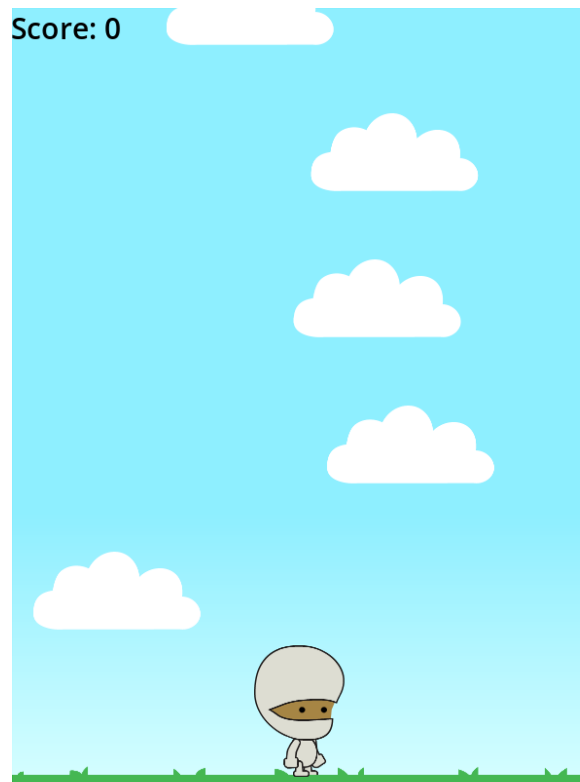
Silver Belt Ninja Guide

Activity 05: Cloud Hop

Activity 05: Cloud Hop

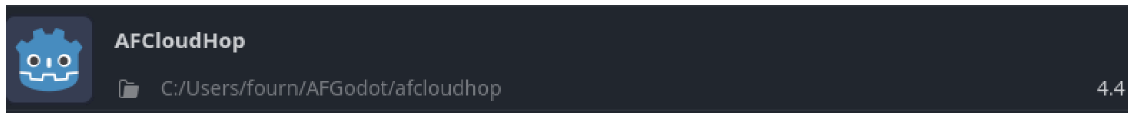
In this activity, you will create a platformer game to learn more about physics in Godot. You will enable the player to jump using `Input.is_action_just_pressed` to check for input, `move_toward` to simulate the jump, and `velocity` to check if the player is grounded. You'll also learn more about using the **CharacterBody2D** node.

In Cloud Hop, the sky's the limit! Jump from cloud to cloud to see how far you can go. Each cloud earns a point but be careful – falling off restarts the game!

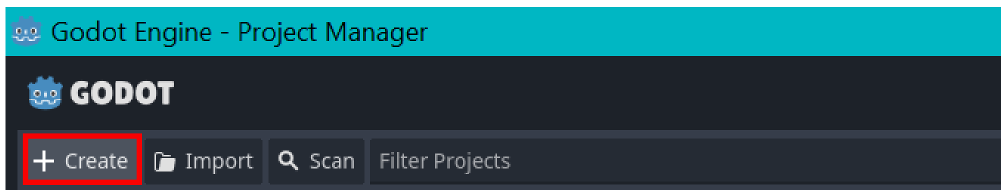


1 All projects will be stored in a path like: **/Users/[MyComputerUsername]/[MyInitials]Godot**

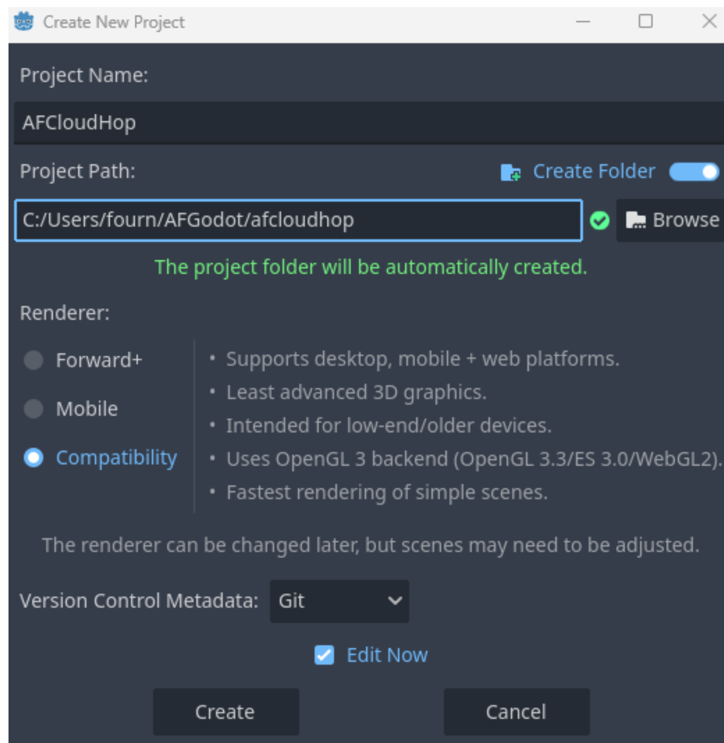
Don't worry if your path looks slightly different from the image shown! All computers have their own username.



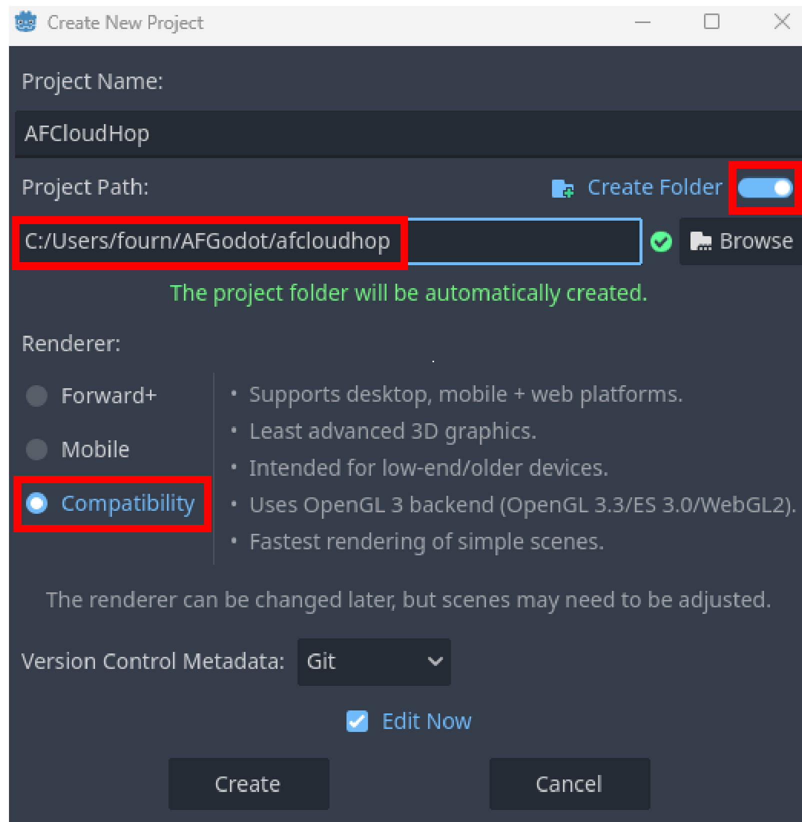
2 After opening Godot, in the top left corner select **+ Create.**



A **Create New Project** window will pop up. Name the project **MyInitialsCloudHop.**



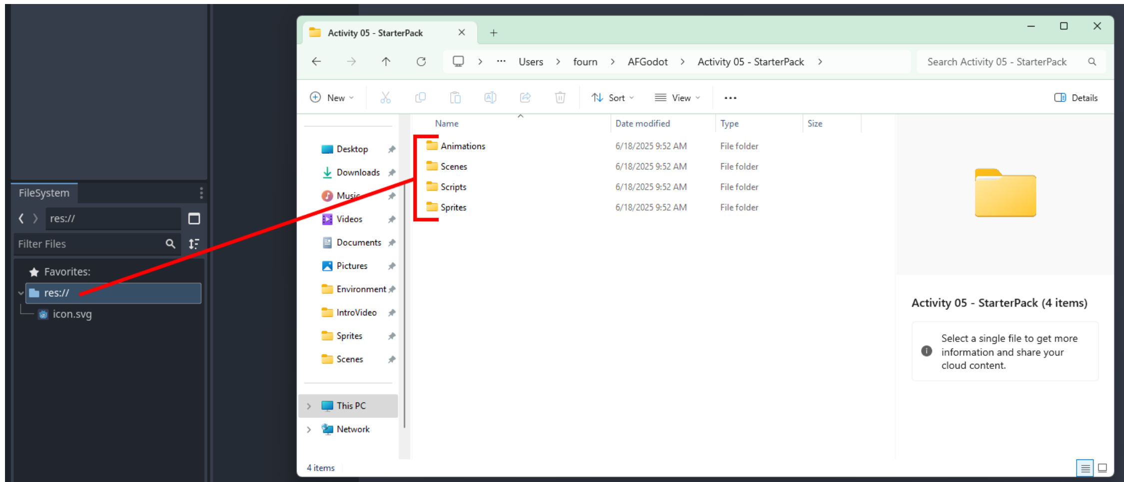
3 Check that **Create Folder** is turned on, and that the **Compatibility** mode for the renderer is being used. Then click **Create.**



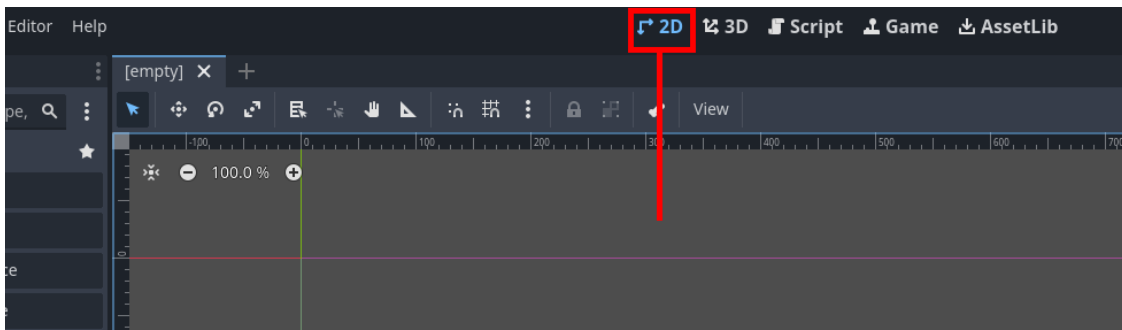
4

Don't create the **main scene** and **Main root** node just yet!

Extract **SB Activity 05 – Ninja Starter Pack** and select all folders inside. Drag them into the **res://** folder in **FileSystem**.



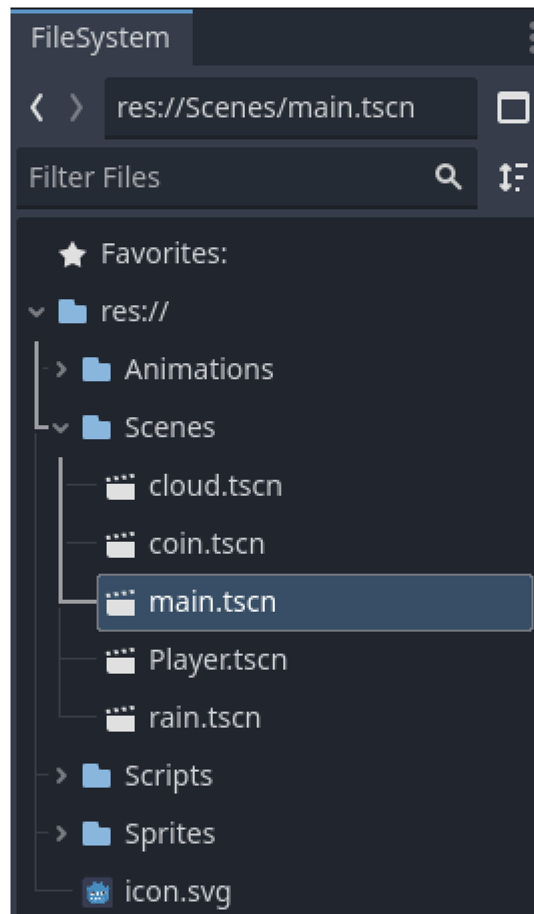
At the top center of the Godot editor, check that **2D** is selected.



Reminder:

Double-click on the folder icon from Downloads. Then right-lick on the zip file and select Extract All. Press CTRL + J on the keyboard to reopen the File Explorer.

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



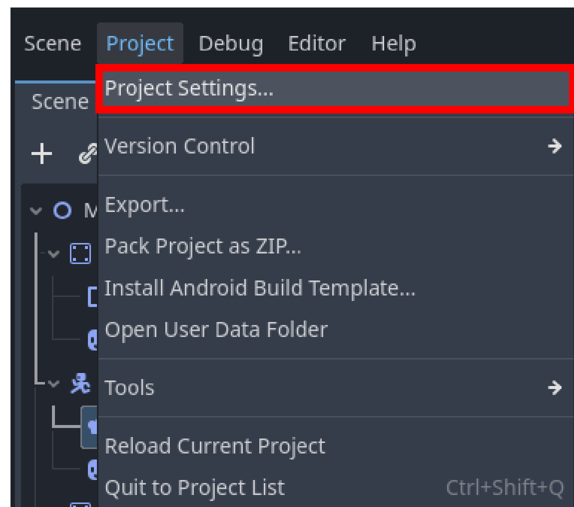
Reminder:

Click the arrows next to the folders to view their contents.

6

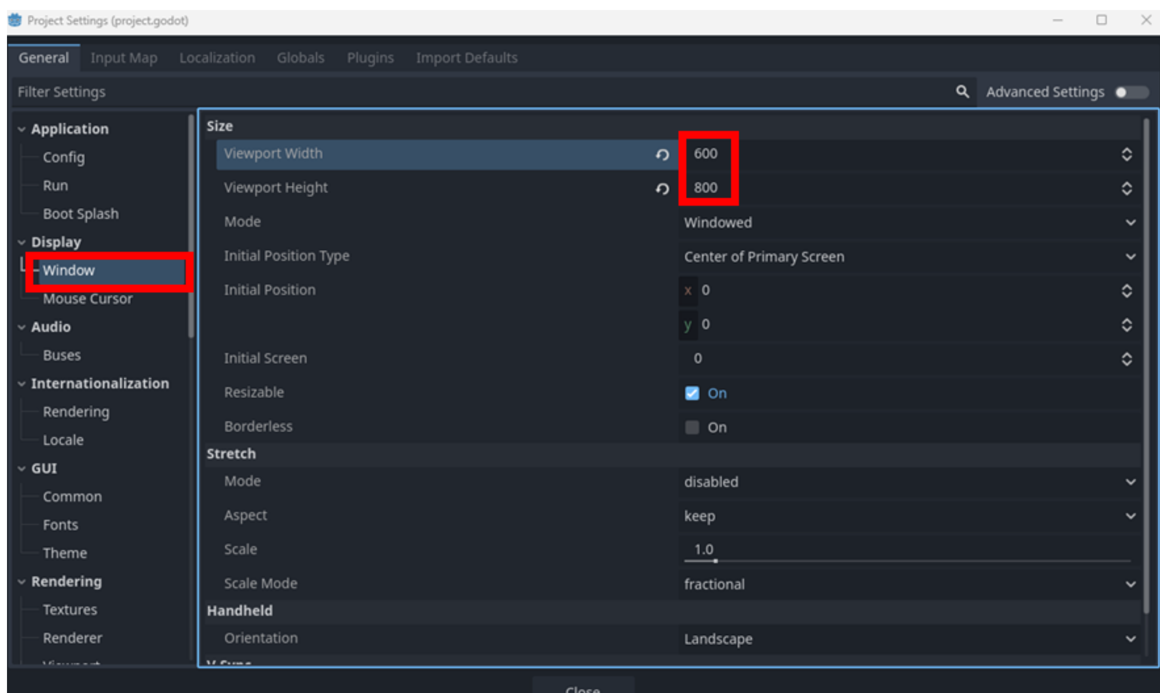
Update the viewport's dimensions!

Navigate to **Project**-> **Project Settings...**



In the **General** tab, scroll down to find **Display**. Open **Window**.

Set the **Viewport Width** to **600** and the **Viewport Height** to **800**.

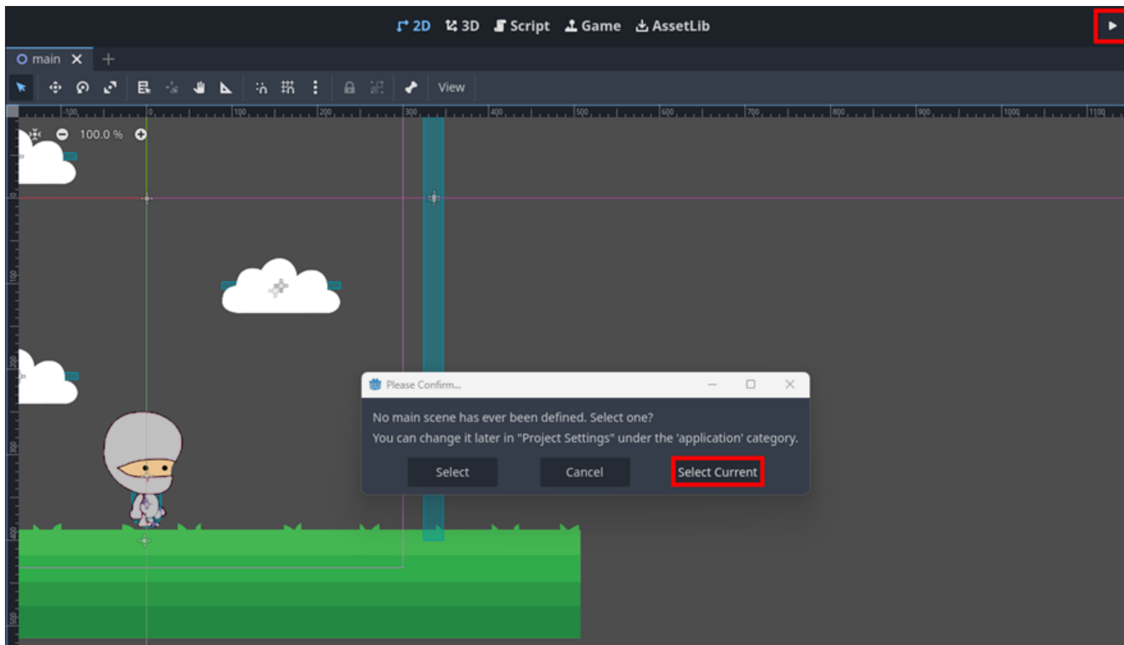


Close the **Project Settings** window.

7

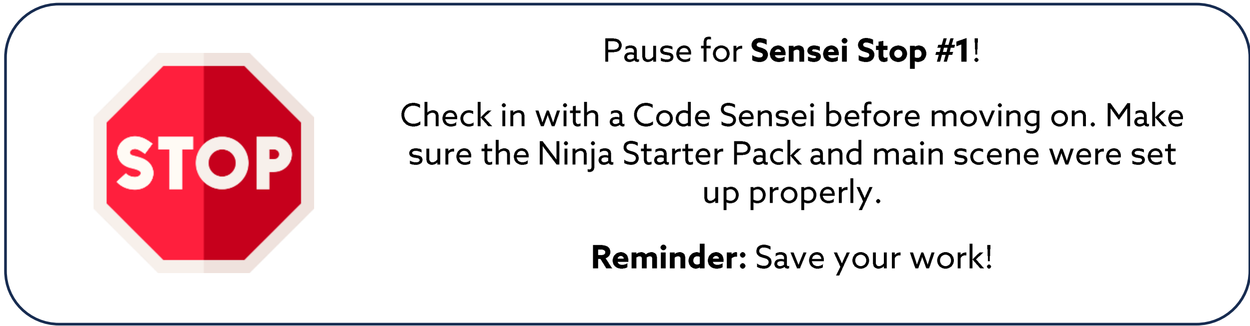
In the top right corner, click the **play** button to run the game.

Click **Select Current** to define the main scene.



Notice that the player does not move or jump currently.

Close the **Game** window.



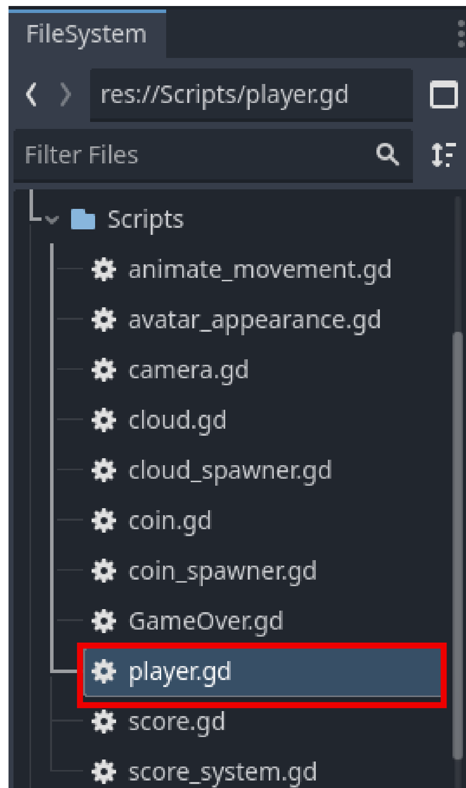
Pause for **Sensei Stop #1**!

Check in with a Code Sensei before moving on. Make sure the Ninja Starter Pack and main scene were set up properly.

Reminder: Save your work!

8 In **FileSystem**, navigate to the **player.gd** script located in the **Scripts** folder.

Double click the **player.gd** script to open.



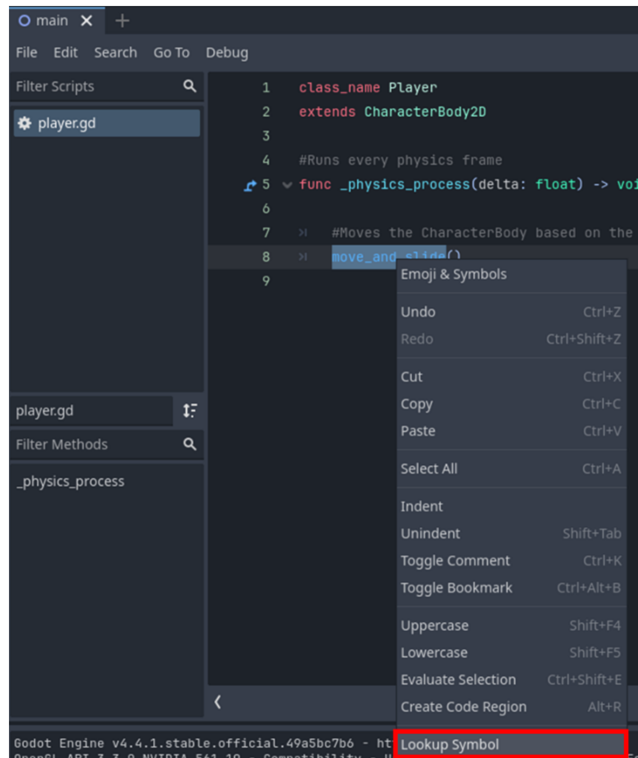
Notice that the script is fairly empty.

This script **extends** from **CharacterBody2D**, giving access to **CharacterBody2D** functions.

```
1   class_name Player
2   extends CharacterBody2D
3
4   func _process(delta: float) -> void:
5       return
6
7   #runs every physics frame
8   func _physics_process(delta: float) -> void:
9
10      #moves the CharacterBody based on the velocity
11      move_and_slide()
12
```

9 Notice the `move_and_slide()` function called at the end of `_physics_process`.

Highlight the `move_and_slide()` function, right click, then select **Lookup Symbol**.



Review the `move_and_slide()` section. This function uses velocity to update the player's location and considers collisions.

- `bool is_on_wall_only() const`

  Returns `true` if the body collided only with a wall on the last call of `move_and_slide()`. Otherwise, returns `false`. The `up_direction` and `floor_max_angle` are used to determine whether a surface is "wall" or not.

- `bool move_and_slide()`

  Moves the body based on `velocity`. If the body collides with another, it will slide along the other body (by default only on floor) rather than stop immediately. If the other body is a **CharacterBody2D** or `RigidBody2D`, it will also be affected by the motion of the other body. You can use this to make moving and rotating platforms, or to make nodes push other nodes.

  Modifies `velocity` if a slide collision occurred. To get the latest collision call `get_last_slide_collision()`, for detailed information about collisions that occurred, use `get_slide_collision()`.

  When the body touches a moving platform, the platform's velocity is automatically added to the body motion. If a collision occurs due to the platform's motion, it will always be first in the slide collisions.

  The general behavior and available properties change according to the `motion_mode`.

  Returns `true` if the body collided, otherwise, returns `false`.

10

Add code to program the player's movement!

At the top of the **player.gd** script, declare a new `export` `gravity_force` variable of type `Vector2`. Assign the variable an initial value of `Vector2(0.0, 3000.0)`.

```
1   class_name Player
2   extends CharacterBody2D
3
4   @export var gravity_force: Vector2 = Vector2(0.0, 3000.0)
5
6   func _process(delta: float) -> void:
7       return
8
9   #runs every physics frame
10  func _physics_process(delta: float) -> void:
11
12      #moves the CharacterBody based on the velocity
13      move_and_slide()
```

This will be used to control **gravity density** applied to the player each **physics** frame.

The larger the number, the more force that will be applied to the player when in the air.

Reminder:

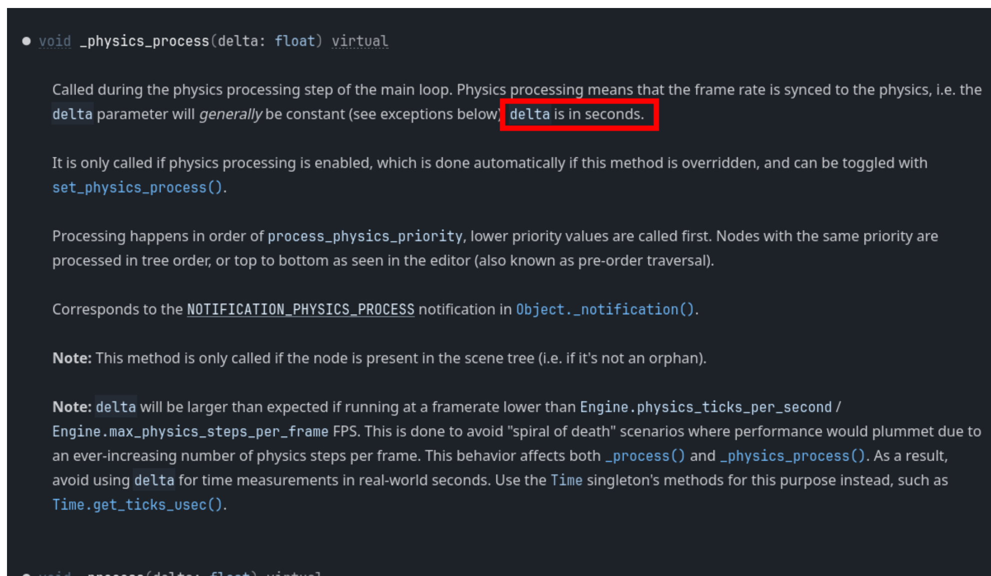
Because the `gravity_force` variable is exported, this parameter can be adjusted in the Inspector.

11

Indented inside the `_physics_process` function, use the `is_on_floor()` method to check if the Player is **not** touching the floor. Underneath, increase **velocity** by the value of `gravity_force` multiplied by `delta`.

The `is_on_floor()` function is used to make sure the player is in the air, before applying gravity. If the player is on the ground, gravity need not be applied.

Notice that gravity only affects the **y-axis** in this project since gravity is only changing the y-axis in `gravity_force`**.**



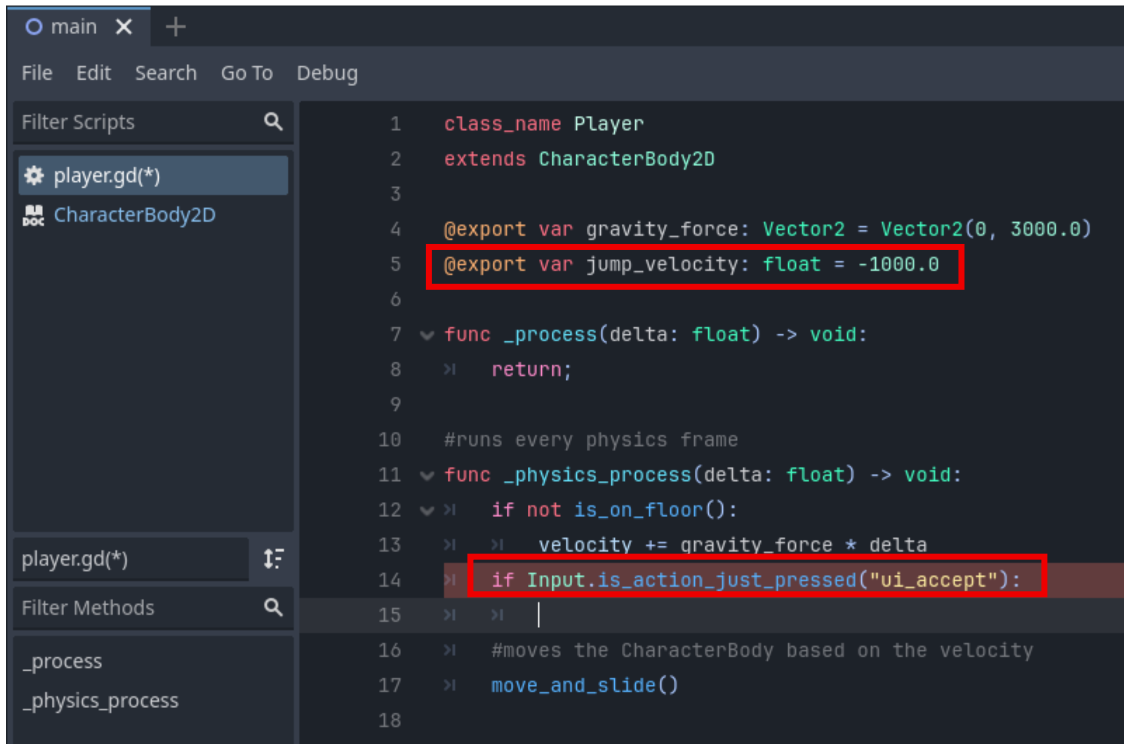
Reminder:

Observe that delta is in seconds and will differ depending on the frame rate of the game.

12 At the top of the **player.gd** script, declare a new `export` `jump_velocity` variable of type `float`. Set its value to `-1000.0`.

Inside the `_physics_process()` function and below the code that checks if the player is not on the floor, add another `if` statement that uses the `Input.is_action_just_pressed()` method. Use the code completion to select `"ui_accept"` as the parameter.

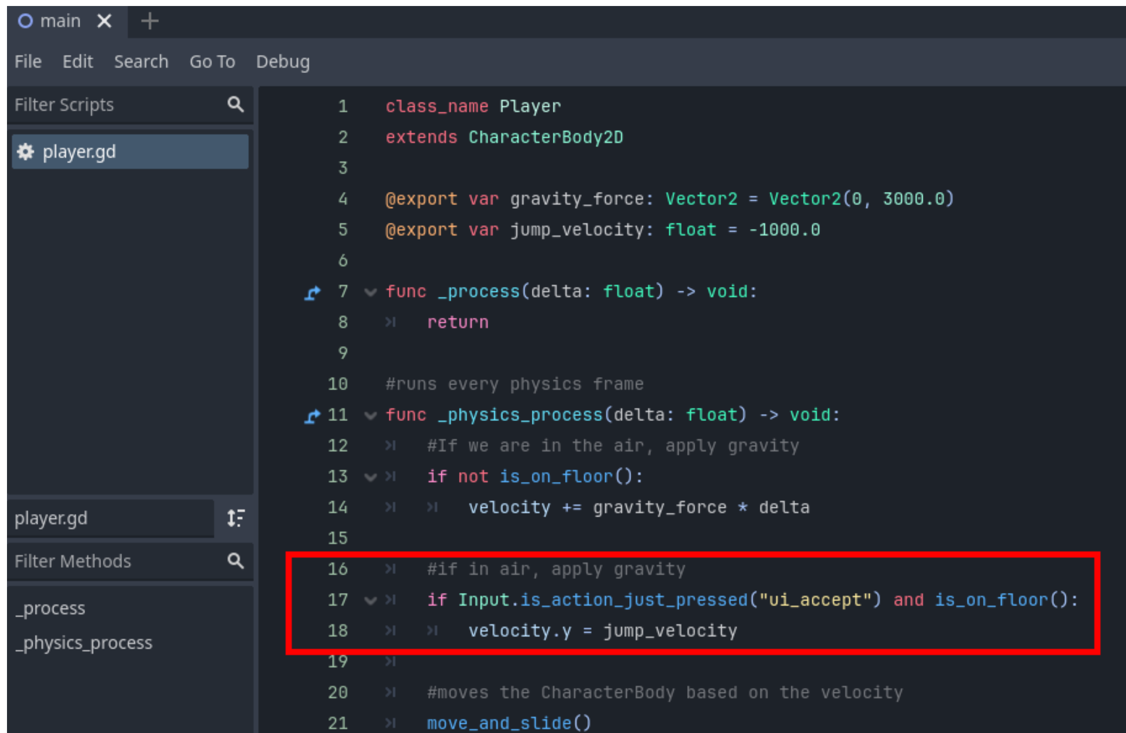
`Input.is_action_just_pressed()` is calibrated to use the space bar by default.



13

Add onto the **`if`** statement to check if the space bar has been pressed **`and`** the player **`is_on_floor()`**.

Opposite to **gravity**, the player should only jump when it **`is_on_floor()`**. Inside the **`if`** statement, set the **`velocity.y`** to the value of **`jump_velocity`**.




```
1  class_name Player
2  extends CharacterBody2D
3
4  @export var gravity_force: Vector2 = Vector2(0, 3000.0)
5  @export var jump_velocity: float = -1000.0
6
7  func _process(delta: float) -> void:
8      return
9
10 #runs every physics frame
11 func _physics_process(delta: float) -> void:
12     #If we are in the air, apply gravity
13     if not is_on_floor():
14         velocity += gravity_force * delta
15
16     #if in air, apply gravity
17     if Input.is_action_just_pressed("ui_accept") and is_on_floor():
18         velocity.y = jump_velocity
19
20     #moves the CharacterBody based on the velocity
21     move_and_slide()
```

Pro Tip:

`velocity.y` is set to an exact value (using an assignment operator =), instead of increased over time (using an addition assignment operator +=). This is to give the player an initial burst of speed, that will trail off when gravity is applied.

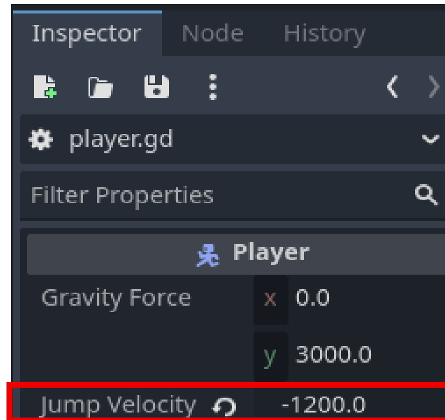
14

Playtest the game!

Press the **space bar**, then notice that the player jumps in the air.

Notice that the player doesn't have quite enough height to clear a cloud and is unable to move left and right.

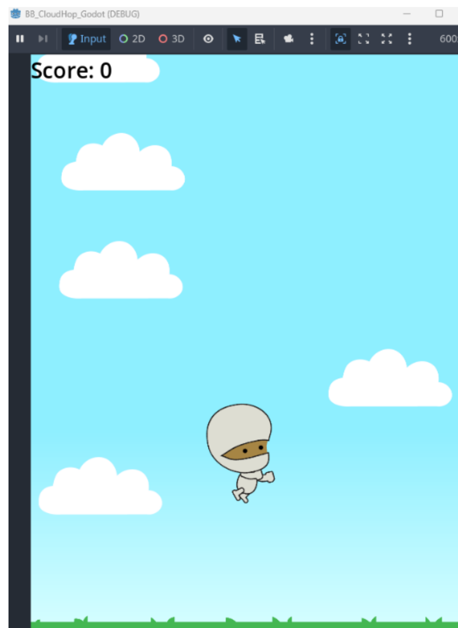
To fix this, update the **jump_velocity** paremeter within the **Player** node located in **Main**. What might the value be set to?



15

Now that jumping is taken care of, what about moving **left** and **right**?

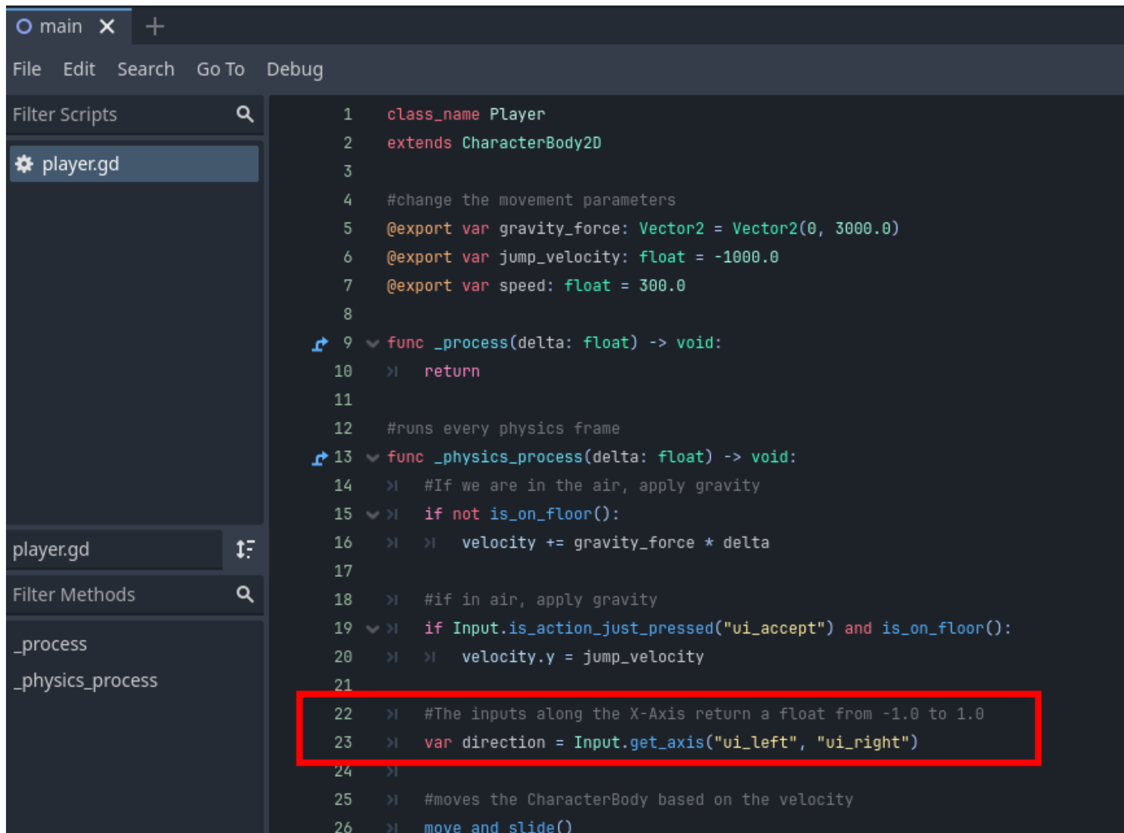
Next is adding directional movement.



Close the game window.

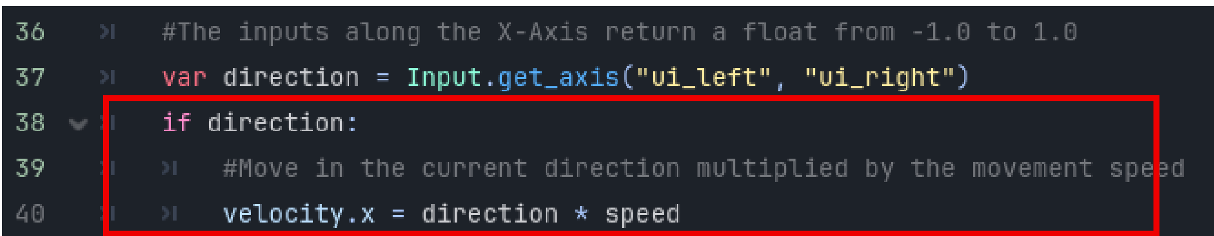
16 Inside the `_physics_process()` function and above the `move_and_slide()` method, add code to check if the player is pressing left or right movement keys.

Declare a `direction` variable equal to `Input.get_axis()` to see if the player is moving sideways. Pass `"ui_left"` and `"ui_right"` as the negative and positive actions.



17 On the line below, check if the `direction` variable has value to determine if the left or right buttons are pressed.

If so, set the player's x-velocity to the value of `direction` multiplied by `speed`.



18 Add an `else` statement to reset the player's x-velocity, if the left or right buttons are *not* pressed.

Inside the `else` statement, reassign the player's x-velocity using the `move_toward` method.

> **`move_toward()`:** Returns a new vector moved toward to by the fixed delta amount. Will not go past the final value.
>
> **Parameters:**
>
> 1. **from *(Vector2)*:** the current value.
> 2. **to *(Vector2)*:** the target value.
> 3. **Delta *(float)***: the maximum amount to move in one step.
>
> Returns: **Float**

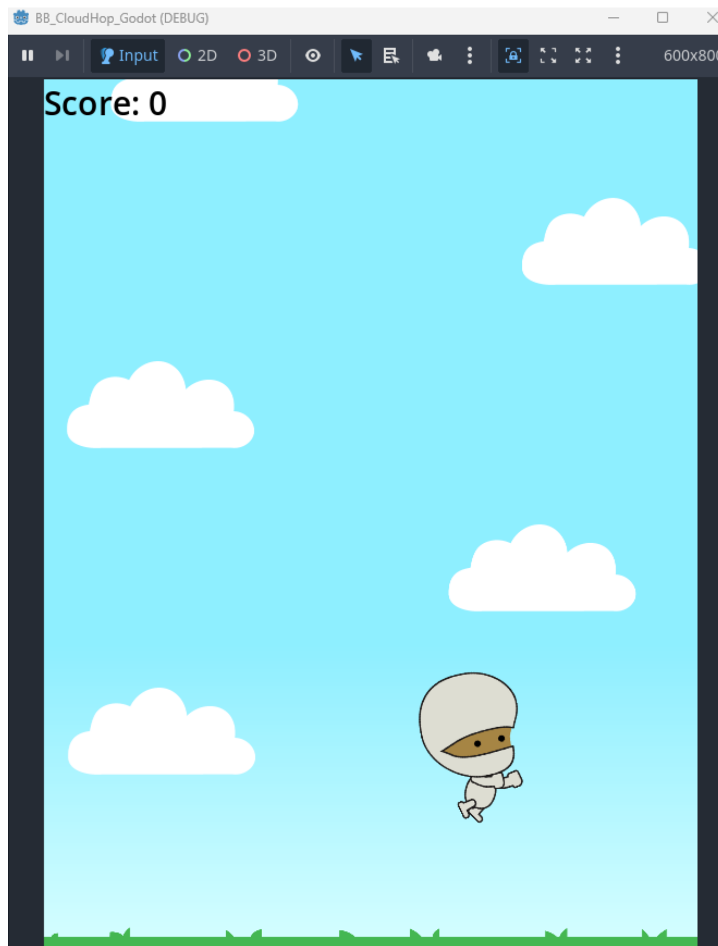
Pass `velocity.x`, `0`, and `speed`, as the three parameters. This will cause the player's x-velocity to gradually move towards `0` by the value of `speed`. The `move_toward` method helps to simulate realistic movement.

```
36  	#The inputs along the X-Axis return a float from -1.0 to 1.0
37  	var direction = Input.get_axis("ui_left", "ui_right")
38  	if direction:
39  		#Move in the current direction multiplied by the movement speed
40  		velocity.x = direction * speed
41  	else:
42  		#Otherwise move the velocity to 0
43  		velocity.x = move_toward(velocity.x, 0, speed)
44  
45  	#moves the CharacterBody based on the velocity
46  	move_and_slide()
```

19

Playtest the game!

Observe that the player can now **jump**, move **left**, and move **right**.



Close the game window.

Pause for **Sensei Stop #2**!

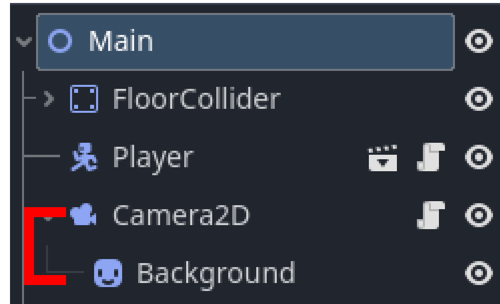
Check in with a Code Sensei before moving on. Make sure the **Player** is moving correctly.

Reminder: Save your work!

20

Notice that when playing the game, the background is stuck in place.

In **Scene**, reparent the **Background** node by dragging it underneath the **Camera2D** node.



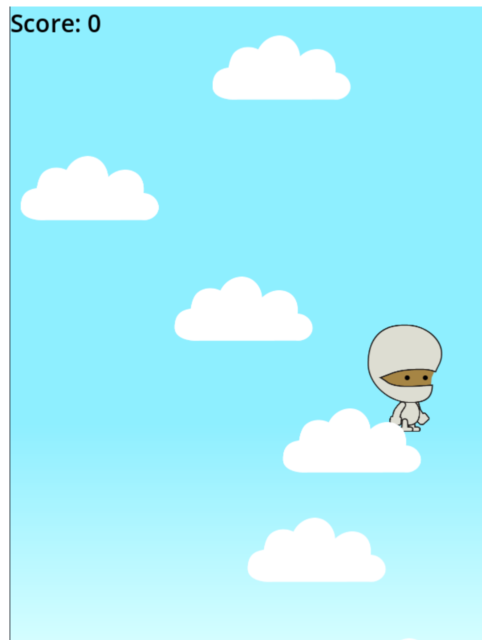
Reminder:

When parenting an object to another, the child will follow the parents

21

Playtest the game again.

Notice that when playing the game now, the background position will update with the camera position.



Close the **Game** window.

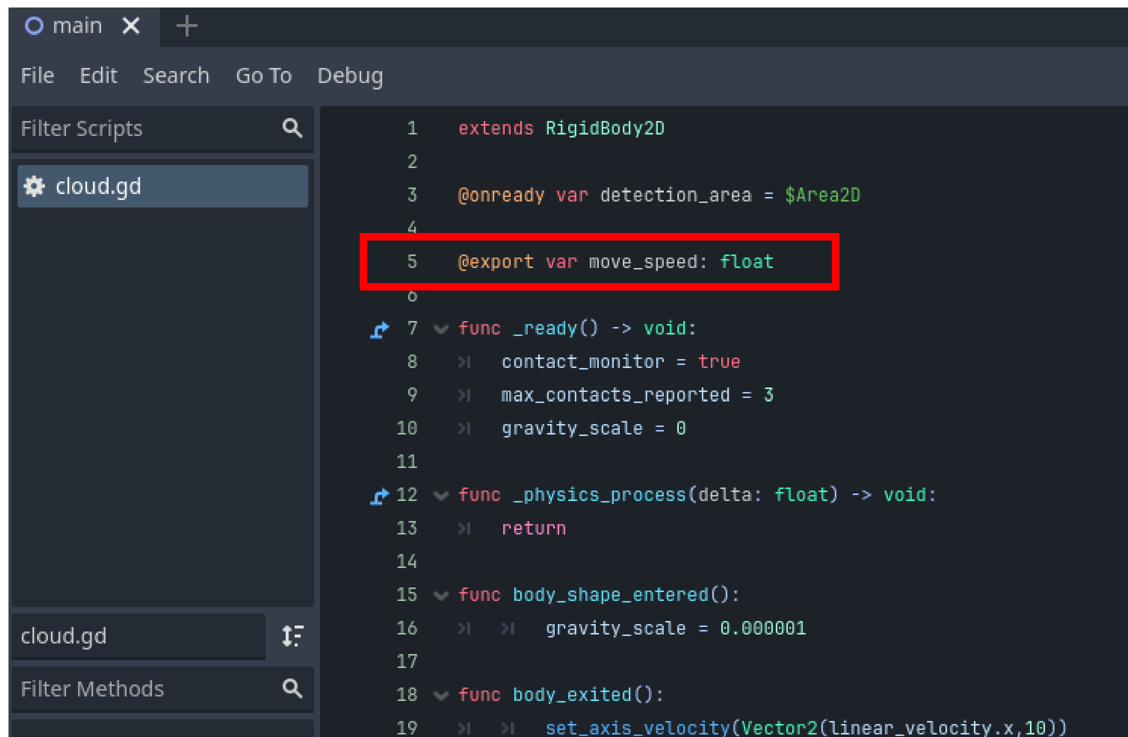
22

Add code so that the clouds move left and right, and bounce when on the edge of the screen.

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

Notice that the **cloud.gd** script already has some code in it. The `_ready()` method is called when the cloud node is first initialized. `body_shape_entered` and `body_exited` are used for collision.

At the top of the script, declare an `export` `move_speed` variable of type `float` to control how fast the cloud moves.



23 In the `_physics_process()` method, remove the `return` statement.

Add an `if` statement that checks if the cloud's x-position is less than or equal to `250`. If so, use `apply_impulse()` to move the cloud left. Pass a `Vector2()` as the parameter, then use `move_speed * 5` and `0` as the two coordinates to control the movement strength.

> `apply_impulse()`: applies a positioned impulse to the body.
>
> **Parameters:**
>
> 1. **impulse *(Vector2)***: the strength of the impulse.
> 2. **position *(Vector2)***: the offset from the body origin in global coordinates.

`apply_impulse()` is used here since the force is only applied once and is instant. This helps simulate the force as a collision to give that instant direction change.

Use the same `apply_impulse()` call in `_ready()` to initialize the clouds' speed.

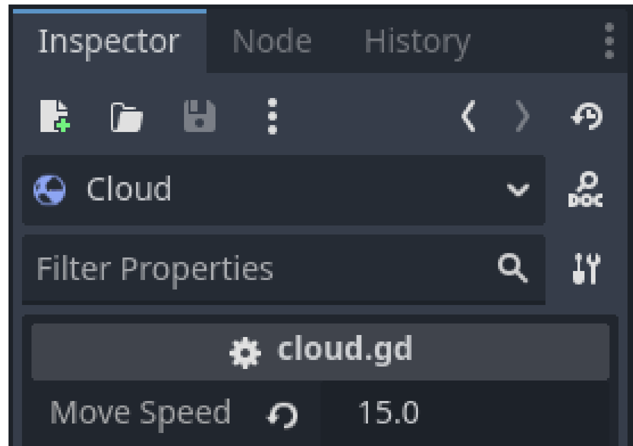
```
extends RigidBody2D

@onready var detection_area = $Area2D

@export var move_speed: float

func _ready() -> void:
	contact_monitor = true
	max_contacts_reported = 3
	apply_impulse(Vector2(move_speed * 5,0))
	gravity_scale = 0

func _physics_process(delta: float) -> void:
	if position.x <= 250:
		apply_impulse(Vector2(move_speed * 5,0))
	if position.x >= -250:
		apply_impulse(Vector2(-move_speed * 5,0))

func body_shape_entered():
		gravity_scale = 0.000001

func body_exited():
		set_axis_velocity(Vector2(linear_velocity.x,10))
```

Repeat this step to add another **if** statement that checks if `position.x` is greater than or equal to **-250**. Use `-move_speed` to move the cloud right.

24 In **FileSystem**, open **cloud.tscn** in the **Scenes** folder.

In the **Cloud** node's **Inspector**, tinker with the value of **Move Speed**.

Keep the value positive and try different speeds. Test by saving and playing the game!



Pause for **Sensei Stop #3**!

Check in with a Code Sensei before moving on. Make sure the Background and Clouds are moving correctly.

Reminder: Save your work!

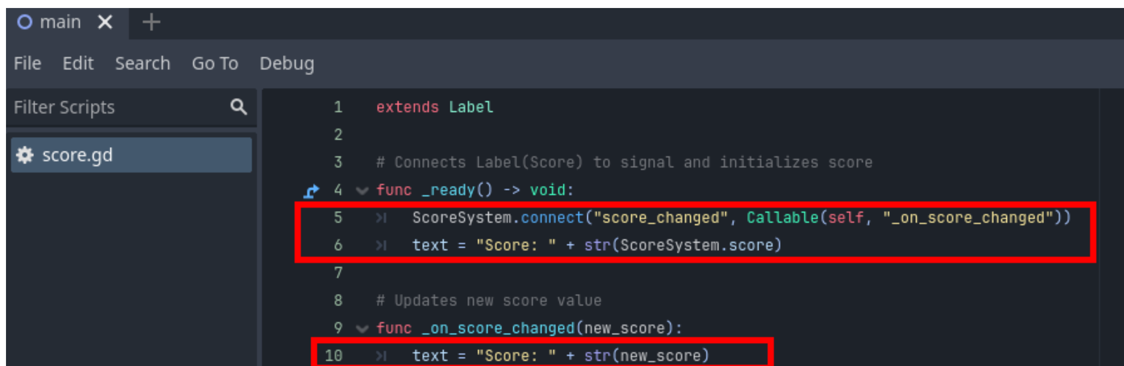
25 Add code to increase the score when the player passes a cloud.

In **FileSystem**, open the **score.gd** script. The **score.gd** script is connected to the Player and shows the text `Score: 0`. The score needs to be updated.

Inside the `_on_score_changed()` function, update the value of `text` by using string concatenation and the `str` method to convert the `new_score` number into text to be displayed on screen.

When the game runs, this function will be triggered by a signal whenever the score changes.

Uncomment the two lines in the `_ready()` method, and remove the `return` statement.

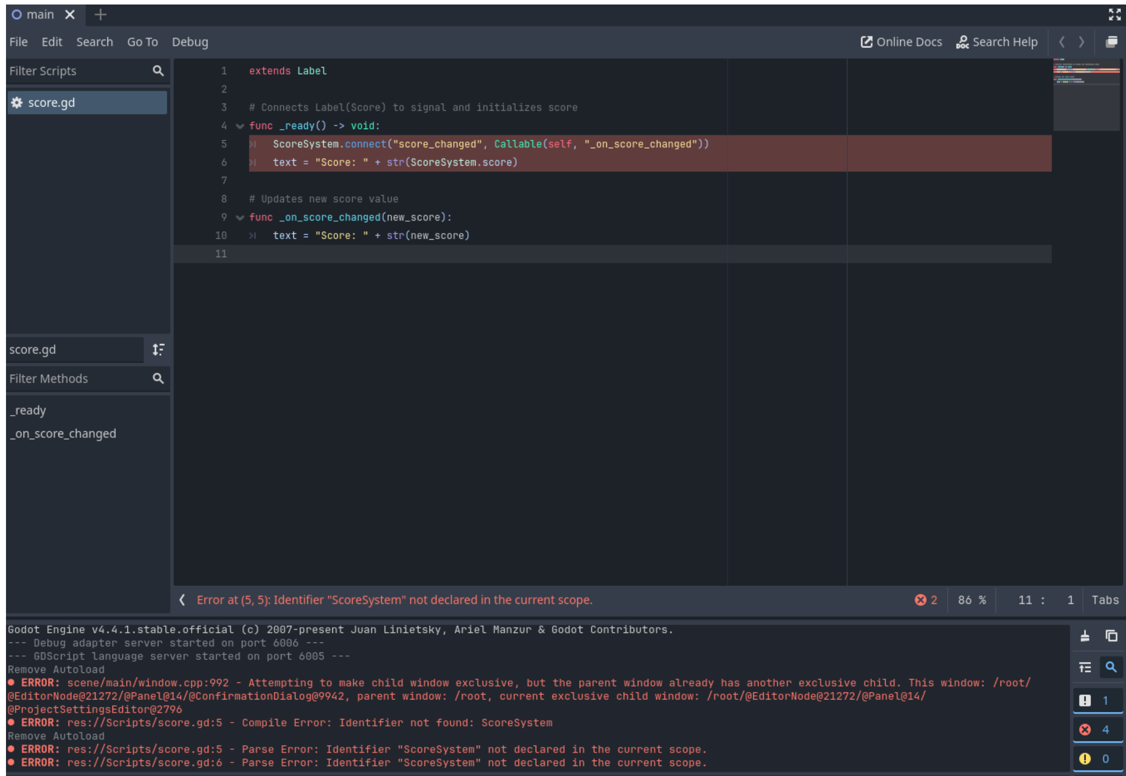


26

Playtest the game. What happens?

There's an **error**! This is because the script expects to see a **ScoreSystem** script to increase its score.

Add a **Global Script** to resolve the error**.**



Reminder:

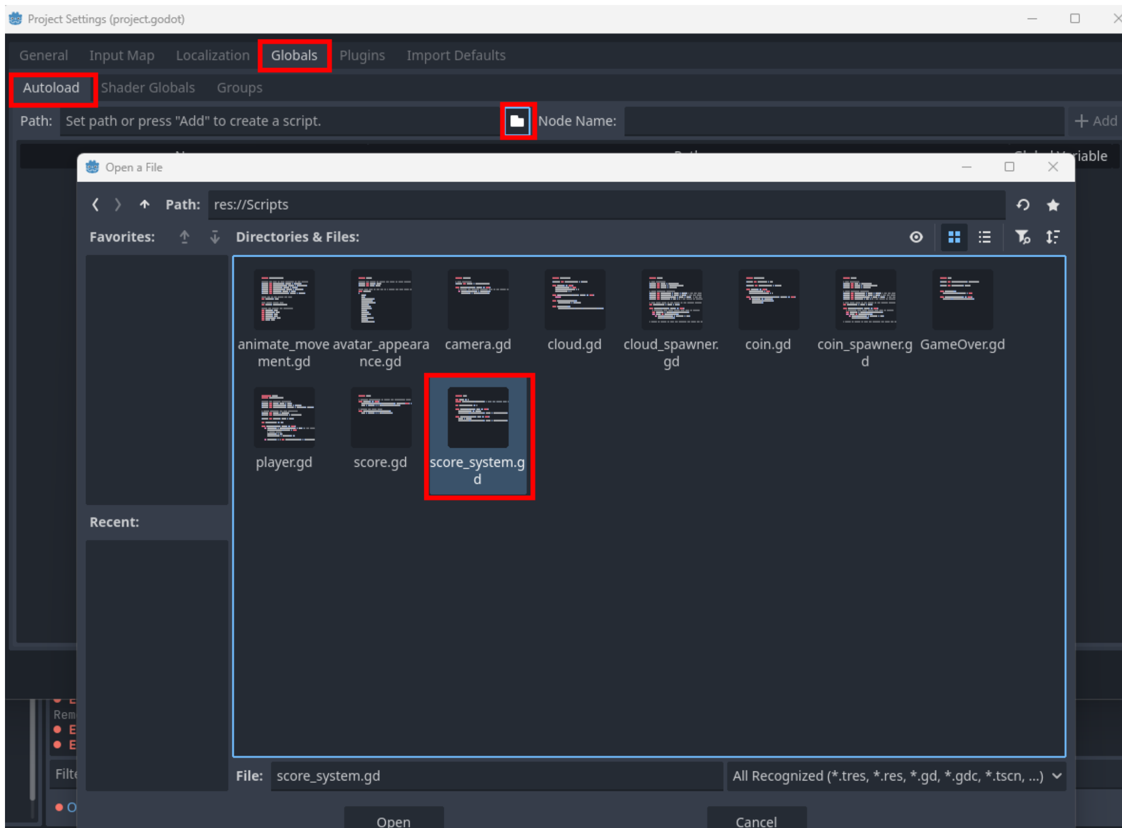
Global scripts live *outside* of individual scenes and can be accessed by all other scripts in the game, allowing them to keep track of the player's score, for example.

27

Navigate to **Project** > **Project Settings**.

Click **Globals** and ensure the **Autoload** section is open. Click the **folder** icon.

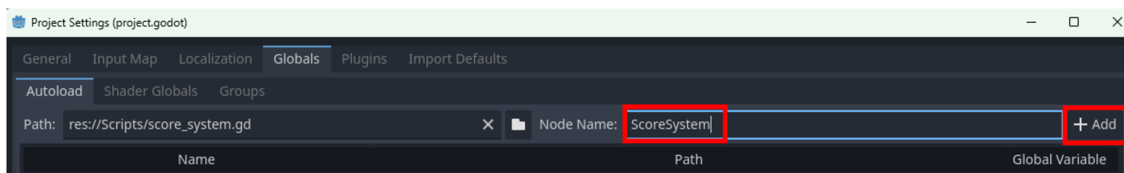
Open the **Scripts** folder and select **score_system.gd**.



28

Check that the Node Name is **ScoreSystem**.

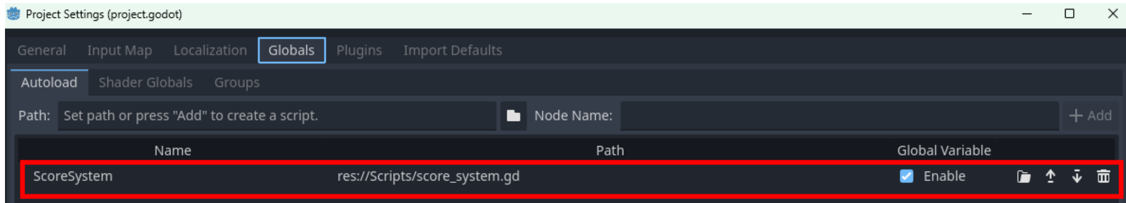
Click **+Add** on the right to add the **Global Script**.



29

The **Project Settings** window should match the image. If it doesn't, review the previous steps.

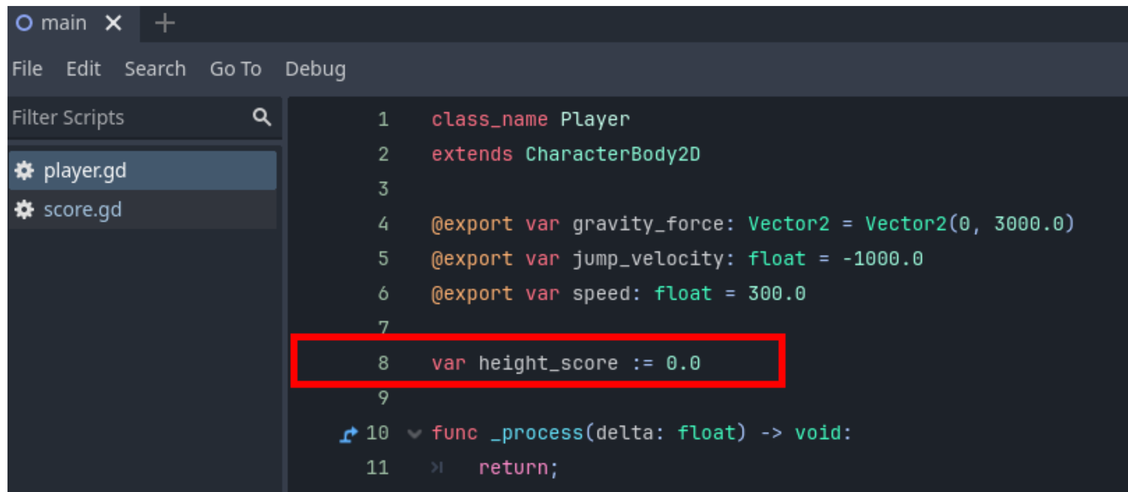
Click **Close** to exit the **Project Settings** window.



30

Navigate to the player.gd script.

Declare a `height_score` variable and use `:=` to assign it an initial value of 0. This should not be an export. This will be used to store the highest `position.y` the **Player** has reached.




```
1  class_name Player
2  extends CharacterBody2D
3
4  @export var gravity_force: Vector2 = Vector2(0, 3000.0)
5  @export var jump_velocity: float = -1000.0
6  @export var speed: float = 300.0
7
8  var height_score := 0.0
9
10 func _process(delta: float) -> void:
11     return;
```

Reminder:

"`:=`" is shorthand for "`: type = value`". What type of value does the `height_score` variable hold?

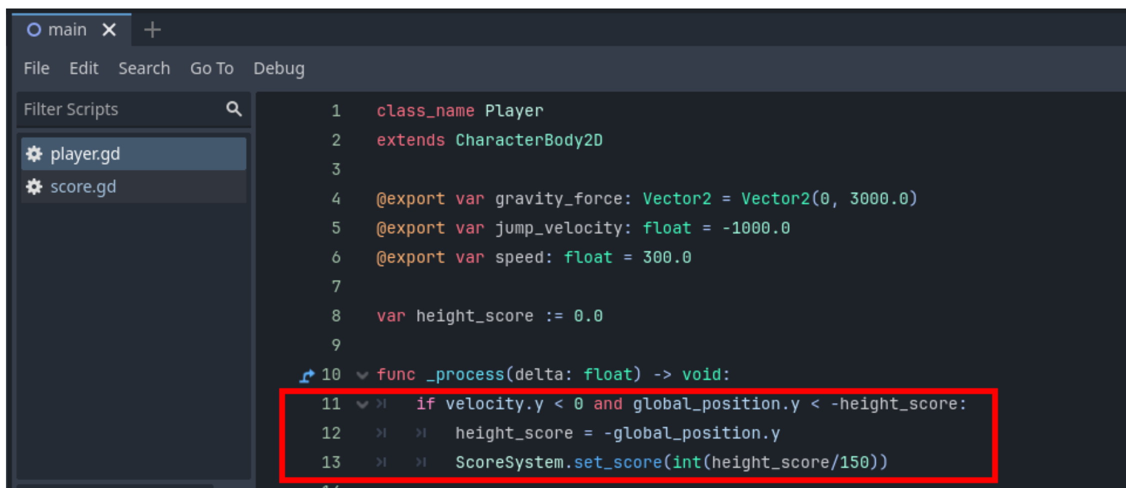
31

Next, add a process for calculating the player's current score relative to the player's position.

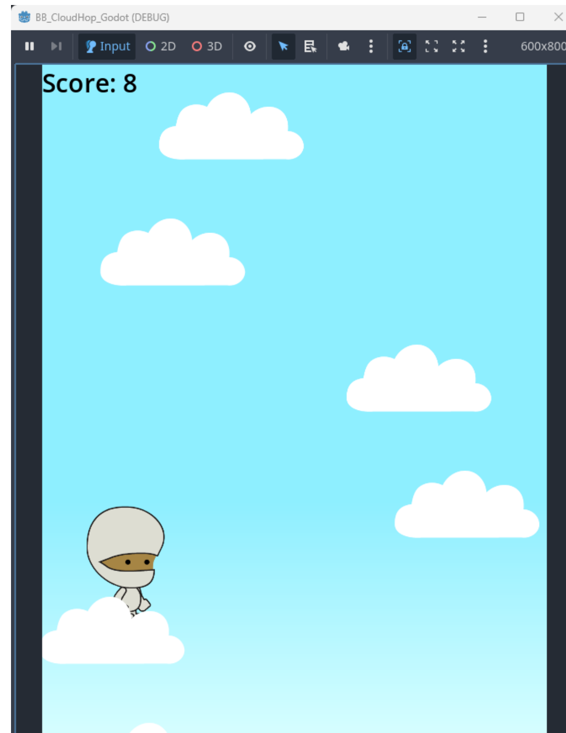
In the `_process()` method, remove the `return` statement.

Add an `if` statement that checks if the player's `velocity.y` is moving upwards `and` the `global_position.y` is less than `-height_score`.

If so, update the `height_score` to `-global_position.y`. Underneath, broadcast the updated score, by calling the `set_score` method of `ScoreSystem`. Pass `int(height_score/150)` as the parameter.



32 Play the game and watch as the score updates every time the player jumps on a cloud.



Close the **Game** window.

33 Code what happens when the game ends!

Open the **player.gd** script.

Declare 2 new `export` variables: `camera`, of type `Camera2D`, and `game_over_canvas`, of type `CanvasLayer`.

```
8
9    # export variables for node references
10   @export var camera : Camera2D
11   @export var game_over_canvas : CanvasLayer
12
```

34 Update the `_process()` function to check the position of the player relative to the camera.

Add an `if` statement that checks if the player's `position.y` is greater than the camera's `position.y + 548`.

```
13 func _process(delta: float) -> void:
14 	if velocity.y < 0 and global_position.y < -height_score:
15 		height_score = -global_position.y
16 		ScoreSystem.set_score(int(height_score/150))
17 		
18 	if position.y > camera.position.y + 548: ## The player falls beyond the camera (downwards)
```

Pro Tip:

Make sure **548** is the number used, otherwise the game won't end when the player falls off the screen.

35 If the player's `position.y` is greater than the camera's `position.y` plus a certain value **(**`+548`**)**, that means the player has fallen off the screen.

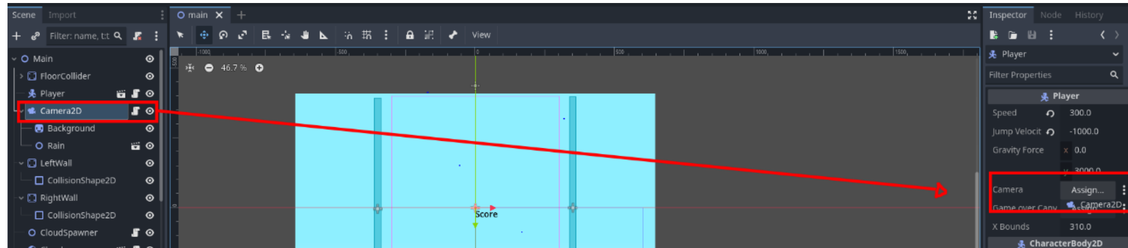
When the player falls off the screen, enable the `game_over_canvas` and set the player's **velocity y** to `0`. Set `gravity_force` to `Vector2(0, 0)`.

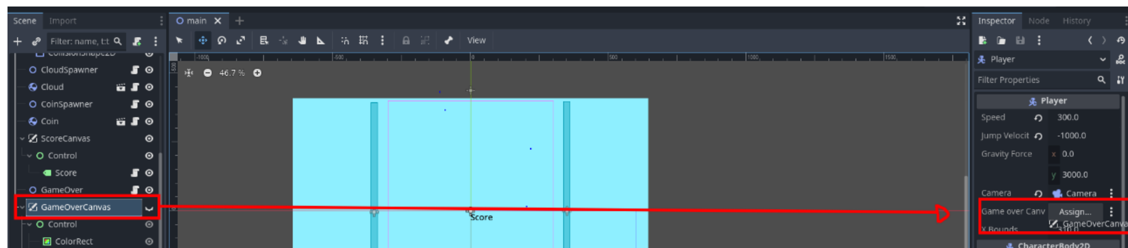
```
13 func _process(delta: float) -> void:
14 	if velocity.y < 0 and global_position.y < -height_score:
15 		height_score = -global_position.y
16 		ScoreSystem.set_score(int(height_score/150))
17 		
18 	if position.y > camera.position.y + 548: ## The player falls beyond the camera (downwards)
19 		game_over_canvas.visible = true
20 		# freeze player
21 		velocity.y = 0.0
22 		gravity_force = Vector2(0, 0)
```

36 Navigate to the **Player** node located in **Main.**

In the Player's **Inspector**, assign the **Camera** and **Game over Canvas** export variables.

Drag and drop the **Camera2D** and **GameOverCanvas** nodes into the **Assign..** space in the **Inspector**.



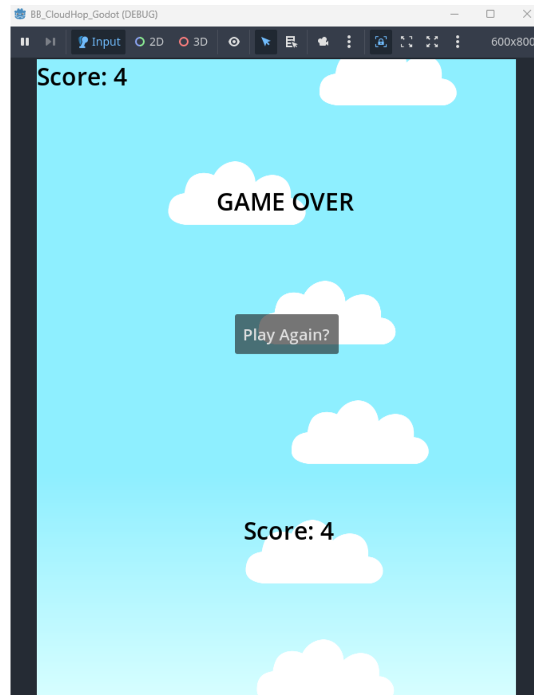


37

Playtest the game!

Observe the Game Over canvas appearing when the player falls below the bottom of the screen.

Close the **Game** window.



Pause for **Sensei Stop #4**!

Before submitting, check in with a Code Sensei to make sure the player **Scoring** is happening when passing a cloud and **Game Over** is reachable after climbing a few clouds and then falling, then reflect on the following:

- ☐ What did you learn about CharacterBody2D?
- ☐ What did you enjoy most when creating this project?
- ☐ What was something you found difficult and why?

Reminder: Save your work!