CODE NINJAS®

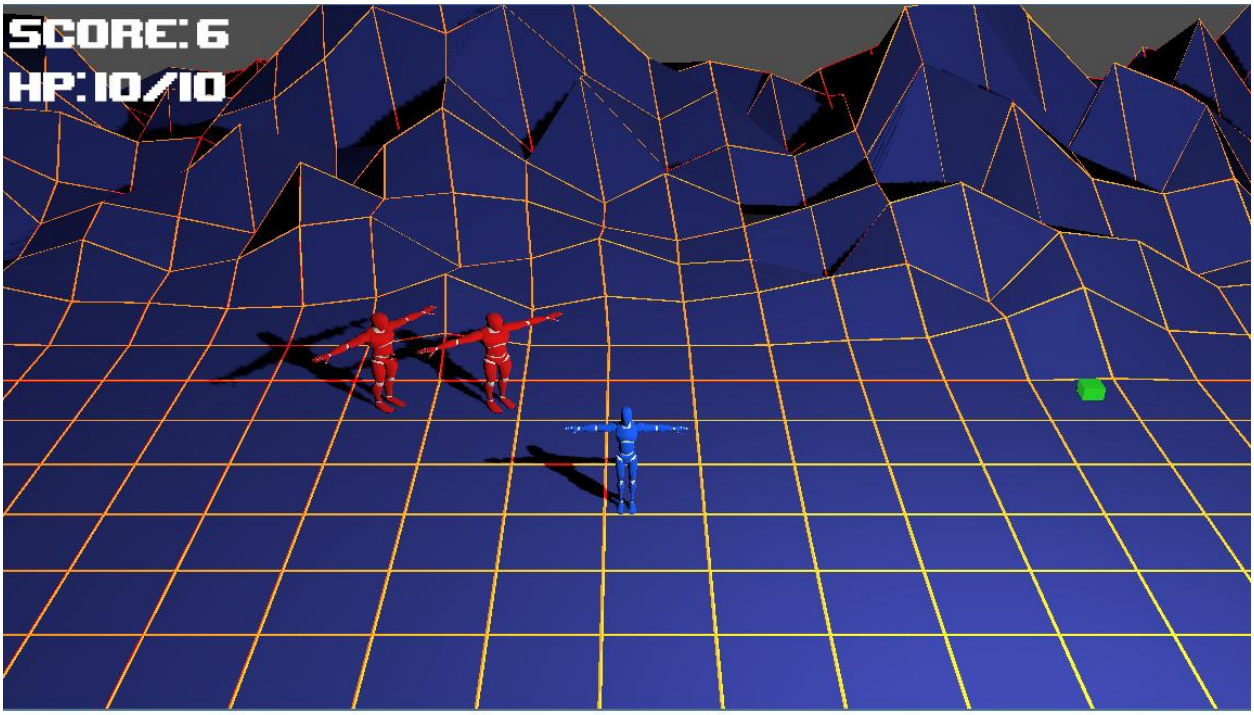
Silver Belt Ninja Guide

Activity 10: CyberFu Part 2

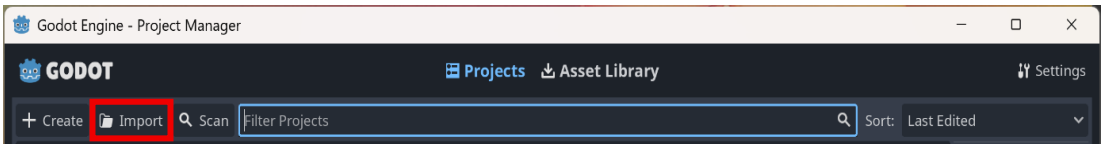
Activity 10: CyberFu Part 2

In this activity you'll learn to attack enemies and avoid being attacked by them.

New Mission: Add Player Attacking. Update Part 1 to create new enemy behaviors for attacking the player.

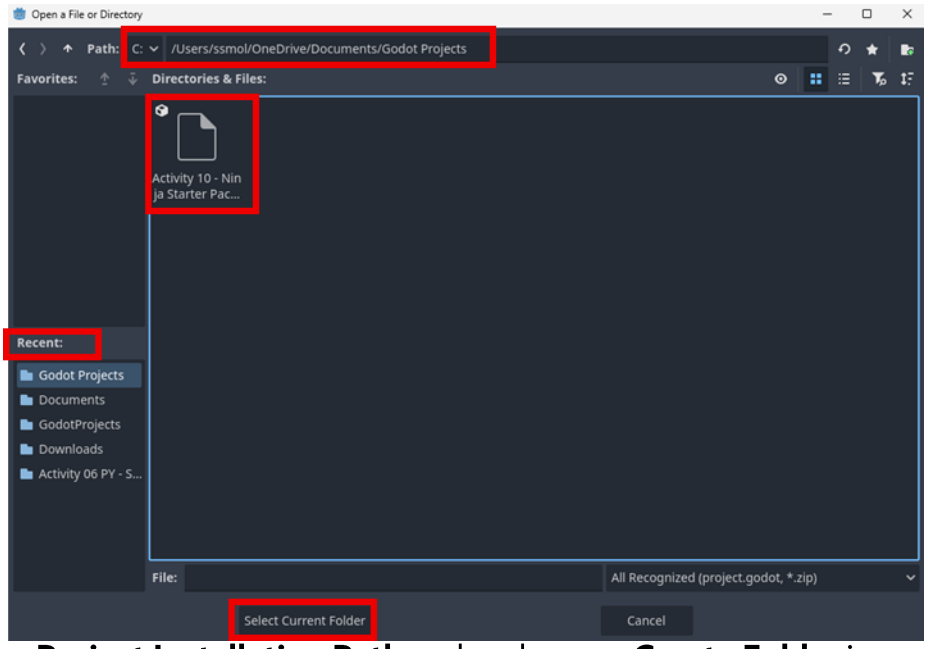


1 Open Godot and click **Import.**



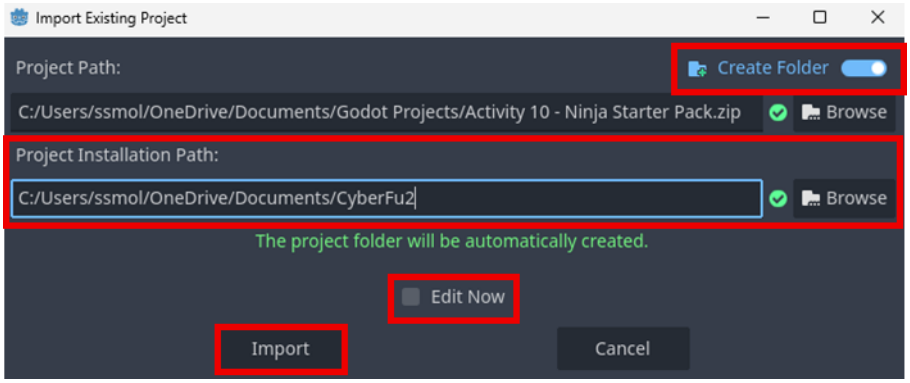
2 In the File Directory, navigate to the correct file path.

Select **SB Activity 10 – Ninja Starter Pack.zip** and click **Open**.



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

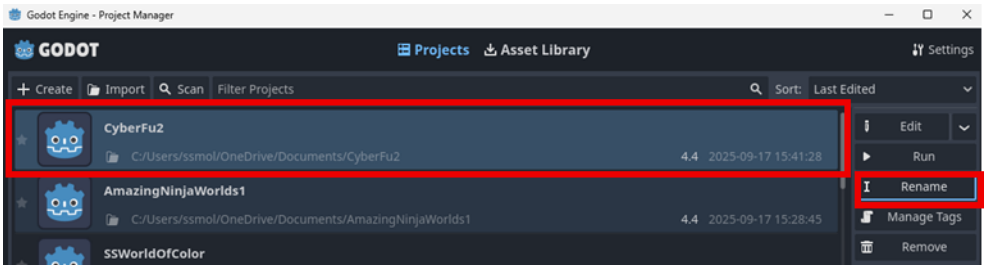
Uncheck Edit Now and click **Import**.



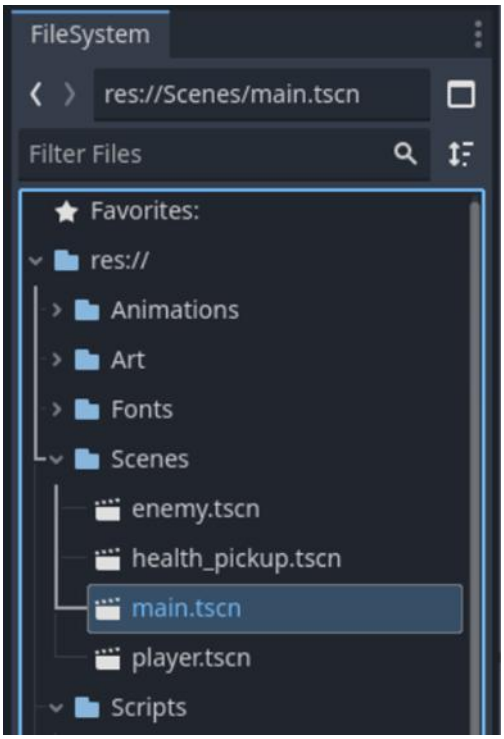
4 The project will appear at the top. Click on the project and select **Rename** on the right.

Update the Project Name to **[YourInitials]CyberFu2**.

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



5 In **FileSystem**, navigate to **main.tscn** and double click to open it.



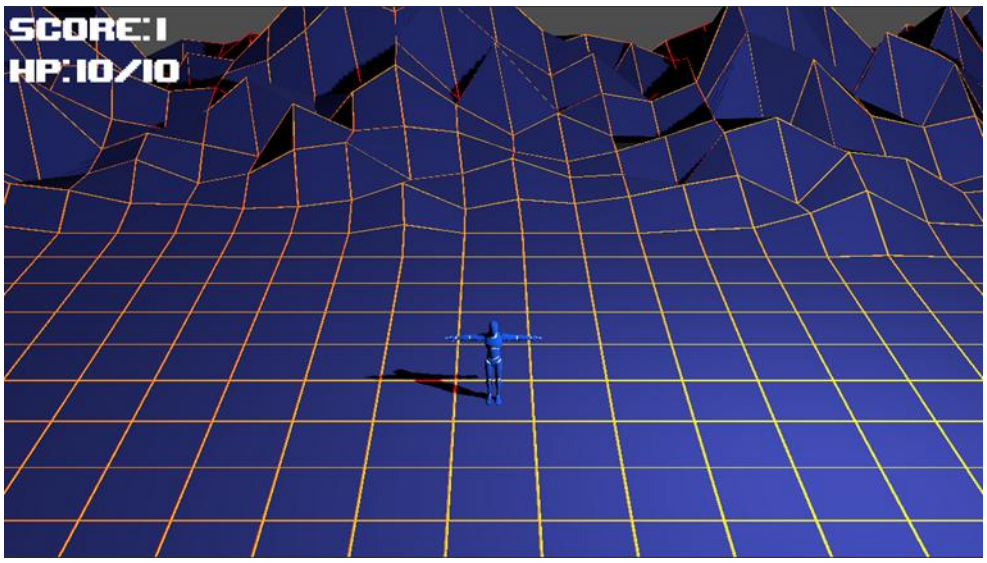
6 In the top right corner, click the **play** button to run the game.

Notice that the **player** is able to move around.

Enemies spawn and will turn to face the **player**.

Health packs spawn to restore **health**, and game over is called when the **player** runs out of **health**.

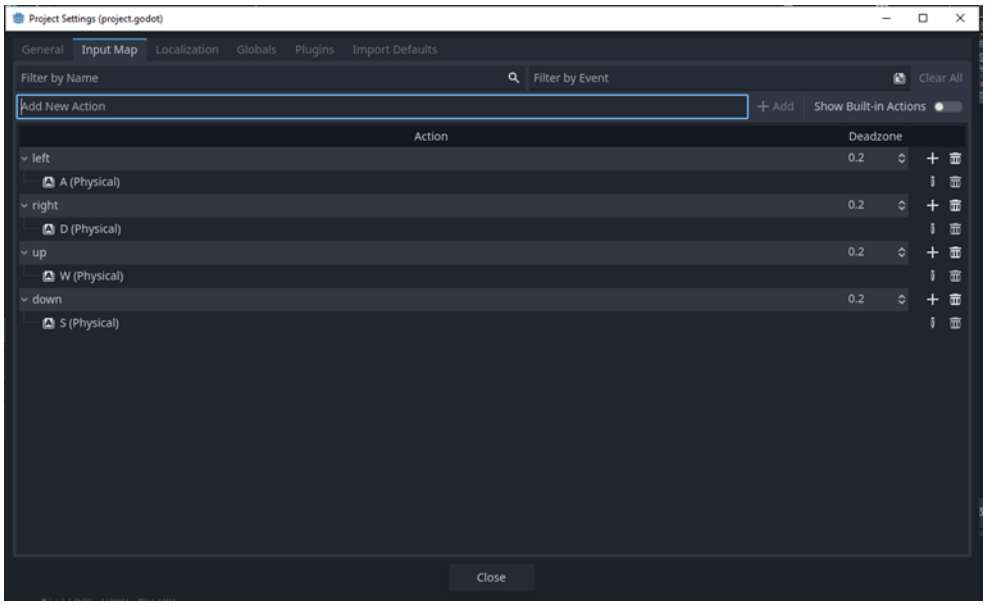
Is the player animated? Does it attack when the spacebar is pressed on the keyboard?



Close the playtest window.

7 Navigate to **Project Settings > Input Map**. Notice that inputs have already been defined for left, right, up, and down.

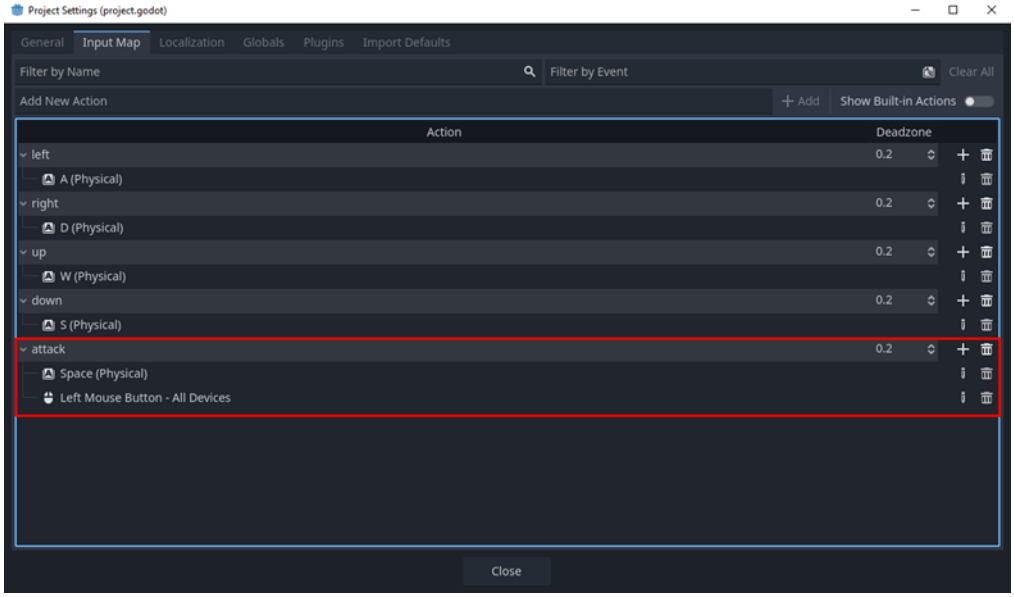
What other inputs might be needed?



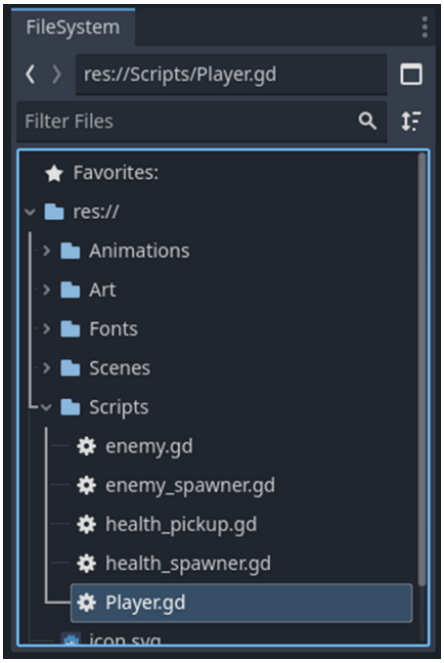
8 An attack input is needed so the player can defeat enemies.

Add a new **"attack"** action. Set **Space** and **Left Mouse Button** as events.

Close the Project Settings window.



9 In **FileSystem**, open **player.gd.**



Notice that the code in this script picks up where CyberFu Part 1 left off; the player can move and change health, and the game ends if the player's health reaches 0.

```
28
29  func _physics_process(_delta: float) -> void:
30      # TODO 1
31
32      var input_direction = Input.get_vector("left", "right", "up", "down")
33      velocity.x = input_direction.x * move_speed
34      velocity.z = input_direction.y * move_speed
35      rig.rotation.y = atan2(input_direction.x, input_direction.y)
36      move_and_slide()
37
38
39  func change_health(change: int):
40      health += change
41      health = clamp(health, 0, max_health)
42
43      if health > max_health:
44          health = max_health
45
46      update_health_ui()
47
48      if health <= 0:
49          game_over_triggered()
50
```

Currently, the player can't attack and is not animated. Notice the reference to the player's **`AnimationPlayer`** is declared near the top of the script as the **`anim`** variable.

```
1   extends CharacterBody3D
2
3   @export var move_speed: float = 5.0
4   @export var max_health: int = 10
5
6   @onready var anim: AnimationPlayer = $AnimationPlayer
7
8   @onready var hitbox: Area3D = $Area3D
9   @onready var restart_button: Button = $"../UI/RestartButton"
10  @onready var gameover_label: Label = $"../UI/GameOverLabel"
```

10

Code the player to fight back on user input!

Navigate to **TODO 1** inside the **`_physics_process()`** method.

Underneath, write an **`if`** statement. Use **`Input.is_action_just_pressed()`** to check if the **`"attack"`** input is detected.

```
29  func _physics_process(_delta: float) -> void:
30      # TODO 1
31      if Input.is_action_just_pressed("attack"):
32
```

11 If the attack input is detected, code the player to fight the enemy.

Inside the `if` statement, `play()` the `"Punch_Cross"` animation on the `AnimationPlayer`.

```
28
29 func _physics_process(_delta: float) -> void:
30     # TODO 1
31     if Input.is_action_just_pressed("attack"):
32         anim.play("Punch_Cross")
33
```

Reminder:

Pressing the space bar or left clicking triggers the "attack" input.

12

Check if the **"attack"** input has not been detected, but the player is moving. If so, reset the player's animation.

Add an `elif` statement that checks if the player's `velocity.length()` is greater than `0.1`. If so, play the `"Walk"` animation instead.

```
30      # TODO 1
31      if Input.is_action_just_pressed("attack"):
32          anim.play("Punch_Cross")
33      elif velocity.length() > 0.1:
34          anim.play("Walk")
35
```

Pro Tip:

Velocity is a **Vector3** with x, y, and z components, so it cannot be directly compared with a **float**. Instead, its length can be used here.

13 If neither of these conditions are met, the player is standing still.

Use an `else` statement to play the `"Idle"` animation.

```
30      # TODO 1
31      if Input.is_action_just_pressed("attack"):
32          anim.play("Punch_Cross")
33      elif velocity.length() > 0.1:
34          anim.play("Walk")
35      else:
36          anim.play("Idle")
37
```

Pro Tip:

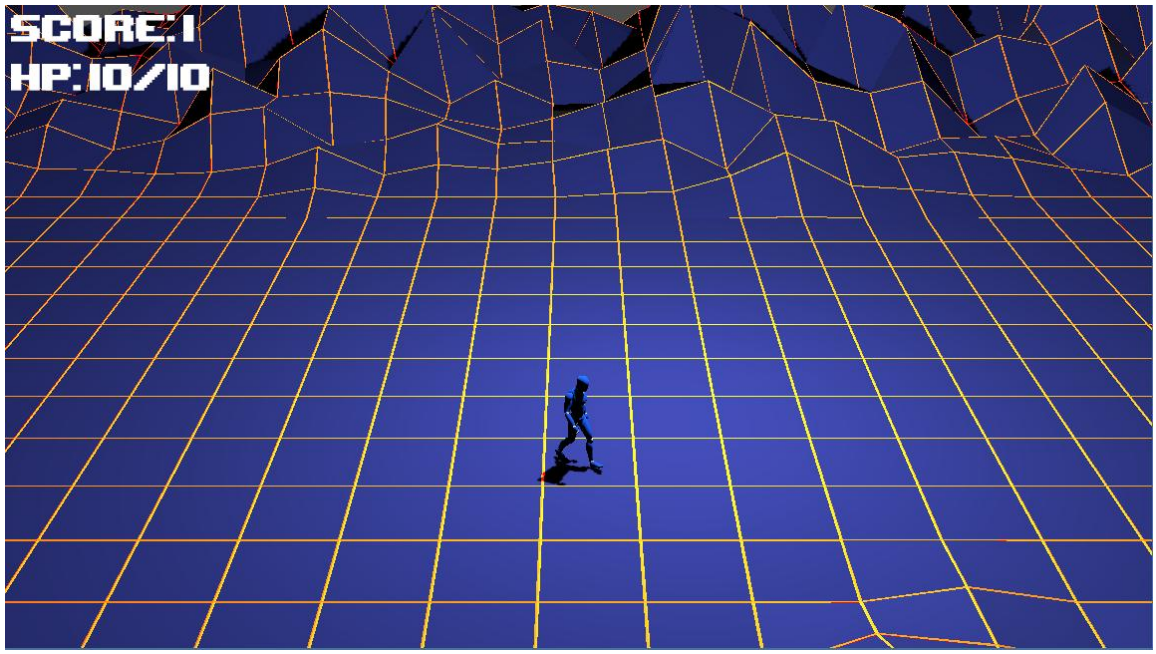
Be sure to note the capitalization of the action and animation names!

14

Playtest the game!

Use **WASD** to walk around and the **spacebar** or **left click** to punch.

Do the walk and idle animations work as intended? What about the punch animation?



This is because `Input.is_action_just_pressed()` is only true on the first frame the input is detected. On the next frame, this condition is false, and the code moves on to play either the walk or idle animations.

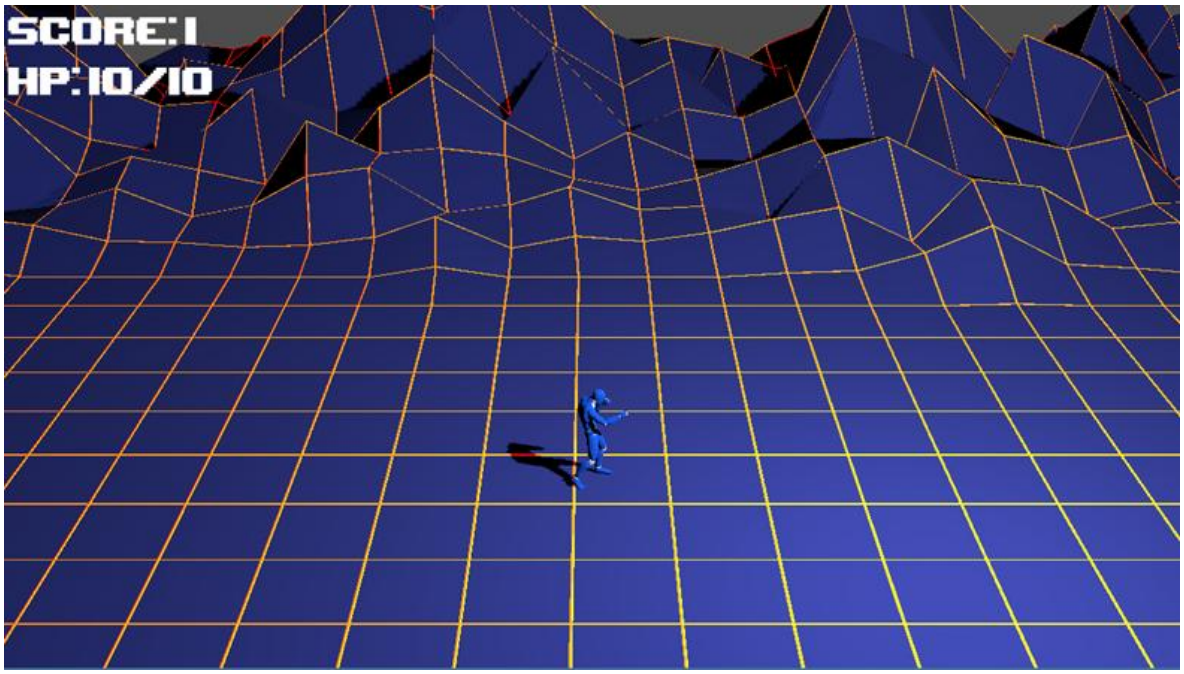
15 Make sure the punch animation has fully played before switching to another animation.

Add a second condition to the `if` statement that uses an `or` operator and the `anim.current_animation` property to check if the player's current animation is `"Punch_Cross"`.

> **`current_animation`:** Can be used to check the key of the currently playing animation. If no animation is playing, the property's value is an empty string.
>
> **Parameters:** None
>
> **Returns (*Void*)**: This property is void; it doesn't return anything.

```
28
29  func _physics_process(_delta: float) -> void:
30      # TODO 1
31      if Input.is_action_just_pressed("attack") or anim.current_animation == "Punch_Cross":
32          anim.play("Punch_Cross")
33      elif velocity.length() > 0.1:
34          anim.play("Walk")
35      else:
36          anim.play("Idle")
37
```

16 Playtest the game! Does the punch animation fully play out?



There is one more change to make to the player animation. Currently, the player can slide around while punching, which looks strange!

To keep the player standing still while punching, `_physics_process()` will be exited before moving the player when the punch animation is playing.

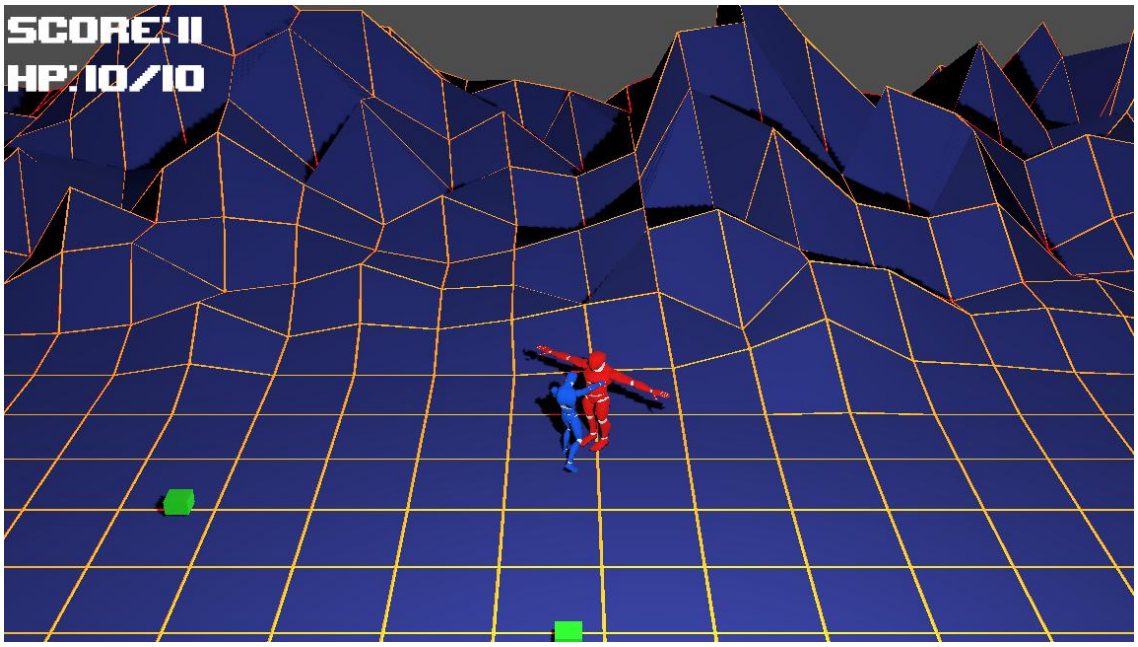
17 After `anim.play("Punch_Cross")` inside the `if` statement, add `return` on a new line.

```
30      # TODO 1
31      if Input.is_action_just_pressed("attack") or anim.current_animation == "Punch_Cross":
32          anim.play("Punch_Cross")
33          return
34      elif velocity.length() > 0.1:
```

Reminder:

The `return` keyword exits the current method or function, returning to its previous place.

18 Playtest the game one more time and notice how the player is animating perfectly! The player stands in place while the punch animation plays.



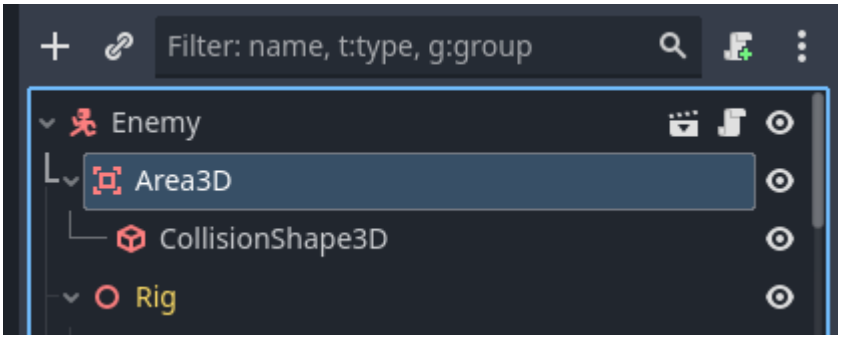
However, the enemies don't do anything yet except turn to face the player. They can't even be defeated by the player's punches!

Pause for **Sensei Stop #1**!

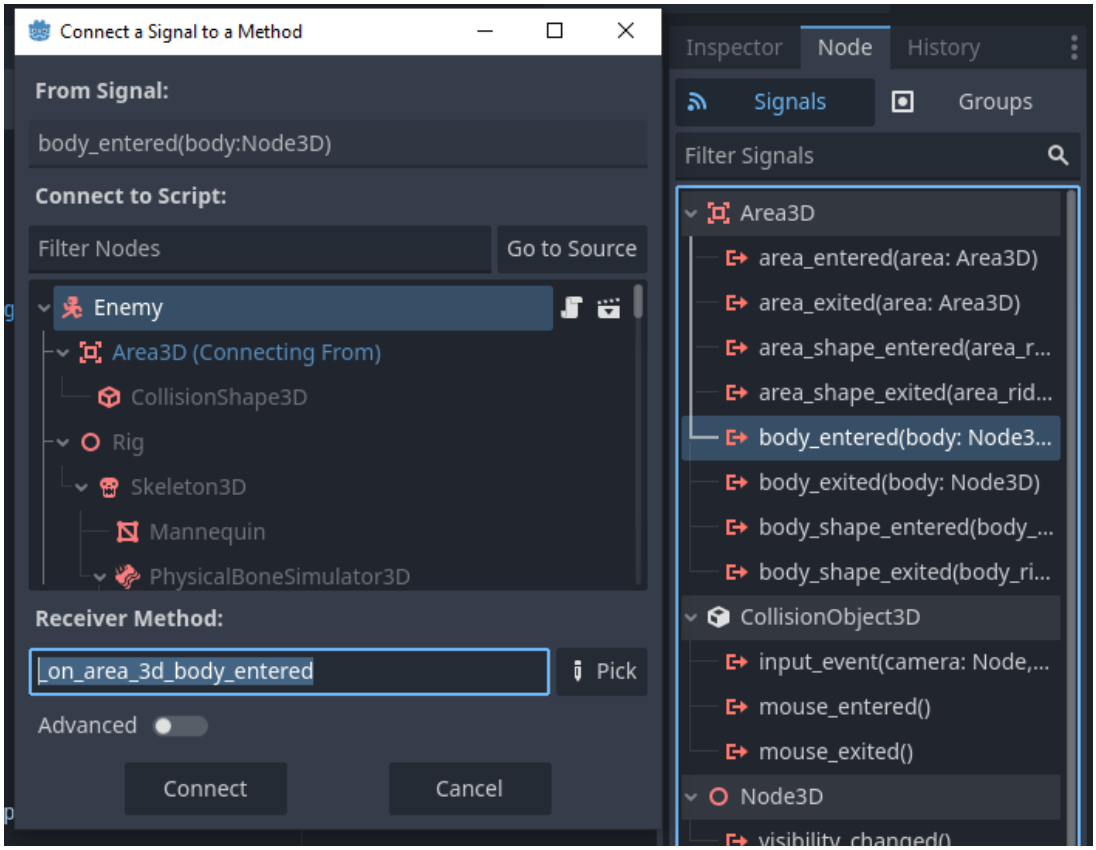
Check in with a Code Sensei before moving on. Make sure the punch, walk, and idle animations play at the right times, and the player cannot move while punching.

Reminder: Save your work!

19 Navigate to **enemy.tscn**. In **Scene**, select the **Area3D** node.



In **Inspector > Node**, connect the **Area3D** node's `body_entered` signal to **enemy.gd**. This will create `_on_area_3d_body_entered()` in the enemy script, which will detect when an enemy is punched by the player.



20 Navigate to `_on_area_3d_body_entered()` in **enemy.gd**.

Inside the method, check if the body is in the **`"player"`** group.

Add a second condition to also check if the player's current animation is `"Punch_Cross"` so the enemies aren't destroyed when the player walks into them.

```
23
24  func _on_area_3d_body_entered(body: Node3D) -> void:
25      # if in "player" group and punch anim is playing
26
```

21 Update the score and destroy the enemy when punched by the player.

Inside the `if` statement, increase the player's score and destroy the enemy.

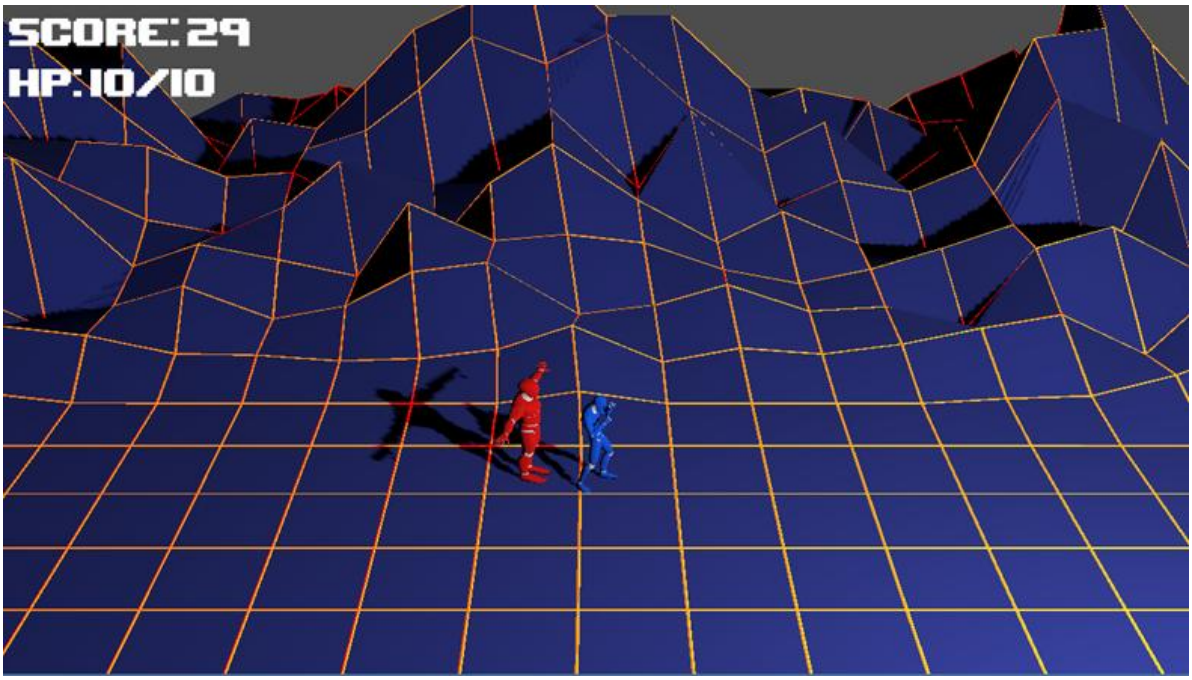
```
23
24  func _on_area_3d_body_entered(body: Node3D) -> void:
25      # if in "player" group and punch anim is playing
26          # increase player score
27          # destroy enemy
28
```

22 Check the code! Update the script as needed.

```
23
24 func _on_area_3d_body_entered(body: Node3D) -> void:
25     if body.is_in_group("player") and player.anim.current_animation == "Punch_Cross":
26         player.score += 5
27         queue_free()
28
```

23 Playtest the game! Enemies should now be destroyed by the player's punches.

What happens with the Score UI when an enemy is punched by the player?



24 There is a delay between an enemy being destroyed and points being added to the score label. This is because currently, the score label is only updated once per second when the score timer adds 1 to the score.

To fix this, add a line updating the score label to the current score just above `queue_free()`.

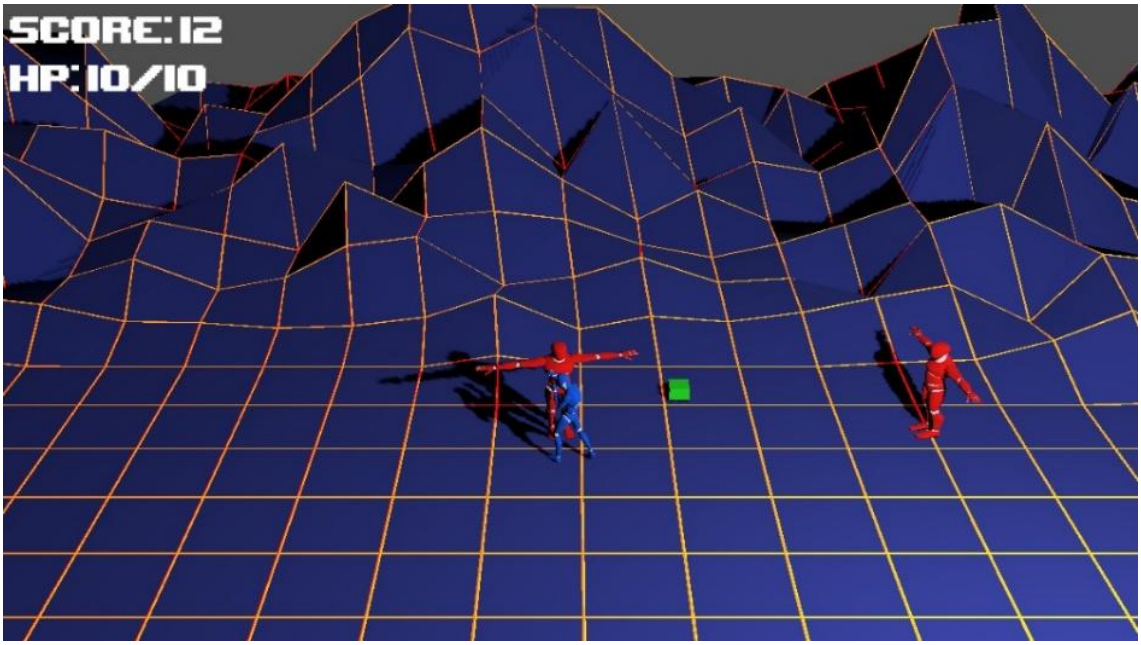
Use `$"/root/Main/UI/ScoreLabel"` to access the **ScoreLabel** in **main.tscn**. Set the **ScoreLabel** node's `text` property to the player's score.

Because the text property only takes strings, the player's score must be converted from an integer to a string.

```
23
24  func _on_area_3d_body_entered(body: Node3D) -> void:
25      if body.is_in_group("player") and player.anim.current_animation == "Punch_Cross":
26          player.score += 5
27          $"/root/Main/UI/ScoreLabel".text = str(player.score)
28          queue_free()
29
```

25 Playtest the game again and notice how the score updates instantly!

Now that the player's animations and punches are working, it's time to add in enemy behavior.



26

The enemies should only move when they aren't close enough to punch the player.

Find the `hit_distance` variable at the top of **enemy.gd**. This will be used to check if the enemy is close to the player.

In **enemy.gd**, find **TODO 2** inside `_physics_process()`. Add an `if` statement and use the `distance_to()` method to check if the distance to the player is greater than the enemy's hit distance. If so, call the `move_and_slide()` method.

```
16 func _physics_process(_delta: float) -> void:
17     if player:
18         var direction = (player.global_position - global_position).normalized()
19         velocity = direction * move_speed
20
21         # TODO 2
22         if global_position.distance_to(player.global_position) > hit_distance:
23             move_and_slide()
24
```

Reminder:

The `distance_to()` method returns the length of the vector between two positions.

27

Match the enemy's movement with its animation.

Below `move_and_slide()`, play the `"Walk"` animation.

```
21      # TODO 2
22      if global_position.distance_to(player.global_position) > hit_distance:
23          move_and_slide()
24          anim.play("Walk")
```

28

If the player is within the enemy's hit distance, play the punch animation.

Add an `elif` that checks if the enemy's current animation is `not in "Punch_Cross"`. The `in` keyword is used to check if a value is contained **within** another value.

Add a second condition that uses `is_playing()` to check if there is `not` an animation playing.

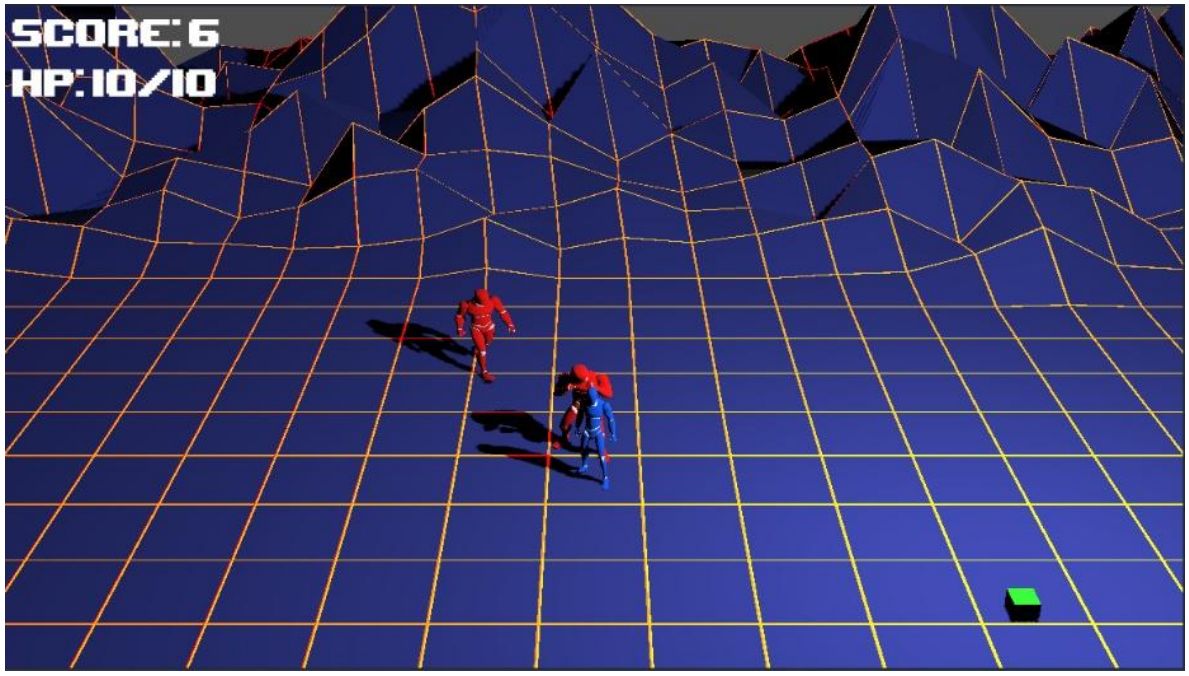
Now, if the enemy's current animation is **not** `"Punch_Cross"` **or** they do not have a current animation within the hit distance, the punch animation will play.

```
20
21      # TODO 2
22      if global_position.distance_to(player.global_position) > hit_distance:
23          move_and_slide()
24          anim.play("Walk")
25      elif anim.current_animation not in "Punch_Cross" or not anim.is_playing():
26          anim.play("Punch_Cross")
27
```

29 Playtest the game!

Enemies now walk toward the player; they also stop and punch the player when they get close.

Notice that their punches don't hurt the player yet, so that part needs to be added next.



Pause for **Sensei Stop #2**!

Check in with a Code Sensei before moving on. Make sure the enemies walk up to the player and play the punch animation when they get close.

Reminder: Save your work!

30 Navigate to **player.gd** to handle collisions with the enemies.

Under **TODO 3**, create a `handle_body_entered()` function that takes a `body` parameter, of type `Node3D`.

```
20    # TODO 3
21    func handle_body_entered(body: Node3D):
22        
```

31 Inside the function, check if `body` is a `PhysicalBone3D` node using the `is` operator.

```
20    # TODO 3
21    func handle_body_entered(body: Node3D):
22        if body is PhysicalBone3D:
23            
```

New Concept: `is` operator

In GDScript, the **is** operator functions differently than the == equality operator. Rather than checking if two things are equal, **is** checks if two things are the same type. Here, it is being used to check the body parameter's node type.

New Concept: PhysicalBone3D

The **PhysicalBone3D** node represents a segment of an animated 3D model, which connects to other **PhysicalBone3D** nodes by joints on each end.

32 Add a second condition using the **and** operator to check if **body** is in the **"enemy"** group.

```
20    # TODO 3
21  func handle_body_entered(body: Node3D):
22      if body is PhysicalBone3D and body.is_in_group("enemy"):
23
```

33 If the body colliding with the player is a bone in the enemy's body, the player is being punched!

Inside the **if** statement, use the **change_health()** function to decrease the player's health by **1** and **print()** a message to the Output panel.

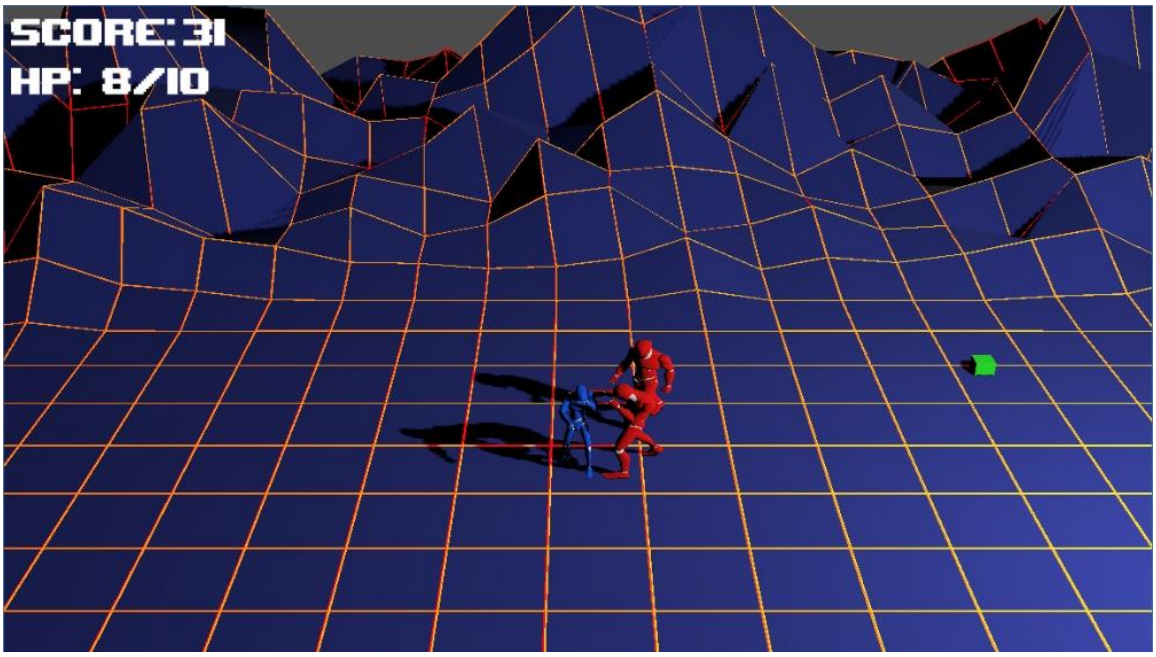
```
20    # TODO 3
21  func handle_body_entered(body: Node3D):
22      if body is PhysicalBone3D and body.is_in_group("enemy"):
23          change_health(-1)
24          print("Enemy hit player!")
25
```

34 Under **TODO 4** in the **_ready()** method, **connect()** the hitbox's **body_entered** signal to the **handle_body_entered()** function.

```
26
27  func _ready() -> void:
28      health = max_health
29      update_health_ui()
30      # TODO 4
31      hitbox.body_entered.connect(handle_body_entered)
32
```

35 Playtest the finished game!

The enemies can now damage the player. The player will need all their cyber fu skills to survive for long!



Pause for **Sensei Stop #3**!

Congratulations on upgrading the CyberFu Part 2 project in Godot! Great job!

Before submitting, check in with a Code Sensei to check that the enemies can damage the player when they get close, then reflect on the following:

- What did you learn about controlling animations?
- 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 10: CyberFu Part 2** 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 11: Labyrinth** Ninja Guide.