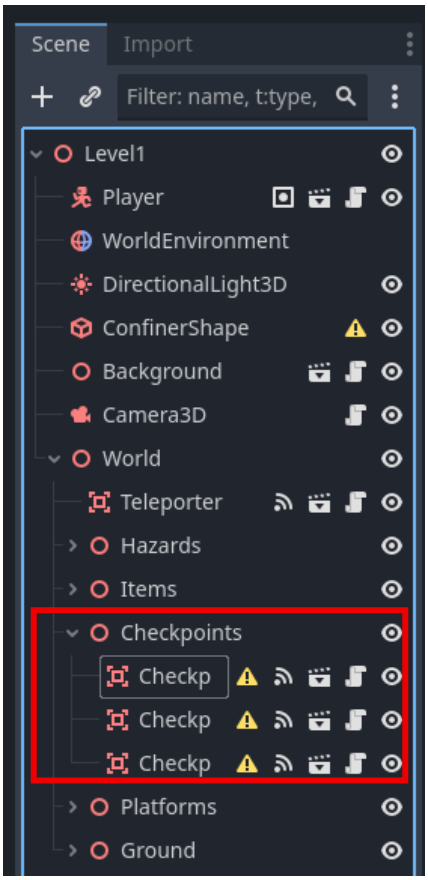


6 Implement checkpoints.

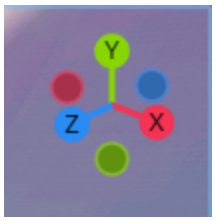
Near the top of the screen, select the **level_1** tab to open the **level_1 scene**.



In **Scene**, notice that there are 3 checkpoint nodes as children to World.



Select the **3D Workspace** and click the **blue Z** node on the viewpoint gizmo in the top right corner to better view the scene by setting the camera to look down the Z axis.



Notice the 3 checkpoints in the scene already; they are the crystals sticking out of the platforms.

Pro Tip:

Click 1 on the keyboard to set the camera to look down the Z axis.

7 In **FileSystem**, open the **level_manager.gd** script.

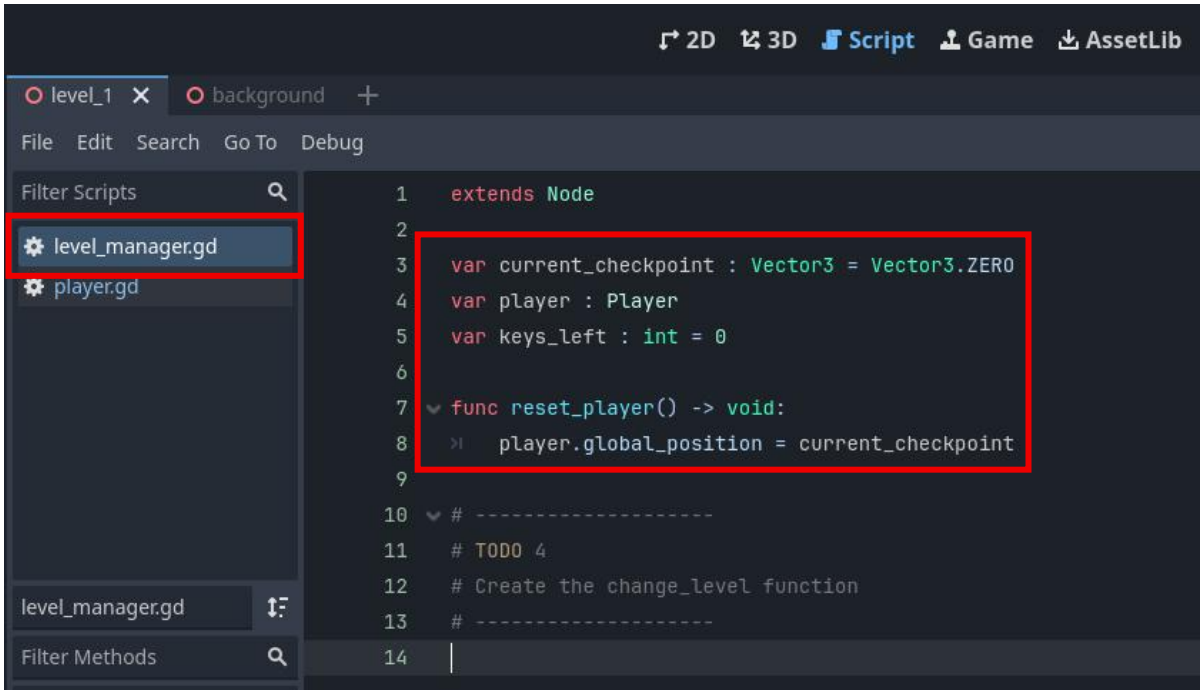
In the Script Workspace, notice that there are 3 variables: **`current_checkpoint`**, **`player`**, and **`keys_left`**.

The **`current_checkpoint`** variable is of type **`Vector3`**, which represents a point in space, (x,y,z). It is currently set to **`Vector3.ZERO`**, which represents the point (0,0,0), also known as the "origin." This variable will store the position of the current checkpoint the player has achieved.

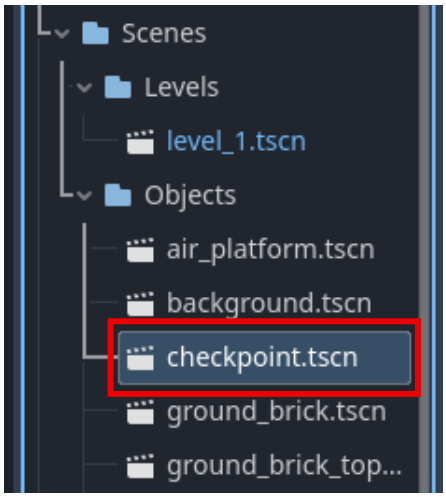
The **`player`** variable stores a reference to the **Player** node.

The **`keys_left`** variable is an integer initialized to **`0`** that keeps track of how many keys remain in the scene. **`keys_left`** will increase each time a key node is loaded into the scene.

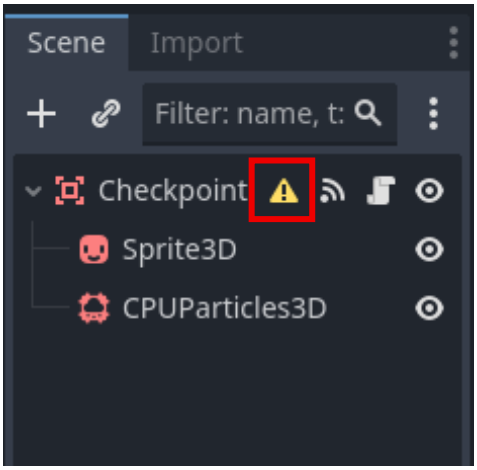
Additionally, notice the function **`reset_player()`**, which sets the **player**'s global position to the position saved in **`current_checkpoint`**.

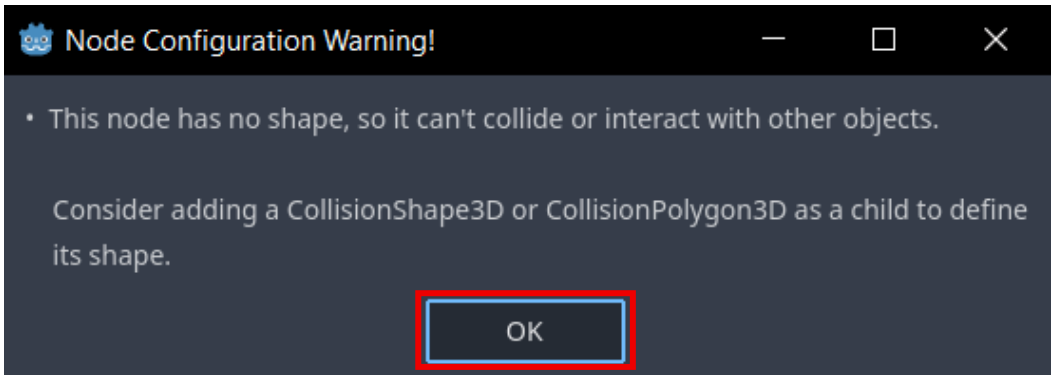


8 In **FileSystem**, navigate to **Scenes > Objects > checkpoint.tscn** and double click to open it.



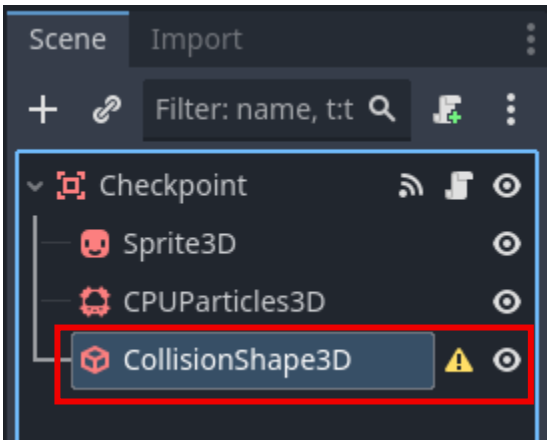
Notice that the main root node has a warning icon next to it. Click on the icon to read the warning. Click OK to exit.



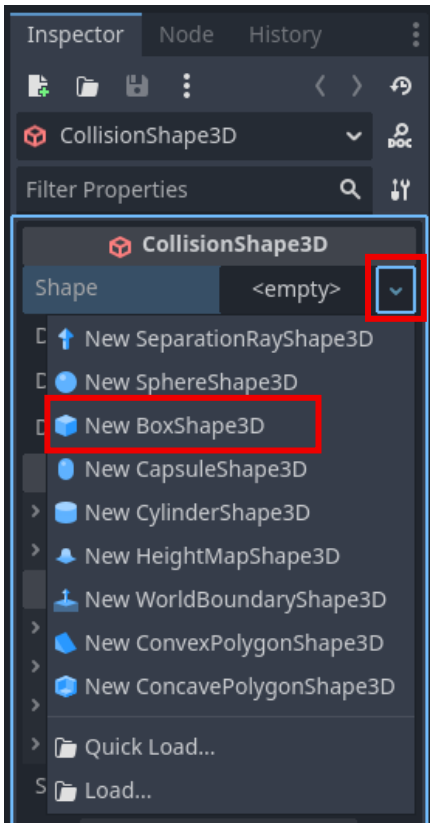


9 Add a collision shape to make something happen when the player collides with a checkpoint!

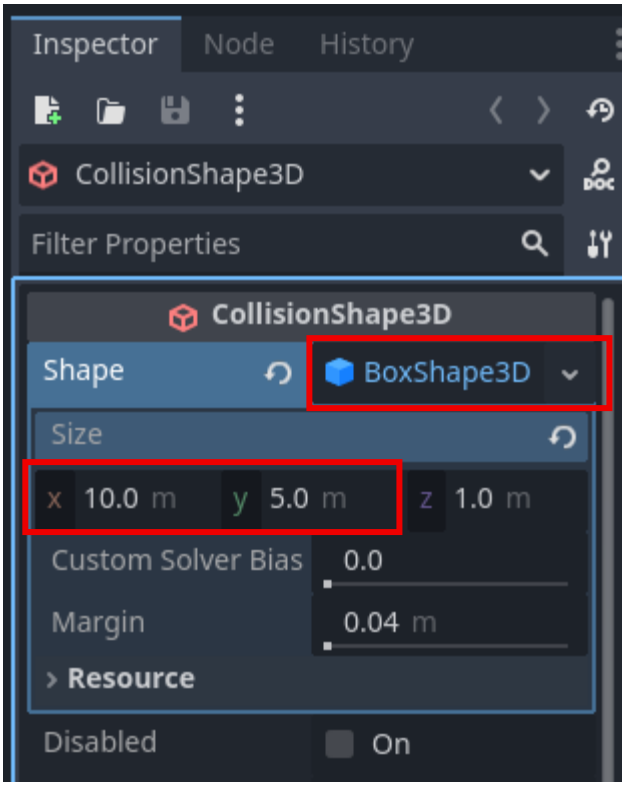
Add a **CollisionShape3D** node as a child to the main root.



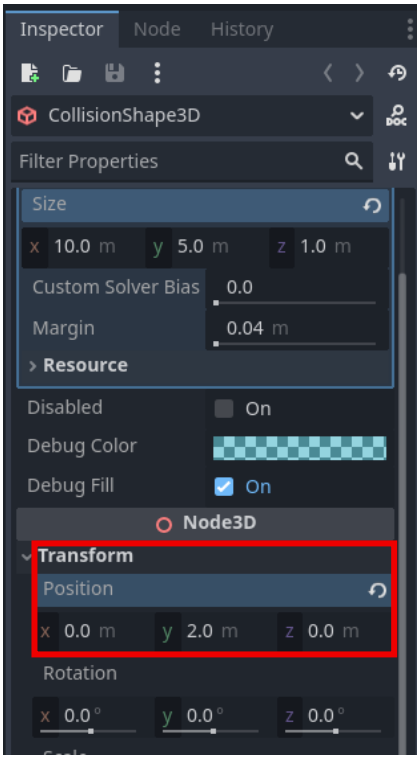
10 In **Inspector**, click the arrow to the right of **Shape <empty>.** Select **New BoxShape3D** from the drop-down menu.



11 Click **BoxShape3D** to open its submenu. Set **Size x** to **10** and **size y** to **5**.



12 In **Inspector**, under **Transform**, set **position y** to **2**.



13

In **FileSystem**, open the **checkpoint.gd** script.

Under **TODO 1**, define an `_on_body_entered` function. It should have one parameter called `body` of type `Node3D` and should return `void`.

This function runs when the body of the checkpoint overlaps with another object.

```
3  # --------------------
4  # TODO 1
5  # Create and link the _on_body_entered method
6  # --------------------
7  func _on_body_entered(body: Node3D) -> void:
8      
```

14

Within the `_on_body_entered` function, write an `if` statement that checks if `body.is_in_group("Player")` is true.

This condition checks to see if the other body that has come into contact with the checkpoint is the Player.

```
3  # --------------------
4  # TODO 1
5  # Create and link the _on_body_entered method
6  # --------------------
7  func _on_body_entered(body: Node3D) -> void:
8      if body.is_in_group("Player"):
9          
```

15 Within the **if** statement, set `LevelManager.current_checkpoint` to `global_position`.

This sets the global `current_checkpoint` variable to the location of the checkpoint that just came into contact with the player.

```
3  # --------------------
4  # TODO 1
5  # Create and link the _on_body_entered method
6  # --------------------
7  func _on_body_entered(body: Node3D) -> void:
8      if body.is_in_group("Player"):
9          LevelManager.current_checkpoint = global_position
10
```

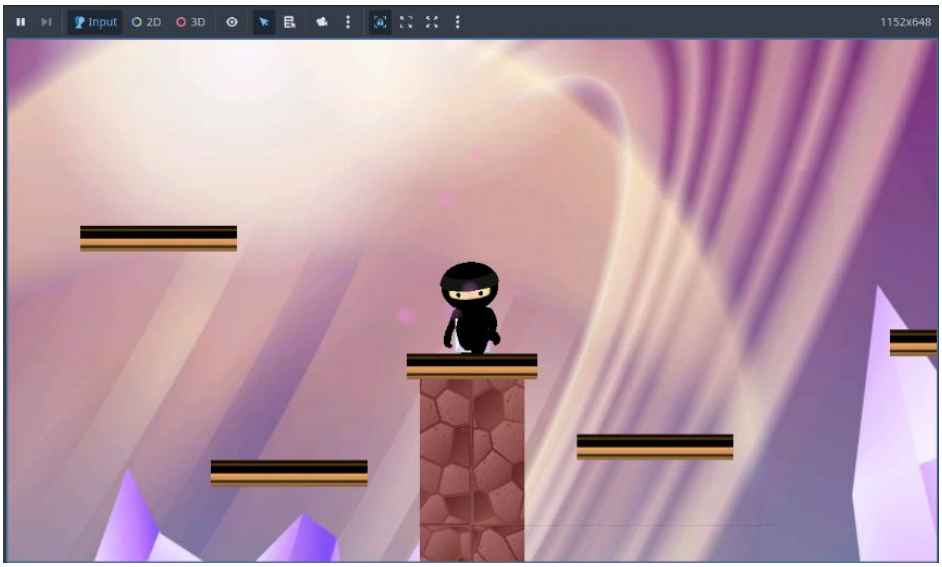
Pro Tip:

Since the global position of the checkpoint is being used to set `current_checkpoint`, it is important that when the checkpoint is placed in the scene, its center is above the platforms. This prevents the Player from being teleported inside the platforms when it gets set back to the current checkpoint.

16 Playtest the game.

Run into a checkpoint and then jump into the vines to see if the Player gets teleported back to the first checkpoint.

Run to the next checkpoint and check that the Player teleports back to the second checkpoint upon death.



Pause for **Sensei Stop #1**!

Check in with a Code sensei before moving on. Make sure the **checkpoints** are set up properly.

Reminder: Save your work!

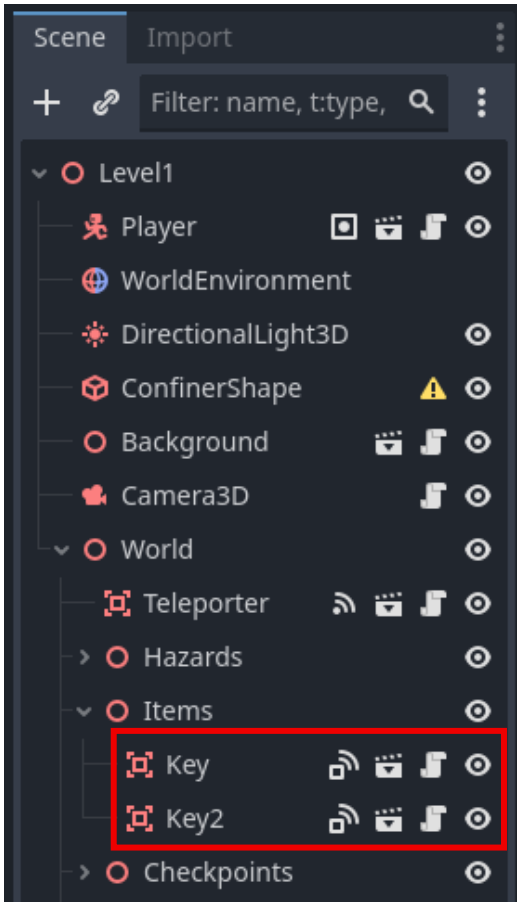
17

Implement keys.

Recall that the `keys_left` global variable initialized to `0` from the `LevelManager`. This variable will be incremented as the keys spawn in and decrement as they get collected.

```
3    var current_checkpoint : Vector3 = Vector3.ZERO
4    var player : Player
5    var keys_left : int = 0
```

In the **level_1** scene, notice that there are 2 key nodes as children to Items.



18 In **FileSystem**, open the **key.gd** script.

Find **TODO 2**. On the next line, define a `_ready()` function that has no parameters and returns **void**.

```
3  # --------------------
4  # TODO 2
5  # Create the _ready method and register a new key
6  # --------------------
7  func _ready() -> void:
8
```

19 Within the `_ready()` function, increment `LevelManager.keys_left` by `1`.

This will add 1 to the total number of keys that still exist in the scene each time a key loads.

```
3  # --------------------
4  # TODO 2
5  # Create the _ready method and register a new key
6  # --------------------
7  func _ready() -> void:
8      LevelManager.keys_left += 1
9
```

20 Find **TODO 3.** On the next line, define an `_on_body_entered()` function. It should have one parameter called `body` of type `Node3D` and should return `void`.

Within the `_on_body_entered()` function, write an `if` statement that checks if `body` is in the group `"Player".`

Within the `if` statement, decrement the `keys_left` variable from `LevelManager` by `1`.

On the next line, call the `queue_free()` method.

```
10
11  # --------------------
12    # TODO 3
13    # Create the _on_body_entered method to check collection
14    # --------------------
15
16
17
18
19
```

21

Check that the code matches the screenshot.

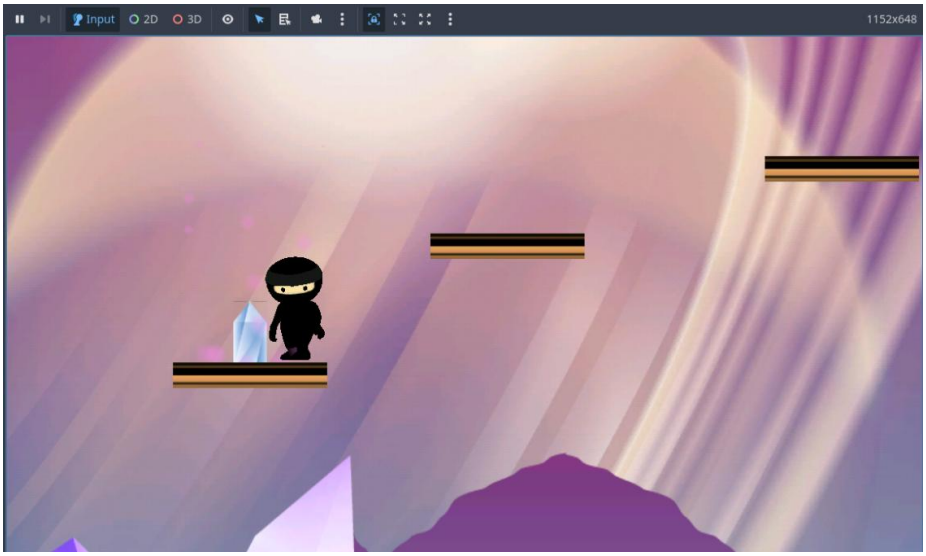
Decrementing the `keys_left` variable signifies that the key has been removed from the scene.

The call to `queue_free()` will remove the key that came into contact with the player from the scene.

```
# --------------------
# TODO 3
# Create the _on_body_entered method to check collection
# --------------------
func _on_body_entered(body: Node3D) -> void:
    if body.is_in_group("Player"):
        LevelManager.keys_left -= 1
        queue_free()
```

22

Playtest the game.



Run into the keys and ensure that they are being removed from the scene.

After collecting the keys, run to the right of the level and step on the teleporter. Notice that nothing happens.

Pause for **Sensei Stop #2**!

Check in with a Code sensei before moving on. Make sure the **key pickups** are set up properly.

Reminder: Save your work!

23

Change the level once all the keys have been collected!

In the **Script** workspace, open the **level_manager** script.

Find **TODO 4**. On the next line, define a `change_level()` function that takes in one parameter called `scene_path` of type String and returns `void`.

```
10  # --------------------
11  # TODO 4
12  # Create the change_level function
13  # --------------------
14  func change_level(string_path : String) -> void:
15      
```

24

Within the `change_level()` function, set `keys_left` to `0`.

```
10  # --------------------
11  # TODO 4
12  # Create the change_level function
13  # --------------------
14  func change_level(string_path : String) -> void:
15      
16      
```

25 Check that the code matches the screenshot.

When the level changes, the `keys_left` variable must be set back to `0` so the new level's keys can increment `keys_left` up to the correct number of keys.

On the next line, chain together calls to methods `get_tree()` and `call_deferred("change_scene_to_file", string_path)`.

```
10  # --------------------
11  # TODO 4
12  # Create the change_level function
13  # --------------------
14  func change_level(string_path : String) -> void:
15      keys_left = 0
16
17
```

26

Check that the code matches the screenshot.

The chained calls to **`get_tree()`** and **`call_deferred()`** wait until idle time in the game loop, then run the built-in method **`change_scene_to_file()`** with the string **`string_path`** as the argument. This will then load the level specified by **`string_path`**.

```
10  # --------------------
11  # TODO 4
12  # Create the change_level function
13  # --------------------
14  func change_level(string_path : String) -> void:
15      keys_left = 0
16      get_tree().call_deferred("change_scene_to_file", string_path)
```

New Concept: `call_deferred()`

This method accepts a string as the first argument containing the name of a different method. The following parameters contain arguments that the method specified in the first parameter would need in order to run.

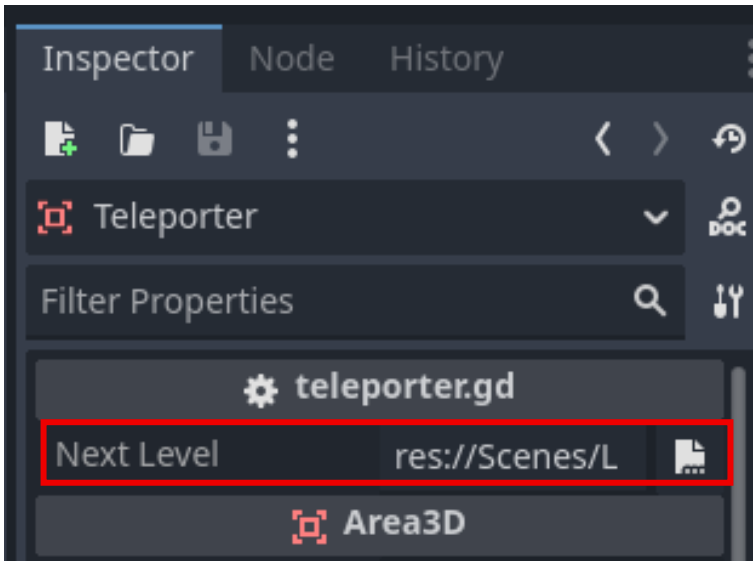
27

Script the teleporter to load the next level.

In **FileSystem**, open the **teleporter.gd** script.

Notice on line 3: `@export_file("*.tscn") var next_level : String`. The `@export_file` keyword denotes that the string variable `next_level`'s value will be set in the inspector. The argument `"*.tscn"` specifies that the variable will be a `.tscn` file. The `*` character denotes that that section can be a wildcard fill in the blank – in other words, the .tscn file can have any name, as long as it is the correct filetype.

```
2
3   @export_file("*.tscn") var next_level : String
4
```



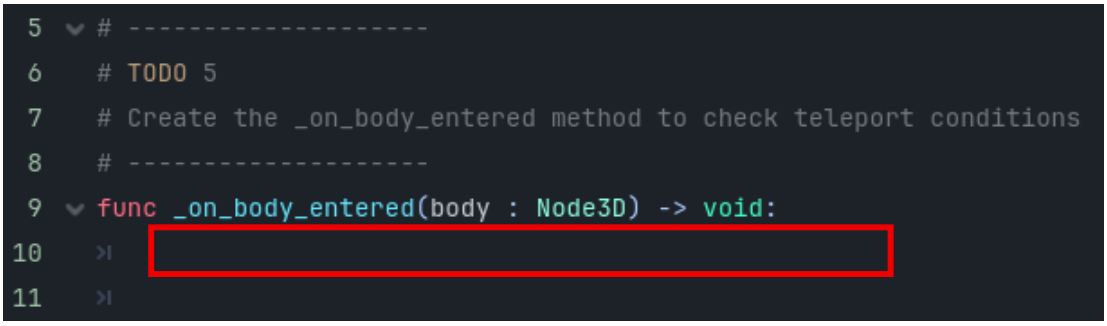
28

Find **TODO 5**. On the next line, define an `_on_body_entered()` function.

```
5  # --------------------
6  # TODO 5
7  # Create the _on_body_entered method to check teleport conditions
8  # --------------------
9
10
```

29 Check that the code matches the screenshot.

Within the `_on_body_entered()` function, write an `if` statement that checks if `body` is in the group `"Player"` `and` `keys_left` from `LevelManager` is `0`.



30 Check that the code matches the screenshot.

The `if` statement will check that when the teleporter body is entered, it has come into contact with the Player, and that there are no keys left in the scene to collect.

Within the `if` statement, call the `change_level()` function defined in **level_manager** with the argument `next_level`.

```
5  # --------------------
6  # TODO 5
7  # Create the _on_body_entered method to check teleport conditions
8  # --------------------
9  func _on_body_entered(body : Node3D) -> void:
10     if body.is_in_group("Player") and LevelManager.keys_left == 0:
11
12
```

31

Check that the code matches the screenshot.

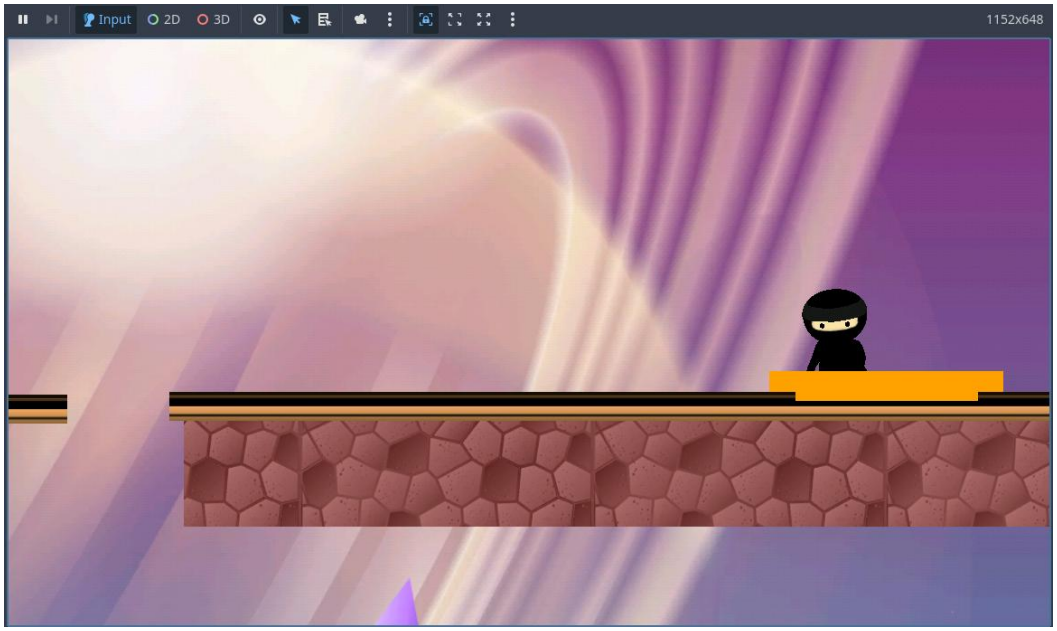
The call to `change_level()` will load the scene whose file path is held by the variable `next_level`. Since there is only one level scene defined in this project, **level_1**, the code will reload the first level upon its completion. The set value of `next_level` can be seen in the **Inspector** of **level_1 > World > Teleporter**.

```
5  # --------------------
6  # TODO 5
7  # Create the _on_body_entered method to check teleport conditions
8  # --------------------
9  func _on_body_entered(body : Node3D) -> void:
10     if body.is_in_group("Player") and LevelManager.keys_left == 0:
11         LevelManager.change_level(next_level)
```

32

Playtest the game.

Attempts to leave through the teleporter without collecting all keys should not reload the scene. The scene should be reloaded when all keys are collected and the player runs into the teleporter.



Pause for **Sensei Stop #3**!

Congratulations on creating your first game with keys and checkpoints in Godot! Great job!

Before submitting, check in with a Code Sensei to make sure the checkpoints, keys, and level changes work then reflect on the following:

- What did you learn about scripting checkpoints? Keys? Level changes?
- What did you enjoy most when creating this project?
- What was something you found difficult and why?

Reminder: Save your work!

Congratulations on completing **SB Activity 09: Amazing Ninja Worlds Part 1** in Godot – **You Rock**! You are now ready to save this project and submit it.

Continue your exploration with Godot by opening the **SB Activity 10: CyberFu Part 2** Ninja Guide.