CODE NINJAS®

Bronze Belt Ninja Guide

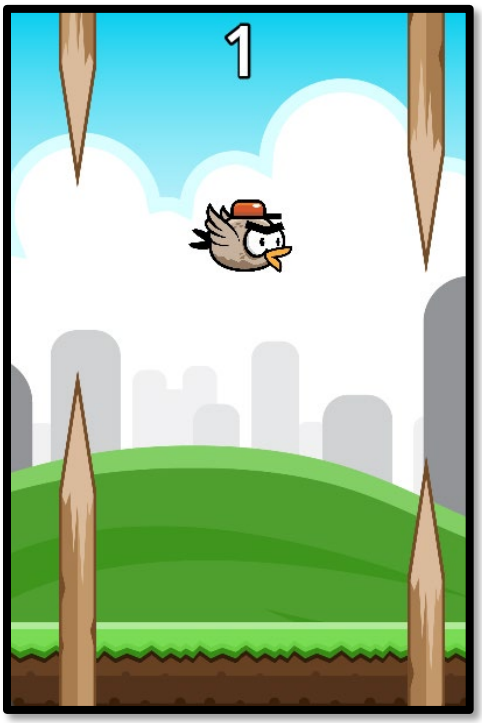
BB Activity 04: Sketch Head

CharacterBody vs RigidBody

Godot offers many different objects (nodes) with varying uses, some of which create better Player characters than others. So far, both **CharacterBody** and **RigidBody** have been introduced, but when should these different objects be used?

A **CharacterBody** node is controlled via code. It can detect collisions with other objects when moving, but is not affected by engine physics properties, like gravity or friction. In-game physics can still be applied through code to mimic this behavior, giving developers more precise control over movement.

The **CharacterBody** class contains built-in methods to simplify collision responses, which makes this node class extremely useful as a user-controlled character in platform (Scavenger Hunt) or top-down (MakeCode-style maze) games.

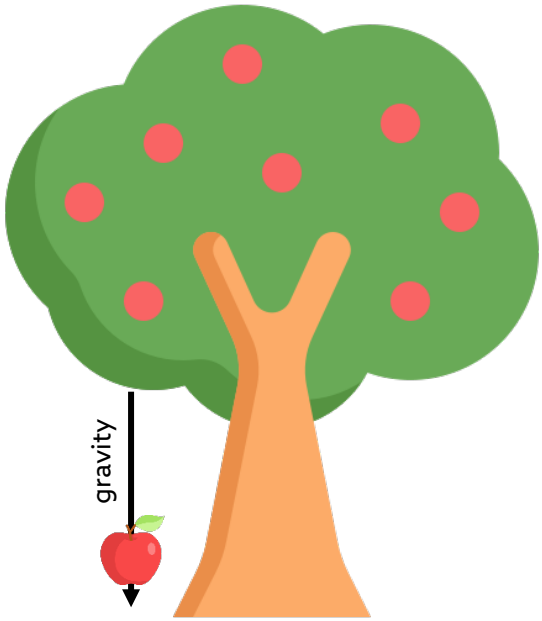


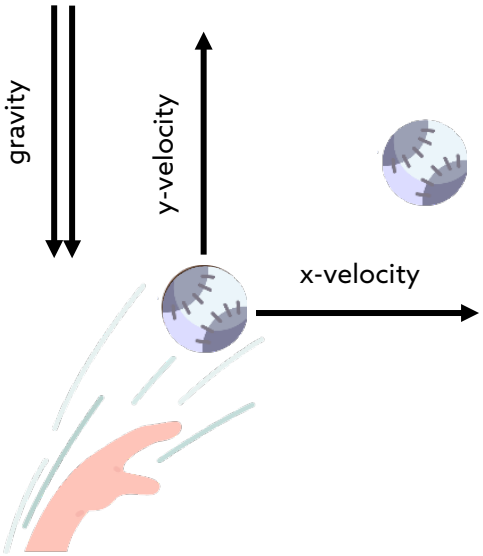
A **RigidBody** node is directly controlled by the physics engine to simulate in-game physics. **RigidBodies** are more commonly used when creating falling objects or projectiles but can be used to create user-controlled characters when the character is controlled through forces, like velocity or acceleration. **RigidBodies** do not make ideal code-controlled characters, but are still useful in games, such as Meany Bird, where realistic physics and collision responses are required.

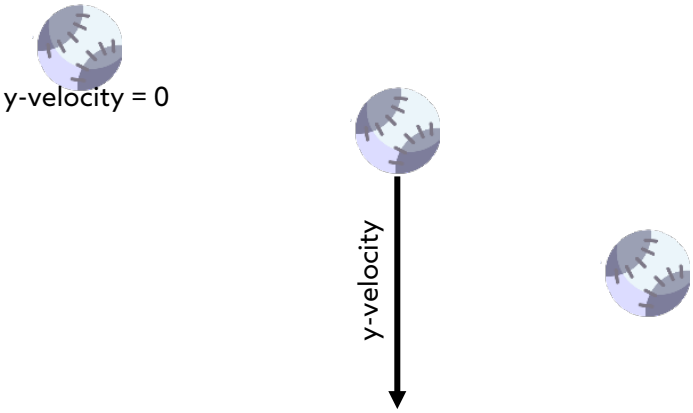
Physics and Forces

Gravity is the force that pulls objects towards earth. It is the reason apples fall from trees and tennis balls return to the ground after being thrown up into the air. The **force of gravity** is **constant** and does not change.

When throwing a ball in the air or jumping on a trampoline, instantaneous force is applied to the object, causing movement in the form of **y-velocity**. The y-velocity decreases as gravity pulls the object back towards earth.



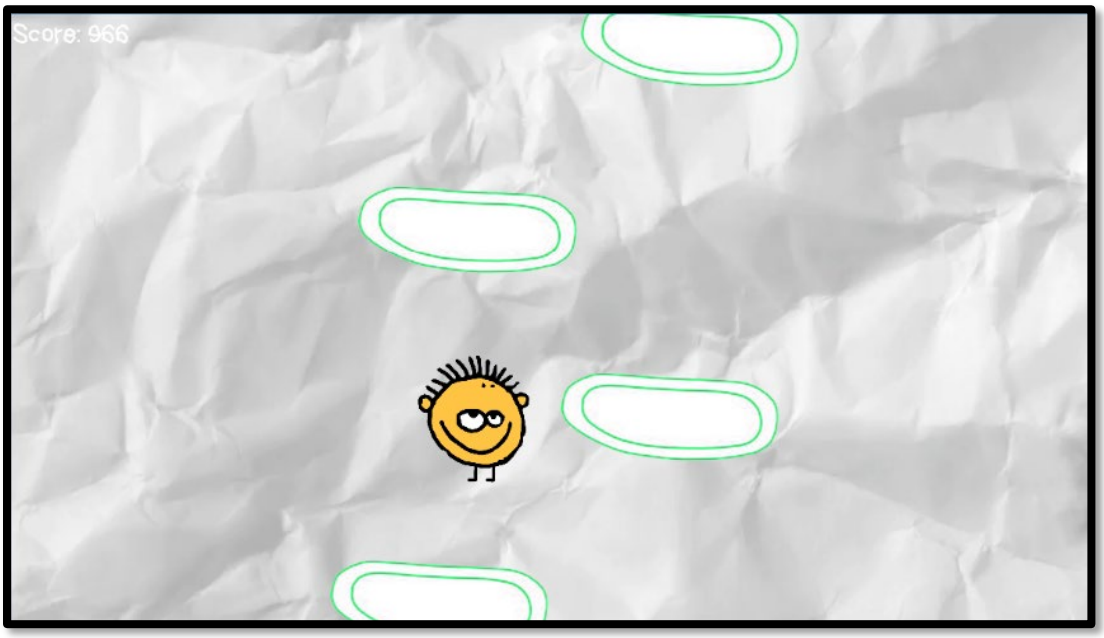




Activity 04: Sketch Head

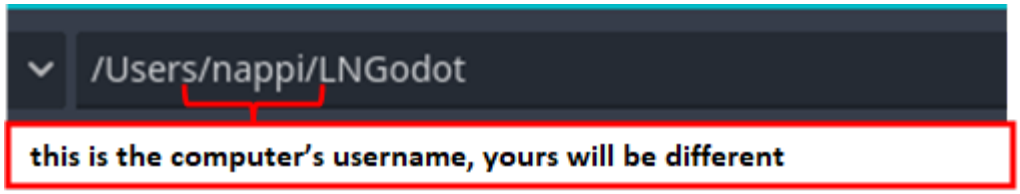
In this project, you will create a platformer game where DoodleHead must bounce off platforms to reach new heights. This project will explore in-game physics with a CharacterBody, object collisions, character movement and camera tracking.

By the end of this activity, you will have explored the relationship between gravity and y-velocity and built-in collision methods in the CharacterBody class.

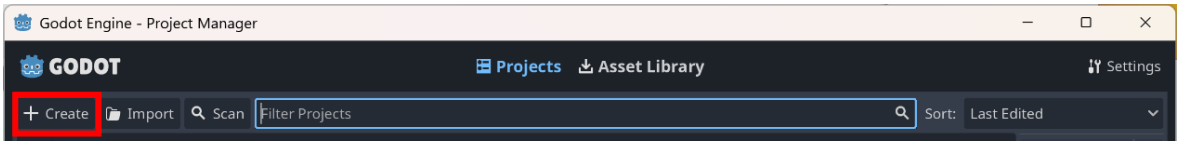


1 Remember all projects will be stored in a path like:
/Users/ [MyComputerUsername]/ [MyInitials]Godot

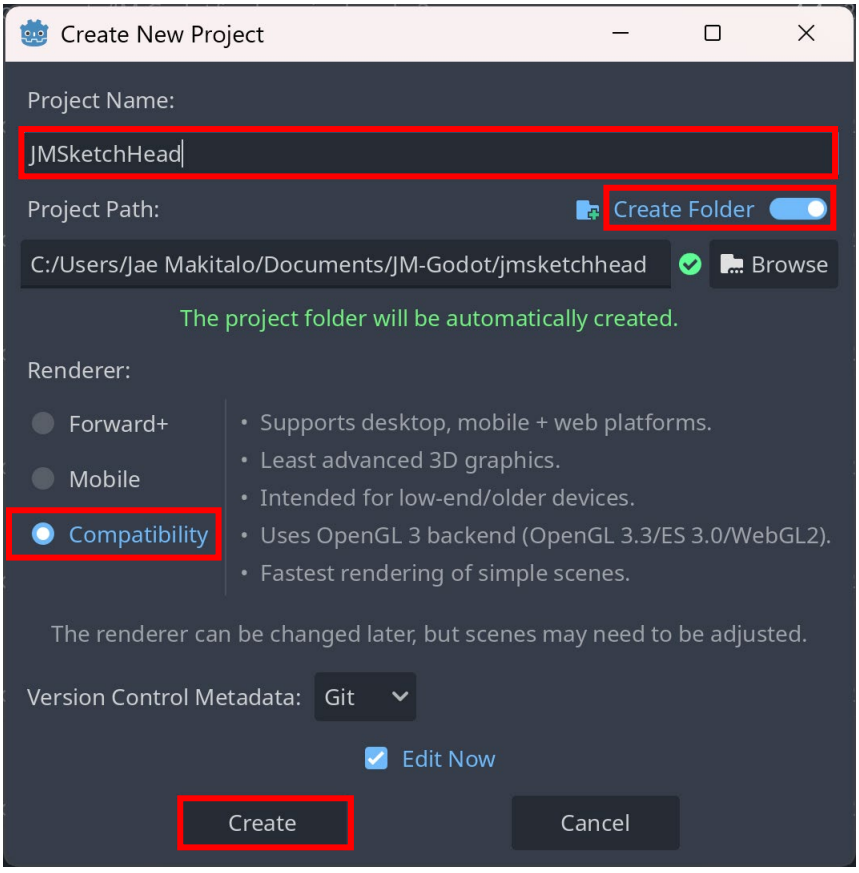
Don't worry if your path looks slightly different from the picture shown! All computers have their own username.



2 In the Godot Project Manager, create a new project.

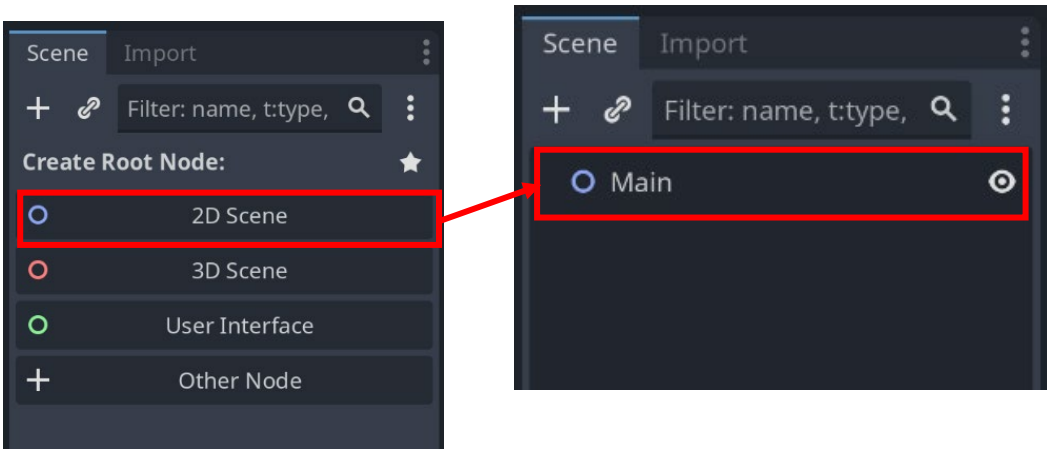


3 Name the project **[MyInitials]SketchHead** and adjust the project path so the project is being saved in your folder. Make sure **Create Folder** is toggled on, set the renderer to **Compatibility**, then click **Create**.



4 A **Main root node** and **main scene** need to be set for the project. This project is set in a **2D environment** so all nodes will be **2D**.

Select the **2D Scene** as the **root node** for this project. Rename the **Node2D** to **Main**.

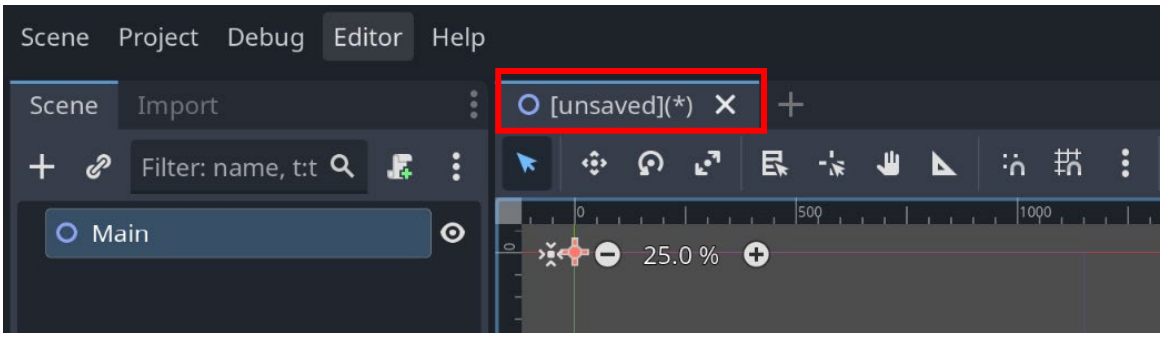


Reminder:

Rename a node by double-clicking on it.

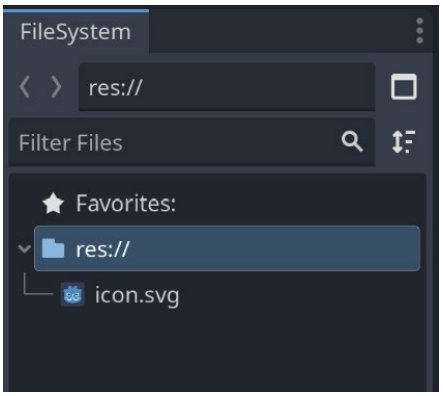
5 The **main scene** needs to be saved.

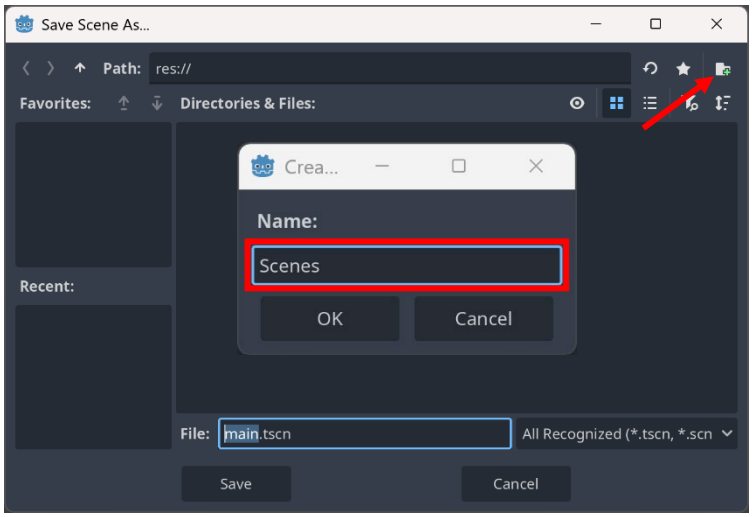
On the keyboard, press **CTRL + S** on to save the scene.



6 This project will contain multiple scenes, so use a **Scenes folder** to stay organized.

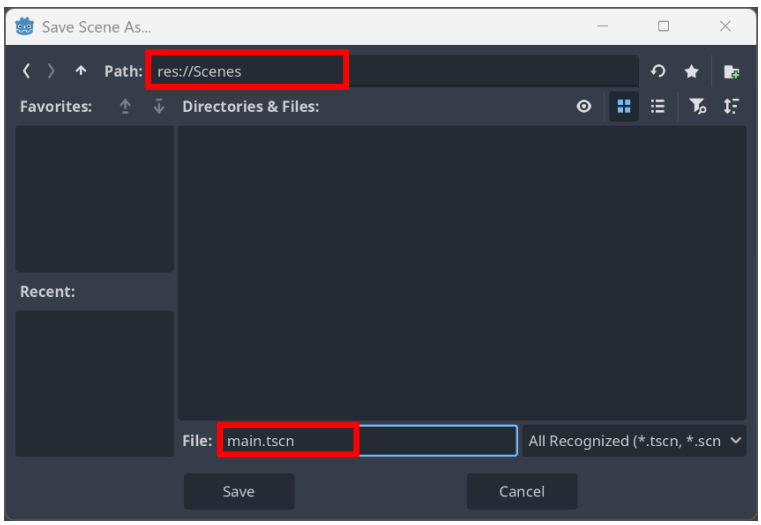
Currently, there are no folders in the **FileSystem**. Click the **new folder icon,** name the new folder **Scenes**, then click **Ok**.

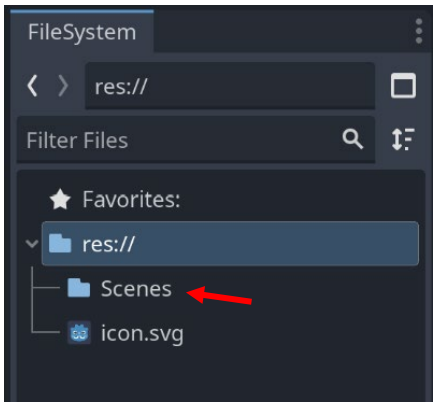




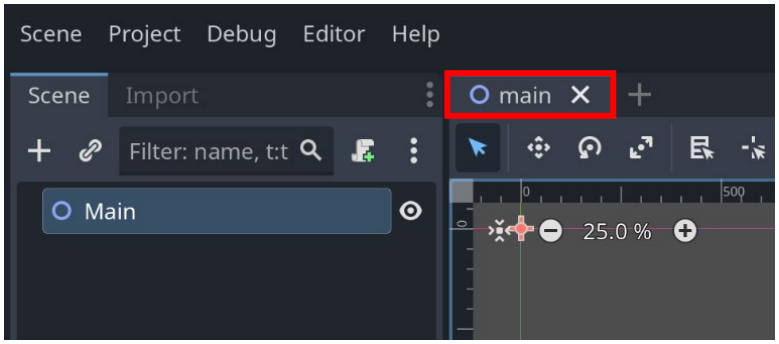
7 The **path** will update from res:// to **res://Scenes** and the Scenes folder will appear in **FileSystem**.

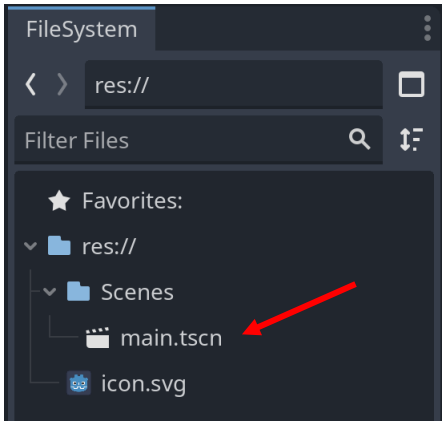
Check that the file is called **main.tscn** and click **Save**.





8 The main scene has been saved. Click the **>** arrow beside the **Scenes** folder to see **main.tscn** inside.

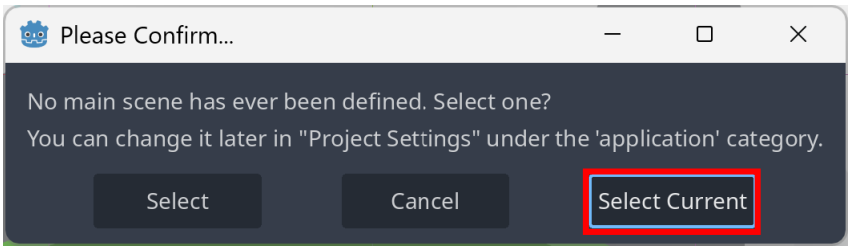




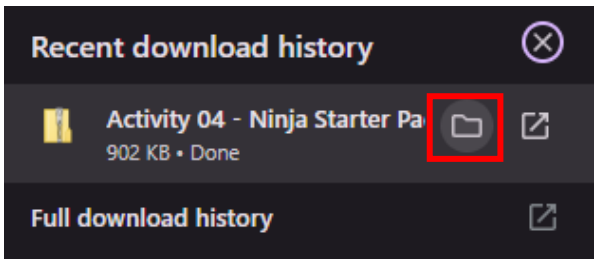
9 Click the **play** button to **playtest** the project.

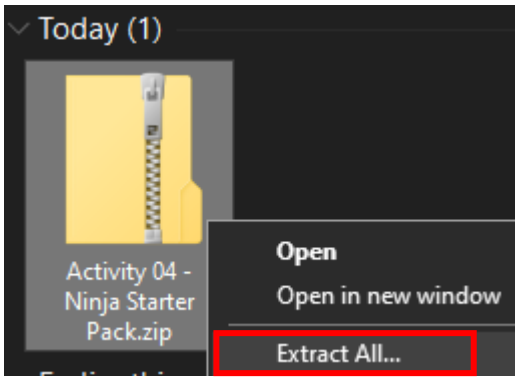
A window will pop up asking to define the **main scene**. Click **Select Current**.

Close the playtest window.

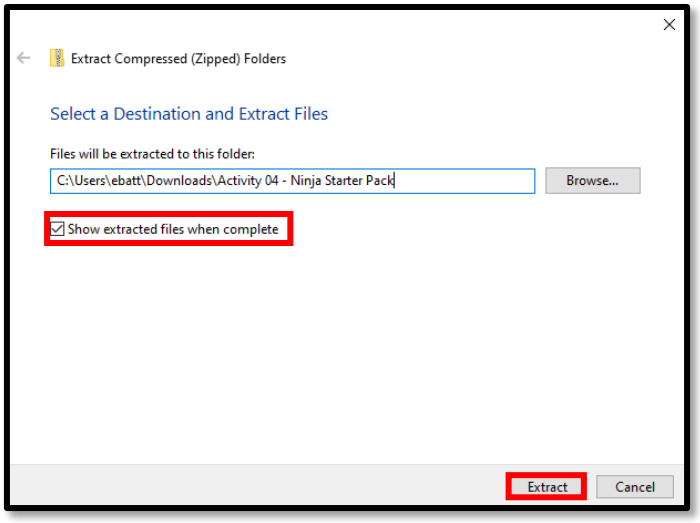


10 Download **BB Activity 04 – Ninja Starter Pack.zip**. Open the file in file explorer and right-click on the **zip** file to select Extract All.



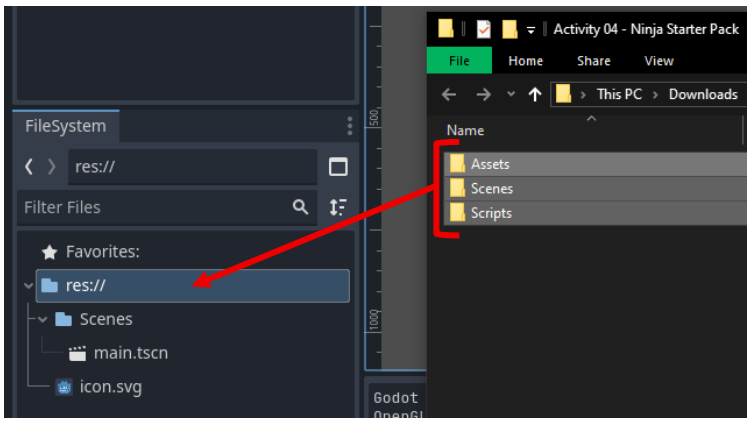


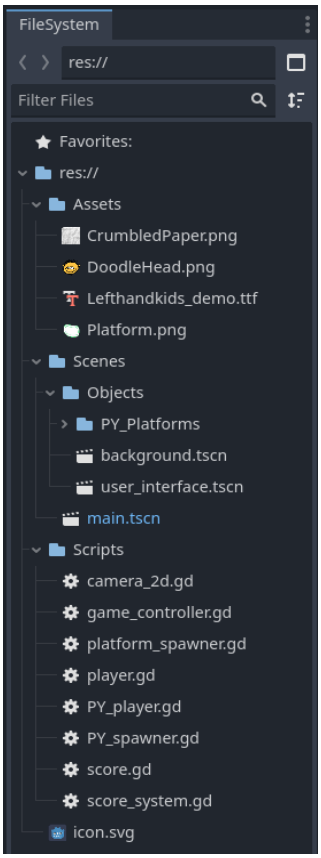
11 Make sure the **Show extracted files when complete** is checked, then click **Extract.**



12 In Godot, select the **res://** folder in **FileSystem**.

In the **File Explorer**, press **CTRL + A** to select all the folders and drag them into **res://** as seen in the image. Check that the following files and folders are present.





Ignore the console errors; these bugs will be fixed at a later point.

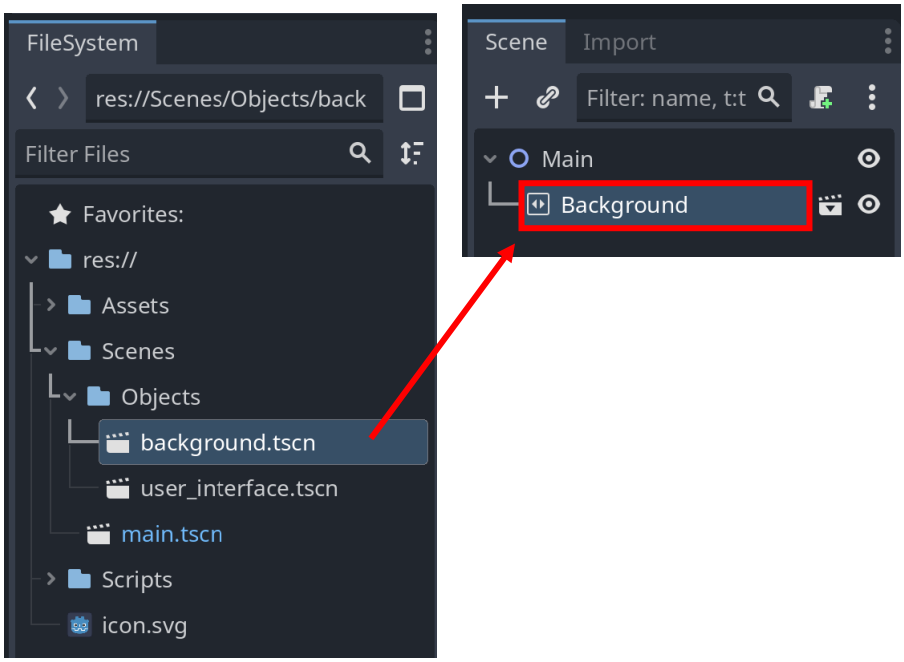
Pause for **Sensei Stop #1**!

Before continuing, check with a Code Sensei and make sure the **main scene** is set up and all files are imported correctly.

Reminder: Save your work!

13 This project requires a scrolling background, which has been set up and included in the starter code.

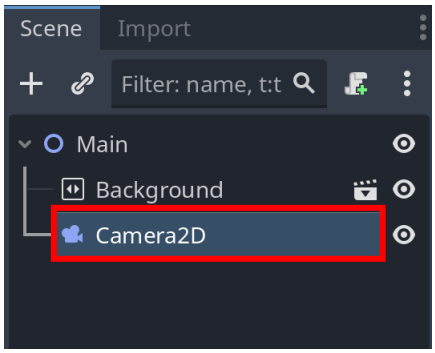
In **FileSystem**, find **background.tscn** in the **Objects** folder. Drag the scene into the node hierarchy to make **Background** a child of **Main**.

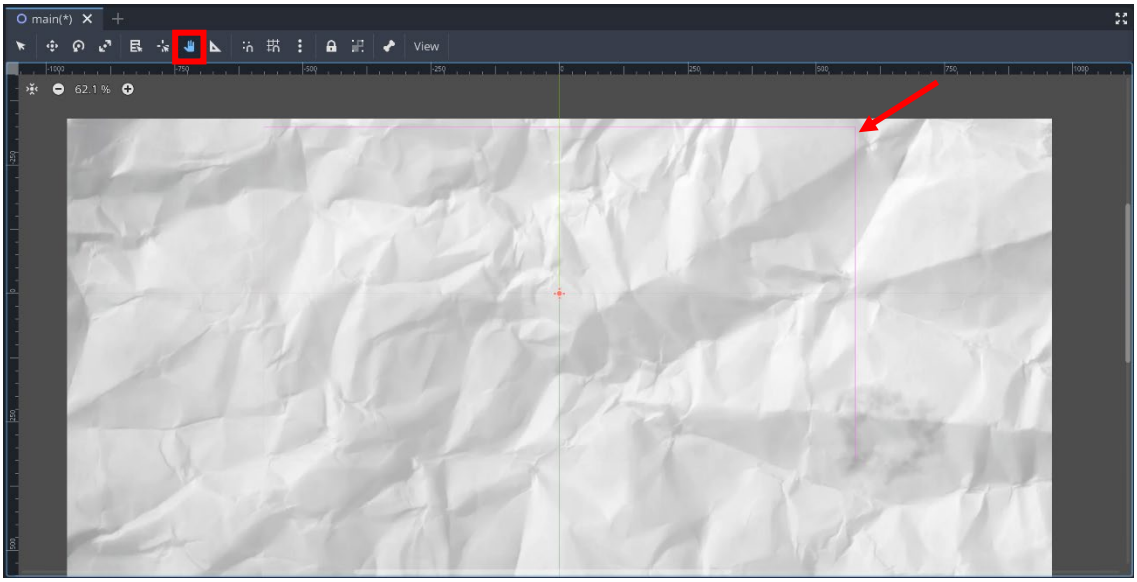


14 Add a **Camera2D** as a child node to **Main**.

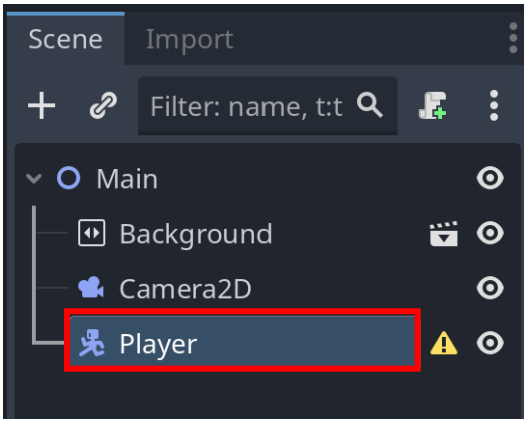
Zoom in as needed (using the mouse wheel) so the outline for the camera view can be seen.

Using **Pan Mode** (), adjust the view in the game window so most of the background image and the outline showing the camera view can be seen.





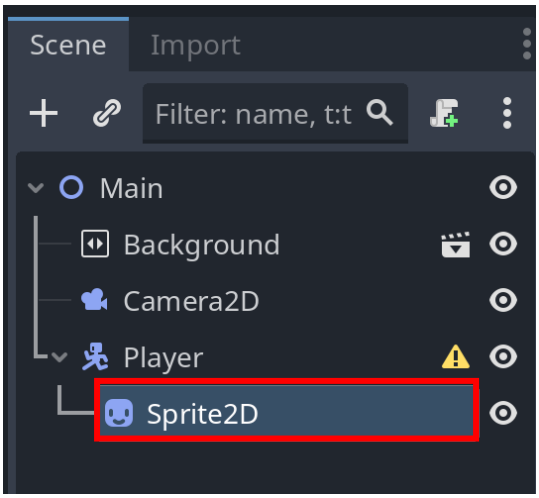
15 Add a **CharacterBody2D** as a child to **Main** and rename the node **Player**.



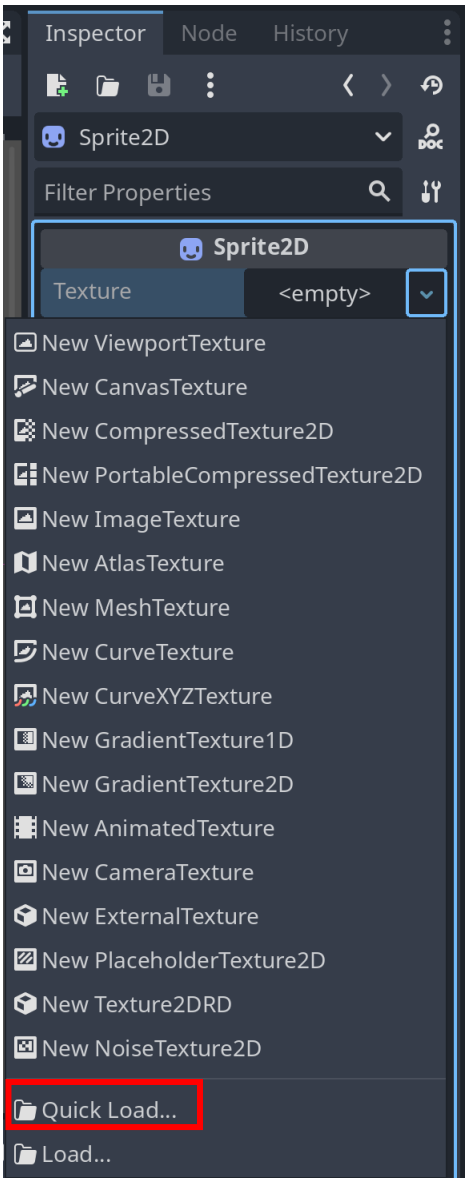
New Concept: CharacterBody

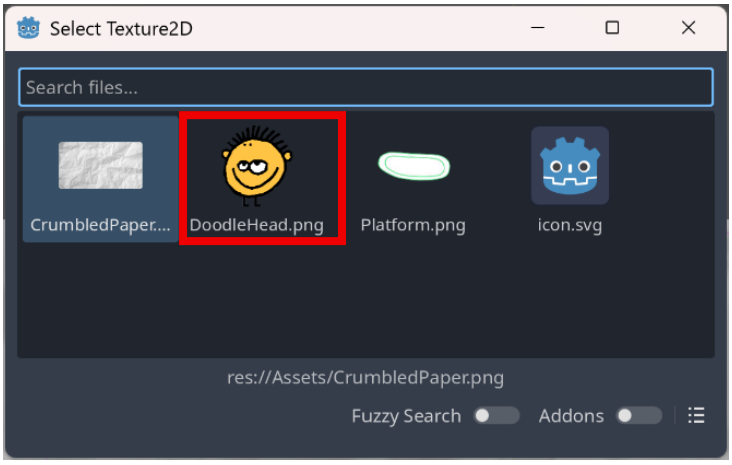
A **CharacterBody** is a specialized physics body that gives the developer direct control over movement mechanics. Unlike a **RigidBody** (used in Meany Bird), which is affected by gravity and friction, a **CharacterBody** is not affected by these forces so they must be implemented in code.

16 The **Player** needs a shape and a texture. Add a **Sprite2D** as a child to **Player**.



17 In the **Inspector** for **Sprite2D**, select the **drop-down arrow** next to Texture <empty>. Use **Quick Load** to assign the **DoodleHead.png** texture.

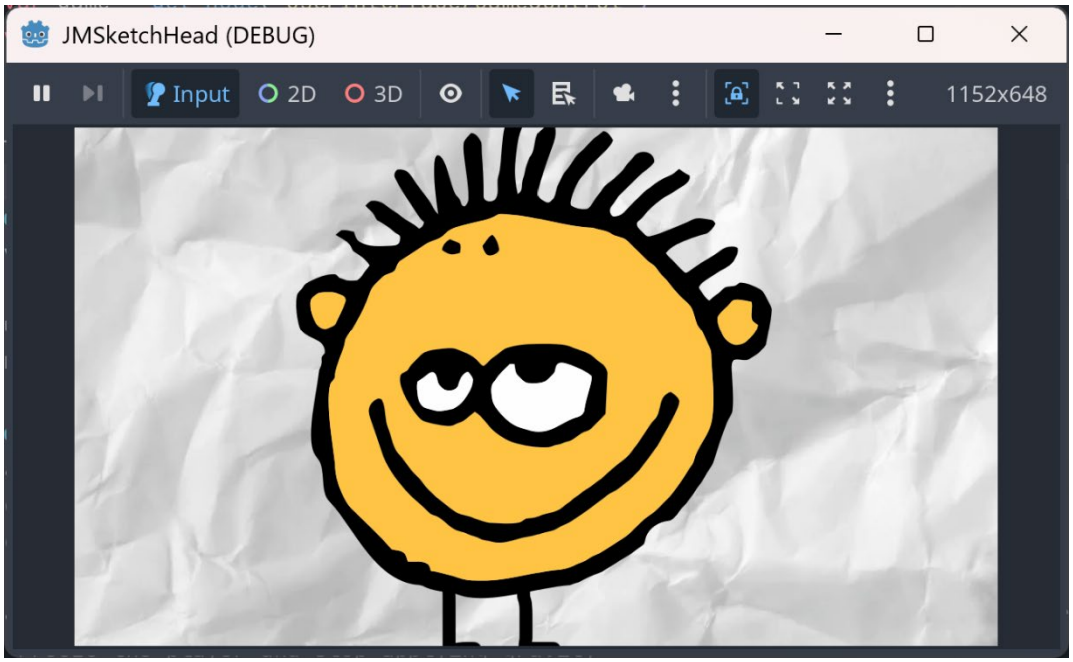




18 Playtest the project. What do you notice?

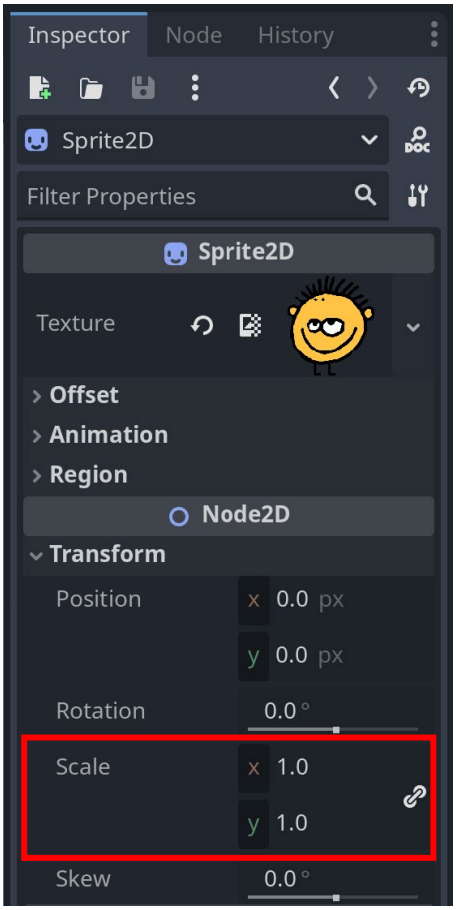
DoodleHead takes up the whole screen!

Close the playtest window.



19 In the **Inspector** for **Sprite2D**, adjust the **Scale** of DoodleHead to better fit the camera view. DoodleHead should have plenty of space to move and collide with obstacles in the viewport window.

Tinker with values between **0.2 – 0.5**.

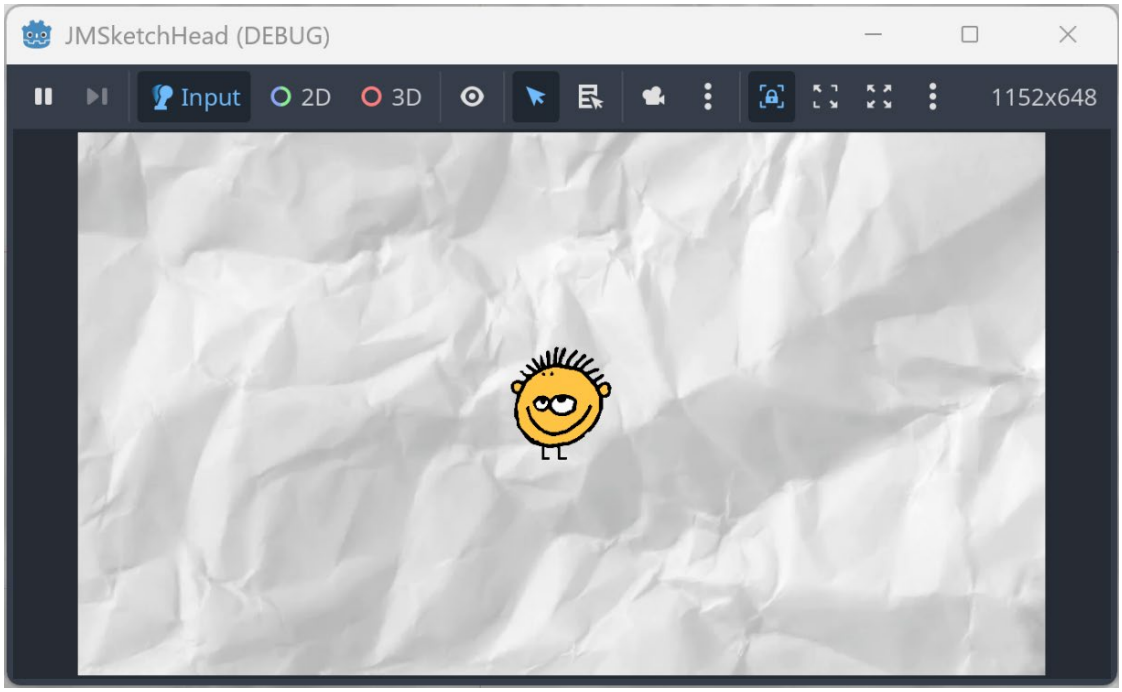


20 Playtest the project.

How does DoodleHead's size compare to the background image and the camera view? Does it need to be adjusted further?

Close the playtest window.

Note: *This playtest window shows a DoodleHead scaled at 0.2.*

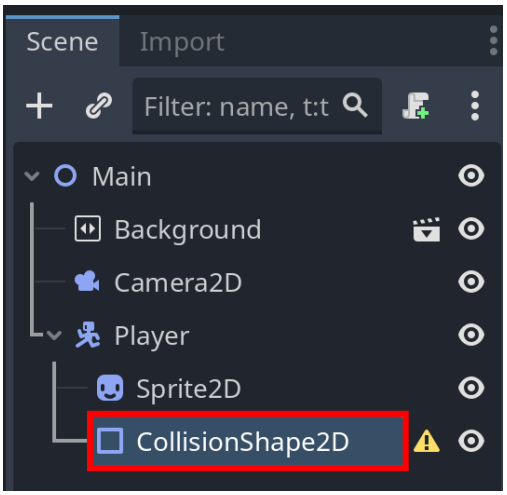


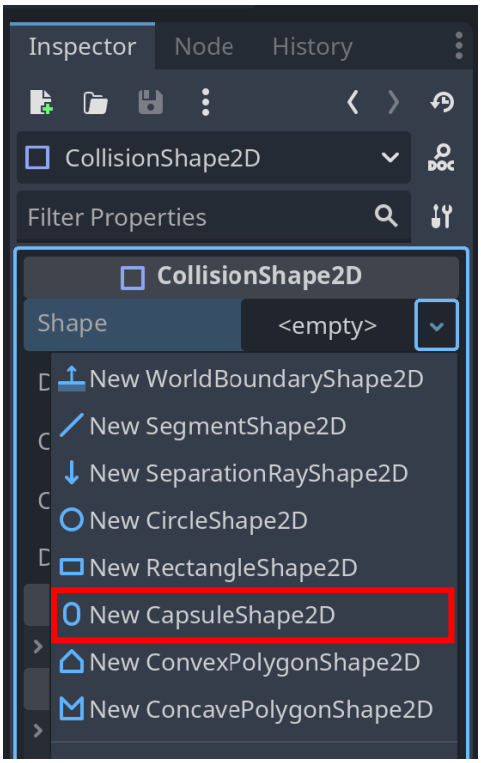
21

The Player still needs a shape.

Add a **CollisionShape2D** as a child node to **Player**. The **CollisionShape2D** will monitor collisions for the Player.

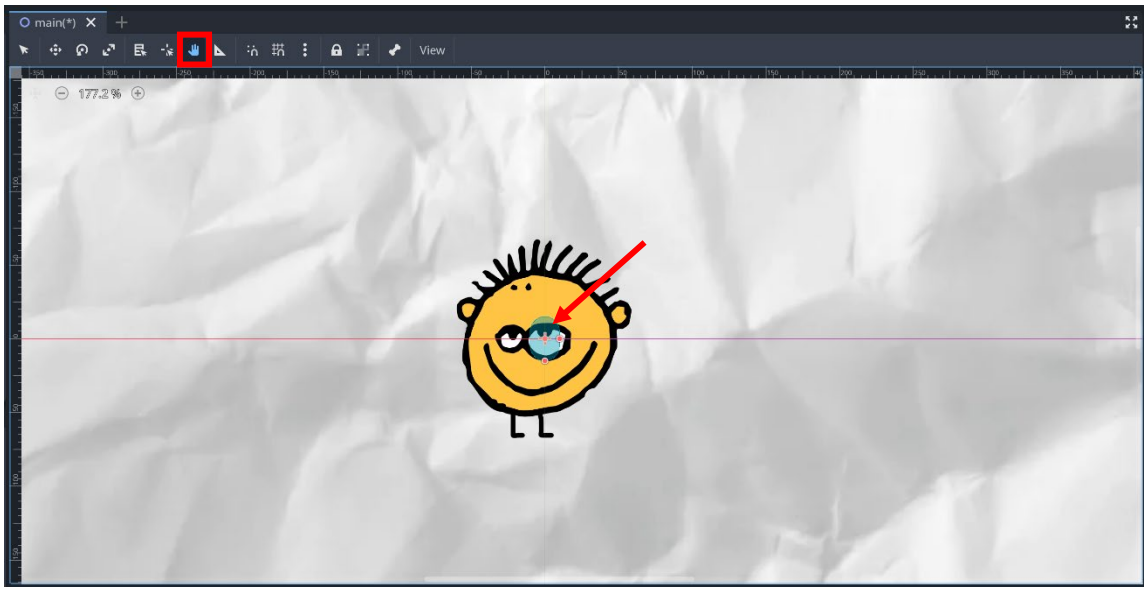
In the **Inspector** for **CollisionShape2D** next to Shape <empty>, select the **drop-down arrow** to set the shape to a **New CapsuleShape2D**.



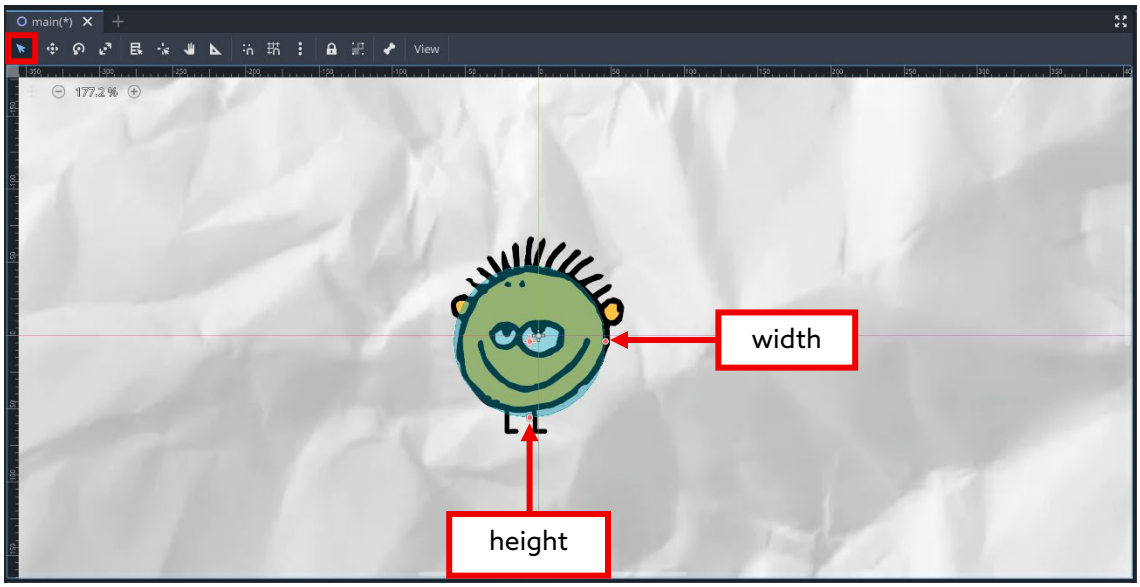


22

With **CollisionShape2D** selected in the node hierarchy use **Pan Mode** to center DoodleHead and zoom in on the collision shape in the game window.



Using **Select mode** (), resize the collision shape to cover **DoodleHead's** face. Choose whether to resize the shape so it's slightly *larger* than **DoodleHead's** face, or slightly *smaller*. A larger collision shape will make the game easier to play.



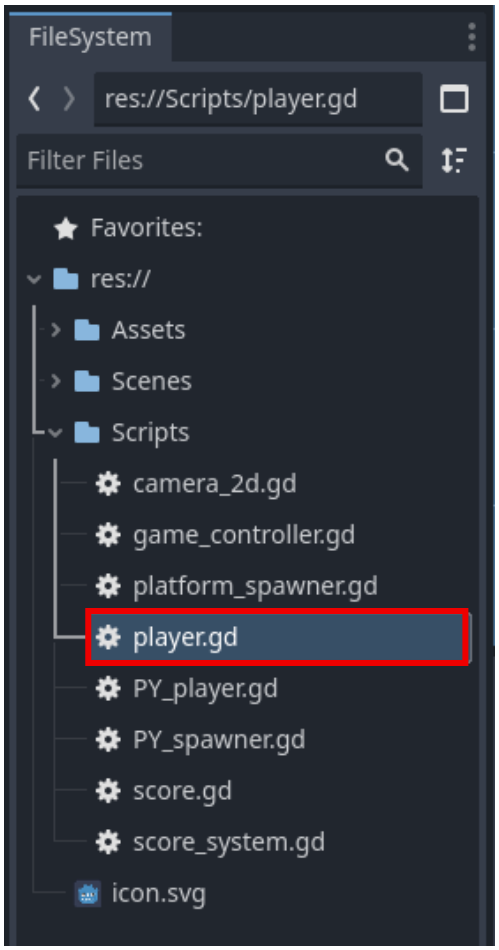
The red dot at the **bottom** adjusts the **height** of the shape and the red dot on the **right** adjusts the **width**.

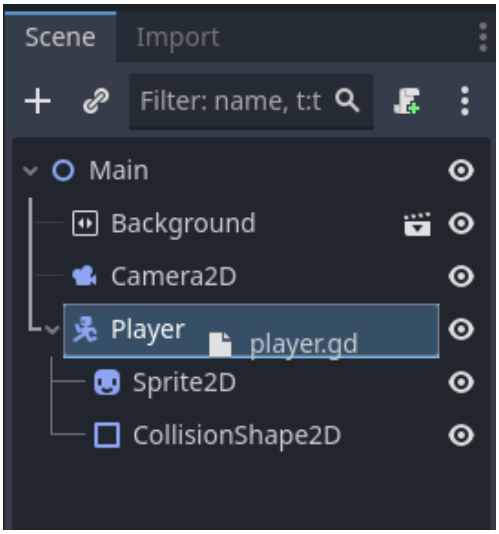
Pause for **Sensei Stop #2**!

Check to make sure DoodleHead is properly **scaled down** and has a fitting **collision** shape!

Reminder: Save your work!

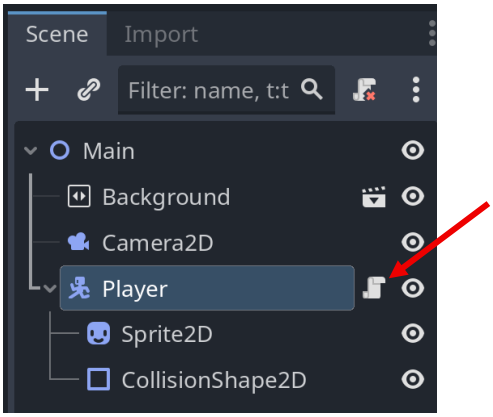
23 In **FileSystem**, find the **player.gd** script inside the **Scripts** folder. Attach the script to the **Player** node.





24

Click on the **script icon** beside **Player** to open the player.gd script.



25

Create a new **@export** variable for **x_velocity** of **type "float"**. This variable will set the Player's side to side movement speed.

```
1  extends CharacterBody2D
2
3  # --------------------
4  # TODO STEP 25
5  # Create the physics variables
6  # --------------------
7
8
```

New Concept: float

float is a built-in type used for **floating-point** numbers, which are numbers that include decimal points, unlike an integer.

26

Check the code for the `x_velocity` variable.

Make sure a **colon (:)** is added after the variable name, before the variable type is declared.

```
1   extends CharacterBody2D
2
3   # --------------------
4   # TODO STEP 25
5   # Create the physics variables
6   # --------------------
7   @export var x_velocity: float
8
```

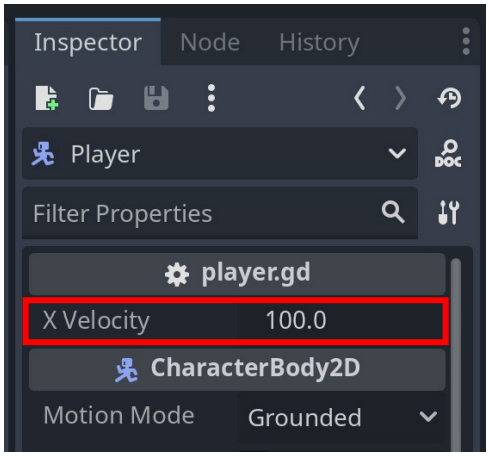
27

A **default value** can be assigned to an export variable in the script and later updated in the **Inspector**.

In the script, set `x_velocity` to `100` using the **assignment operator (=)**. This value can be updated later in the **Inspector** to customize the project.

```
1   extends CharacterBody2D
2
3   # --------------------
4   # TODO STEP 25
5   # Create the physics variables
6   # --------------------
7   @export var x_velocity: float = 100
8
```

assignment operator



28 Review: Functions in GDScript

GDScript uses a **colon** and **indentation** for the function body, unlike **JavaScript** which uses **{ } (curly brackets)**.

```
2   function function_name(parameters: any) {
3       //function body
4   }
```

JavaScript

```
1   func function_body(parameters: any):
2       #function body
3
```

GDScript

29

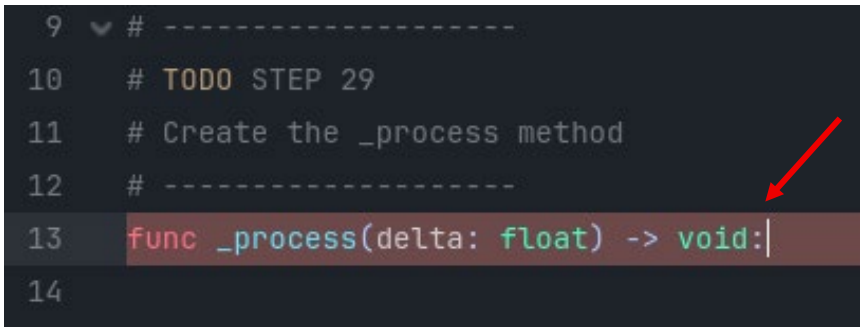
Use the keyword **func** to define a new method, followed by the method name, **_process**. The **_process** method takes one parameter: **delta**.

The **-> void** shows what the method returns. This method is **void**, meaning it does not return anything.

Make sure the **: (colon)** is added after the method definition (the method's name, parameters and return type).

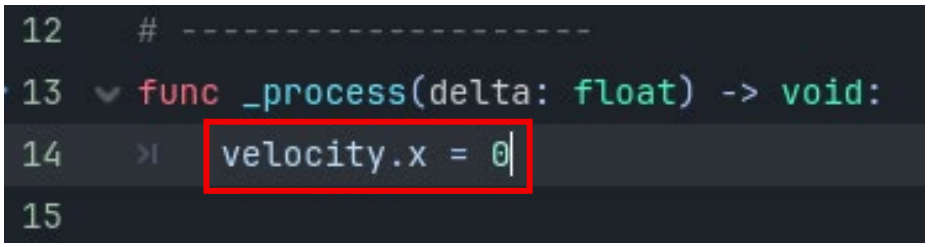
Currently, the editor will show an error. This is because the editor is expecting the **indentation** and method **body**, which has not been added yet.

> **`_process()`:** part of the Node class, this method is called during the processing step of the main loop. Processing happens at every frame and as fast as possible.
>
> **Parameters:**
>
> 1. **`delta`** ***(float):*** the time since the previous frame, in seconds. Since processing happens as fast as possible, delta is not constant.
>
> **Returns *(void):*** this method is void, it does not return anything.



30

Inside the **_process** method, set the **velocity.x** to **0**.



31 Review: If-statements in GDScript

If-statements in GDScript are formatted similarly to functions, with a **colon** and **indentation** for the statement body. This is unlike JavaScript, which relies on curly brackets**.**

In GDScript, **parentheses** can be used but are not needed in an if-statement.

```
7       if (condition) {
8           // then
9       }
```

JavaScript

```
11      if condition:
12          #then
```

GDScript

32 With the help of if-statements, the **_process()** method will be coded to:

- Check if the left or right directions buttons are pressed
- Set the Player's x velocity according to the direction button input
- Set the Player's x velocity to 0 when the direction buttons are not being pressed.

Using the keyword **Input**, code the first **if-statement** to check if the **ui_left** button is pressed. The **ui_left** button is the left direction button on the computer keyboard.

```
12  # --------------------
13  func _process(delta: float) -> void:
14      velocity.x = 0
15
16      if Input.is_action_pressed("ui_left"):
17
18
```

```
21    if (controller.left.isPressed()) {
22
23    }
```

Pro Tip:

The key word **Input** accesses the Input class which handles key presses, mouse buttons and movement, gamepads, and input actions. Actions and their events can be set in the **Input Map** tab in the project settings.

33 Inside the if-statement, set the Player's **velocity.x** to **-** (negative) **x_velocity**.

Now, when the left button is pressed, the Player is moved to the **left**.

Don't forget to **indent** the code inside the if-statement.

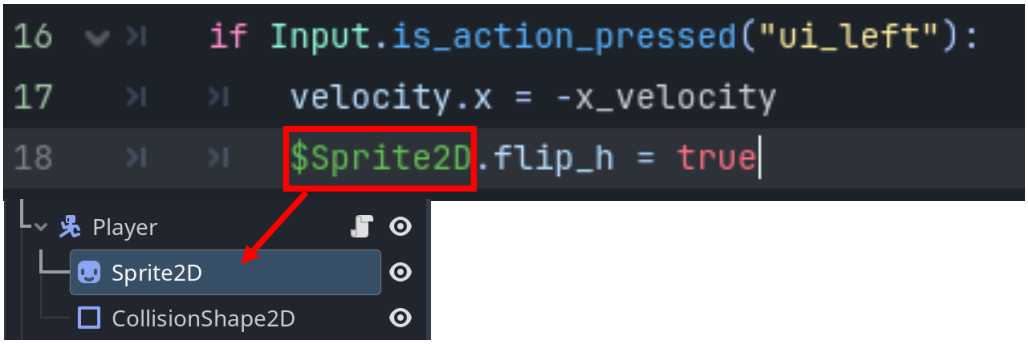
> **`Input.is_action_pressed()`:** returns true or false depending on if the action event is being pressed in the current frame.
>
> **Parameters:**
>
> 1. **`action`** ***(StringName):*** the name of the action event.
>
> **Returns *(boolean):*** whether the action event is pressed

```
12    # --------------------
13    func _process(delta: float) -> void:
14        velocity.x = 0
15
16        if Input.is_action_pressed("ui_left"):
17            velocity.x = -x_velocity
18
```

34

Flip the DoodleHead texture horizontally by setting the **$Sprite2D.flip_h** property to **true**. **flip_h** flips the texture horizontally, which sets the Player to face in the same direction that it's moving.

$Sprite2D accesses the Player's child node, **Sprite2D** by node name. Make sure the spelling and capitalization match.



Reminder:

The **$** symbol is used in GDScript to refer to a node directly by name. This can only be done when the node that is called on is a child of the node the script is attached to.

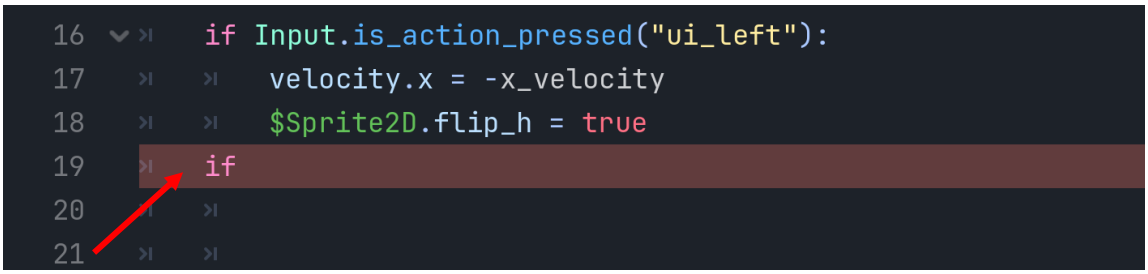
35 The if-statement is complete. The code will check if the **ui_left** action event is pressed, then....

- Move DoodleHead to the left.
- Flip the DoodleHead texture.

Copy the code for the if-statement and paste it underneath to create a second if-statement. Update the code to:

- Check if the **ui_right** action event is pressed.
- Set DoodleHead to move to the right.
- Flip the DoodleHead texture back.

Make sure the second if-statement is added **underneath** the first, **not inside** it.



36 Check the code for the second if-statement and save the script.

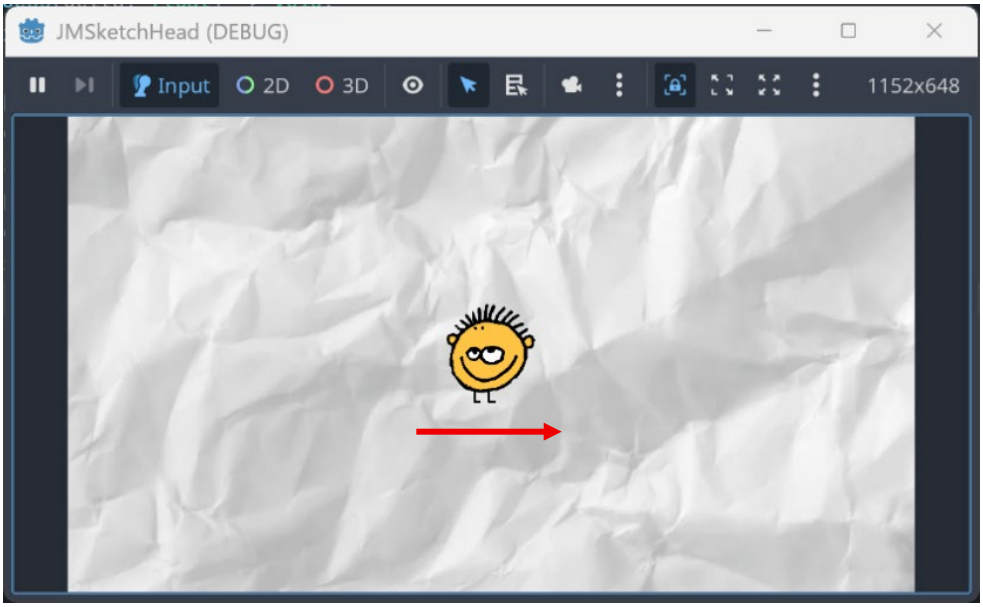
```
# --------------------
# TODO STEP 29
# Create the _process method
# --------------------
func _process(delta: float) -> void:
    velocity.x = 0

    if Input.is_action_pressed("ui_left"):
        velocity.x = -x_velocity
        $Sprite2D.flip_h = true
    if Input.is_action_pressed("ui_right"):
        velocity.x = x_velocity
        $Sprite2D.flip_h = false
```

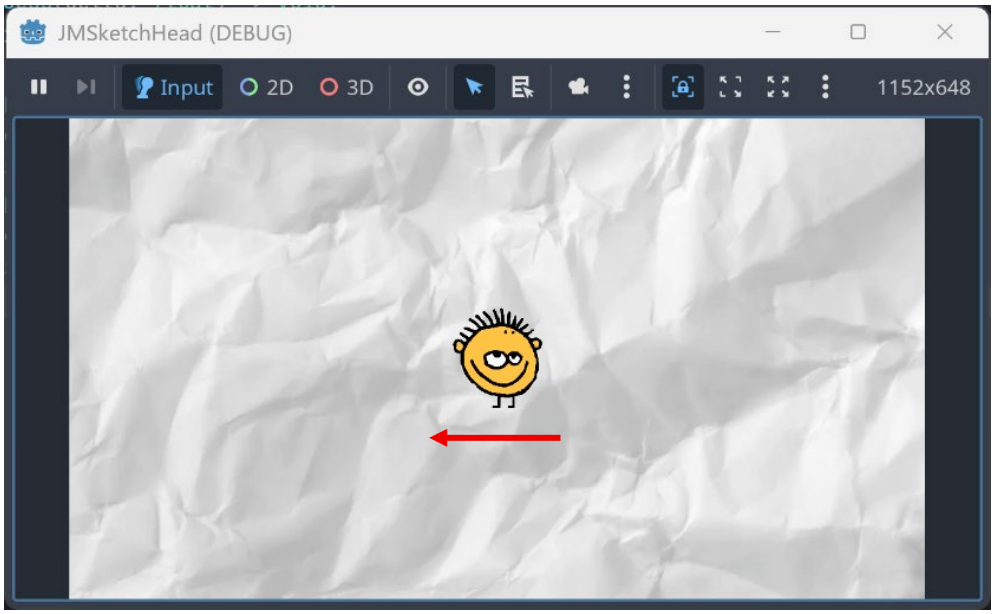
37 Playtest the project using the direction buttons.

Notice how the DoodleHead texture flips but the Player does not move.

This is because a specific method is needed to update the **position** of a **CharacterBody** based on its **velocity**. Without the method, the CharacterBody can't move! This method will be added later.

Close the playtest window.

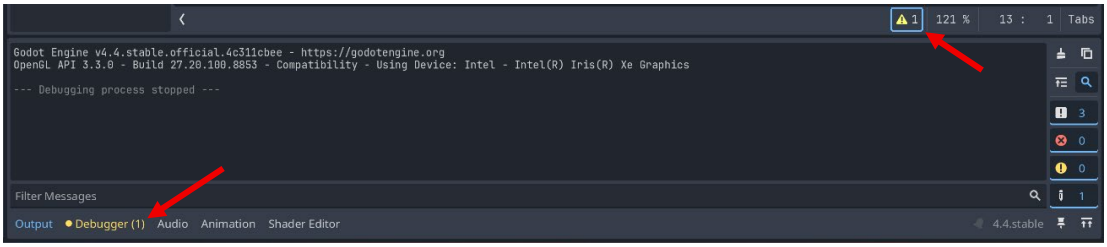




38 Look at the bottom panel. What do you notice?

A warning can be seen. A warning does not prevent the code from running.

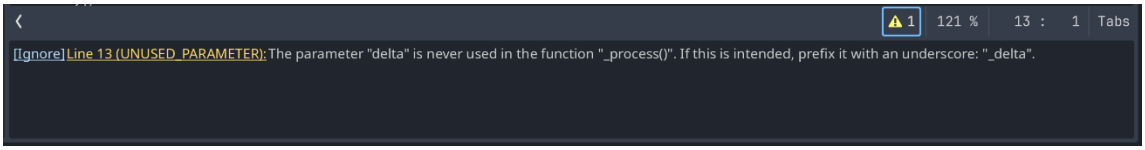
Click on the warning symbol to view the warning.



39 Read the warning.

What could the warning mean? Where in the code could the warning be referencing?

Return to the **player.gd** script and try to find the line of code causing the warning.



40

The warning points out that the parameter, **delta**, in the **_process** method is not used.

In the **player.gd** script, change the parameter from **delta** to **_delta**. Since delta is not used in the method, the **underscore (_)** will tell GDScript to ignore the argument.

```
13  func _process(_delta: float) -> void:
14      velocity.x = 0.0
15
16      if Input.is_action_pressed("ui_left"):
17          velocity.x = -x_velocity
18          $Sprite2D.flip_h = true
19      if Input.is_action_pressed("ui_right"):
20          velocity.x = x_velocity
21          $Sprite2D.flip_h = false
```

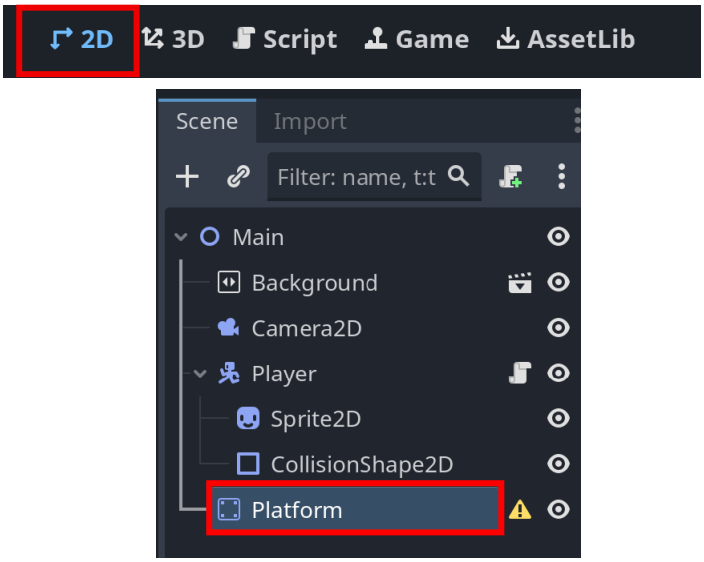
Pause for **Sensei Stop #3**!

Before continuing, check the code in the **player.gd** script.

Reminder: Save your work!

41 Click **2D** to switch from the script editor to the 2D workspace.

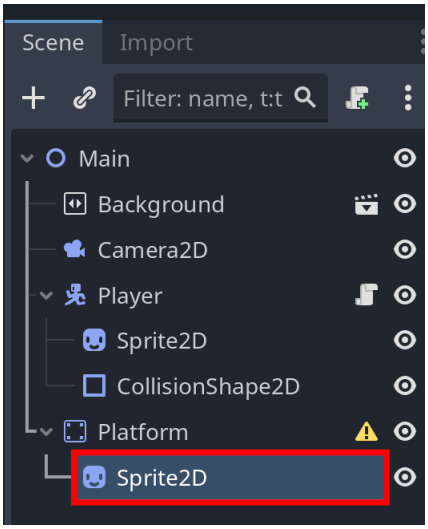
Add a **StaticBody2D** as a child to **Main** and rename the node **Platform**.

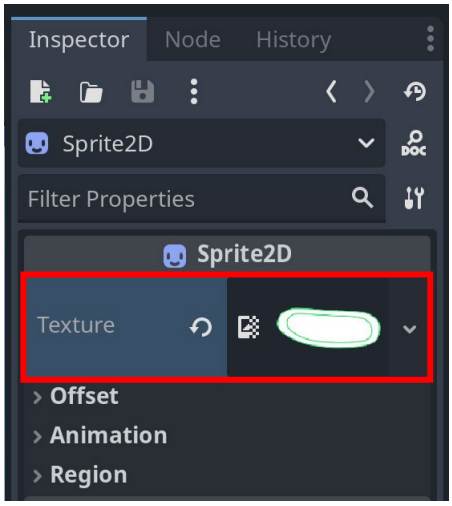


New Concept: StaticBody2D

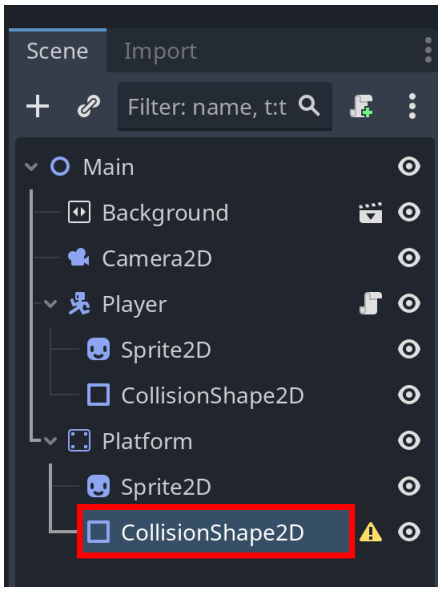
A **StaticBody2D** is a 2D physics body that can't be moved by external forces or contacts.

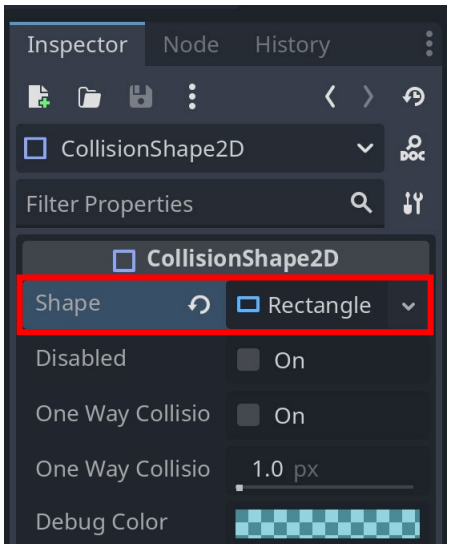
42 The Platform needs a texture and shape. Add a **Sprite2D** as a child to **Platform**. In the **Inspector** for the **Sprite2D,** assign the platform's Texture to **Platform.png** using **Quick Load**.



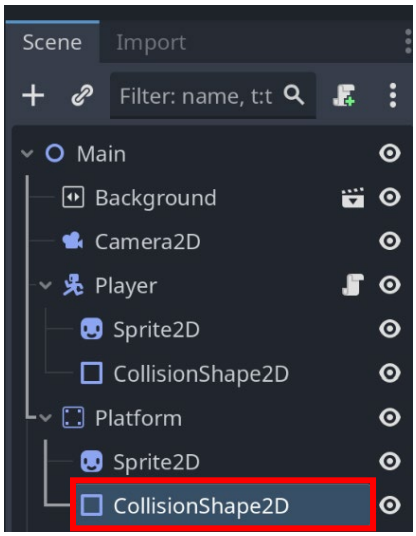


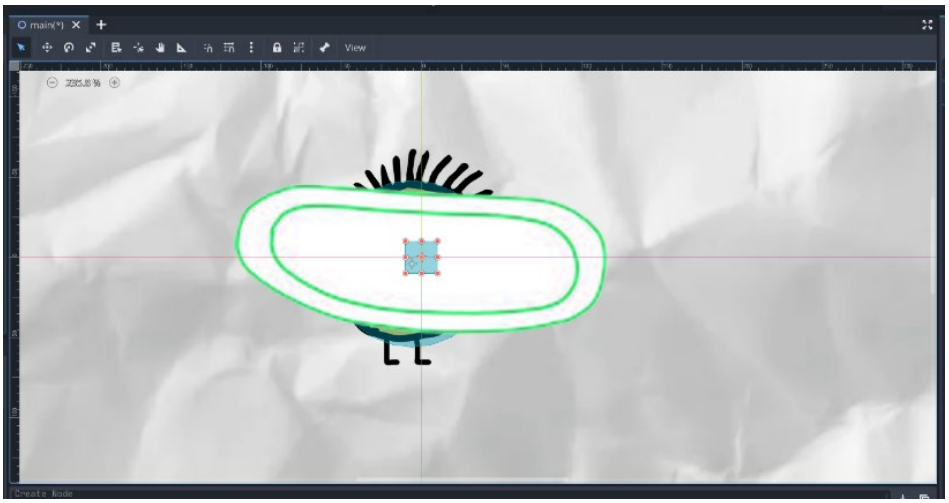
43 Add a **CollisionShape2D** as a child to **Platform**. In the **Inspector** for **CollisionShape2D**, set the shape to a **New RectangleShape2D.**

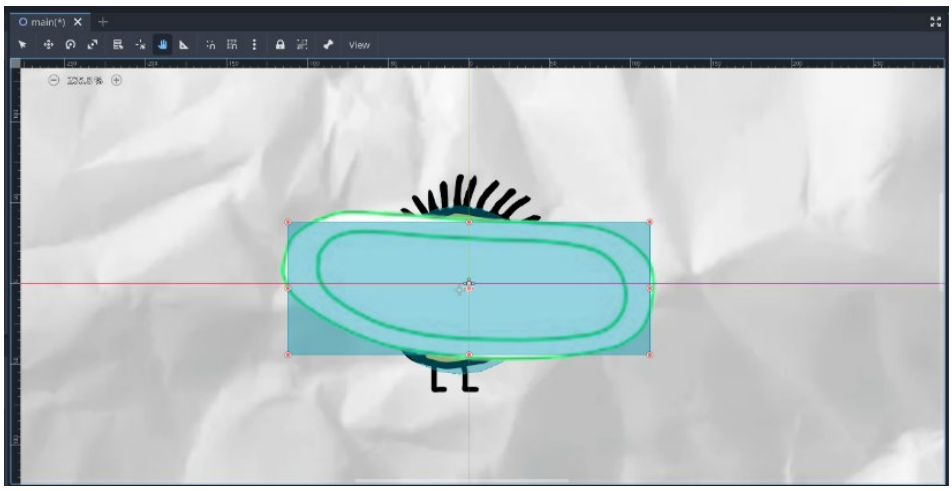




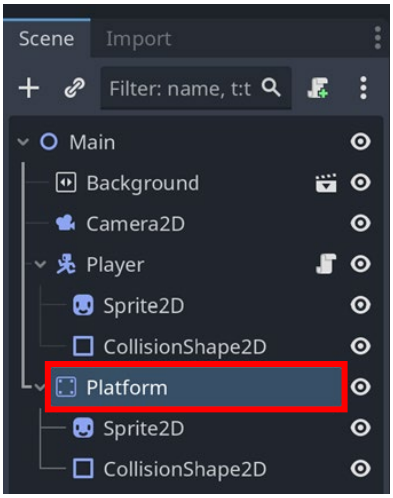
44 With the **Platform's CollisionShape2D** selected in the Scene menu, zoom in on the platform in the game window.

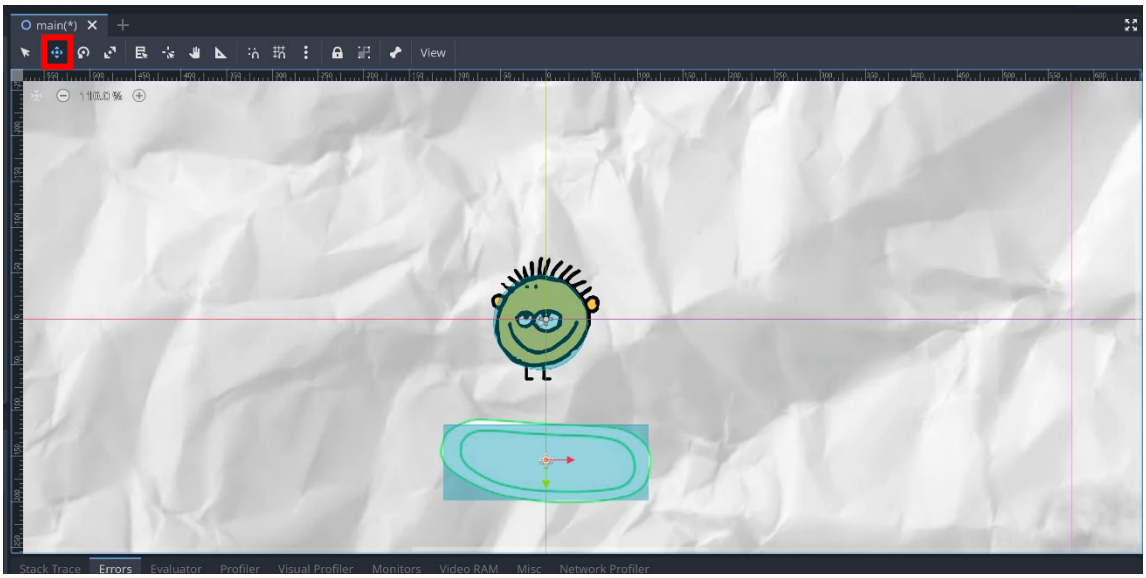






45 Select the **Platform** in **Scene**. Then, use the **Move** tool ( ) to position the **Platform** underneath DoodleHead. The game window may need to be zoomed out when relocating the Platform.





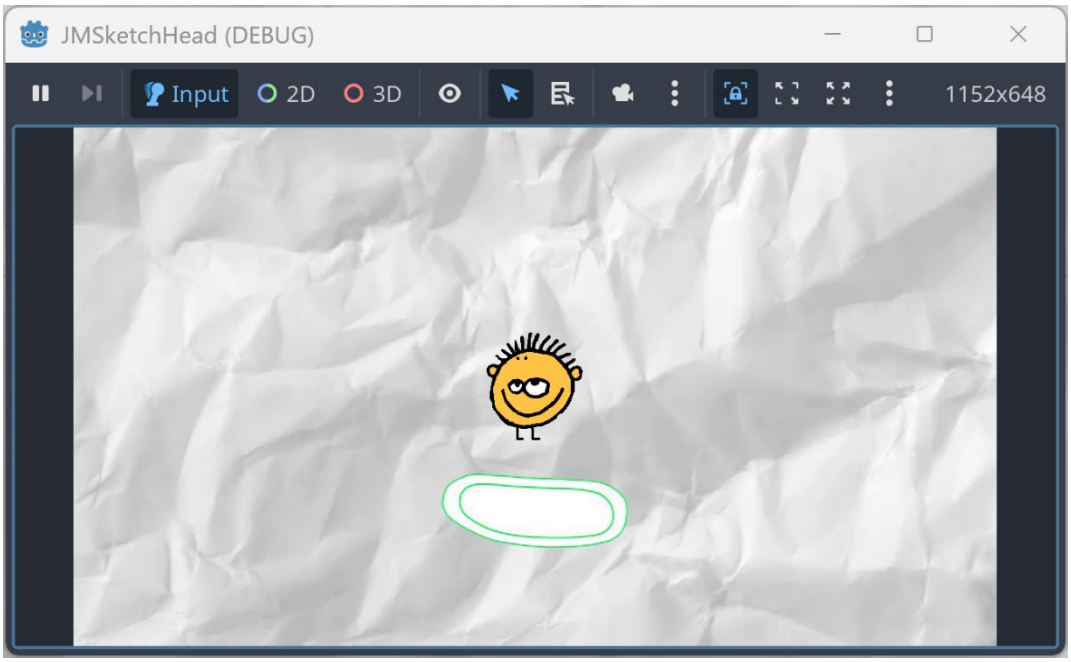
46

Playtest the project.

Does DoodleHead fall onto the platform?

DoodleHead and the platform both appear in the viewport but DoodleHead doesn't fall on the platform yet.

Close the playtest window.



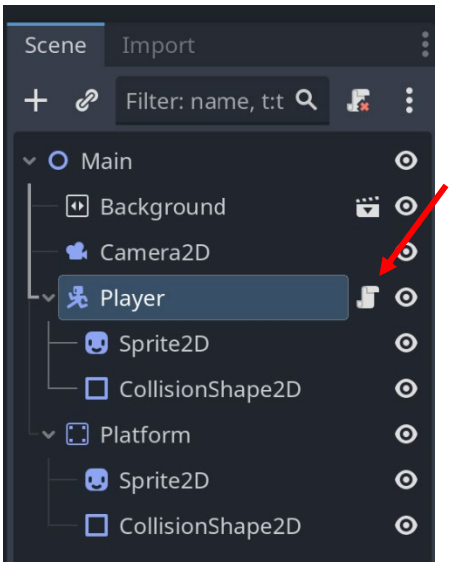
Pause for **Sensei Stop #4**!

Before continuing, check that the platform is set up correctly.

Reminder: Save your work!

47 Oh no! The Player doesn't fall during playtest. Add code to simulate gravity.

Click on the **script icon** beside the Player to open **player.gd**.



48 To simulate gravity, two new variables need to be added to the script. Using **@export**, create the following variables:

- A variable **y_velocity**, of type **float.**
- A variable **gravity** of type **float**.

Add a few empty lines under the **x_velocity** variable and create the two new variables. Refer to the **x_velocity** variable declaration for guidance.

```
3  # --------------------
4  # TODO STEP 25
5  # Create the physics variables
6  # --------------------
7  @export var x_velocity: float = 100
8
9
```

49 What could these two new variables represent? How will they help simulate gravity in the game?

The **y_velocity** variable will be used for Player's y-velocity. This will allow DoodleHead to jump off the platforms.

The **gravity** variable will be used to create y-acceleration in the game. This will cause DoodleHead to fall down back onto platforms, or out of the viewport.

```
3  # --------------------
4  # TODO STEP 25
5  # Create the physics variables
6  # --------------------
7  @export var x_velocity: float = 100
8  @export var y_velocity: float
9  @export var gravity: float
10
```

50 In the script, set **y_velocity** to **-700** and **gravity** to **1200** as default values. The y_velocity must be large enough so DoodleHead can jump but not so large that it isn't affected by gravity. **@export** allows these values to be adjusted in the Inspector later.

```
3  # --------------------
4  # TODO STEP 25
5  # Create the physics variables
6  # --------------------
7  @export var x_velocity: float = 100
8  @export var y_velocity: float = -700
9  @export var gravity: float = 1200
10
```

51

At the very **bottom** of the player.gd script, add in the **_physics_process()** method by typing in the code or using the code completion.

The **_physics_process()** method is called a set number of times per second, at the same time the engine calculates physics. By default this is 60 updates each second (about once every 16 milliseconds). This method is comparable to an onUpdateInterval loop in MakeCode Arcade.

```
25  # --------------------
26  # TODO STEP 51
27  # Create the _physics_process method
28  # --------------------
29  func _phy
        fn _physics_process(delta: float) -> void:
```

_physics_process(): This method is called during the physics processing step of the main loop. Physics processing allows for the frame rate to be synced to the game physics.

Parameters:

1. **delta *(float):*** the time in seconds since the previous frame. Delta will *generally* be constant in this method.

Returns *(void):* this method is void, it returns nothing.

52

Inside the **_physics_process()** method, increase the Player's **y velocity** by multiplying **gravity** by **delta**.

This will mimic real life physics by increasing (**+=**) how fast the player falls (**velocity.y**) based on the gravity (**gravity**), depending (*****) on the time elapsed since the last _physics_process() run (**delta**).

```
28  # --------------------
29  func _physics_process(delta: float) -> void:
30      velocity.y += gravity * delta
```

53

This game contains platforms for DoodleHead to jump on. If DoodleHead lands on a platform, it should bounce.

Using the **is_on_floor()** method, which checks whether the physics body has collided with the floor, try and code the **if**-statement to change DoodleHead's **y velocity** after hitting a platform.

Hint: The **y_velocity** variable created earlier is **negative**, unlike gravity which has a positive value. What could this variable be used for?

```
28  # --------------------
29  func _physics_process(delta: float) -> void:
30      velocity.y += gravity * delta
31
32      if is_on_floor():
33
```

`is_on_floor()`: Part of the CharacterBody2D/3D classes, this method returns true if the physics body collided with the floor on the last call of the **`move_and_slide()`** method.

Parameters: None

Returns *(boolean)*: whether the CharacterBody2D/3D is on the floor

54 Check the code. Was DoodleHead assigned a y velocity?

This line will cause DoodleHead to bounce up after colliding with a platform.

```
28    # --------------------
29    func _physics_process(delta: float) -> void:
30        velocity.y += gravity * delta
31
32        if is_on_floor():
33            velocity.y = y_velocity
```

55 Create a new line underneath (not inside) the if-statement inside the method, then call the **move_and_slide()** method.

move_and_slide() is the method that allows Godot to update the velocity of the CharacterBody. This method will allow DoodleHead to move side-to-side with the direction buttons using the set **x-velocity**.

Check the indentation and spacing for the method and save the script.

```
29    func _physics_process(delta: float) -> void:
30        velocity.y += gravity * delta
31
32        if is_on_floor():
33            velocity.y = y_velocity
34
35        move_and_slide()
```

move_and_slide(): Part of the CharacterBody2D/3D classes, this method moves the physics body based on velocity. If the physics body collides with another physics body (like a platform), it will redirect its velocity to slide along the surface of the object instead of stopping immediately.

Parameters: None

Returns *(boolean):* whether a collision happened

Pro Tip:

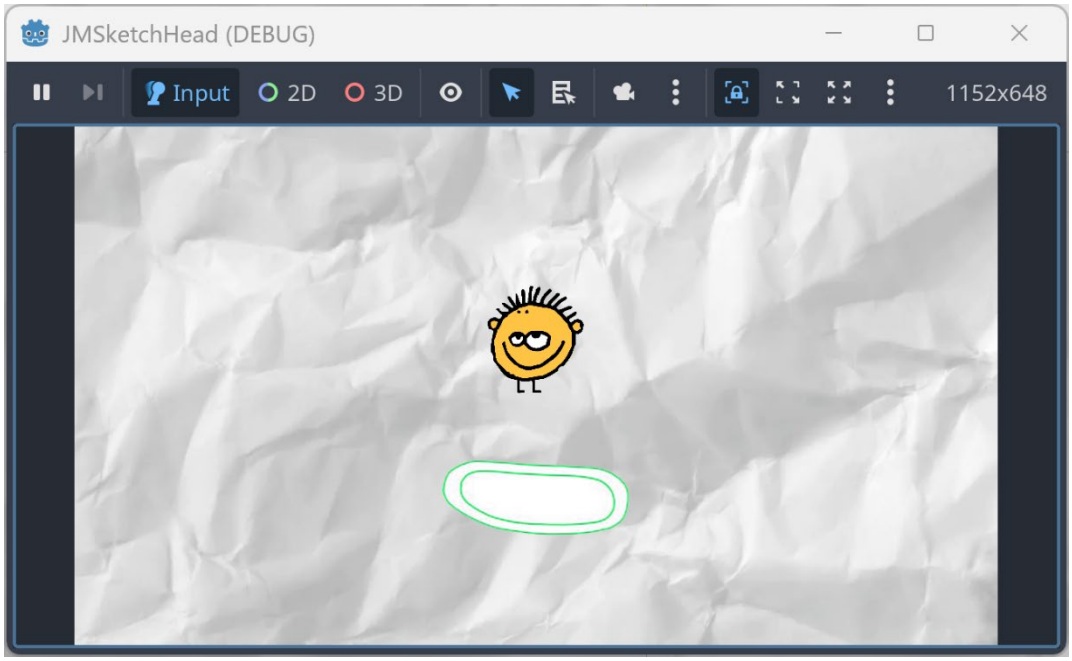
Sliding along another physics body makes game collisions more accurate! The CharacterBody will slide along angled physics bodies (like slopes) after colliding instead of stopping immediately. To do this, the CharacterBody's velocity is projected onto the surface normal of the colliding object.

56 Playtest the project.

Does DoodleHead bounce on the platform?

Does DoodleHead appear in front of or behind the platform?

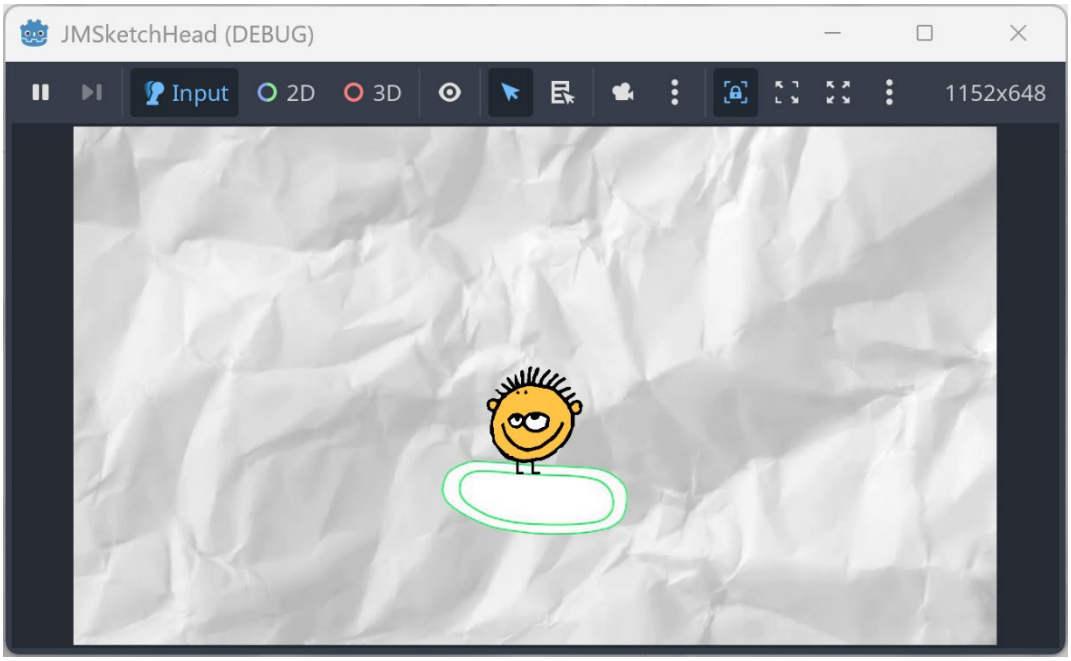
Close the playtest window.



57 In the **Inspector** for **Player**, open the **Ordering** drop-down menu to change the **Z Index** from **0** to **1** so DoodleHead appears in front of the platform.

58 Playtest the project.

Use the direction buttons to see how side to side movement affects DoodleHead's bouncing.



Pause for **Sensei Stop #5**!

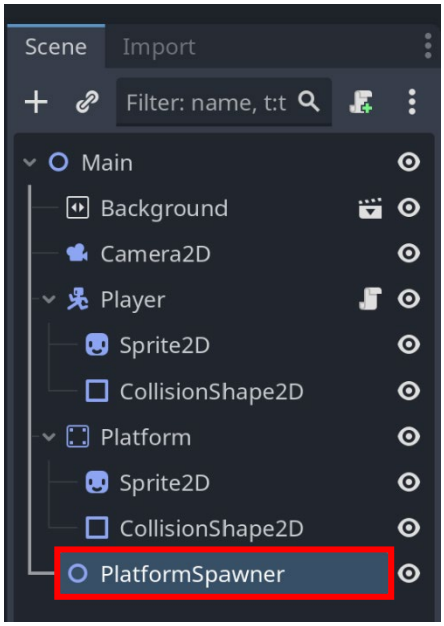
Before continuing, check that the code in the **player.gd** script is correct.

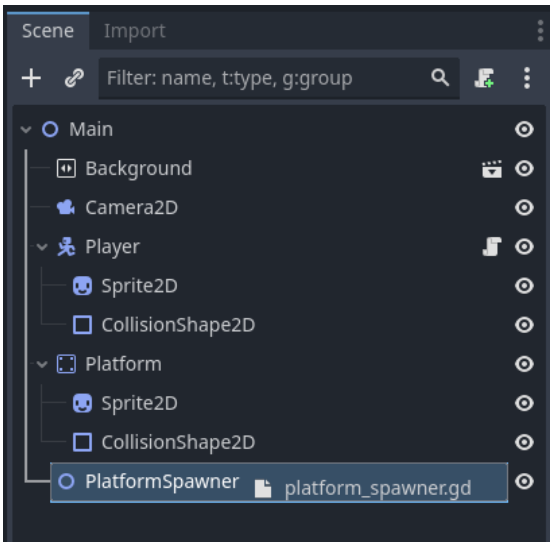
Reminder: Save your work!

59 DoodleHead needs more platforms to jump on.

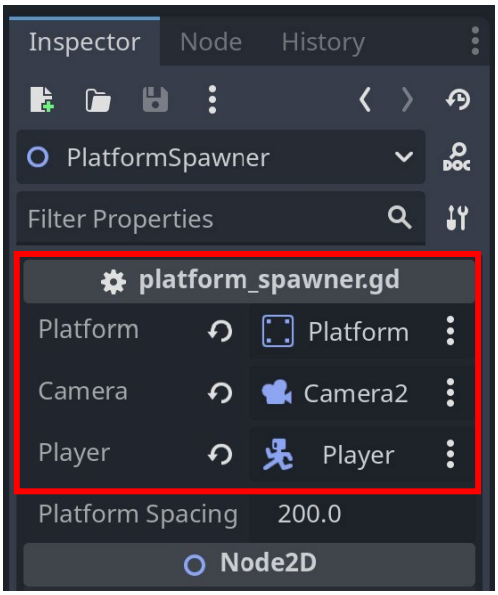
Add a **Node2D** as a child to **Main** and rename the node **PlatformSpawner**.

In **FileSystem**, find **platform_spawner.gd** in the **Scripts** folder. Drag the script to the **PlatformSpawner** node to attach it.





60 In the **Inspector** for **PlatformSpawner**, set **Platform** to the Platform node, **Camera** to the Camera2D node and **Player** to the Player node.

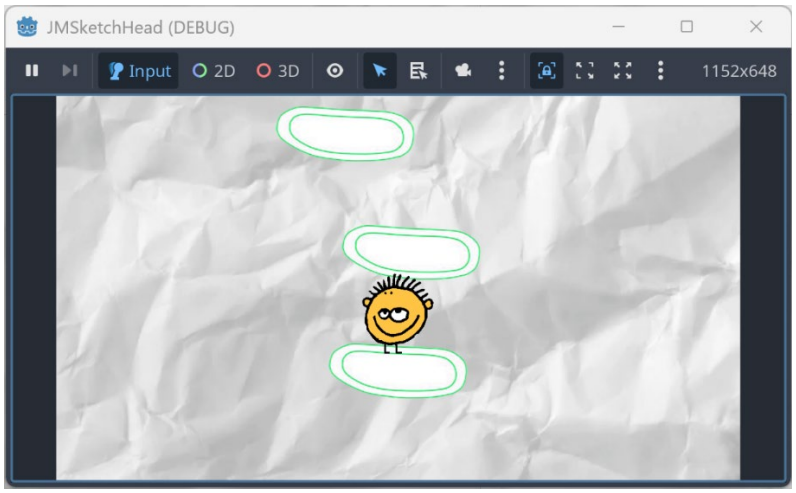


61 Playtest the project.

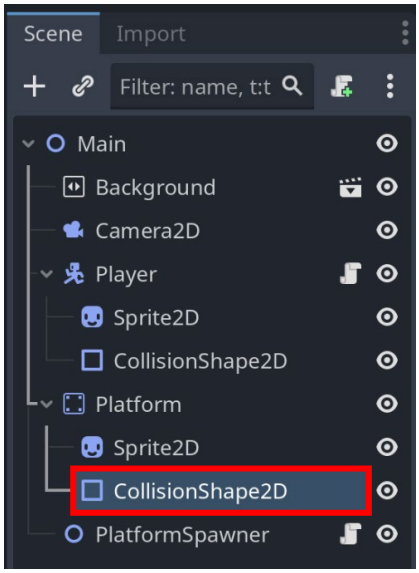
Do more platforms spawn in the viewport? Where do they appear?

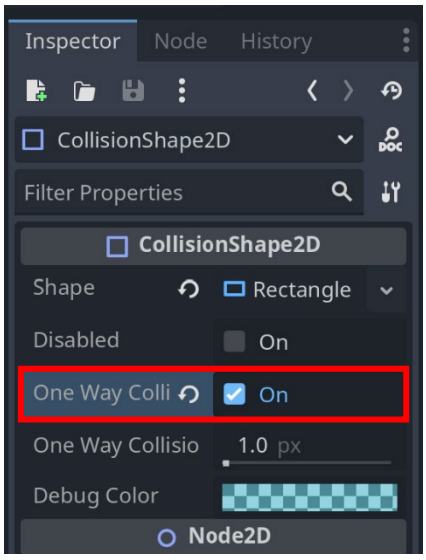
What happens to DoodleHead if a platform spawns directly above it?

Close the playtest window.



62 If a platform spawns above DoodleHead, the player gets stuck bouncing between the two platforms.



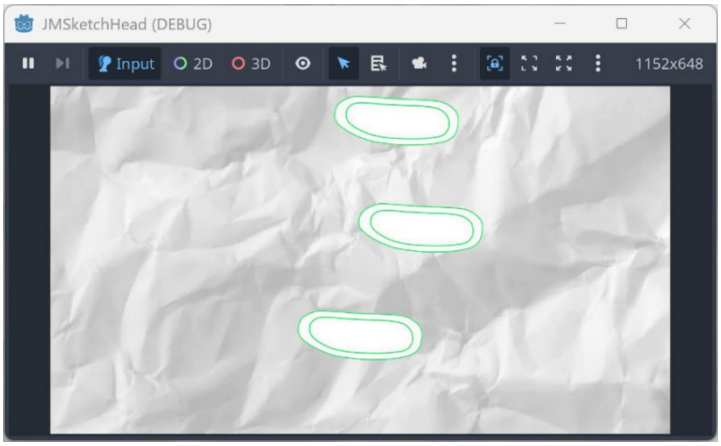


In the **Inspector** for the Platform's **CollisionShape2D**, turn on **One Way Collisions**. This allows items to pass through the collision shape from one direction but not the other.

63 Playtest the project.

DoodleHead can pass through the bottom of the platforms and jump up! And... out of the camera view?

Close the playtest window.

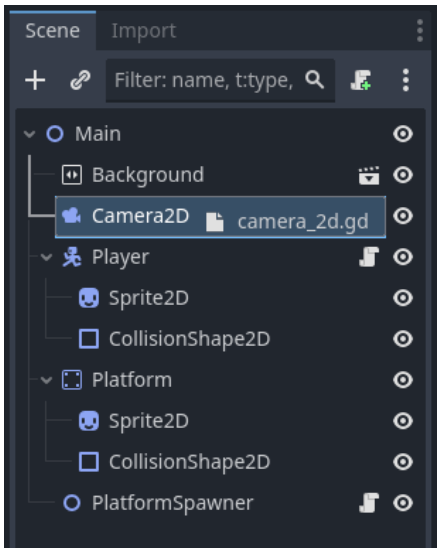


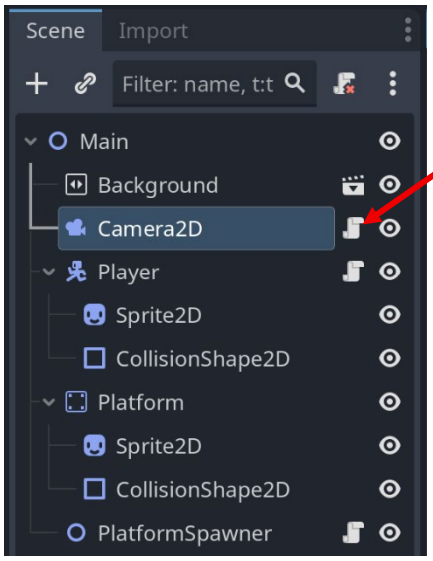
64 When DoodleHead bounces too high, it disappears! The camera needs to follow the player.

Is there a function used in MakeCode Arcade that does something similar?

In **FileSystem** in the **Scripts** folder, find **camera_2d.gd** and drag the script to attach it to **Camera2D**.

Beside **Camera2D**, click on the **script icon** to open camera_2d.gd.





65

The camera will need to follow the player, so a node reference is needed.

Since this variable will be a reference to the **Player** node, and the **Player** is a sibling node to the **Camera2D**, accessing **Player** via **$** (dollar sign) is not sufficient. Instead, it will be assigned in the **Inspector**.

Create a new **@export** variable named **player** of type **CharacterBody2D**.

```
1   extends Camera2D
2
3   # --------------------
4   # TODO STEP 65
5   # Create the player variable
6   # --------------------
7
8
```

66

Underneath the player variable, add in the **_process** method by typing in the code or using the code completion tool.

The **_process** method is similar to an onUpdate loop in MakeCode Arcade.

```
1   extends Camera2D
2
3   # --------------------
4   # TODO STEP 65
5   # Create the player variable
6   # --------------------
7   @export var player: CharacterBody2D
8
9
10  # --------------------
11  # TODO STEP 66
12  # Create the _process method
13  # --------------------
14  func _pro
        fn _process(delta: float) -> void:
        fn _property_can_revert(property: StringName) -> bool:
        fn _property_get_revert(property: StringName) -> Variant:
        fn _get_property_list() -> Array[Dictionary]:
        fn _physics_process(delta: float) -> void:
        fn _validate_property(property: Dictionary) -> void:
        fn _get(property: StringName) -> Variant:
```

67 Inside the method add an **if**-statement.

Add code to check if the **player's y position** is **less than** the **camera's y position**.

```
10  # --------------------
11  # TODO STEP 66
12  # Create the _process method
13  # --------------------
14  func _process(delta: float) -> void:
15      if 
```

68 In the **if** -statement, **player.position.y** accesses the position of the player during game play as long as the variable is assigned to the player node.

To get the camera's position, **position.y** is used. Unless specified, the **position property** will get the position of the node the script is attached to, which is **Camera2D** in this case.

Write a line of code inside the if-statement to **set** the **camera's y position** to the **player's y position**.

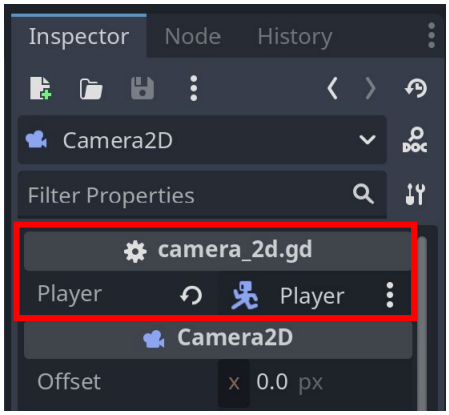
```
10  # --------------------
11  # TODO STEP 66
12  # Create the _process method
13  # --------------------
14  func _process(delta: float) -> void:
15      if player.position.y < position.y:
16          
```

69 This code will move the camera's position up with DoodleHead to keep the player in the camera view.

```
14  func _process(delta: float) -> void:
15      if player.position.y < position.y:
16          position.y = player.position.y
```

Save the script.

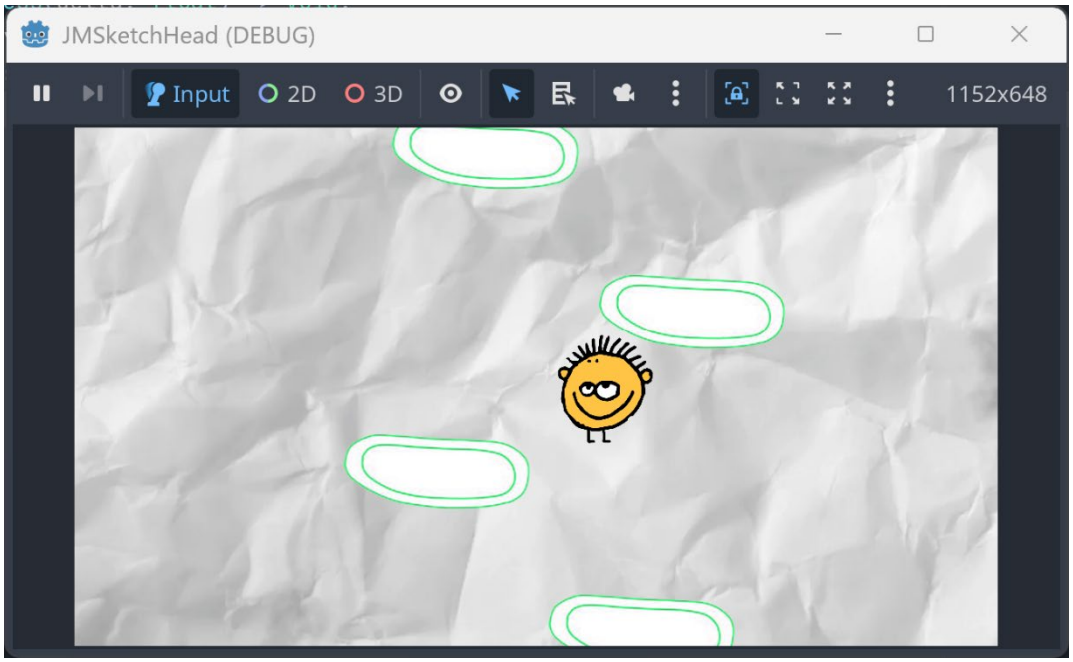
70 In the **Inspector** for **Camera2D**, set **Player** to Player.



71 Playtest the project.

How high up can DoodleHead go?

Close the playtest window.



72 A warning error appears in the debugger. Does this error look familiar? How can it be fixed?

73 The parameter **delta** is not used in the method. Add an underscore in front of delta to prevent warning errors.

Save the script.

```
13    # --------------------
14    func _process(_delta: float) -> void:
15        if player.position.y < position.y:
16            position.y = player.position.y
```

74 Customize the project.

In the **Inspector** for **PlatformSpawner**, the value of **platform spacing** is set to 200. Adjust the value of **platform spacing** to change how much *vertical* space appears between the spawned platforms.

A larger value will spawn the platforms further apart and a smaller value will spawn the platforms closer together. Tinker with the Platform Spacing value and playtest the project to determine how much vertical space is right.

75 In the **Inspector** for **Player**, adjust the values of **x velocity** and **y velocity**.

X velocity sets how fast DoodleHead moves along the **x axis.**

Y velocity sets how high DoodleHead jumps off a platform.

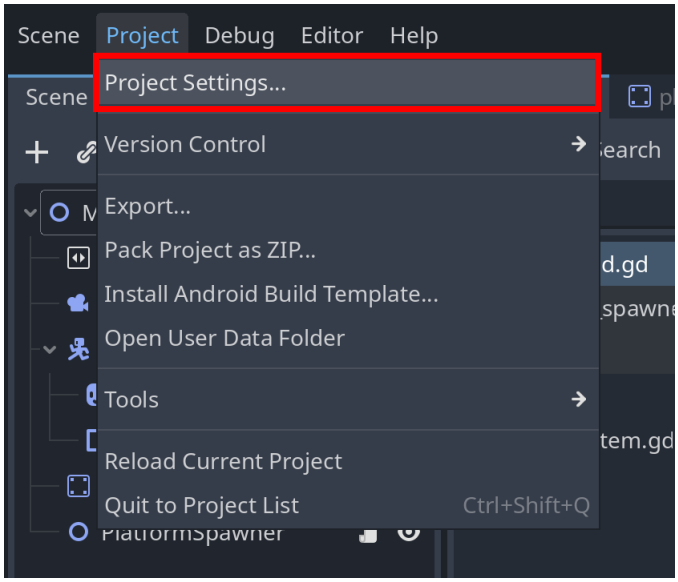
Pause for **Sensei Stop #6**!

Before continuing, check that the code in the **camera_2d.gd** script is correct.

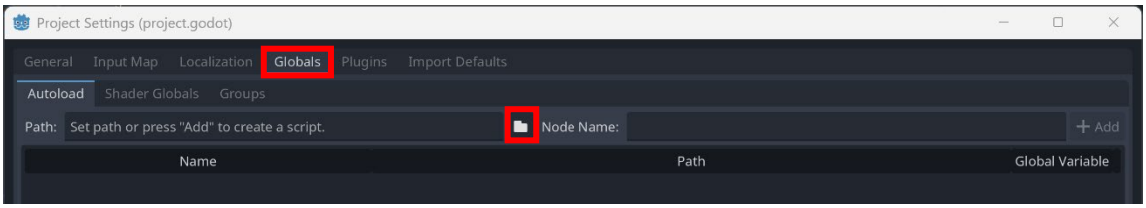
Reminder: Save your work!

76 The game needs a score and game over screen! A global score variable needs to be added first.

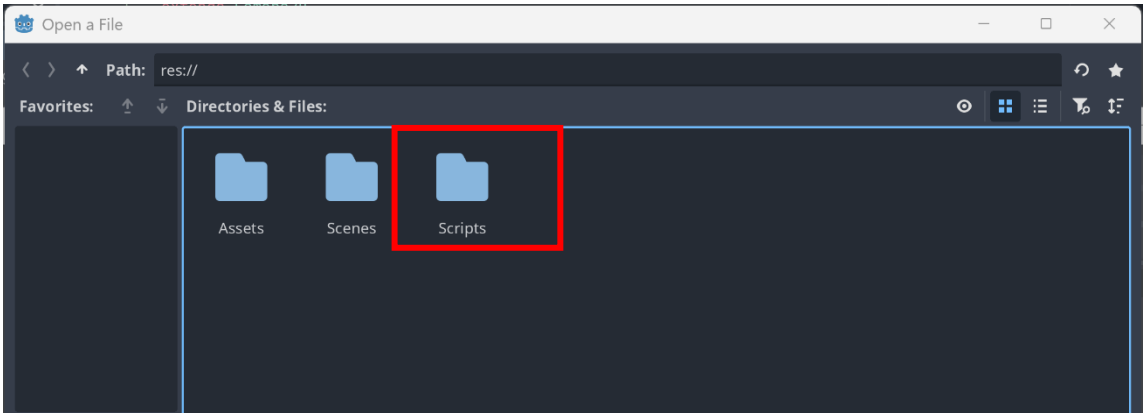
Under **Project**, open **Project Settings**.



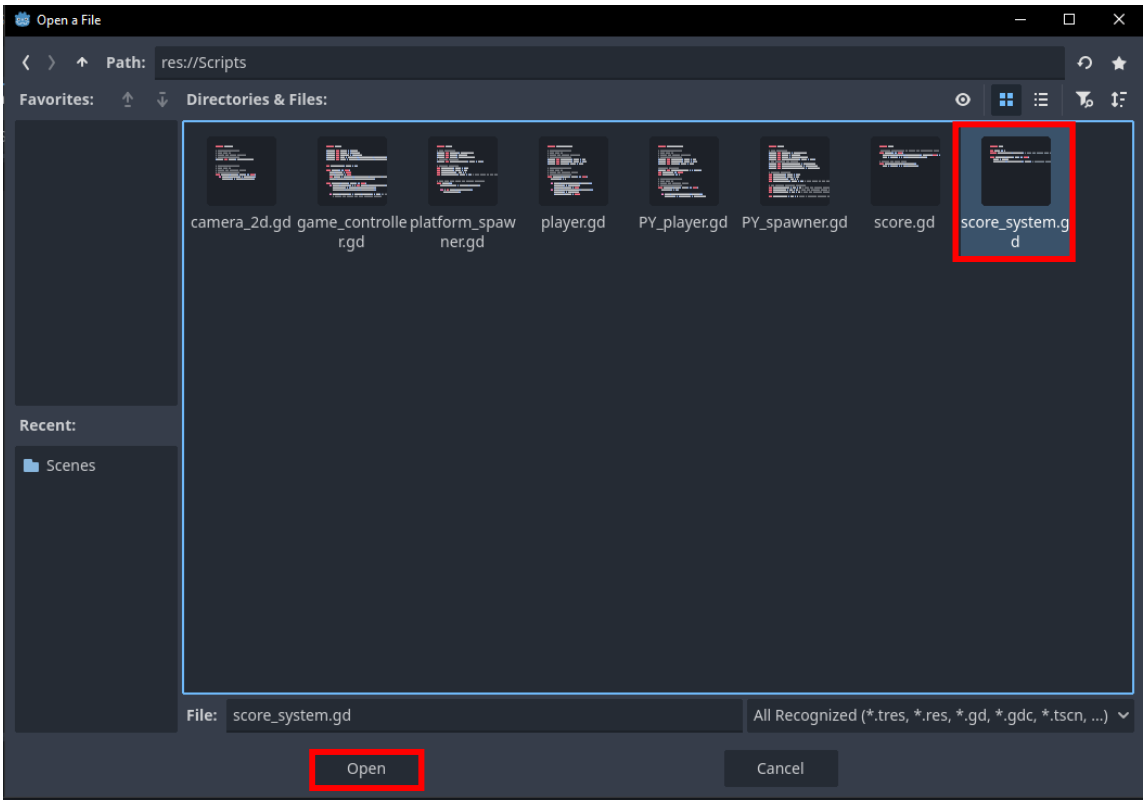
77 In the project settings, select **Globals** and click the **file icon**.



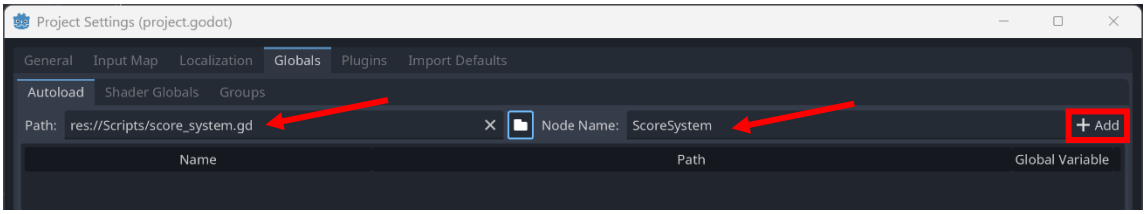
78 Double click to open the **Scripts** folder.



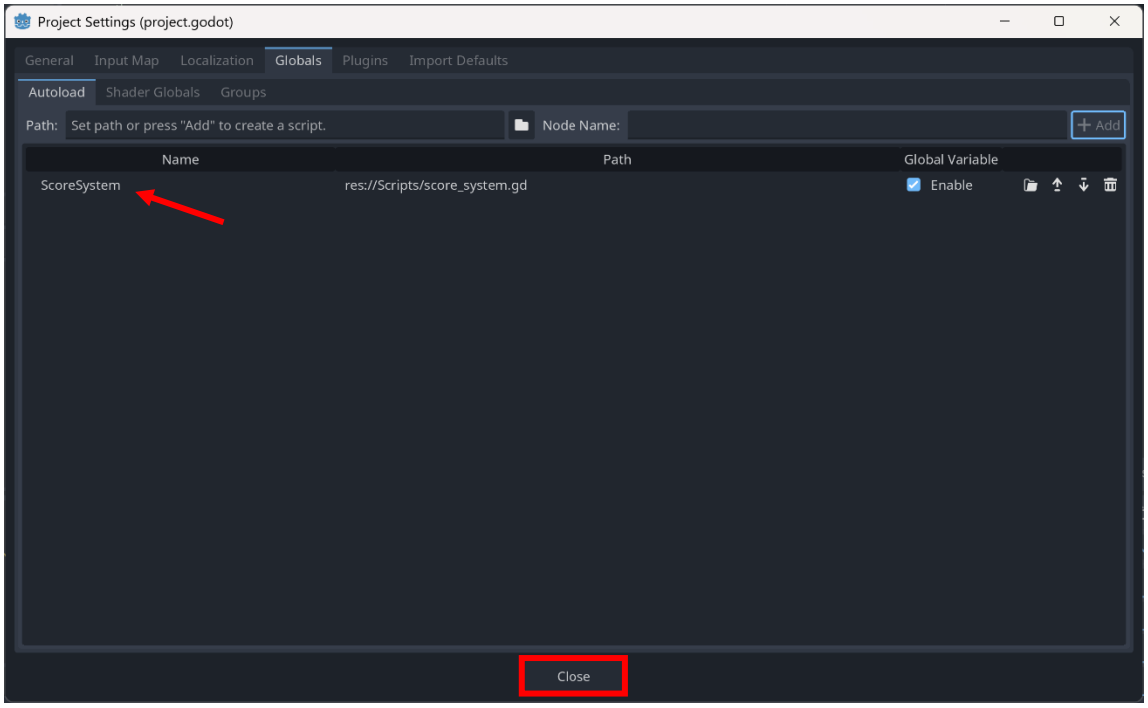
79 Inside the **Scripts** folder, select **score_system.gd** and click **Open**.



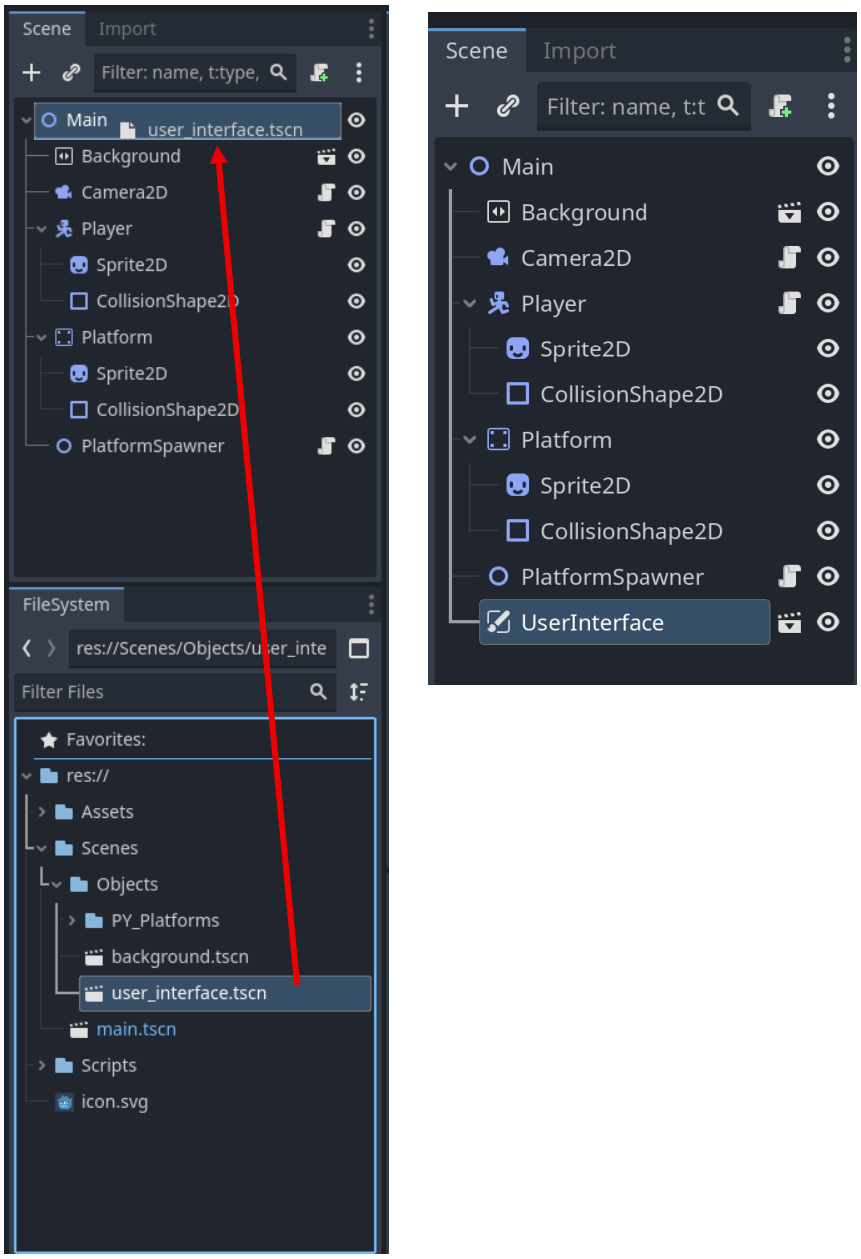
80 The Path and Node Name will update with the added script. Click **Add** to add the global variable.



81 A global variable **ScoreSystem** has been added. Click **Close**.



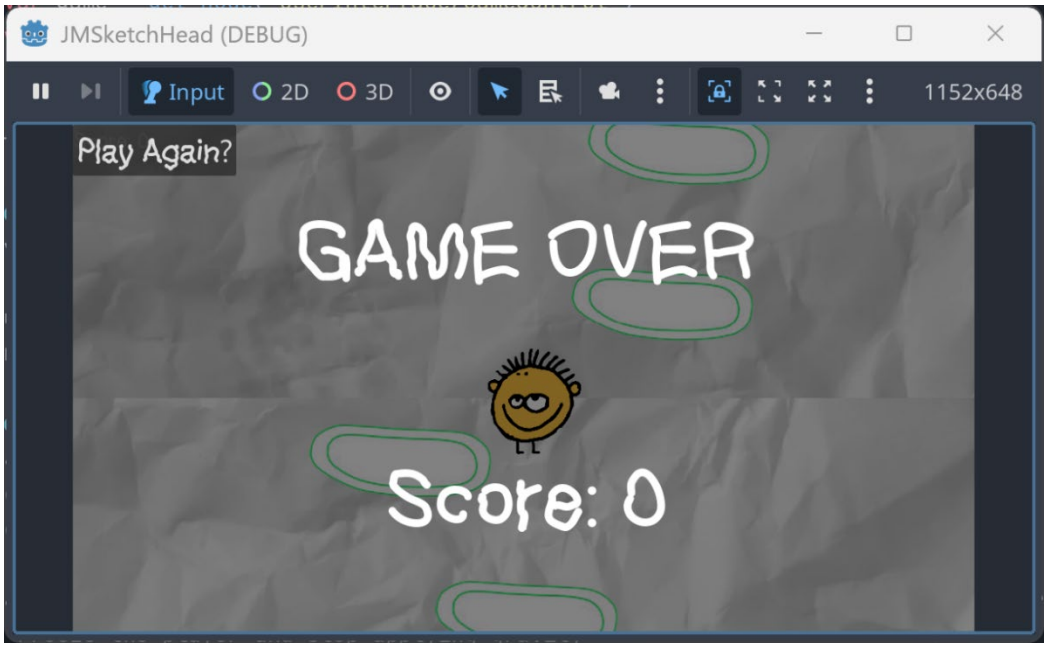
82 In the **FileSystem**, find the **user_interface.tscn** scene in the **Objects** folder inside of **Scenes**. Drag the scene onto the **main root**.



83 Playtest the project.

What happens with the score? What happens with the game over screen?

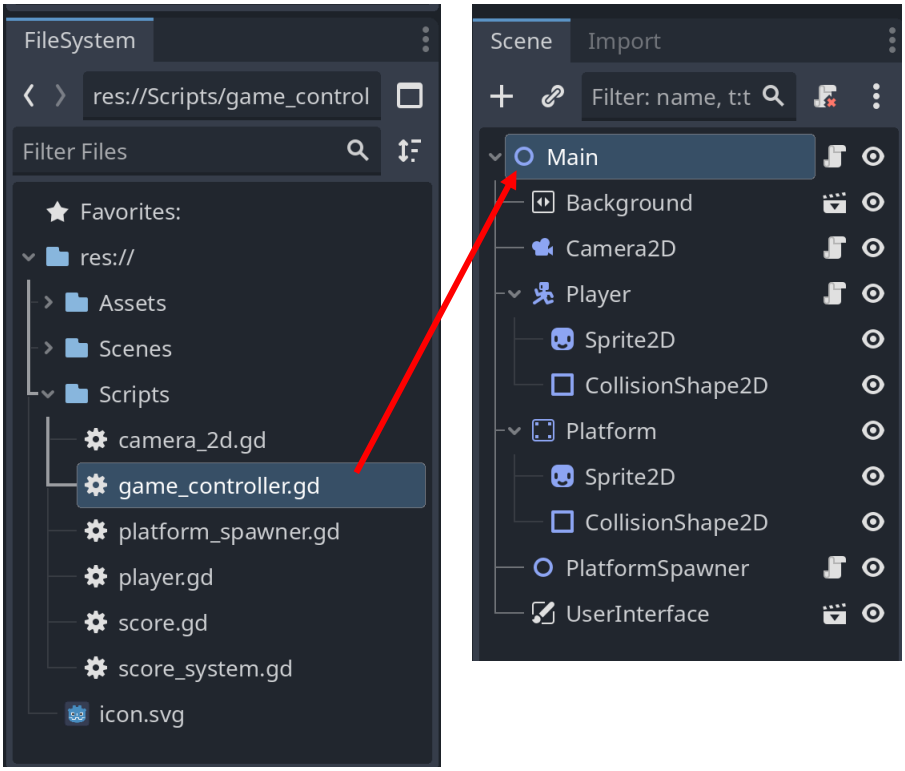
Close the playtest window.



84 The score does not update and the game over screen is present during game play.

In the **FileSystem**, find the **game_controller.gd** script in the **Scripts** folder. Drag the script and attach it to **Main**.

This script will control the game over canvas and update the score, as DoodleHead jumps onto platforms.



85 Playtest the project.

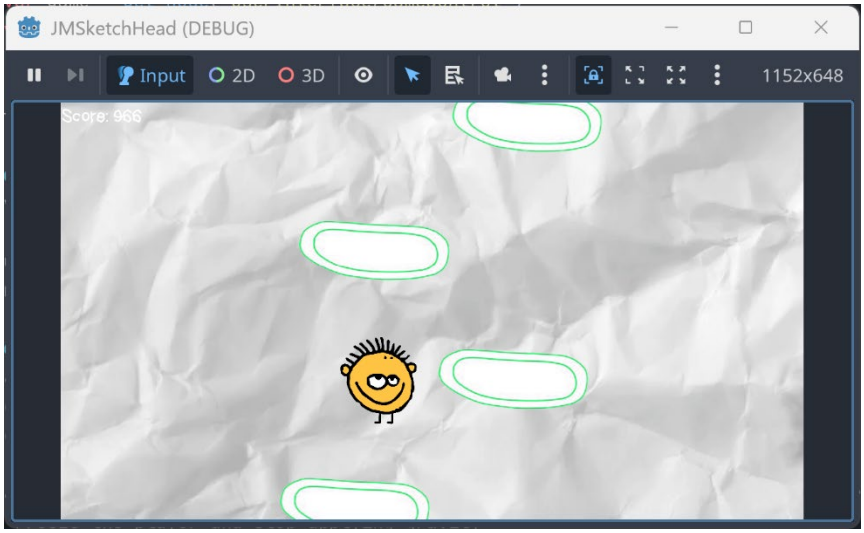
Does the score update?

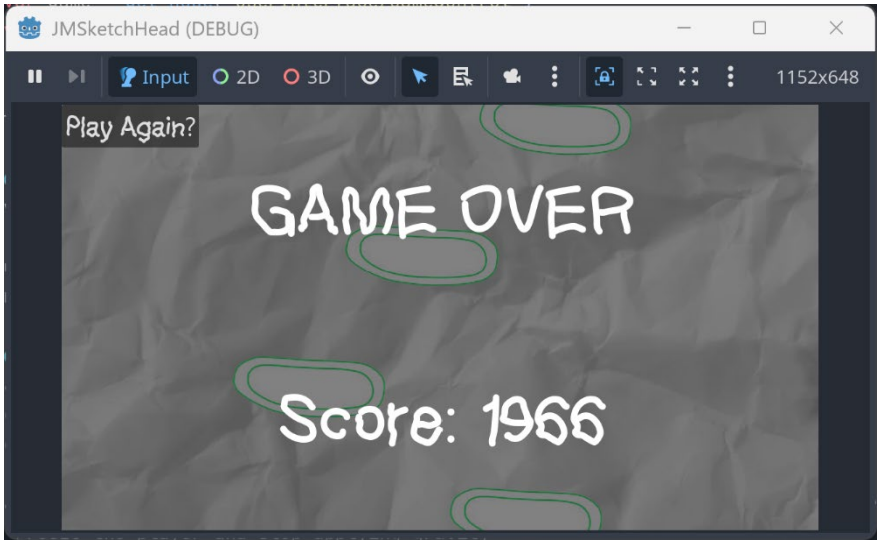
The score should continuously update with each "unit" of height DoodleHead gains has he jumps higher!

What happens when DoodleHead falls off the screen?

Does the game restart when the **Play Again?** button is clicked?

Close the playtest window.





Pause for **Sensei Stop #7**!

Congratulations on making your first endless platformer in Godot! Amazing!

- When might a CharacterBody2D be used to create a player instead of a RigidBody2D?
- How do if-statements differ in JavaScript and GDScript?
- When might platforms with one-way collisions be useful?

Reminder: Press CTRL + S to save your work and submit!