Build Your First 2D Game with Godot 4

Brandon Lewis ~ CodeMash 2025

Download Art Assets and Slides at:

https://github.com/brandon-lewis/CodeMash2025-PixelDash

Download the Standard version of Godot 4.3 (not Mono)

(muted for sanity)

About Me

Session Resources: https://github.com/brandon-lewis/CodeMash2025-PixelDash

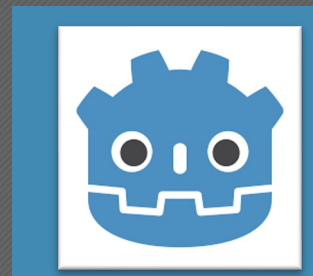
Consultant, gamer, hobbyist game dev, general geek.

brandonmlewis

brandon-lewis

What is Godot?

Session Resources: https://github.com/brandon-lewis/CodeMash2025-PixelDash



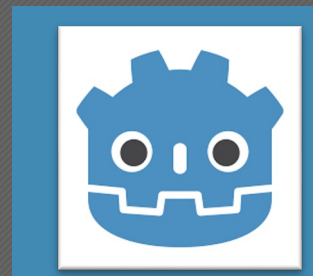
- A completely* free and open-source (MIT license) general-purpose 2D and 3D game engine.
- Build games or applications for desktop (Windows, macOS, Linux), mobile (Android and iOS), web** (HTML5), AR/VR, and even Steam Deck!
- Offers multiple languages for development – GDScript, C#, and C++.

* Seriously, it's totally free – no licensing or royalty contracts, ever.

** Web export is currently experimental in Godot 4.x; consider using Godot 3.x for web/HTML exports.

Why use Godot?

Session Resources: https://github.com/brandon-lewis/CodeMash2025-PixelDash



BENEFITS

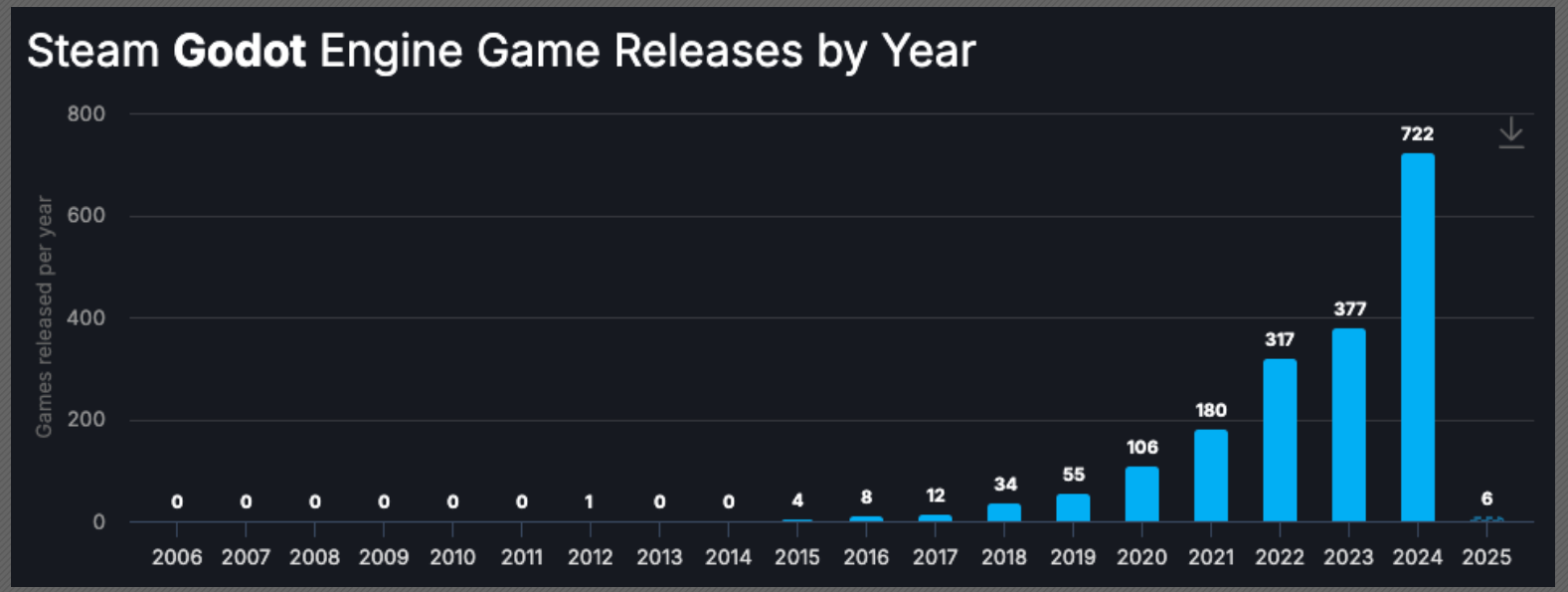
- Offers focused support for Vulkan and OpenGL renderers.
- GDScript is a high-level, dynamically-typed language.
- Version-control friendly.
- Completely free and open source.
- Editor plugins are first-class citizens.

DRAWBACKS

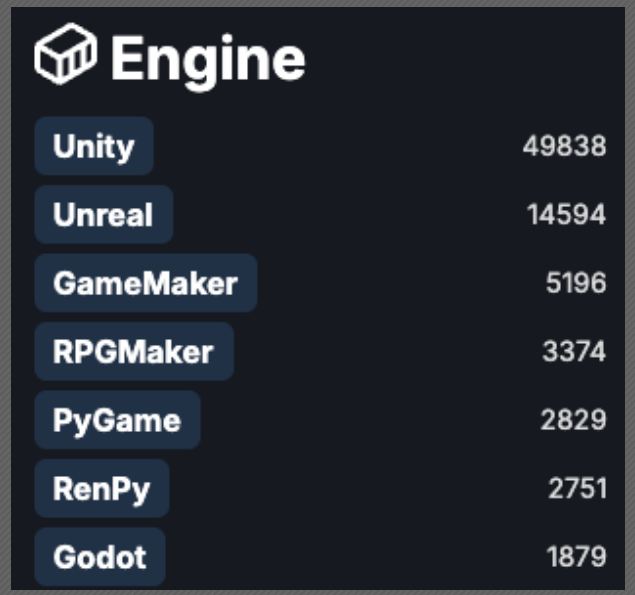
- Direct3D (prevalent on Windows and Xbox) support is currently experimental.
- GDScript is a high-level, dynamically-typed language.
- Not as mature or widely adopted as other engines (Unity, Unreal, GameMaker).

Growing Adoption

Session Resources: *https://github.com/brandon-lewis/CodeMash2025-PixelDash*



https://steamdb.info/stats/releases/?tech=Engine.Godot



https://steamdb.info/tech/

All stats as of Jan 9, 2025

Notable games/apps made with Godot

Session Resources: https://github.com/brandon-lewis/CodeMash2025-PixelDash

Brotato

(84.2k Steam reviews, 96% positive)

Dungeondraft

Halls of Torment

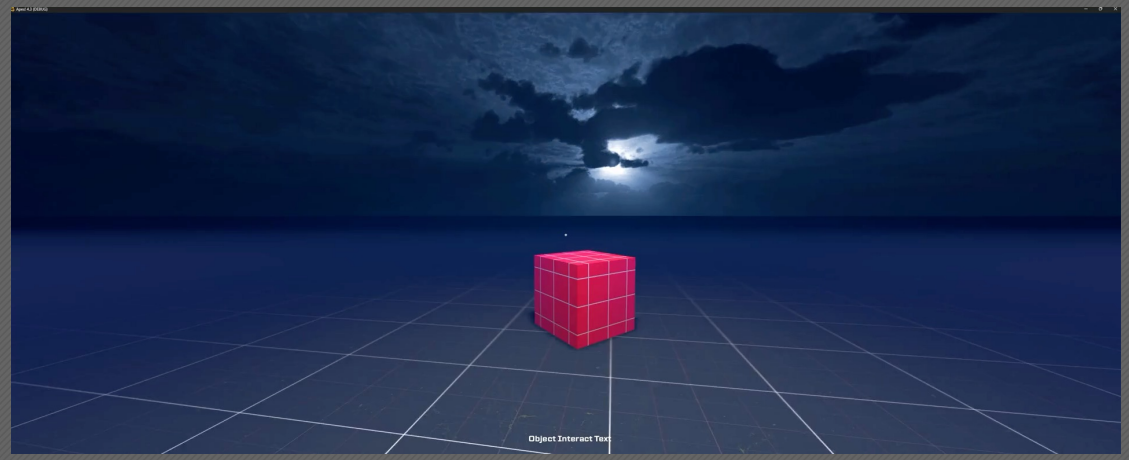
(26k Steam reviews, 96% positive)

Dome Keeper

(12.6k Steam reviews, 91% positive)

Progress shared in the r/Godot subreddit.

Session Resources: https://github.com/brandon-lewis/CodeMash2025-PixelDash



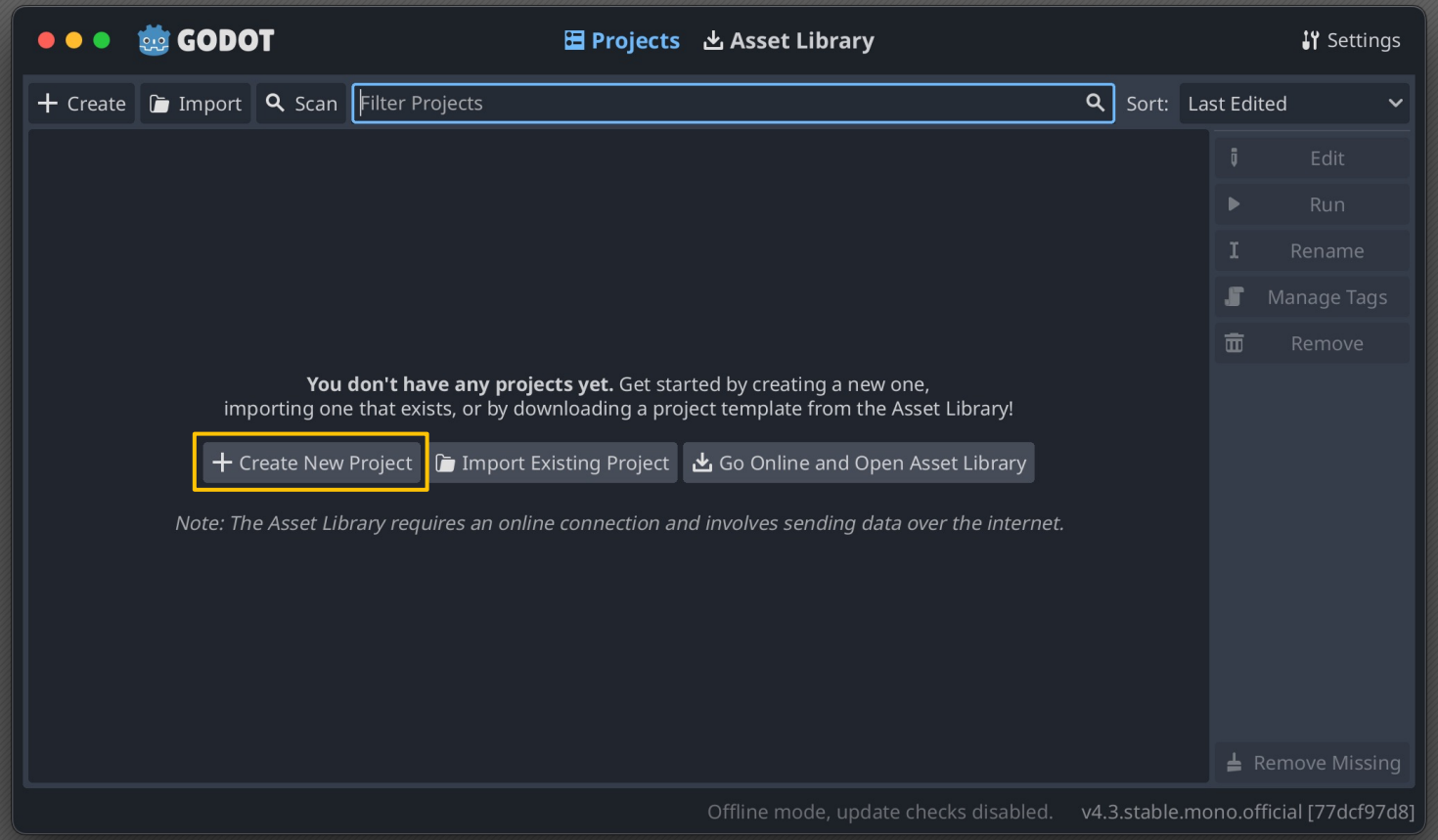
Workshop Goals

Session Resources: https://github.com/brandon-lewis/CodeMash2025-PixelDash

1. Control & Animate a Character
2. Build a Level with Tiles
3. Add Player & Level Polish
   (Camera, Gravity, Sound, Background)
4. Detect Collisions with Trampolines
5. Create a basic Mushroom Enemy
6. Add Collectible Fruit and HUD Display
7. Respawn Character when Defeated
8. Create a Main Menu & Change Scenes
9. Open Workshop, Tinker Time

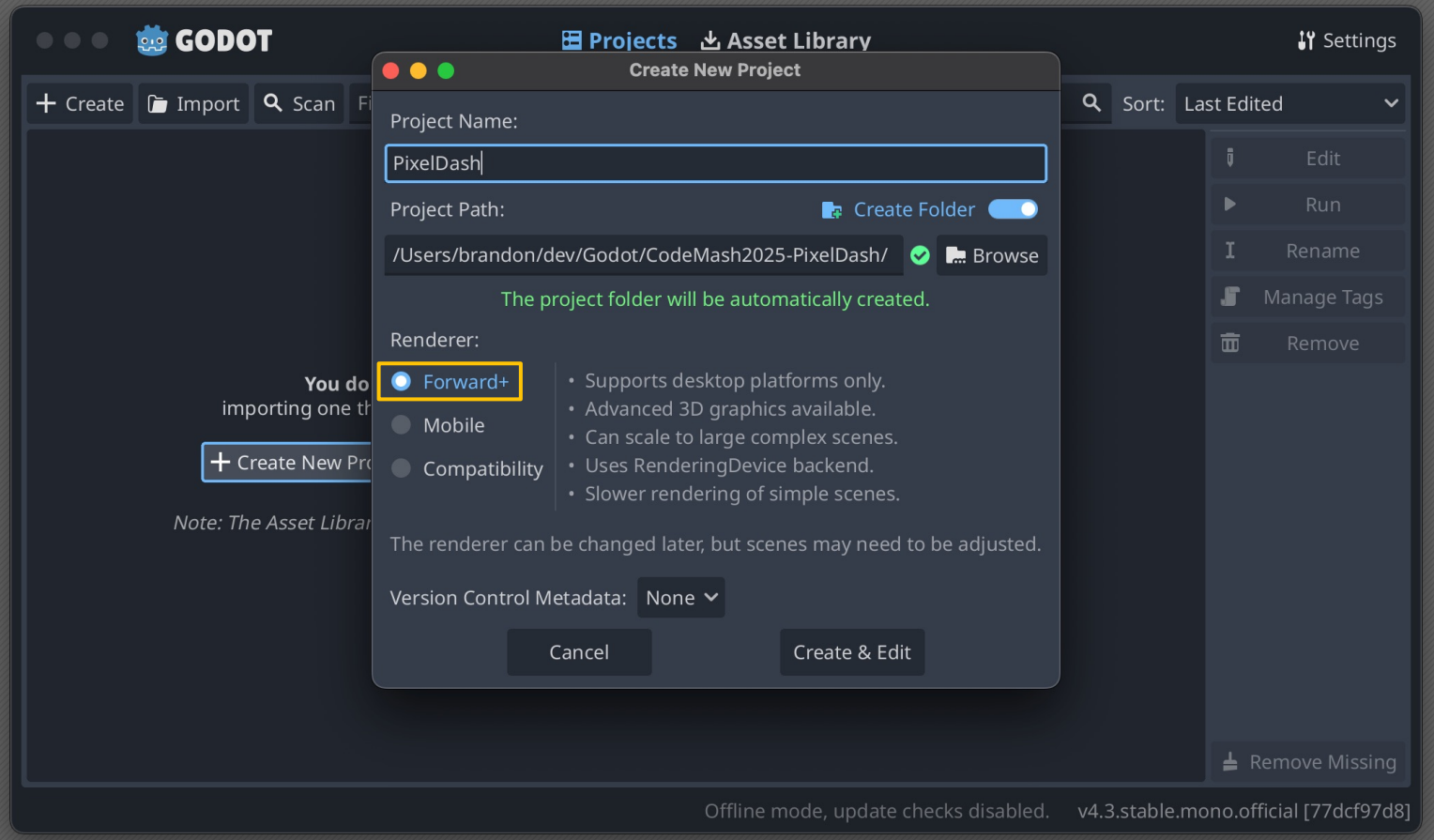
Opening Godot for the First Time

Session Resources: https://github.com/brandon-lewis/CodeMash2025-PixelDash



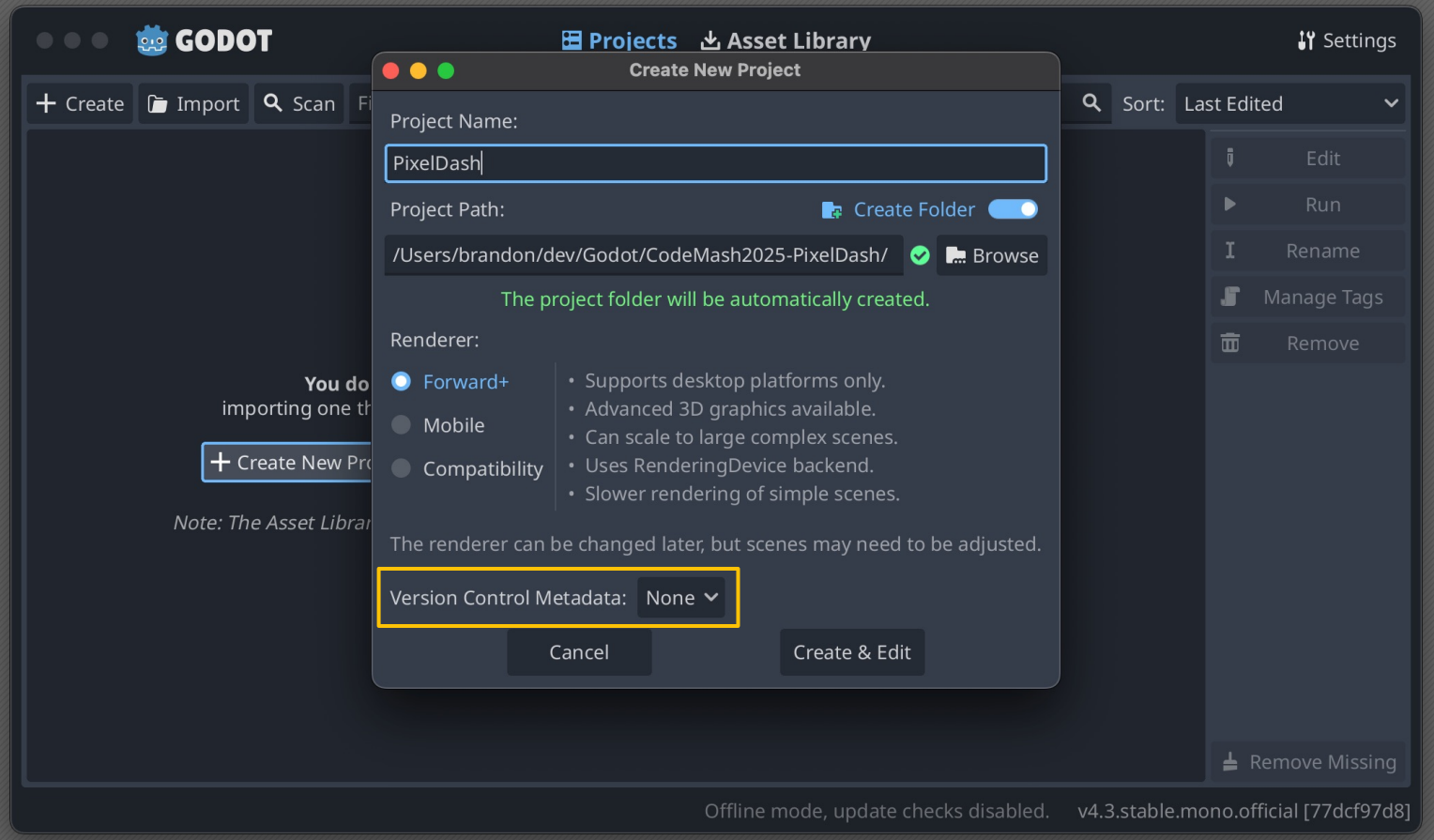
Creating a New Project

Session Resources: https://github.com/brandon-lewis/CodeMash2025-PixelDash



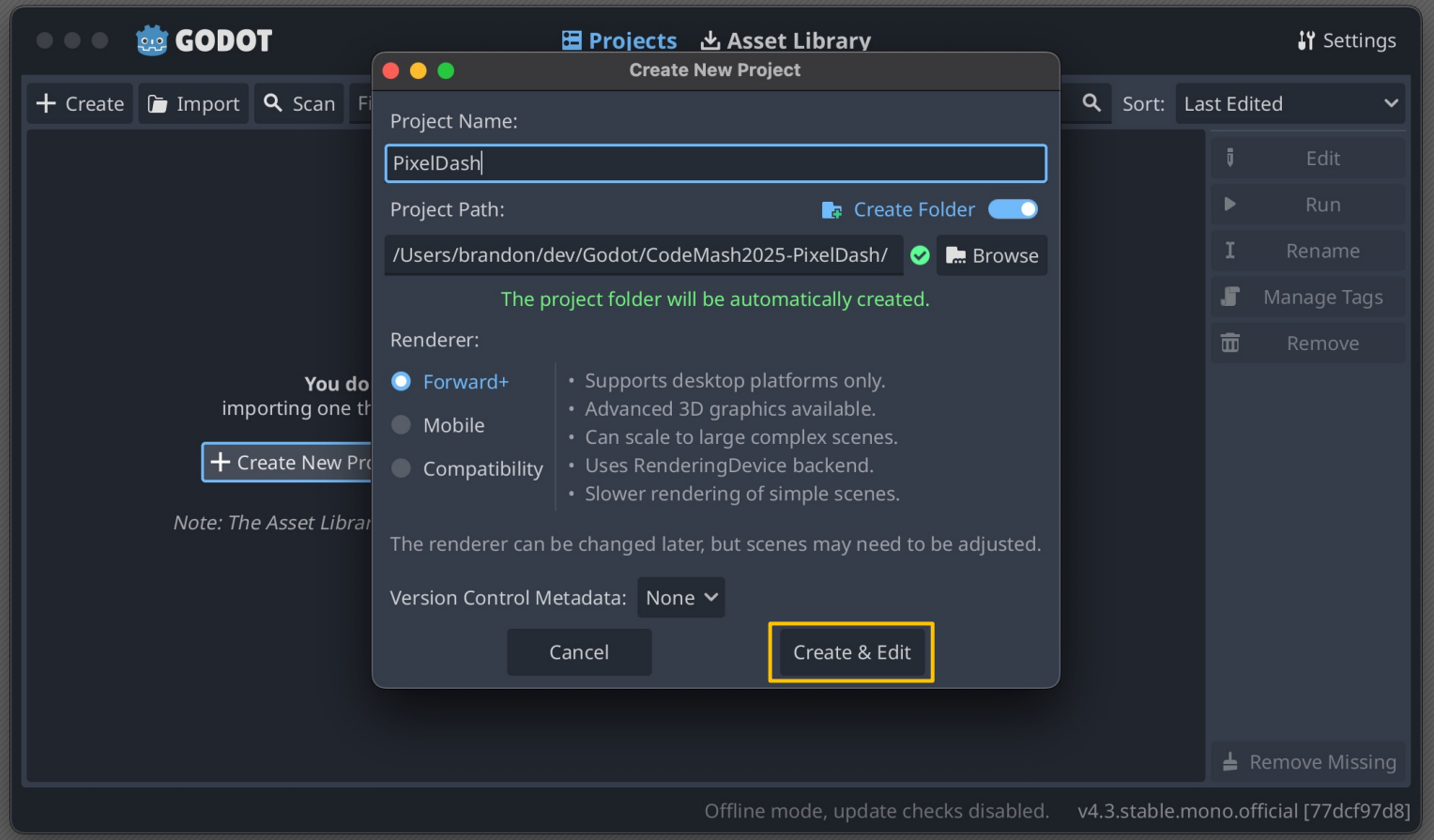
Creating a New Project

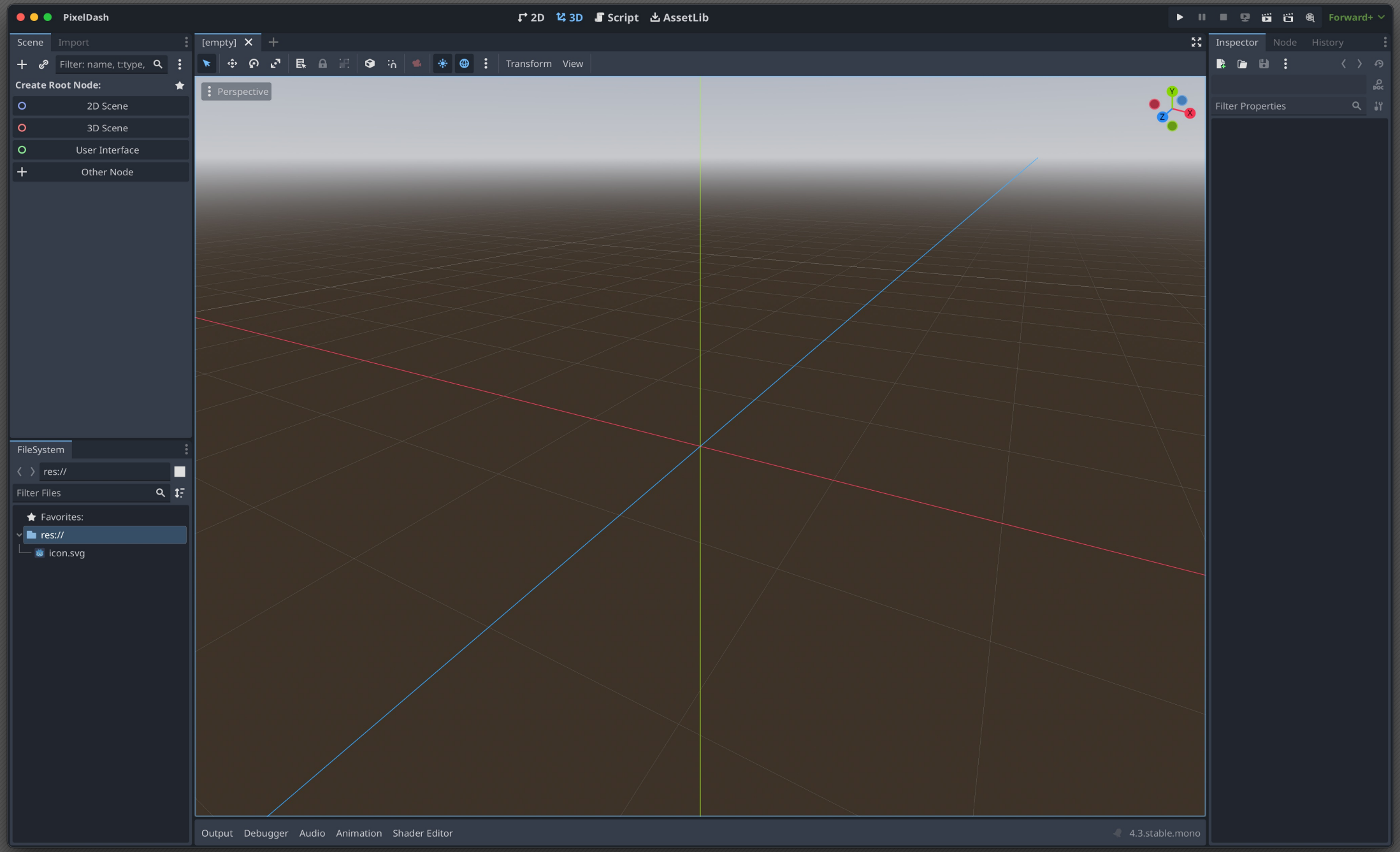
Session Resources: https://github.com/brandon-lewis/CodeMash2025-PixelDash

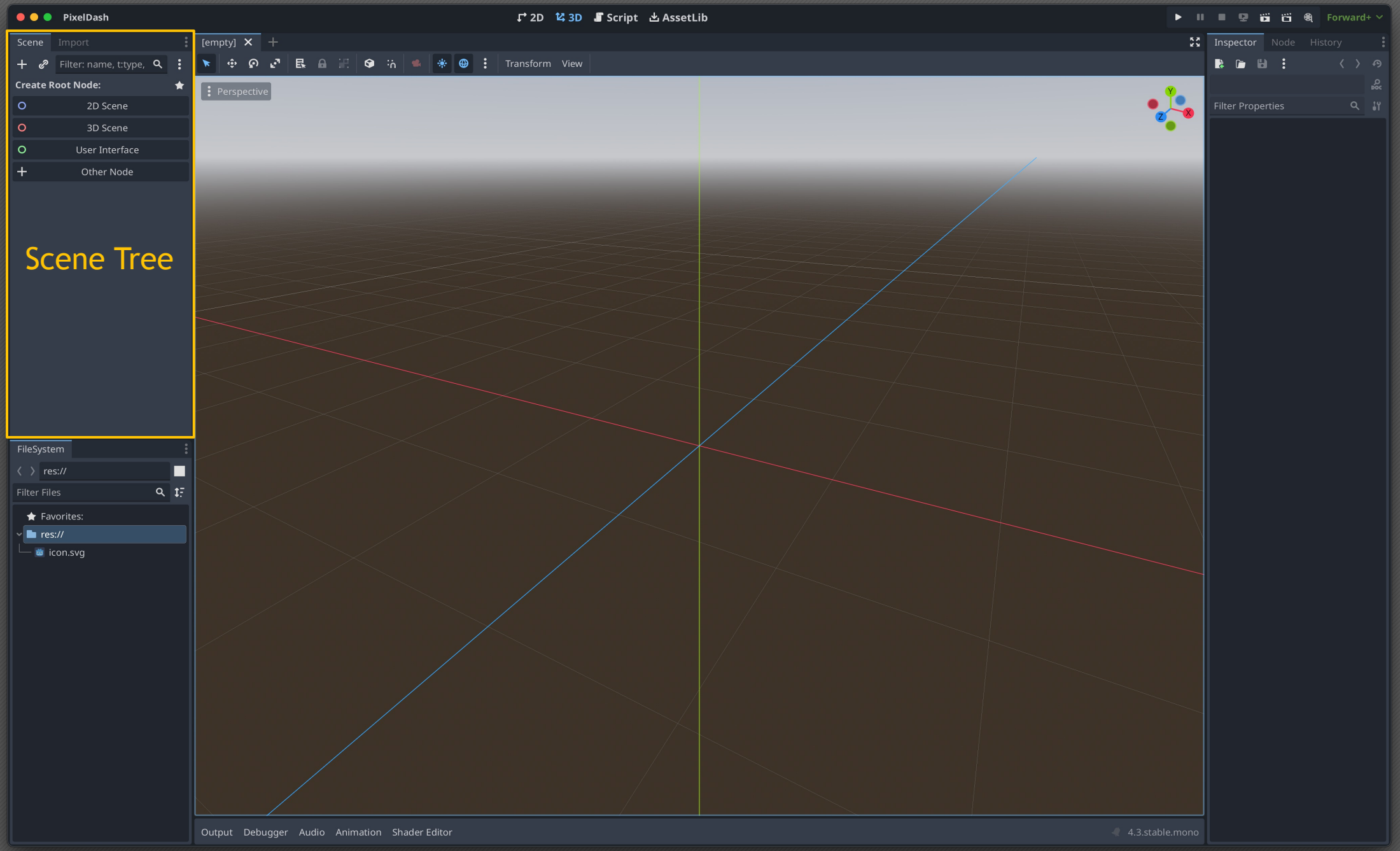


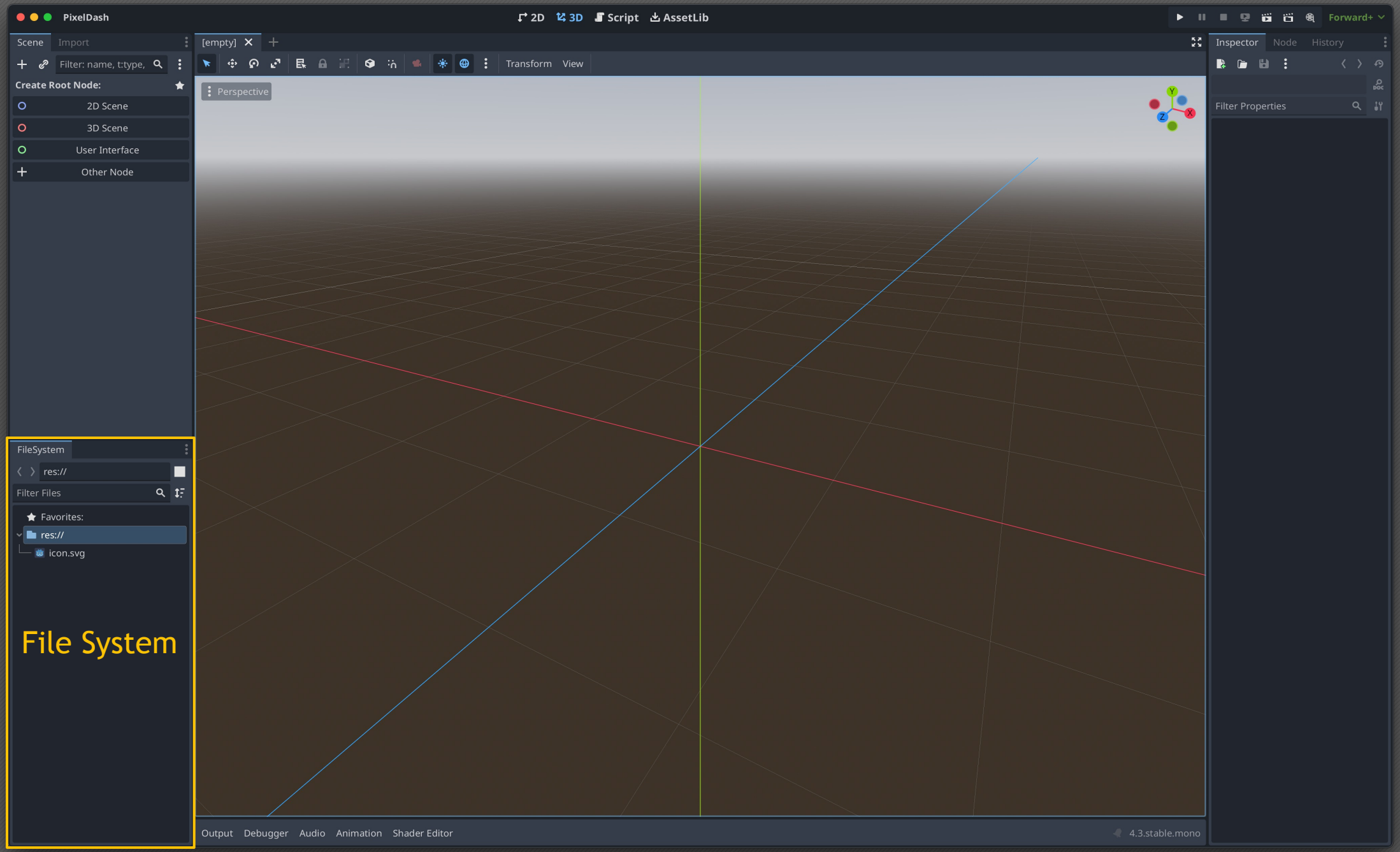
Creating a New Project

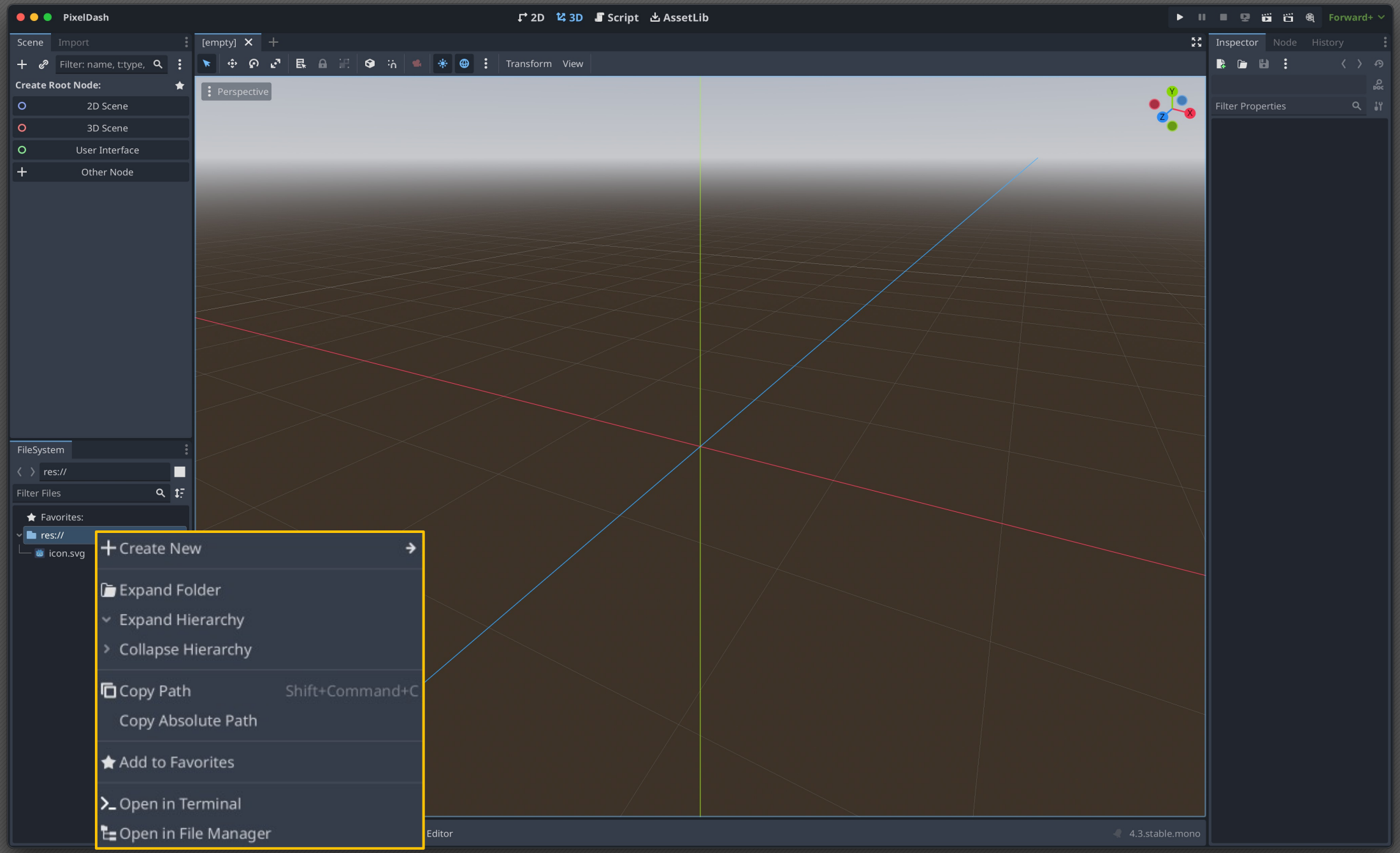
Session Resources: https://github.com/brandon-lewis/CodeMash2025-PixelDash




PixelDash
2D
3D
Script
AssetLib
Forward+
Scene
Import
Filter: name, t:type,
Create Root Node:
2D Scene
3D Scene
User Interface
Other Node
[empty]
Transform
View
Perspective
Inspector
Node
History
Filter Properties
FileSystem
res://
Filter Files
Favorites:
res://
icon.svg
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono

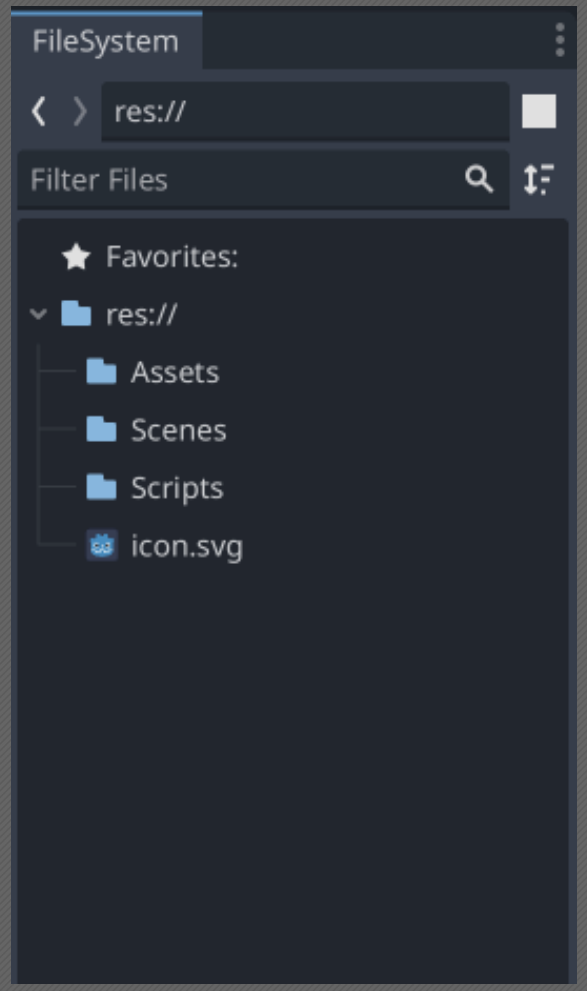

PixelDash
2D
3D
Script
AssetLib
Forward+
Scene
Import
Filter: name, t:type,
Create Root Node:
2D Scene
3D Scene
User Interface
Other Node
Scene Tree
[empty]
Transform
View
Perspective
Inspector
Node
History
Filter Properties
FileSystem
res://
Filter Files
Favorites:
res://
icon.svg
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono


PixelDash
2D
3D
Script
AssetLib
Forward+
Scene
Import
Filter: name, t:type,
Create Root Node:
2D Scene
3D Scene
User Interface
Other Node
[empty]
Transform
View
Perspective
Inspector
Node
History
Filter Properties
FileSystem
res://
Filter Files
Favorites:
res://
icon.svg
File System
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono

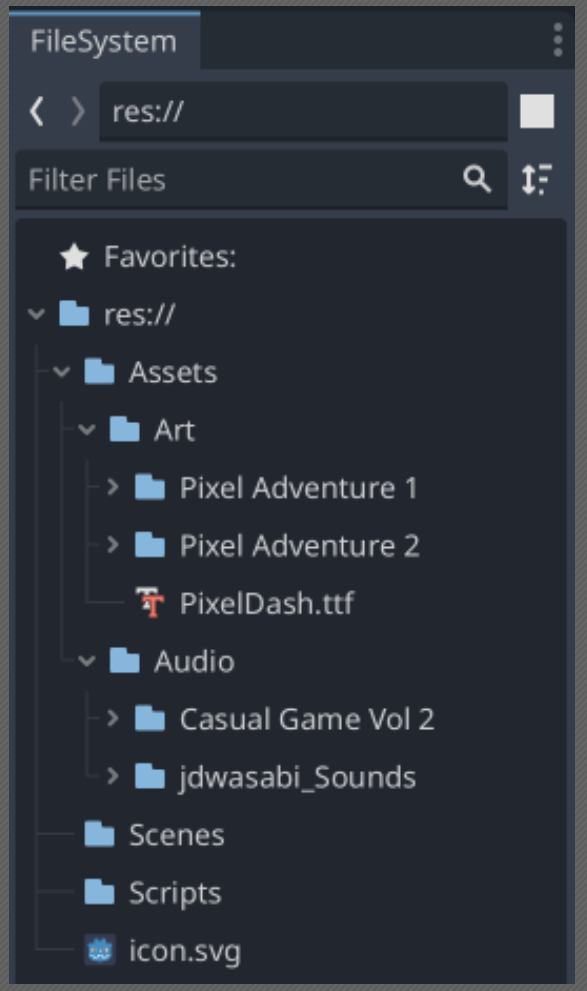

PixelDash
2D
3D
Script
AssetLib
Forward+
Scene
Import
Filter: name, t:type,
Create Root Node:
2D Scene
3D Scene
User Interface
Other Node
[empty]
Transform
View
Perspective
Inspector
Node
History
Filter Properties
FileSystem
res://
Filter Files
Favorites:
res://
icon.svg
Create New
Expand Folder
Expand Hierarchy
Collapse Hierarchy
Copy Path
Shift+Command+C
Copy Absolute Path
Add to Favorites
Open in Terminal
Open in File Manager
Editor
4.3.stable.mono

Create Folders & Import Files

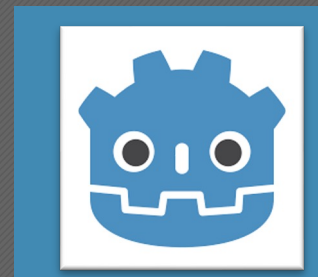
Session Resources: *https://github.com/brandon-lewis/CodeMash2025-PixelDash*



DRAG & DROP
FROM OS
FILE SYSTEM

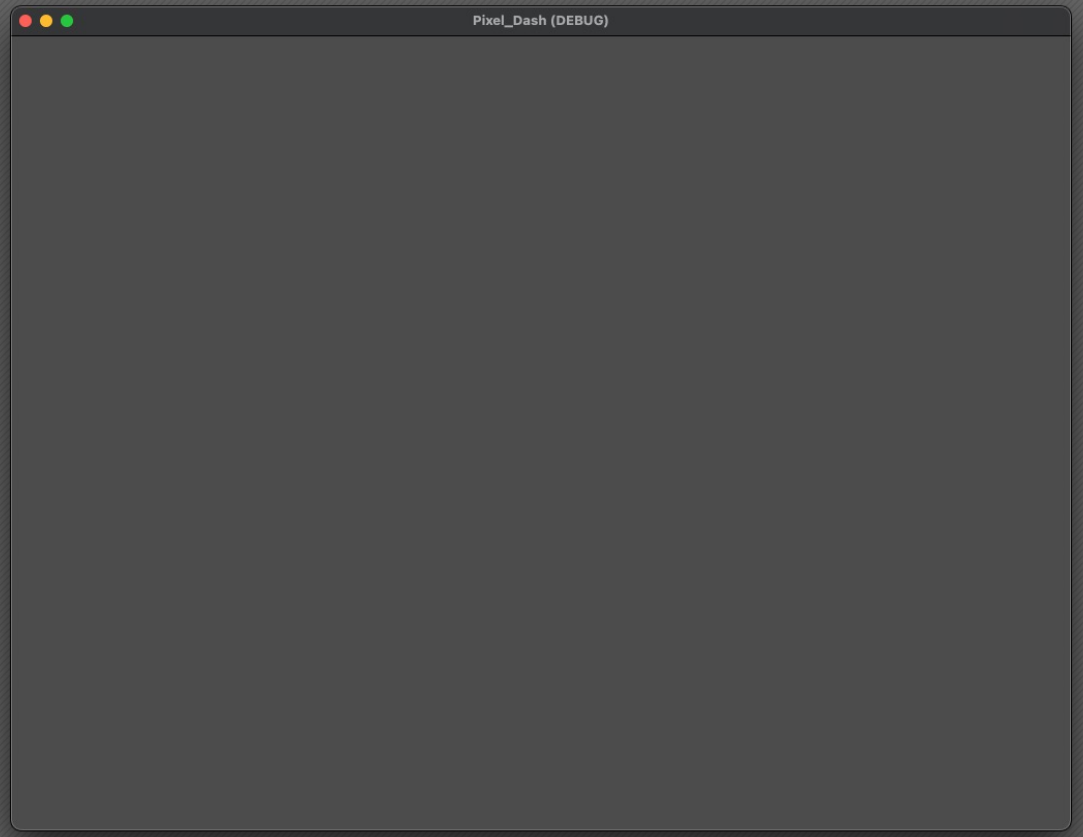


Lab Time (~5 Minutes)

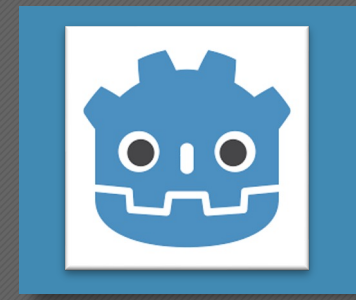


Session Resources: *https://github.com/brandon-lewis/CodeMash2025-PixelDash*

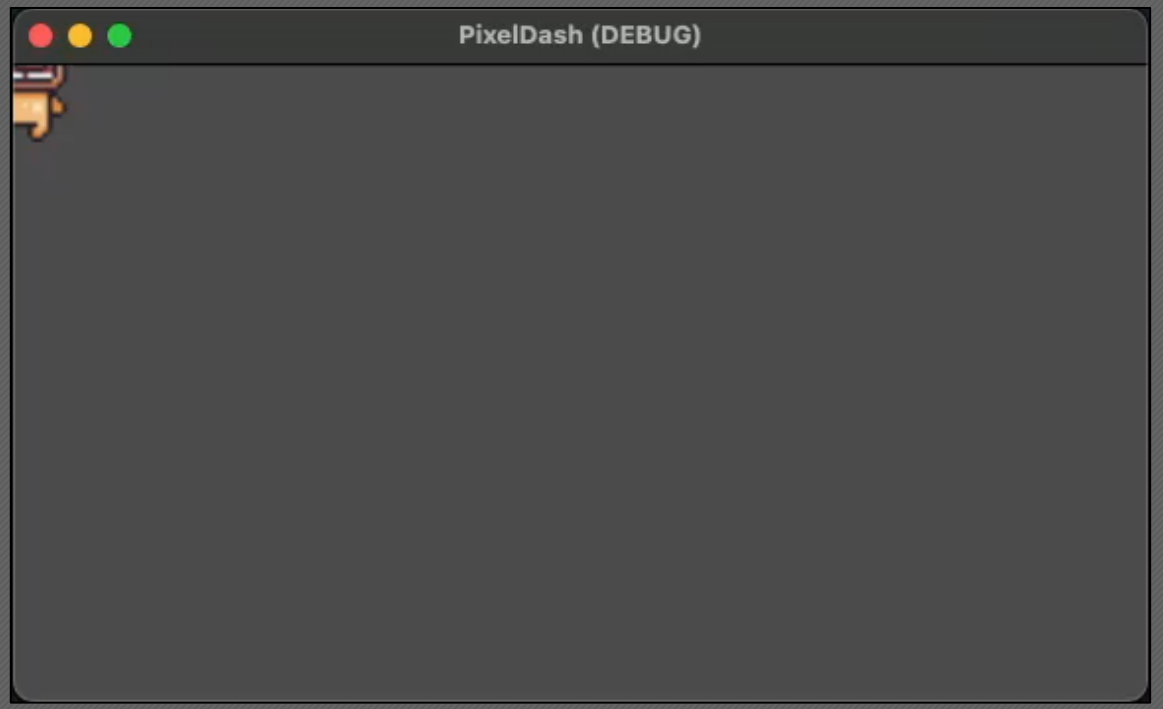
- Download workshop dependencies.
- Create the new project.
- Create base folders and import assets.
- Click around and get a feel for the editor.

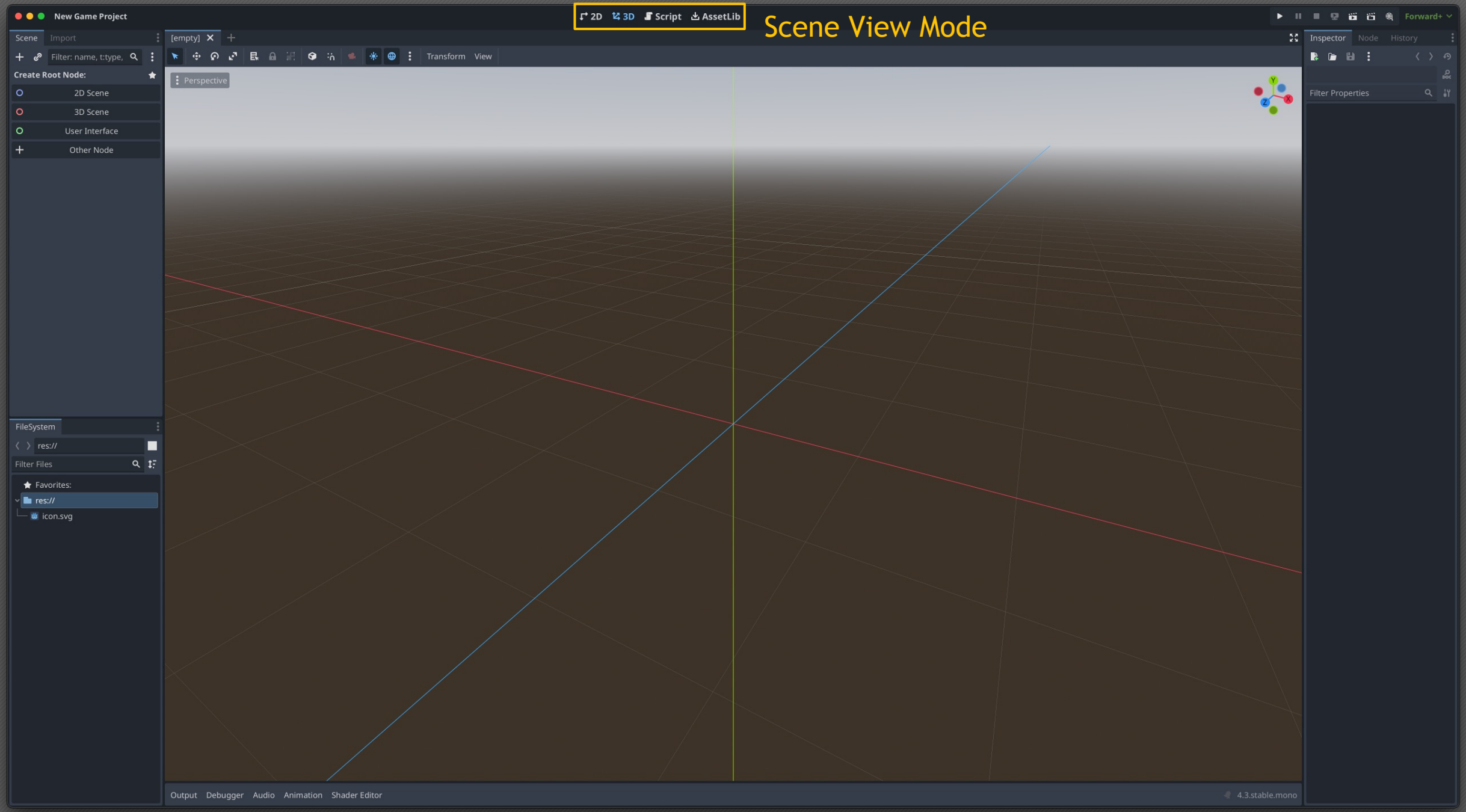


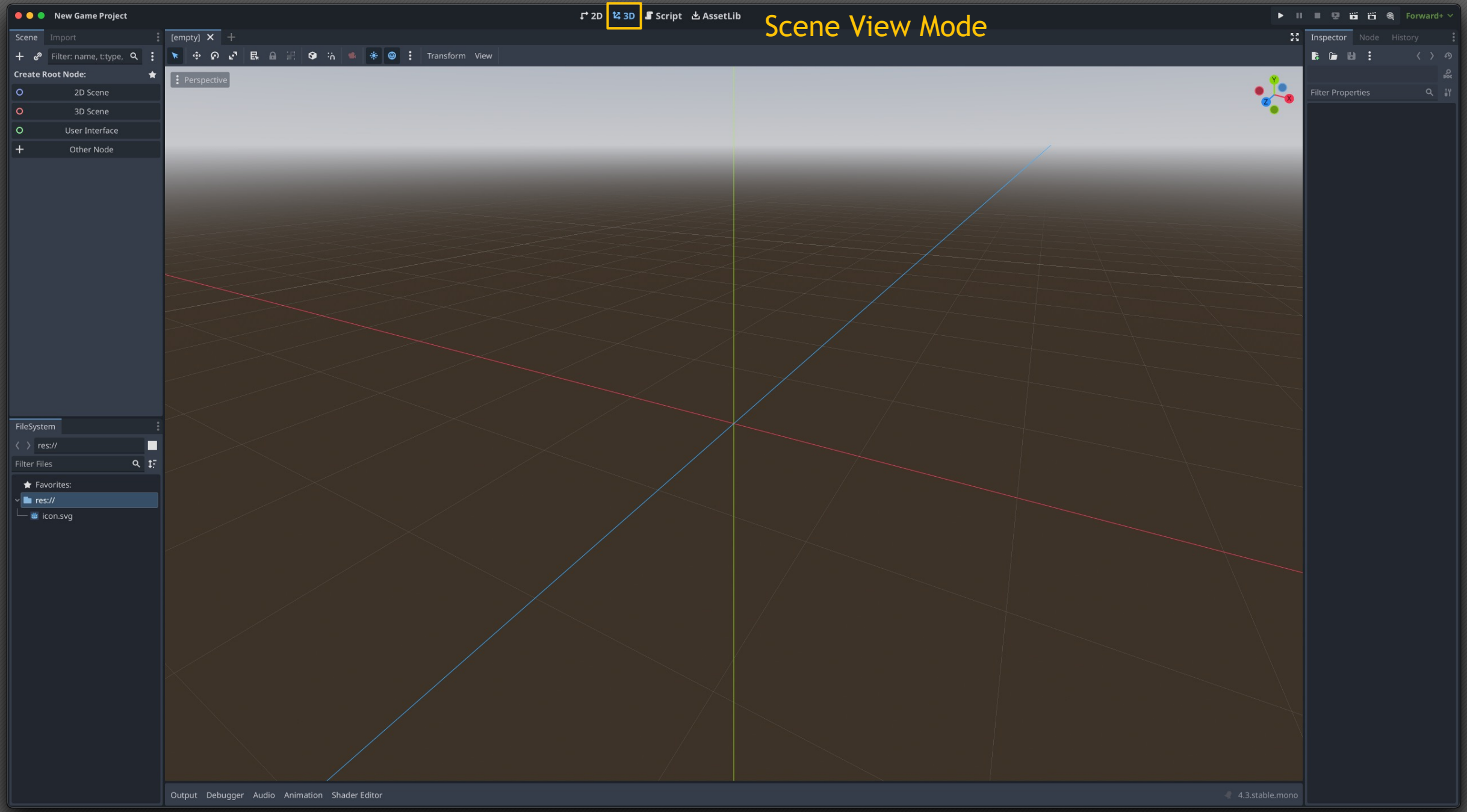
Workshop Goal #1

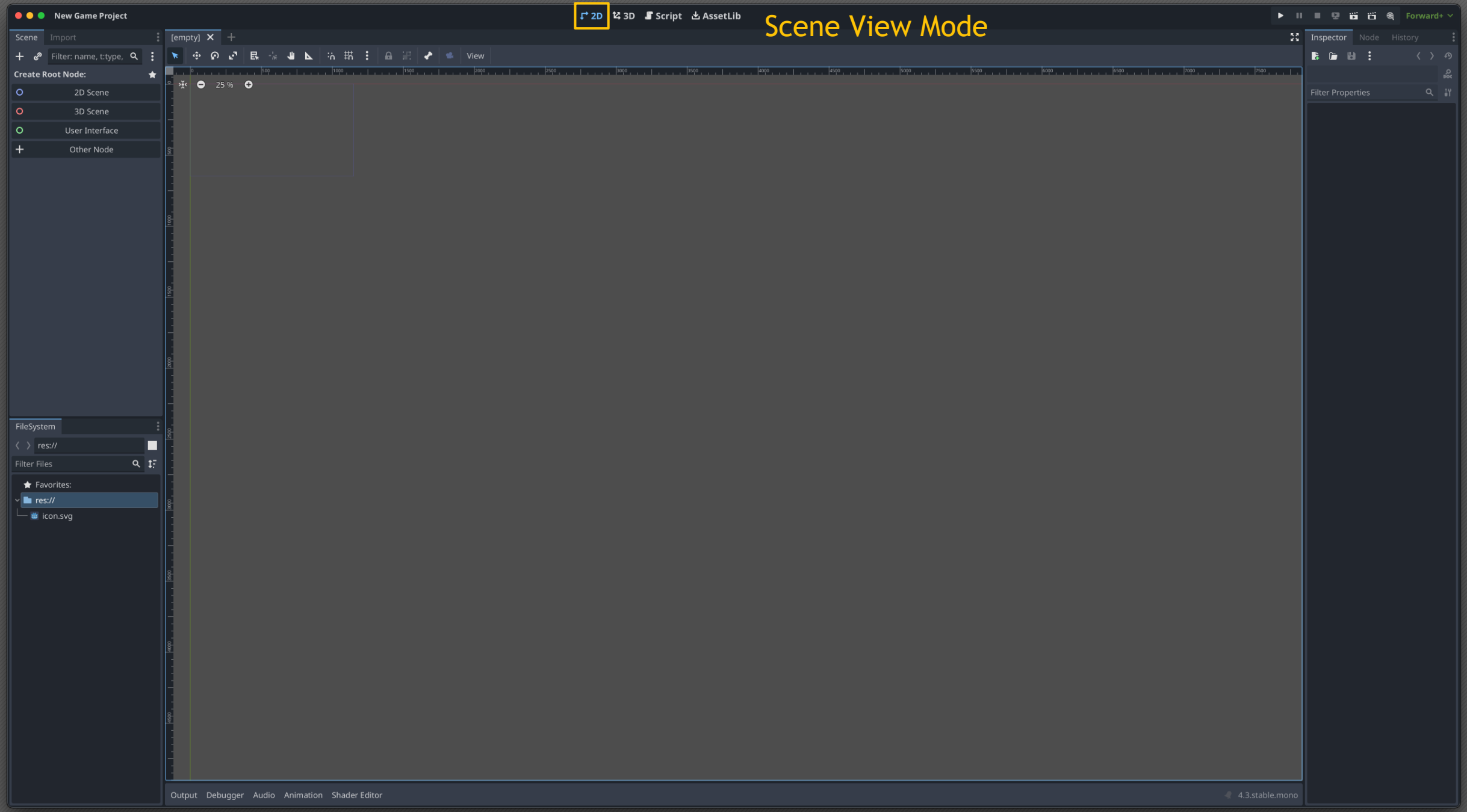


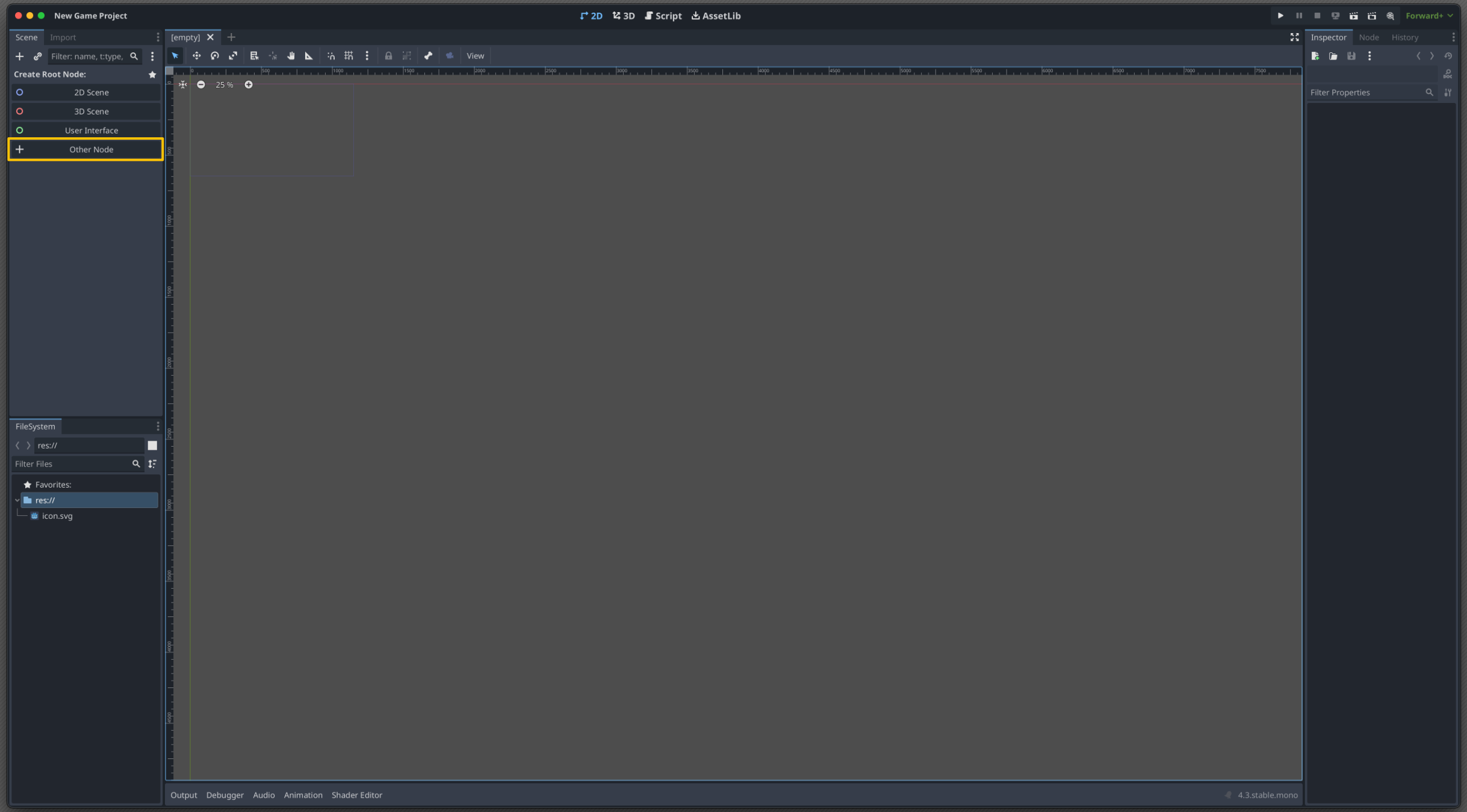
1. Control & Animate a Character
2. Build a Level with Tiles
3. Add Player & Level Polish
   (Camera, Gravity, Sound, Background)
4. Detect Collisions with Trampolines
5. Create a basic Mushroom Enemy
6. Add Collectible Fruit and HUD Display
7. Respawn Character when Defeated
8. Create a Main Menu & Change Scenes
9. Open Workshop, Tinker Time

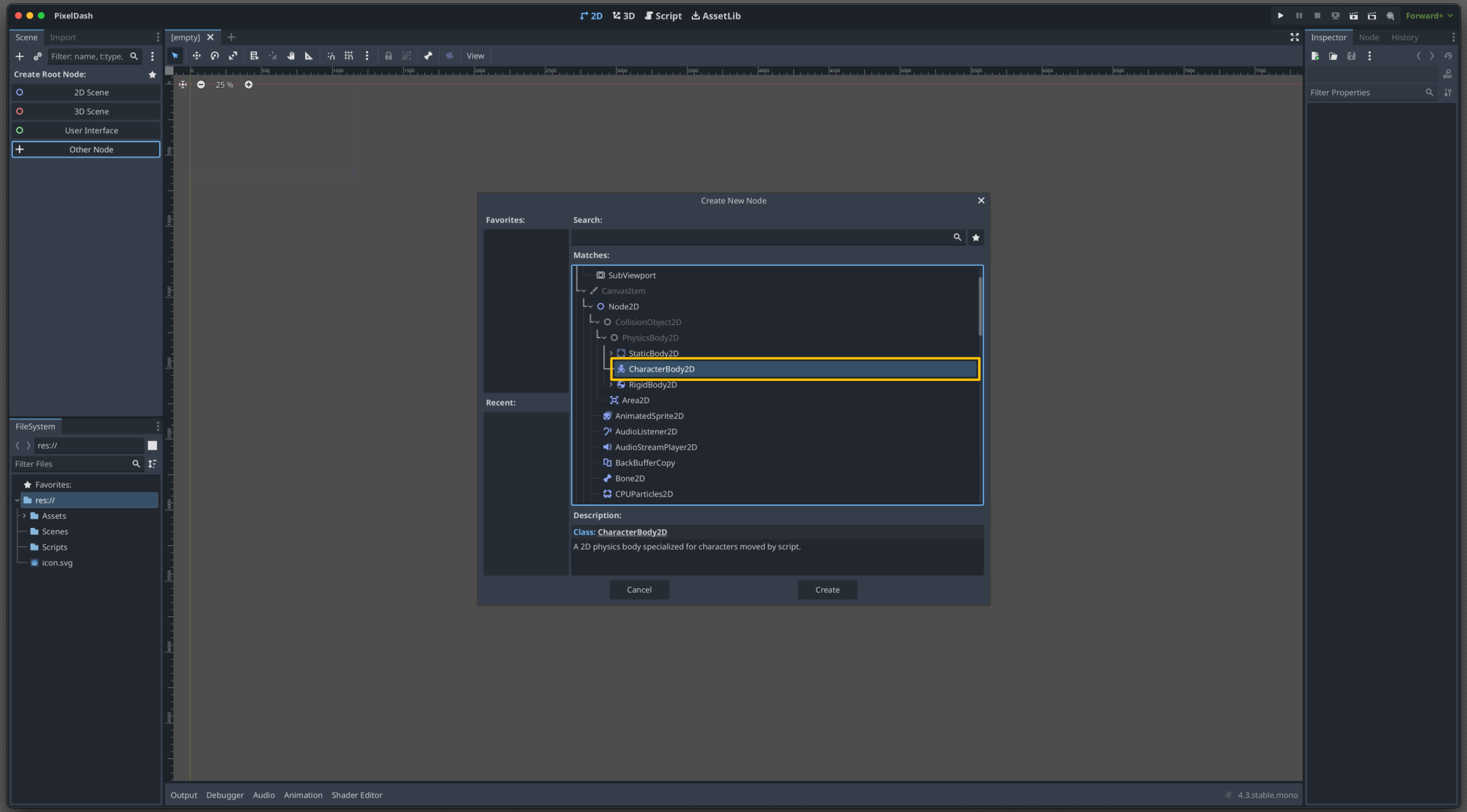


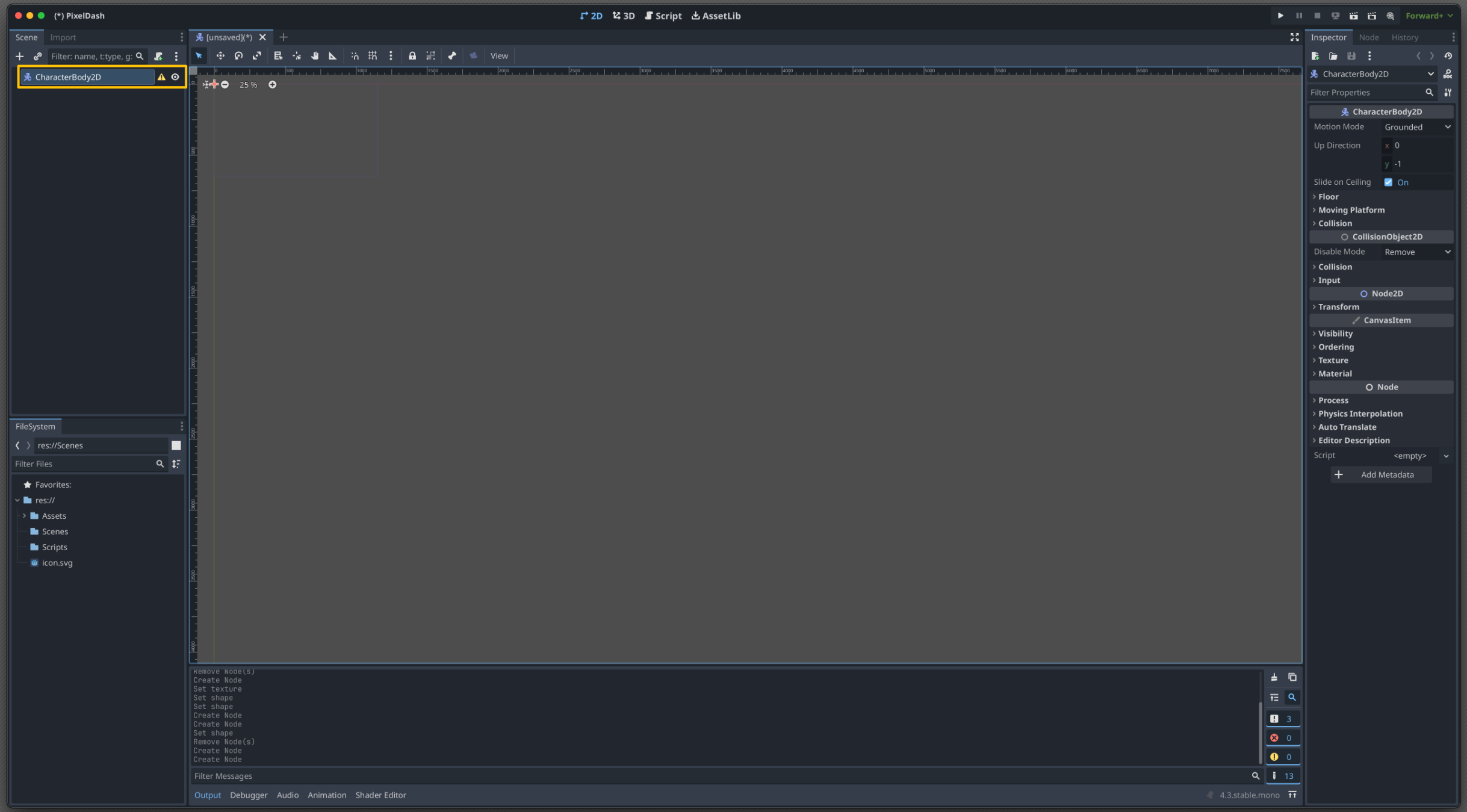

New Game Project
2D
3D
Script
AssetLib
Scene View Mode
Forward+
Scene
Import
Filter: name, t:type,
Create Root Node:
2D Scene
3D Scene
User Interface
Other Node
[empty]
Transform
View
Perspective
Inspector
Node
History
Filter Properties
FileSystem
res://
Filter Files
Favorites:
res://
icon.svg
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono


New Game Project
2D
3D
Script
AssetLib
Scene View Mode
Forward+
Scene
Import
Filter: name, t:type,
Create Root Node:
2D Scene
3D Scene
User Interface
Other Node
[empty]
Transform
View
Perspective
Inspector
Node
History
Filter Properties
FileSystem
res://
Filter Files
Favorites:
res://
icon.svg
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono

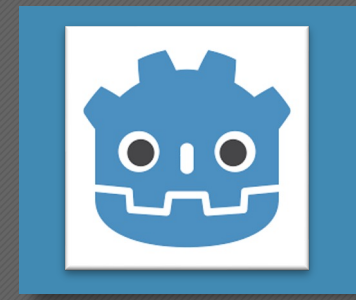

New Game Project
2D
3D
Script
AssetLib
Scene View Mode
Forward+
Scene
Import
Filter: name, t:type,
Create Root Node:
2D Scene
3D Scene
User Interface
Other Node
[empty]
View
25 %
Inspector
Node
History
Filter Properties
FileSystem
res://
Filter Files
Favorites:
res://
icon.svg
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono


New Game Project
2D
3D
Script
AssetLib
Forward+
Scene
Import
Filter: name, t:type,
Create Root Node:
2D Scene
3D Scene
User Interface
Other Node
[empty]
View
25 %
Inspector
Node
History
Filter Properties
FileSystem
res://
Filter Files
Favorites:
res://
icon.svg
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono

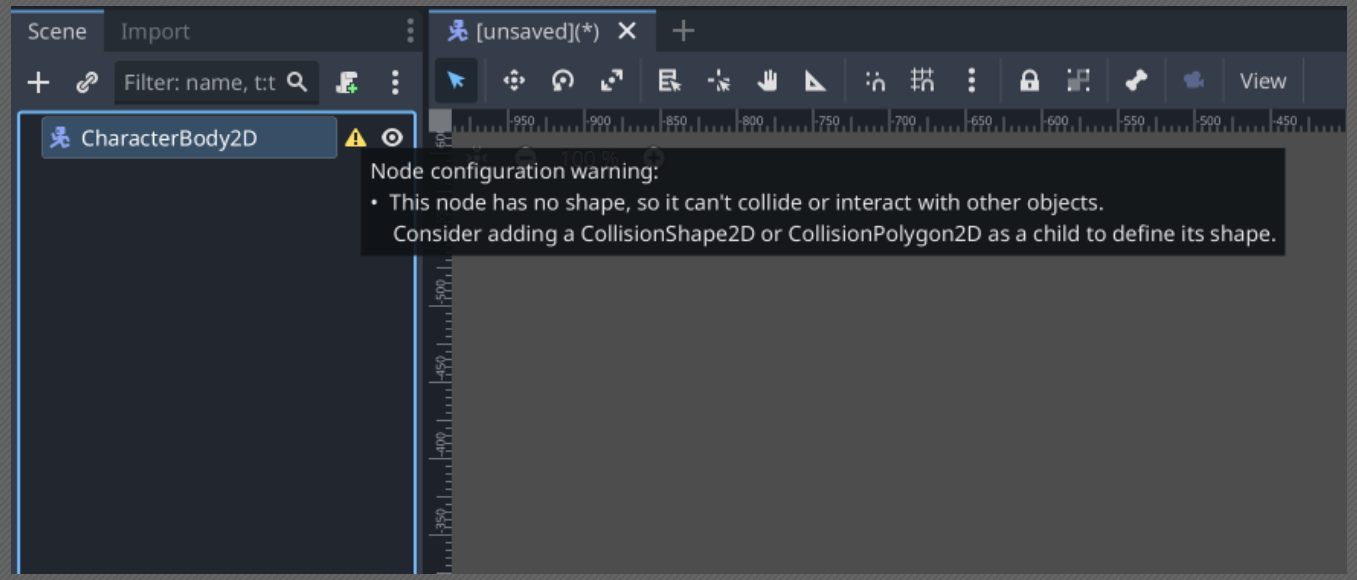

PixelDash
2D
3D
Script
AssetLib
Forward+
Scene
Import
Filter: name, t:type,
Create Root Node:
2D Scene
3D Scene
User Interface
Other Node
[empty]
View
25 %
Inspector
Node
History
Filter Properties
FileSystem
res://
Filter Files
Favorites:
res://
Assets
Scenes
Scripts
icon.svg
Create New Node
Favorites:
Recent:
Search:
Matches:
SubViewport
CanvasItem
Node2D
CollisionObject2D
PhysicsBody2D
StaticBody2D
CharacterBody2D
RigidBody2D
Area2D
AnimatedSprite2D
AudioListener2D
AudioStreamPlayer2D
BackBufferCopy
Bone2D
CPUParticles2D
Description:
Class: CharacterBody2D
A 2D physics body specialized for characters moved by script.
Cancel
Create
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono


(*) PixelDash
2D
3D
Script
AssetLib
Forward+
Scene
Import
Filter: name, t:type, g:
CharacterBody2D
[unsaved](*)
View
25 %
Inspector
Node
History
CharacterBody2D
Filter Properties
CharacterBody2D
Motion Mode
Grounded
Up Direction
x 0
y -1
Slide on Ceiling
On
Floor
Moving Platform
Collision
CollisionObject2D
Disable Mode
Remove
Collision
Input
Node2D
Transform
CanvasItem
Visibility
Ordering
Texture
Material
Node
Process
Physics Interpolation
Auto Translate
Editor Description
Script
<empty>
Add Metadata
FileSystem
res://Scenes
Filter Files
Favorites:
res://
Assets
Scenes
Scripts
icon.svg
Remove Node(s)
Create Node
Set texture
Set shape
Set shape
Create Node
Create Node
Set shape
Remove Node(s)
Create Node
Create Node
Filter Messages
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono

Adding Collision to a CharacterBody2D

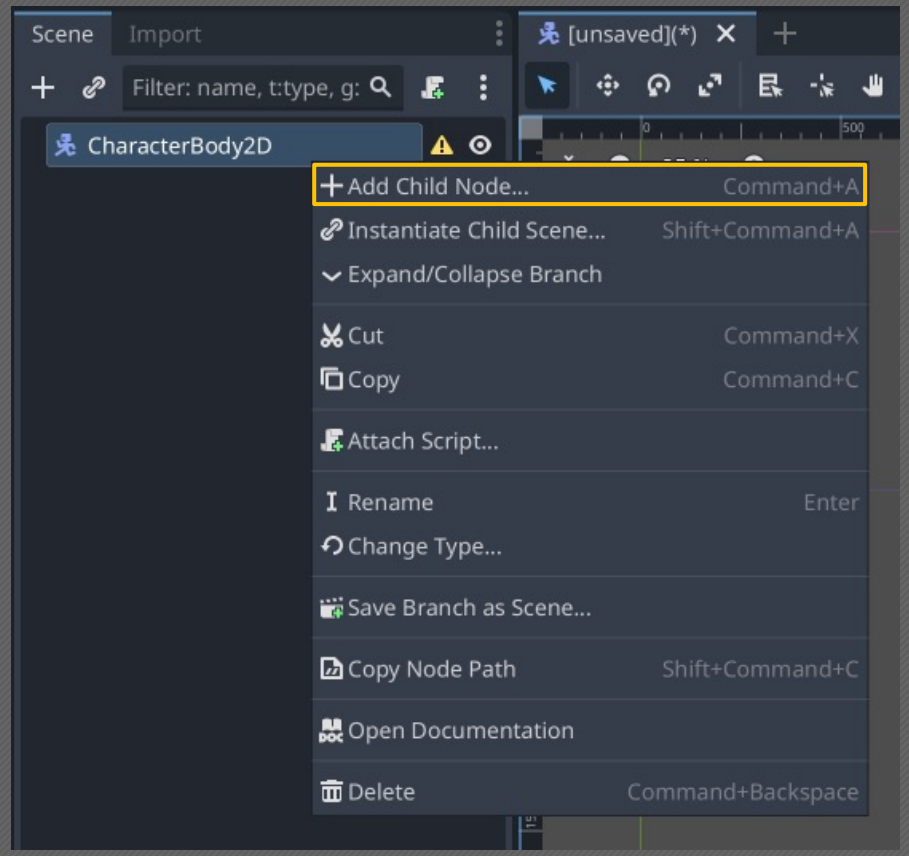


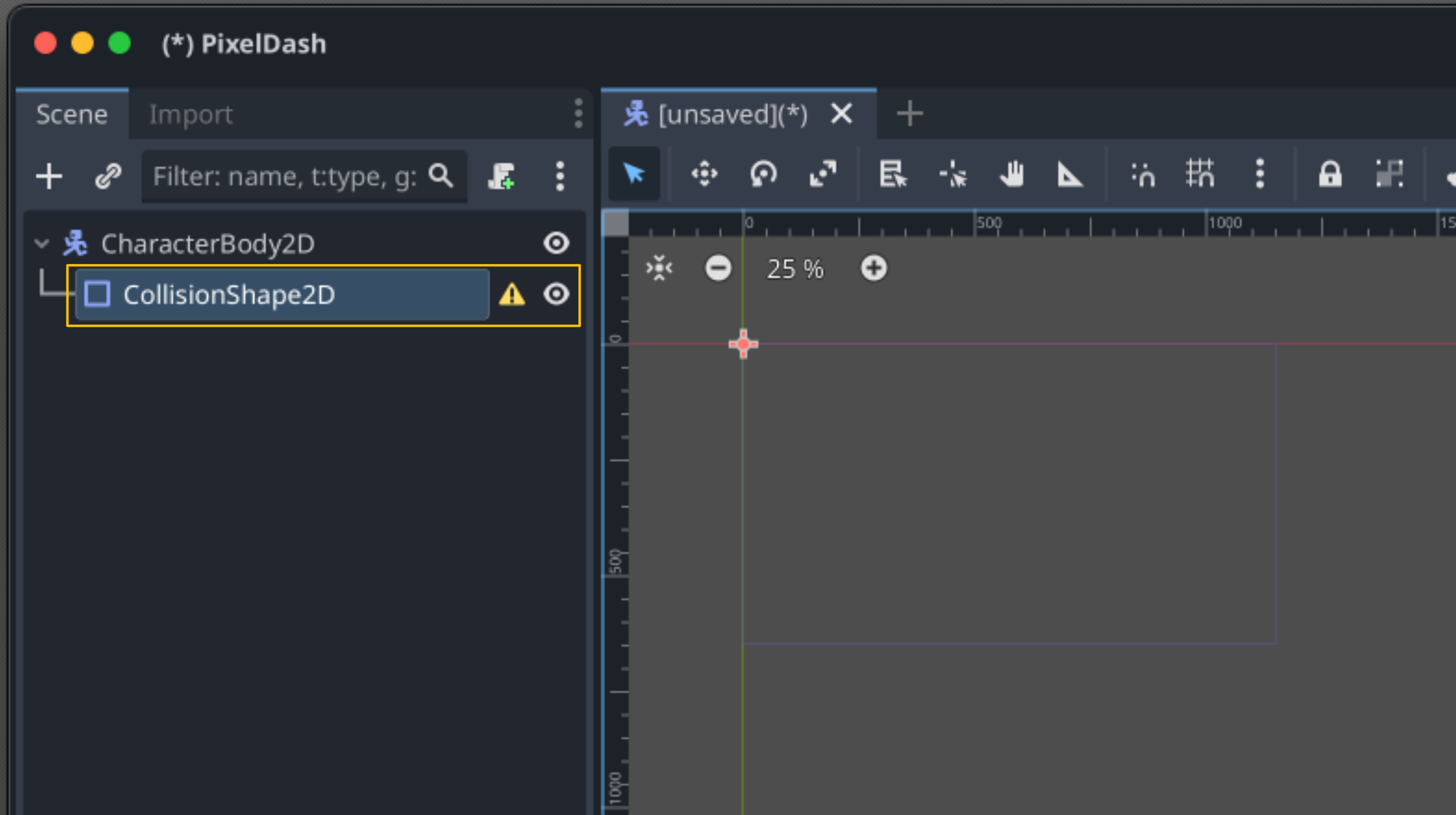
- A CharacterBody2D is a type of physics body that is meant to be moved and controlled via script.
- CharacterBody2Ds are not affected by physics, but they affect other physics bodies in their path.
- A physics body requires a CollisionShape to interact with other physics bodies.

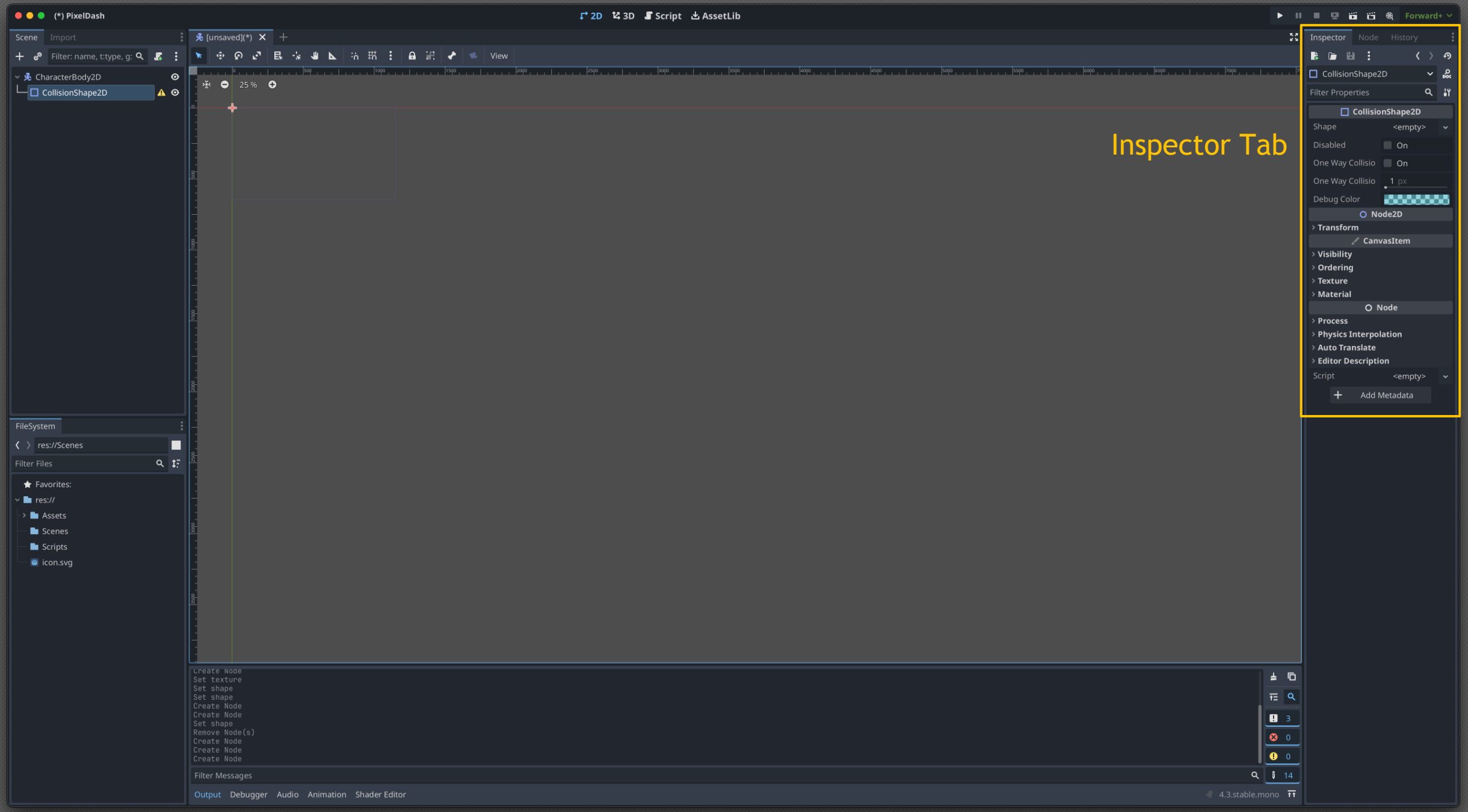


Add Child Node to the root CharacterBody2D

- A Scene in Godot is composed of base Nodes and other Scenes.
- Right-click the root node in the Scene Tree and select Add Child Node to add a new child; add a CollisionShape2D node.
- A Scene may contain one and only one root/top-level node.




(*) PixelDash
Scene
Import
Filter: name, t:type, g:
CharacterBody2D
CollisionShape2D
[unsaved](*)
25 %
0
500
1000
500
1000


(*) PixelDash
2D
3D
Script
AssetLib
Forward+
Scene
Import
Filter: name, t:type, g:
CharacterBody2D
CollisionShape2D
[unsaved](*)
View
25 %
Inspector Tab
Inspector
Node
History
CollisionShape2D
Filter Properties
CollisionShape2D
Shape
<empty>
Disabled
On
One Way Collisio
On
One Way Collisio
1 px
Debug Color
Node2D
Transform
CanvasItem
Visibility
Ordering
Texture
Material
Node
Process
Physics Interpolation
Auto Translate
Editor Description
Script
<empty>
Add Metadata
FileSystem
res://Scenes
Filter Files
Favorites:
res://
Assets
Scenes
Scripts
icon.svg
Create Node
Set texture
Set shape
Set shape
Create Node
Create Node
Set shape
Remove Node(s)
Create Node
Create Node
Create Node
Filter Messages
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono

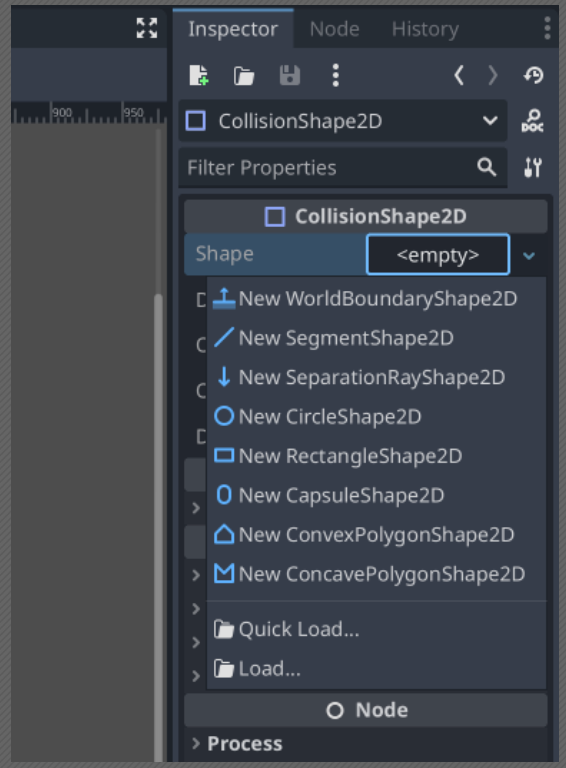
Understanding the Inspector Tab

- The Inspector Tab allows you to configure various variables that exist in the select node's inheritance hierarchy.

- For example; the *CharacterBody2D* inherits from *CollisionObject2D*, which inherits from *Node2D*, etc.

Adding Collision to a CharacterBody2D

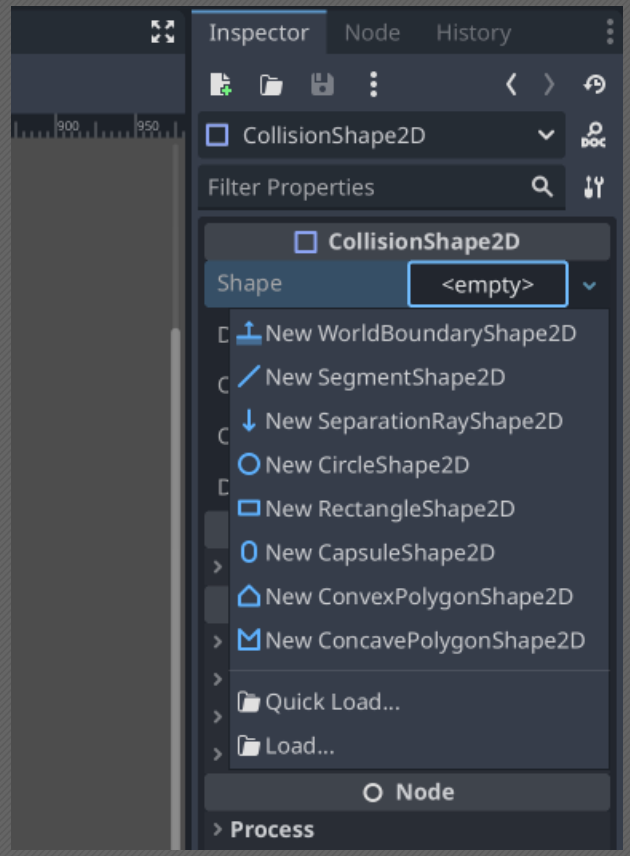
Click the <empty> Shape value dropdown.



Adding Collision to a CharacterBody2D

Click the <empty> Shape value dropdown and choose CapsuleShape2D.

- The CapsuleShape2D can provide a more “slippery” feel around corners.
- The RectangleShape2D will make platforms and corners feel blocky and more forgiving to navigate.

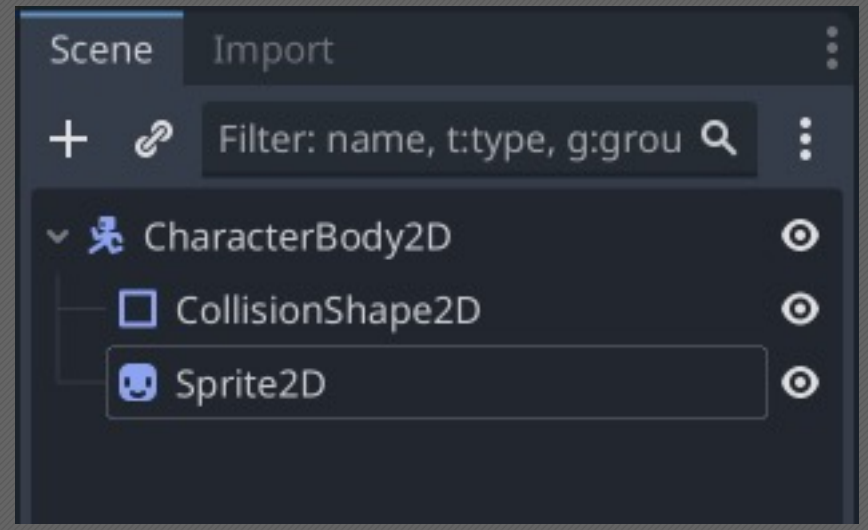


Adding Collision to a CharacterBody2D

- The Stop On Slope value under the Floor section of the CharacterBody2D is On by default.
- This will make your character stick to corners, potentially seeming to hover in the air.
- Toggle this value Off if you want your character to slip or slide off slopes and corners.

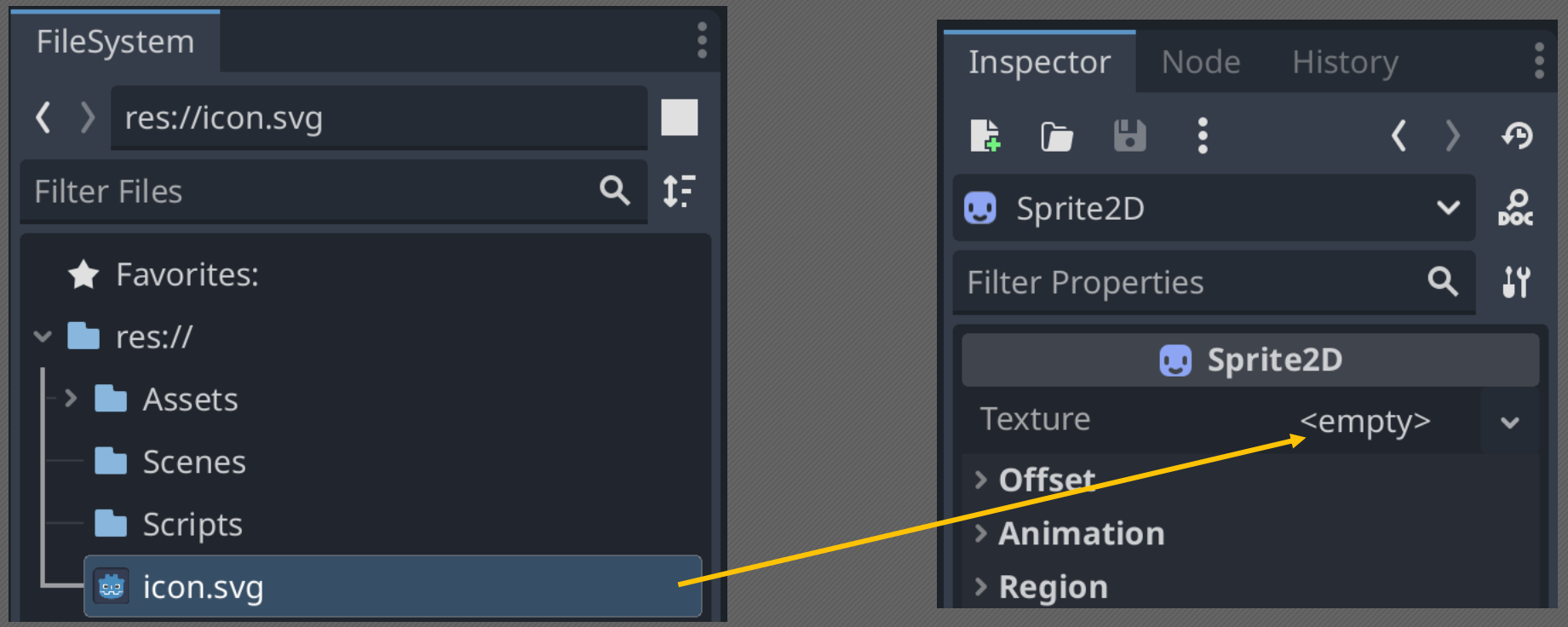
Adding a Sprite

- A Sprite is a single-frame 2D image that represents something/anything in a game.
- Add a new child to the root node of the scene (the CharacterBody2D).
- The new child will be added to the bottom of the scene tree.



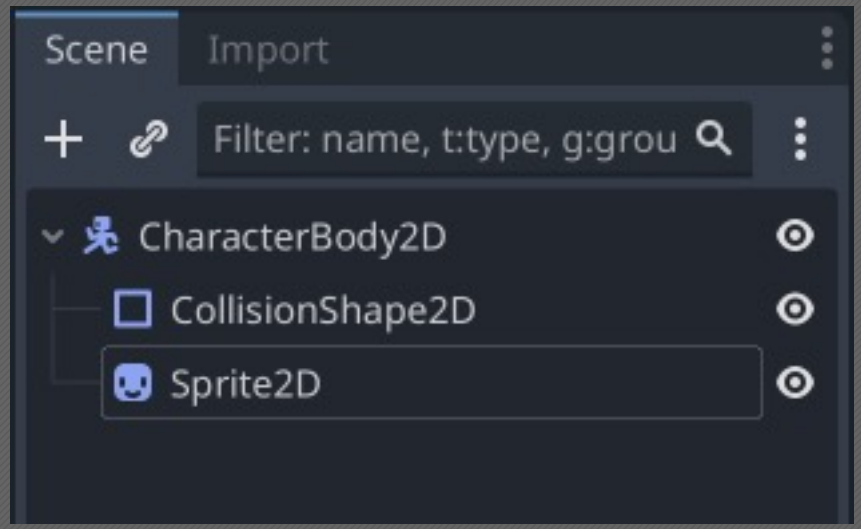
Setting the Sprite Texture

Drag and drop the icon.svg into the Sprite2D's Texture property.



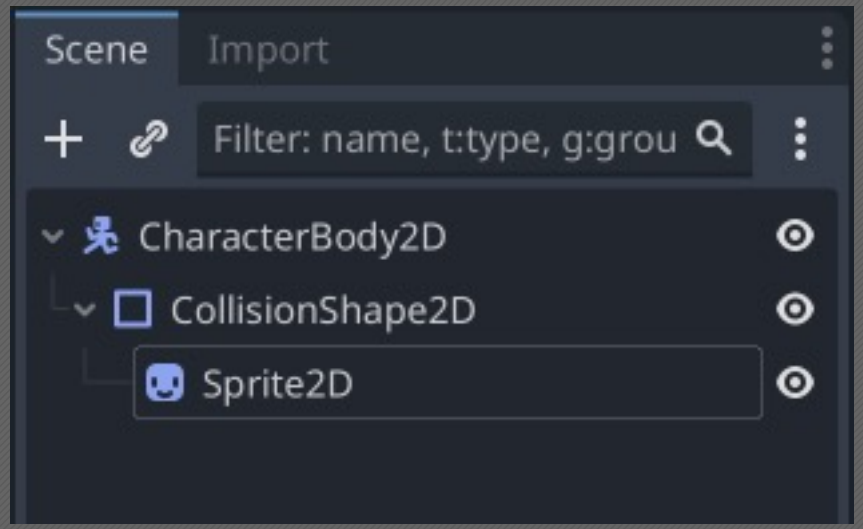
Scene Tree Order Matters

- Visual components of your scene tree are rendered top-to-bottom.
- In our case, the CharacterBody2D will render first, then the CollisionShape2D, then the Sprite2D.
- If we were to add a second Sprite2D underneath the existing node, the second sprite will block view of the Sprite2D above it.

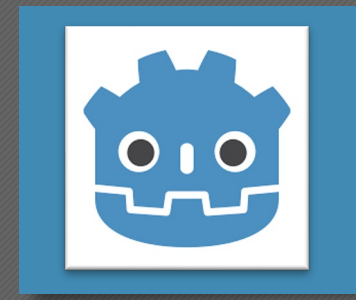


Scene Tree Order Matters

- If you have the CollisionShape2D selected when you add the child, the Sprite2D will be added as a child of the collision shape.
- This, as shown to the right, is incorrect for what we're building.
- You can drag any node around the scene tree to change its parenting.



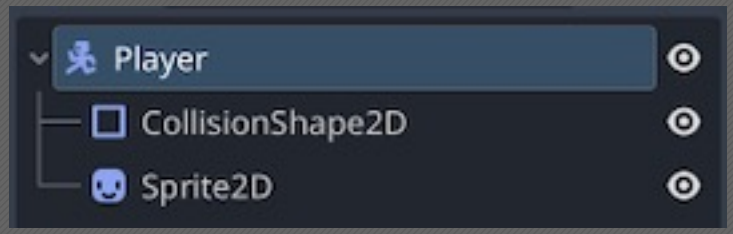
Renaming Nodes



- You can click individual nodes in the scene tree to change the node's name.

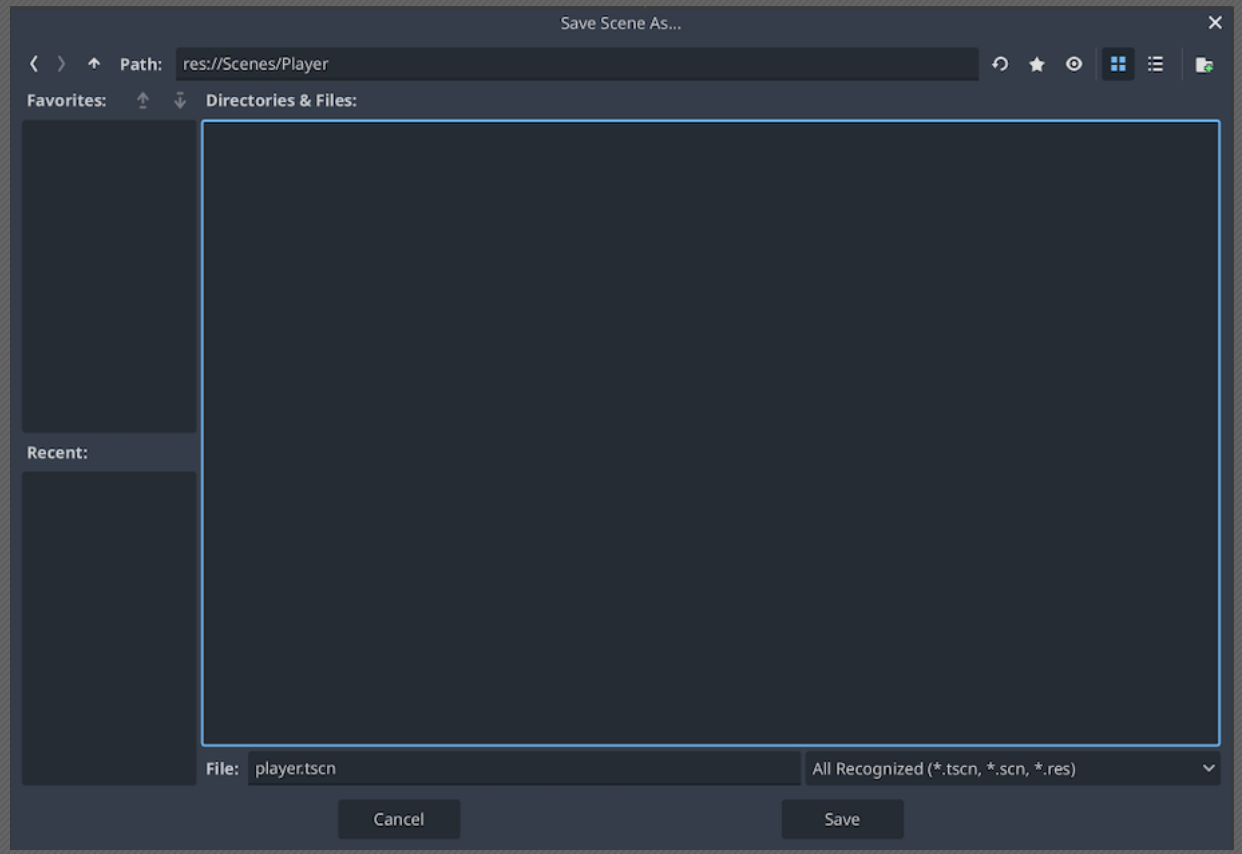
- Rename the CharacterBody2D root node in this scene to "Player".

- There is no need to change the other node names.



Save the “Player” Scene

- Create a new Player folder in the res://Scenes folder.
- The default name for the file is the snake_case version of the name of the scene’s root node.
- Save the scene in this new Scenes/Player folder.

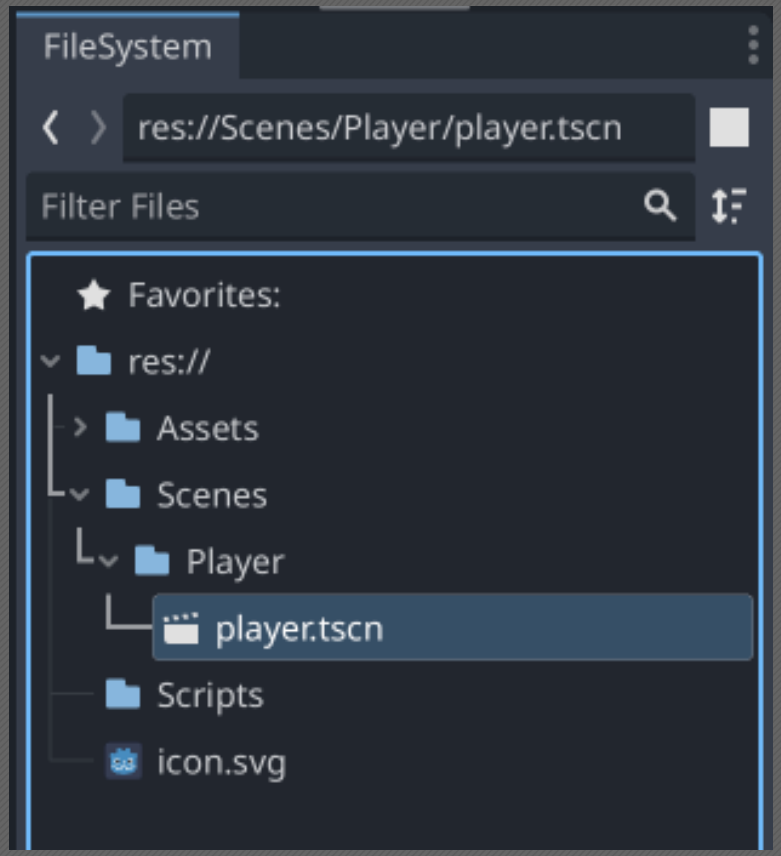


Saved Scene in the FileSystem

You should see the new player.tscn file in your FileSystem view.

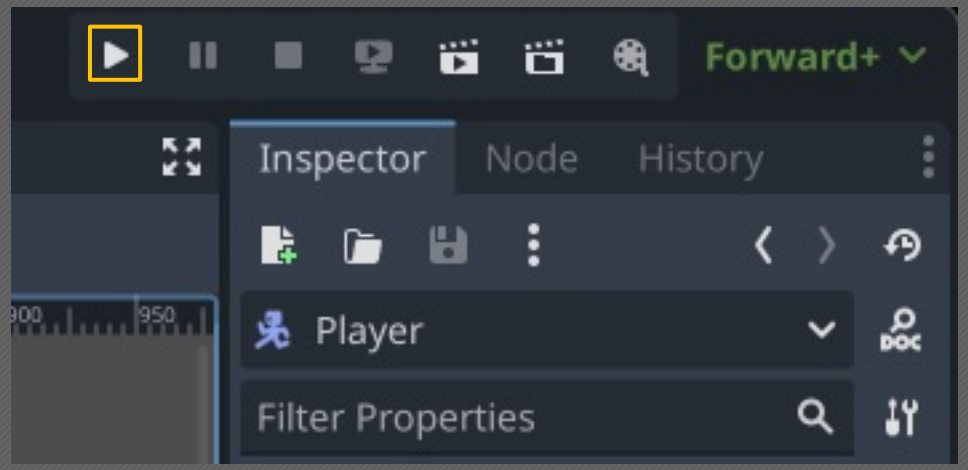
The .tscn extension is short for “text-scene”. Most, if not all, data-points made by Godot are saved in plaintext.

Since they’re plaintext files, Godot is very version-control-friendly.



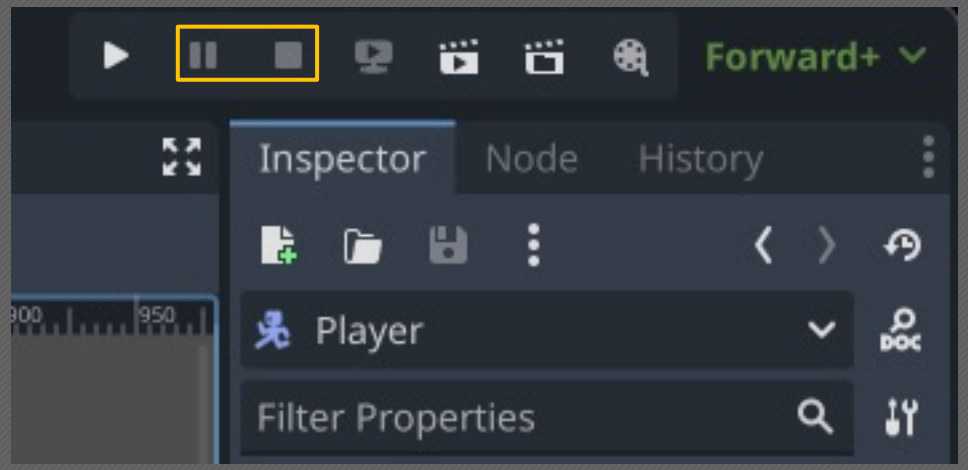
Running the Game

Run the ”Main” Scene

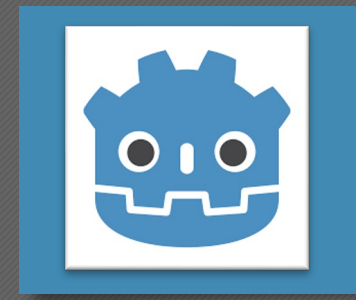


Running the Game

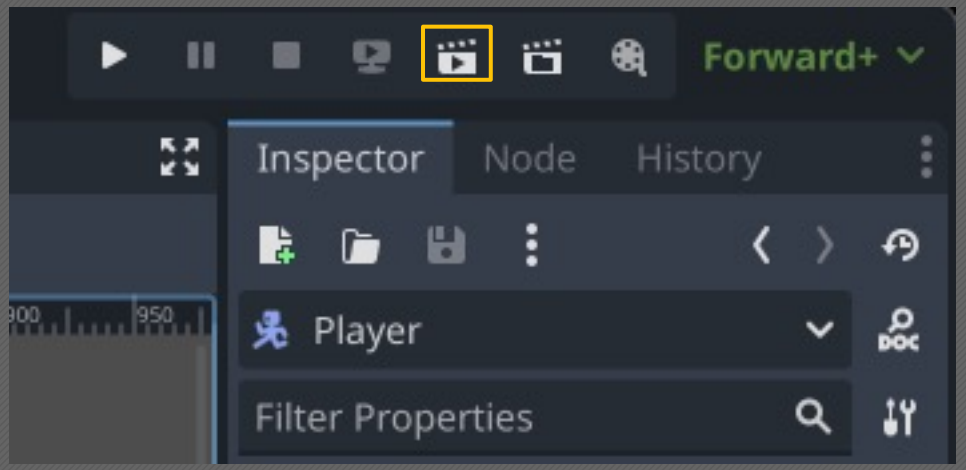
Pause and Stop (when game is running)



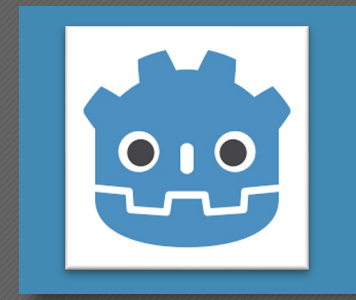
Running the Game



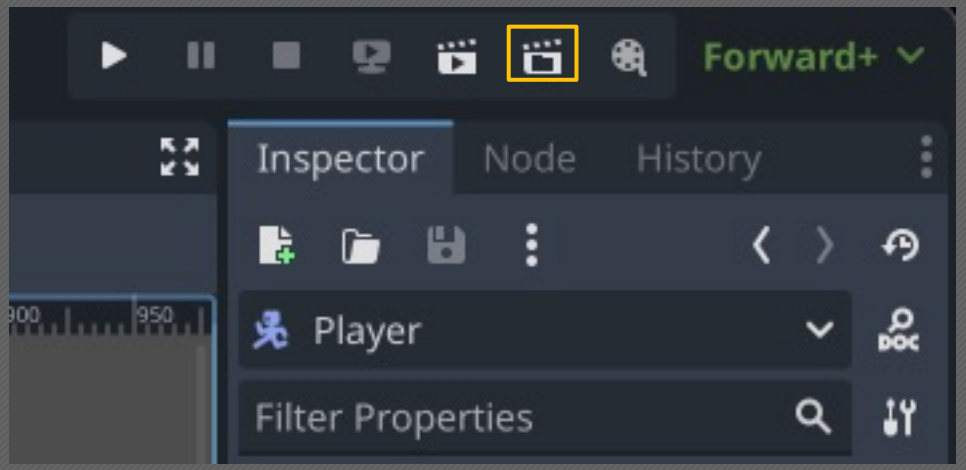
Run Current Scene



Running the Game

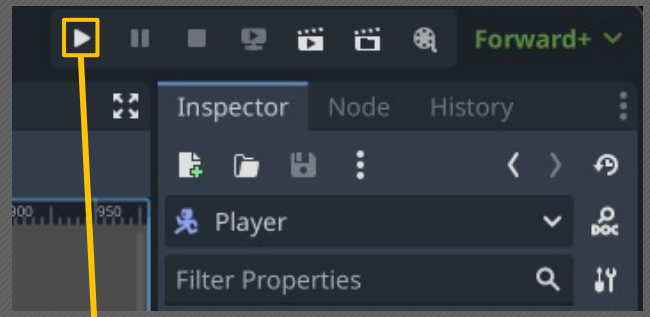


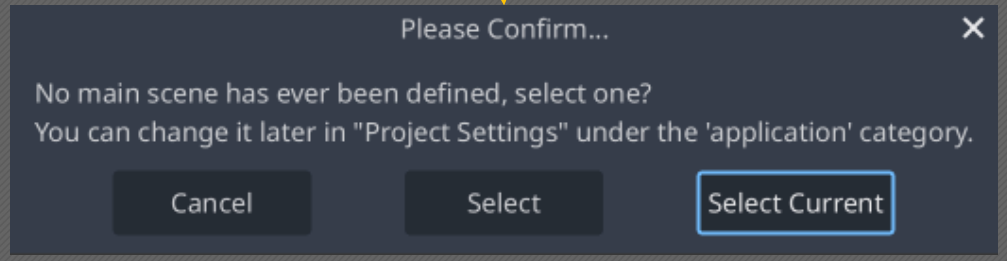
Run Specific Scene



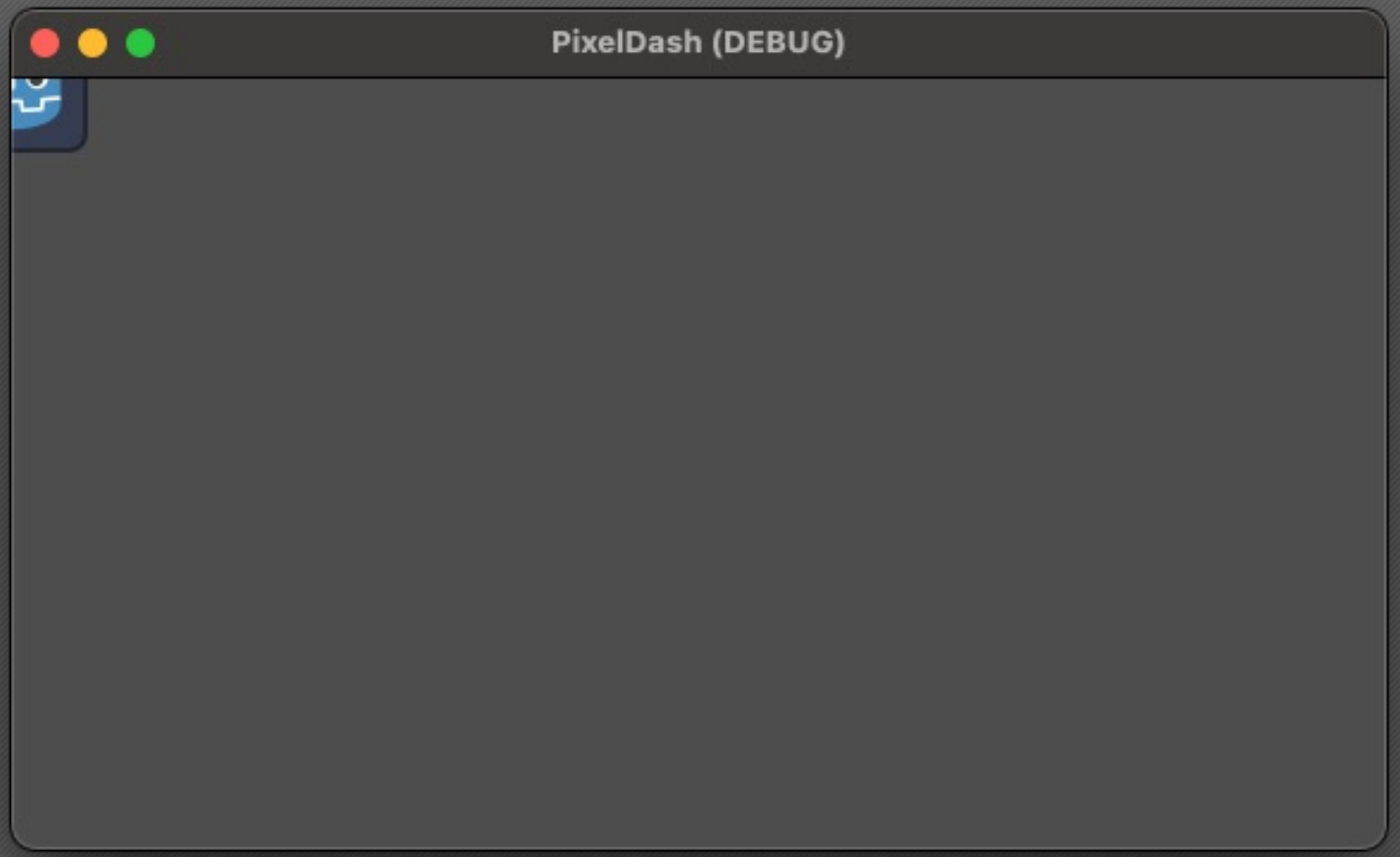
Running the Game

- If you click the Play button, Godot will ask you to confirm the main scene.
- If you set the main scene now, the Player will remain the main run scene until you change it in game settings.
- Changing the main scene will be covered in the Level section.

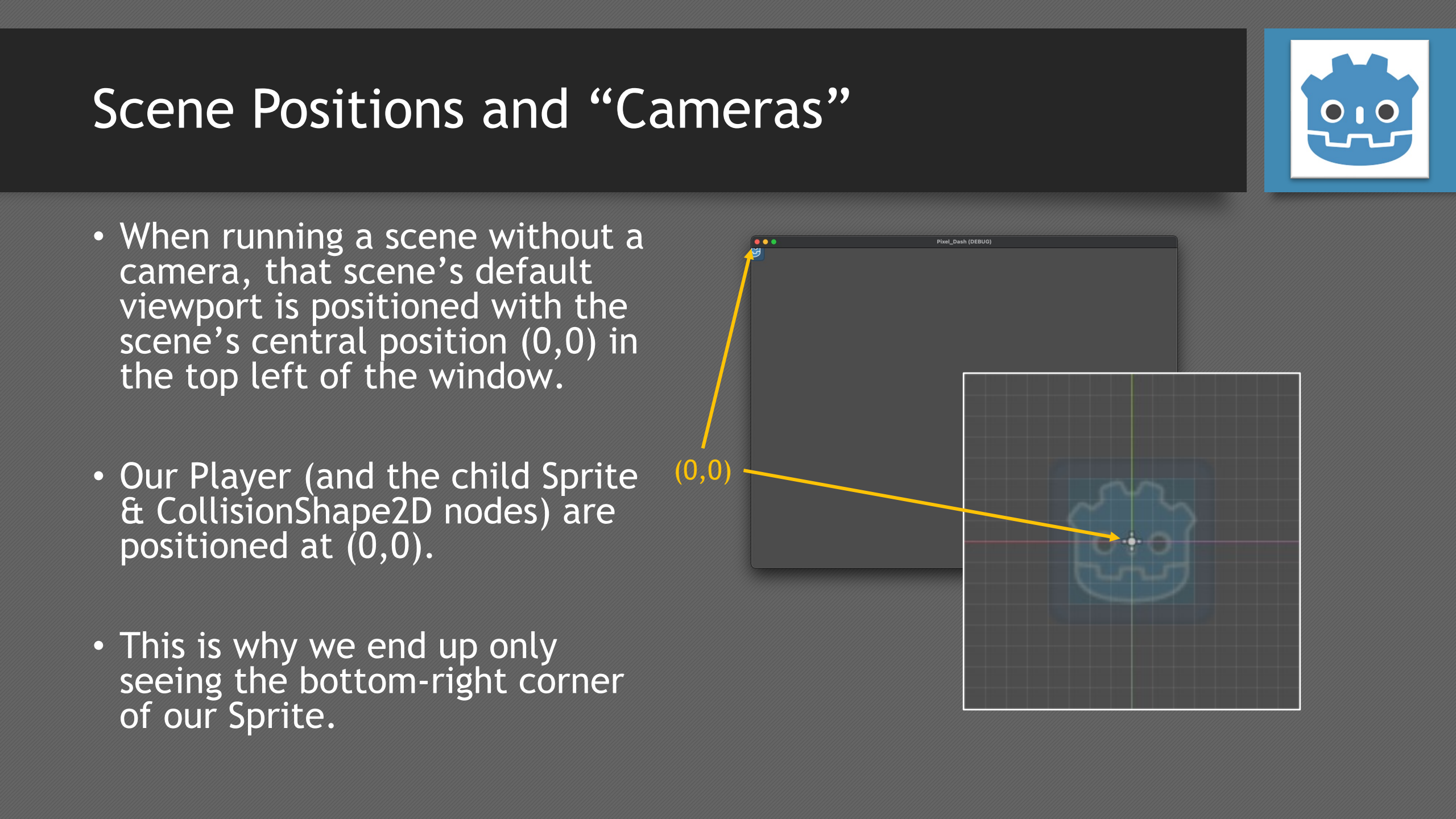




Running the Game

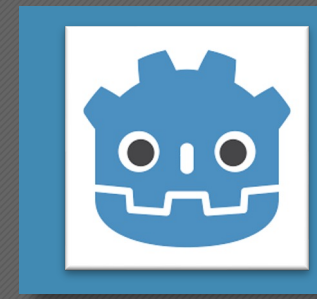


Scene Positions and “Cameras”

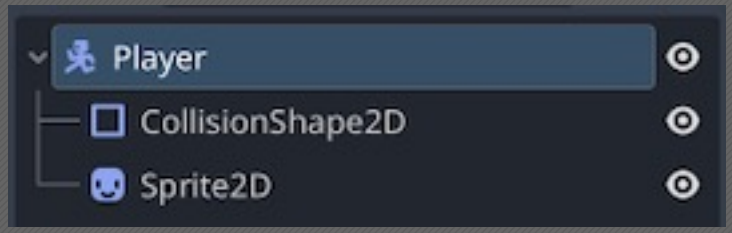


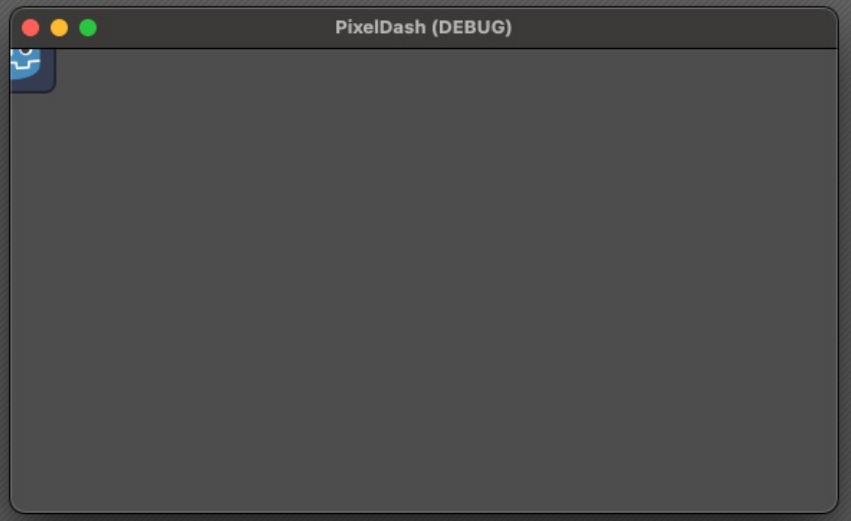
- When running a scene without a camera, that scene’s default viewport is positioned with the scene’s central position (0,0) in the top left of the window.
- Our Player (and the child Sprite & CollisionShape2D nodes) are positioned at (0,0).
- This is why we end up only seeing the bottom-right corner of our Sprite.

Lab Time (~5 Minutes)



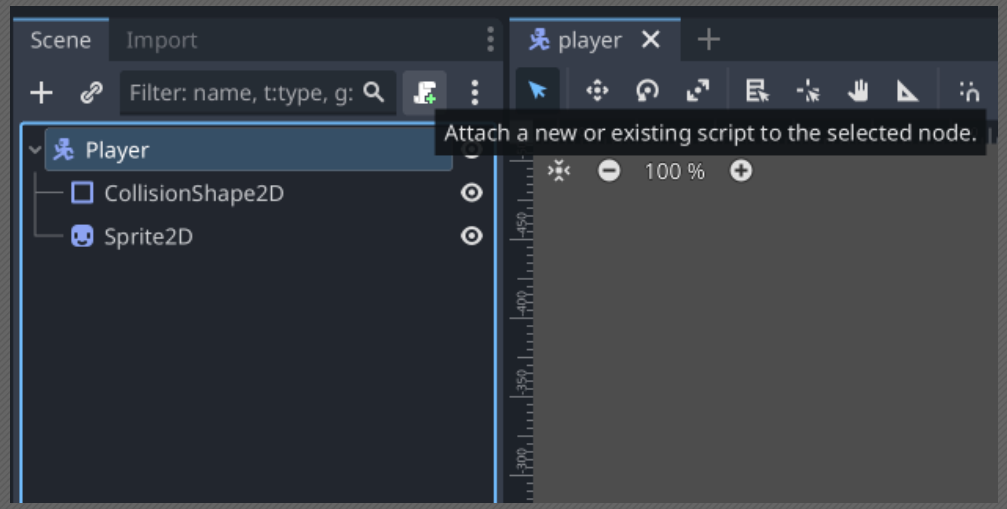
- Create a new scene with a CharacterBody2D root node.
- Add the CollisionShape2D and Sprite2D child nodes.
- Give the CollisionShape2D a shape.
- Set the texture of the Sprite2D using the icon.svg image.
- Save and Run the Player scene.



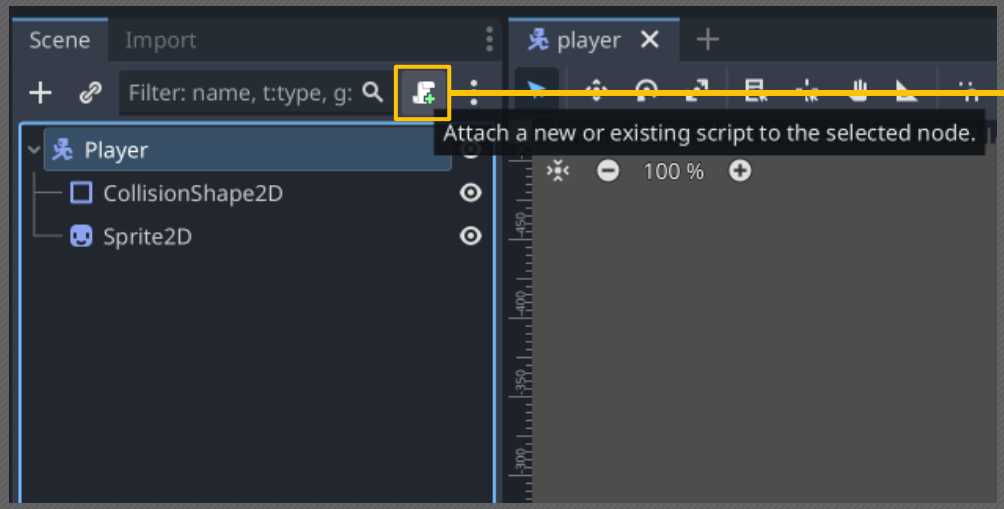


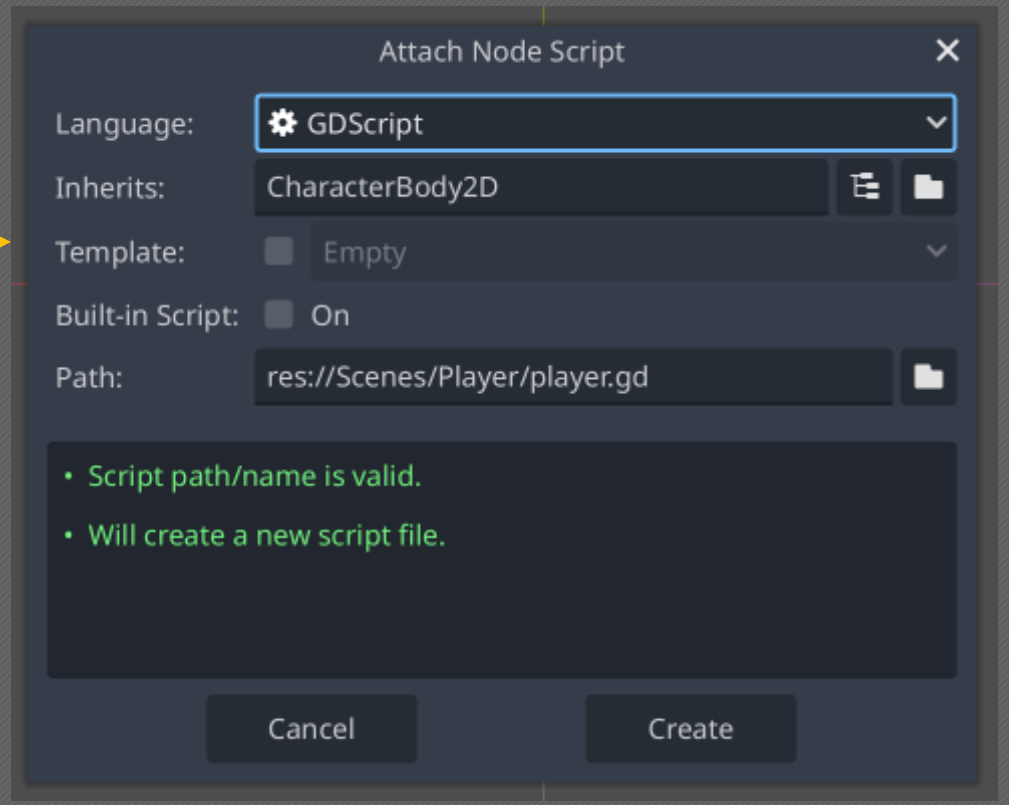
Attaching a Script

- A Script is a code file that can be attached to a Node in the Scene Tree.
- Any Node in the Scene Tree may have one and only one script attached to it.



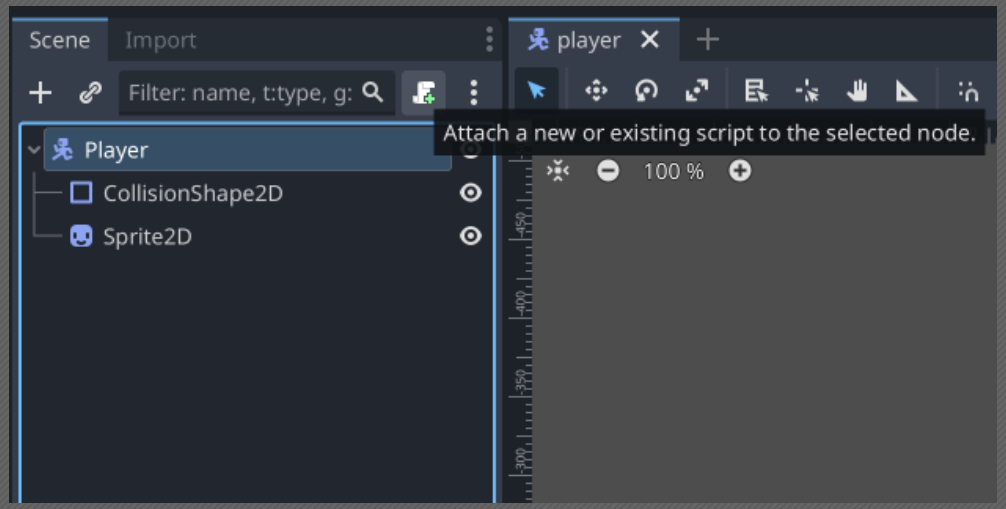
Creating a new Script

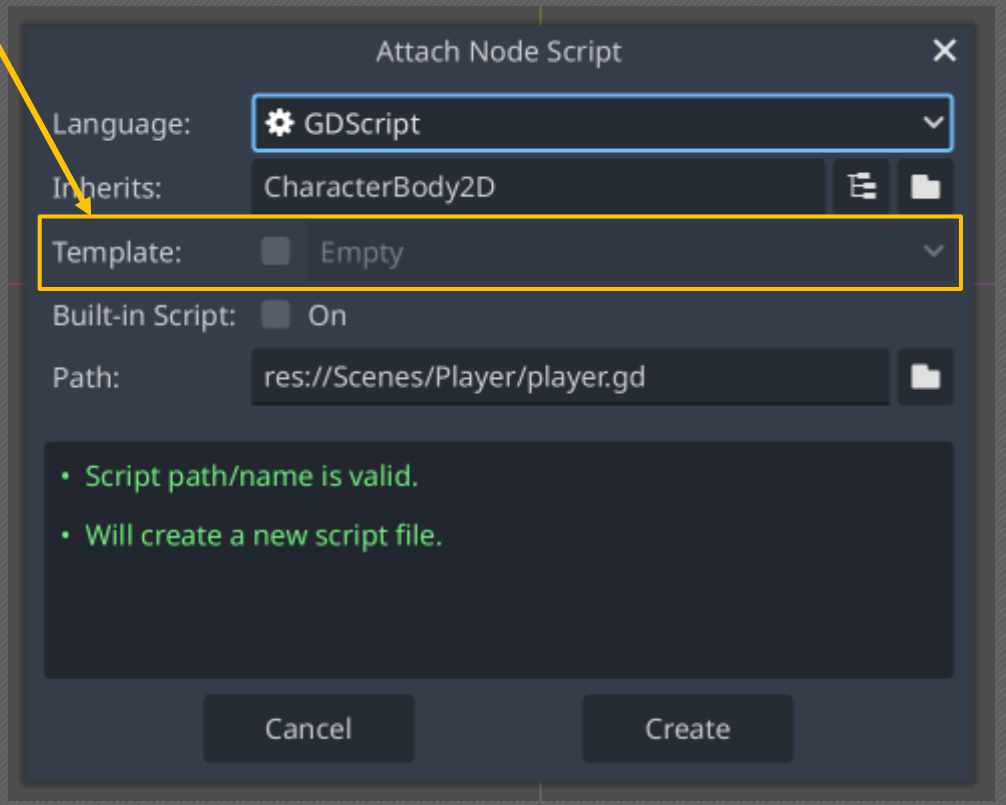




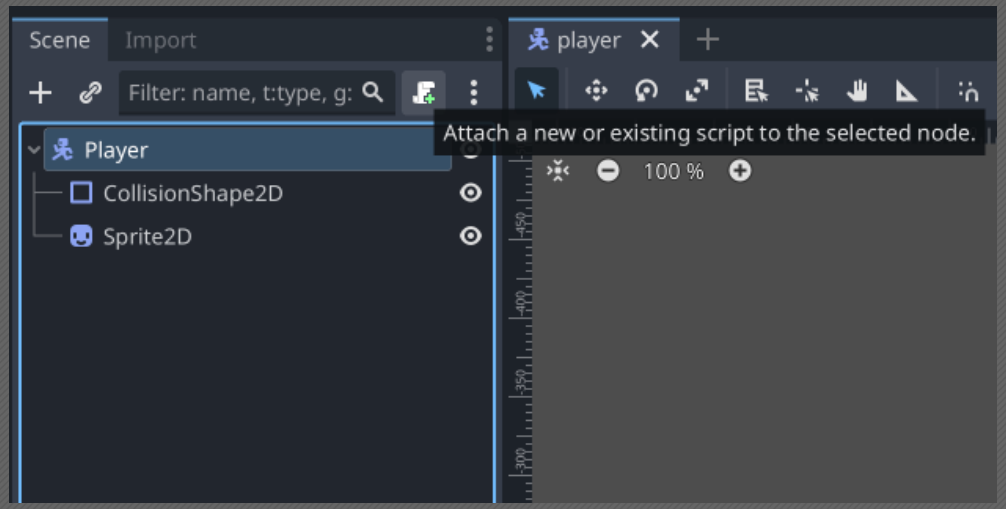
Creating a new Script

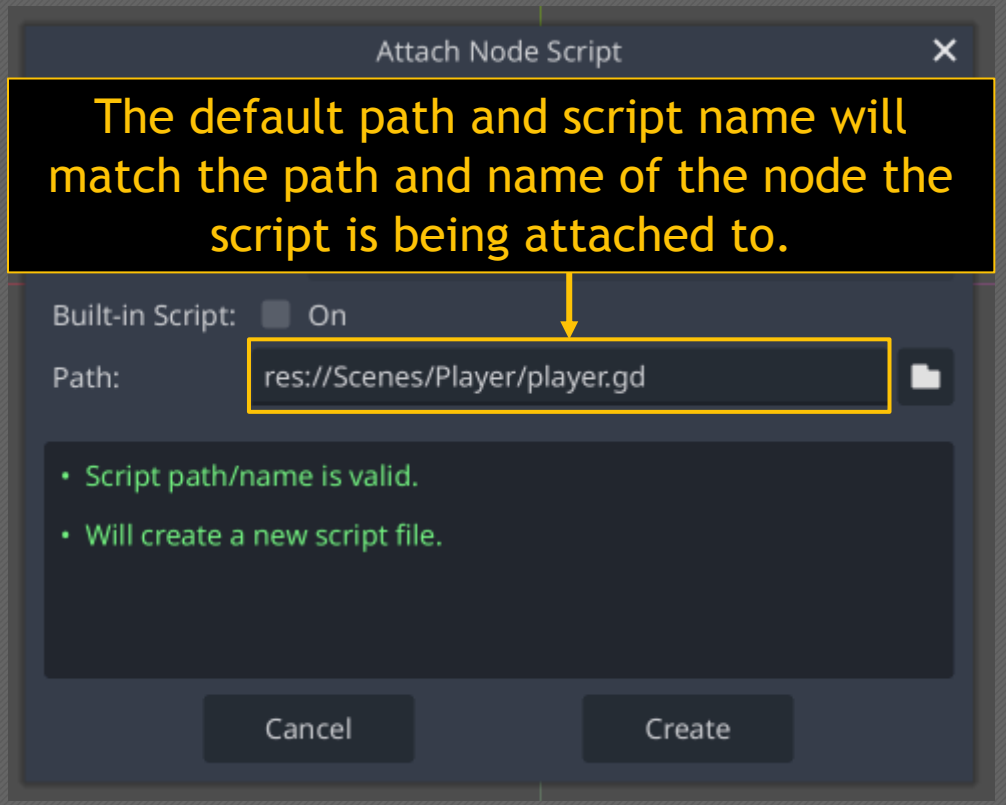
If Template is toggled/enabled, disable it. As a hands-on learning workshop, we want to implemented our own behaviors.

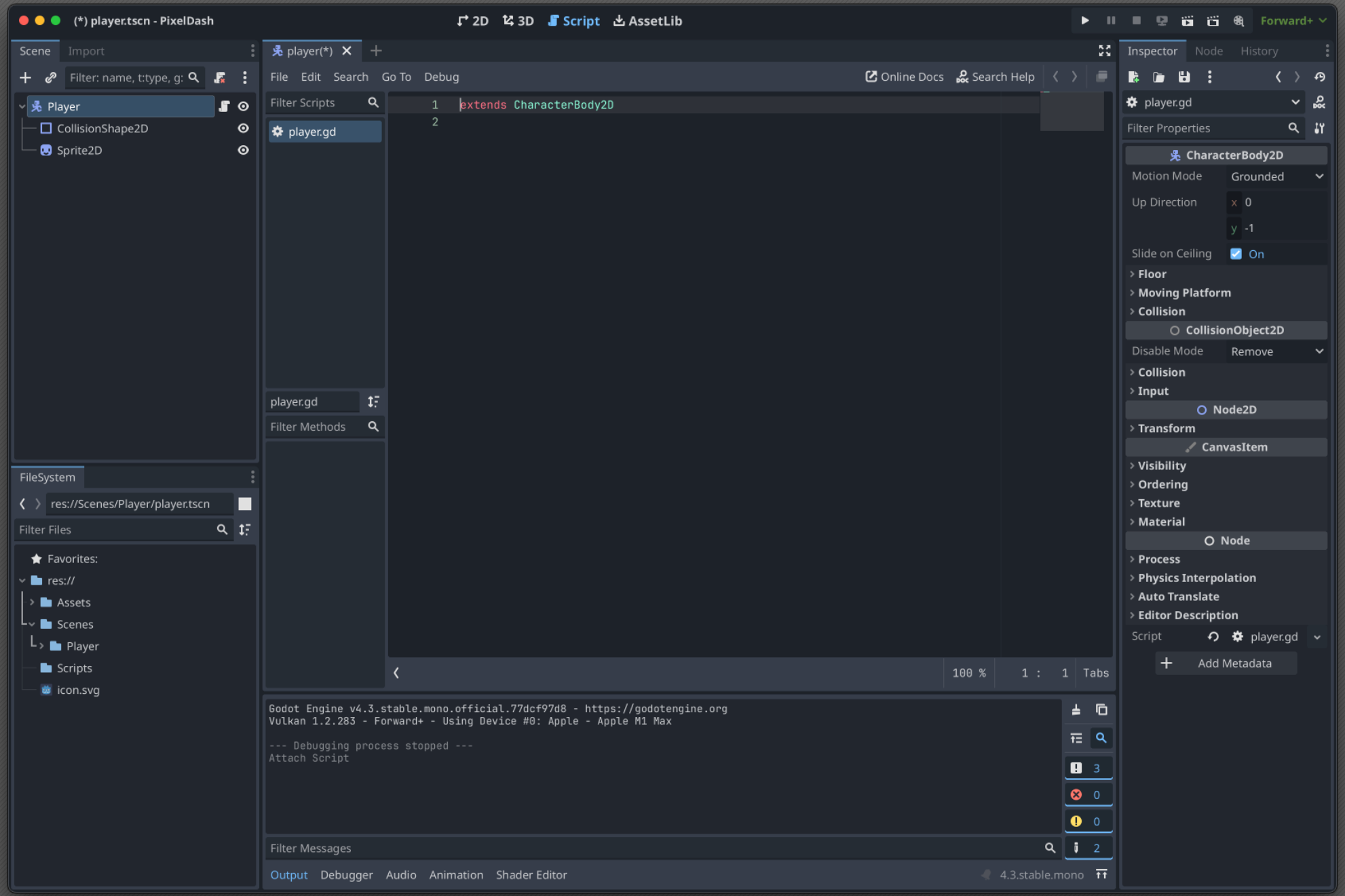


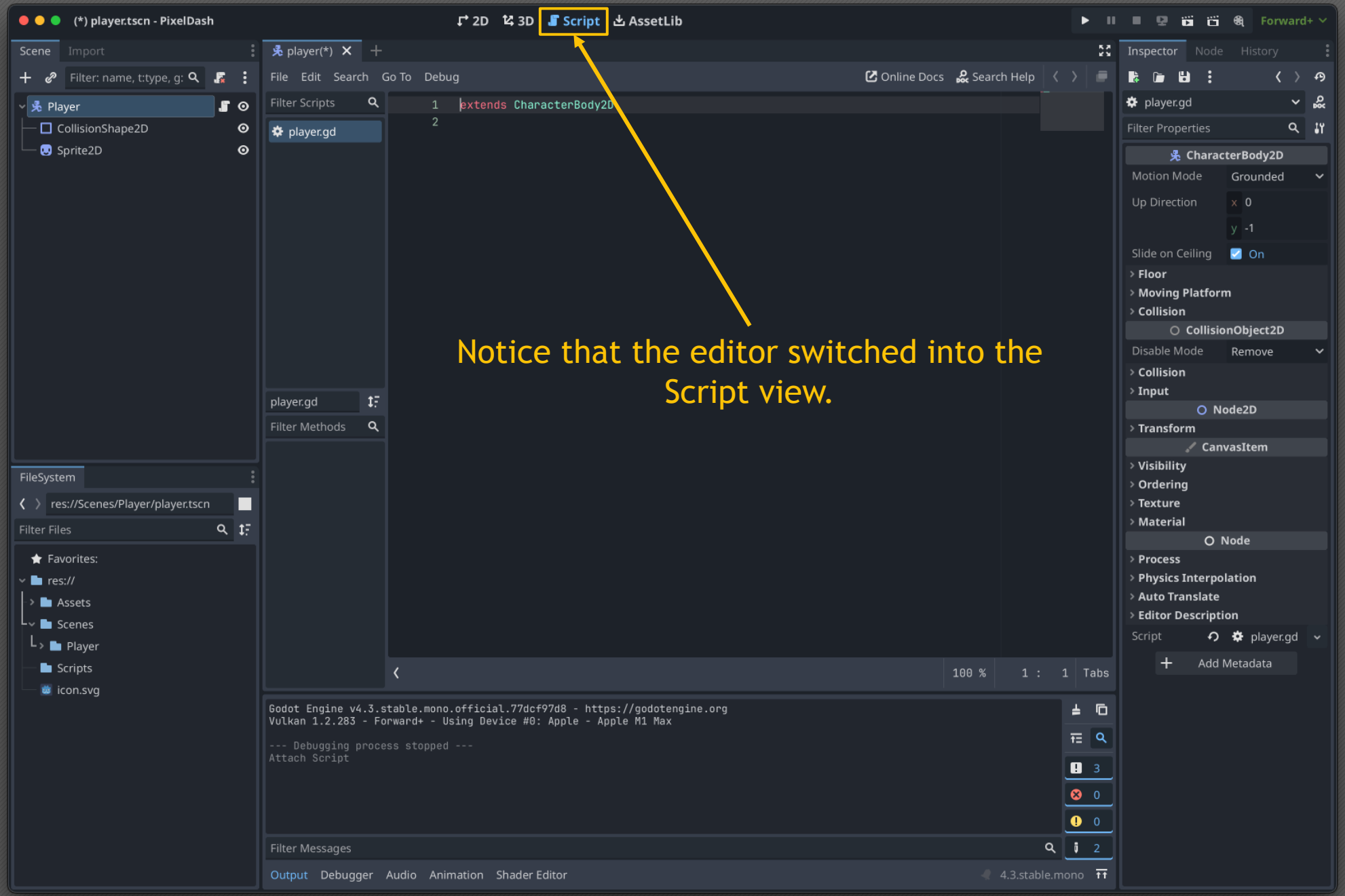


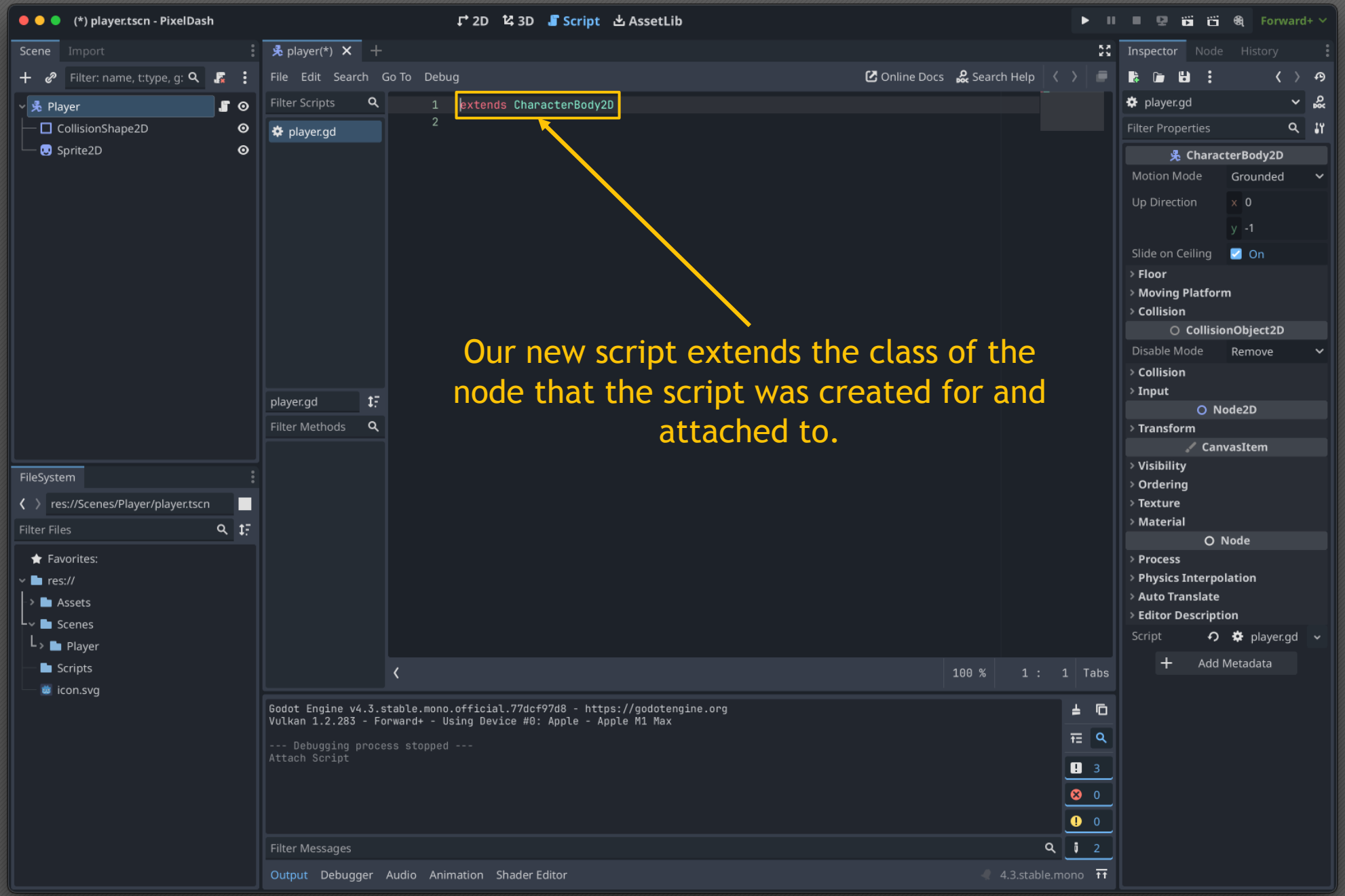
Creating a new Script



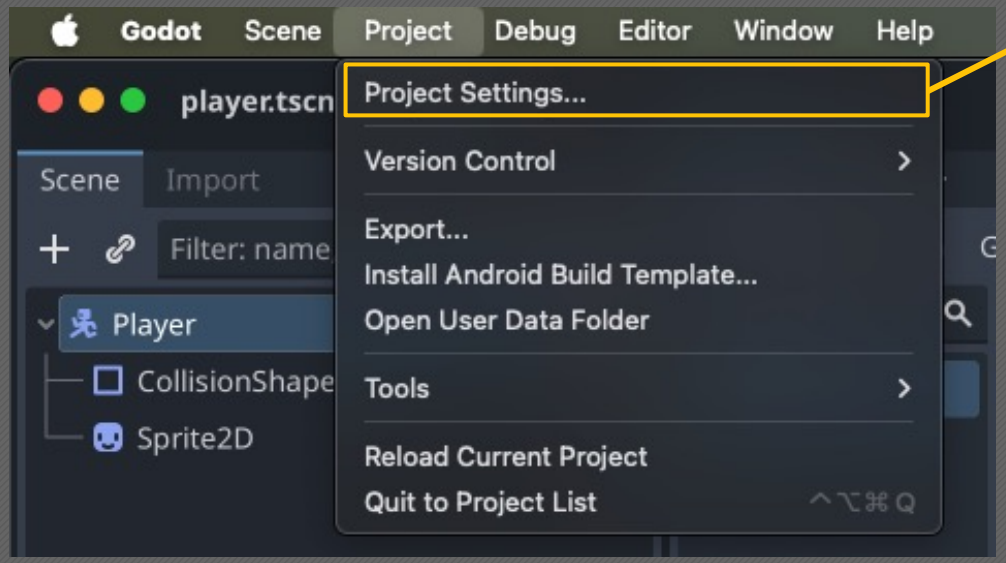


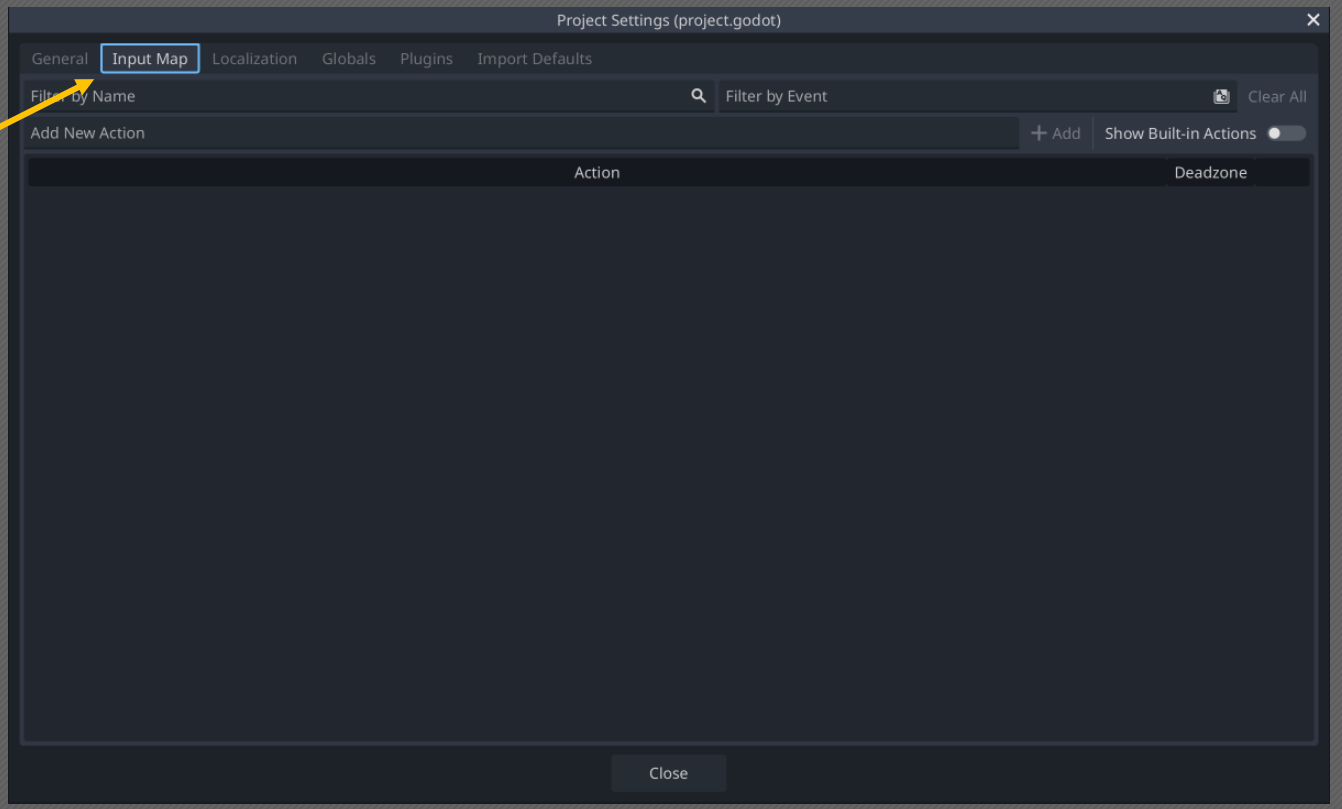

(*) player.tscn - PixelDash
2D
3D
Script
AssetLib
Forward+
Scene
Import
Filter: name, t:type, g:
Player
CollisionShape2D
Sprite2D
FileSystem
res://Scenes/Player/player.tscn
Filter Files
Favorites:
res://
Assets
Scenes
Player
Scripts
icon.svg
player(*)
File
Edit
Search
Go To
Debug
Online Docs
Search Help
Filter Scripts
player.gd
player.gd
Filter Methods
1 extends CharacterBody2D
2
100 %
1 : 1
Tabs
Godot Engine v4.3.stable.mono.official.77dcf97d8 - https://godotengine.org
Vulkan 1.2.283 - Forward+ - Using Device #0: Apple - Apple M1 Max
--- Debugging process stopped ---
Attach Script
Filter Messages
3
0
0
2
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono
Inspector
Node
History
player.gd
Filter Properties
CharacterBody2D
Motion Mode
Grounded
Up Direction
x 0
y -1
Slide on Ceiling
On
Floor
Moving Platform
Collision
CollisionObject2D
Disable Mode
Remove
Collision
Input
Node2D
Transform
CanvasItem
Visibility
Ordering
Texture
Material
Node
Process
Physics Interpolation
Auto Translate
Editor Description
Script
player.gd
Add Metadata


(*) player.tscn - PixelDash
2D
3D
Script
AssetLib
Forward+
Scene
Import
Filter: name, t:type, g:
Player
CollisionShape2D
Sprite2D
player(*)
File
Edit
Search
Go To
Debug
Online Docs
Search Help
Filter Scripts
player.gd
1 extends CharacterBody2D
2
player.gd
Filter Methods
Notice that the editor switched into the Script view.
100 %
1 : 1
Tabs
Inspector
Node
History
player.gd
Filter Properties
CharacterBody2D
Motion Mode
Grounded
Up Direction
x 0
y -1
Slide on Ceiling
On
Floor
Moving Platform
Collision
CollisionObject2D
Disable Mode
Remove
Collision
Input
Node2D
Transform
CanvasItem
Visibility
Ordering
Texture
Material
Node
Process
Physics Interpolation
Auto Translate
Editor Description
Script
player.gd
Add Metadata
FileSystem
res://Scenes/Player/player.tscn
Filter Files
Favorites:
res://
Assets
Scenes
Player
Scripts
icon.svg
Godot Engine v4.3.stable.mono.official.77dcf97d8 - https://godotengine.org
Vulkan 1.2.283 - Forward+ - Using Device #0: Apple - Apple M1 Max
--- Debugging process stopped ---
Attach Script
3
0
0
2
Filter Messages
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono

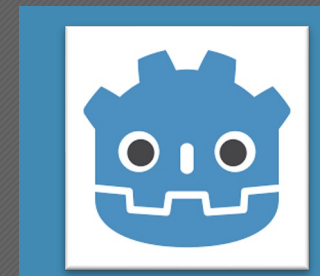

(*) player.tscn - PixelDash
2D
3D
Script
AssetLib
Forward+
Scene
Import
Filter: name, t:type, g:
Player
CollisionShape2D
Sprite2D
player(*)
File
Edit
Search
Go To
Debug
Online Docs
Search Help
Filter Scripts
player.gd
1 extends CharacterBody2D
2
Our new script extends the class of the node that the script was created for and attached to.
player.gd
Filter Methods
100 %
1 : 1
Tabs
Inspector
Node
History
player.gd
Filter Properties
CharacterBody2D
Motion Mode
Grounded
Up Direction
x 0
y -1
Slide on Ceiling
On
Floor
Moving Platform
Collision
CollisionObject2D
Disable Mode
Remove
Collision
Input
Node2D
Transform
CanvasItem
Visibility
Ordering
Texture
Material
Node
Process
Physics Interpolation
Auto Translate
Editor Description
Script
player.gd
Add Metadata
FileSystem
res://Scenes/Player/player.tscn
Filter Files
Favorites:
res://
Assets
Scenes
Player
Scripts
icon.svg
Godot Engine v4.3.stable.mono.official.77dcf97d8 - https://godotengine.org
Vulkan 1.2.283 - Forward+ - Using Device #0: Apple - Apple M1 Max
--- Debugging process stopped ---
Attach Script
3
0
0
2
Filter Messages
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono

Configuring Input Mappings

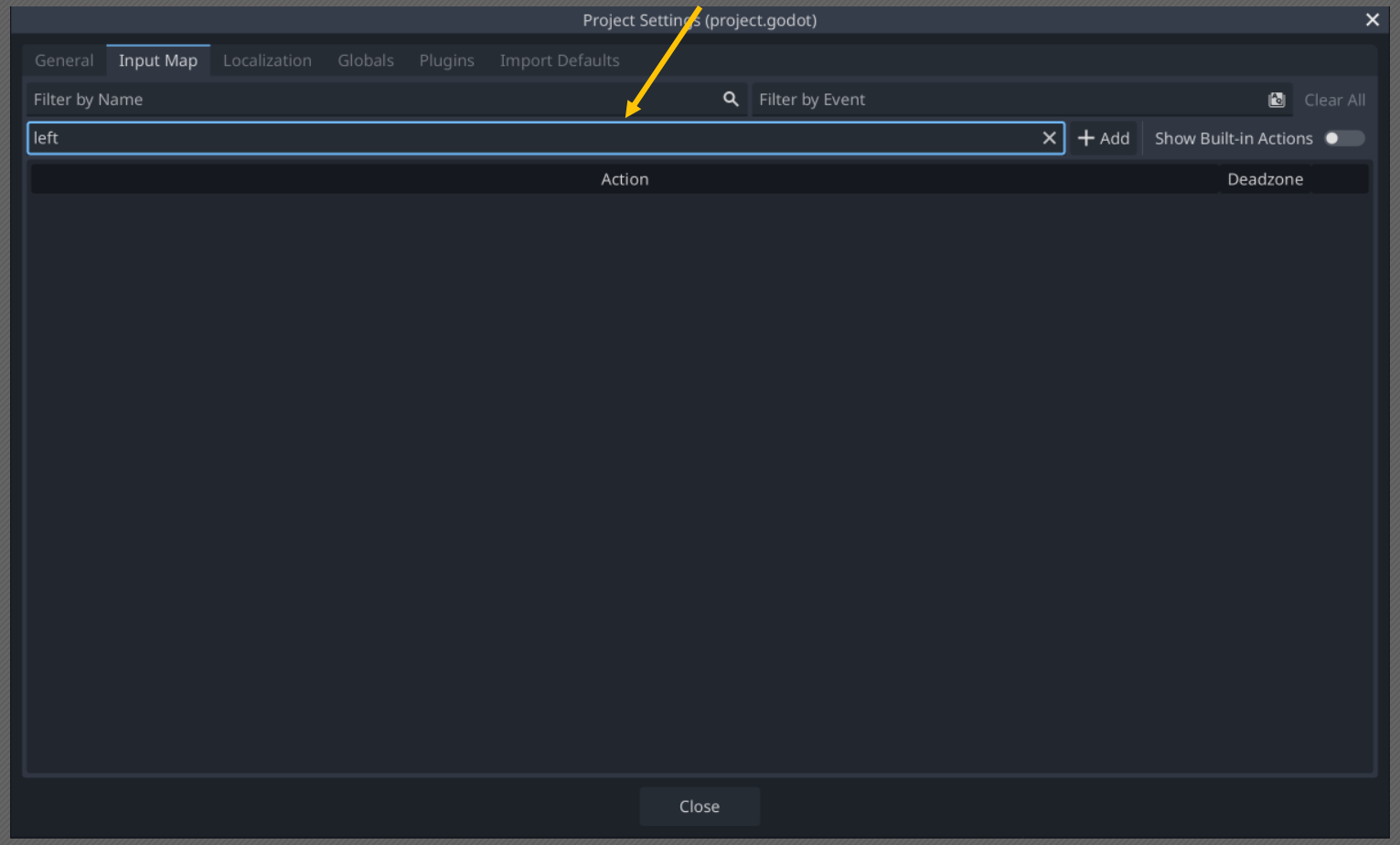




Configuring Input Mappings

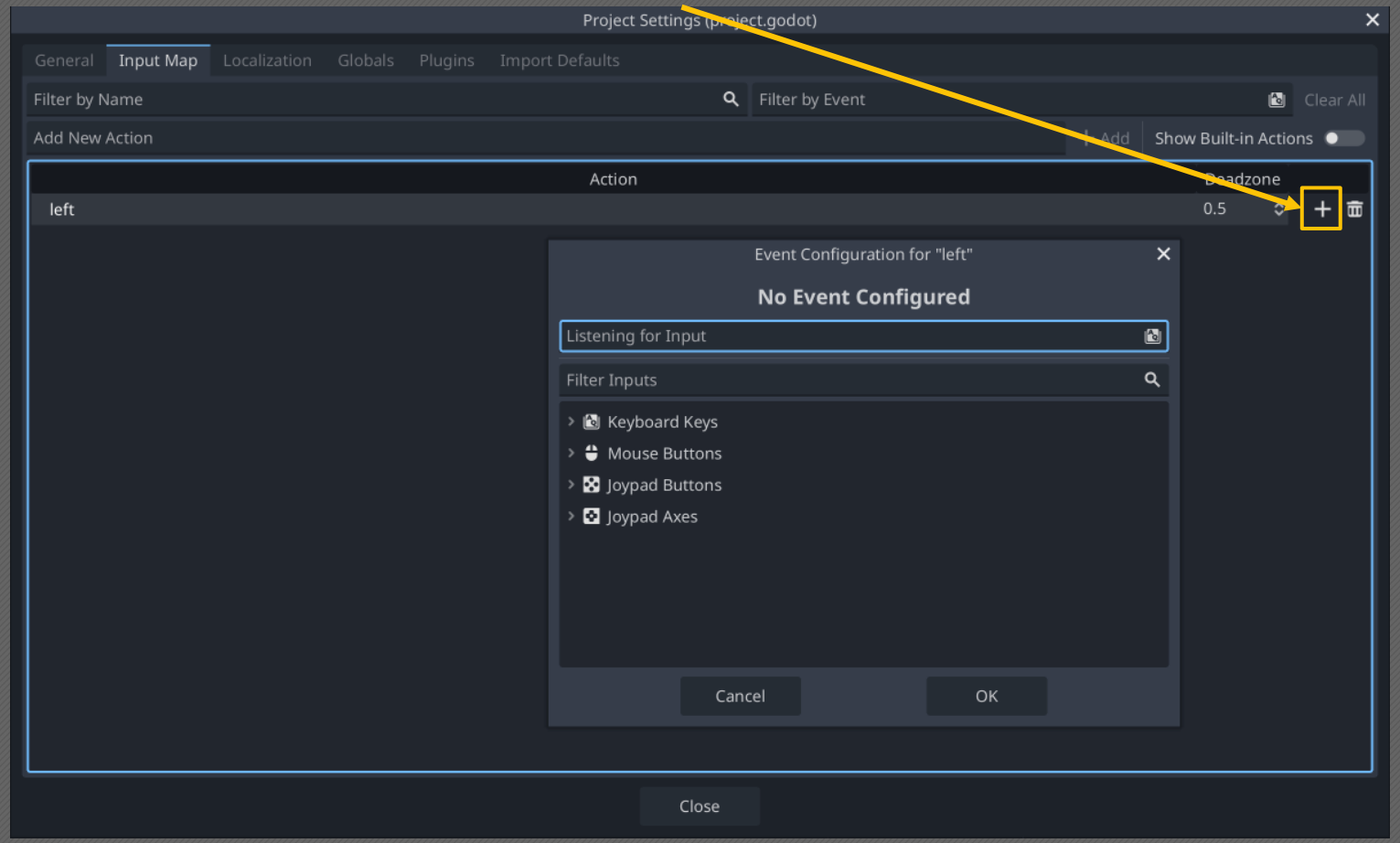


Define the name of the input action, then click the + Add button.



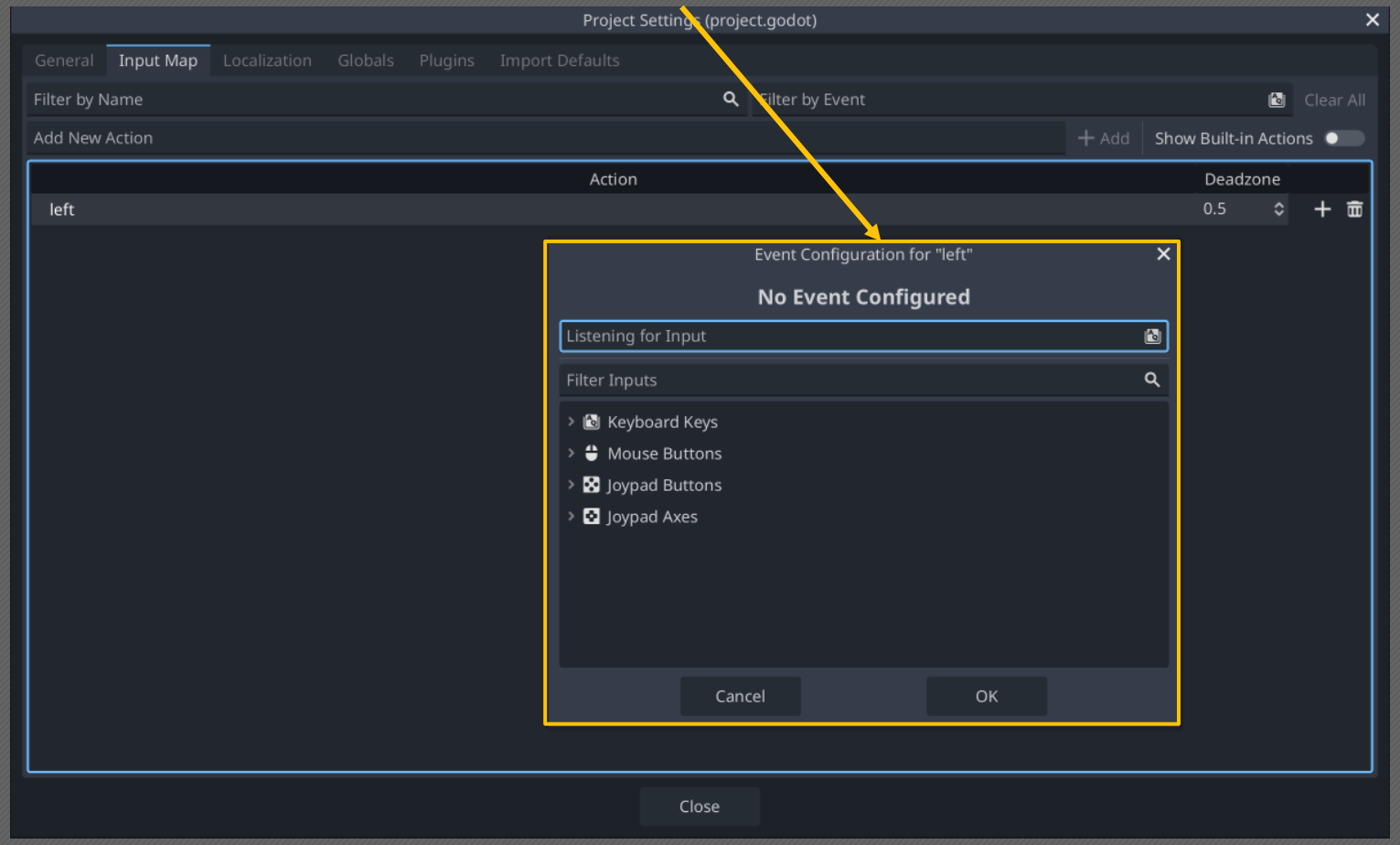
Configuring Input Mappings

The name will be added to the list. Click the + button to configure the input.

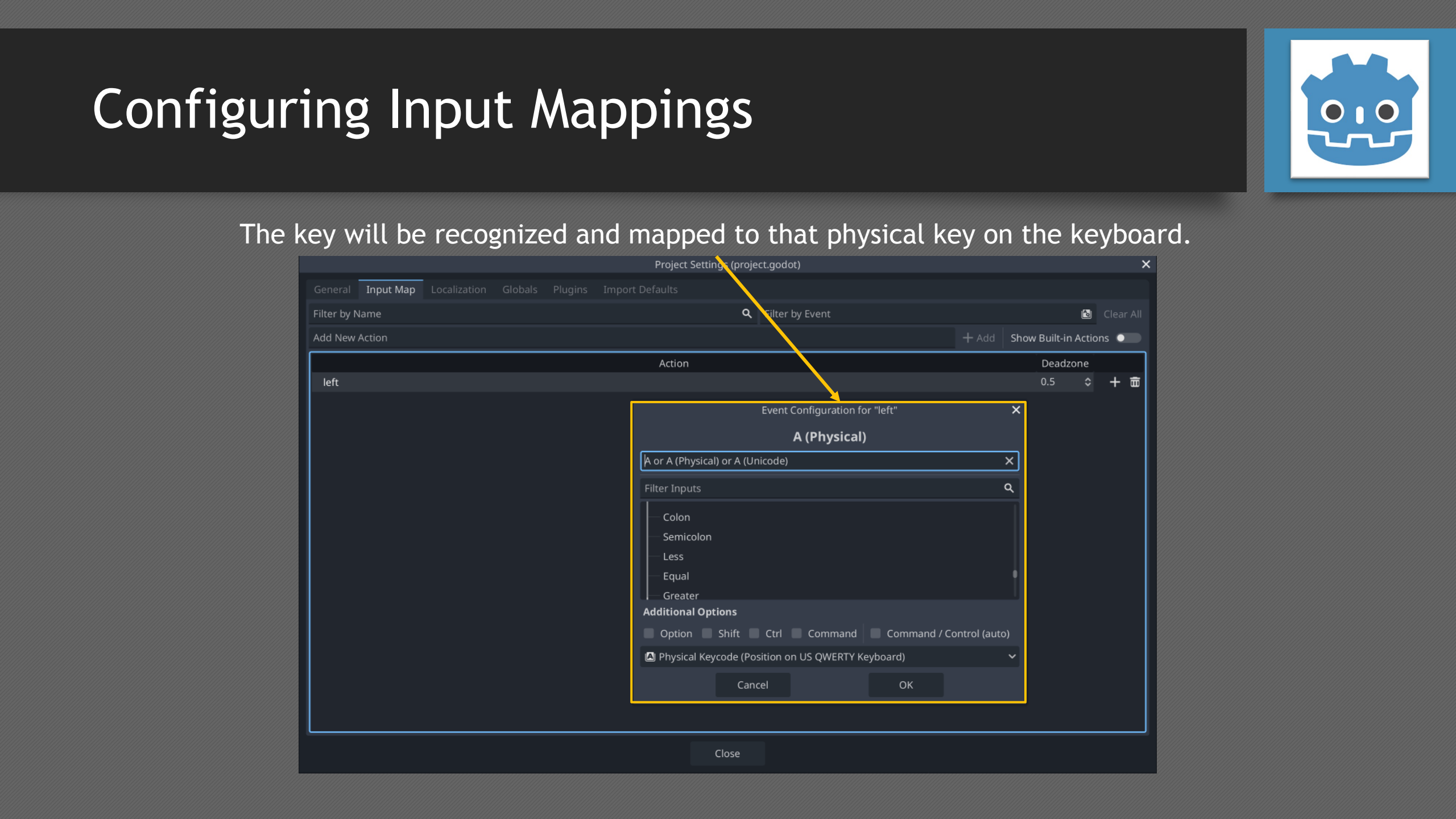


Configuring Input Mappings

The editor will immediately start listening for an input key. Press the “A” key.

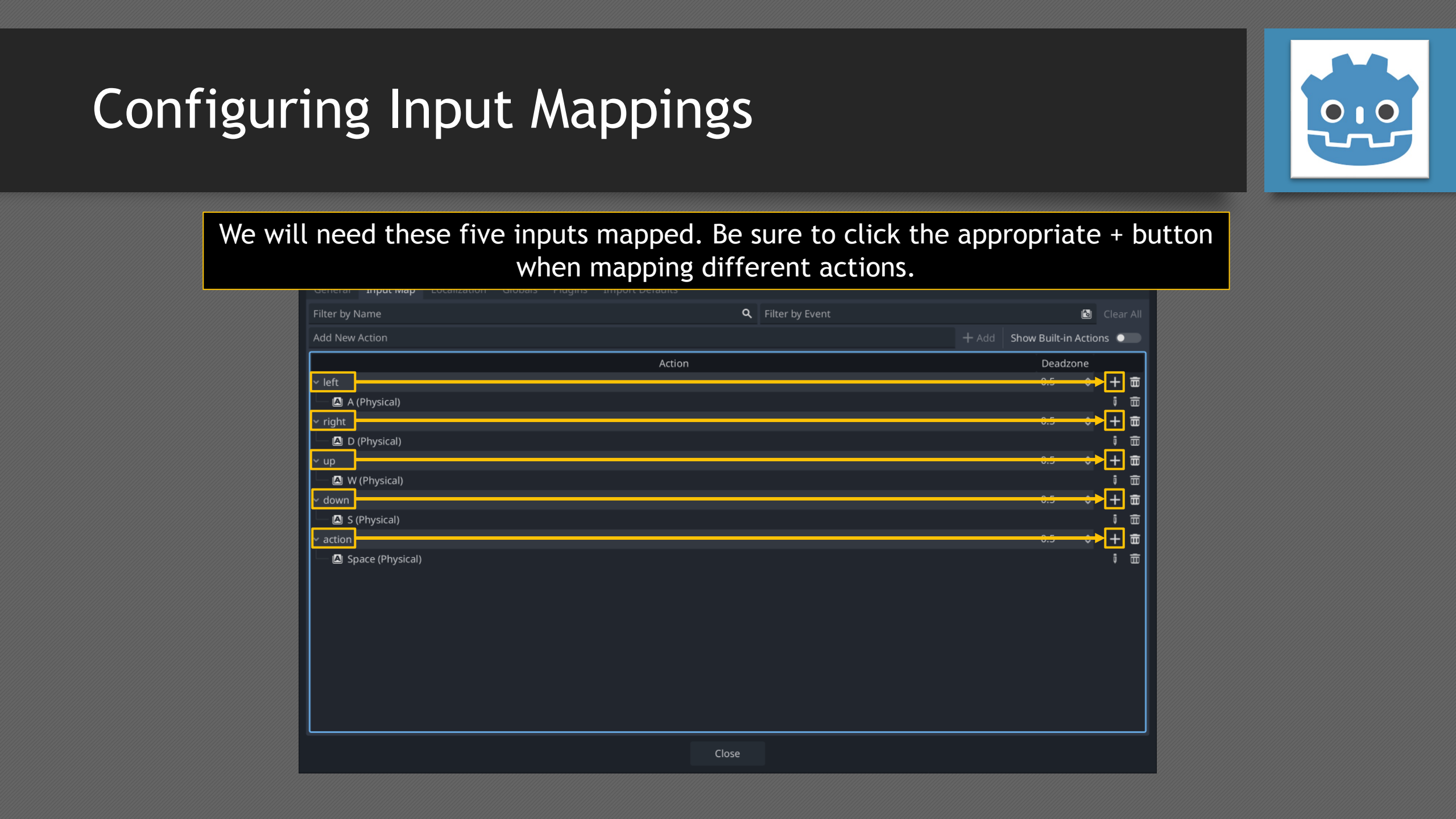


Configuring Input Mappings



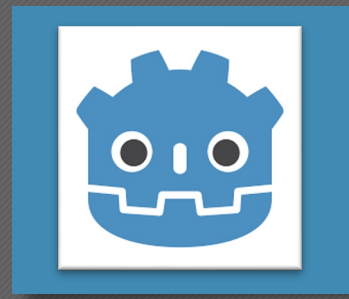
The key will be recognized and mapped to that physical key on the keyboard.

Configuring Input Mappings



We will need these five inputs mapped. Be sure to click the appropriate + button when mapping different actions.

GDScript Syntax



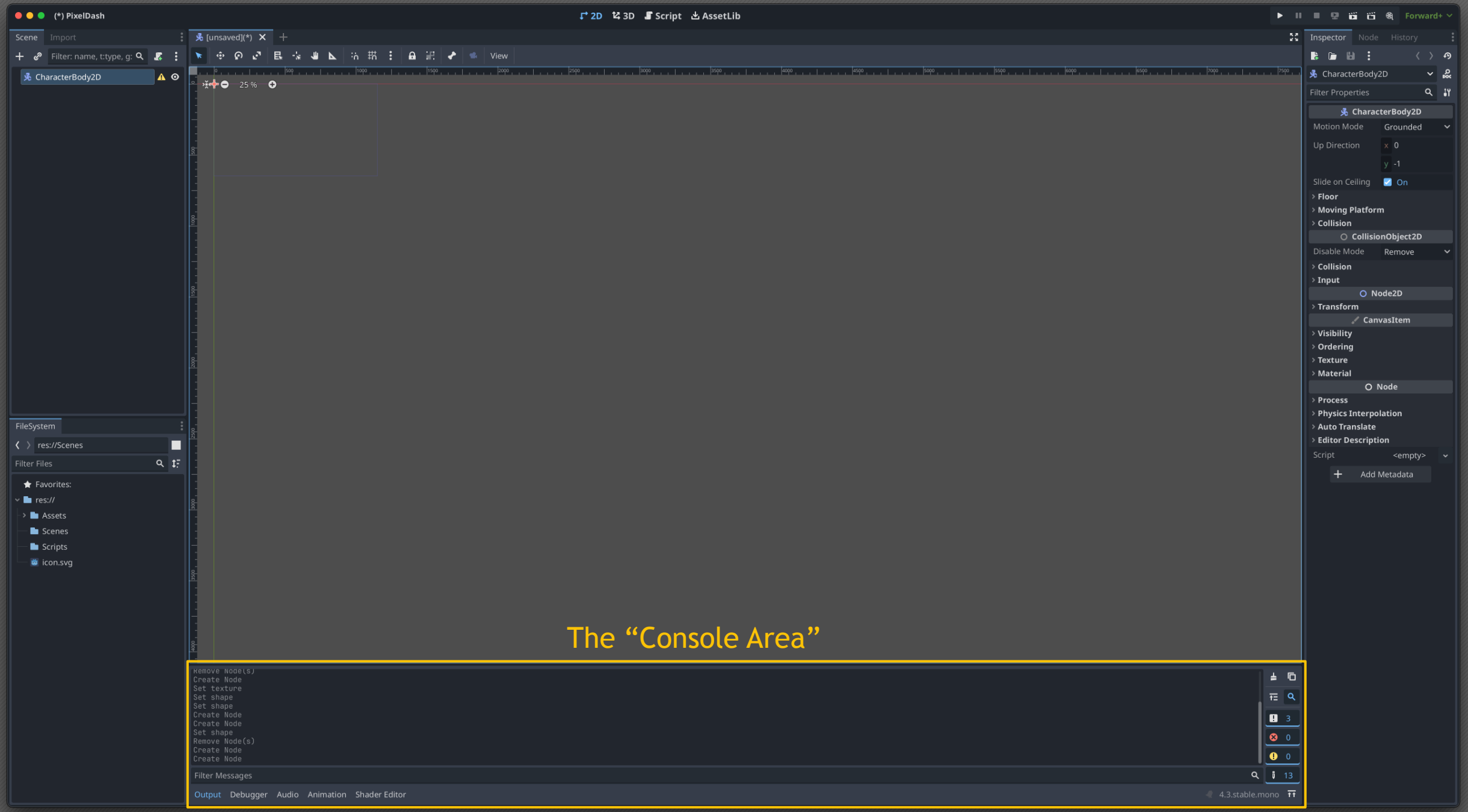
- GDScript is a dynamic language that uses ad indentation-based syntax.
- Line scope is defined by preceding Tab spaces. Spaces *cannot* be used instead of tabs.
- Variable types can be optionally defined. Variables with an explicit type are enforced to remain that type.

```
# Member variables.
var a = 5
var s = "Hello"
var typed_var: int
var inferred_type := "String"

# Constants.
const ANSWER = 42

# Functions.
func some_function(param1, param2, param3):
    const local_const = 5
    if param1 < local_const:
        print(param1)
    elif param2 > 5:
        print(param2)
    else:
        print("Fail!")
```

GDScript Syntax

- Explicitly typed and inferred typed variables are limited to remain that type.
- “Untyped” values are truly dynamic and can be changed to any type at any time.
- The `:` character defines the type, and the walrus operator `:=` is syntactic sugar that enforces the variable to remain to the inferred type based on the init value.

```gdscript
# Member variables.
var a = 5
var s = "Hello"
var typed_var: int
var inferred_type := "String"

# Functions.
func some_function(param1, param2, param3):
    a = "Hello"
    s = 10
    # Errs! Typed vars must remain that type
    typed_var = ""
    inferred_type = 1

    var y := 15
    var x : int = 10
    x = ""
    y = ""
```

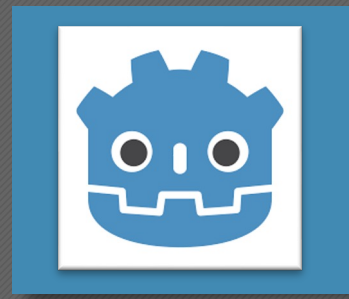

(*) PixelDash
2D
3D
Script
AssetLib
Forward+
Scene
Import
Filter: name, t:type, g:
CharacterBody2D
[unsaved](*)
View
25 %
Inspector
Node
History
CharacterBody2D
Filter Properties
CharacterBody2D
Motion Mode
Grounded
Up Direction
x 0
y -1
Slide on Ceiling
On
Floor
Moving Platform
Collision
CollisionObject2D
Disable Mode
Remove
Collision
Input
Node2D
Transform
CanvasItem
Visibility
Ordering
Texture
Material
Node
Process
Physics Interpolation
Auto Translate
Editor Description
Script
<empty>
Add Metadata
FileSystem
res://Scenes
Filter Files
Favorites:
res://
Assets
Scenes
Scripts
icon.svg
The “Console Area”
Remove Node(s)
Create Node
Set texture
Set shape
Set shape
Create Node
Create Node
Set shape
Remove Node(s)
Create Node
Create Node
3
0
0
Filter Messages
13
Output
Debugger
Audio
Animation
Shader Editor
4.3.stable.mono

GDScript Syntax

- Constant values can be defined with the const keyword. Const variables must be initialized with a constant evaluation and cannot be changed.
- The print function will print to the Console.
- You can perform String interpolation via %s (for String) or %d (for a number) within the desired String, followed by a % [array] operation to interpolate the values by index as they're found.

```
# Member variables.
var a = 5
var s = "Hello"
var typed_var: int
var inferred_type := "String"

const presenter = "Brandon"

# Functions.
func some_function(p1: int, p2: String):
    print("Print a line to the console")
    print("Presenter = %s" % [presenter])
    # Err! Const values can't be changed
    presenter = "Brandon Lewis"
    presenter = 1
```

Node-Inherited Lifecycle Functions



```
func _ready() -> void
```

Called when the node is "ready", i.e. when both the node and its children have entered the scene tree. If the node has children, their _ready callbacks get triggered before the parent.

```
func _process(delta: float) -> void
```

Called during the processing step of the main loop. Processing happens at every frame and as fast as possible, so the delta time since the previous frame is not constant. delta is in seconds.

```
func _input(event: InputEvent) -> void
```

Called when there is an input event. The input event propagates up through the node tree until a node consumes it. For gameplay input, _unhandled_input and _unhandled_key_input are usually a better fit as they allow the GUI to intercept the events first.

```
func _physics_process(delta: float) -> void
```

Called during the physics processing step of the main loop. Physics processing means that the frame rate is synced to the physics, i.e. the delta variable should be constant. delta is in seconds.

Handling Inputs in the Player script

```gdscript
extends CharacterBody2D

const BASE_MOVE_SPEED := 750

func _physics_process(delta: float) -> void:
    # read Input action strengths to easily get the horizontal and vertical input
    var input_vector := Input.get_vector("left", "right", "up", "down")

    # The input vector returns 0 for "not pressed" and 1 for "pressed", so multiply by move speed.
    velocity = input_vector * BASE_MOVE_SPEED

    # 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.
    move_and_slide()
```

Handling Inputs in the Player script

Input.get_vector gets an input vector (Vector2; contains an X and Y value) by specifying four actions for the positive and negative X and Y axes. The vector has it's a and Y lengths limited to 1.

```gdscript
extends CharacterBody2D

const BASE_MOVE_SPEED := 750

func _physics_process(delta: float) -> void:
    # read Input action strengths to easily get the horizontal and vertical input
    var input_vector := Input.get_vector("left", "right", "up", "down")

    # The input vector returns 0 for "not pressed" and 1 for "pressed", so multiply by move speed.
    velocity = input_vector * BASE_MOVE_SPEED

    # 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.
    move_and_slide()
```

Handling Inputs in the Player script

The velocity variable is inherited from the CharacterBody2D class. Set the velocity by multiplying velocity times our BASE_MOVE_SPEED. The Vector2 class has a * operator compatible with ints.

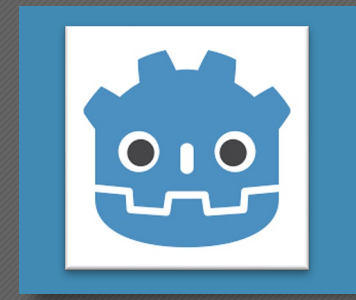
```gdscript
extends CharacterBody2D

const BASE_MOVE_SPEED := 750

func _physics_process(delta: float) -> void:
    # read Input action strengths to easily get the horizontal and vertical input
    var input_vector := Input.get_vector("left", "right", "up", "down")

    # The input vector returns 0 for "not pressed" and 1 for "pressed", so multiply by move speed.
    velocity = input_vector * BASE_MOVE_SPEED

    # 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.
    move_and_slide()
```

Handling Inputs in the Player script

Finally, call move_and_slide(), which is also inherited from CharacterBody2D.
Calling move_and_slide will move our character in the X,Y directions using the velocity variable.

```gdscript
extends CharacterBody2D

const BASE_MOVE_SPEED := 750

func _physics_process(delta: float) -> void:
    # read Input action strengths to easily get the horizontal and vertical input
    var input_vector := Input.get_vector("left", "right", "up", "down")

    # The input vector returns 0 for "not pressed" and 1 for "pressed", so multiply by move speed.
    velocity = input_vector * BASE_MOVE_SPEED

    # 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.
    move_and_slide()
```

Identifying Errors



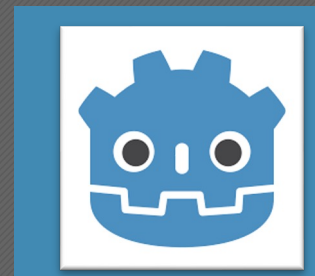
If your script has any errors, you can find the error message at the bottom of the code editor area.

```
1   extends CharacterBody2D
2
3   const BASE_MOVE_SPEED := 750
4
5   func _physics_process(delta: float) -> void:
6       # read Input action strengths to easily get the horizontal and vertical input
7       var input_vector := Input.get_vector("left", "right", "up", "down")
8
9       # The input vector returns 0 for "not pressed" and 1 for "pressed", so multiply by move speed.
10      velocity = input_vector * MOVE_SPEED
11
12      # Moves the body based on velocity. If the body collides with another, it will slide along the
13      # other body (by default only on floor) rather than stop immediately. If the other body is a
14      # CharacterBody2D or RigidBody2D, it will also be affected by the motion of the other body. You
15      # can use this to make moving and rotating platforms, or to make nodes push other nodes.
16      move_and_slide()
17
```

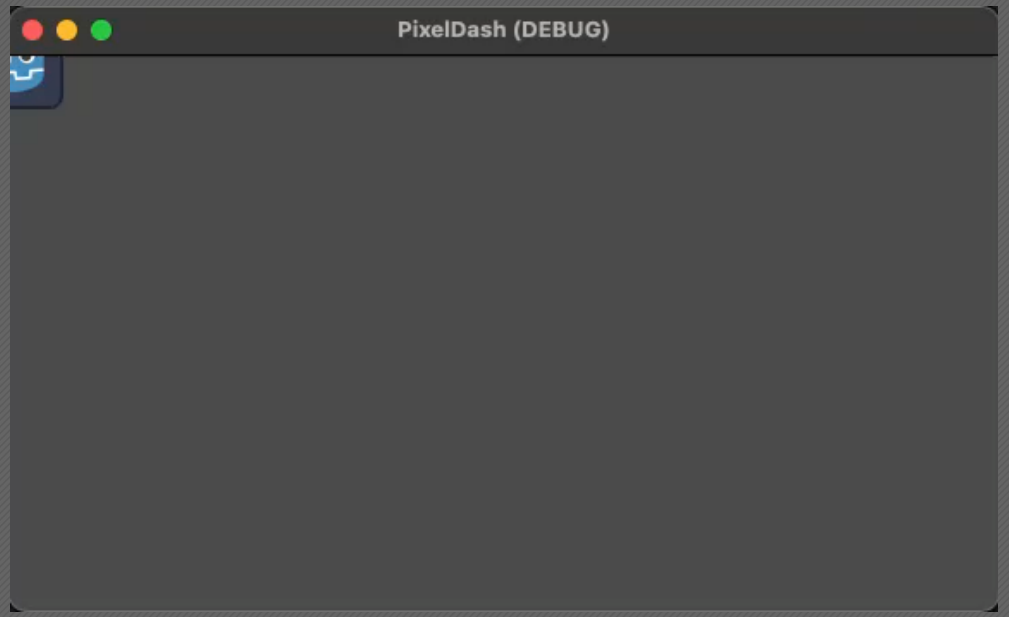
Error at (10, 31): Identifier "MOVE_SPEED" not declared in the current scope.

1 1 100 % 10 : 41 Tabs

Lab Time (~5 Minutes)

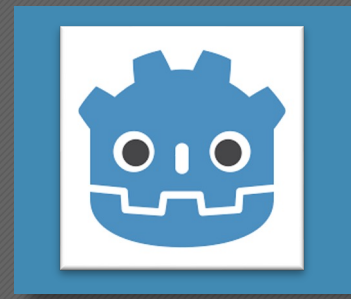


- Create and attach a new script to the root Player node.
- Implement the _physics_process func to get input and move the character.
- Rune the scene.




```gdscript
extends CharacterBody2D

const BASE_MOVE_SPEED := 750

func _physics_process(delta: float) -> void:
    # read Input action strengths to easily get the horizontal and vertical input
    var input_vector := Input.get_vector("left", "right", "up", "down")

    # The input vector returns 0 for "not pressed" and 1 for "pressed", so multiply by move speed.
    velocity = input_vector * BASE_MOVE_SPEED

    # 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.
    move_and_slide()
```

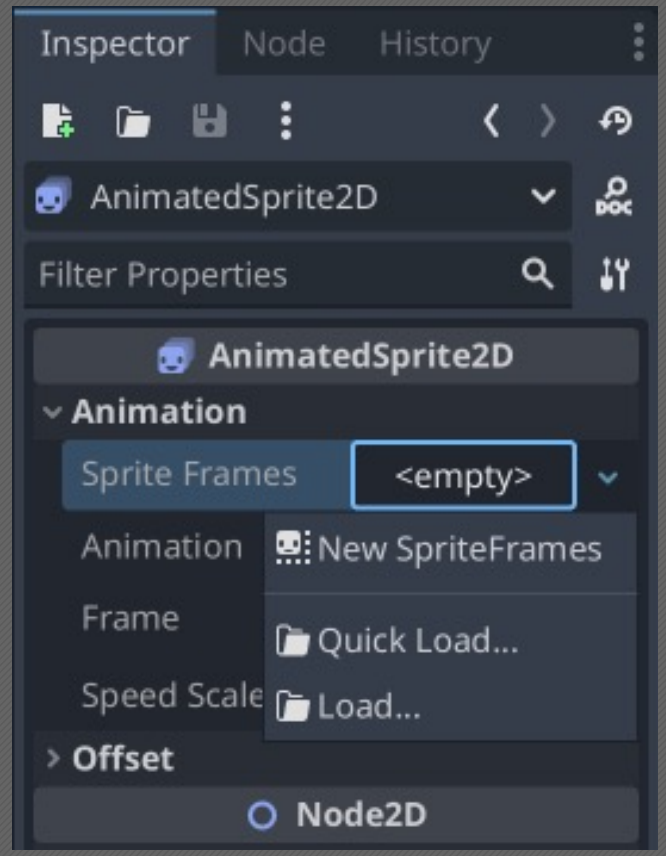
Adding Art & Animating our Character



- Add a new AnimatedSprite2D node as an immediate child of the root in the Player scene.

- Select the new AnimatedSprite2D node and select the Sprite Frames dropdown value in the Inspector tab.

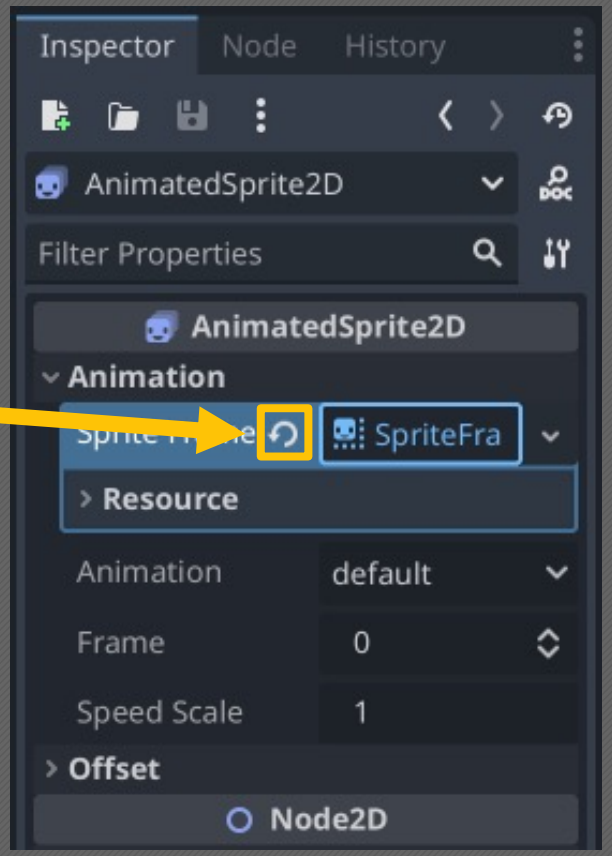
- In the dropdown context list, select “New SpriteFrames”.



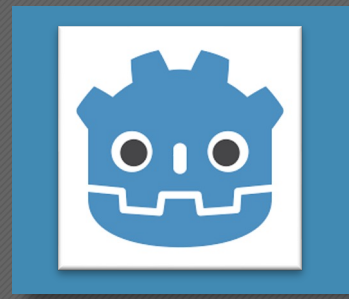
CAUTION: The Scary “Reset” Button

- Any time you change a value in the Inspector tree to be something other than the default value, the editor will offer a “Reset” button for that value.

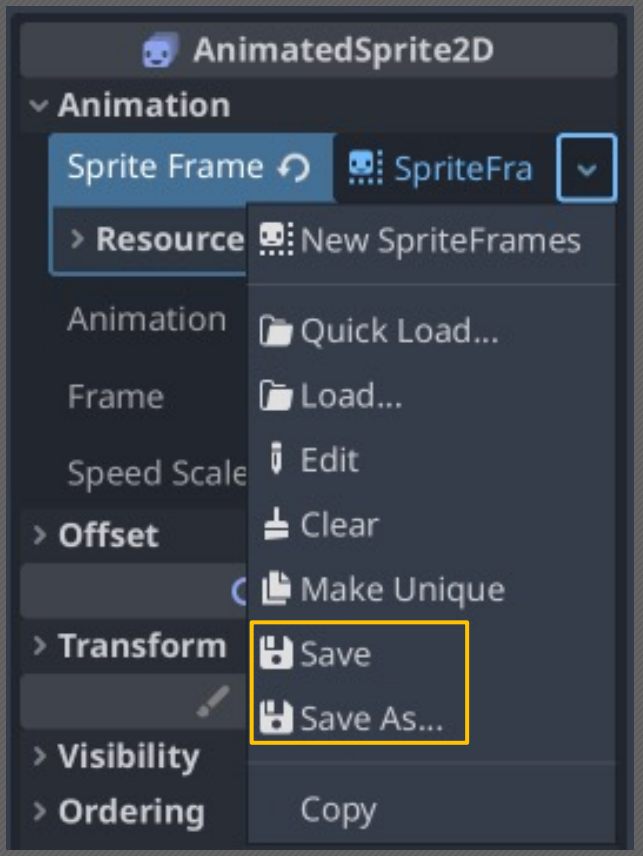
- Clicking it resets the value to the default value.



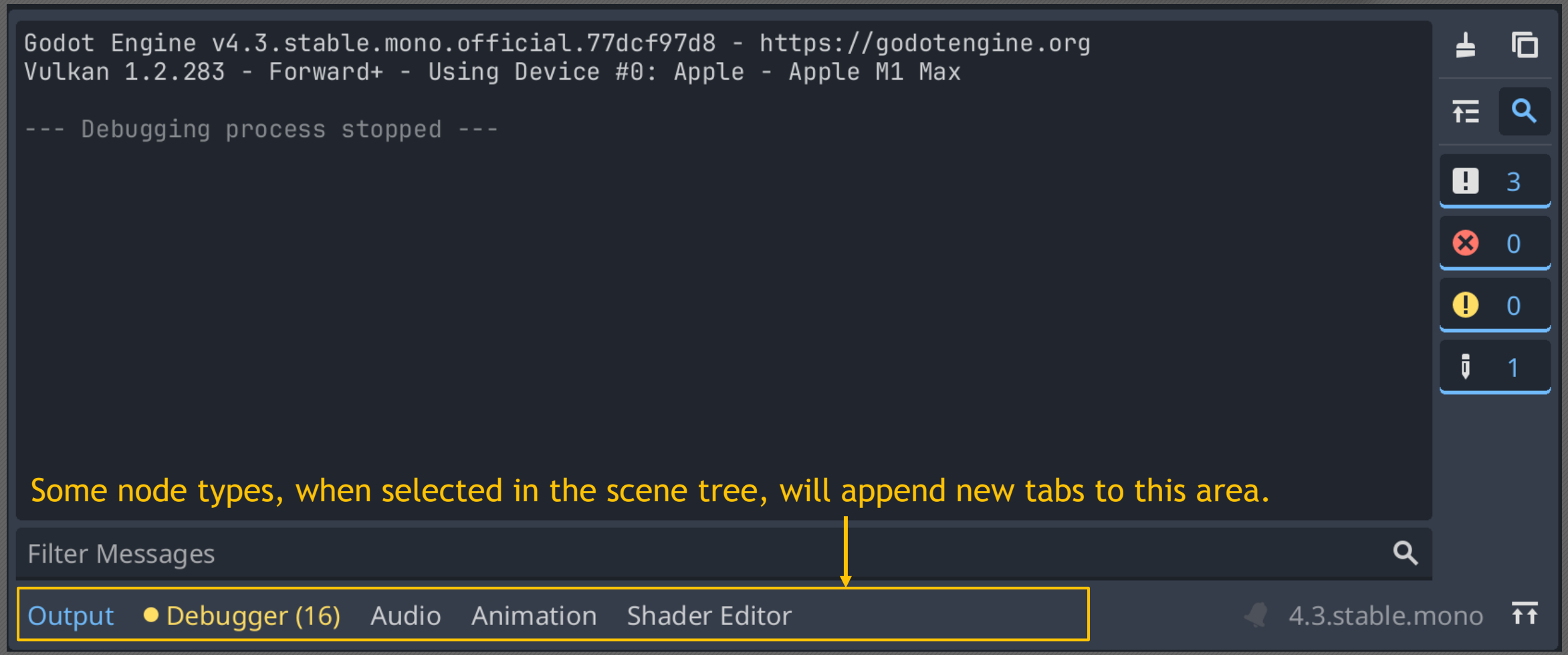
Saving a Resource



- A Resource is an instance of any type of data container.
- The SpriteFrames type on the Player AnimatedSprite2D Sprite Frame property is one example of a Resource.
- To avoid potentially losing all the work put into a “literal/magic” resource, we can save it as a file.
- If the reset button is accidentally clicked, the saved resource file can be dragged into the Sprite Frame value area from the file system.

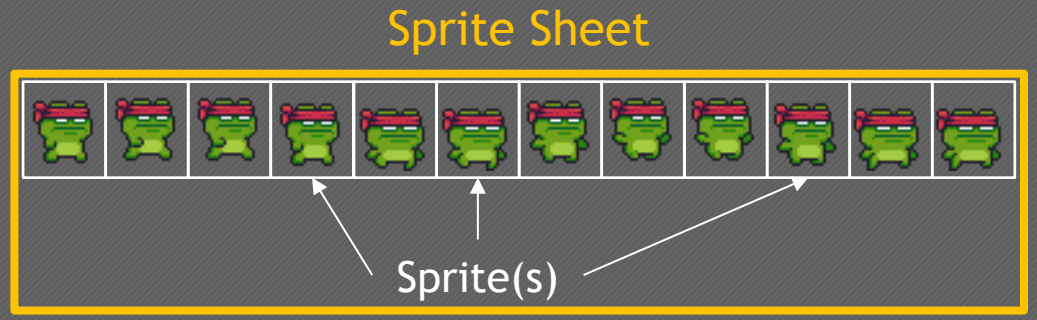


The Console Area



Sprites & Sprite Sheets

- A Sprite Sheet is an image containing a collection (or multiple collections) of logically-grouped, equal-sized images; each individual image slice is called a Sprite and represents a Frame of animation.

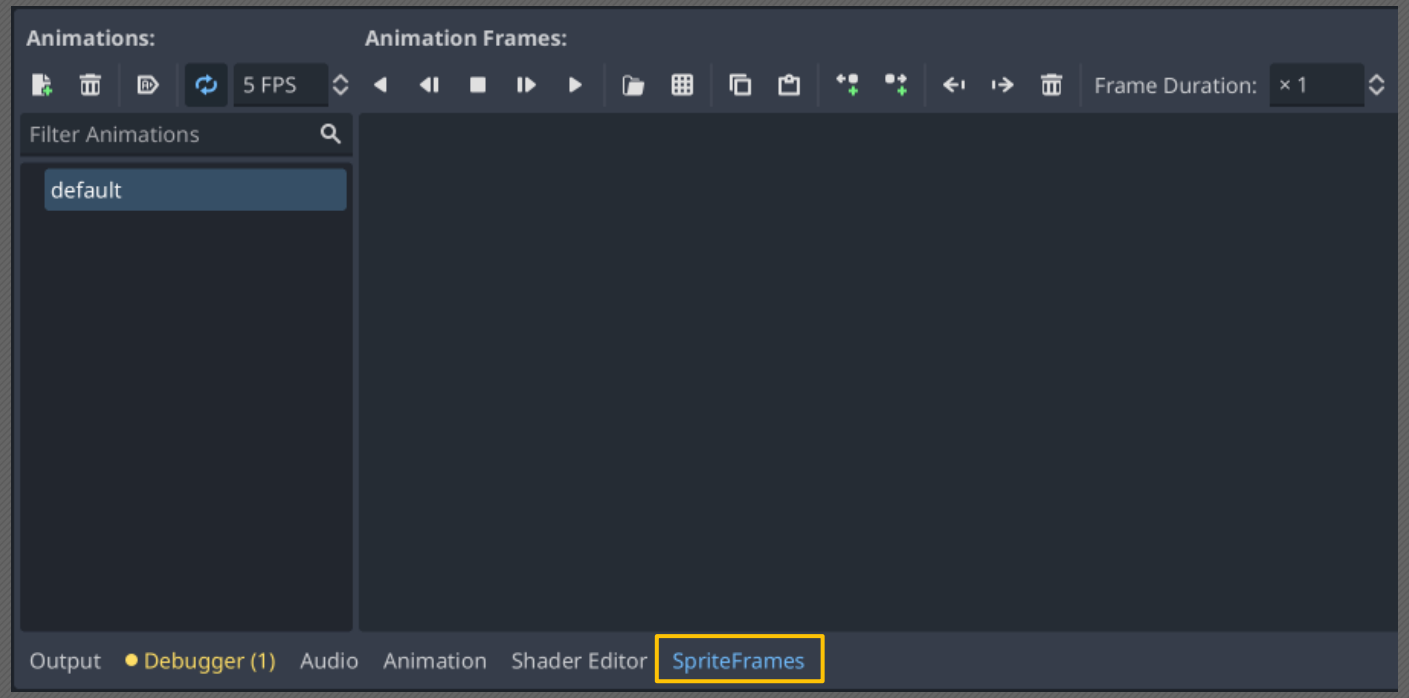


- Sprite Sheets are useful for creating 2D tilesets or animations.

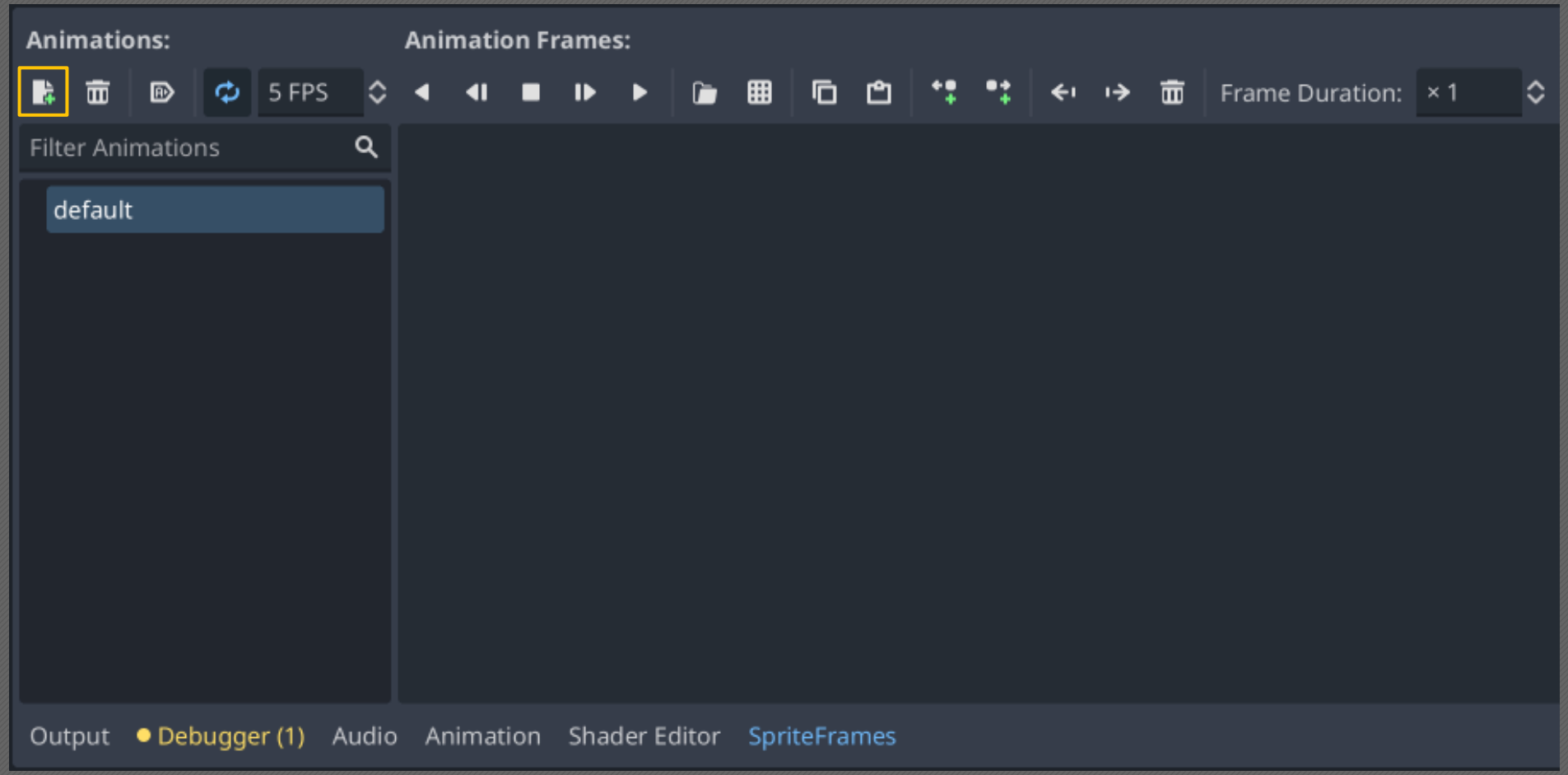
Adding Art & Animating our Character

- The AnimatedSprite2D's SpriteFrames will be added as an additional context option at the bottom of the console area.

- With the AnimatedSprite2D node selected in the scene tree, select the new SpriteFrames tab.

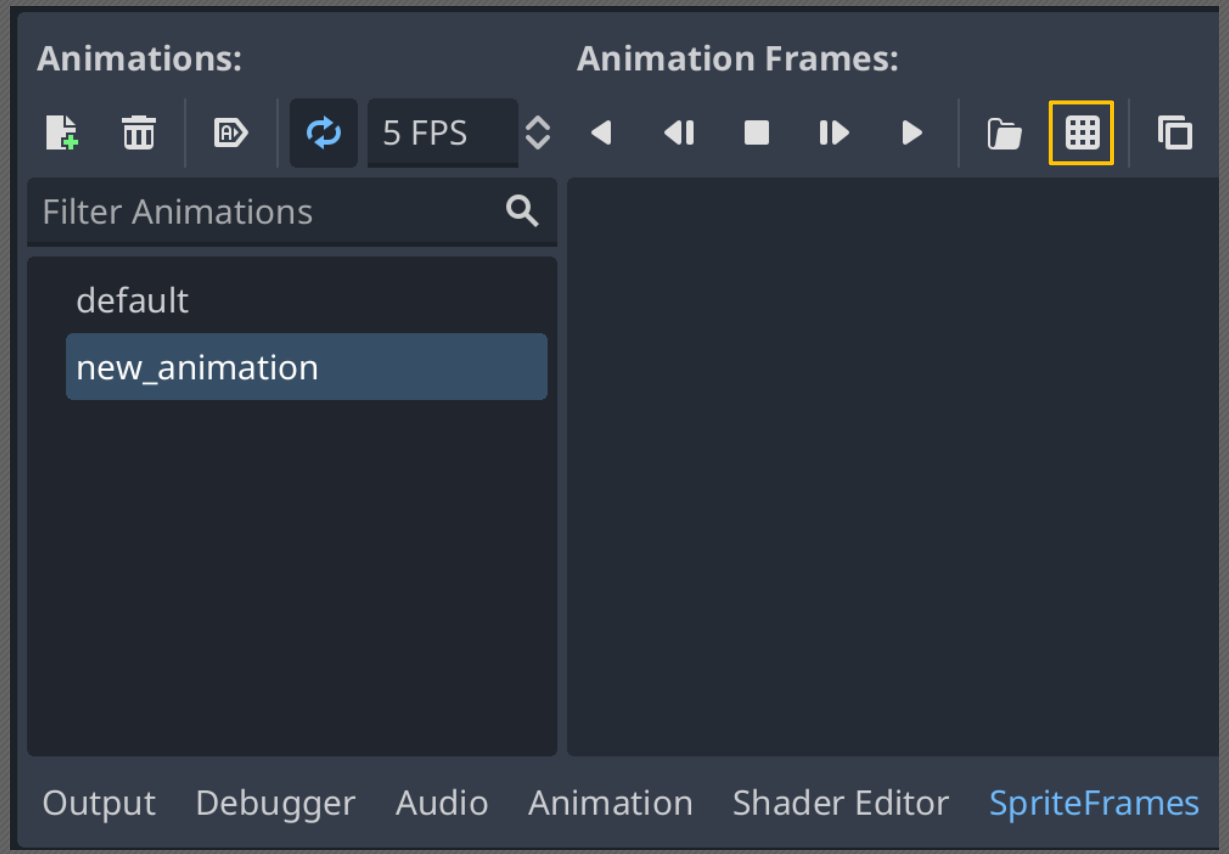


Creating an Idle animation

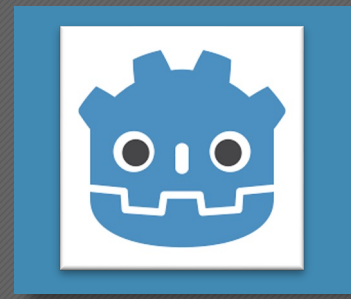


Creating an Idle animation

- Delete the default animation by selecting it and clicking the Trash icon.
- Rename the “new_animation” animation to “idle” by selecting that animation in the list and then clicking the name again.
- Click the “Add Frames from a Sprite Sheet” button to add frames the *currently selected* “idle” animation.



Creating an Idle animation



Open the “Idle (32x32).png” sprite sheet image from your choice of one of the four character directories.
res://Assets/Art/Pixel Adventure 1/Main Characters/*{CHARACTER}*/Idle (32x32).png

Mask Dude

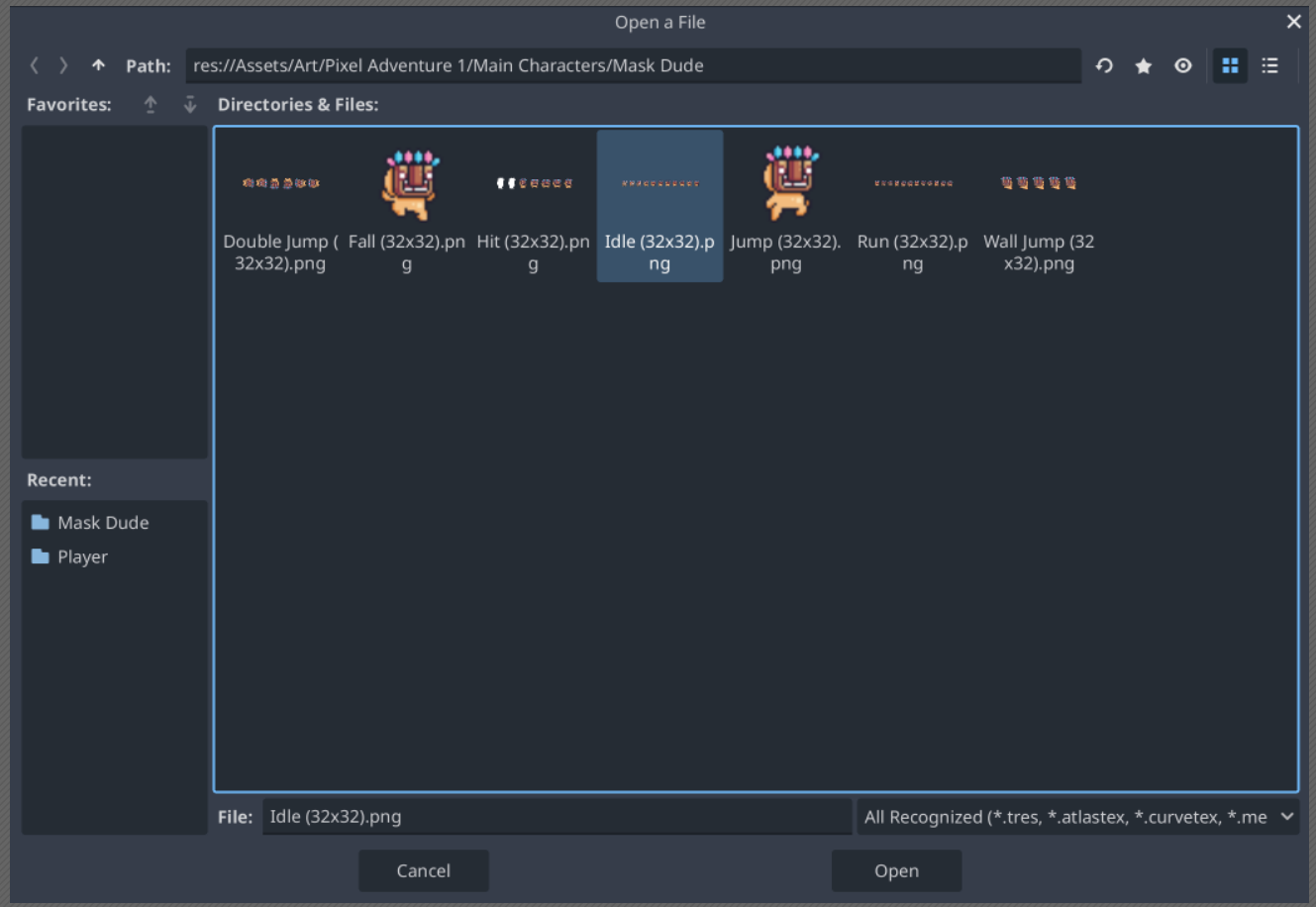
Ninja Frog

Pink Man

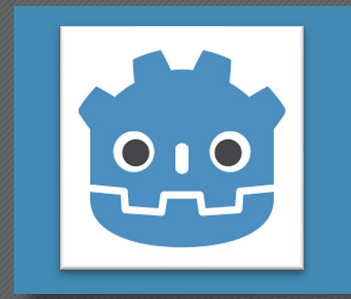
Virtual Guy

Creating an Idle animation

Open the “Idle (32x32).png” sprite sheet image from your choice of one of the four character directories.
res://Assets/Art/Pixel Adventure 1/Main Characters/*{CHARACTER}*/Idle (32x32).png

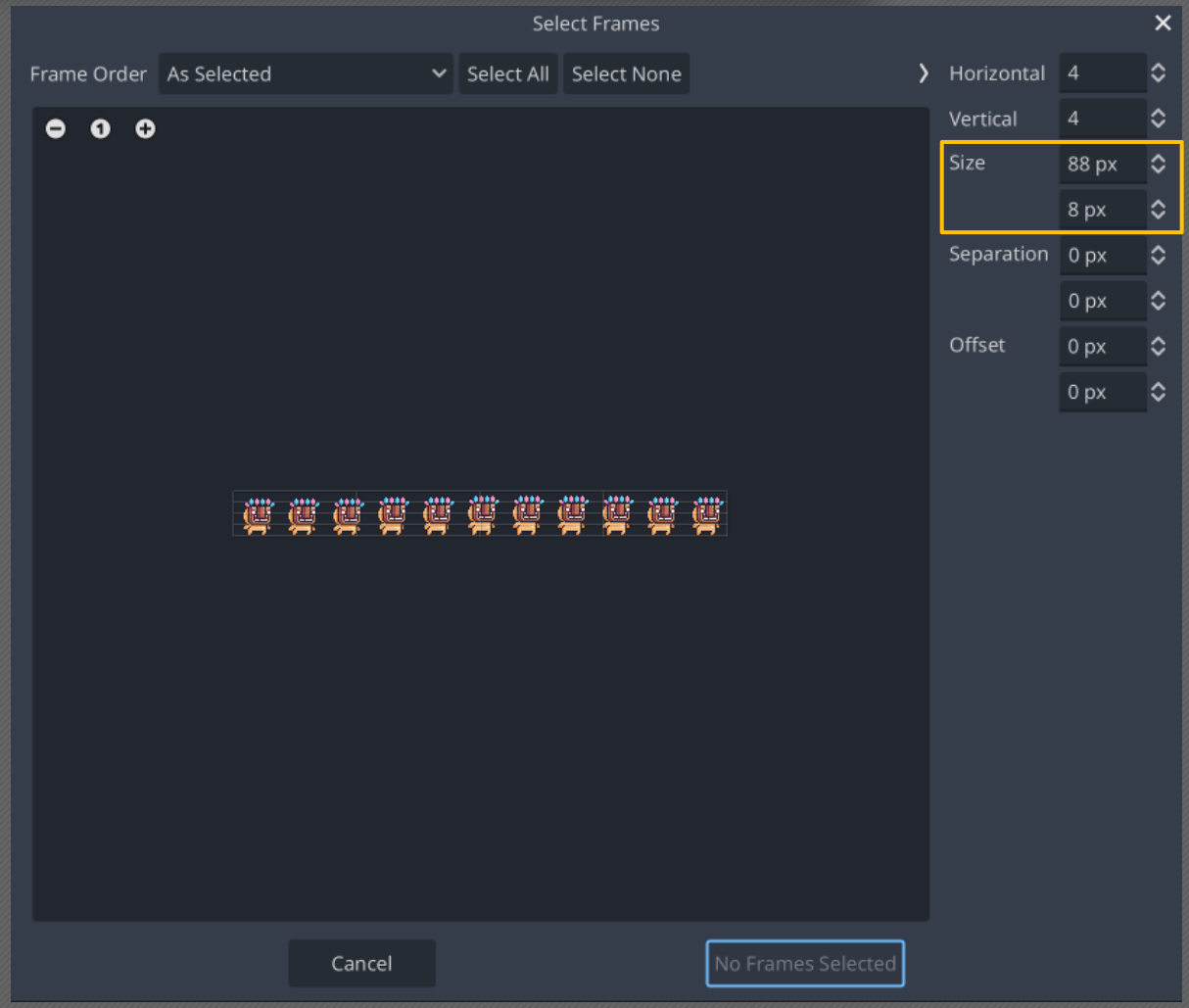


Creating an Idle animation



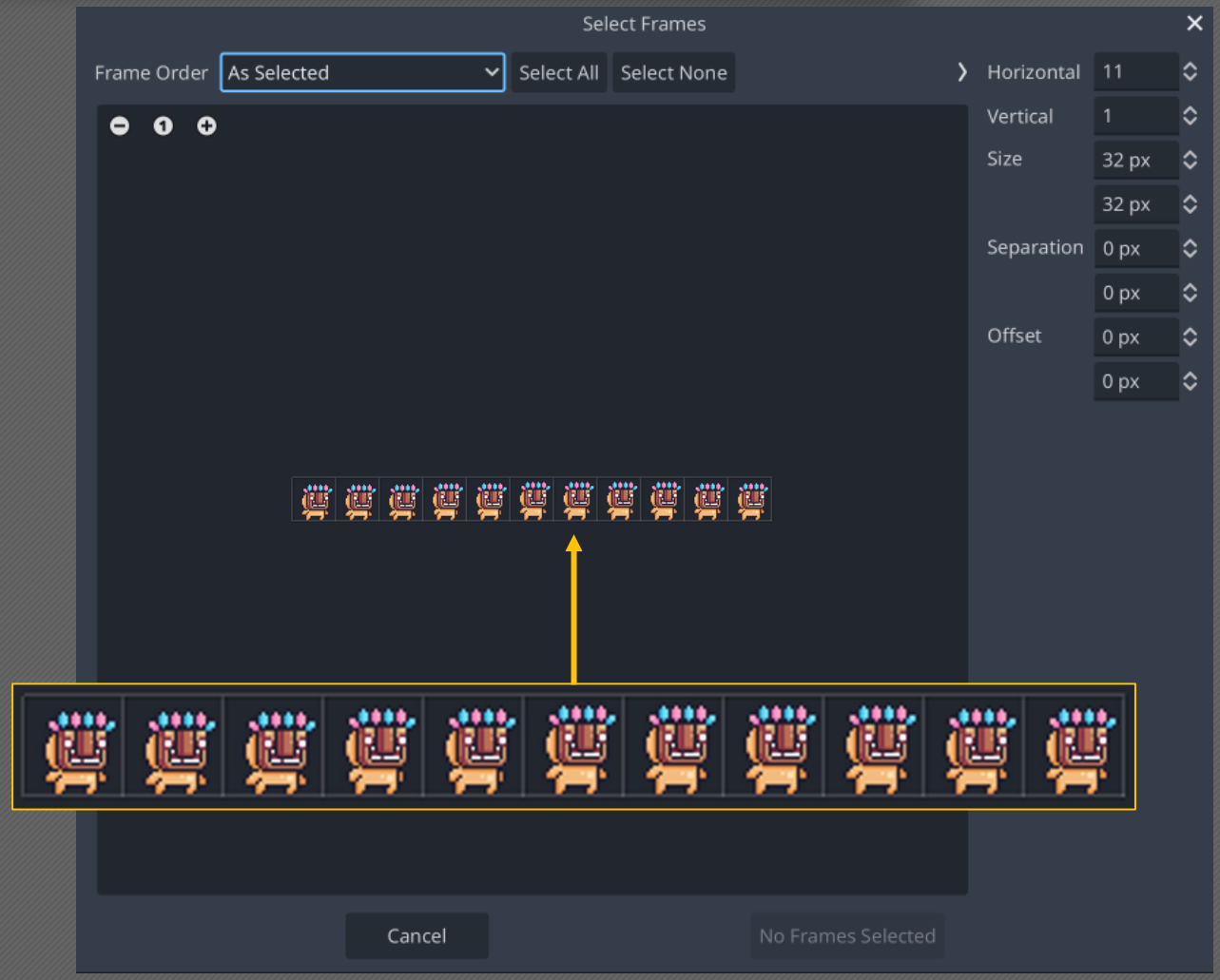
- This Frame selection dialog lets us select which frames we want to add to our animation based on the selected sprite sheet file.

- Adjust the size the sprite sheet tiles to a value of “32” for both Size input fields.



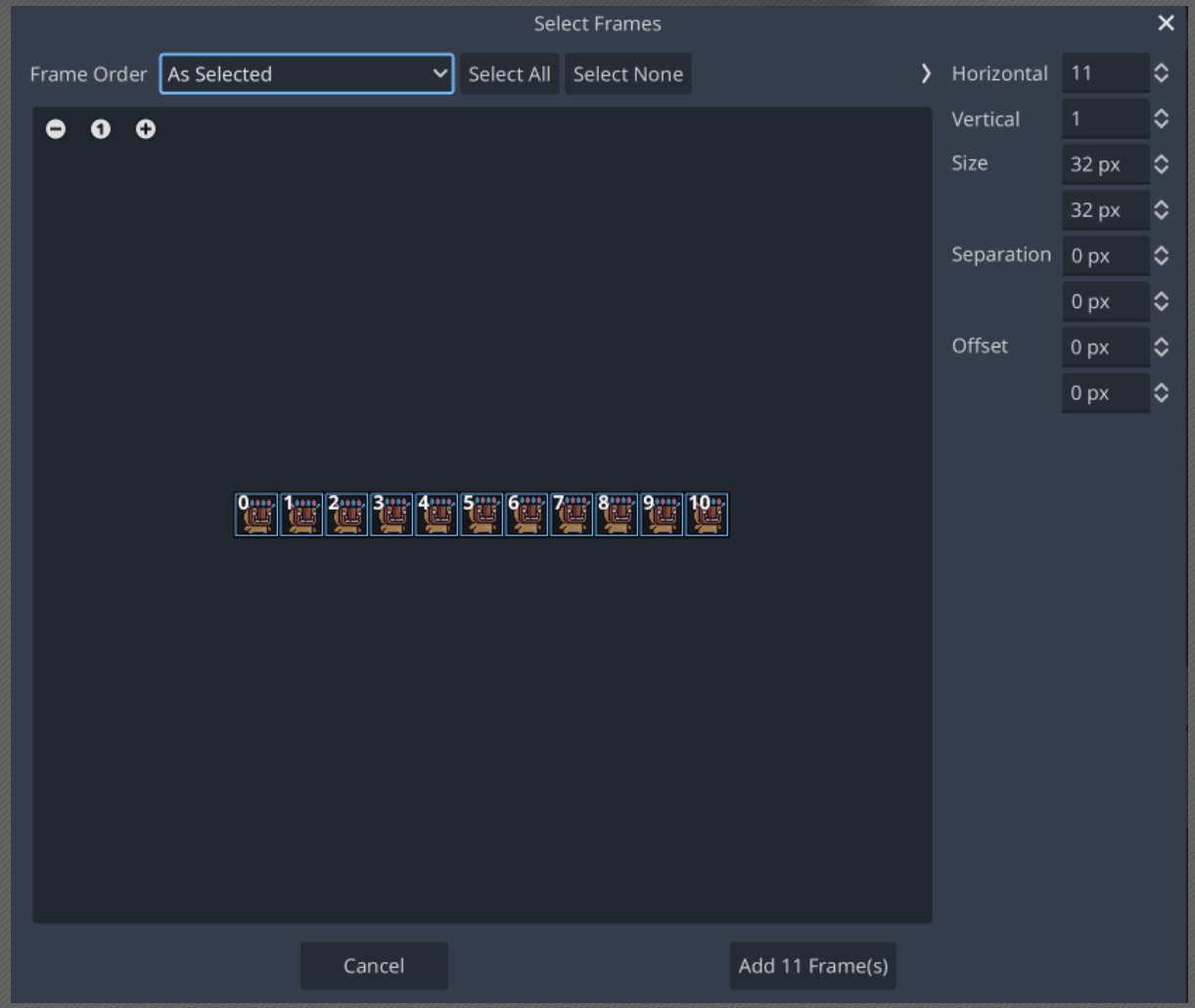
Creating an Idle animation

- Once the sprite Size is set properly, we can select which parts of the sprite sheet to add to the Idle animation.
- Going left to right, select all frames by left-clicking in the area of each individual sprite square.
- You may also hold left-click and drag the cursor through the other sprite areas to select them.



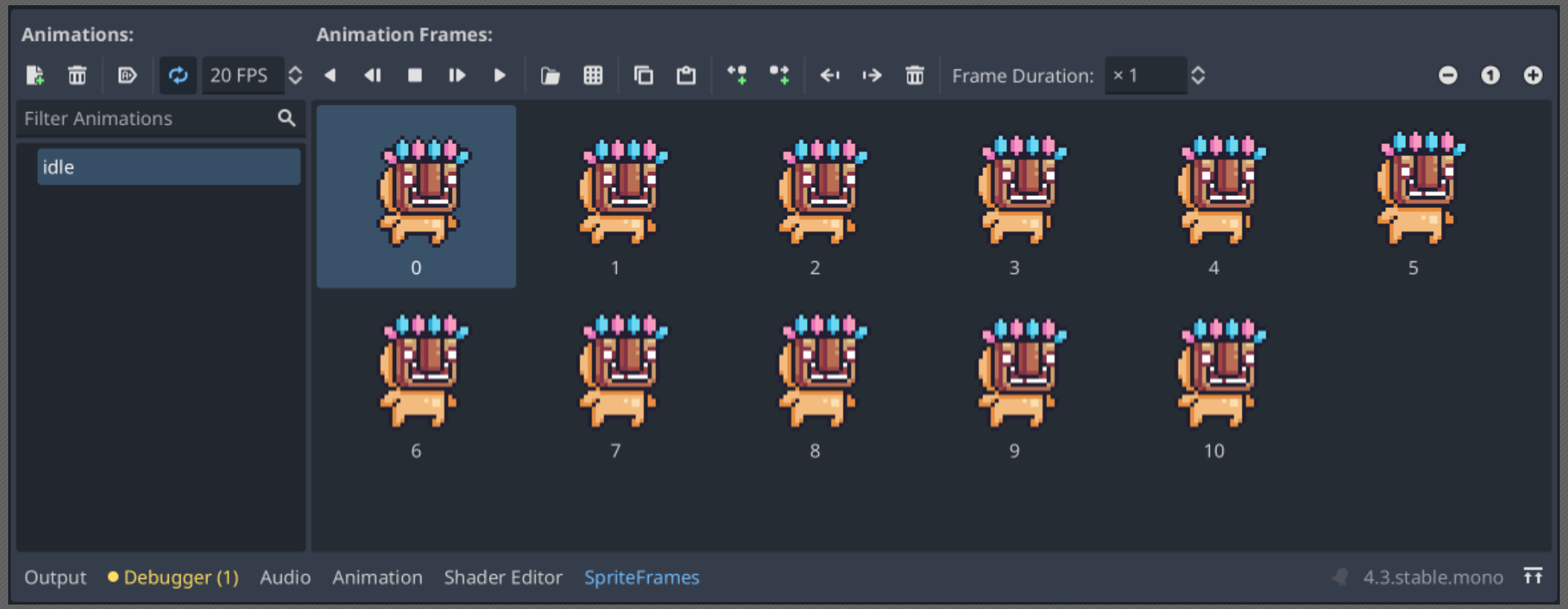
Creating an Idle animation

- The order in which you select the squares matters, as this is the order that the sprite frames will be added to the animation.
- The top-left number shows the order in which each sprite will be added to the animation frames.
- If the order is not right, deselect the incorrect sprites and reselect them in the proper order.
- Once all are selected properly, click the “Add 11 Frame(s)” button.



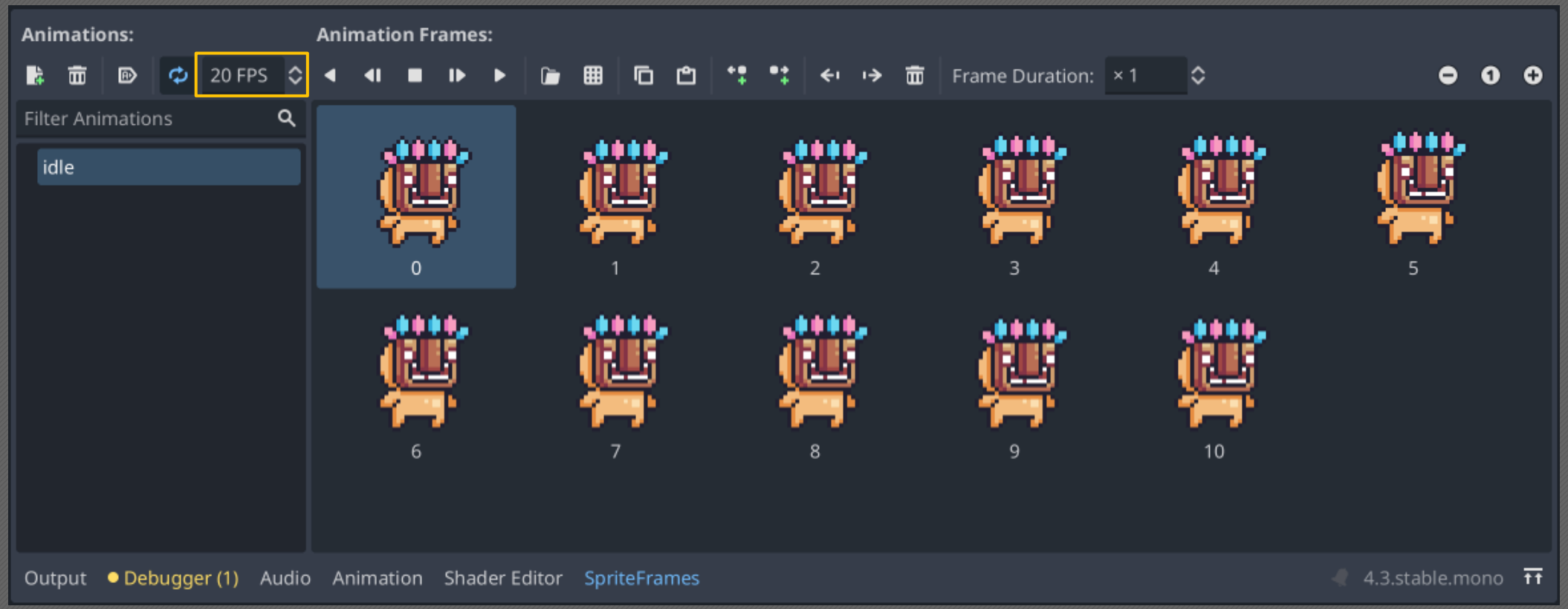
Creating an Idle animation

The sprites from the previous prompt are added to the idle animation in the order they were selected in that prompt.

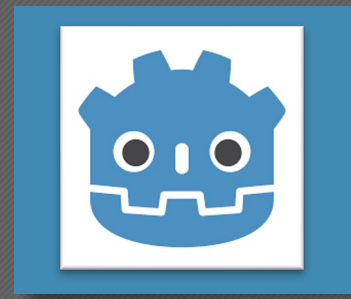


Creating an Idle animation

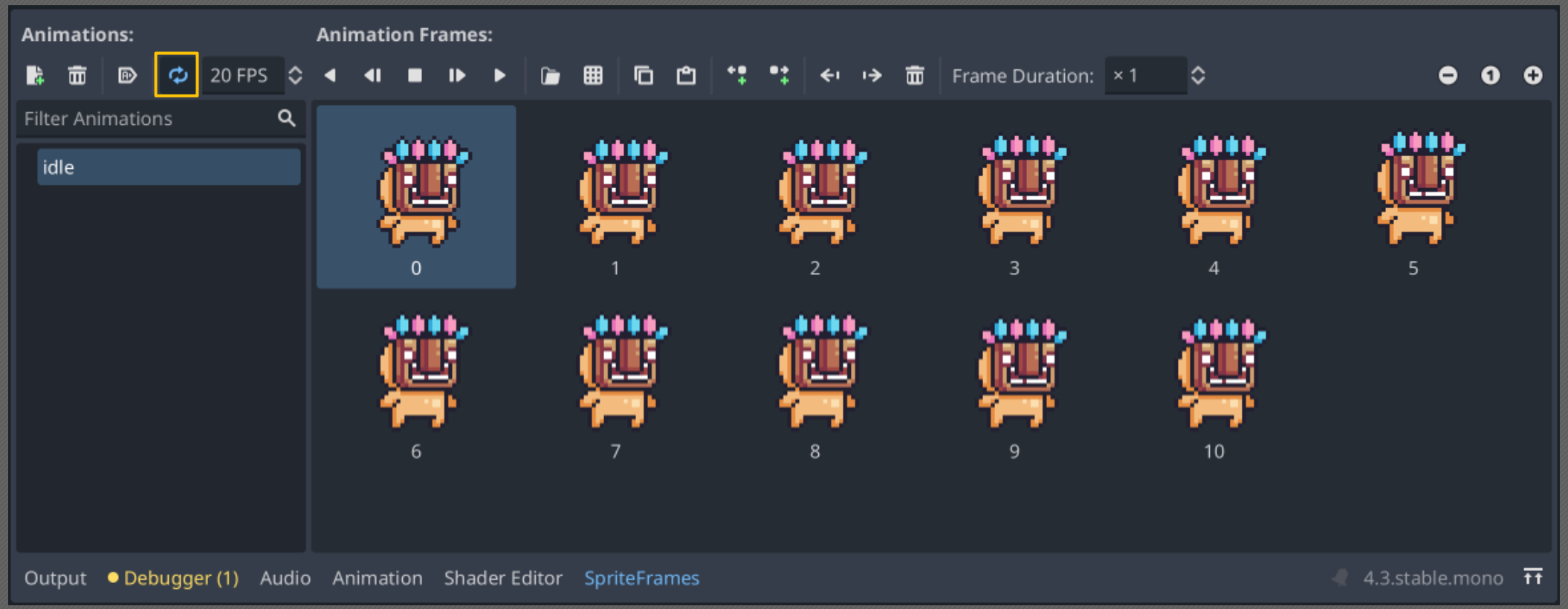
The default animation speed is 5 FPS (frames per second). This is going to be a very slow animation at that speed. Update the FPS to be 20 instead of 5.



Creating an Idle animation



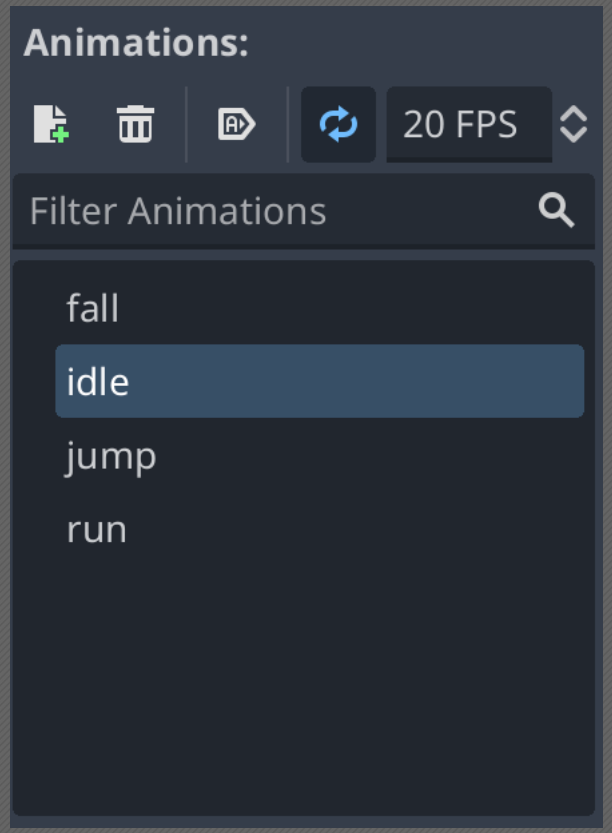
This button makes the animation Loop, meaning the selected animation will automatically restart and replay itself once the animation completes the final frame. Keep this on for idle.



Add More Animations

- Now that the idle animation is complete, there are 3 other basic animations we'll need to animate our character.

- Follow the same process for a fall, jump, and run animation.



Programming the Animations

```gdscript
16      move_and_slide()
17
18      # If this node's X velocity is negative, we're moving left and should flip the sprite.
19      if velocity.x < 0:
20          $AnimatedSprite2D.flip_h = true
21      elif velocity.x > 0:
22          $AnimatedSprite2D.flip_h = false
23
24      # A negative y velocity means this node is moving UP, so we play the jump animation.
25      if velocity.y < 0:
26          $AnimatedSprite2D.play("jump")
27      elif velocity.y > 0:
28          $AnimatedSprite2D.play("fall")
29      elif velocity.x != 0:
30          $AnimatedSprite2D.play("run")
31      else:
32          $AnimatedSprite2D.play("idle")
33
```

Programming the Animations

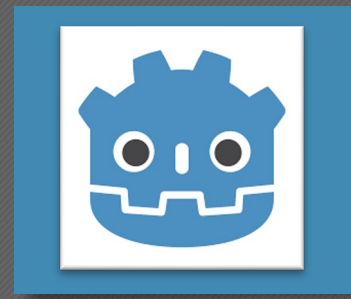
You can reference nodes in the scene tree by using a $ symbol followed by the *full* node path in the scene tree.

The game will crash at runtime if a node is accessed this way which does not exist as a child element of the node the script is attached to.

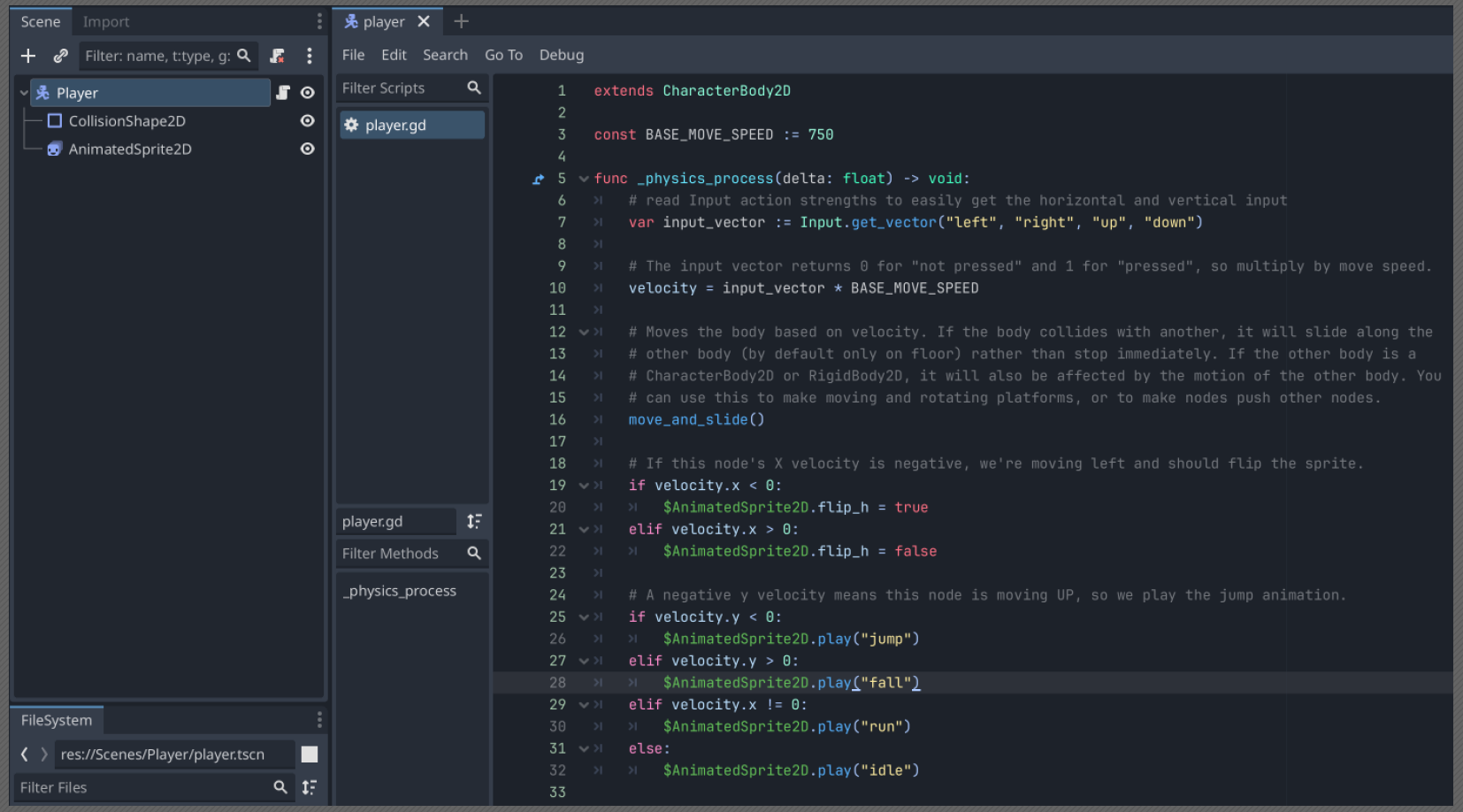
On the AnimatedSprite2D node, we can call the play method to play a specific animation that exists on the SpriteFrames resource.

```
16      move_and_slide()
17
18      # If this node's X velocity is negative, we're movin
19      if velocity.x < 0:
20          $AnimatedSprite2D.flip_h = true
21      elif velocity.x > 0:
22          $AnimatedSprite2D.flip_h = false
23
24      # A negative y velocity means this node is moving UP
25      if velocity.y < 0:
26          $AnimatedSprite2D.play("jump")
27      elif velocity.y > 0:
28          $AnimatedSprite2D.play("fall")
29      elif velocity.x != 0:
30          $AnimatedSprite2D.play("run")
31      else:
32          $AnimatedSprite2D.play("idle")
33
```

Clean Up the Scene Tree

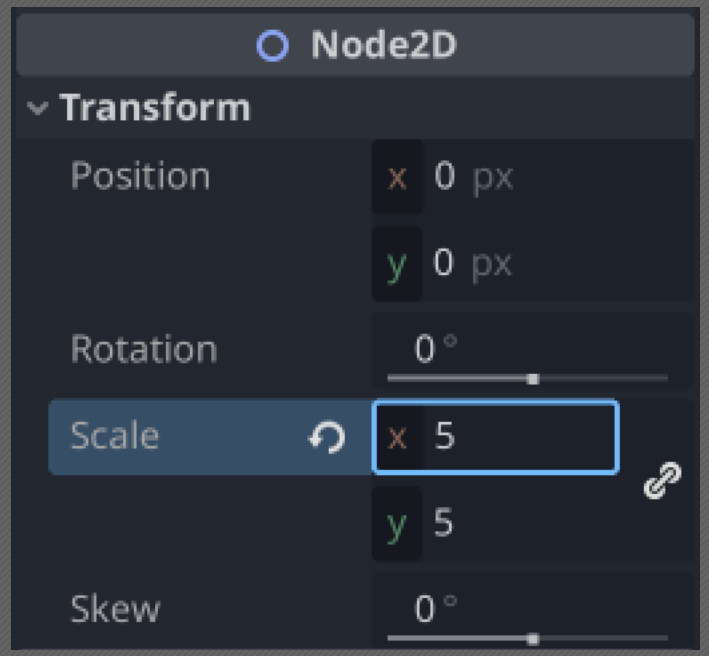


Delete the Sprite2D node now that we have an animated character.

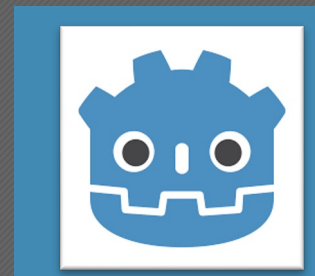


Increase the AnimatedSprite2D Scale

- The sprite frames used in the animations are 32x32, which is going to look very small in-game.
- The Scale property is a multiplier for the X/Y size (X/Y/Z for 3d) of the node.
- Update the Scale to 5. This will make our 32px x 32px character be 160x160 (32x5 for X and Y).



Lab Time (~15 Minutes)

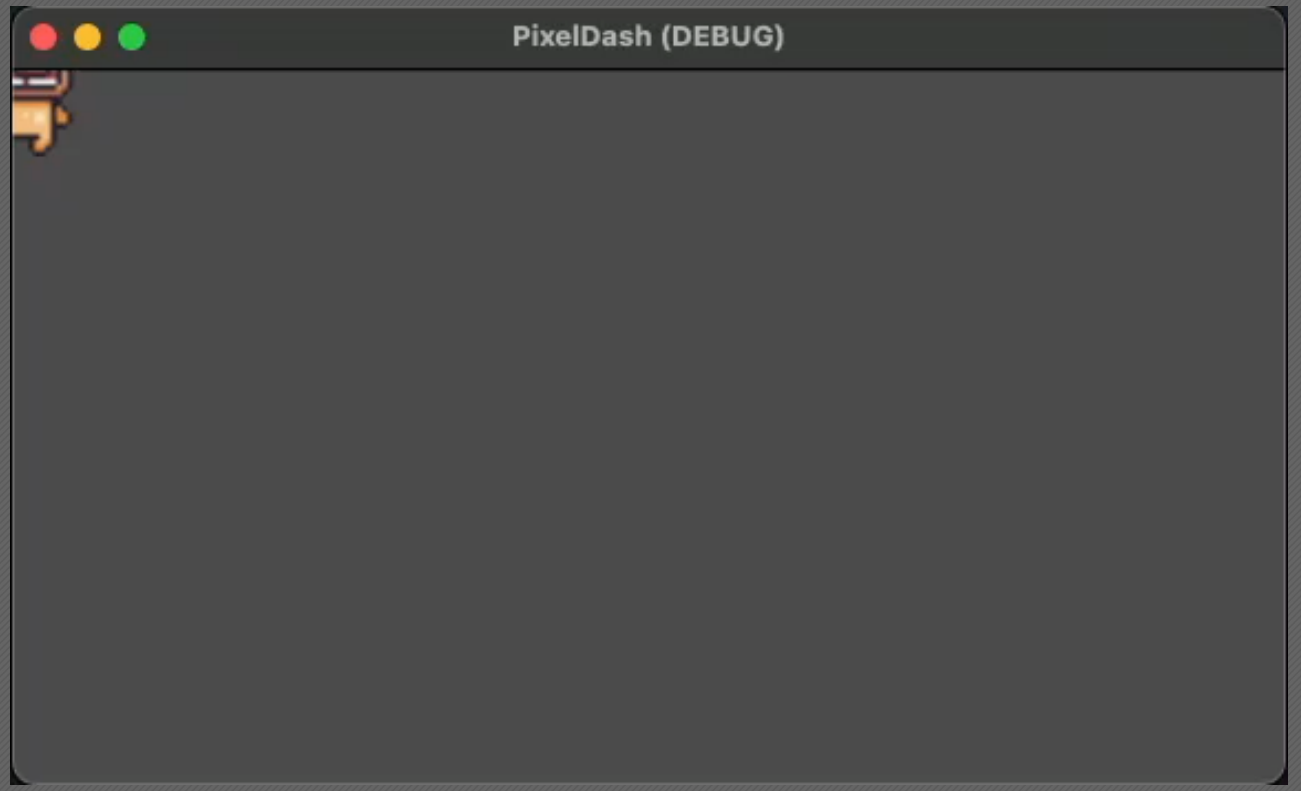


- Add an AnimatedSprite2D to the Player scene.

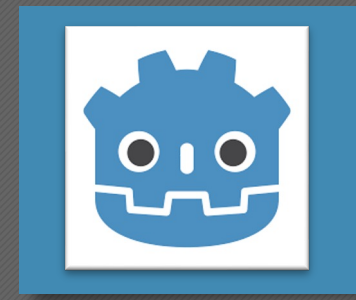
- Add the following four animations:
  - idle (loop ON, 20 FPS)
  - run (loop ON, 20 FPS)
  - fall (loop OFF, 5 FPS)
  - jump (loop OFF, 5 FPS)

- Increase the Player scale to 5.

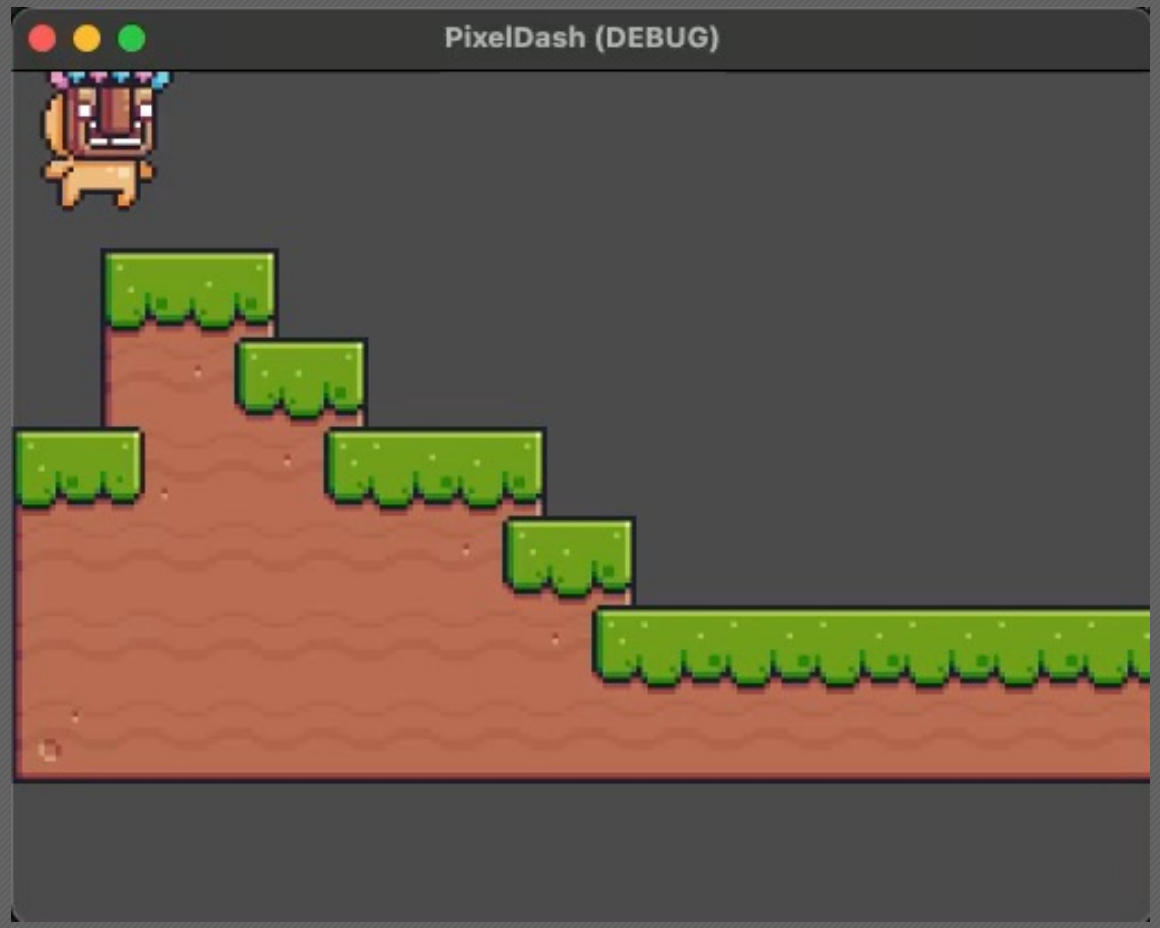
- Delete the Sprite2D from the Player scene.



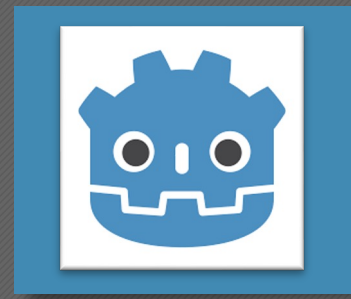
Workshop Goal #2



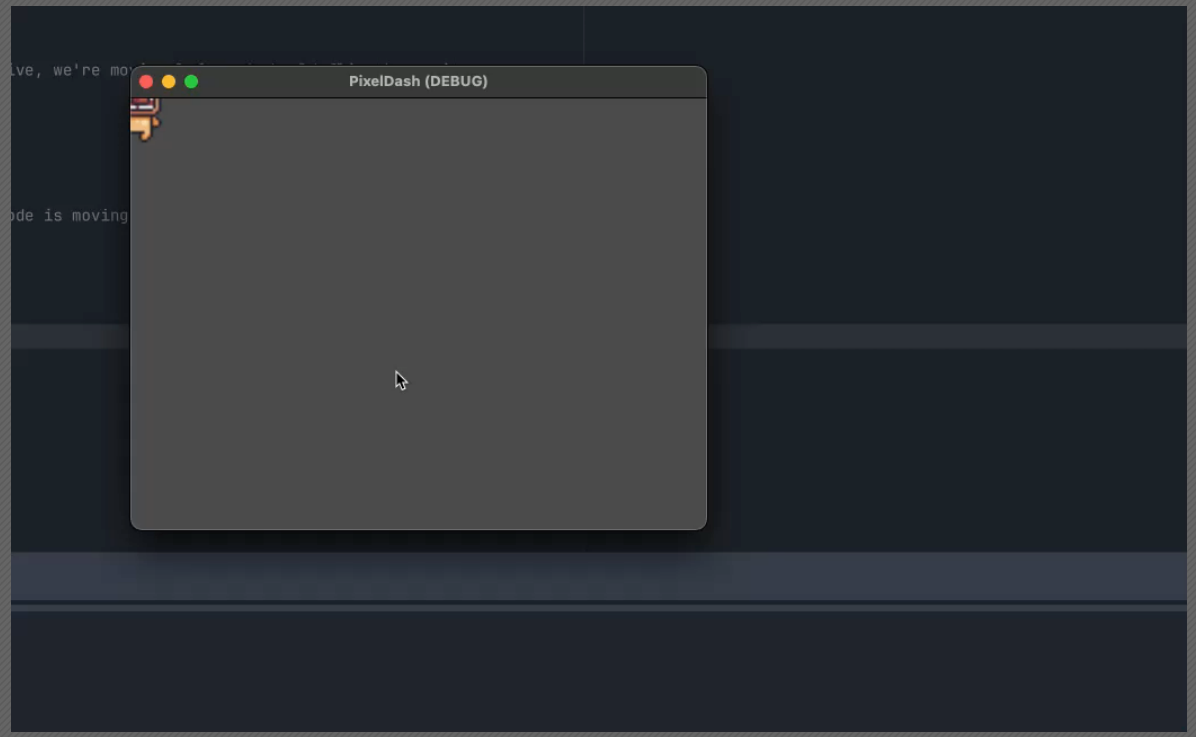
1. ✅ Control & Animate a Character
2. ➡️ Build a Level with Tiles
3. Add Player & Level Polish
   (Camera, Gravity, Sound, Background)
4. Detect Collisions with Trampolines
5. Create a basic Mushroom Enemy
6. Add Collectible Fruit and HUD Display
7. Respawn Character when Defeated
8. Create a Main Menu & Change Scenes
9. Open Workshop, Tinker Time



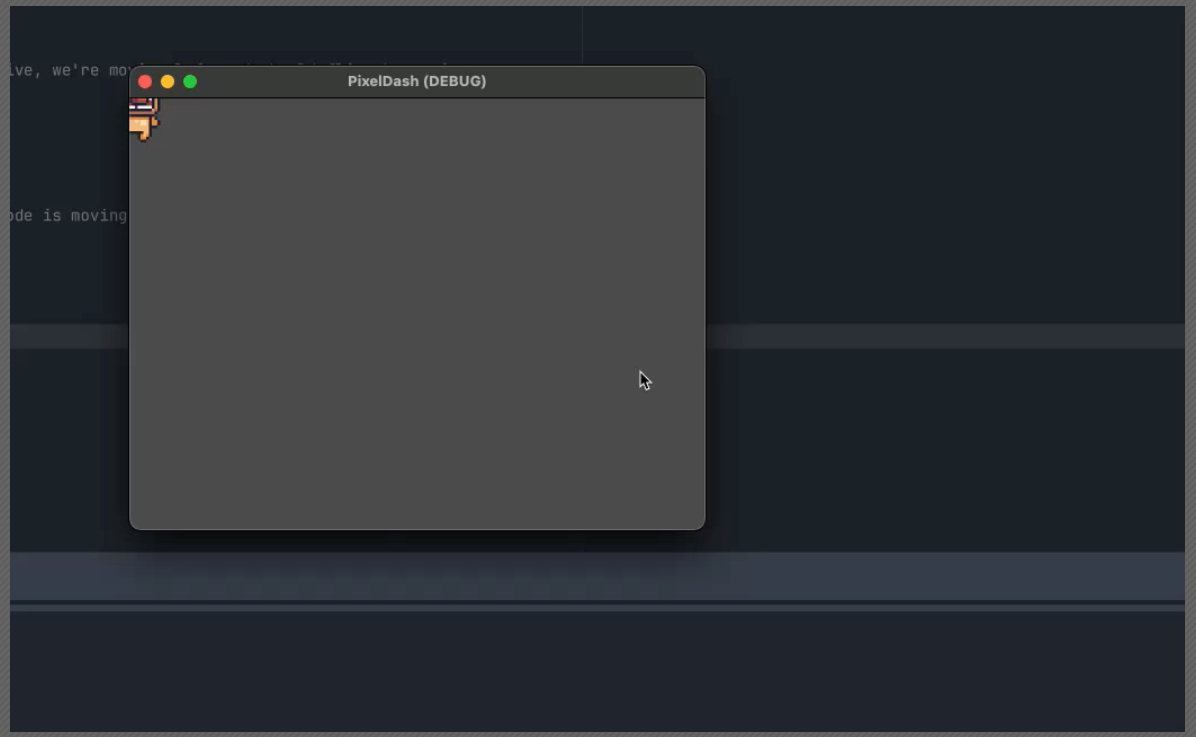
Project Settings: Art Style & Window Sizing



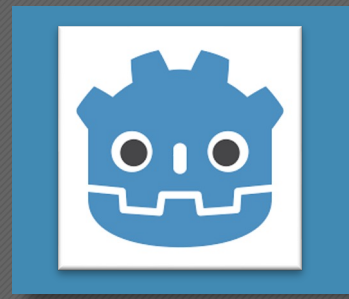
Currently, pixels are ”fuzzy” and window borders *extend* the play area.



We want ”crispy” pixels and a consistently-sized play area.



Project Settings



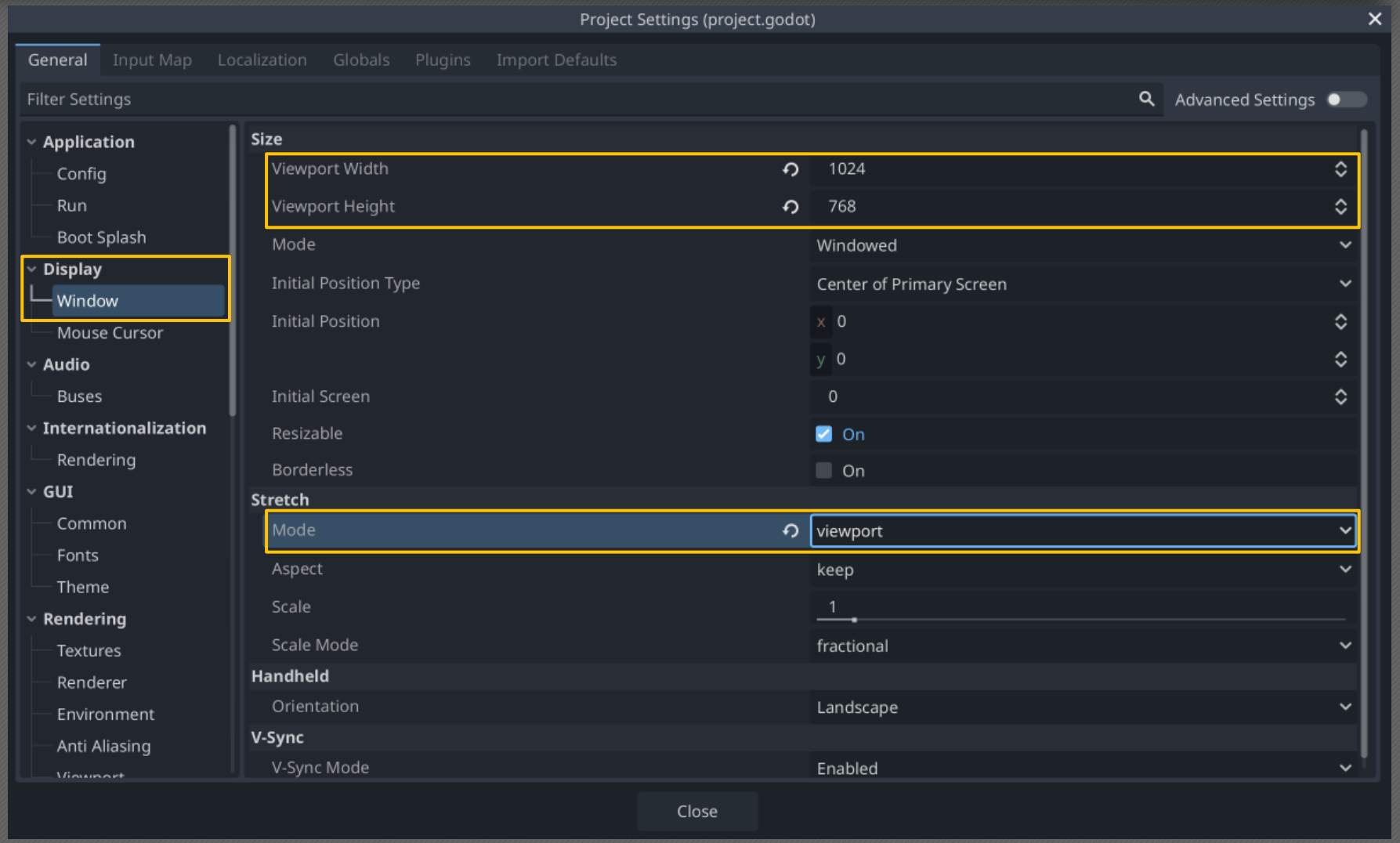
In order to give our playable game window a consistent and standardized size, we need to update a couple project settings.

In the General settings tab, under Display > Window:

- The Viewport Width & Viewport Height settings control the game screen dimensions. Set this to a common resolution, like 1920x1080 or 1280x720.

- A Stretch Mode of “disabled” means the playable game area does not up- or down-scale itself when the window size changes, meaning the game screen is variable. Updating the Stretch Mode to “viewport” will force the game to scale itself to fit the screen based on the above width and height settings.

Project Settings: Display > Window



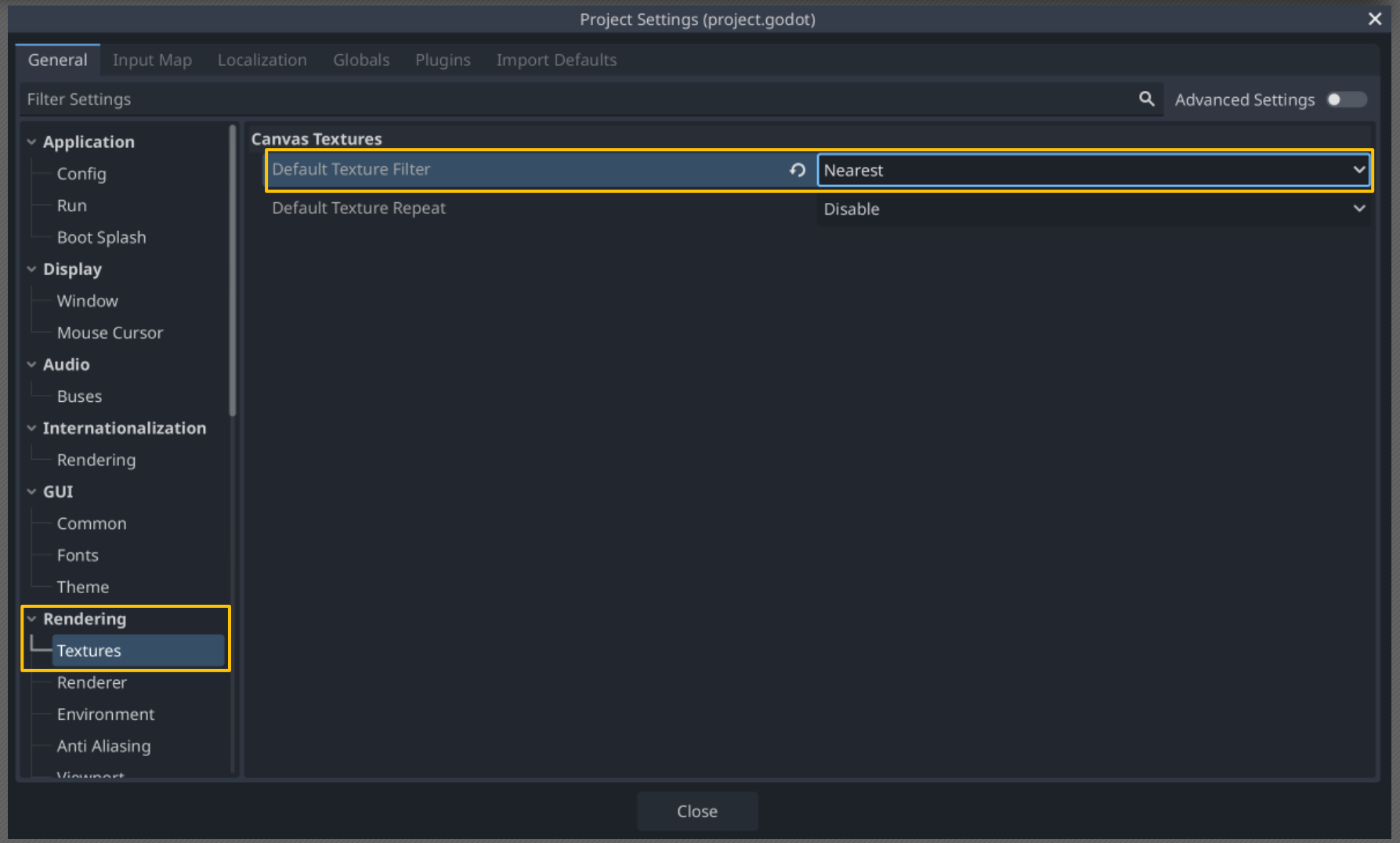
Project Settings

The default texture rendering is “Linear”, which blends the individual pixels in textures to the nearest 4 pixels, which gives our sprite images a smooth or blurry look.

Since we’re working with relatively small pixel art graphics, we may want to opt for that non-blurred, crispy pixel art aesthetic. The “Nearest” filter value blends only the nearest 1 pixel, which will result in no blurring in our 2D textures.

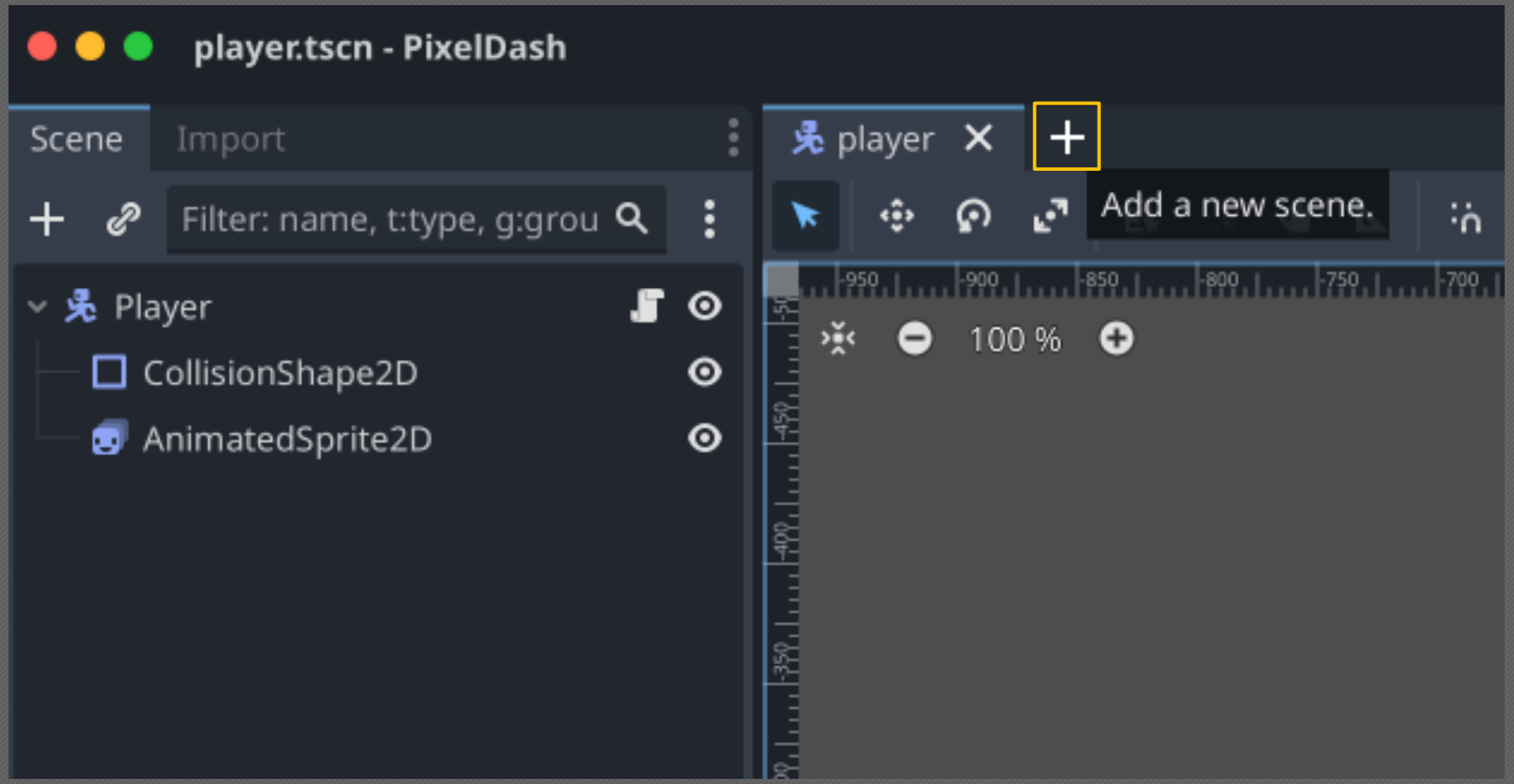
Change the Default Texture Filter to “Nearest”, under Rendering > Textures.

Project Settings: Rendering > Textures



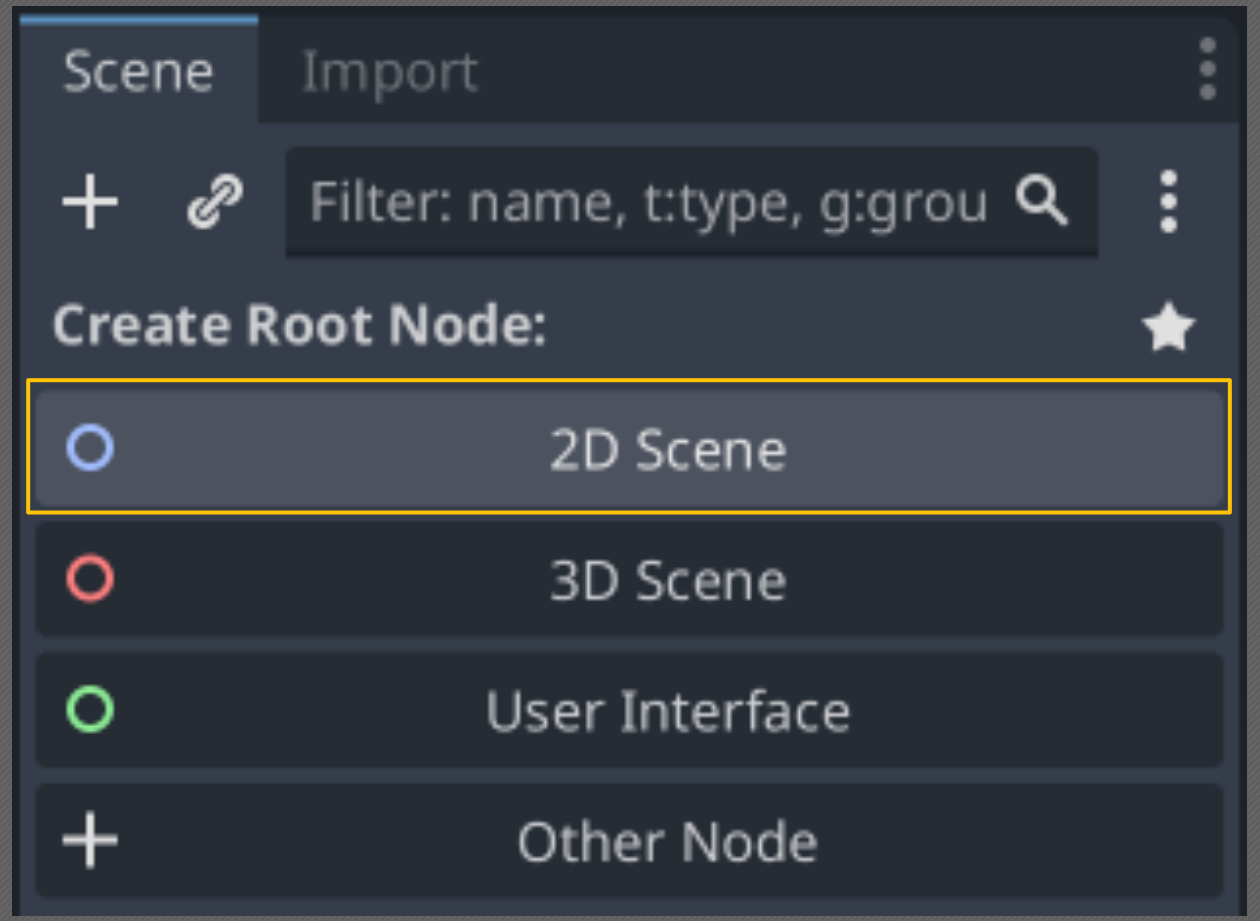
Creating a New Scene

Click the + Add button to the right of the player scene tab to create a new scene.



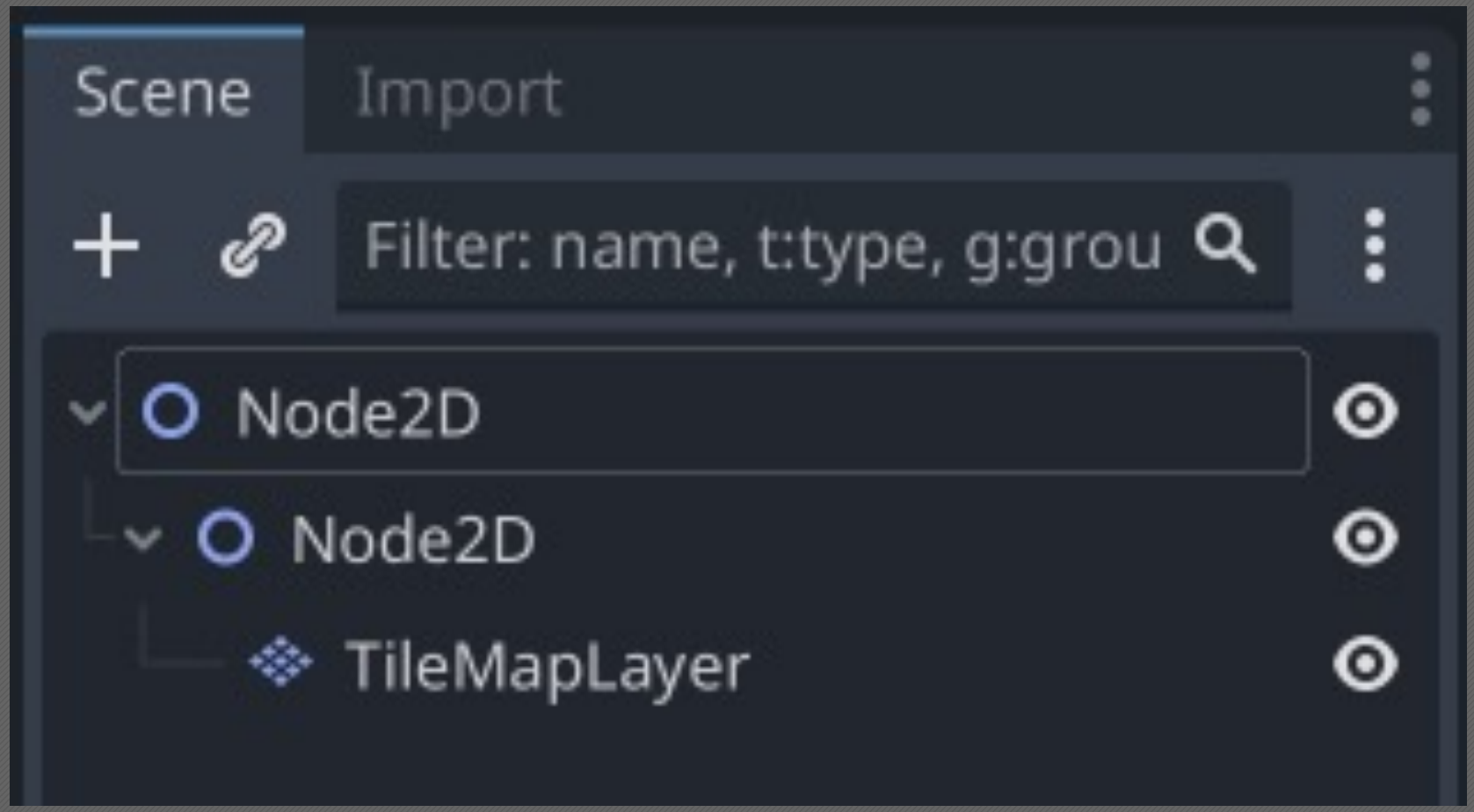
Creating a New Scene

The Scene Tree will ask for a Root Node type; select 2D Scene.



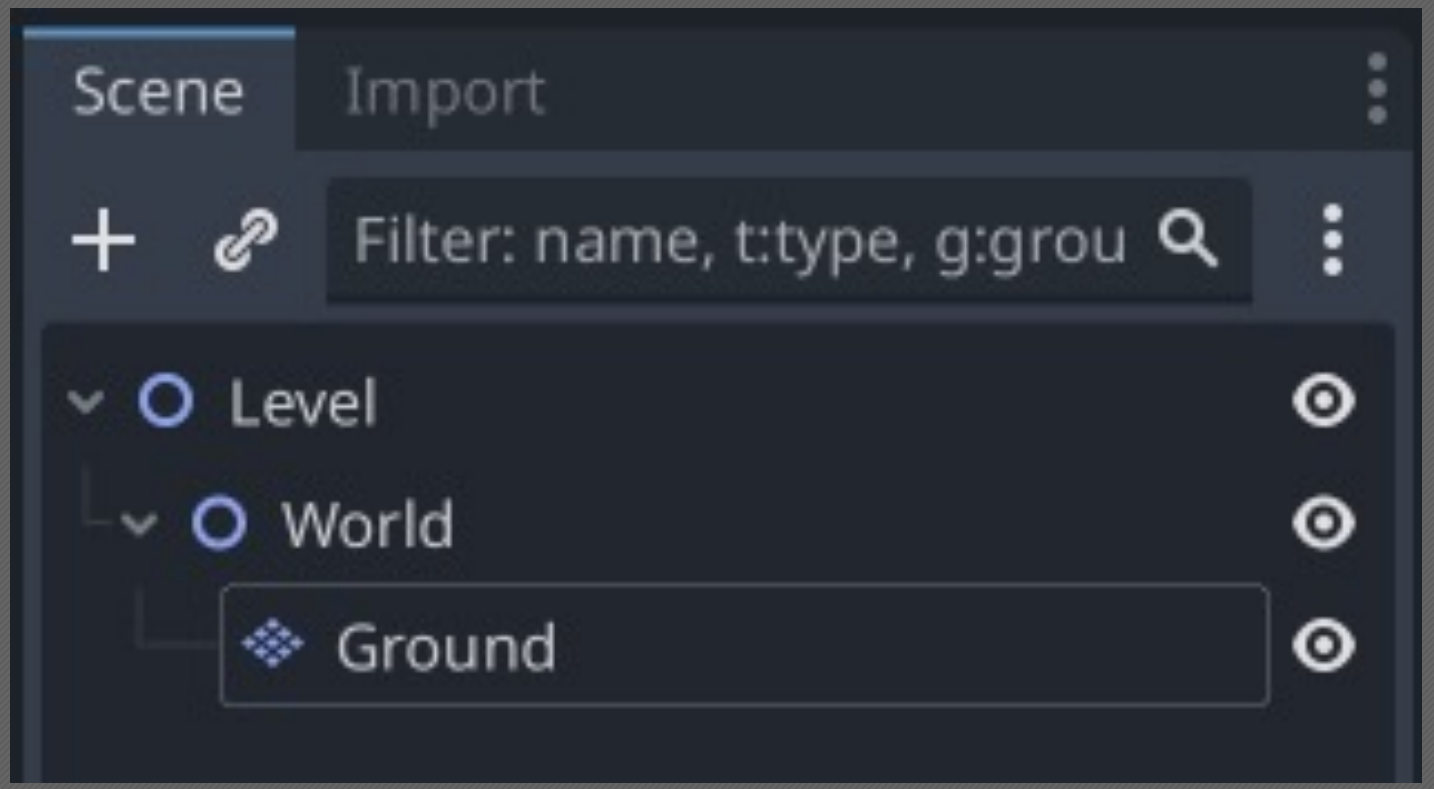
Set Up the new Scene Tree

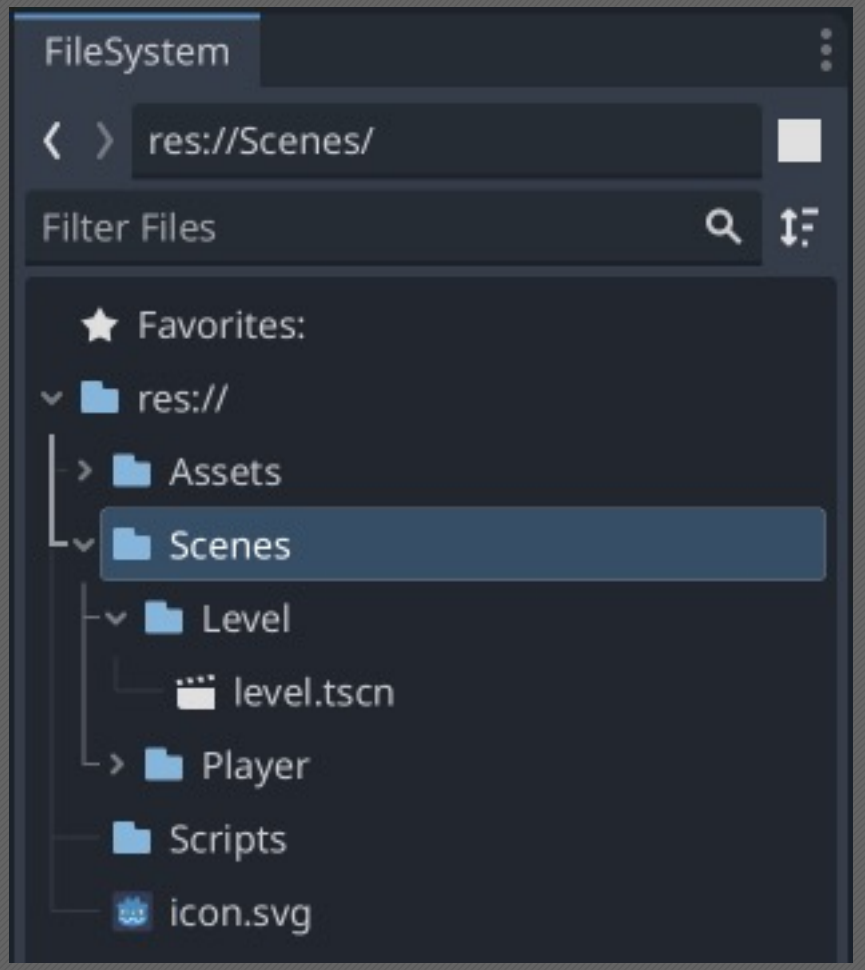
Create the following scene tree structure in the new scene. Notice the TileMapLayer is a child of the Node2D, which is a child of the root Node2D.



Save the new Level Scene

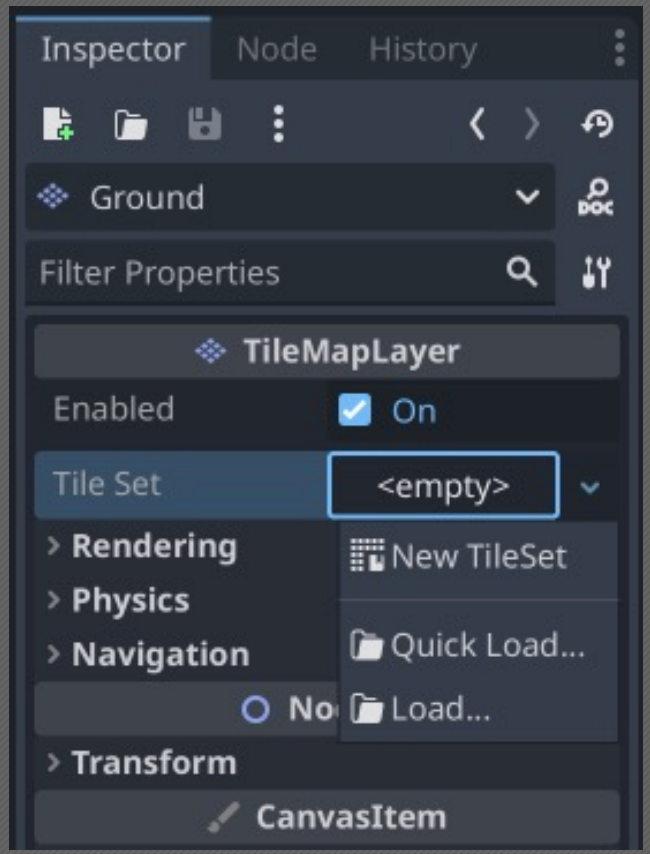
Rename the nodes and save the scene in a new “Level” folder under res://Scenes.



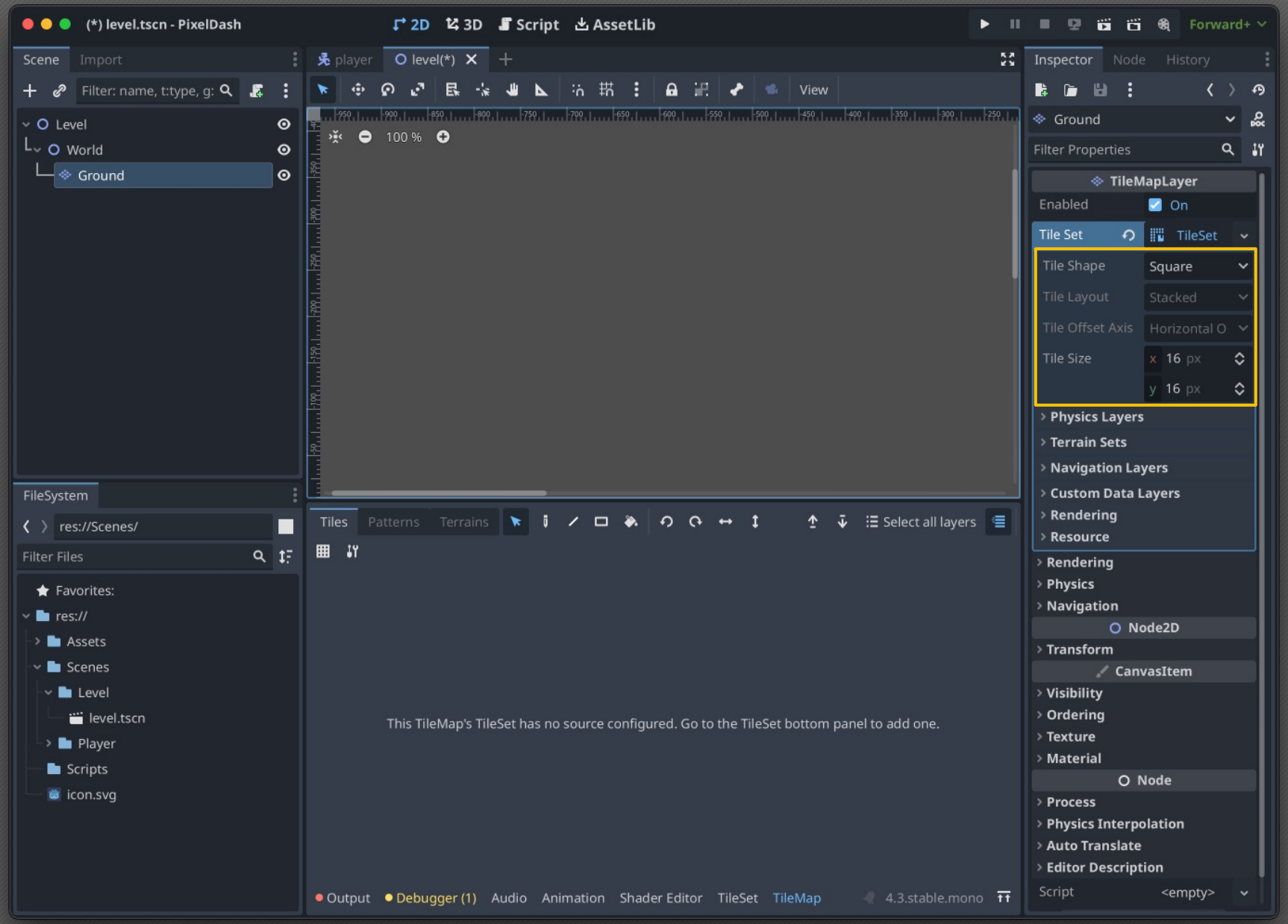


Create a TileSet for the Ground Node

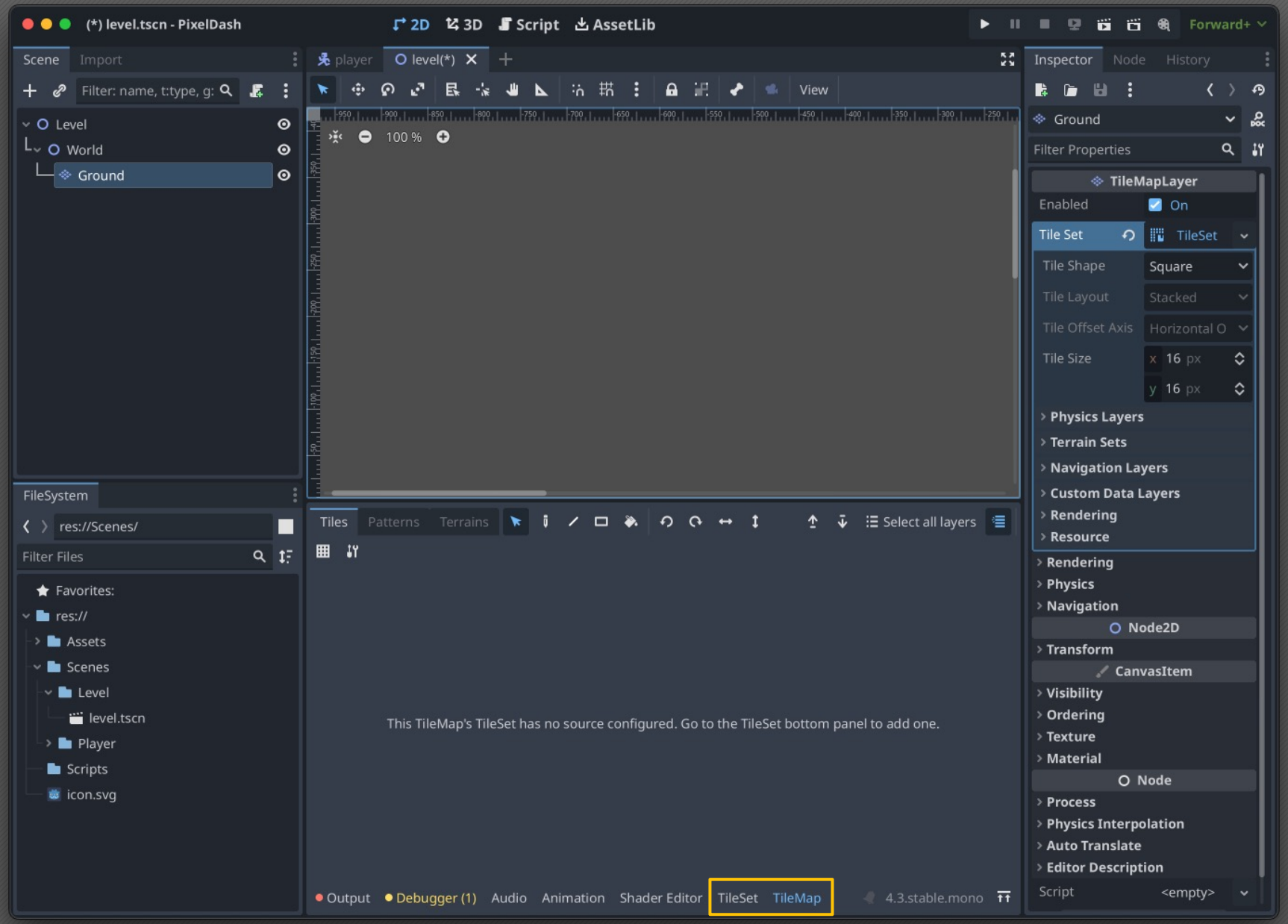
- The Ground node is a TileMapLayer, which is a 2D tile-based map on which tiles can be drawn at grid positions.
- The TileMapLayer uses a TileSet, which is the definition Resource for what tiles can be drawn onto the TileMapLayer.
- Create a new TileSet for the Ground's Tile Set property.



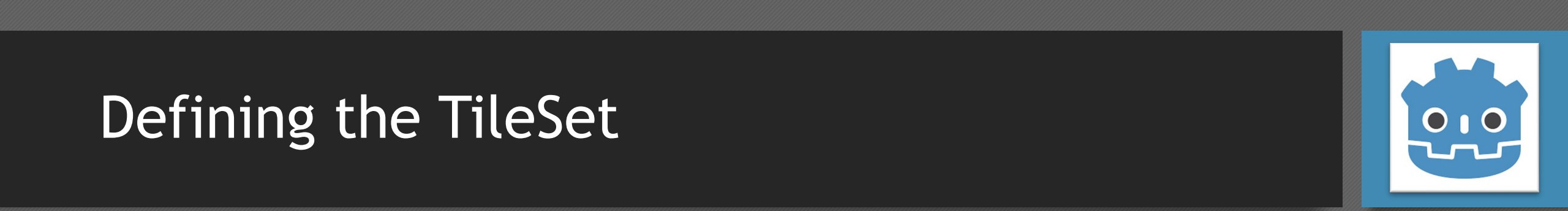
The default Tile Shape of Square, and Tile Size of 16x16 is correct!



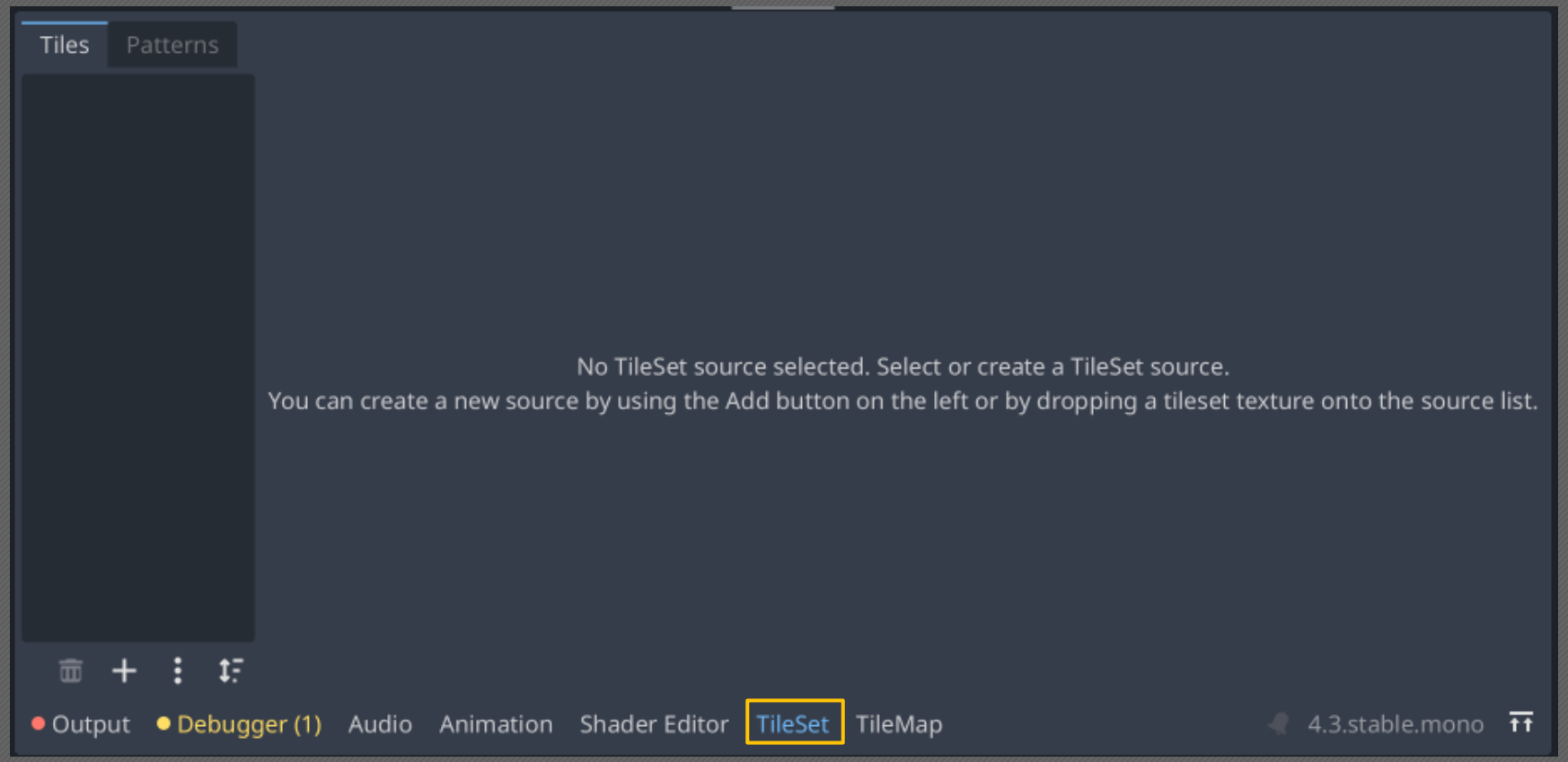
Notice the Console area at the bottom gets two new tabs when a TileMapLayer is selected.



Defining the TileSet



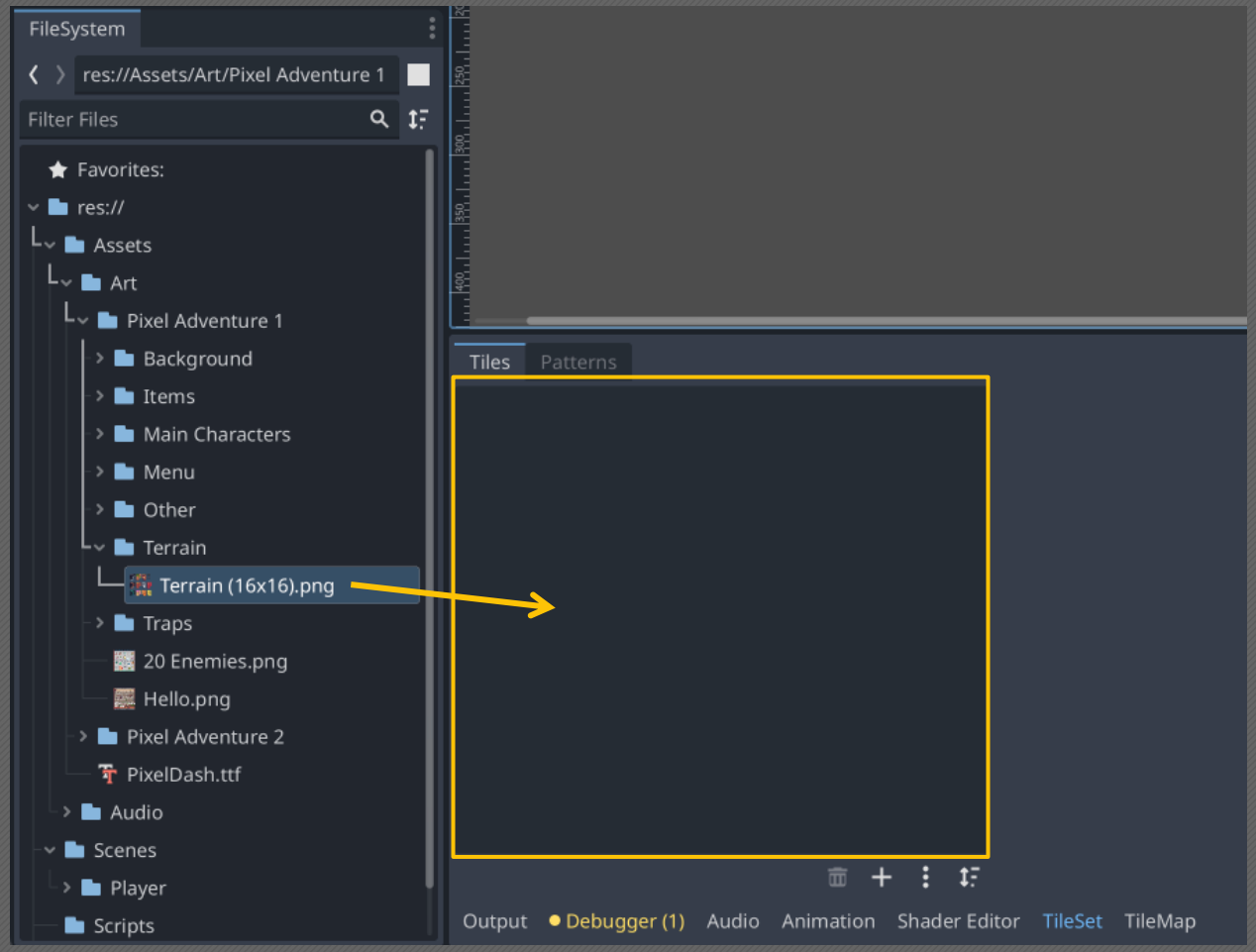
With the Ground node selected (a TileMapLayer), make sure the TileSet tab in the Console Area is the active selected tab.



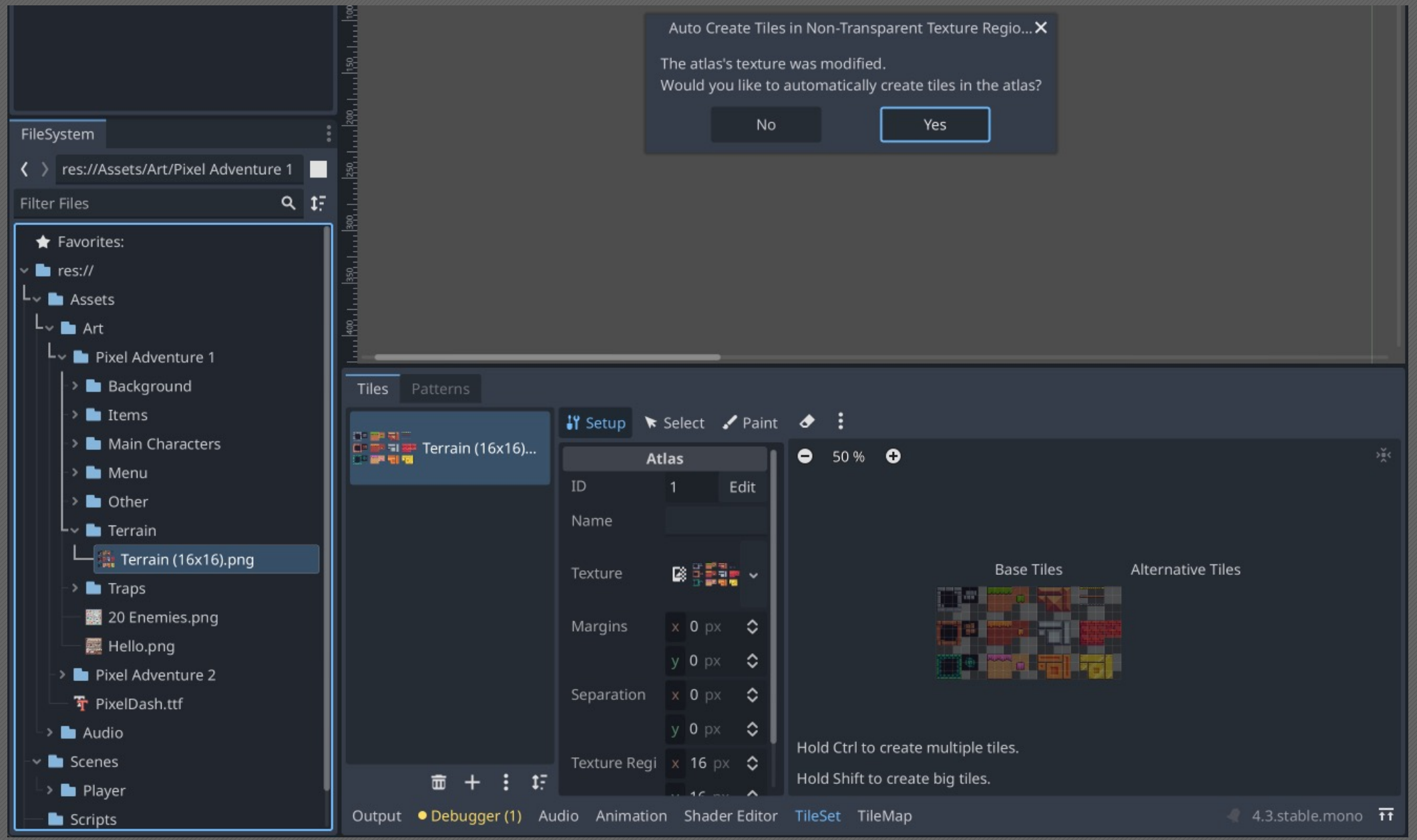
Defining the TileSet

- To define the tiles within the TileSet, we'll be using the Terrain (16x16) Sprite Sheet.

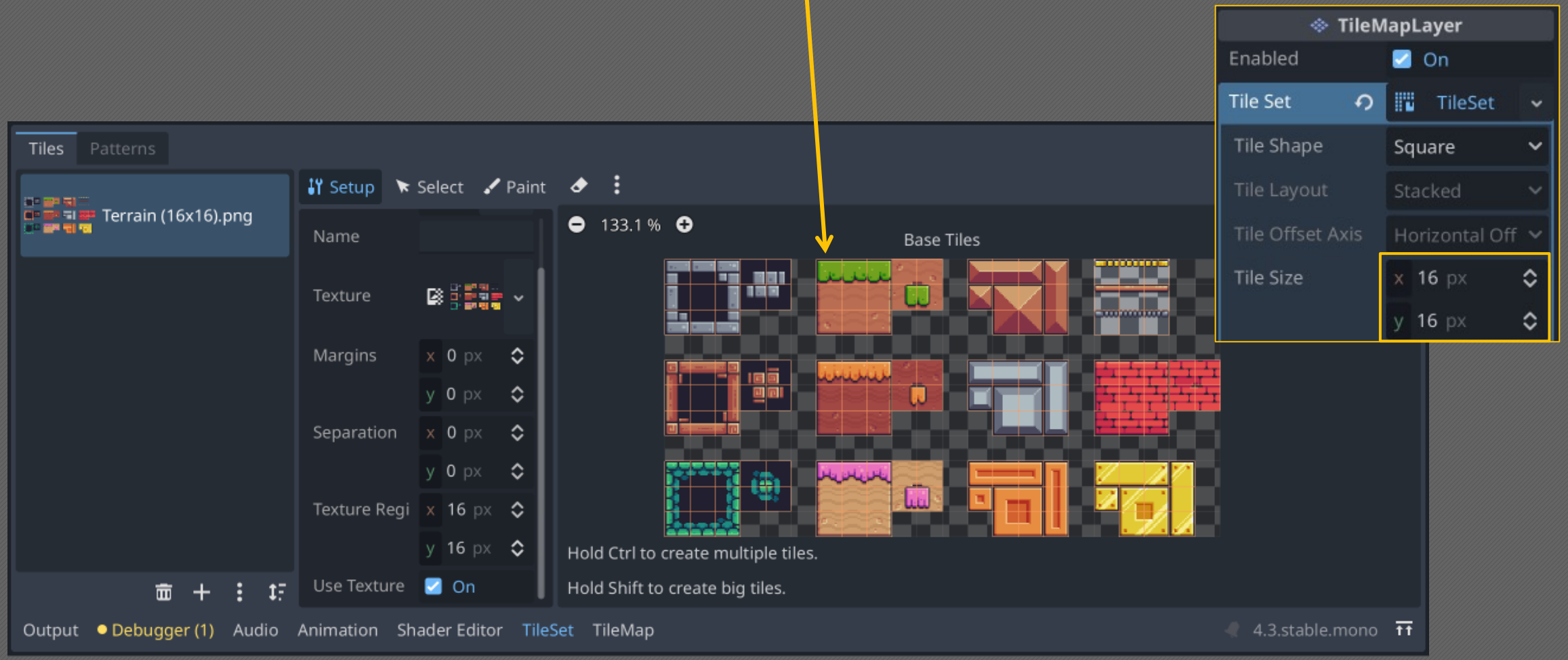
- Navigate to that file in the FileSystem tab and drag-and-drop that file into the blank Tiles space.



Godot will prompt about auto creating tiles; click Yes to have Godot detect and identify tiles for us.



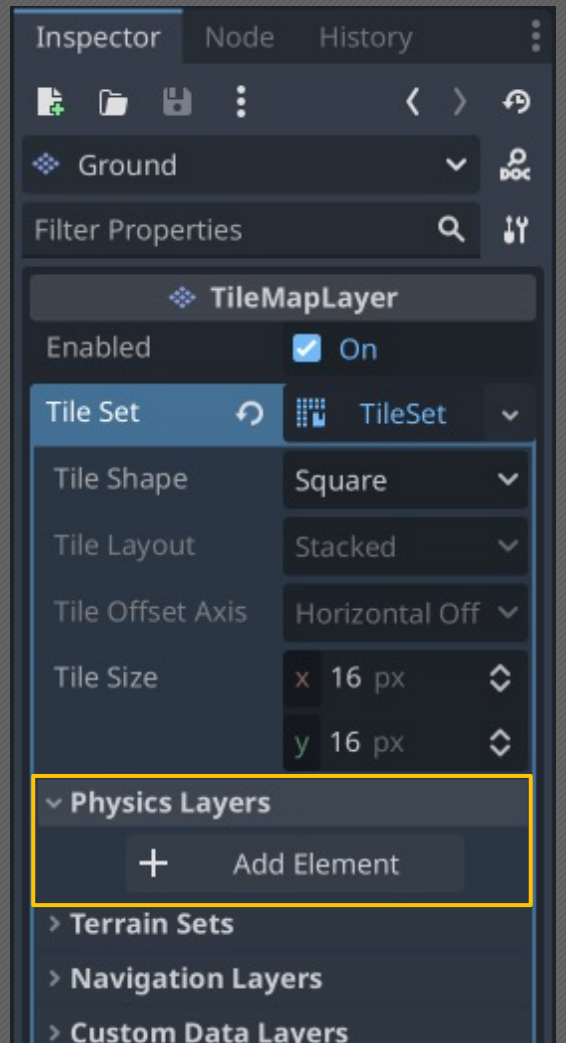
This sprite sheet has a transparency layer with it, just like the character sprites. Godot automatically detects which tiles to use based on the TileMapLayer's 16x16 tile size property.



We can draw the tiles onto the TileMapLayer now, but the tiles have no collisions to stop movement.

Adding Physics/Collisions to the TileMapLayer

- Collisions are handled by Godot's physics engine, via various types of the CollisionShape2D base type.
- To add collisions to the tiles in a Tile Set, the encapsulating TileMapLayer needs to have Physics Layers defined on it.
- Click the Add Element button under the Physics Layers section.



Adding Physics/Collisions to the TileMapLayer

- Clicking the Add Element button in the Physics Layer section adds a grouped data set containing a Collision Layer, a Collision Mask, and a Physics Material.

- A Collision Layer describes the layer that the object appears in. This is a bitmask (on or off flags) for 32 slots.

- Any combination of the layers may be on or off (e.g. all, none, or some).

Adding Physics/Collisions to the TileMapLayer

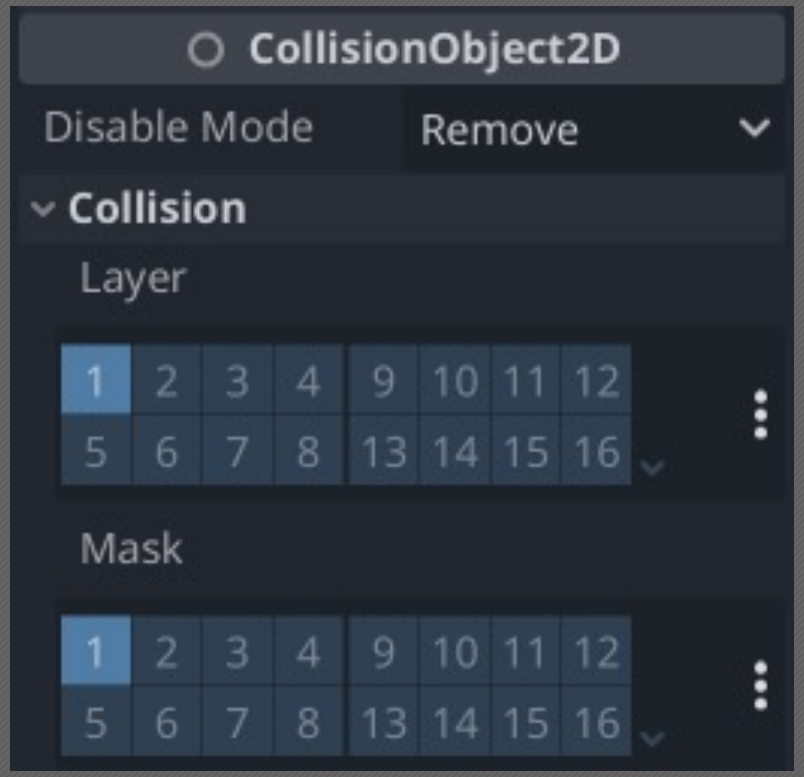
- A Collision Mask describes what layers that the object/body will scan for collisions. This is also a bitmask (on or off flags) for 32 slots.

- If a scanned object is not in one of the collision mask layers, the scanning body will ignore that object's collision.

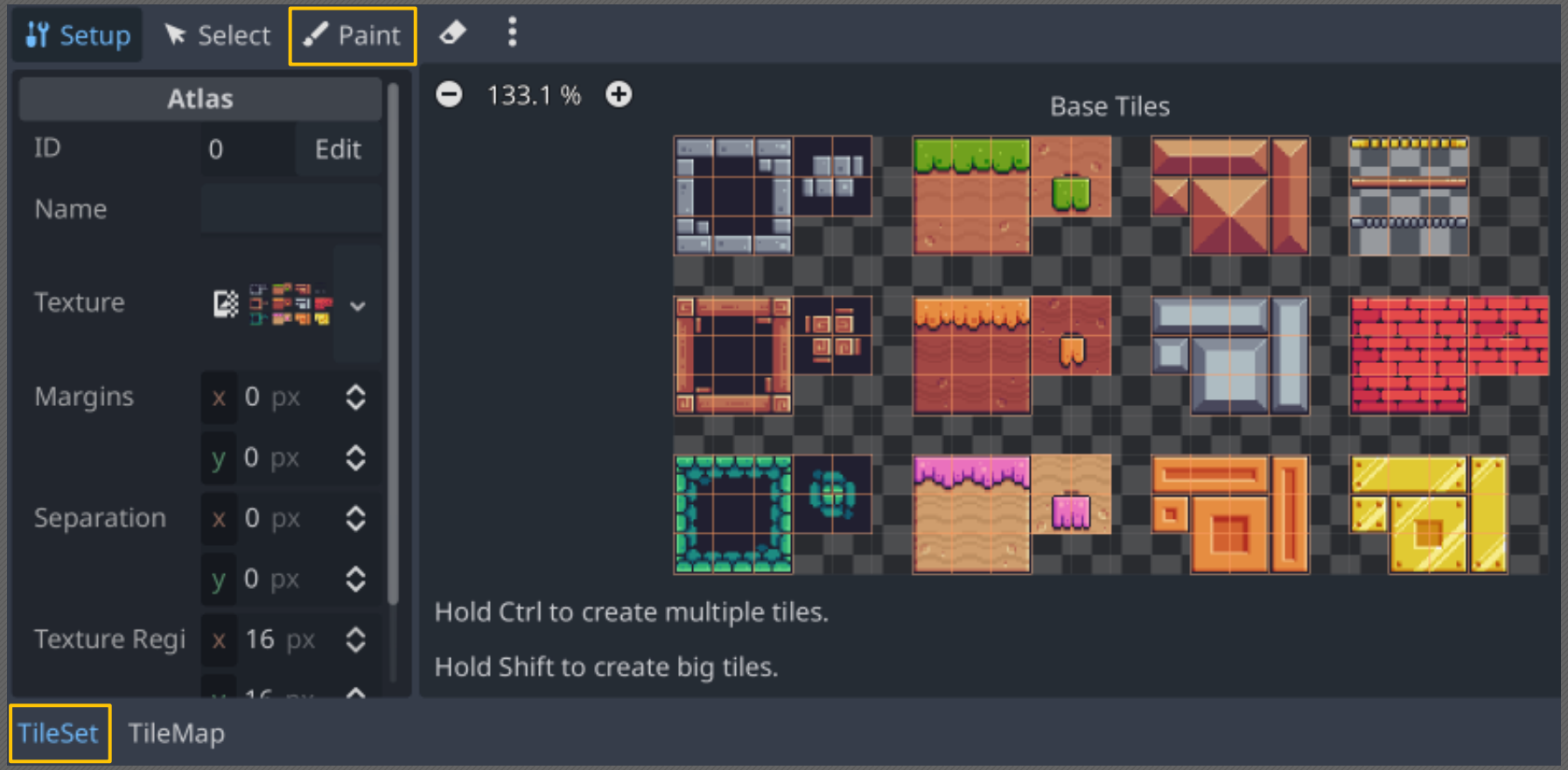
Adding Physics/Collisions to the TileMapLayer

- CollisionShape2D instances, by default, are assigned to Layer 1 on the Collision Layer and Collision Mask.

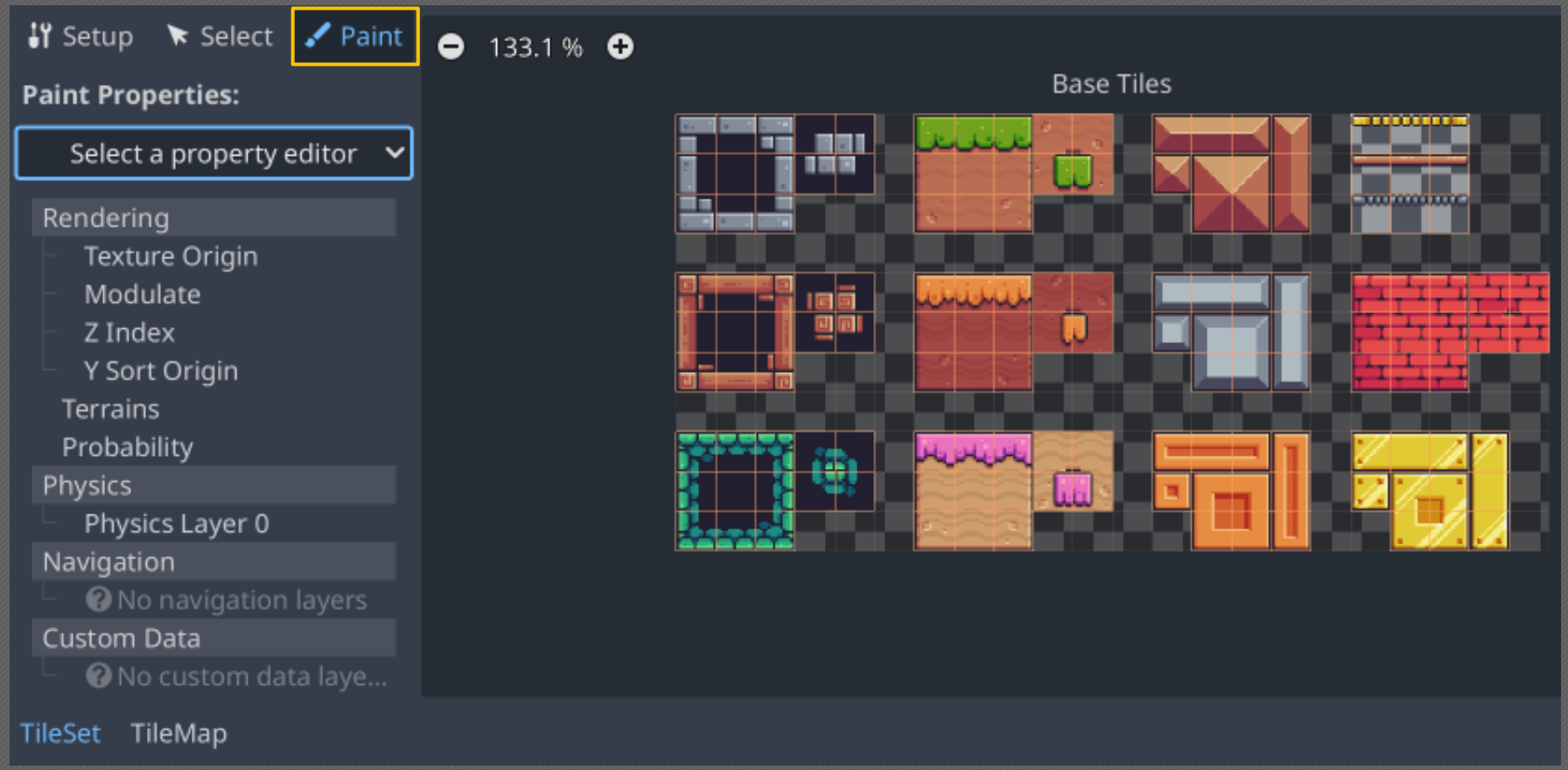
- We did not change the collision object details on the Player scene's root CharacterBody2D, so it is already configured by default to be On/Enabled for Layer 1 and Mask 1 (see right).



Adding Physics/Collisions to the TileMapLayer



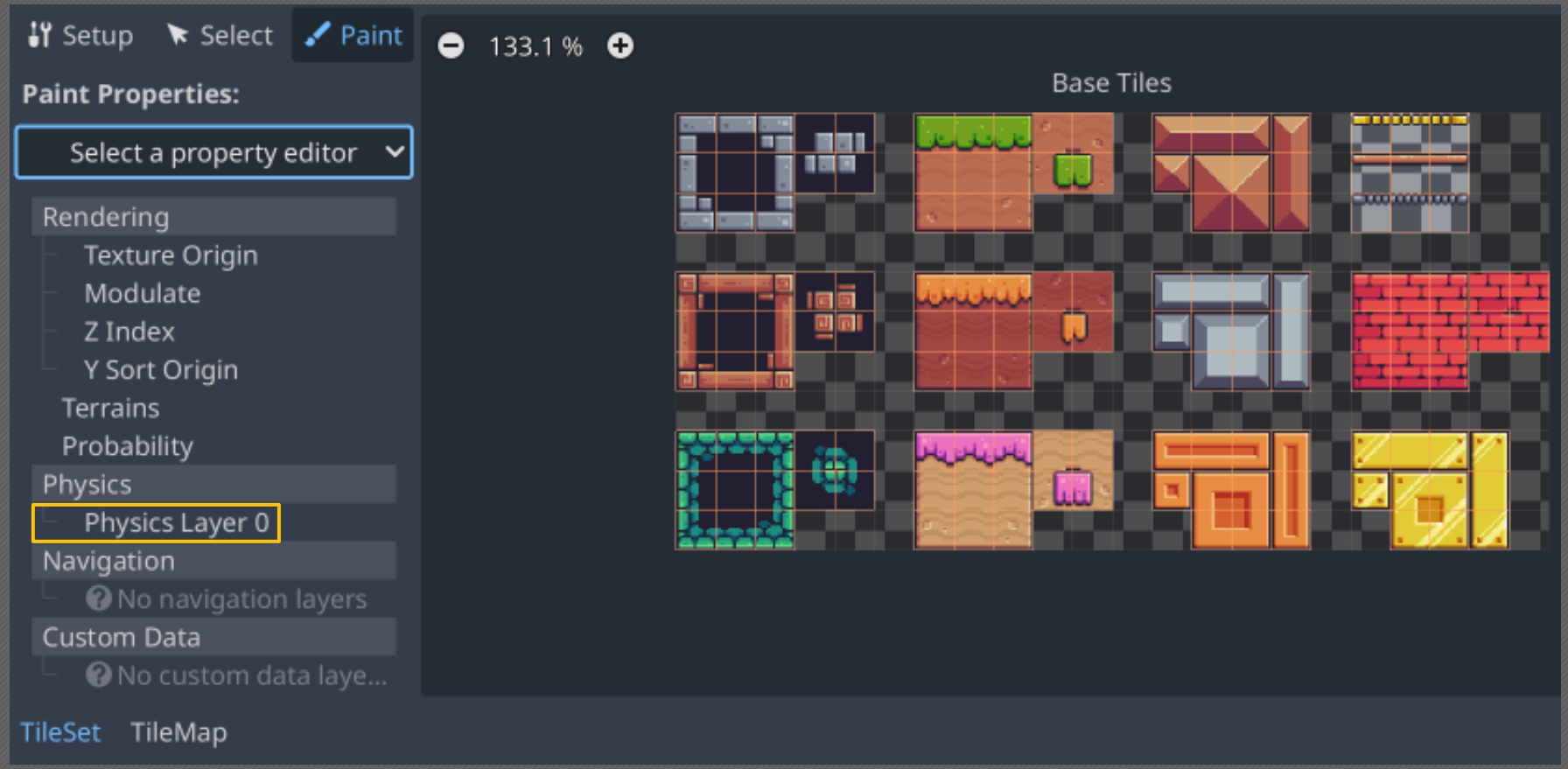
Adding Physics/Collisions to the TileMapLayer



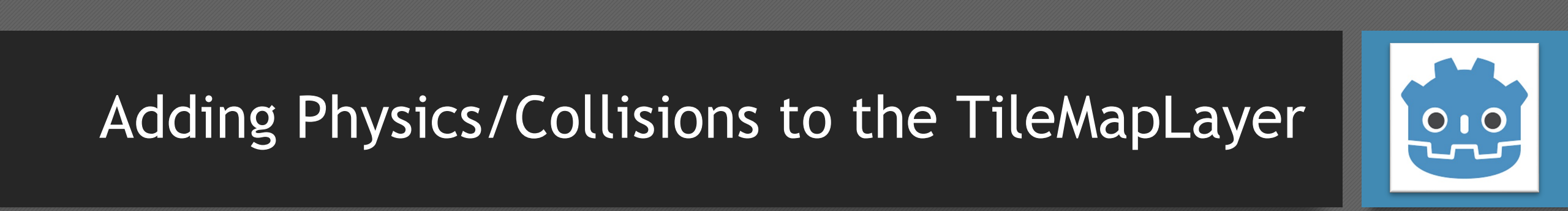
This Paint tab allows us to configure details about each individual Tile in our TileSet. From here, select “Physics Layer 0”.

Adding Physics/Collisions to the TileMapLayer

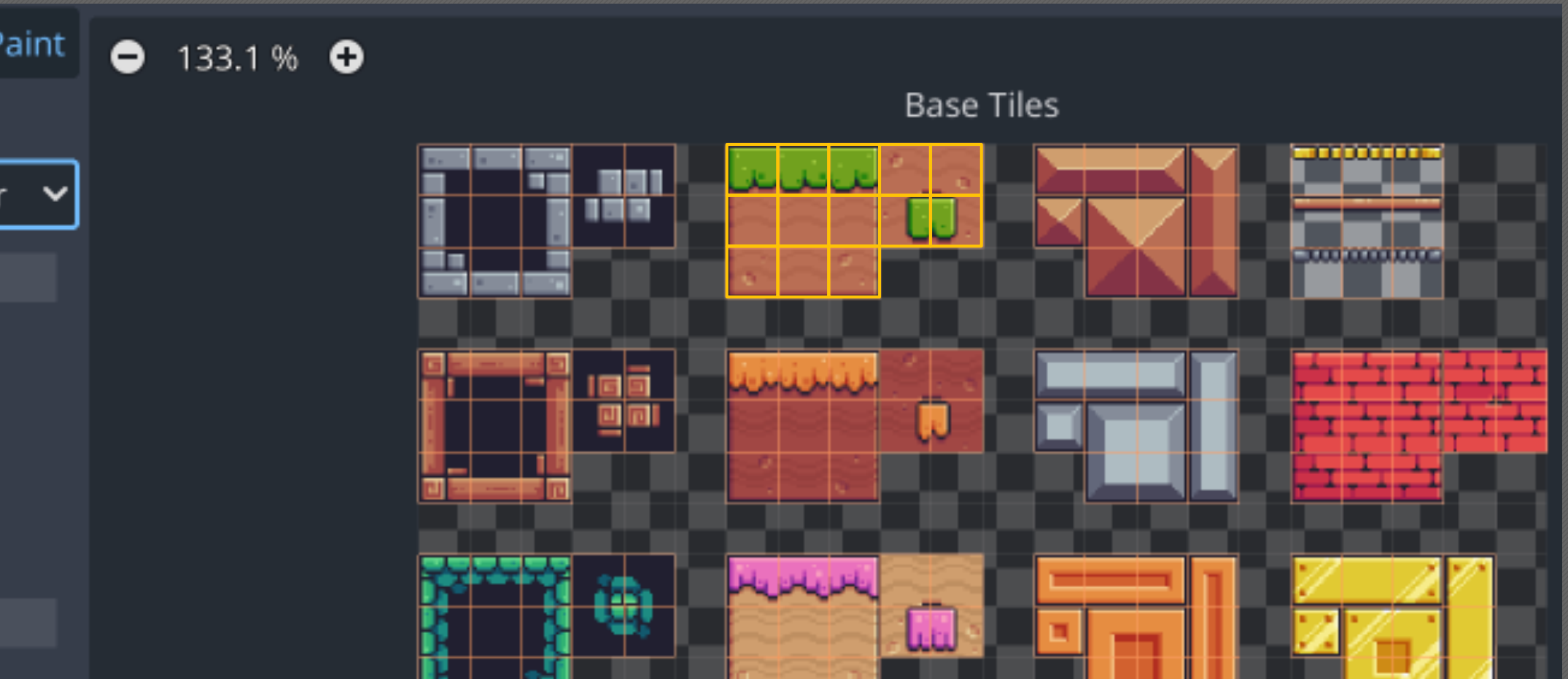
From here, select “Physics Layer 0”. This physics layer maps back to the physics layer recently added to the TileMapLayer.



Adding Physics/Collisions to the TileMapLayer

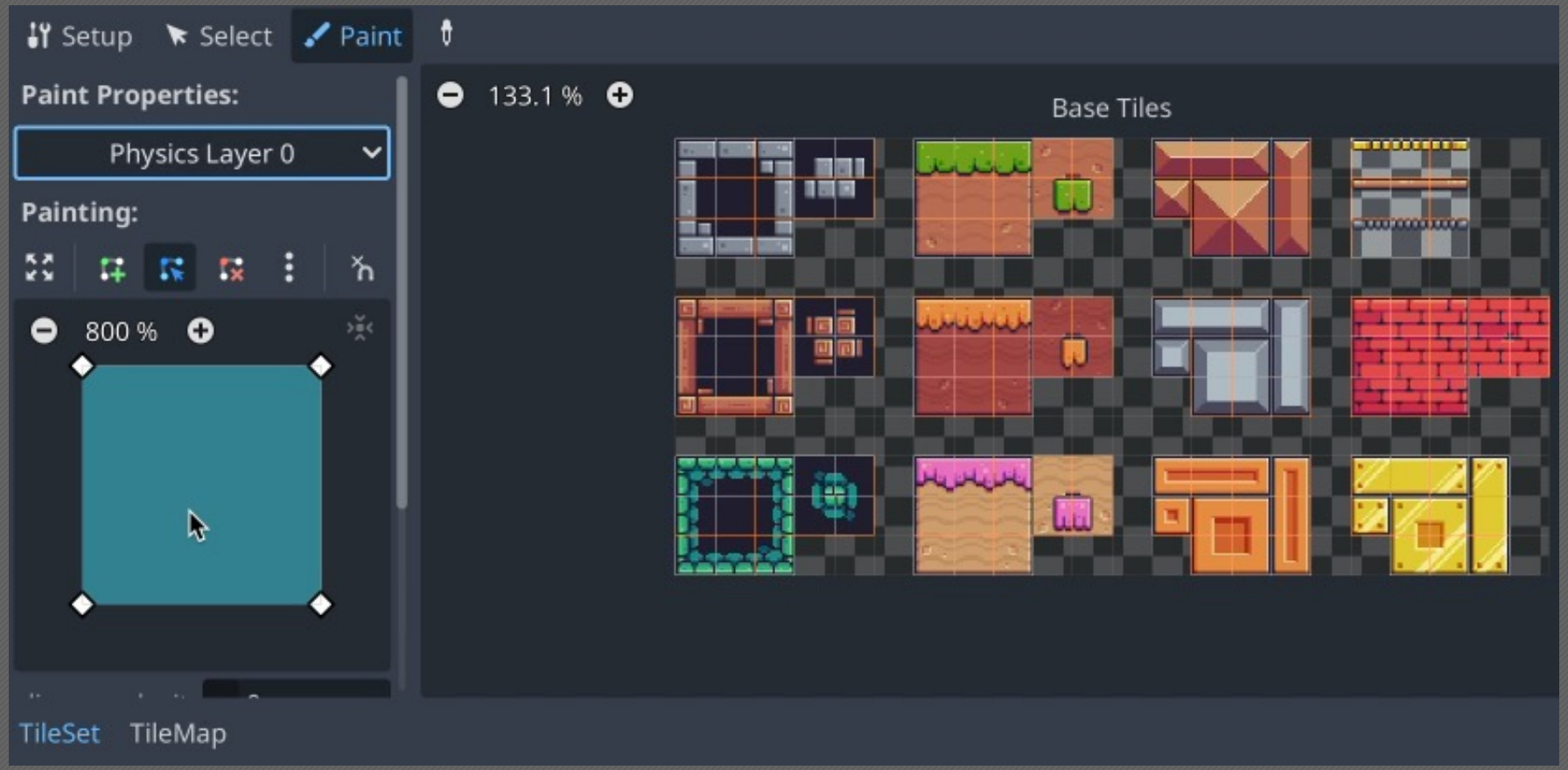


Each tile is independent of and unrelated to its neighbor tiles in the TileSet.
Each individual tile must be defined with its own physics collision shape.



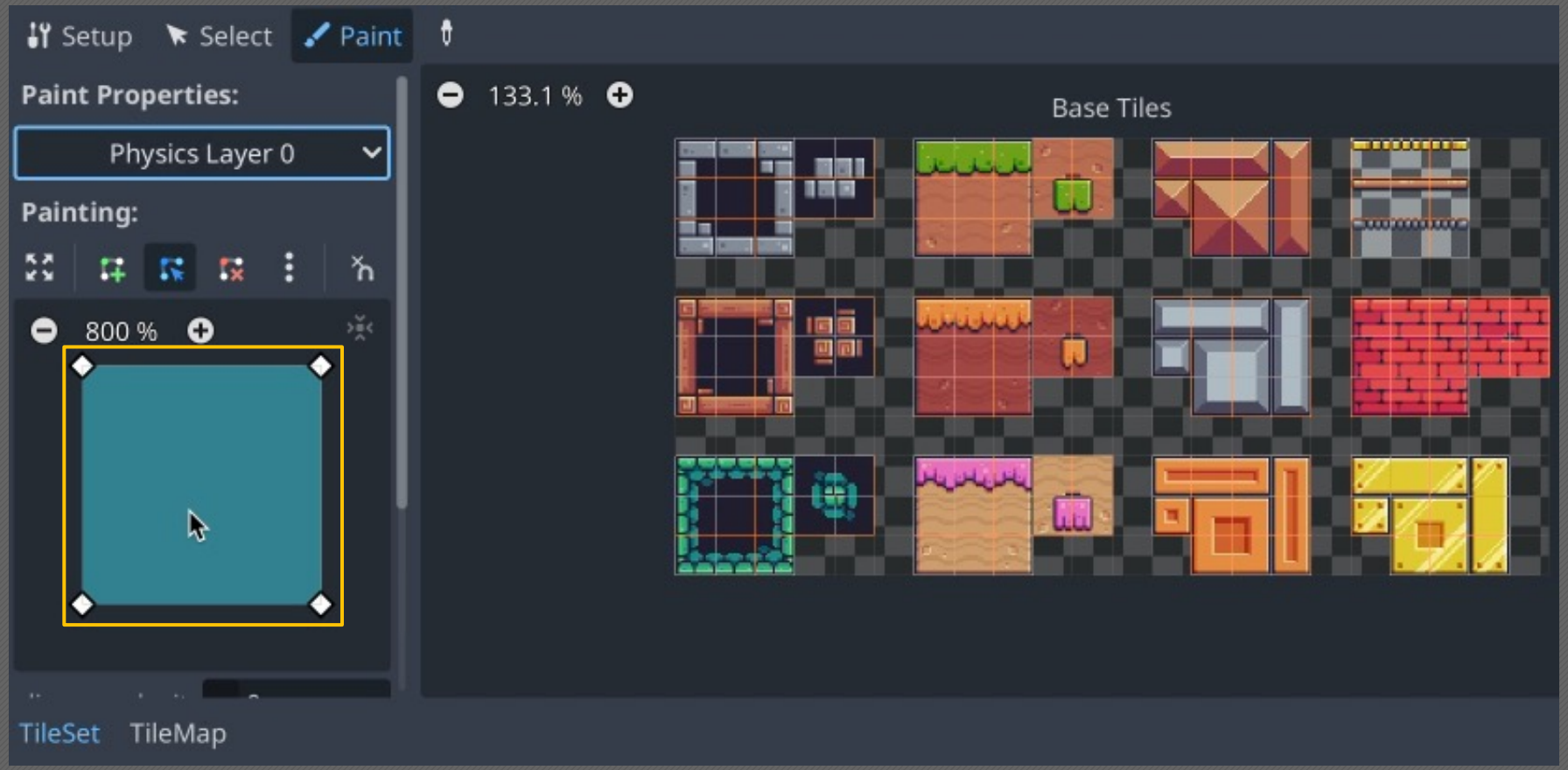
Adding Physics/Collisions to the TileMapLayer

When ”Physics Layer 0” is selected, the TileSet view will change to represent the settings of that Physics Layer.



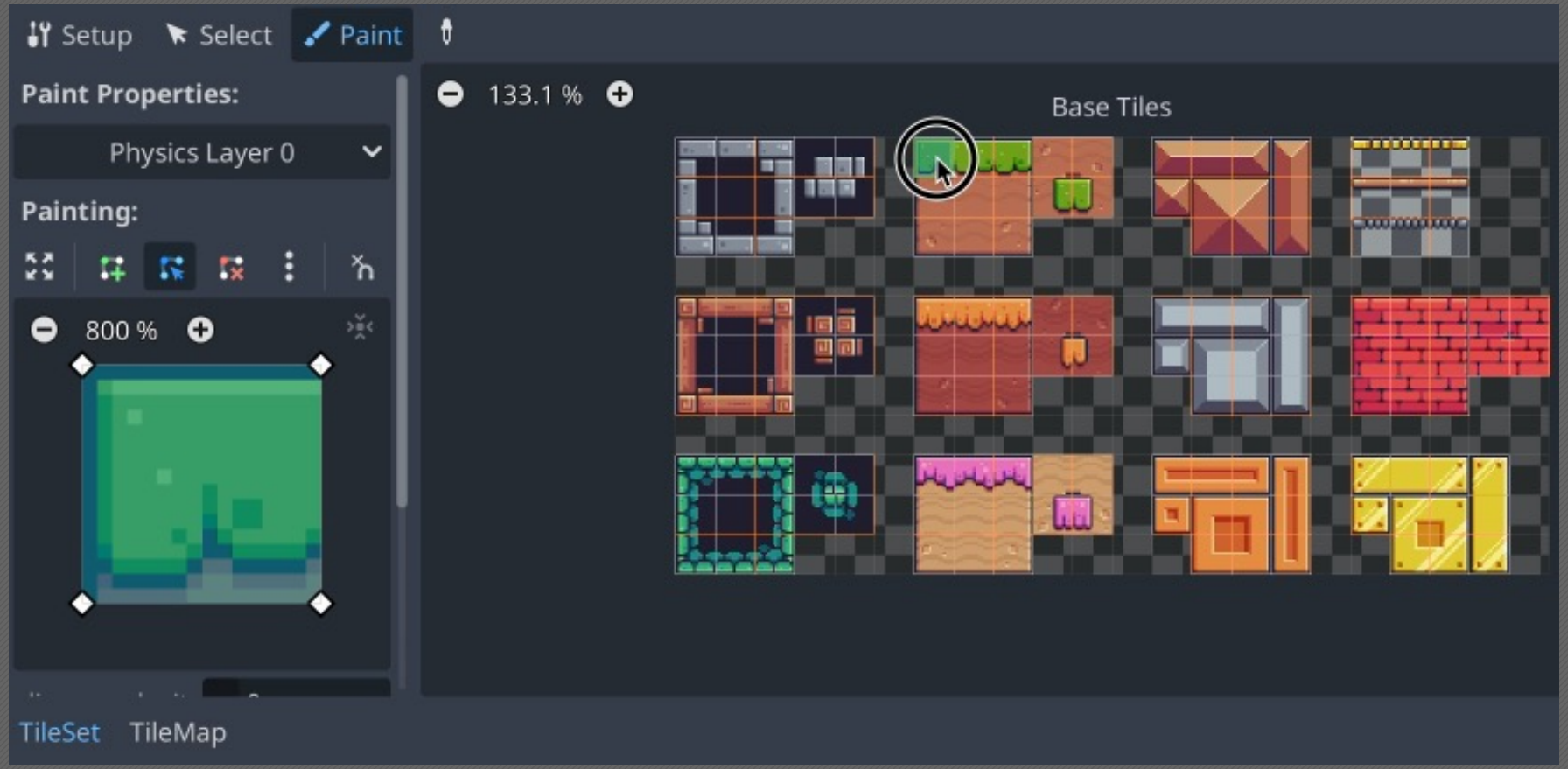
Adding Physics/Collisions to the TileMapLayer

This is the default collision shape that will be used when assigning a shape to tiles in the TileSet. This default shape is appropriate for this use case.



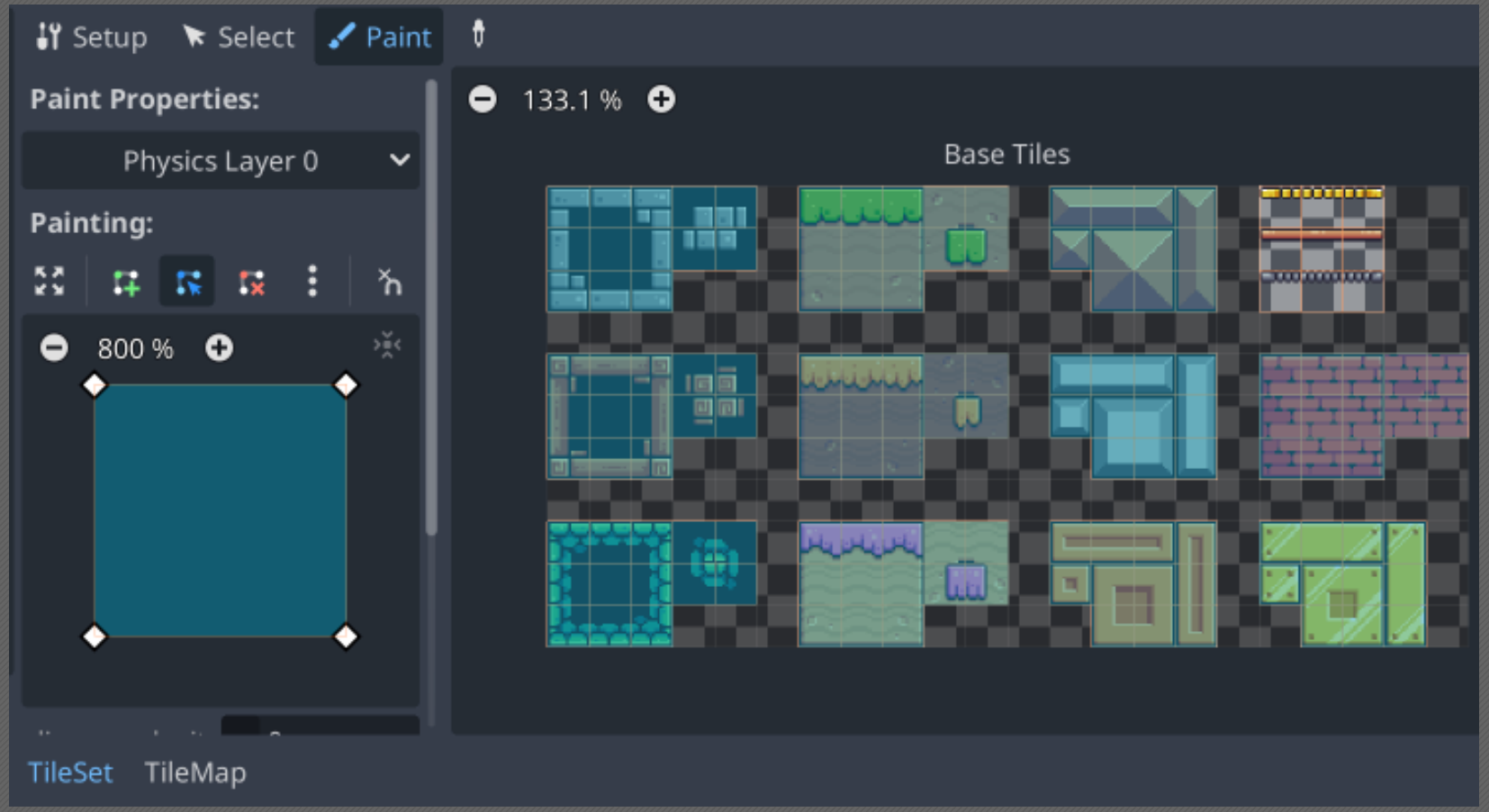
Adding Physics/Collisions to the TileMapLayer

Left click (and optionally drag) individual tiles in the TileSet to apply that collision shape to the selected tile.

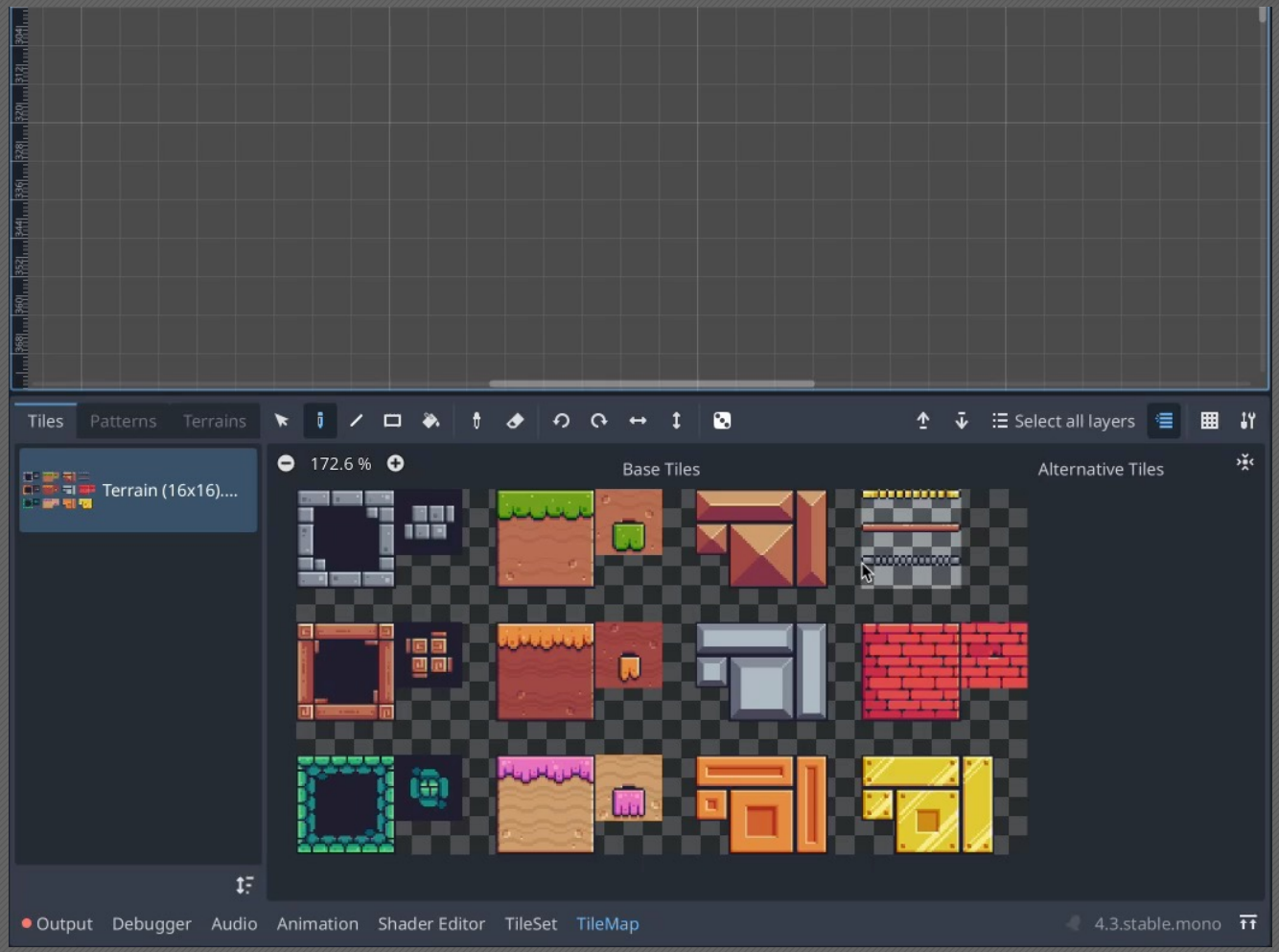


Adding Physics/Collisions to the TileMapLayer

Add collision for all tiles in this TileSet.

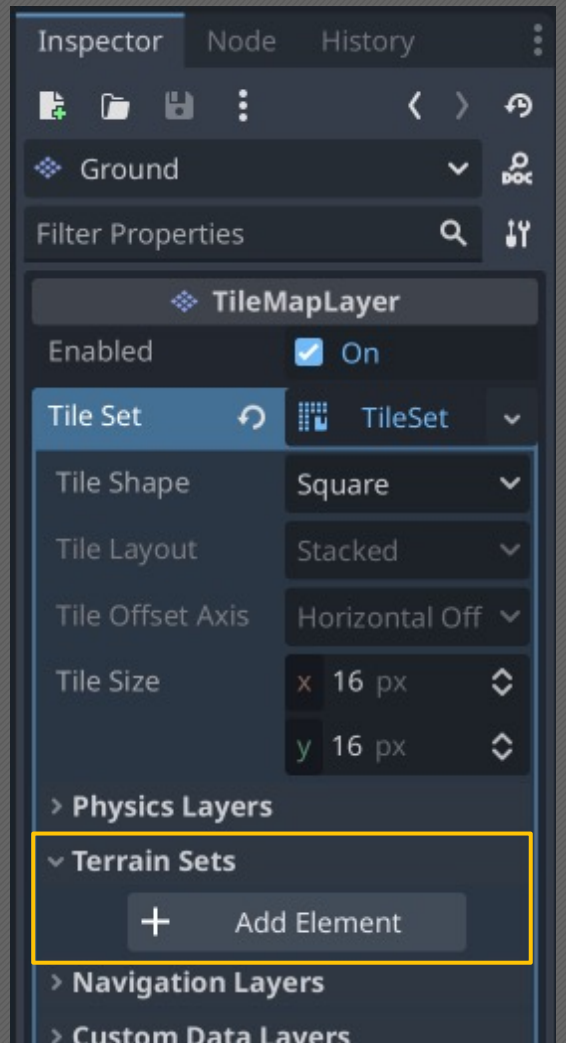


Tiles can be drawn now, but they can only be drawn individually so it’s a slow process.



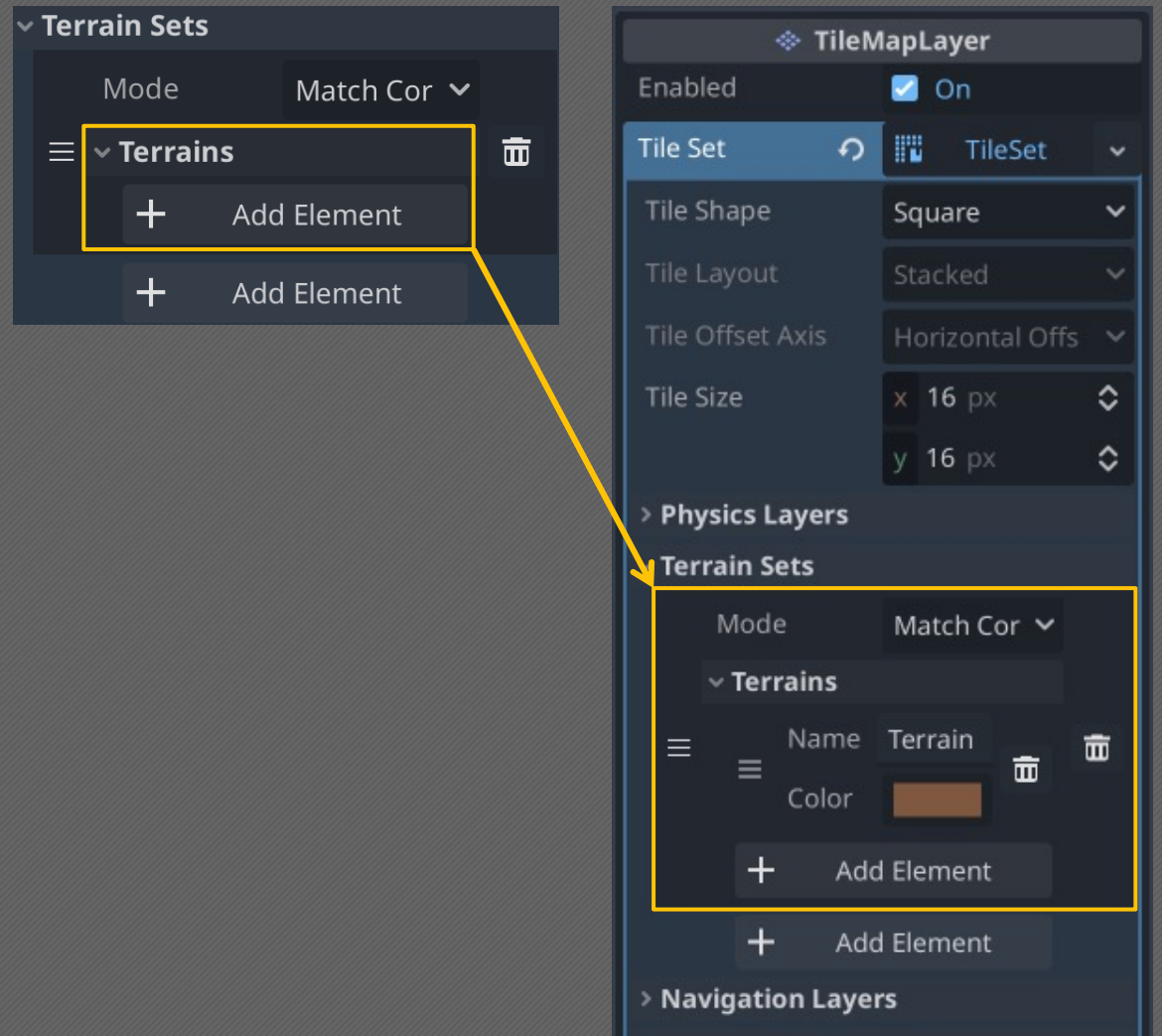
Creating a Terrain Set

- A Terrain Set is way to logically group similar tiles to make it easier to draw those tiles on a TileMapLayer.
- Terrain Sets allow configuring individual tiles to be drawn based on the surrounding tiles at the time the tile is drawn.
- Click this Add Element button.



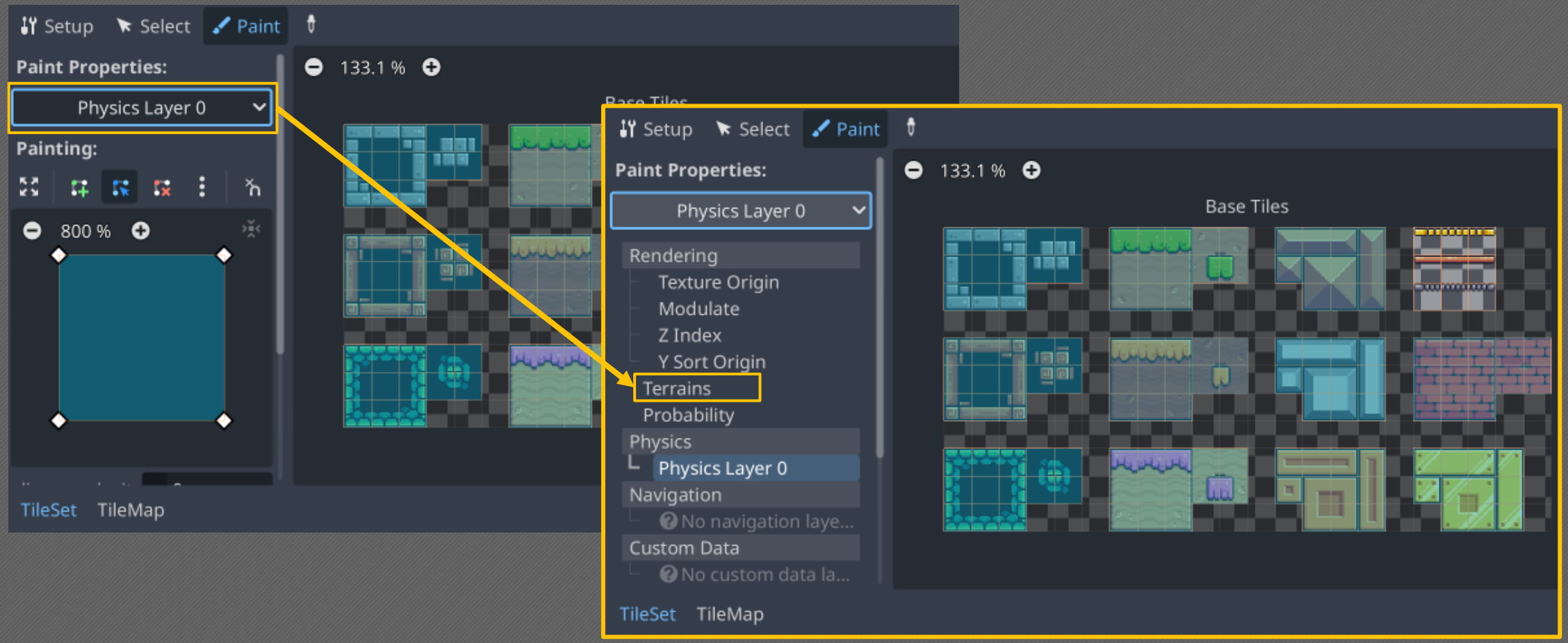
Creating a Terrain Set

- The added section shown with the dark blue background is a singular terrain set. Click Add Element of the recently created Terrain Set.
- The name and color can be changed if desired, but the default values are fine.
- The default mode is “Match Corners and Sides”. This is the correct value.



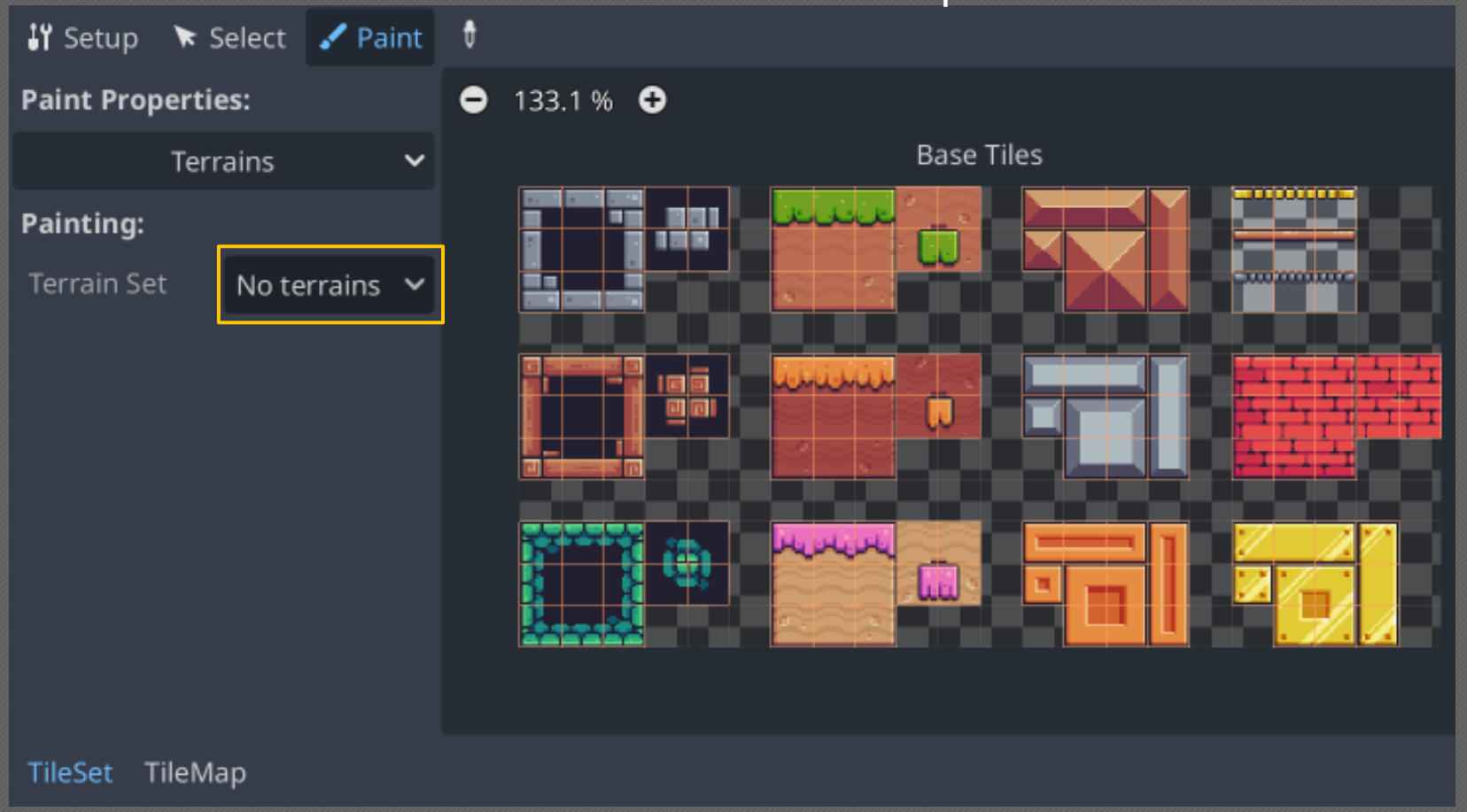
Creating a Terrain Set

After adding the Terrain Set, navigate from the Physics Layer to Terrains in the TileSet.



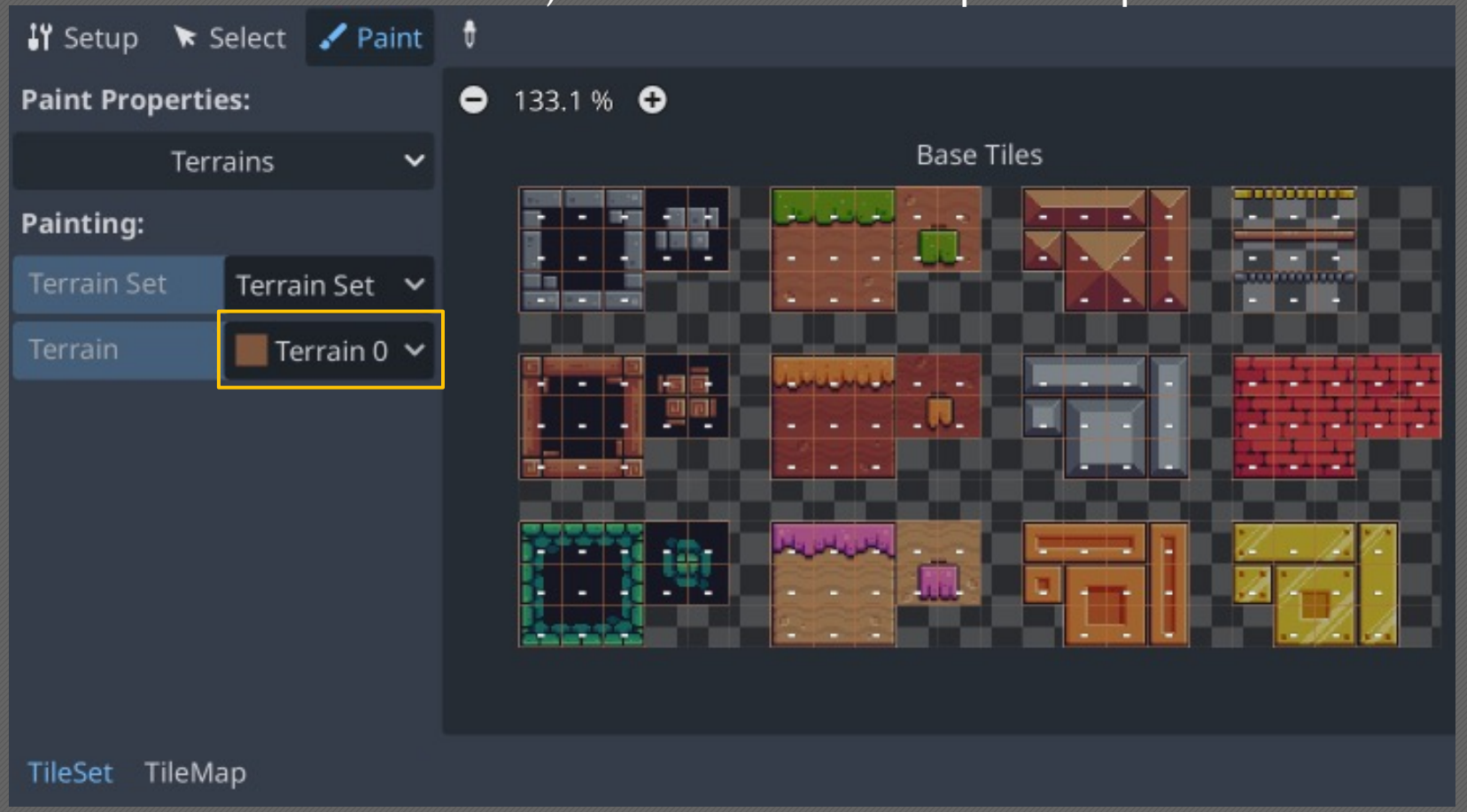
Creating a Terrain Set

Once the Terrains property is selected, expand the Terrain Set dropdown and select the “Terrain Set 0” option.



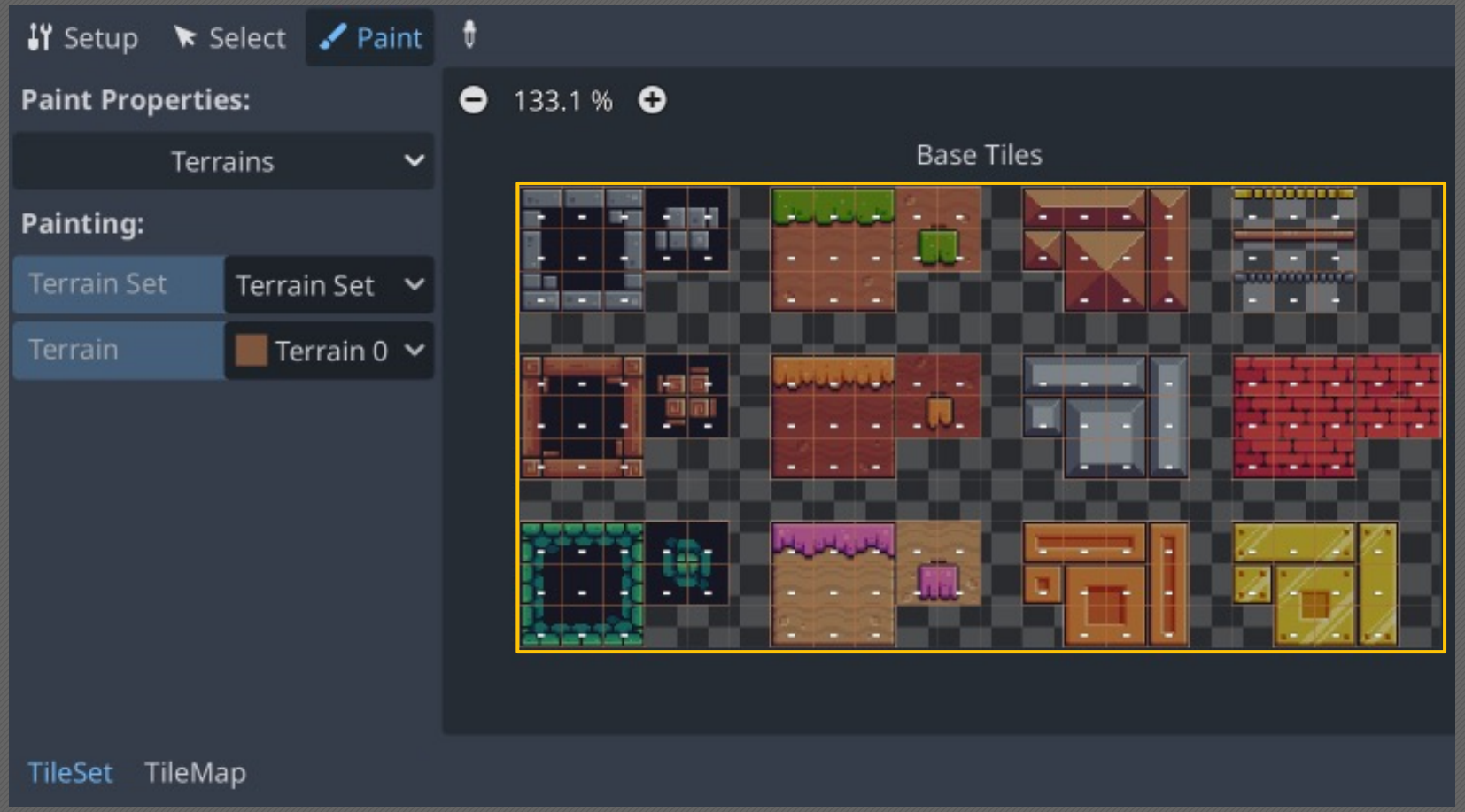
Creating a Terrain Set

Then select the “Terrain Set 0” option (or the name of whatever you changed the terrain set to) from the Terrain dropdown options.



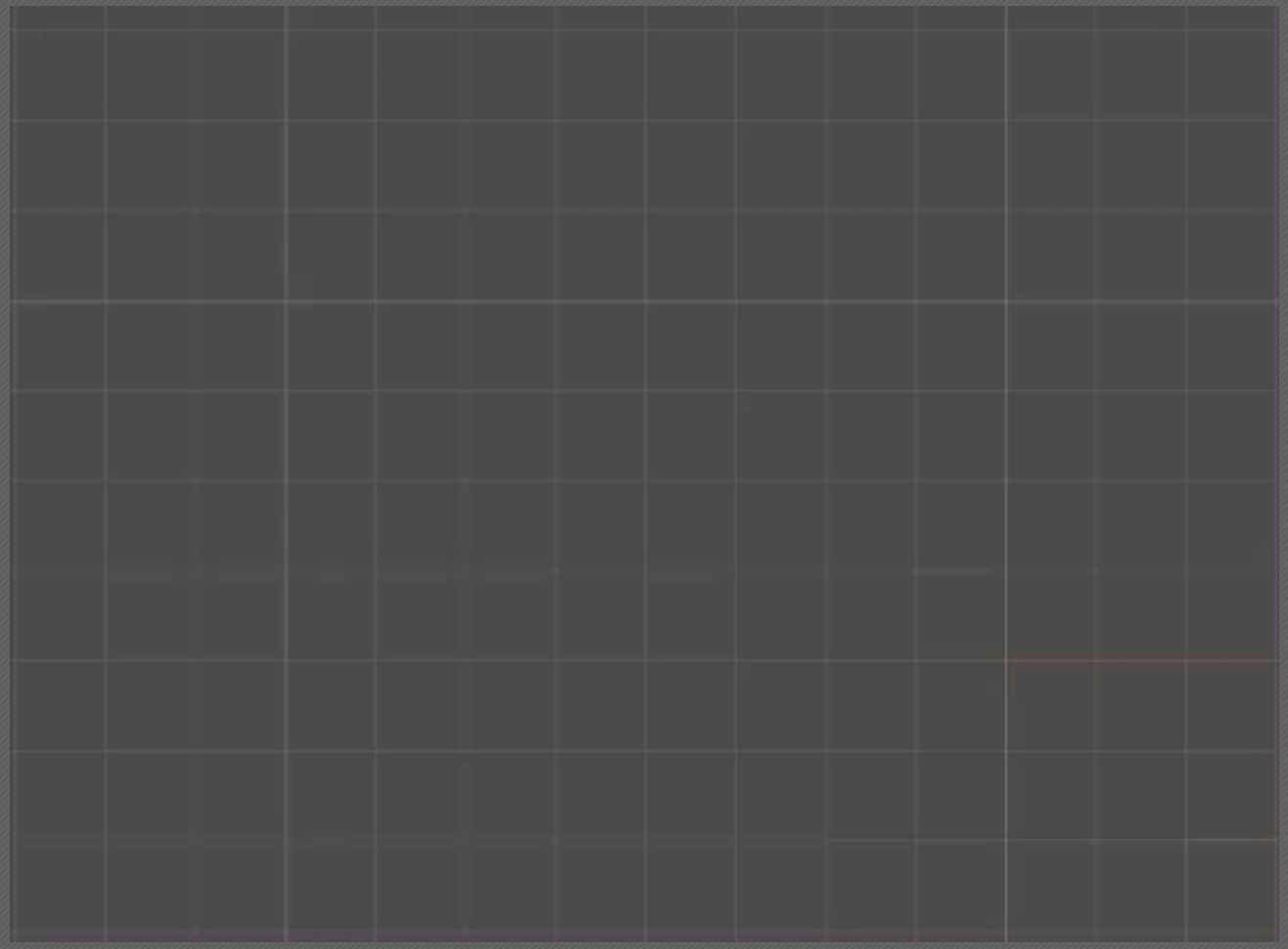
Creating a Terrain Set

By default, all terrain sets are empty of any tiles; tiles must be explicitly added to terrain sets.



Matching Corners and Sides

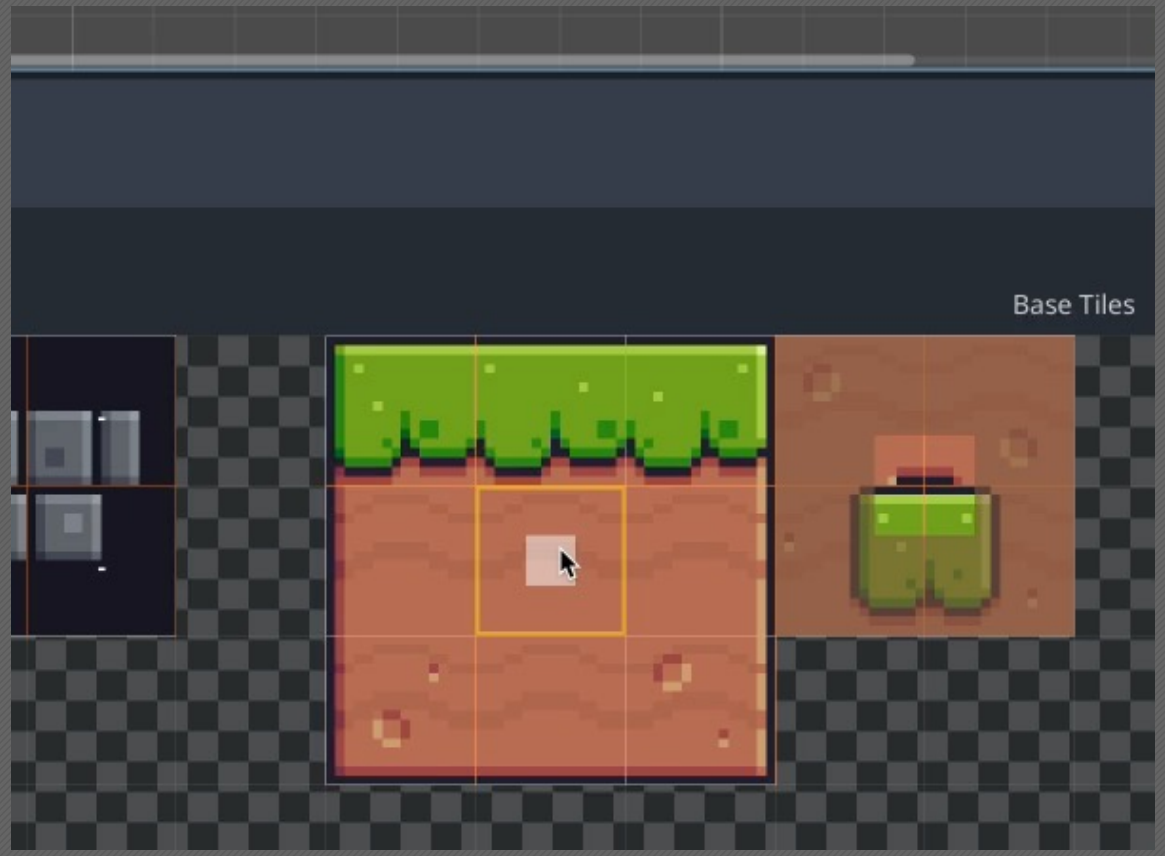
The “Match Corners and Sides” terrain set mode means that when a tile for that terrain set is drawn on the TileMapLayer node, the tile used for that position will be based on the current neighbors (for both sides and corners/diagonals) of that tile.



Matching Corners and Sides

- For terrain tiles to be drawn based on the tile's neighbors, the terrain needs to be configured for which tiles are drawn in which situation.

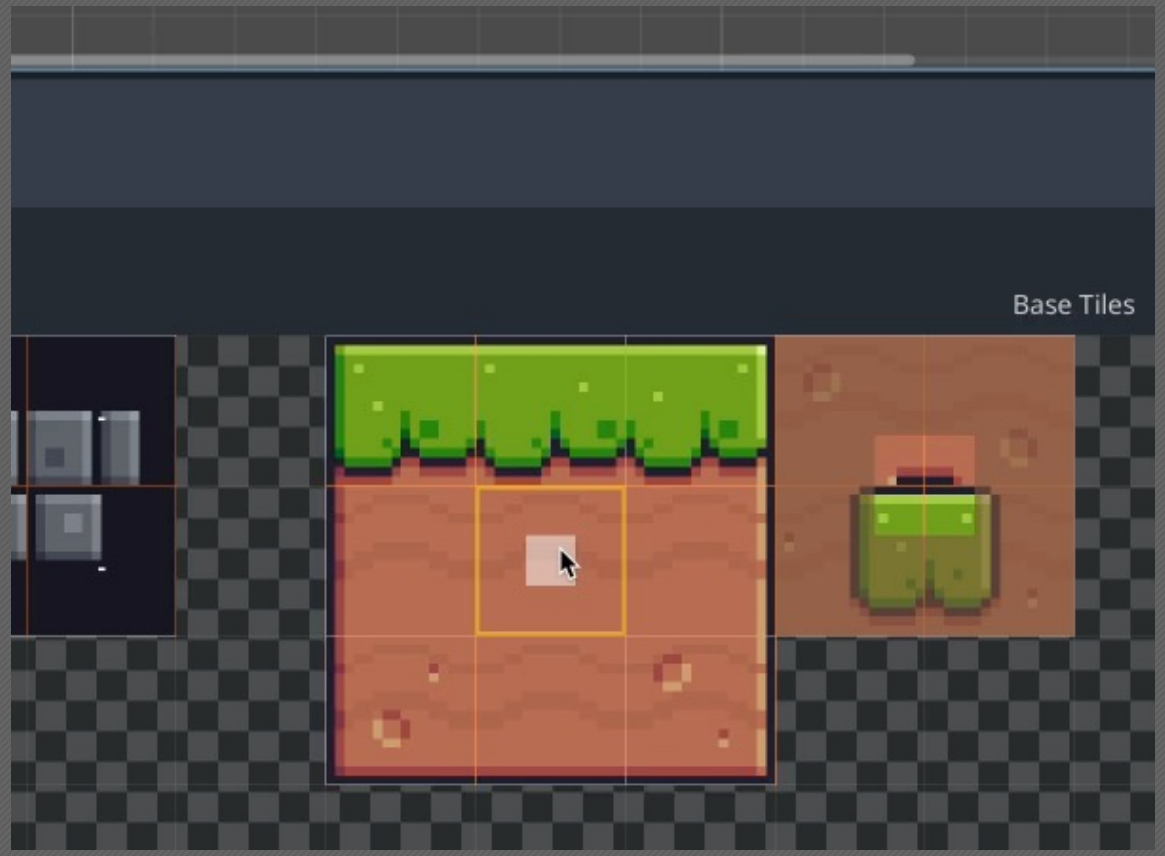
- When a terrain tile is drawn on the TileMapLayer, the tile matching the "best" neighbor configuration is used.



Matching Corners and Sides

- To configure a terrain tile how to draw, parts of the tile can be toggled on/enabled based on the terrain mode.

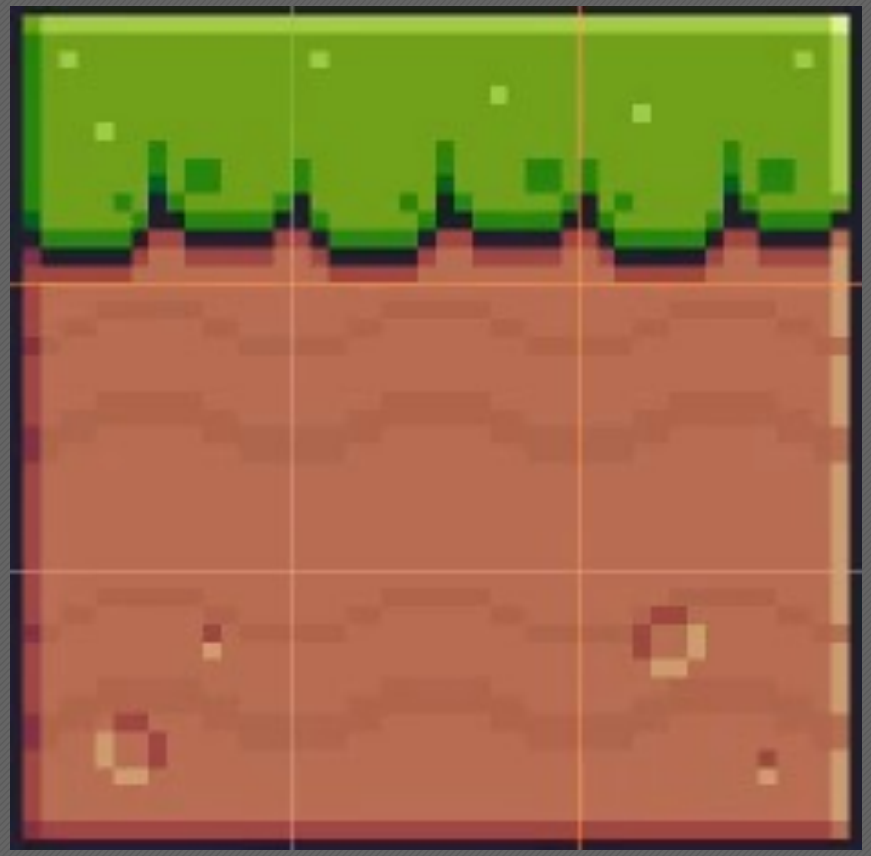
- A tile is drawn when all its corners and sides match the tile's terrain configuration. Behavior is unpredictable when a perfect neighboring match is not configured.



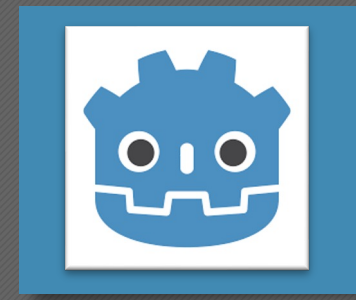
Matching Corners and Sides

- A square tile has 4 corners and 4 sides for a total of 8 possible neighbors.

- The center square of a tile's configuration means "only draw this tile when an actively drawn tile on the TileMapLayer" (as opposed to being drawn as an automatic neighbor for an actively drawn tile).



Matching Corners and Sides



- In the example to the right, the center tile will be drawn when a tile is drawn above itself.

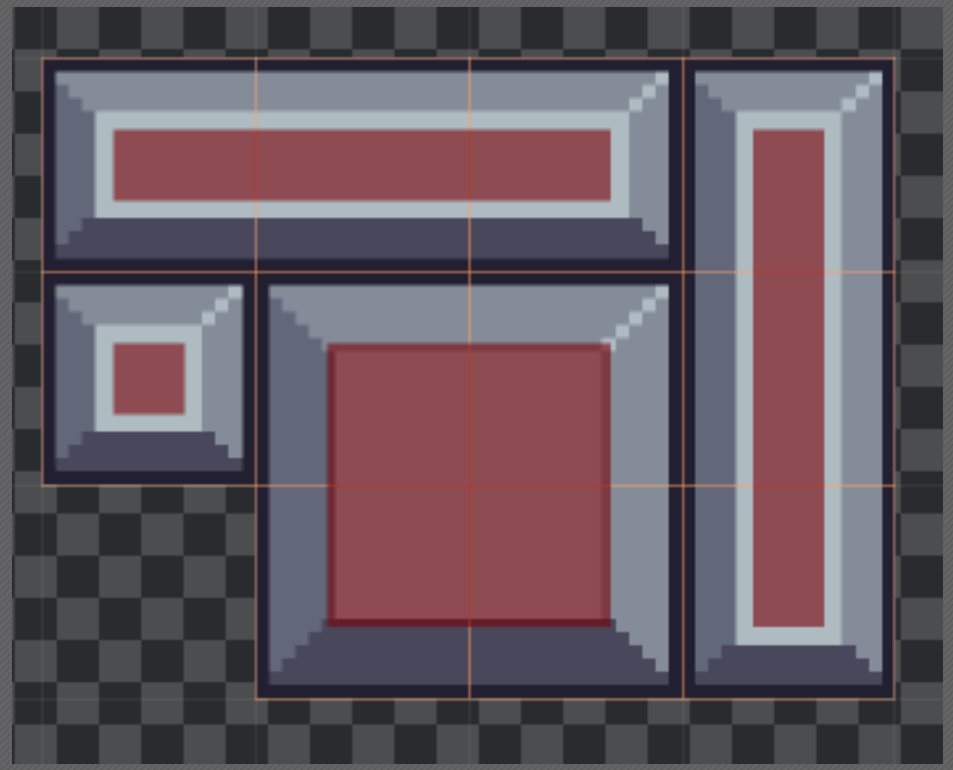
- The top tile will be drawn when a tile is drawn underneath itself.

Matching Corners and Sides

Set up the following terrain matching configuration for only ONE type of tyle (like seen below) on the tile sheet.

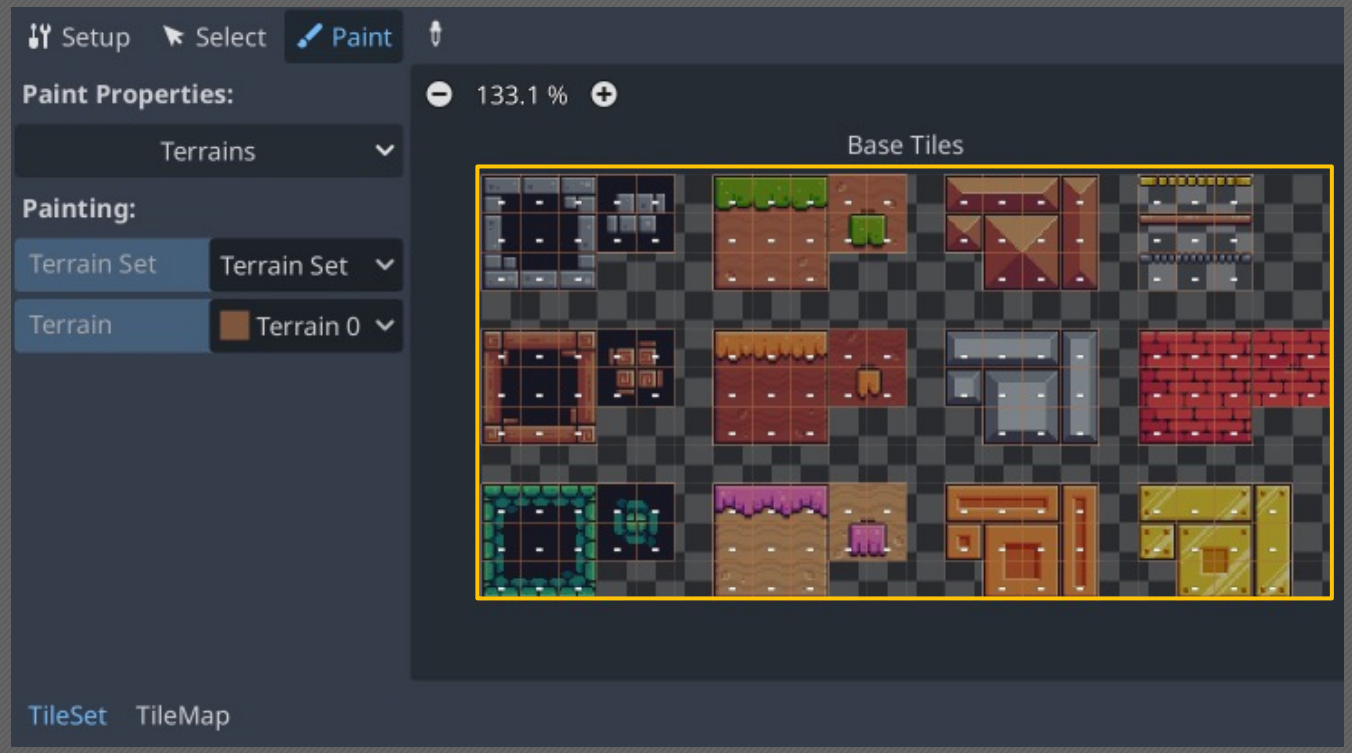
Matching Corners and Sides

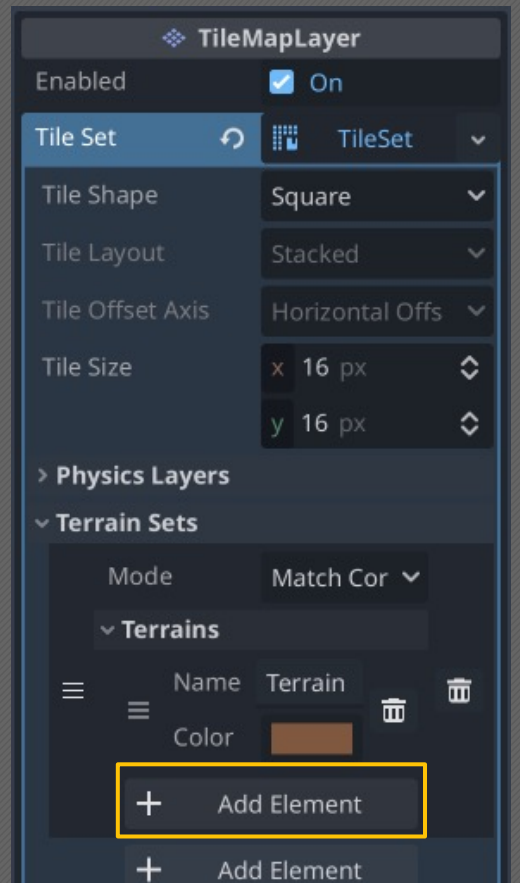
Alternatively, tiles that have a ”1x1, 1x3, 2x2, 3x1” art pattern as seen below should have the following match configuration.



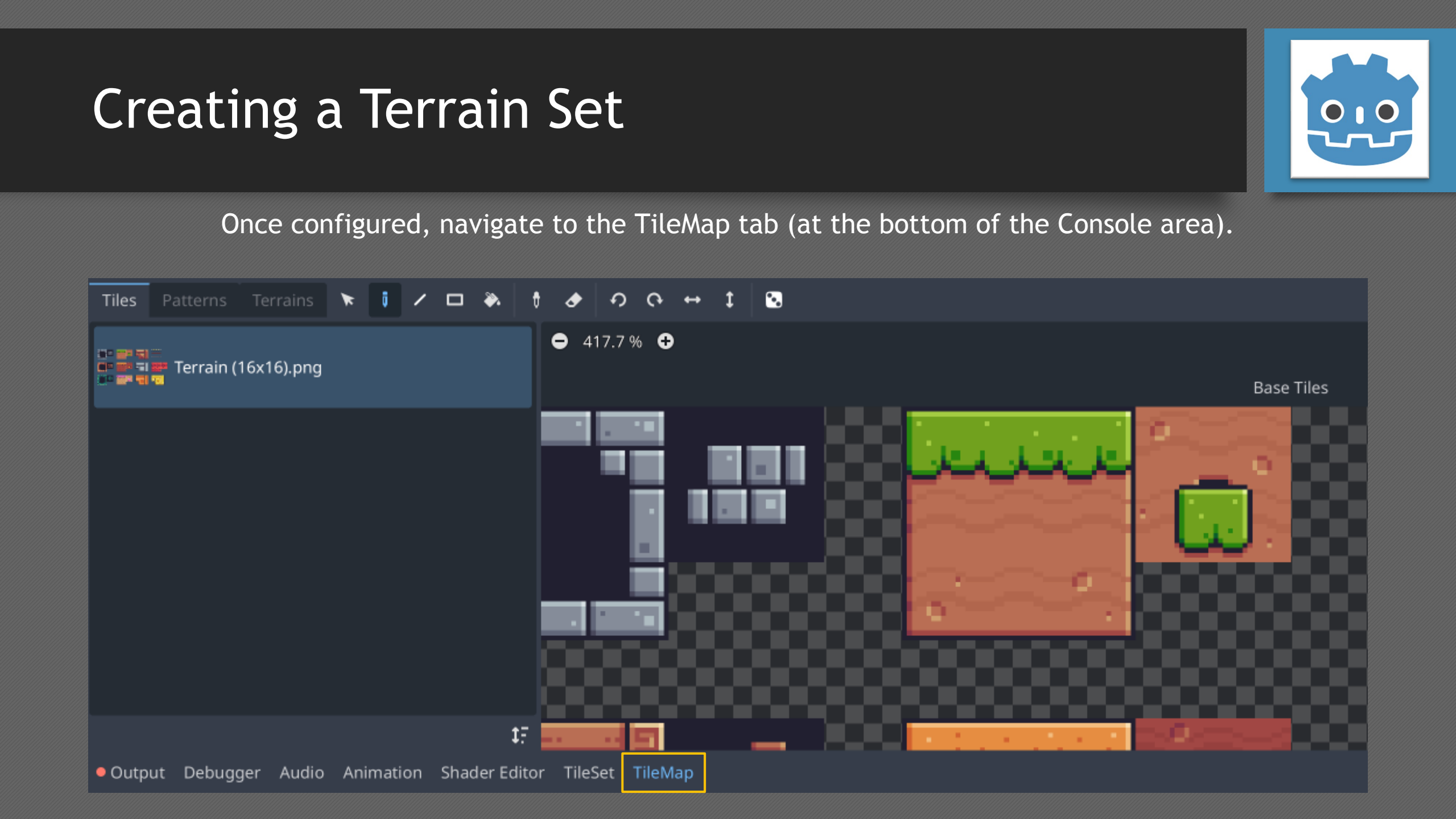
Create New Terrain Sets Per Style

When intending to use various tile styles, we can't "reuse" the terrain since the corner and side matches are the same. Create different terrains with the Add Element button inside of the darker blue area. Be sure to give a proper name to each terrain in this case, so that you can tell them apart when editing the TileSet terrain layers.



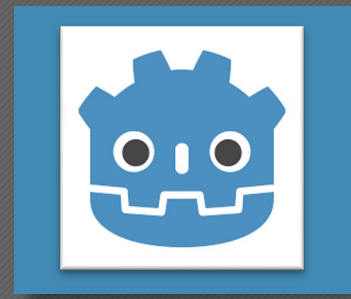


Creating a Terrain Set

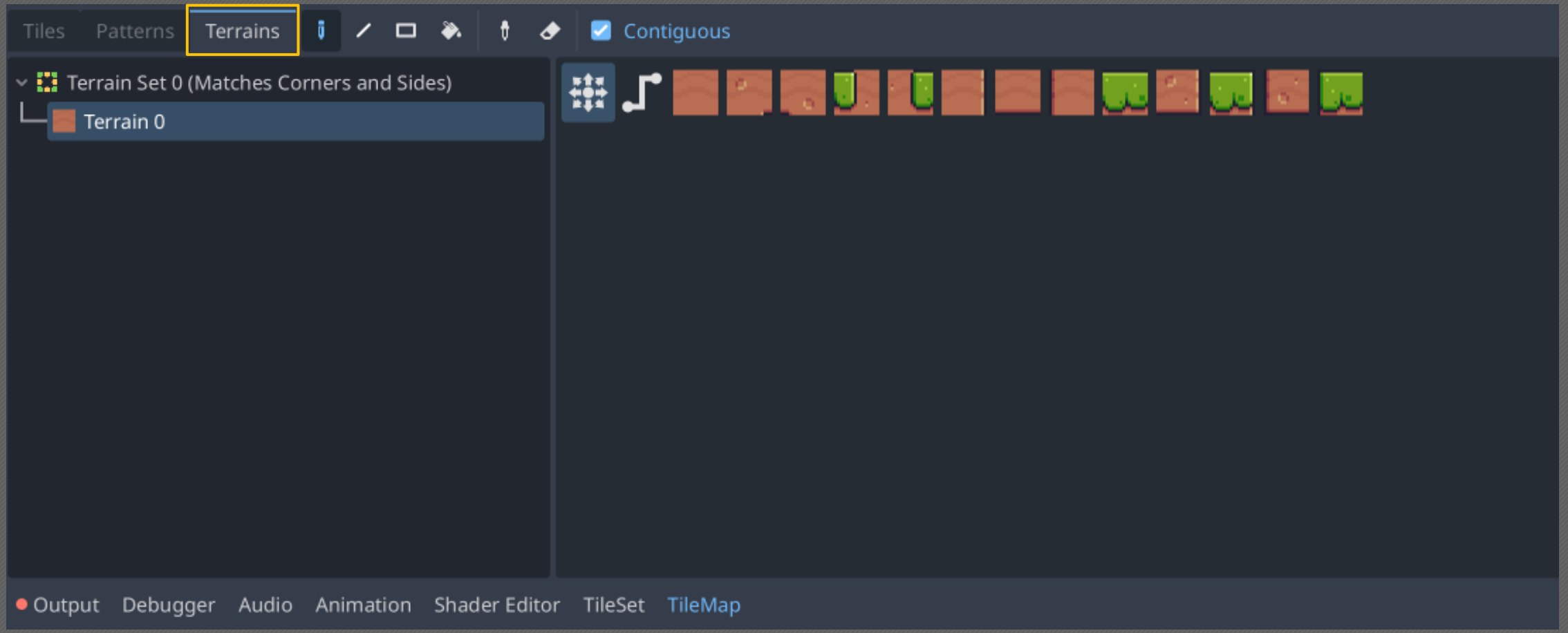


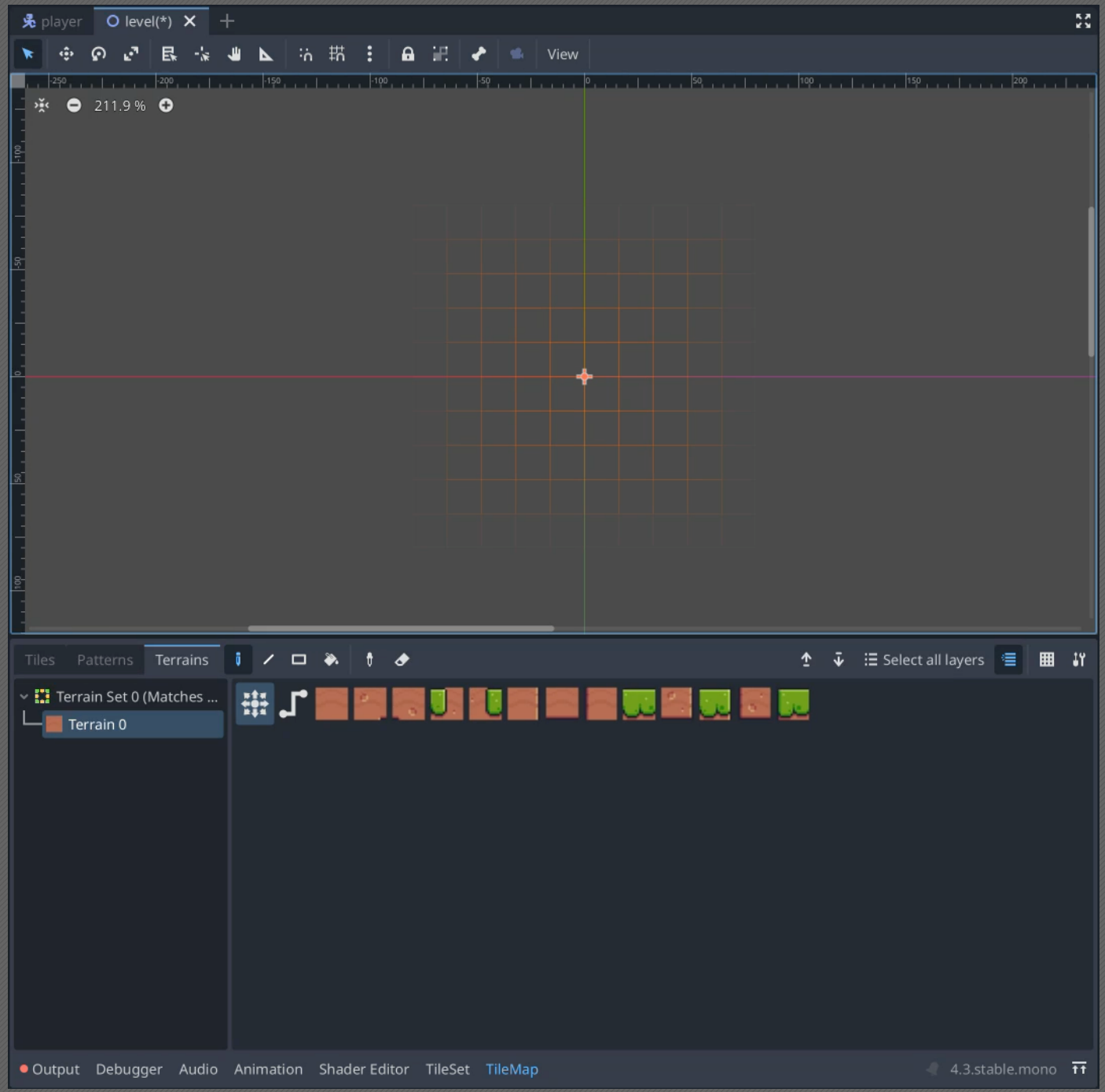
Once configured, navigate to the TileMap tab (at the bottom of the Console area).

Creating a Terrain Set

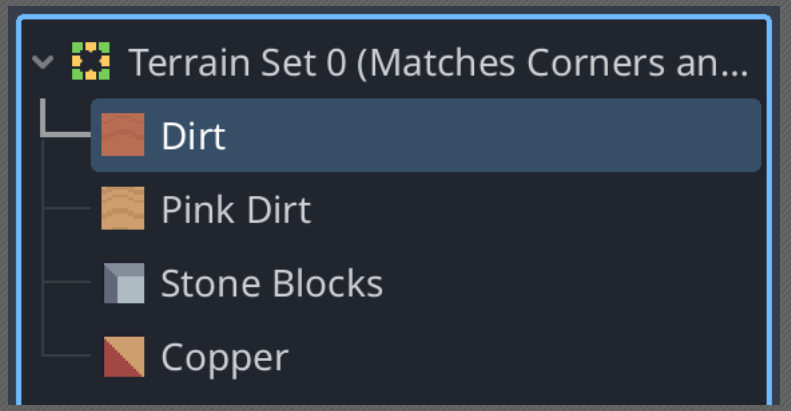


From here, you can select the Terrain 0 terrain and draw it into your TileMapLayer node! Notice how all configured tiles for that terrain show in the right tile area.



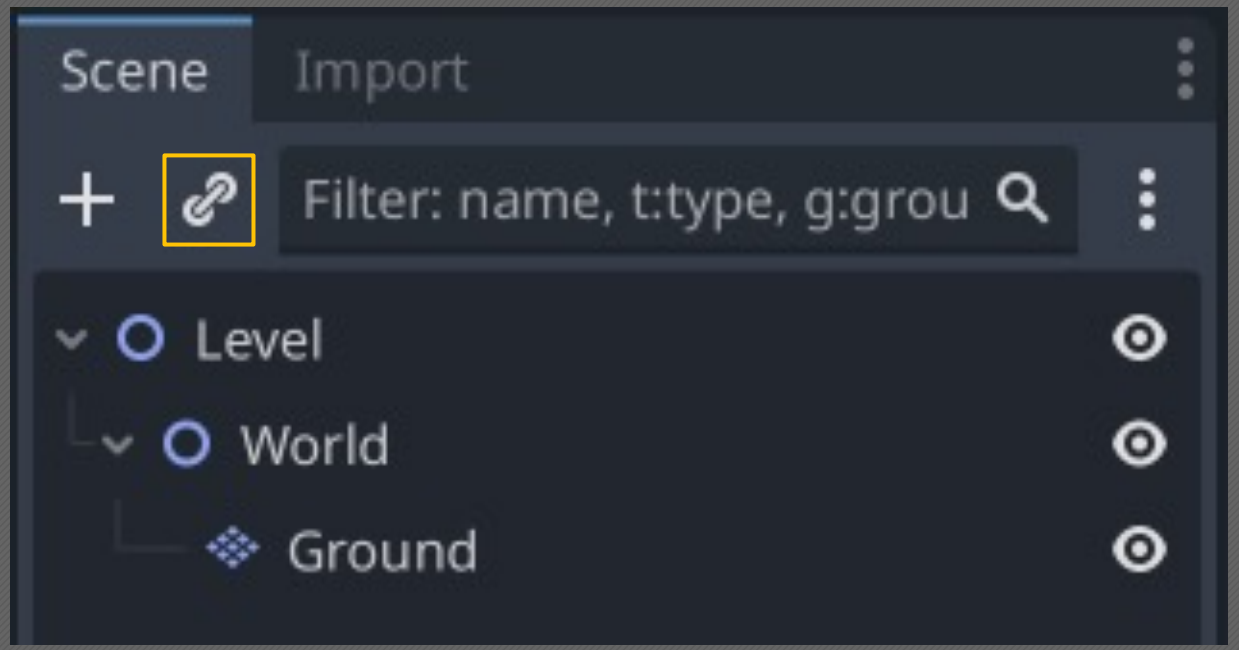


If you set up multiple terrains, they will all show up here.



Instantiating Child Scenes into Other Scenes

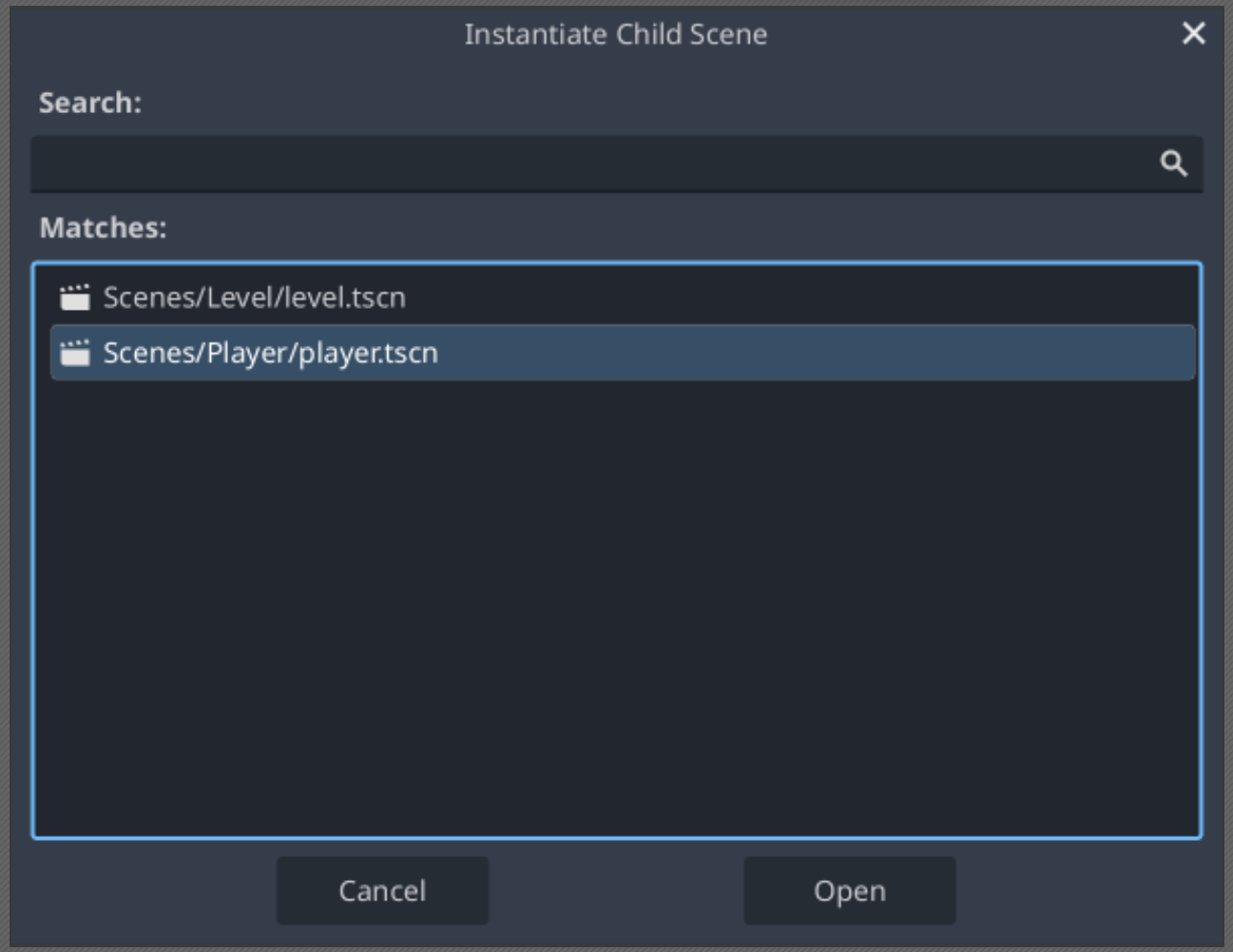
- Custom scenes can be instantiated into other scenes as child nodes.
- Scene nodes can be linked into another scene with the Instantiate Child Scene button.
- The linked scene will be added as a direct child of the currently-selected node.



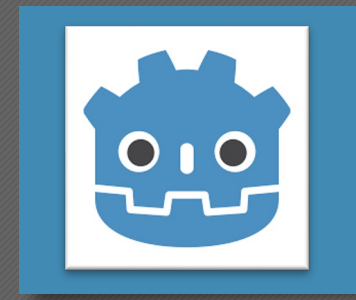
Instantiating Child Scenes into Other Scenes

- When the Instantiate Child Scene button is clicked, a prompt is presented to select which scene to instantiate.

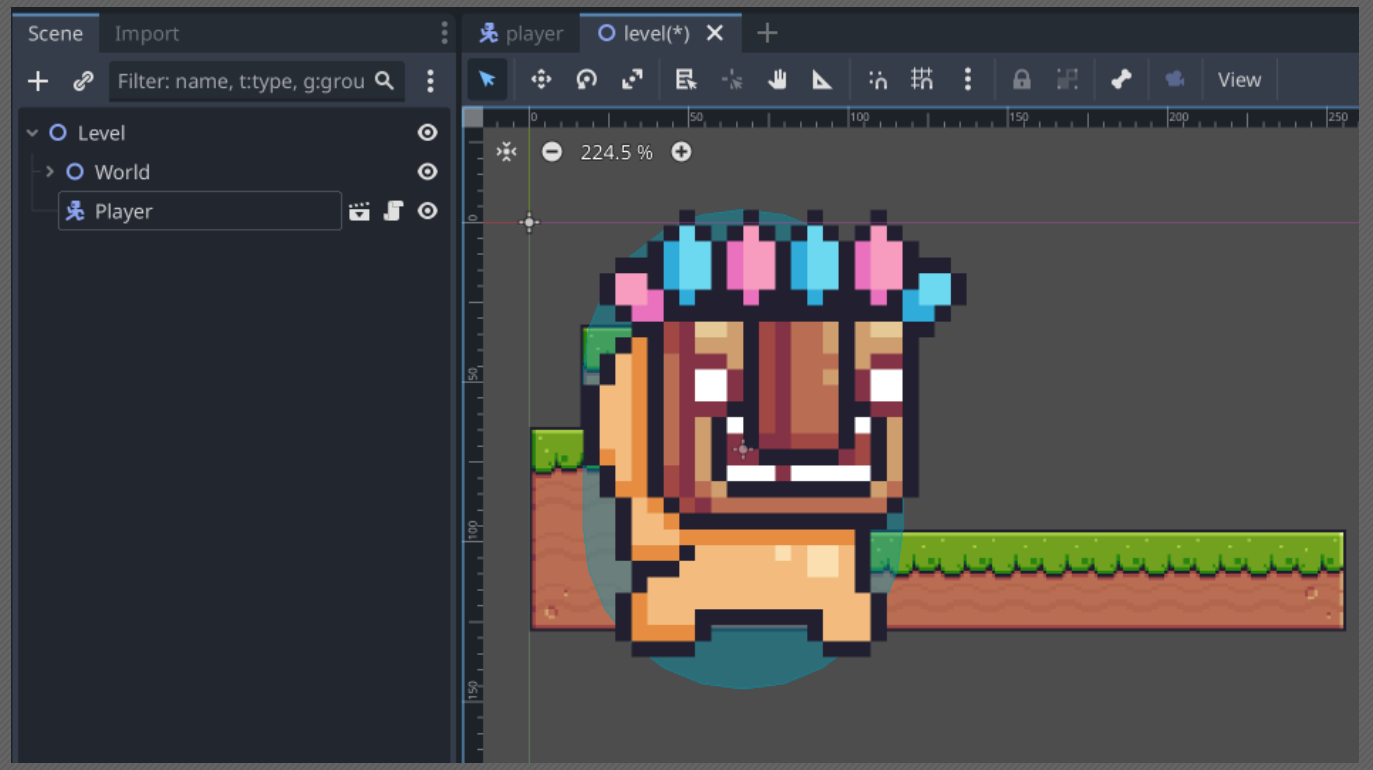
- Select the player scene to add the Player to the Level scene.



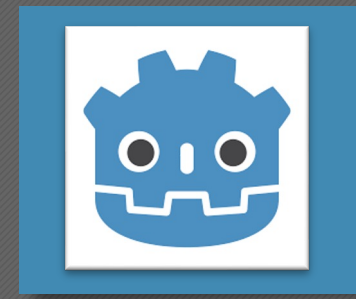
Verify Level Scene Tree



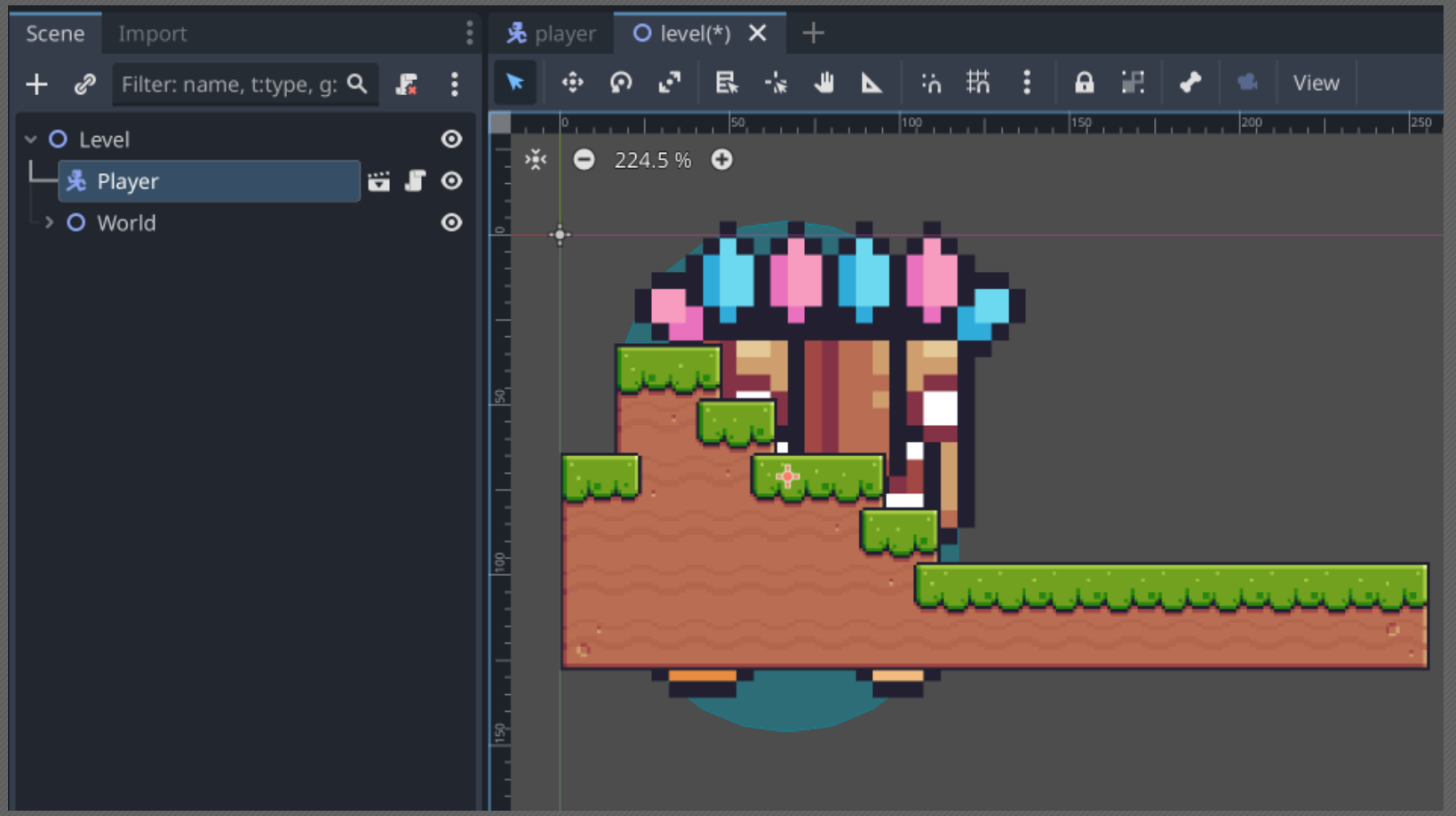
- Make sure the Player scene exists as a direct child of the root node (the Level node).
- Make sure the Player scene is under the World node as a sibling (not as a child of World).
- Notice the Player is huge and is rendered in front of the tiles since it is *after* the World node sibling.



Verify Level Scene Tree

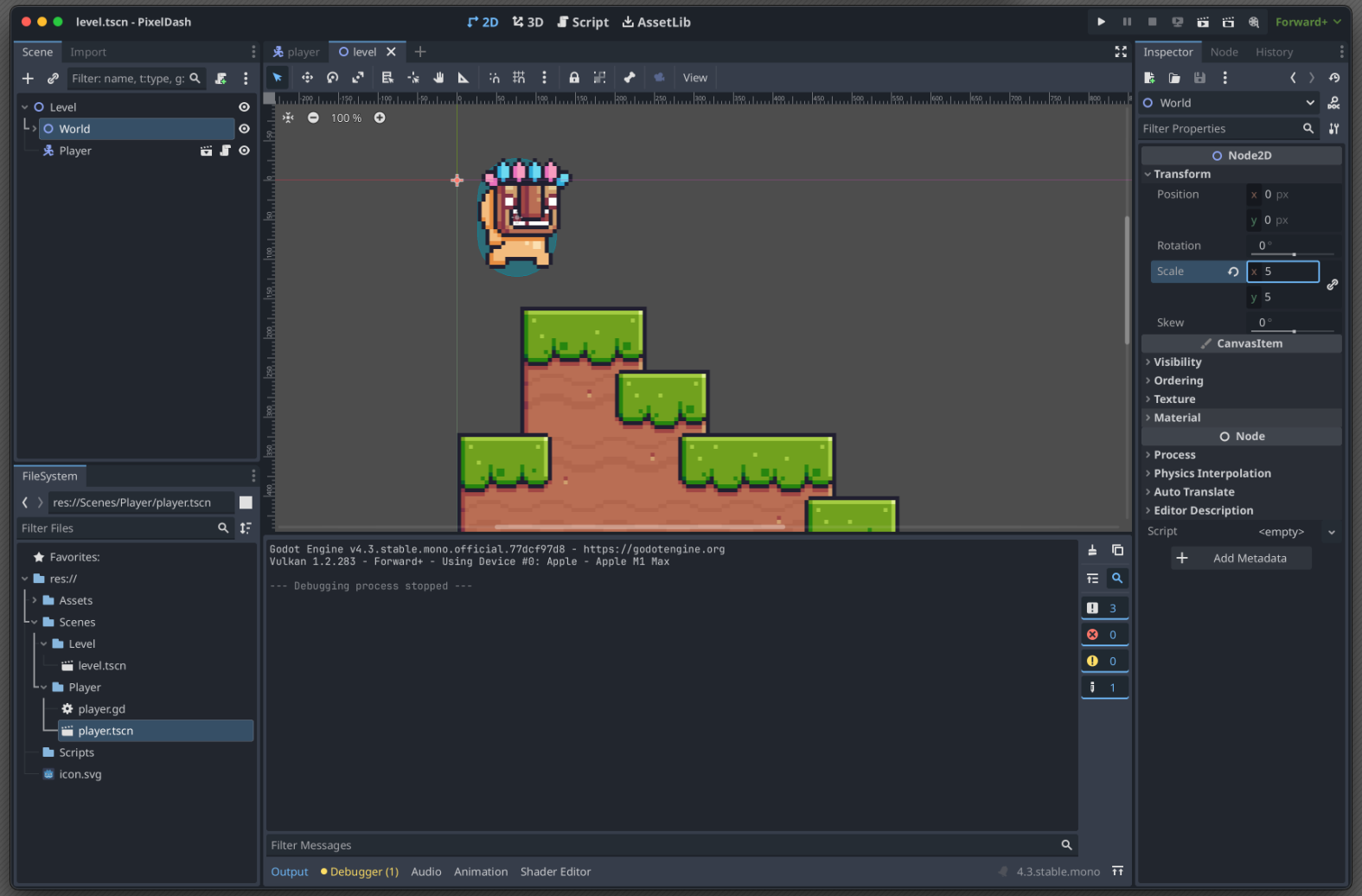


If the Player node comes before the World node, then the World node will be drawn in front of the Player (which is not ideal in the case of the workshop).

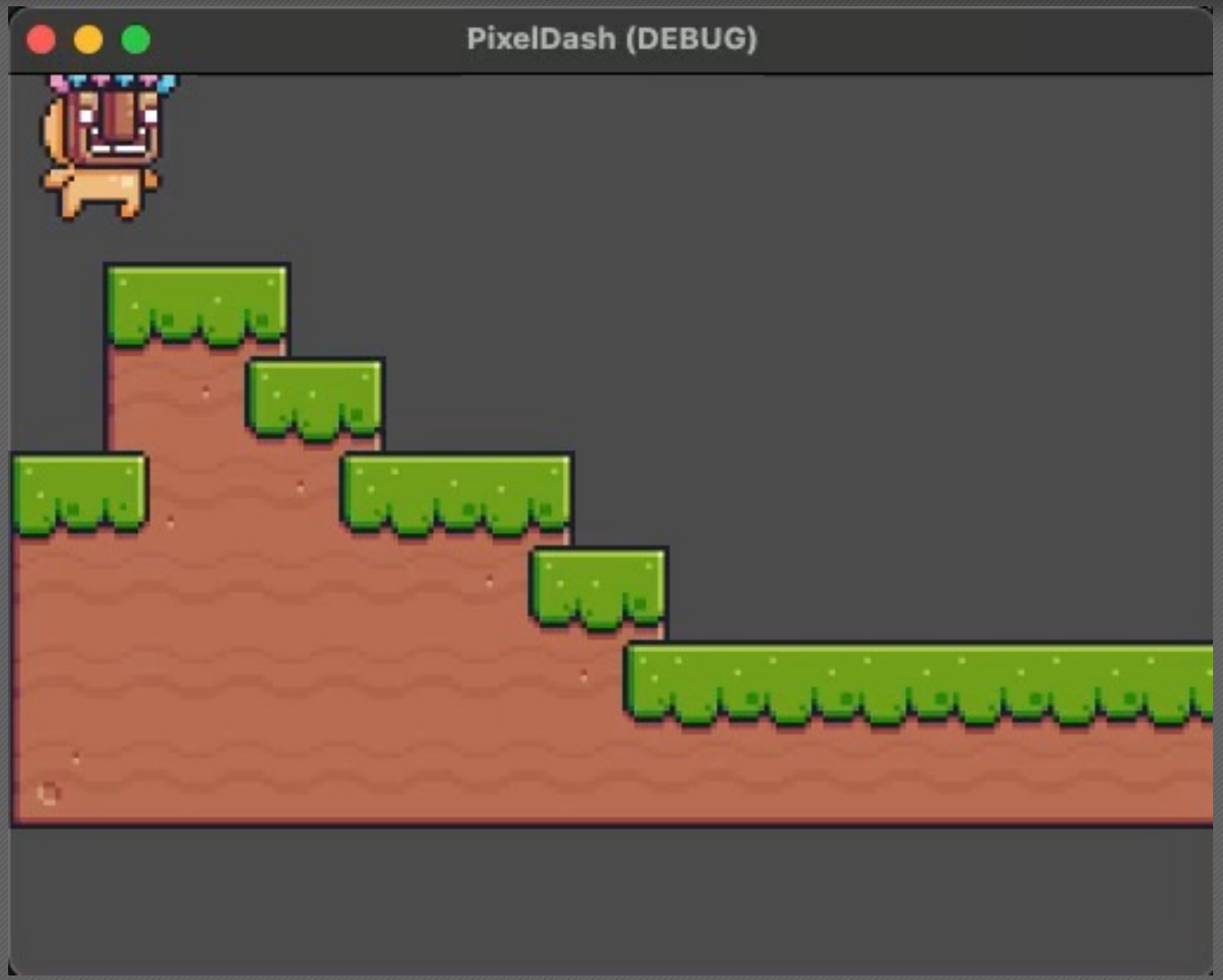


Fix the World's Scale property.

To fix the sizing of the game, update the World node's Scale property to be 5 instead of the default 1.



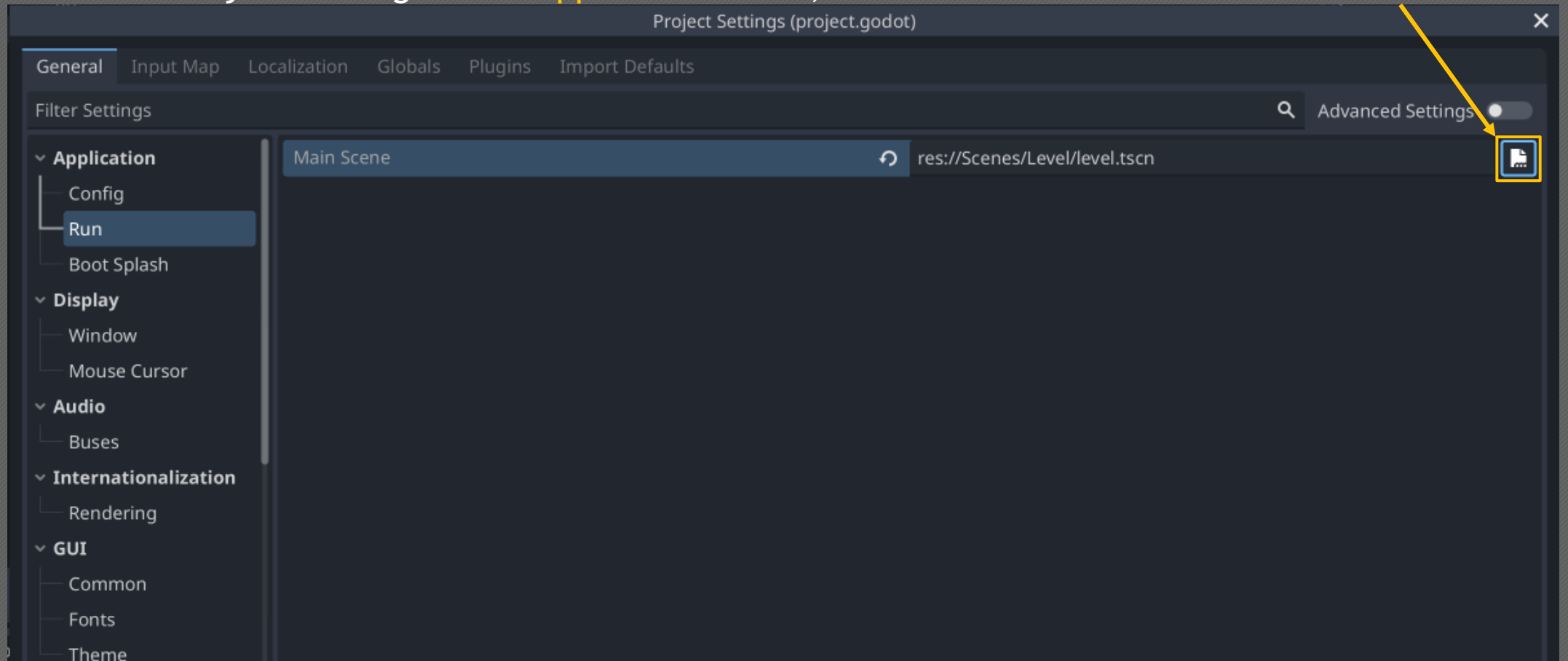
Ta-Da! Tiles that block the Player's movement can now be drawn into the world.



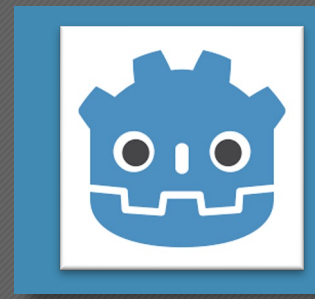
Set the Level as the Main Scene

If this is not already the case, the game should run the Level as the main scene.

Go into Project Settings under Application > Run, and link the Level scene as the Main Scene.



Lab Time (~25 Minutes)



- Update the Project Settings:

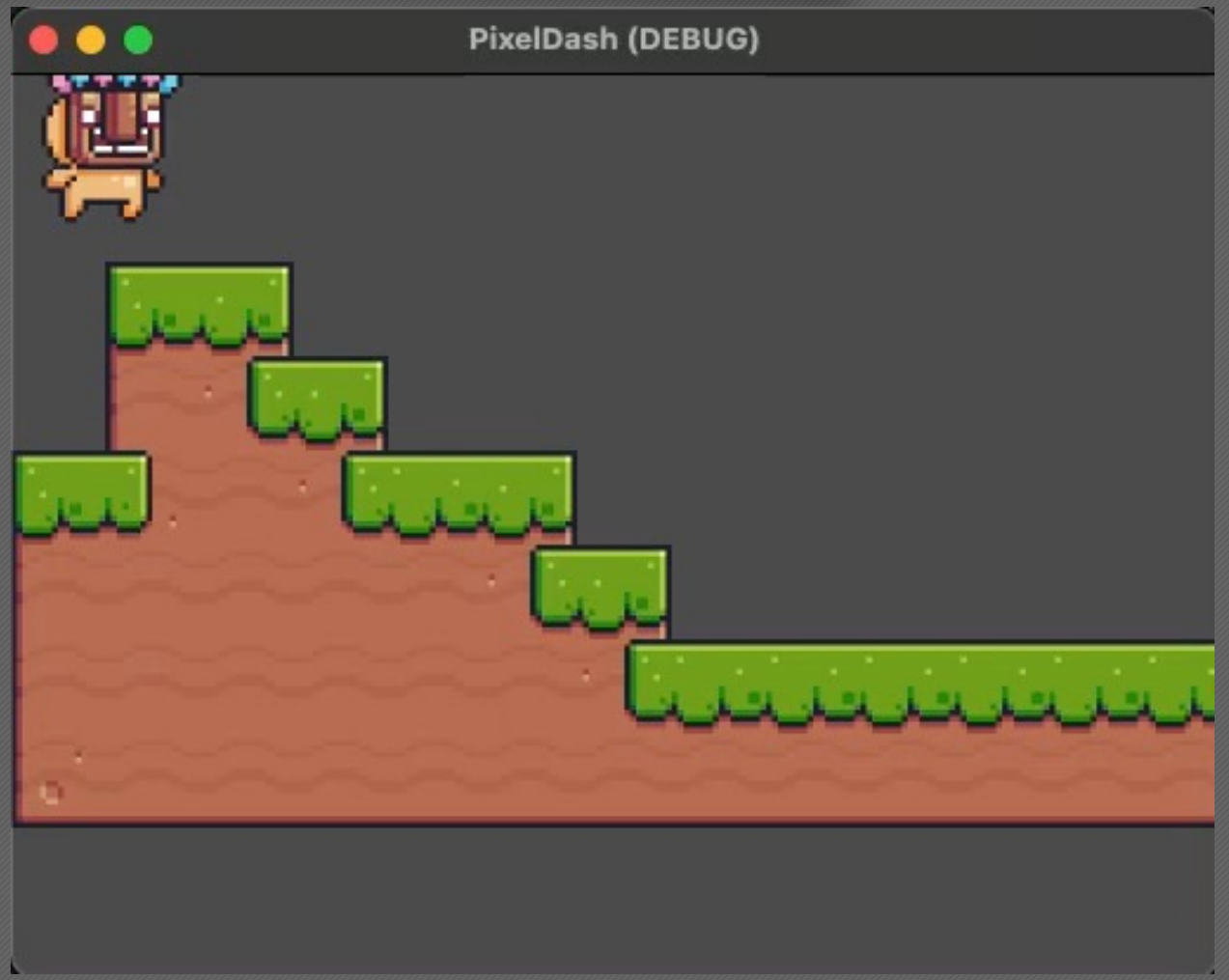
  Display > Window: Viewport Width and Height, and Stretch Mode = viewport.

  Rendering > Textures: Default Texture Filter to Nearest.

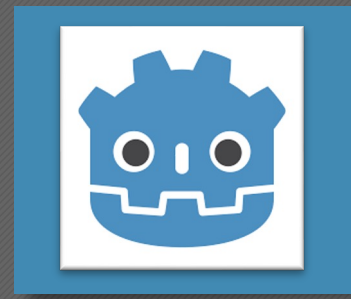
- Create the Level scene and attach an empty Level script to the scene.

- Set up the TileMapLayer's TileSet, with physics collision and at least one Terrain Set of your choice.

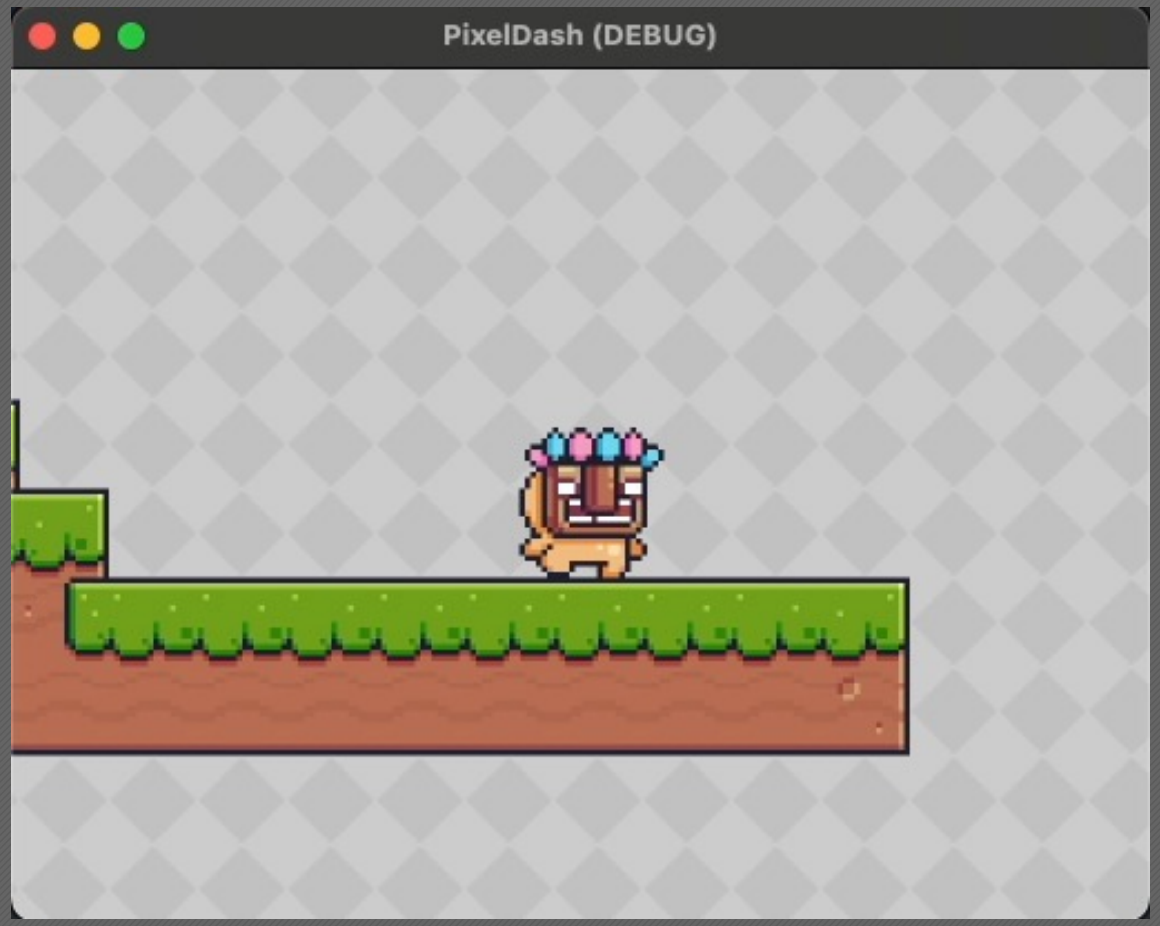
- Draw some tiles and verify the Player cannot pass through the tiles.



Workshop Goal #3

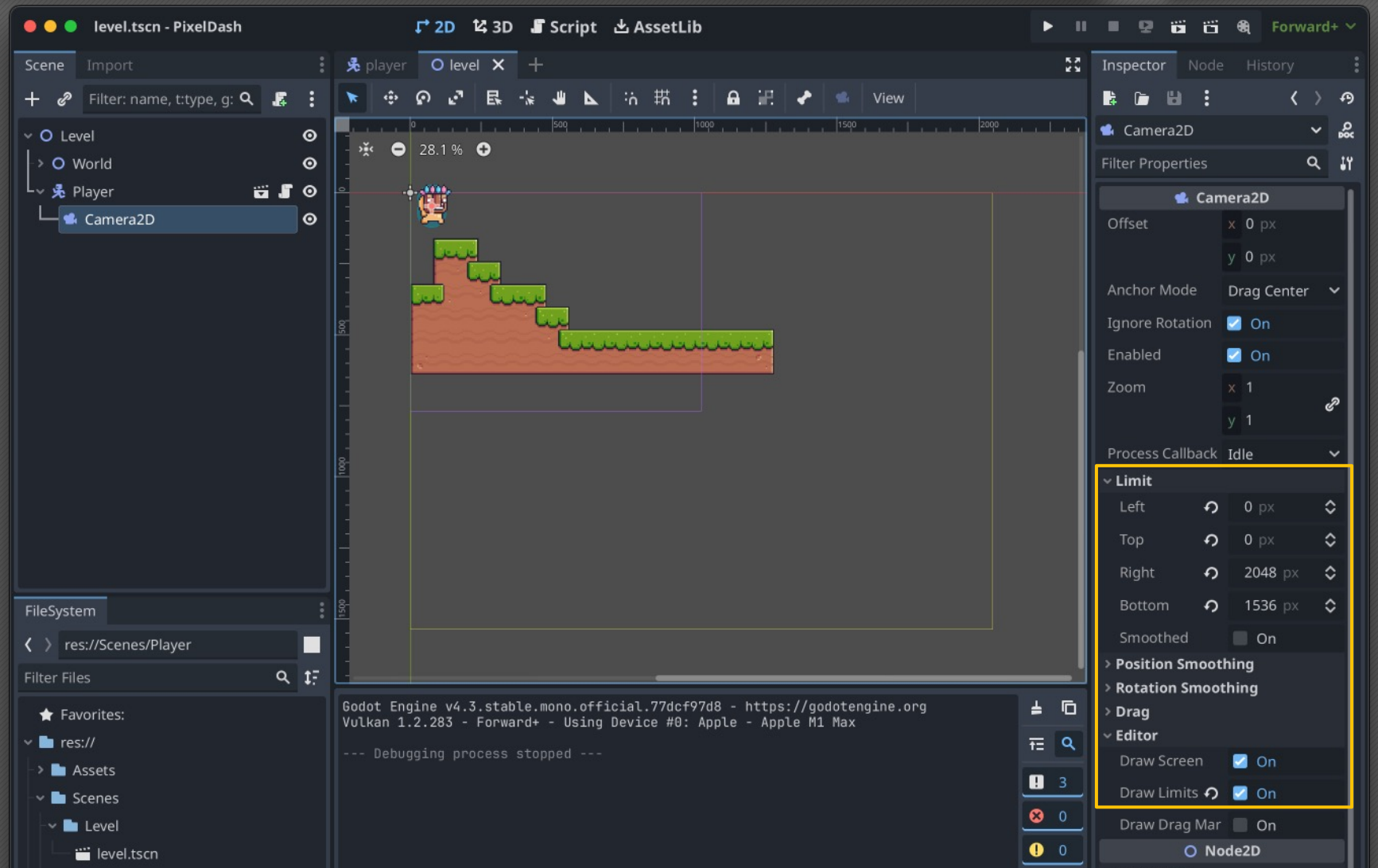


1. ✅ Control & Animate a Character
2. ✅ Build a Level with Tiles
3. ➡️ Add Player & Level Polish
   (Camera, Gravity, Sound, Background)
4. Detect Collisions with Trampolines
5. Create a basic Mushroom Enemy
6. Add Collectible Fruit and HUD Display
7. Respawn Character when Defeated
8. Create a Main Menu & Change Scenes
9. Open Workshop, Tinker Time



Adding a Camera that follows the Player

- Add a Camera2D node as a child of the instantiated Player in the Level Scene.
- Set the camera's Limits to prevent the camera from scrolling "outside" of the level tiles.



Adding a Camera that follows the Player

Currently, the scale of the World and Player nodes are both set to 5. It's far easier to Zoom with a camera to modify the game world size instead of trying to keep individual node scales consistent.

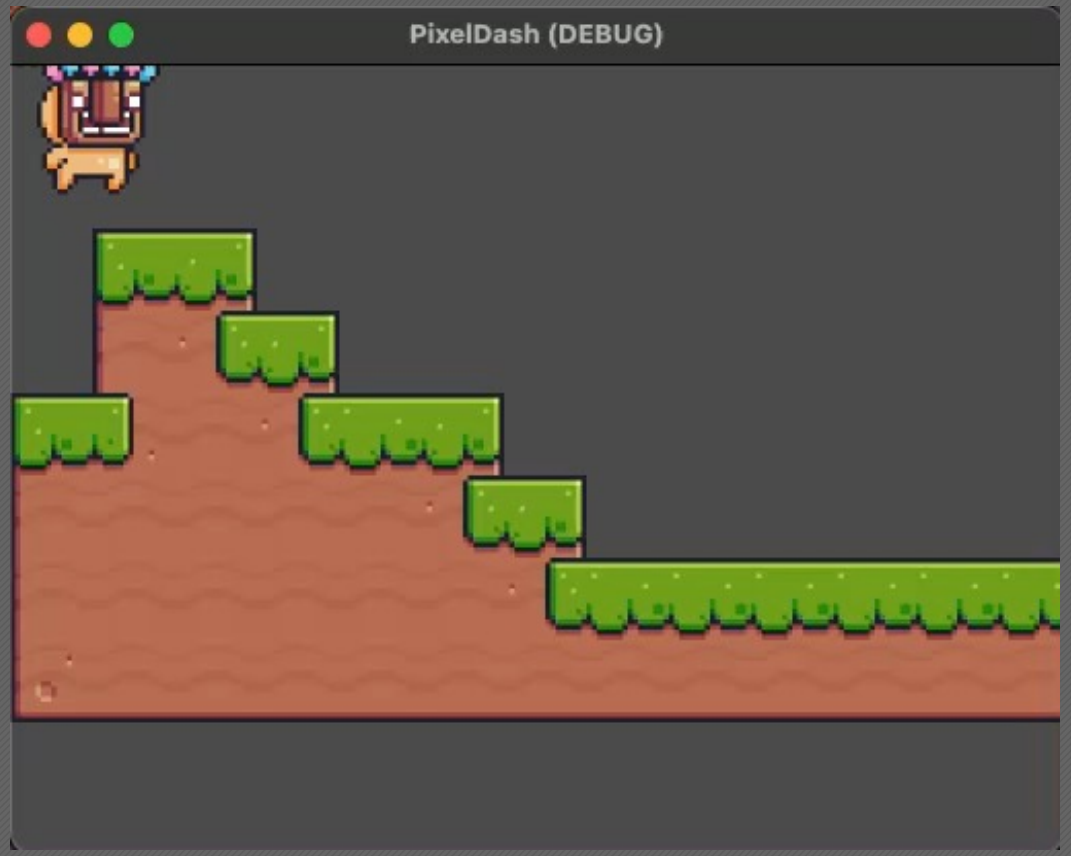
- Set the Camera2D Zoom to 5.
- Set the World and Player scales back to 1.

Adding a Camera that follows the Player

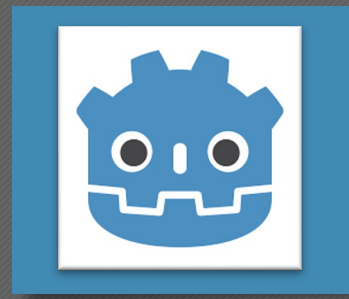
```
3    const BASE_MOVE_SPEED := 250
```

The current BASE_MOVE_SPEED in the Player script is very high to accommodate the scale of the nodes.

- Update the Player script to have a much lower and more reasonable BASE_MOVE_SPEED.



Adding Gravity and Jumping to the Player



In the Player script, declare a couple new const variables (GRAVITY and JUMP_FORCE) and update the input_vector to be the result of Input.get_axis instead of the current Input.get_vector func.

Then set Player's X velocity (for moving left and right) and update the Y velocity to be the current Y velocity times GRAVITY. Remember that negative Y is up, and positive Y is down.

```
3   const BASE_MOVE_SPEED := 250
4   const GRAVITY := 35
5   const JUMP_FORCE := GRAVITY * 14
6
7   func _physics_process(delta: float) -> void:
8       # Shorthand for getting the right (positive) input minus the left (negative) input
9       var input_vector := Input.get_axis("left", "right")
10
11      # The input vector returns 0 for "not pressed" and 1 for "pressed", so multiply by move speed.
12      velocity.x = input_vector * BASE_MOVE_SPEED
13      velocity.y += GRAVITY
```

Adding Gravity and Jumping to the Player

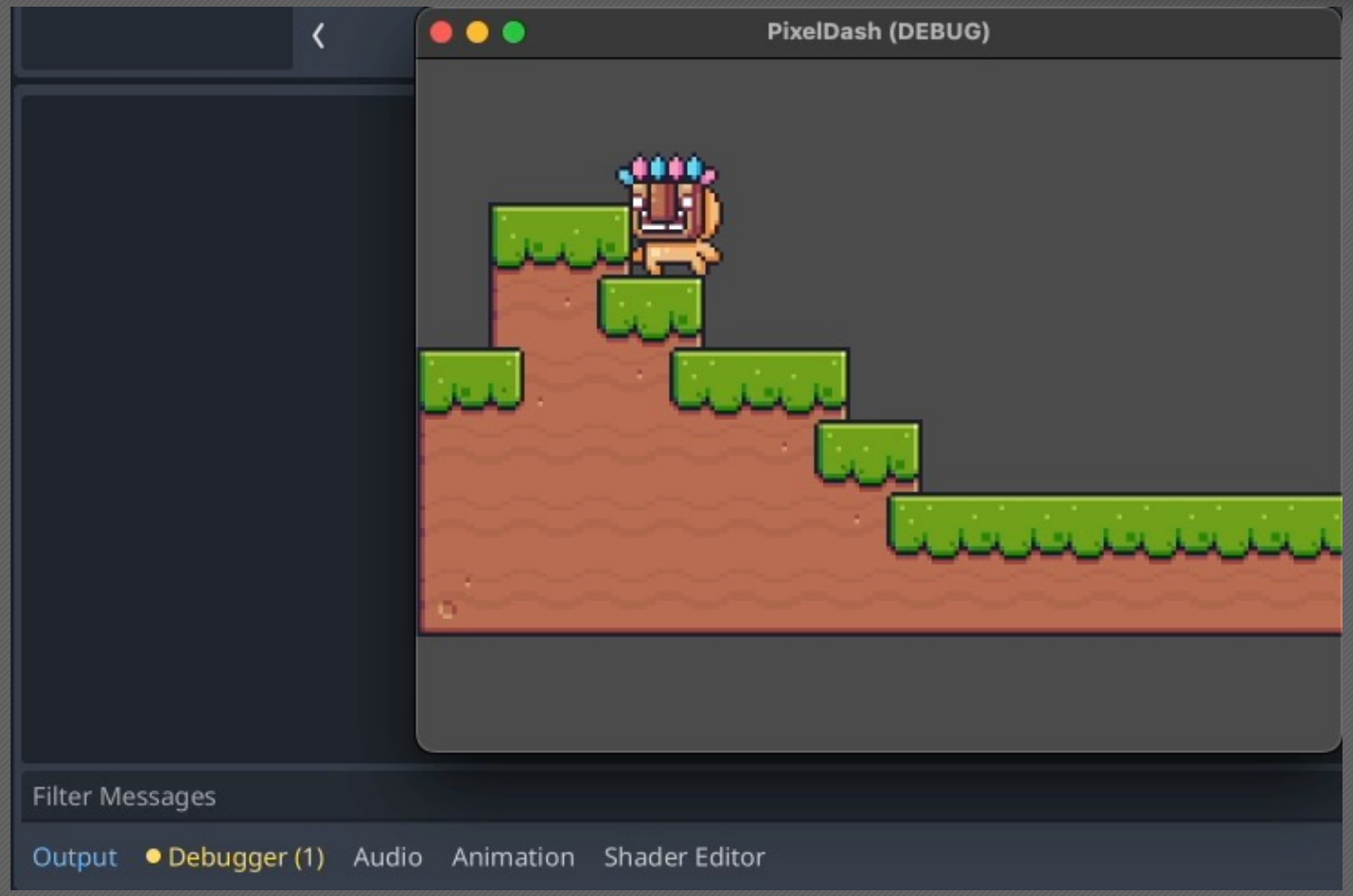
To add a Jump, the Player script needs to listen for input via the _input Node function.

Call the `is_action_pressed` func on the event to check that the “action” input (Space key) is pressed. The `is_on_floor` func is a very useful function from the CharacterBody2D class - it returns true if the body collided with the floor on the last call of `move_and_slide()`, otherwise it returns false.

If both are true, then the Player should jump! Subtract the Player’s Y velocity by the JUMP_FORCE.

```
37  func _input(event: InputEvent) -> void:
38      if event.is_action_pressed("action") and is_on_floor():
39          velocity.y -= JUMP_FORCE
40          print("Velocity: %s" % velocity)
```

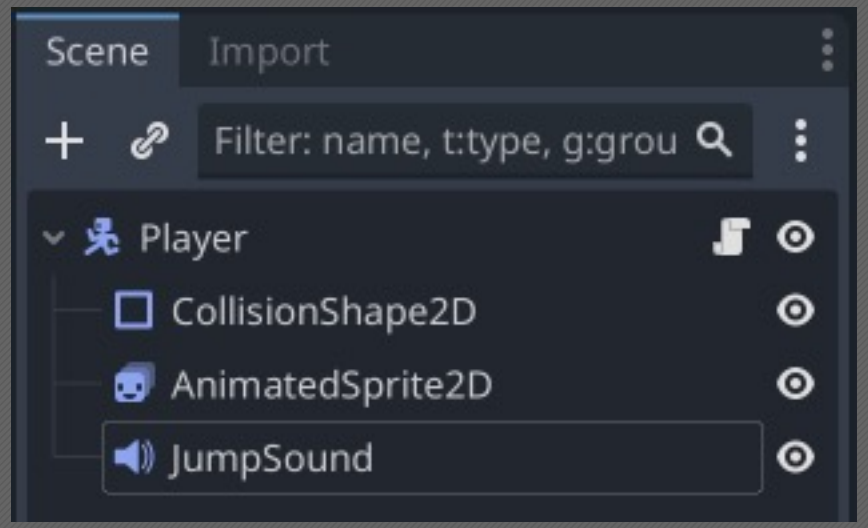
Adding Gravity and Jumping to the Player

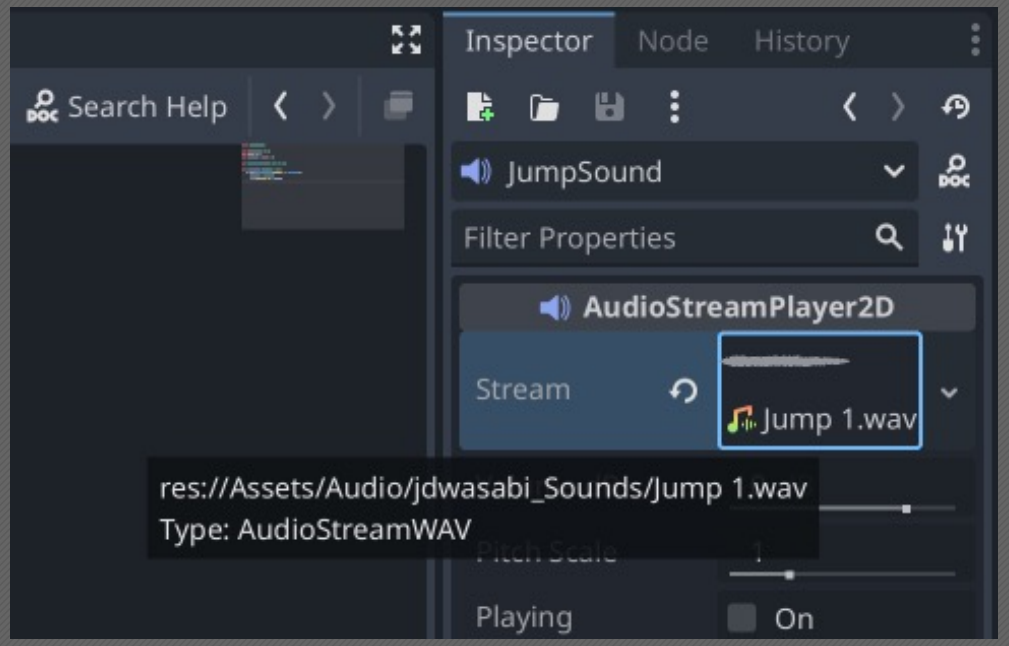


Adding a Jump Sound to the Player

Add an AudioStream2D as a direct child of the root node in the Player scene and name it “JumpSound”.

Then, drag and drop the Assets/Audio/jdwasabi_Sounds/Jump 1.wav file to the Stream property value.





Adding a Jump Sound to the Player

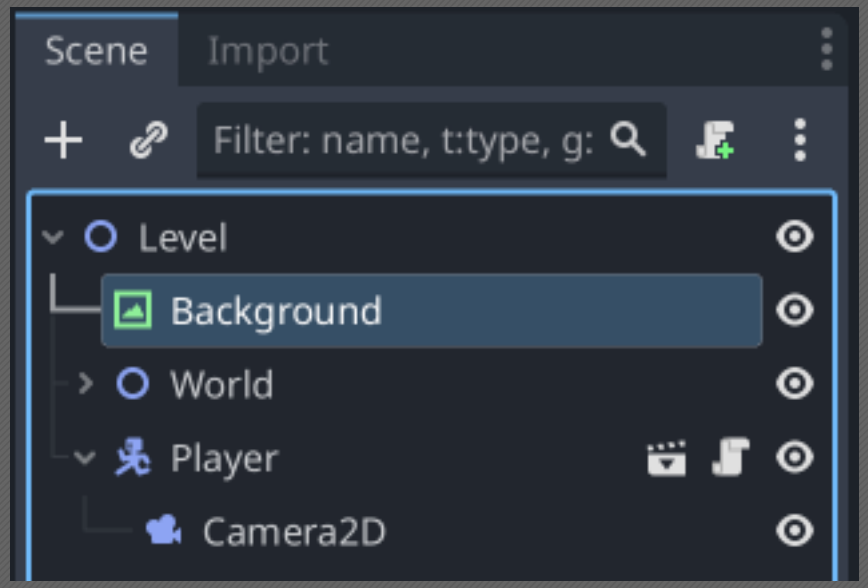
Back in the Player script's _input func, remove the print call (if desired) and add a call to `$JumpSound.play()`.

```
37  func _input(event: InputEvent) -> void:
38      if event.is_action_pressed("action") and is_on_floor():
39          velocity.y -= JUMP_FORCE
40          $JumpSound.play()
```

Adding a Background to the Level

- Add a TextureRect node as an immediate child of the Level scene and name it Background.

- Make sure the Background is the first child.



Adding a Background to the Level

Update the Background's properties as follows.

Texture: any image under

Assets/Art/Pixel Adventure 1/Background/???

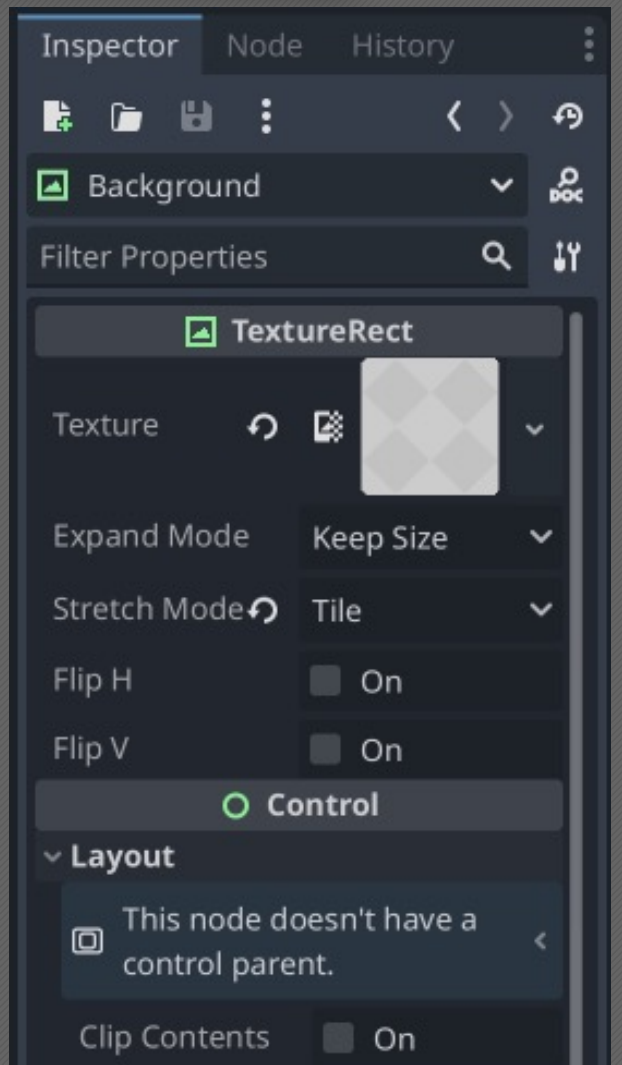
Stretch Mode: Tile

Layout:

Custom Minimum Size: 1920 x 1080

(or your game window size)

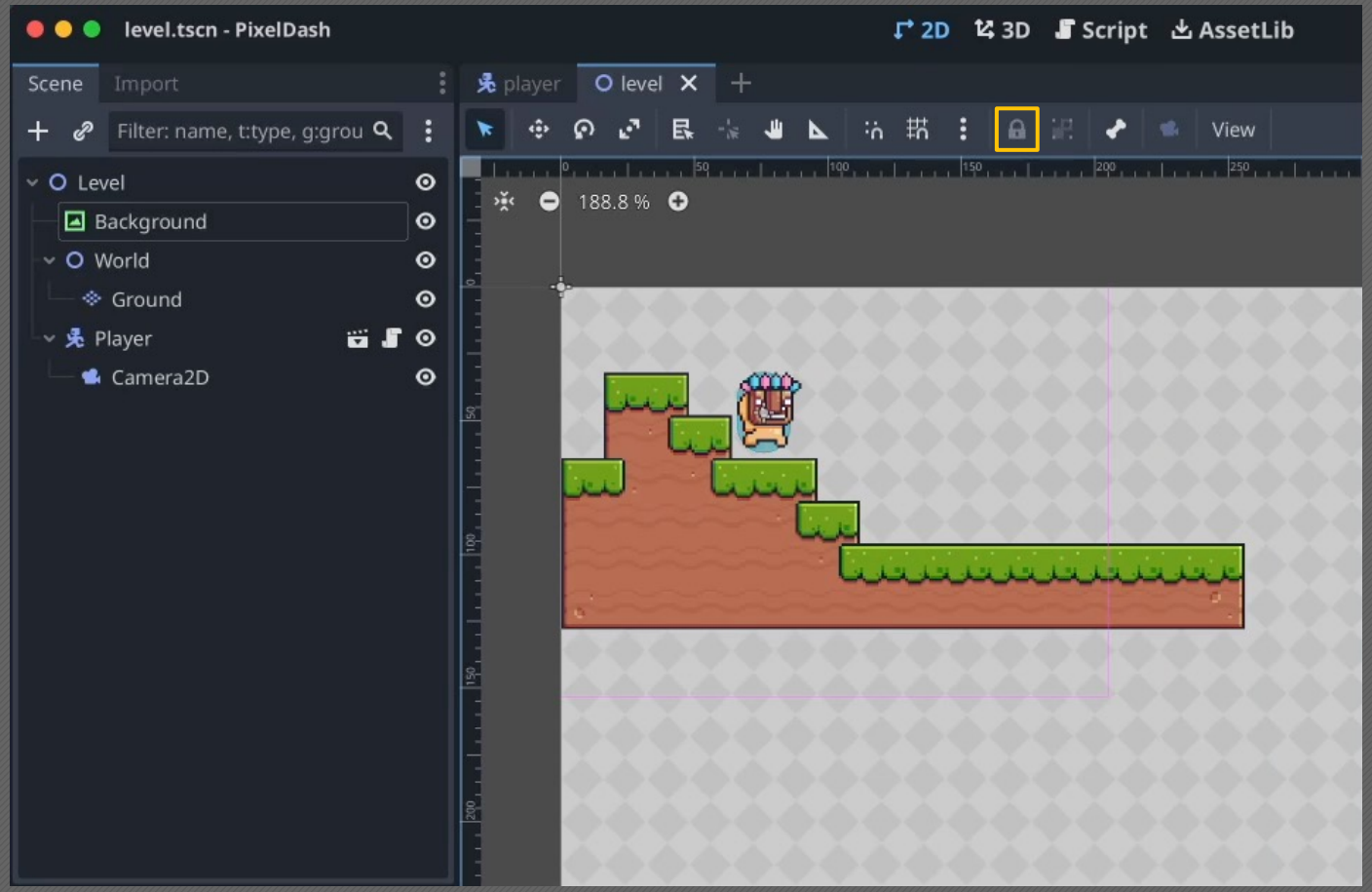
Transform > Scale: 0.5



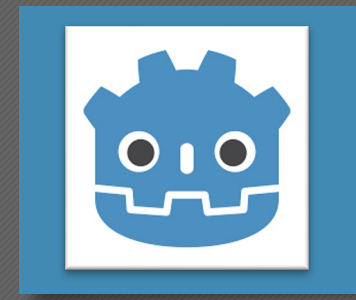
Results of Adding a Background to the Level

Locking Nodes in the Scene View

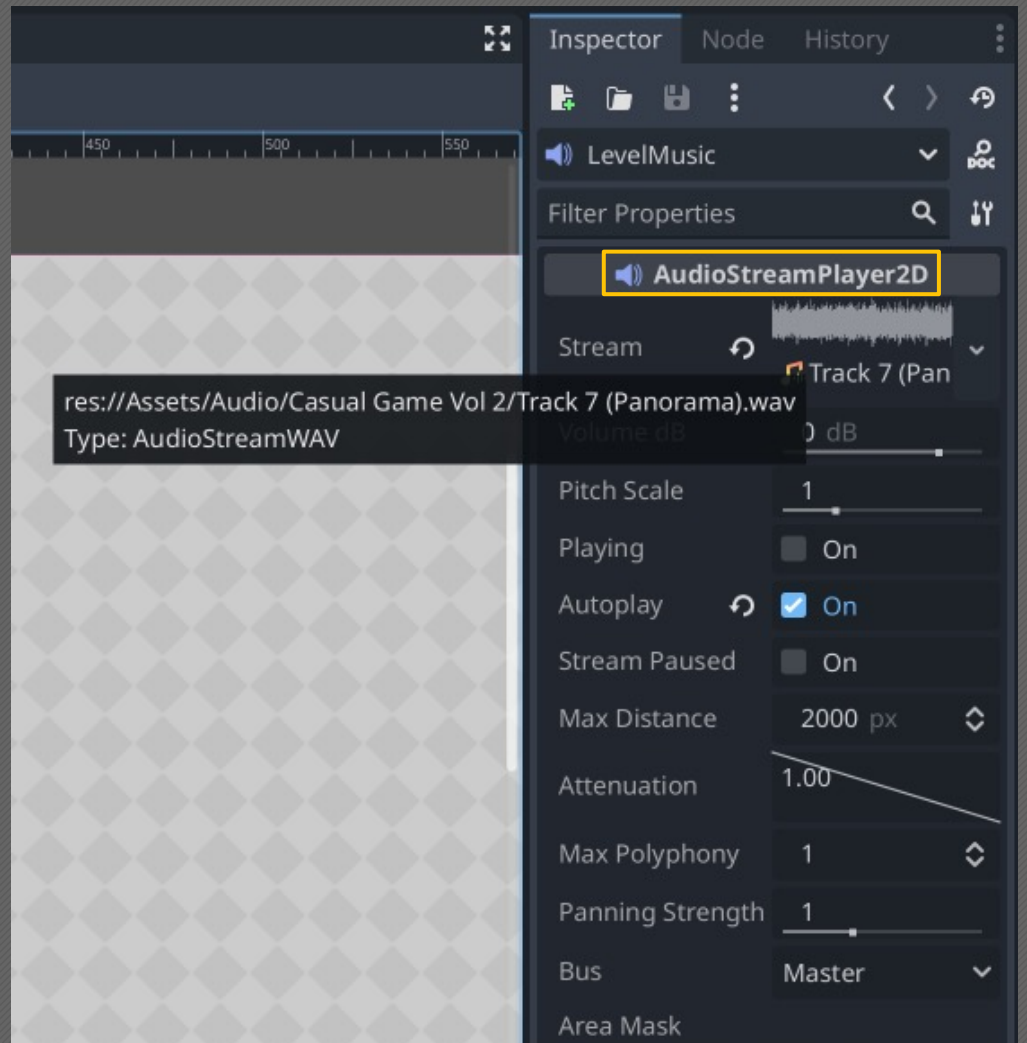
Any node in the scene tree can be toggled locked/unlocked, which can be very helpful for nodes of which are not intended to be selected or interacted with frequently in the scene view.



Adding Background Music to the Level



- Attach a Script to the Level scene.
- Add an AudioStreamPlayer2D node to the Level scene and name it LevelMusic.
- Assign one of the soundtracks from the Assets/Audio/Casual Game Vol 2 folder to the Stream property.
- Set the Autoplay property to On.



Intro to Signals

“Call down, Signal up”

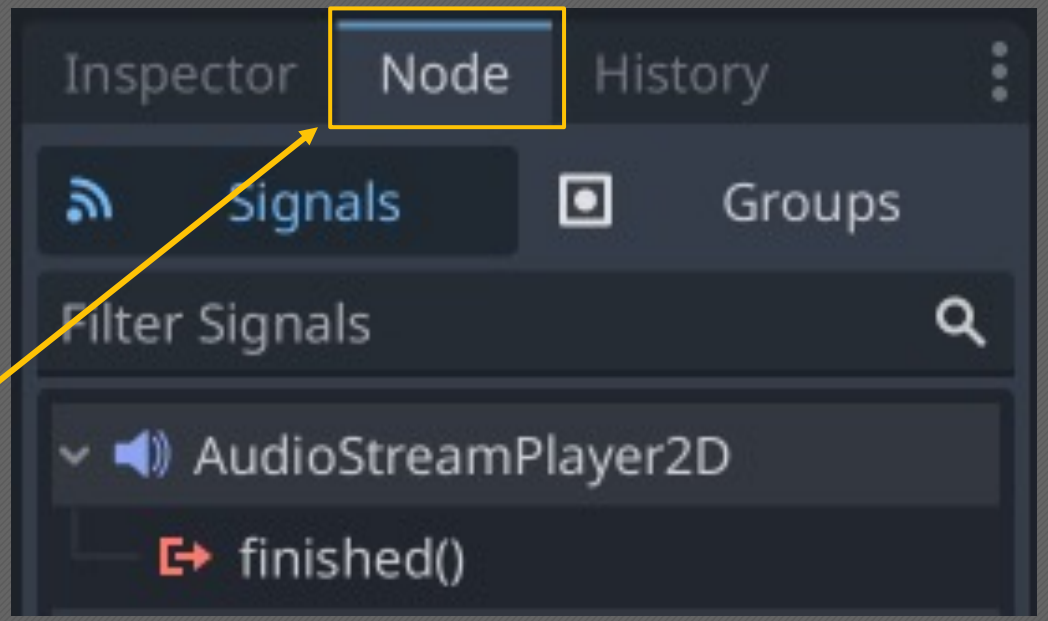
- Signals are messages that nodes can emit when something specific has happened (such as a button being pressed, or an area being entered).
- Other nodes or scripts can connect to and listen for specific signals, allowing those components to react when the signals are emitted.
- You can connect signals via either the Node tab (to the right of the Inspector tab) or directly through code.

Looping the Background Music

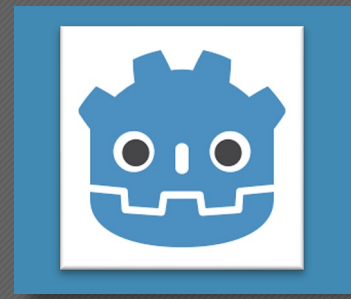
The music will not loop by default, and there is no property to do so.

With the LevelMusic node selected, navigate to the Node tab.

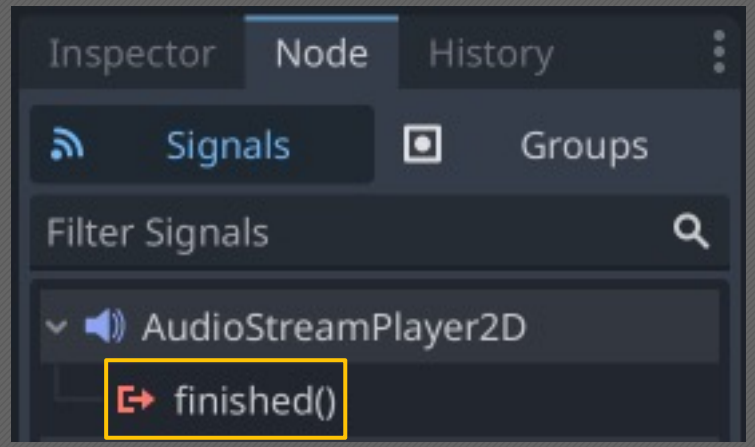
To the right of the *Inspector* tab



Looping the Background Music

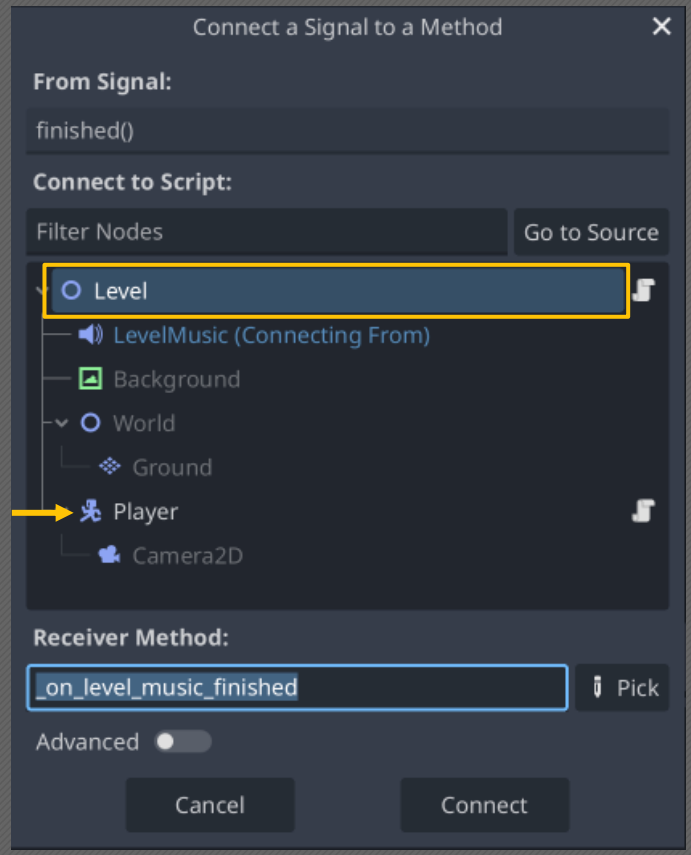


Double click the finished signal.



Accept the defaults, ensuring that the Level node/script is selected.

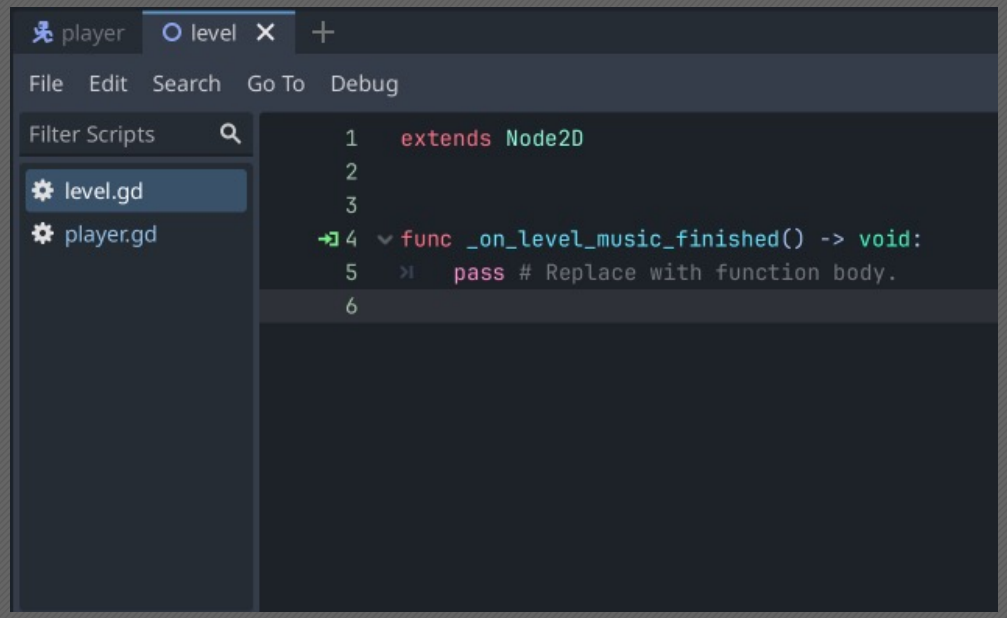
Make sure you do not have the Player selected.



Looping the Background Music

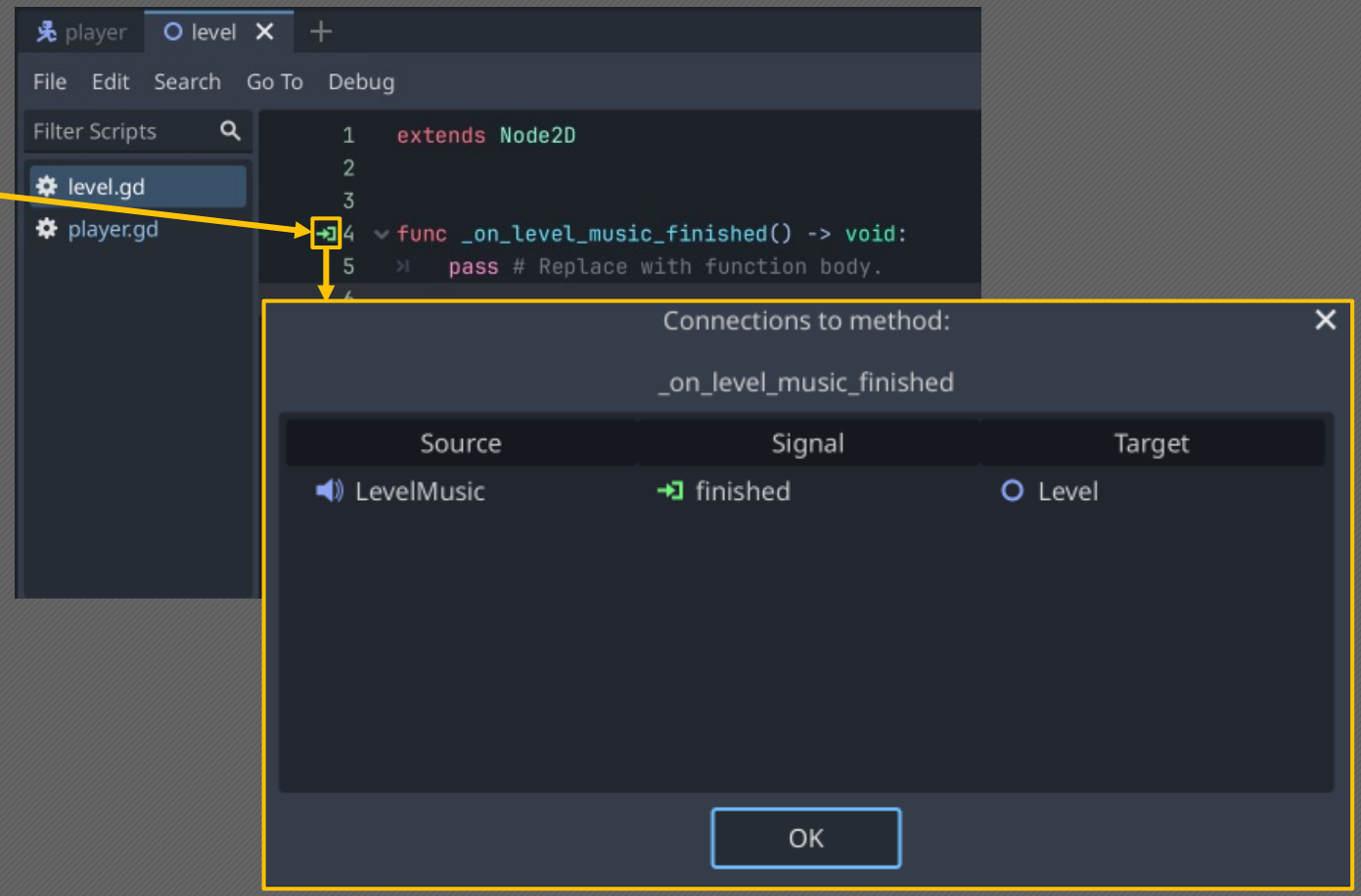
This is the default handler that Godot generates for this signal.

Notice the green “arrow pointing at a box” icon in the gutter – this indicates that the function is linked to a signal.



Looping the Background Music

The icon can be clicked to view the signal connection details.

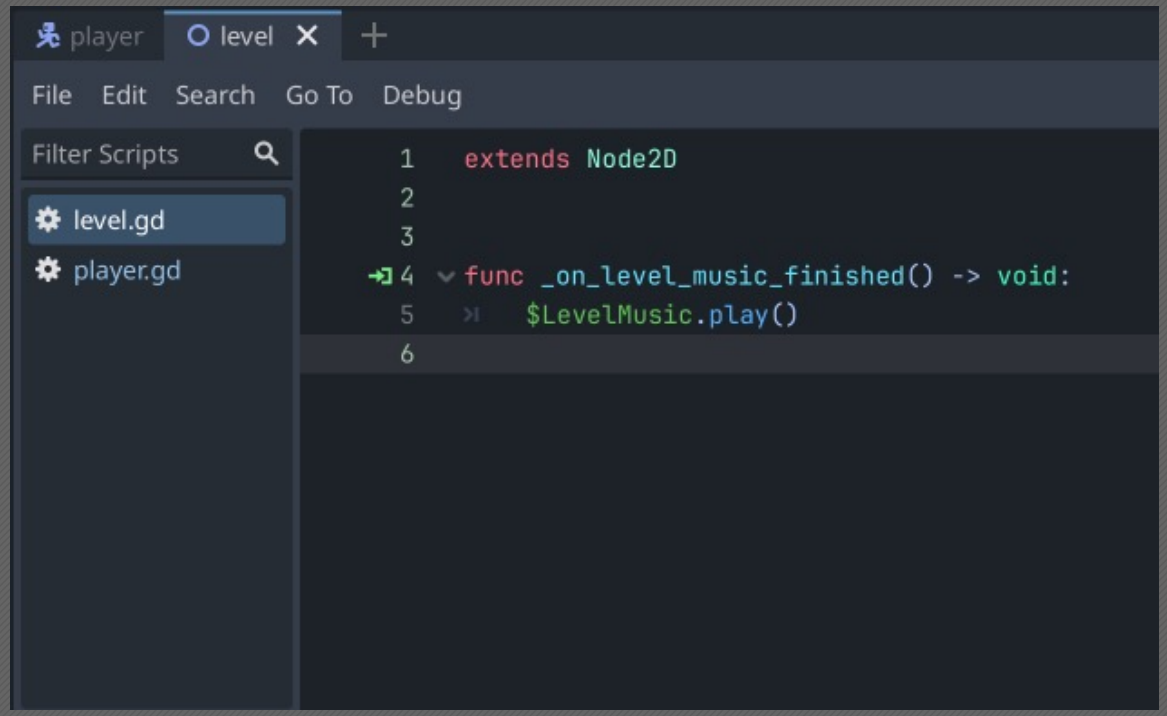


Looping the Background Music

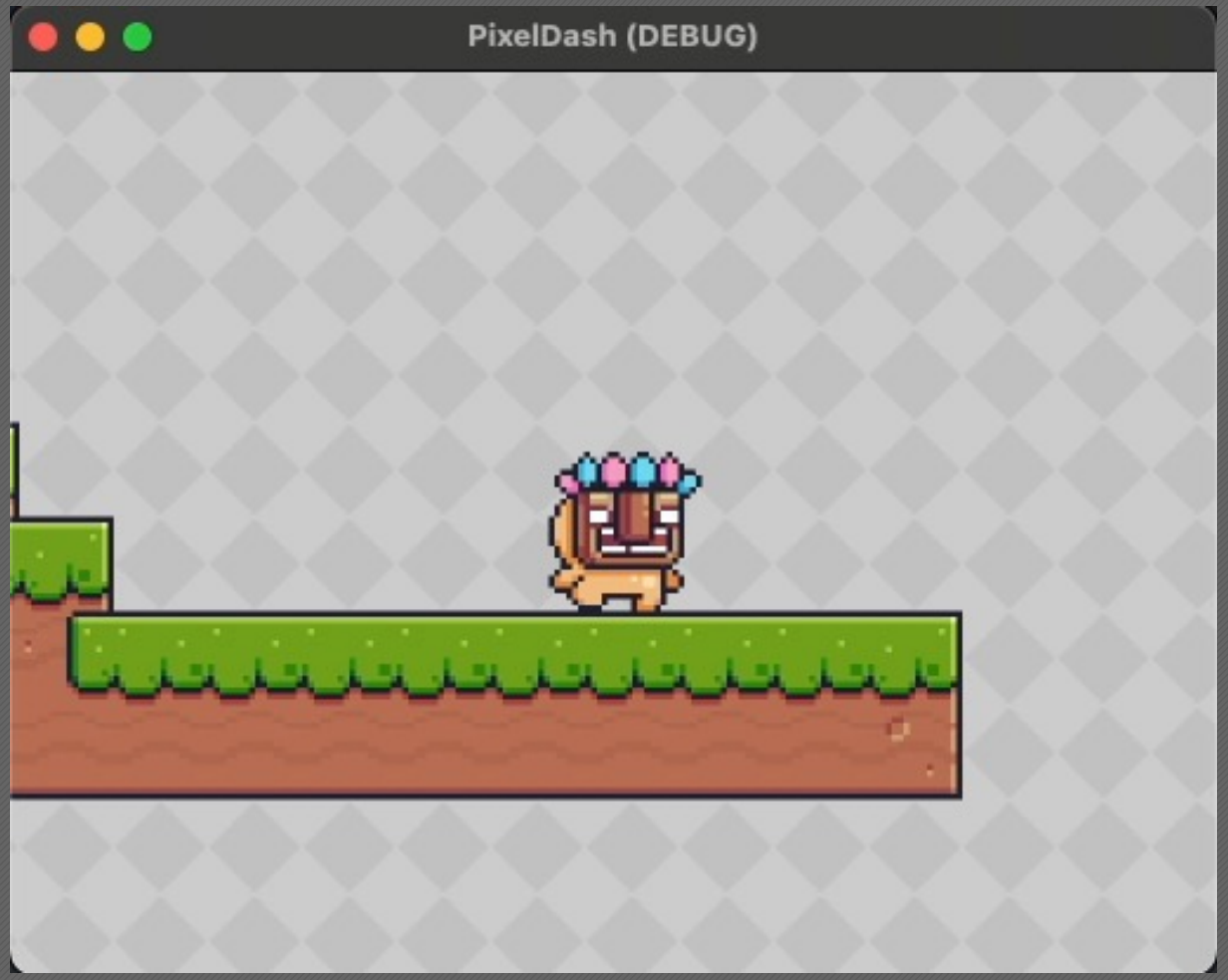
Replace the default pass implementation with a call to $LevelMusic.play().

Now, when the audio stream finishes playing, the finished signal will be emitted.

The Player script will handle that event and play the LevelMusic again, creating an audio loop.

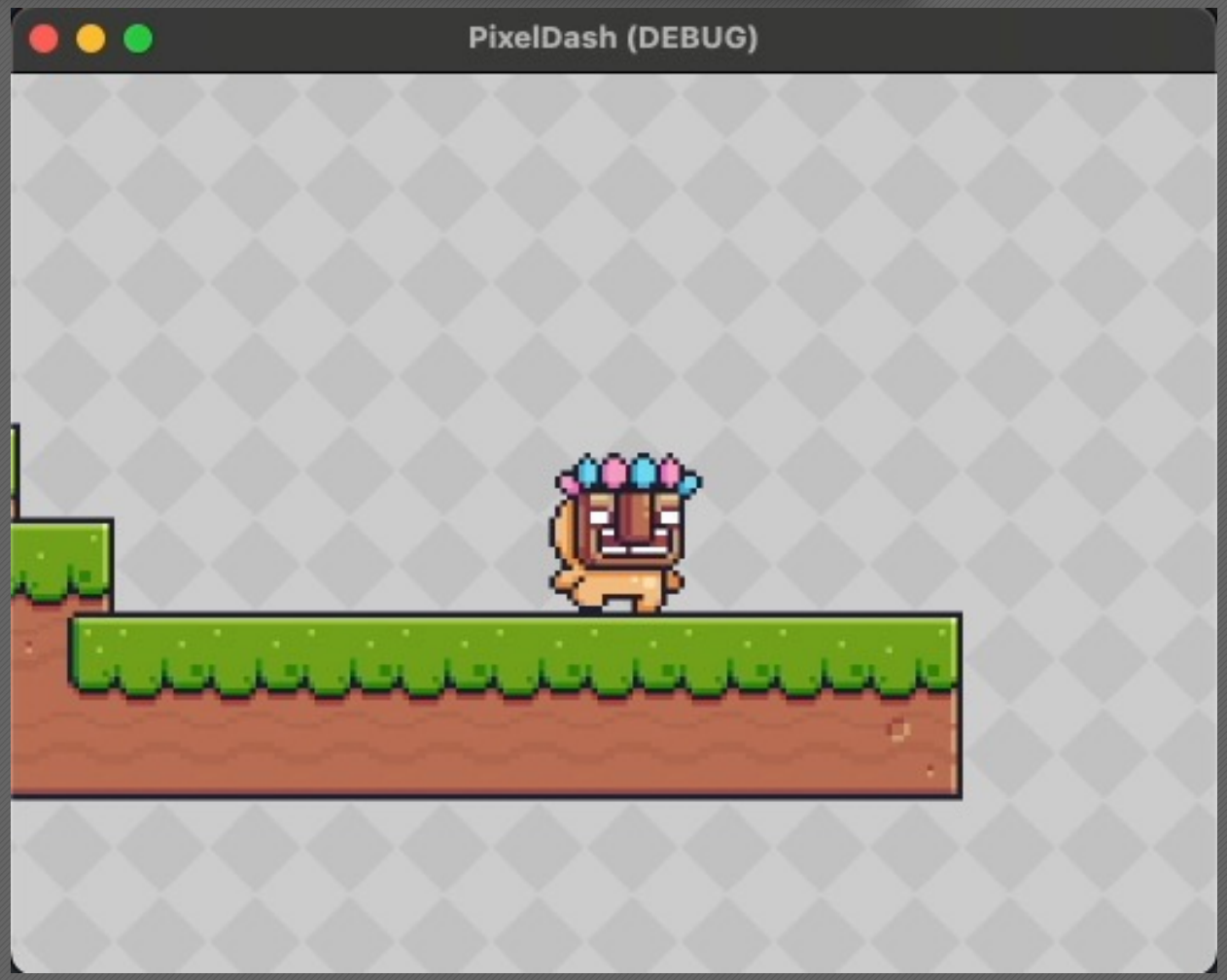


Polish results!

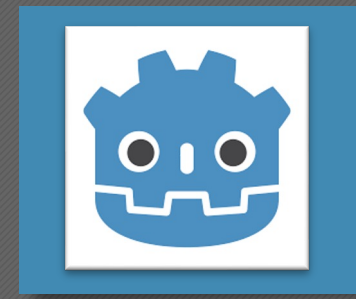


Lab Time (~10 Minutes)

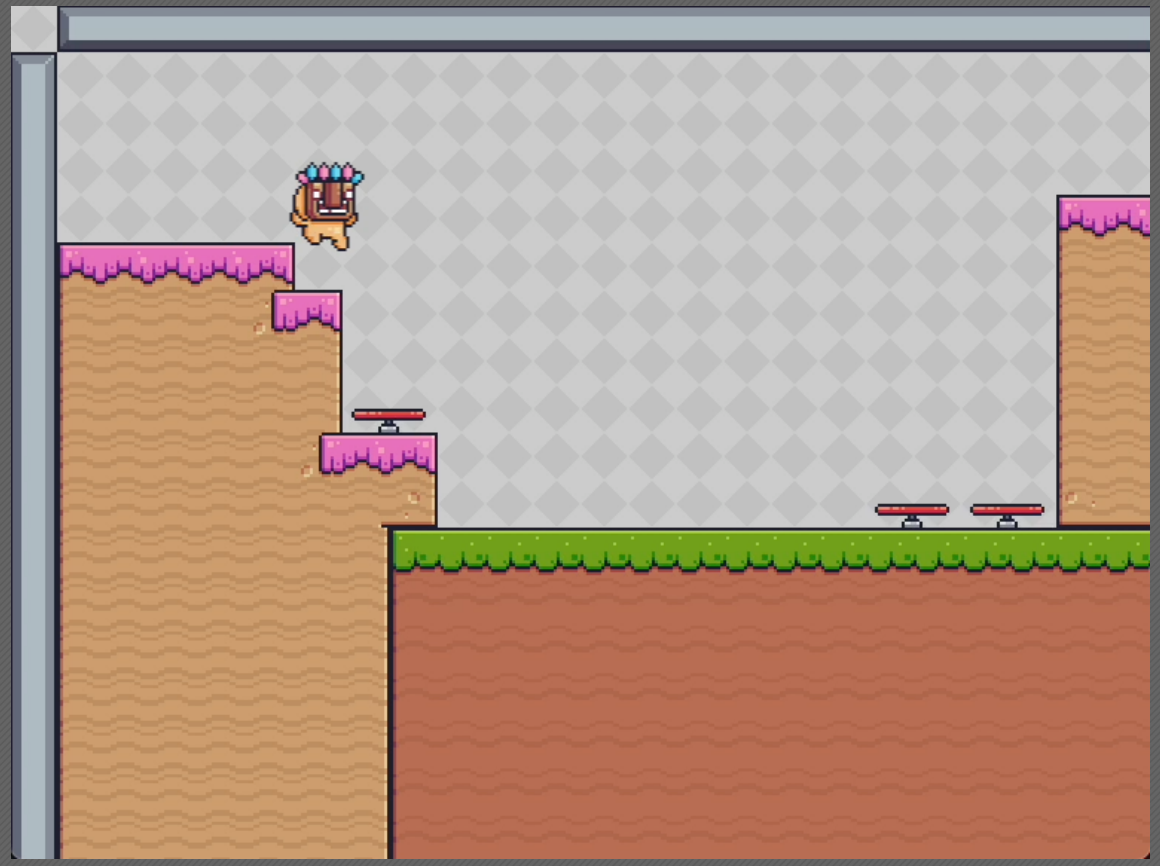
- Add a Camera2D that follows the Player in the Level scene.
- Add gravity and jump to the player, with a Jump sound.
- Add a tiling Level background.
- Add looping background music to the Level.



Workshop Goal #4

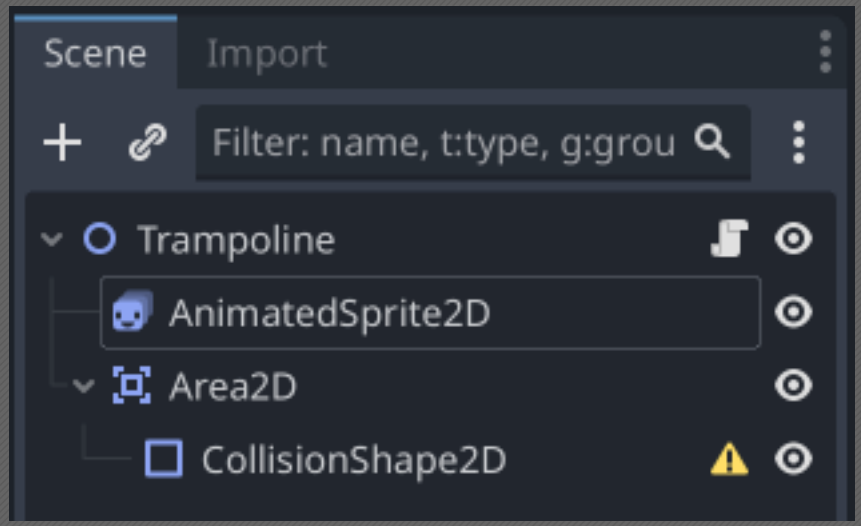


1. ✅ Control & Animate a Character
2. ✅ Build a Level with Tiles
3. ✅ Add Player & Level Polish
   (Camera, Gravity, Sound, Background)
4. ➡️ Detect Collisions with Trampolines
5. Create a basic Mushroom Enemy
6. Add Collectible Fruit and HUD Display
7. Respawn Character when Defeated
8. Create a Main Menu & Change Scenes
9. Open Workshop, Tinker Time

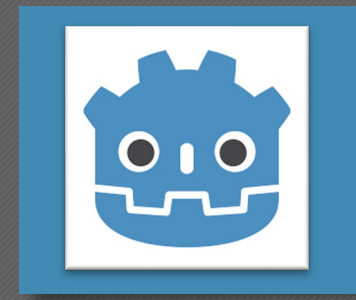


Create a new ”Trampoline” Node2D Scene

- Create a new Node2D scene named “Trampoline” and attach a new script.
- Give it the scene tree structure shown to the right.
- Create/attach a trampoline script.
- Make sure the CollisionShape2D is a child of the Area2D node.

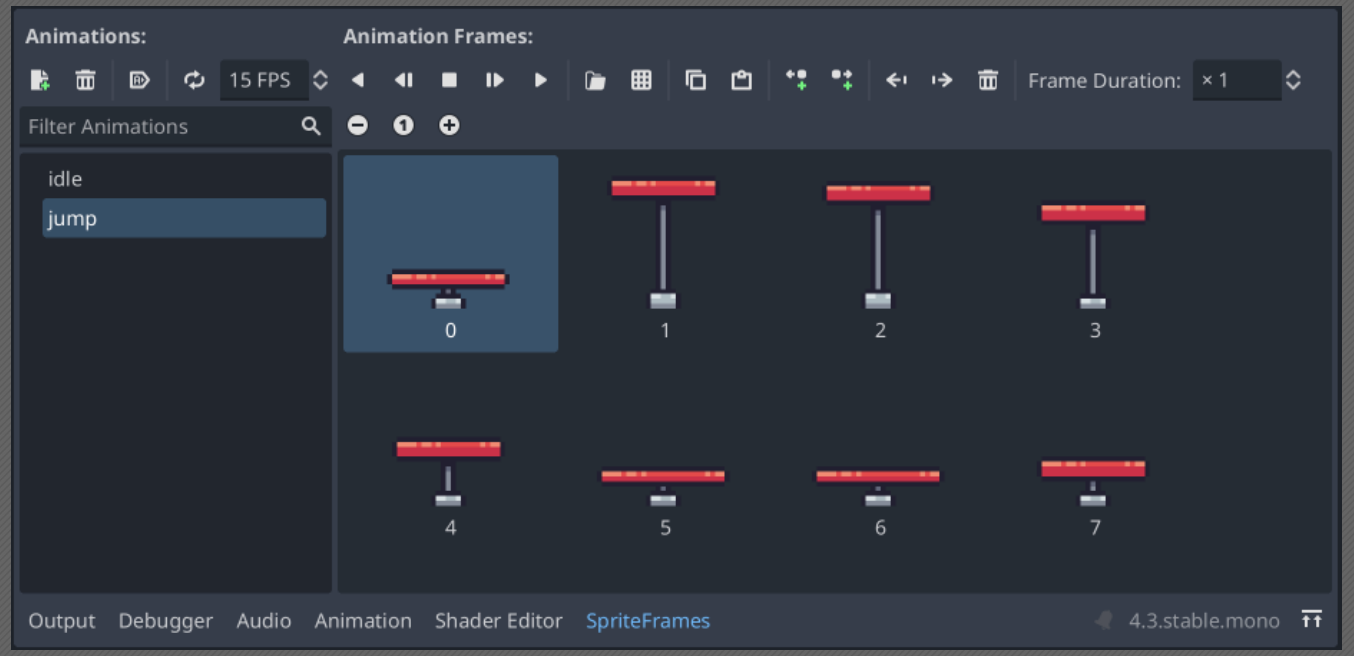


Create Animations for the AnimatedSprite2D



- Create an idle animation that has a single sprite frame that has the trampoline in the full down position.

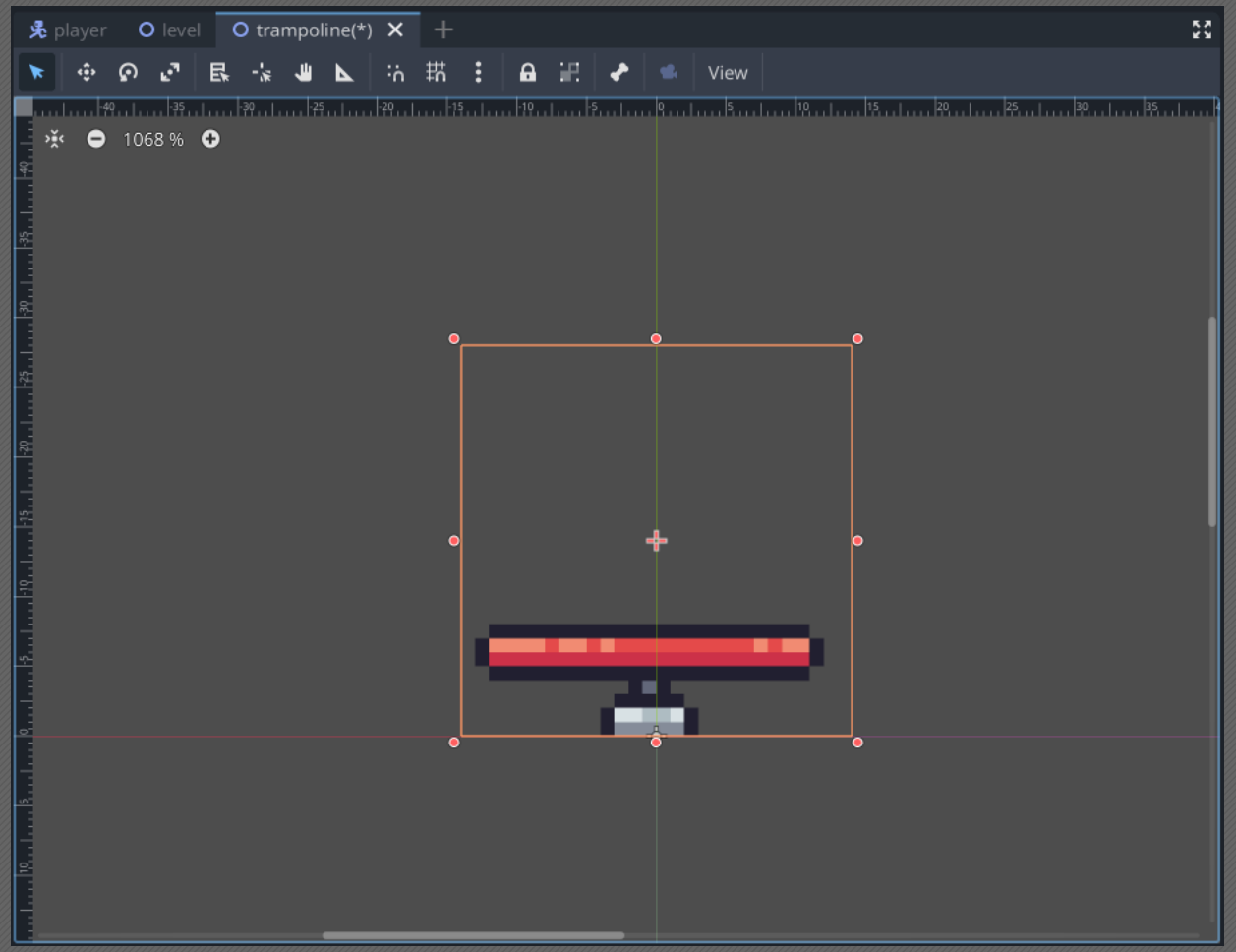
- Create a jump animation with 15 FPS and no looping that uses all frames of the trampoline sprite sheet. The Jump sprite size is 28x28.



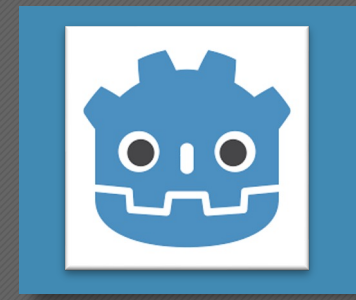
Reposition the Trampoline AnimatedSprite2D

Unlike the Player, which sits in the “center” of the scene, the Trampoline sprite needs to sit on the ground since it won’t be affected by gravity.

Reposition the animated sprite node to a position where the bottom of the trampoline sits on the center point of the scene.

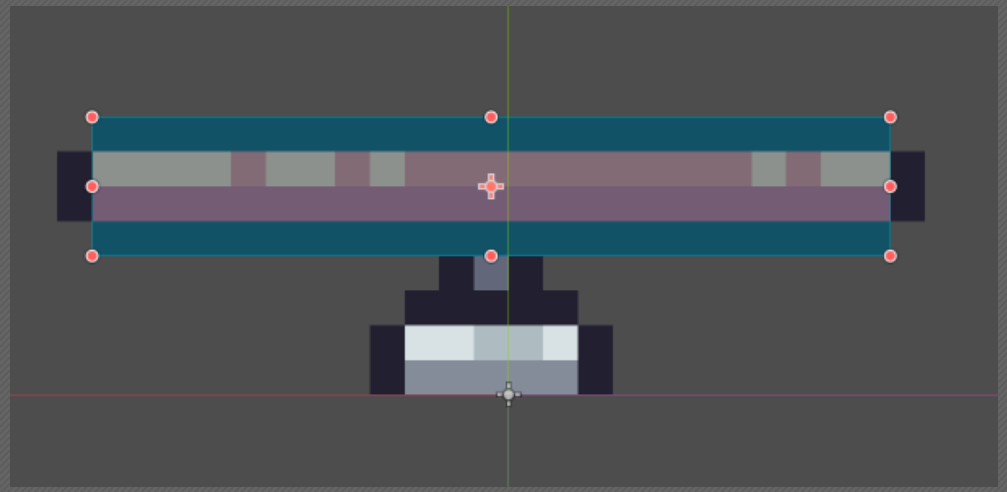


Create a Shape for the CollisionShape2D



- Create a RectangleShape2D shape for the CollisionShape2D node's Shape property.

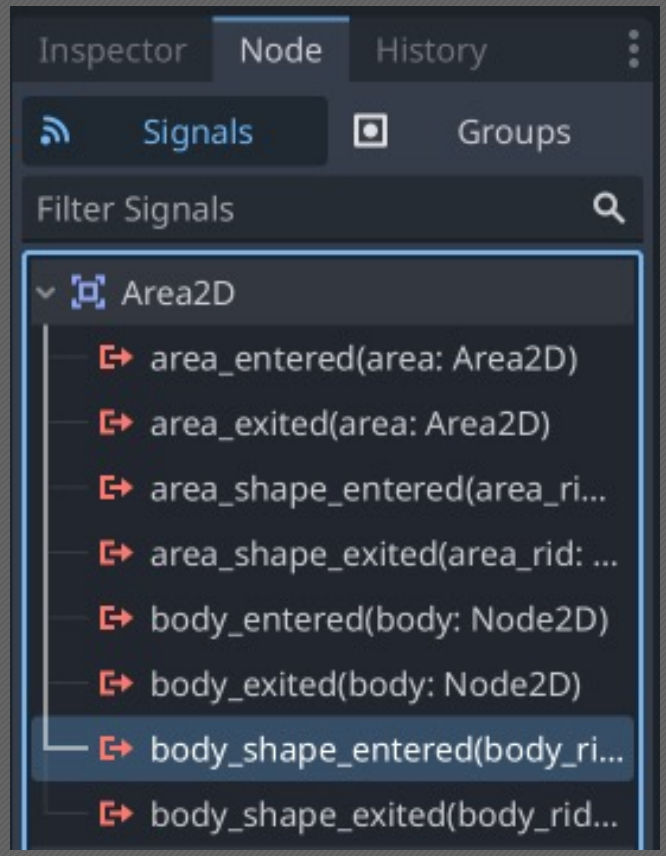
- Give the RectangleShape2D a size and position that encompasses the size of the trampoline pad/platform.



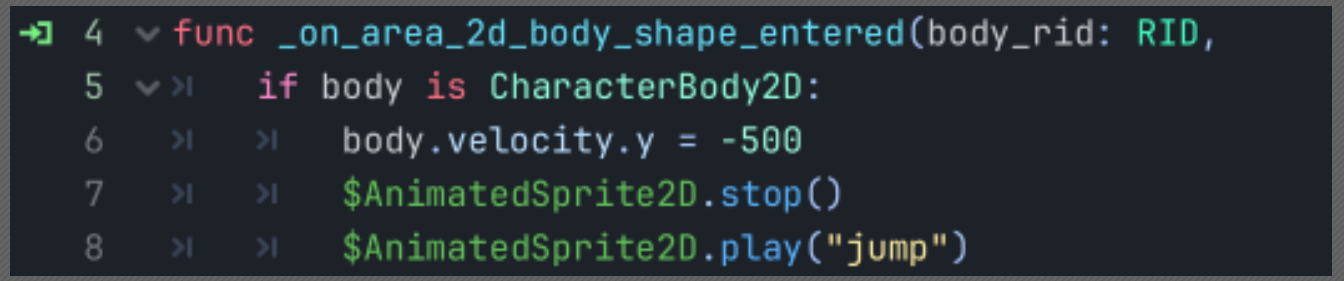
Connect the Area2D Signal

- Like was done to loop the background music for the Level, connect the Trampoline scene's Area2D body_shape_entered signal to the trampoline script.

- The default values provided in the signal connection prompt are fine.



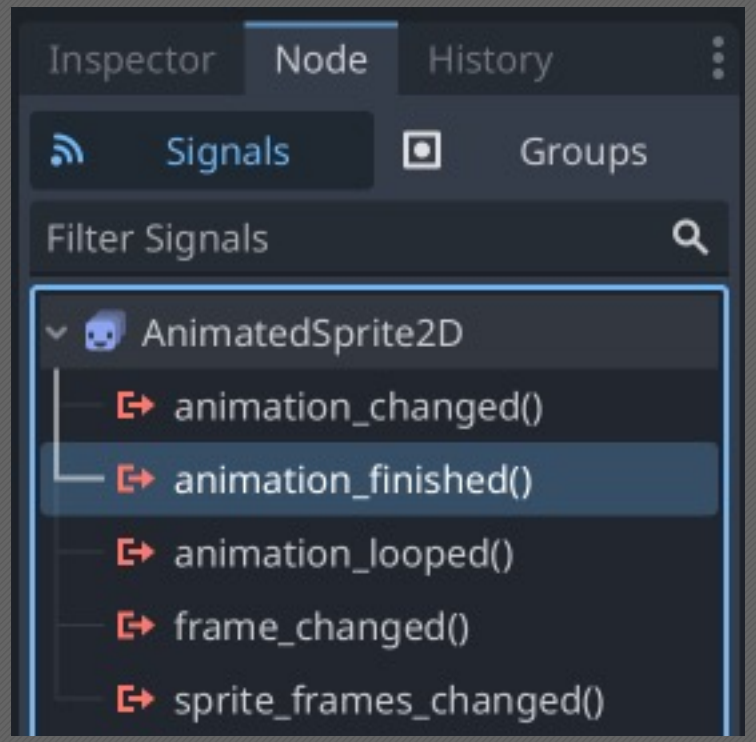
Add Code to Bounce a CharacterBody2D



```gdscript
func _on_area_2d_body_shape_entered(body_rid: RID,
    if body is CharacterBody2D:
        body.velocity.y = -500
        $AnimatedSprite2D.stop()
        $AnimatedSprite2D.play("jump")
```

- The body_shape_entered signal is emitted when the Area2D detects *any* collision shape enter the Area2D's collision shape.
- Type checking can be performed with the is keyword.
- Add code to modify the Y velocity of any detected CharacterBody2D and play the trampoline's "jump" animation.

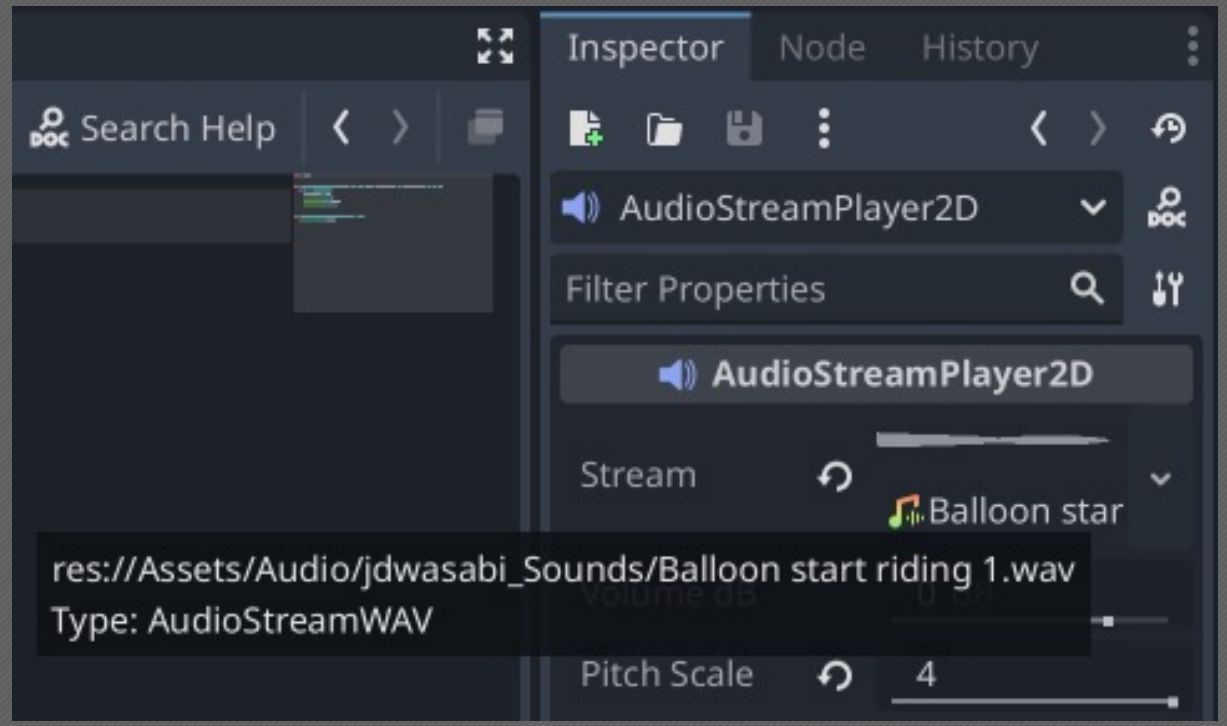
Reset after the “jump” Animation




```
11  func _on_animated_sprite_2d_animation_finished() -> void:
12      $AnimatedSprite2D.play("idle")
13
```

Add Trampoline Bounce Sound

- Add an AudioStreamPlayer2D node to the Trampoline scene.
- Assign the “Balloon start riding 1.wav” sound file to the Stream property.
- Set the Pitch Scale to 4. This will play the sound 4x as fast and at a higher pitch.



Play the Bounce Sound

Update the body_shape_entered handler to also play the new sound.

```
4  func _on_area_2d_body_shape_entered(body_rid: RID,
5      if body is CharacterBody2D:
6          body.velocity.y = -500
7          $AnimatedSprite2D.stop()
8          $AnimatedSprite2D.play("jump")
9          $AudioStreamPlayer2D.play()
```

Adding Configurable Bounce Strength

Exported script variables are editable in the Inspector tab.

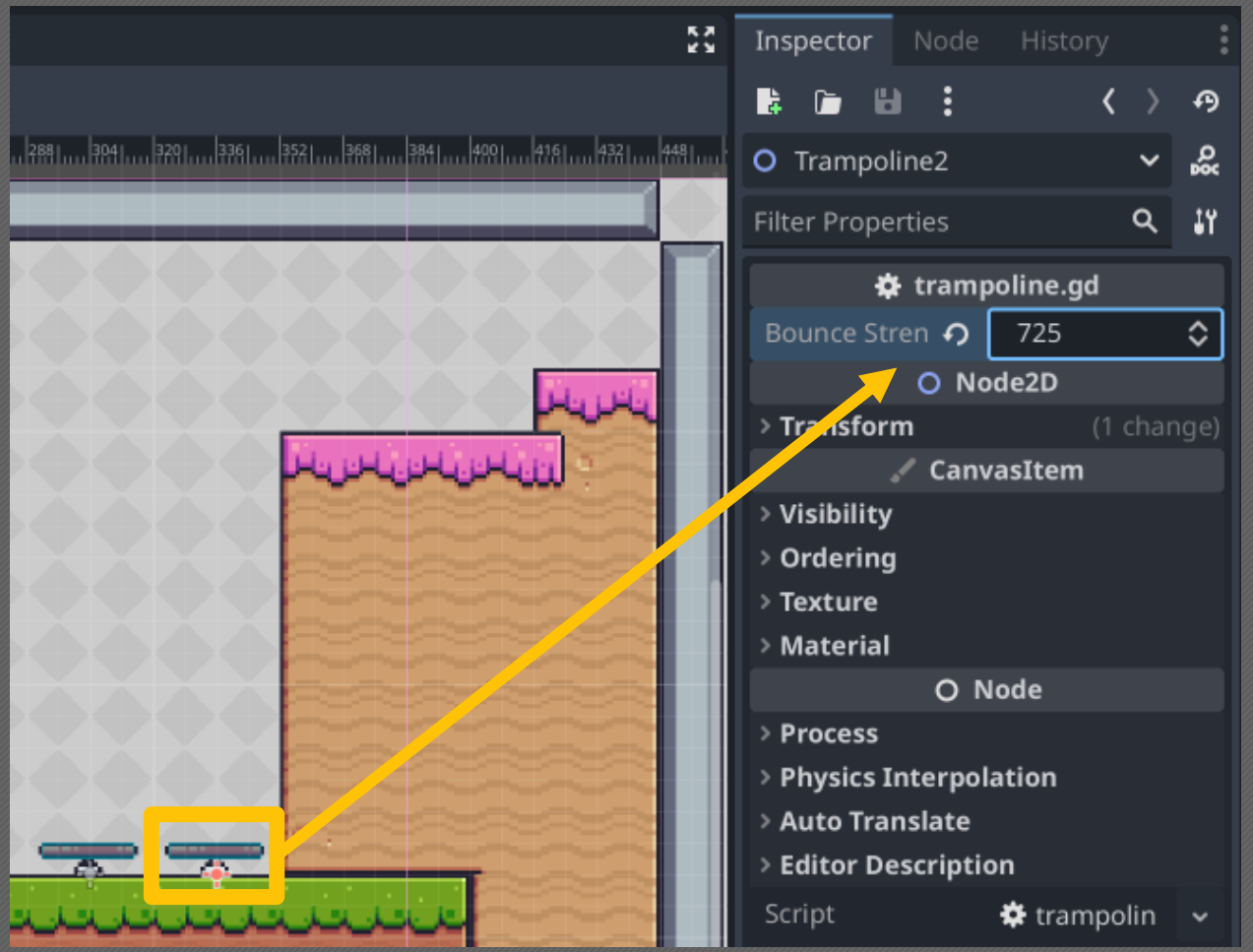
Script variables can be exported via the `@export` annotation.

- Export a bounce_strength var.
- Update the body Y velocity using the new bounce_strength variable.

```
3   @export var bounce_strength := 500
4
5   func _on_area_2d_body_shape_entered(body_rid: RID,
6       if body is CharacterBody2D:
7           body.velocity.y = -bounce_strength
8           $AnimatedSprite2D.stop()
9           $AnimatedSprite2D.play("jump")
10          $AudioStreamPlayer2D.play()
```

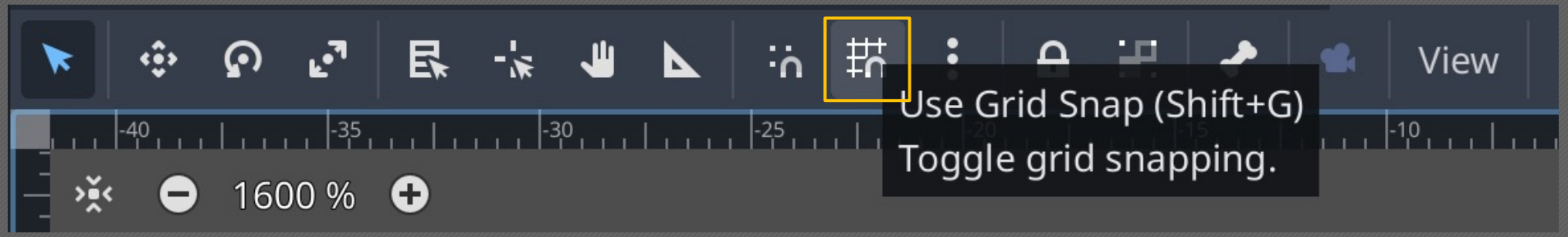
Add Trampolines to the Scene

- Save the Trampoline scene.
- Add some Trampolines to the Level scene.
- Notice that the Bounce Strength of each trampoline can be set independently of each other via the Inspector tab.

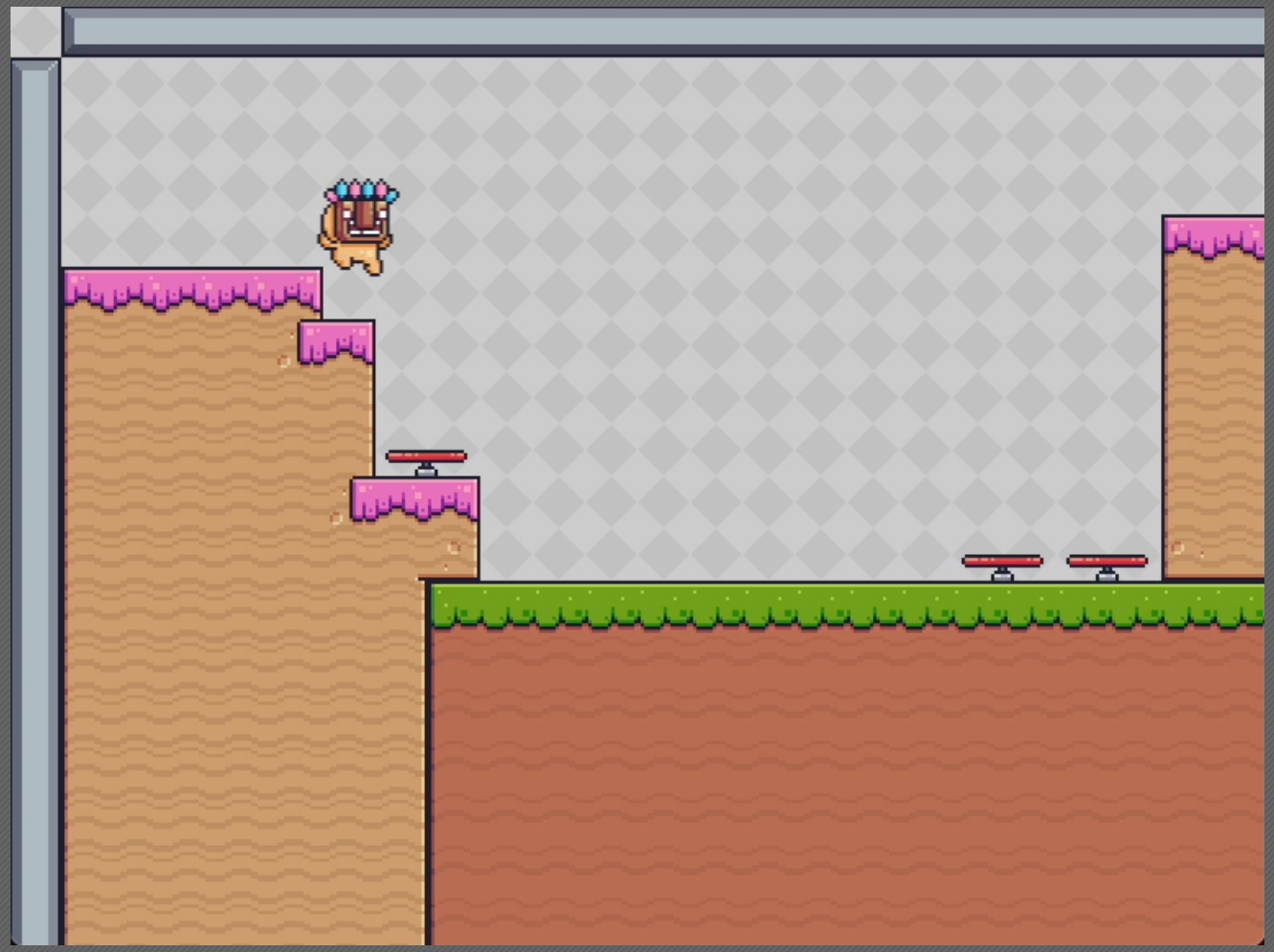


Scene View Grid Snapping

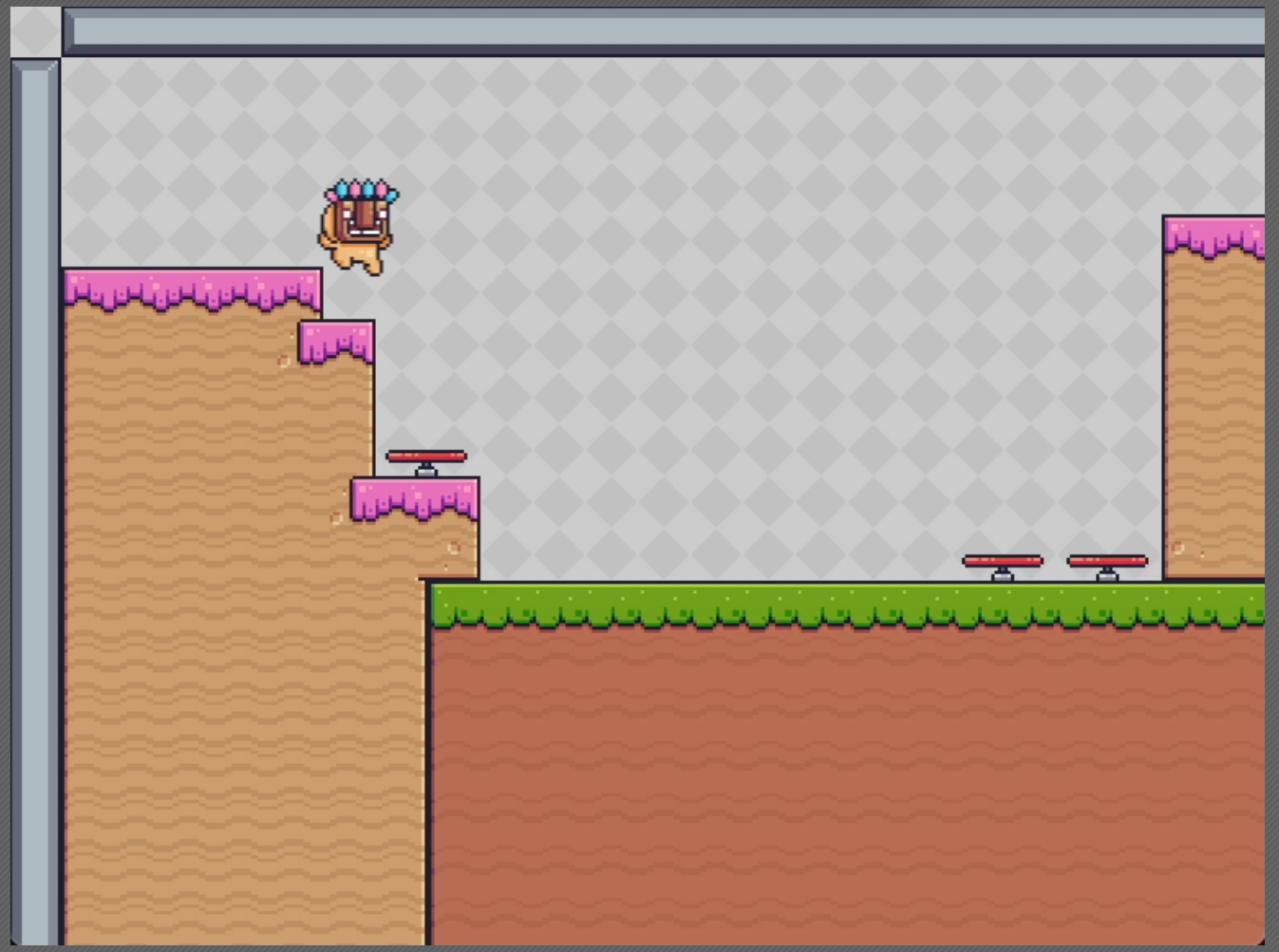
By default, Grid Snapping is not turned on in the 2D scene view. This feature makes it much easier to place nodes in the scene view so that nodes are properly flushed with the ground or other parts of the level. Turn this on to make it easier to align the tramplolines.



The Finished Trampoline

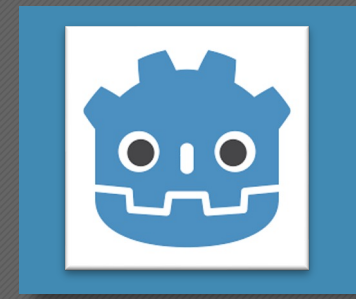


Lab Time (~10 Minutes)

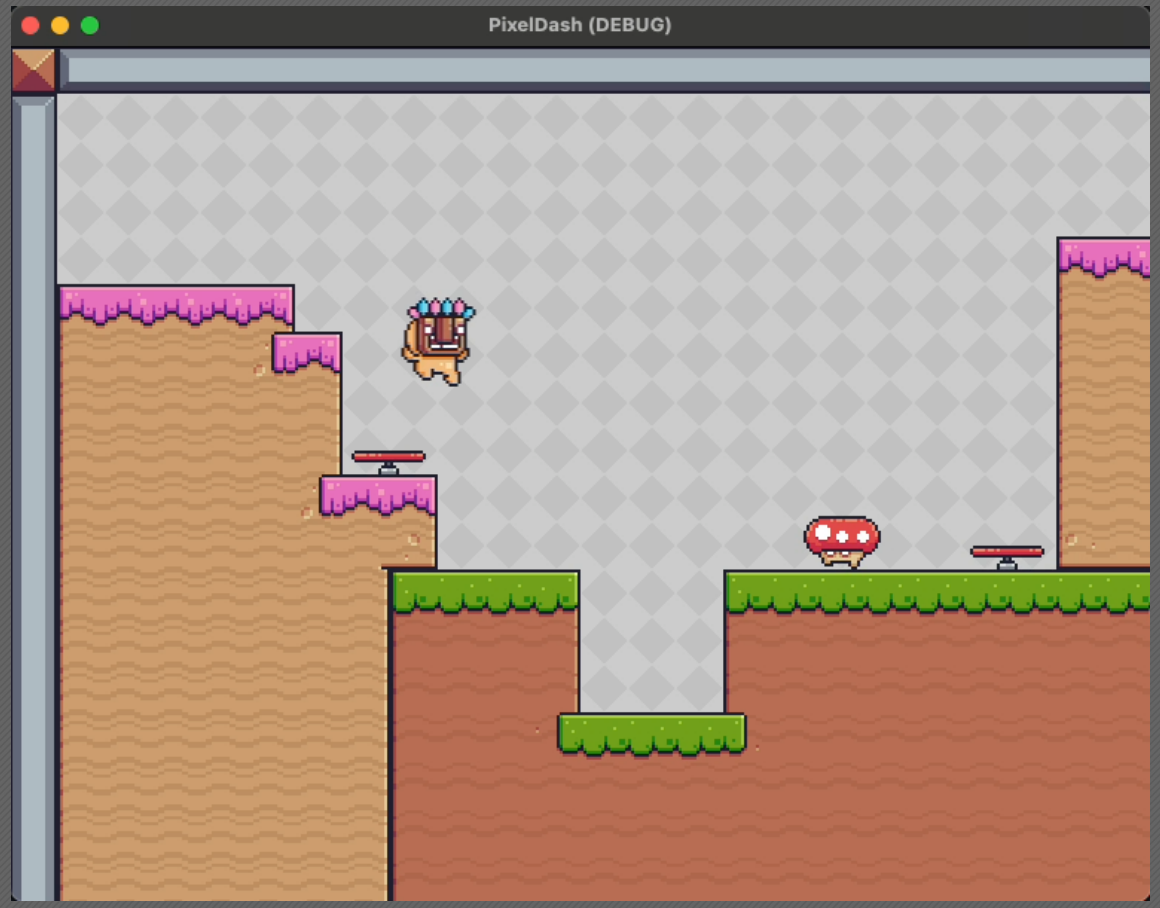


- Create a new Trampoline scene and attach a trampoline script to it.
- Attach the Trampoline's Area2D body_shape_entered signal to the Trampoline script to bounce detected CharacterBody2Ds.
- Add some Trampolines into the level, each with different Bounce Strengths.

Workshop Goal #5

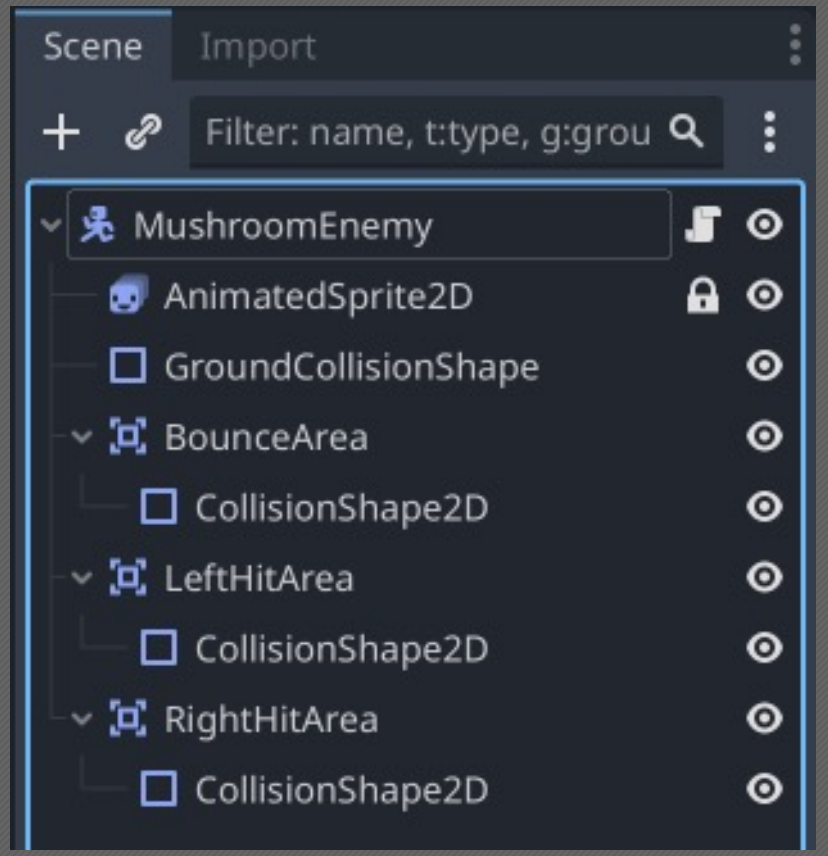


1. ✅ Control & Animate a Character
2. ✅ Build a Level with Tiles
3. ✅ Add Player & Level Polish
   (Camera, Gravity, Sound, Background)
4. ✅ Detect Collisions with Trampolines
5. ➡️ Create a basic Mushroom Enemy
6. Add Collectible Fruit and HUD Display
7. Respawn Character when Defeated
8. Create a Main Menu & Change Scenes
9. Open Workshop, Tinker Time



Create a new “MushroomEnemy” Scene

- Create a new scene with a root node of CharacterBody2D named MushroomEnemy and attach a new script to it.
- Mimic the scene structure to the right.
- The “CollisionShape” named nodes are of type CollisionShape2D.
- The “Area” named nodes are Area2D.



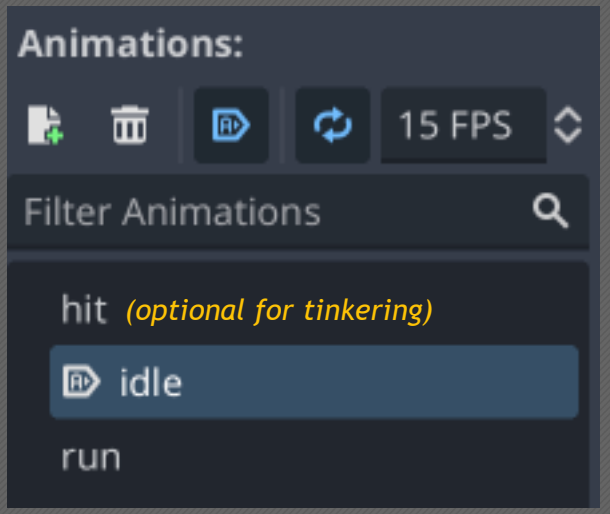
Animate the MushroomEnemy

- Add animations using the sprite sheet files under: Assets/Art/Pixel Adventure 2/Enemies/Mushroom

- All animations should be 15 FPS.

- The idle animation should set to loop and autoplay on load.

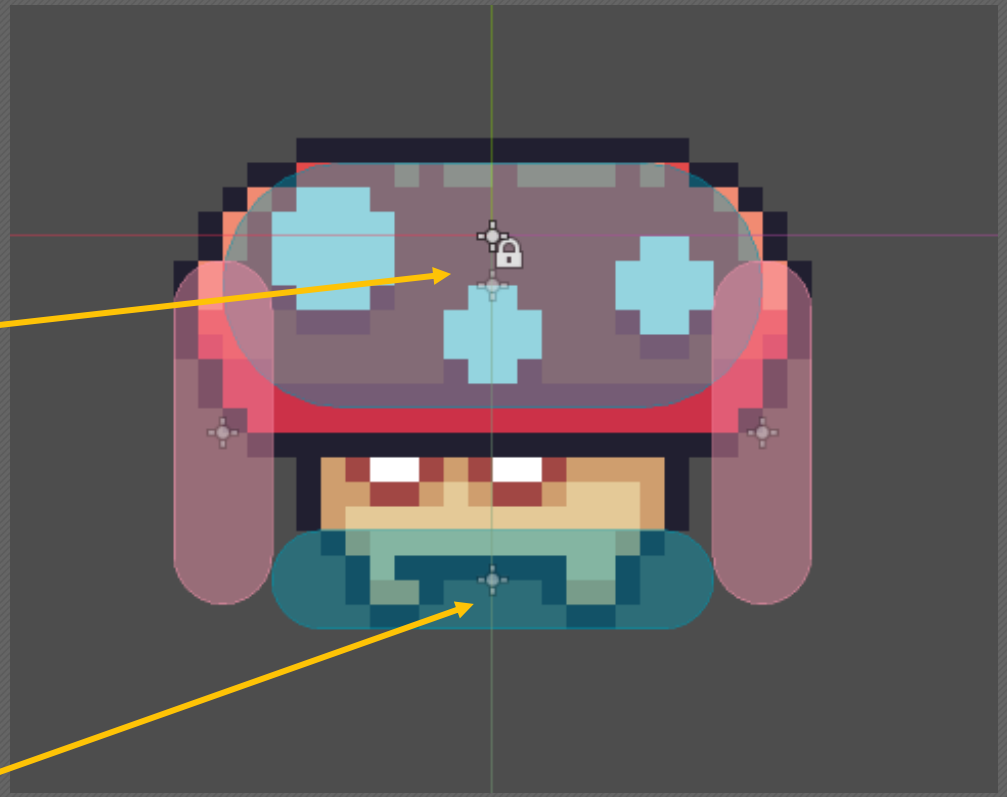
- The run animation should loop (no autoplay).



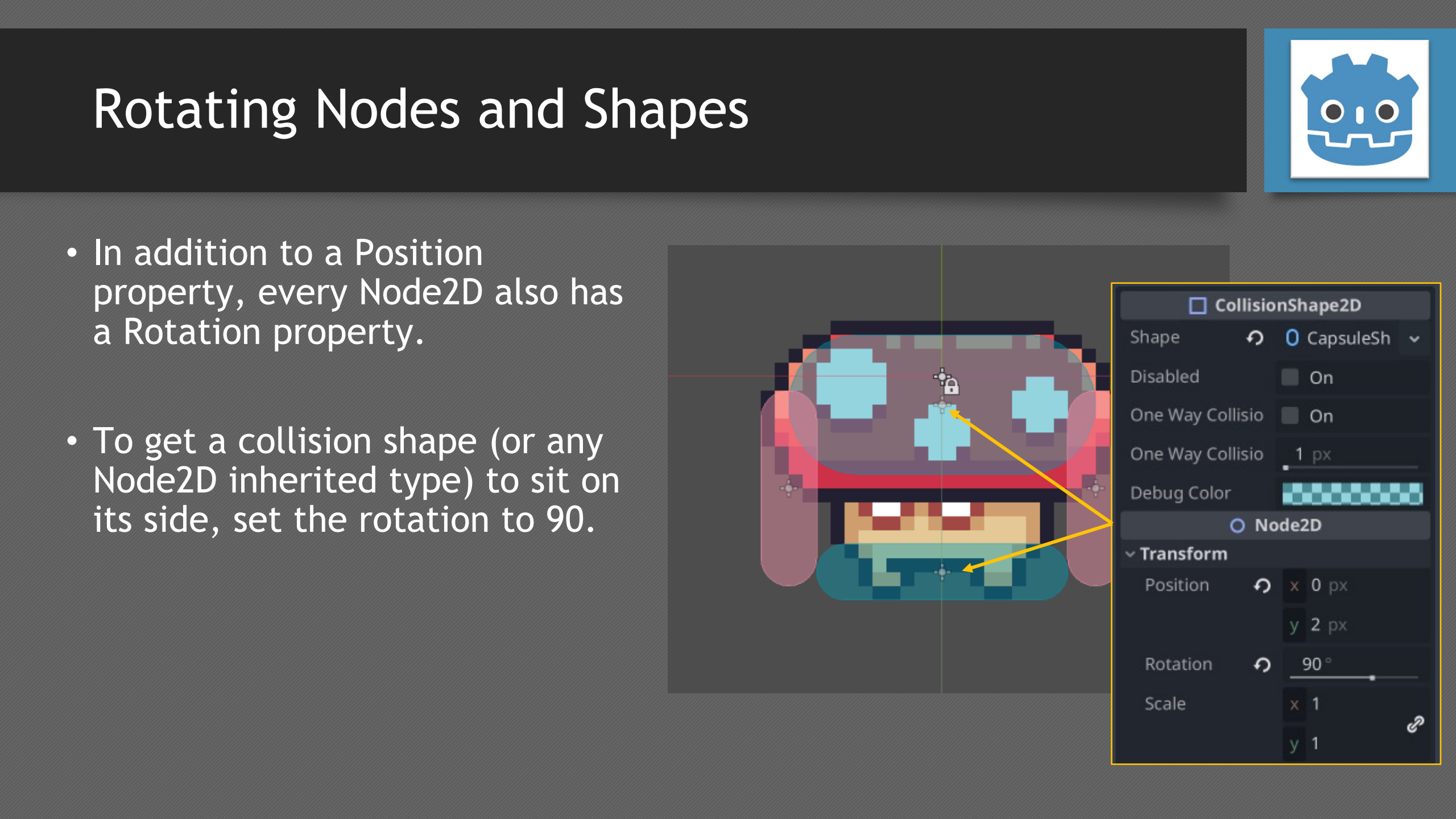
Set Shape Property on All Collision Shapes

The Mushroom enemy will have 4 distinct capsule collision shapes for providing different functionality:

- The BounceArea collision shape should encompass the hat/cap.
- The LeftHit and RightHit areas should be on the sides.
- The GroundCollisionShape is there for movement and ground collisions.



Rotating Nodes and Shapes



- In addition to a Position property, every Node2D also has a Rotation property.
- To get a collision shape (or any Node2D inherited type) to sit on its side, set the rotation to 90.

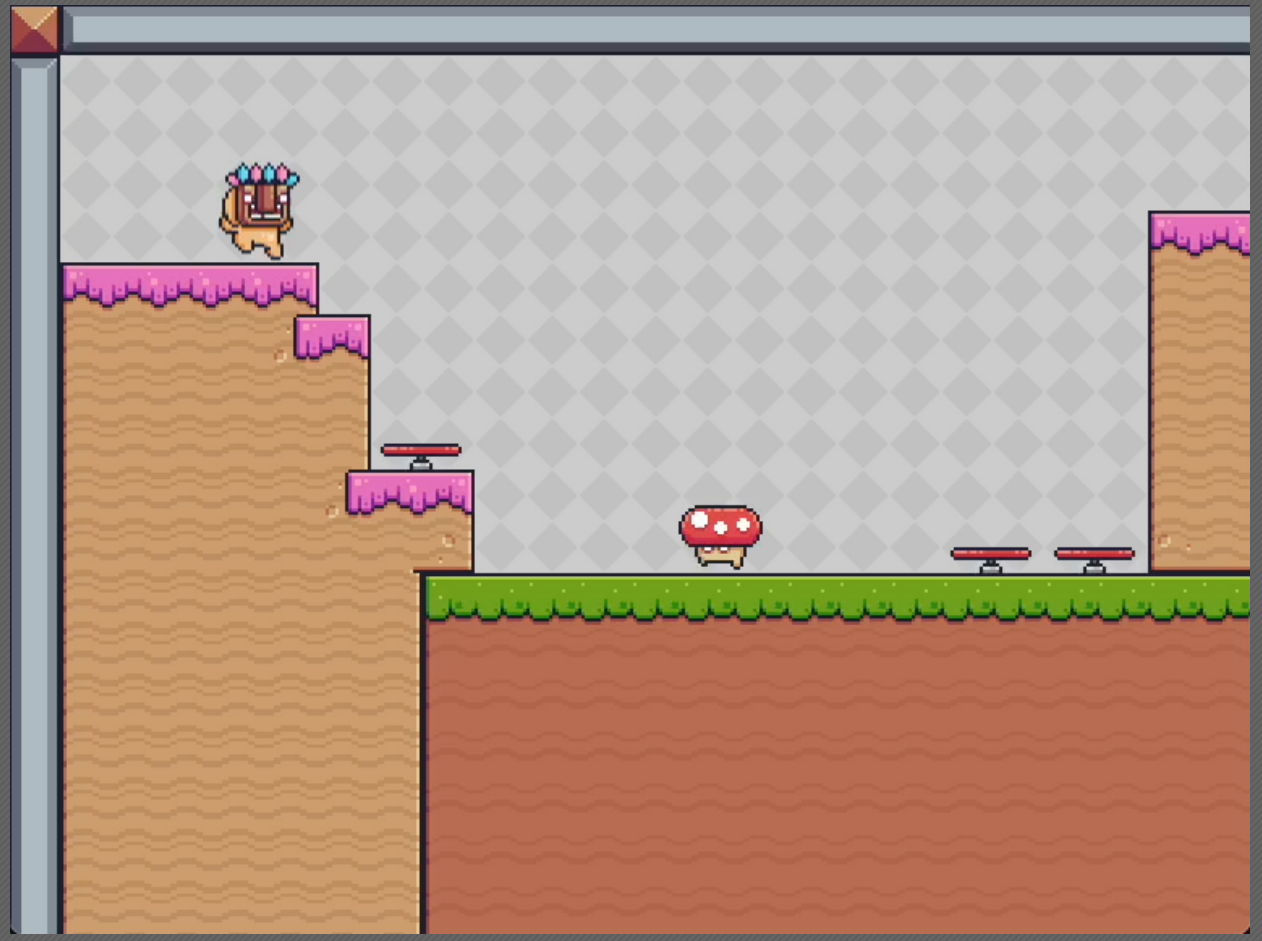
Code the MushroomyEnemy Script

- Inverse of the Player which moves based on Input, the MushroomEnemy will move on its own.

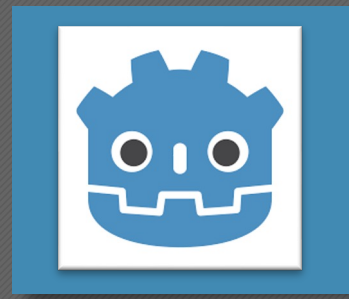
- Add this code to this script to move and animate the MushroomEnemy.

```
1   extends CharacterBody2D
2
3   const BASE_MOVE_SPEED := 35
4
5   var is_facing_left := true
6
7
8   func _ready() -> void:
9       velocity.x = -BASE_MOVE_SPEED
10
11  func _physics_process(delta: float) -> void:
12      move_and_slide()
13
14      if velocity.x != 0:
15          $AnimatedSprite2D.play("run")
16      else:
17          $AnimatedSprite2D.play("idle")
```

The enemy moves, but isn't very smart...



Adding Turn-Around Logic



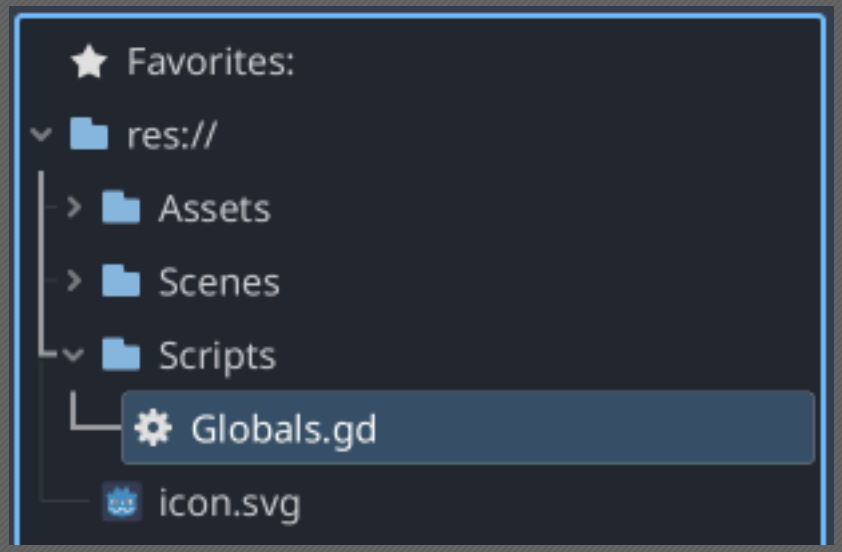
- Add the basic gravity code that was done for Player.
- Refactor movement out of the _ready func. At the start of _physics_process, add a check to conditionally turn_around if is_on_wall is true.
- Update the X and Y velocities.
- Implement the turn_around func.

```
1   extends CharacterBody2D
2
3   const BASE_MOVE_SPEED := 35
4   const GRAVITY := 35
5
6   var is_facing_left := true
7
8   func _physics_process(delta: float) -> void:
9       if is_on_wall():
10          turn_around()
11
12      velocity.x = -BASE_MOVE_SPEED if is_facing_left else BASE_MOVE_SPEED
13      velocity.y += GRAVITY
14
15      move_and_slide()
16
17      if velocity.x != 0:
18          $AnimatedSprite2D.play("run")
19      else:
20          $AnimatedSprite2D.play("idle")
21
22  func turn_around() -> void:
23      $AnimatedSprite2D.flip_h = !$AnimatedSprite2D.flip_h
24      is_facing_left = !is_facing_left
```

Refactoring for Reuse/Redundancies

This GRAVITY const has been implemented a couple of times now.
A Globals script can be created to store these variables for consistent values across the whole game.

Since it's a script that isn't attached to a node in the scene tree, it can instead be accessed by using the class_name keyword which gives the script a "proper class identification".




```
1   class_name Globals
2
3   const GRAVITY := 35
4
```

Refactoring for Reuse/Redundancies

Update the Player and MushroomEnemy scripts to reference the Globals class for the GRAVITY var.

```
3   const BASE_MOVE_SPEED := 35
4
5   var is_facing_left := true
6
7   func _physics_process(delta: float) -> void:
8       if is_on_wall():
9           turn_around()
10
11      velocity.x = -BASE_MOVE_SPEED if is_facing_left else BASE_MOVE_SPEED
12      velocity.y += Globals.GRAVITY
13
14      move_and_slide()
```

MushroomEnemy script

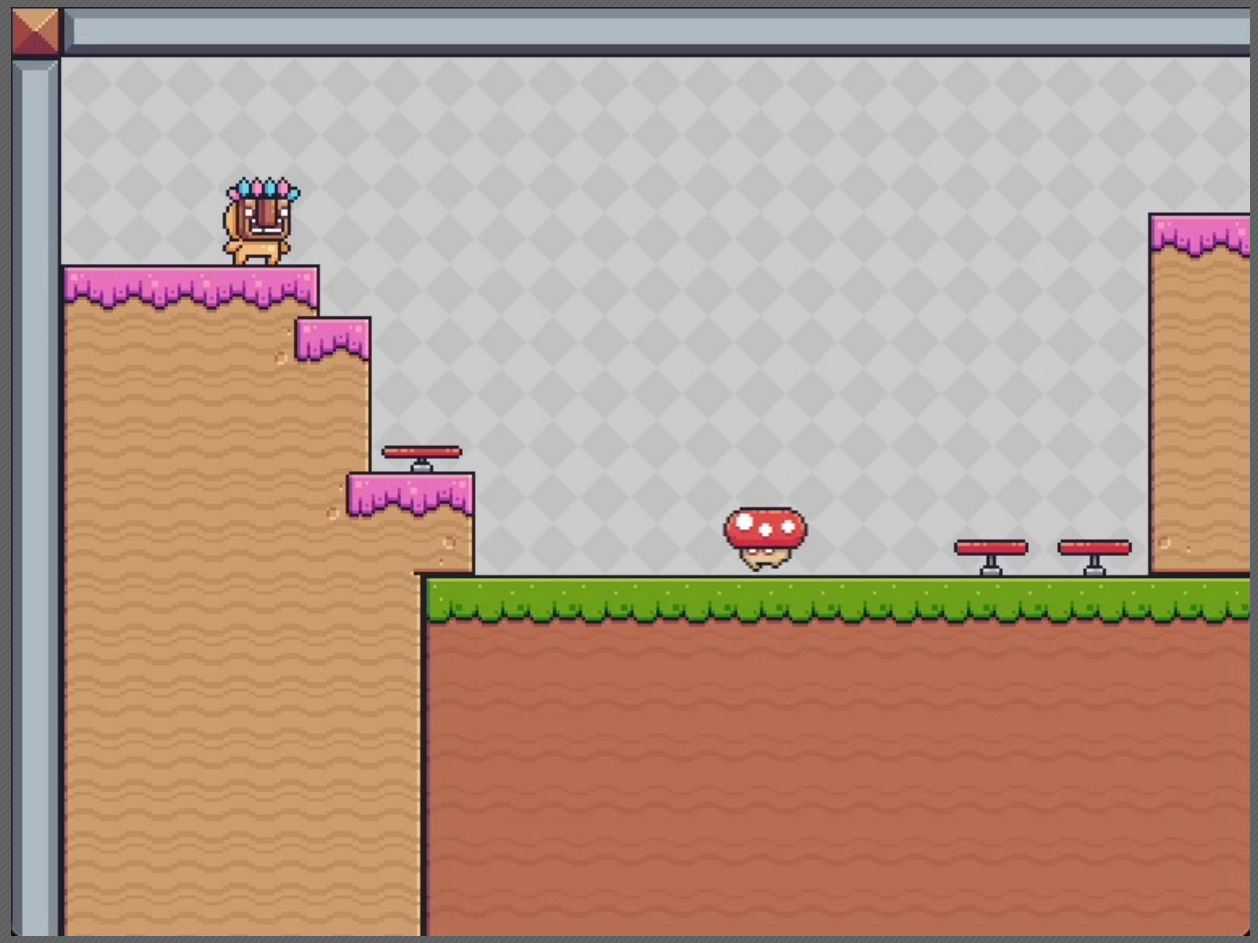
```
3   const BASE_MOVE_SPEED := 250
4   const JUMP_FORCE := Globals.GRAVITY * 14
5
6   func _physics_process(delta: float) -> void:
7       # Shorthand for getting the right (positive) input minus th
8       var input_vector := Input.get_axis("left", "right")
9
10      # The input vector returns 0 for "not pressed" and 1 for "p
11      velocity.x = input_vector * BASE_MOVE_SPEED
12      velocity.y += Globals.GRAVITY
```

Player script

Better, but it still has some issues.

The Player can get stuck inside of the enemy's left/right collision shapes, resulting in being dragged by the enemy's movement.

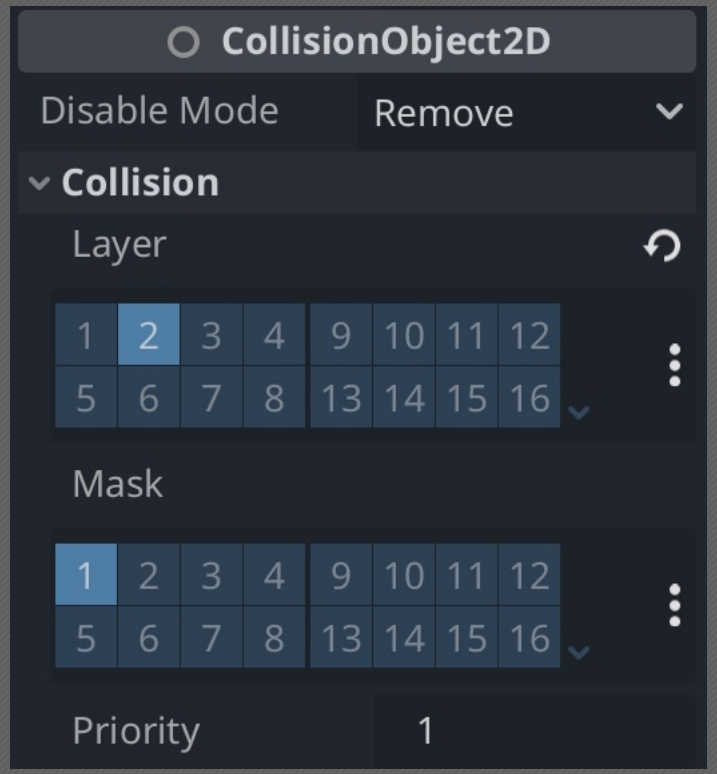
The enemy can bounce on the Trampolines and turns around when it collides with the Player, neither of which are desirable behaviors for the purpose of this workshop.



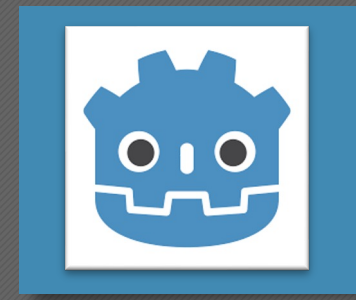
Fixing MushroomEnemy Collision & Movement

- The MushroomEnemy and the Player collisions are both Layer 1 and Mask 1, resulting in them colliding with each other and the Trampolines.

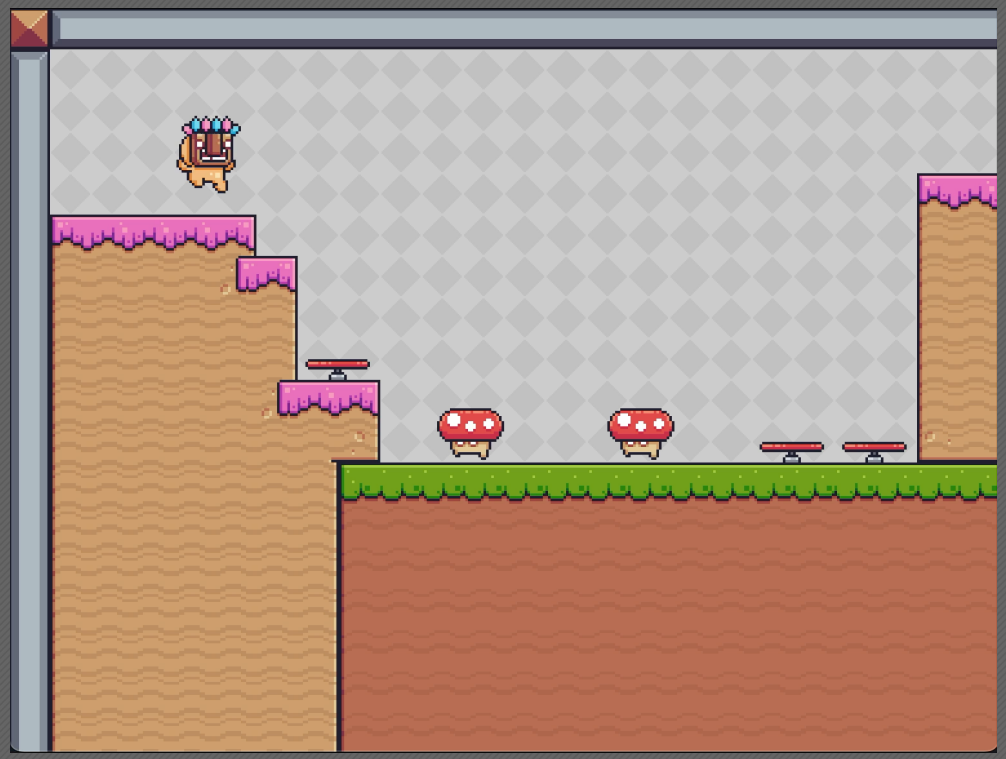
- Update the MushroomEnemy root node to have Collision Layer 1 off and Layer 2 on.



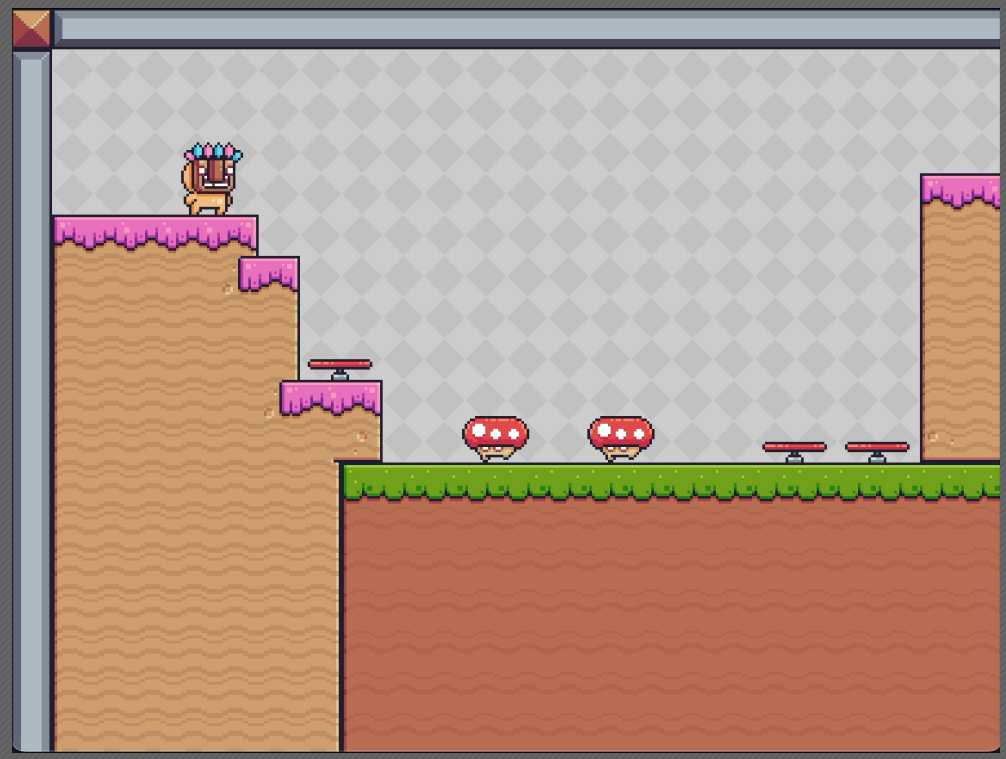
Fixing MushroomEnemy Collision & Movement



With only Layer 1 on



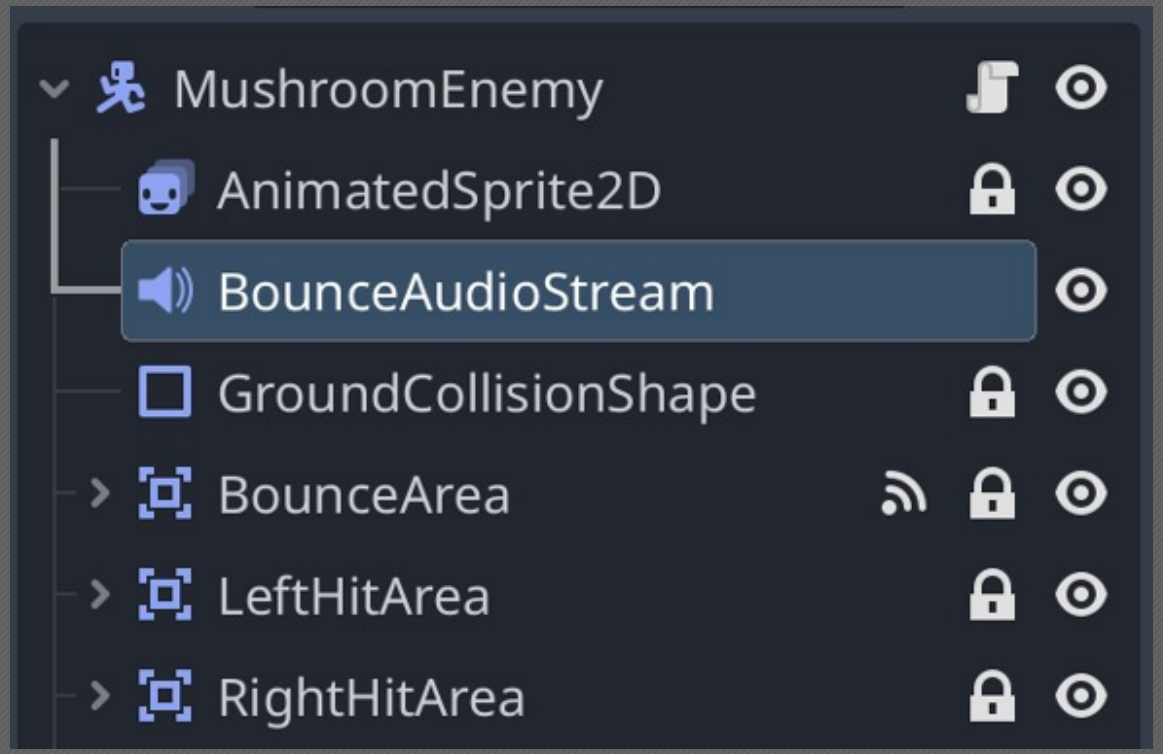
With only Layer 2 on



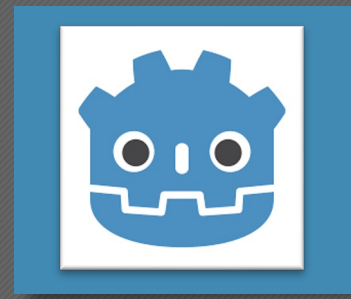
Adding Player Bounce to MushroomEnemy

Before wiring the signals and adding code, add an AudioStream2D to the MushroomEnemy scene tree.

Name the new node BounceAudioStream.

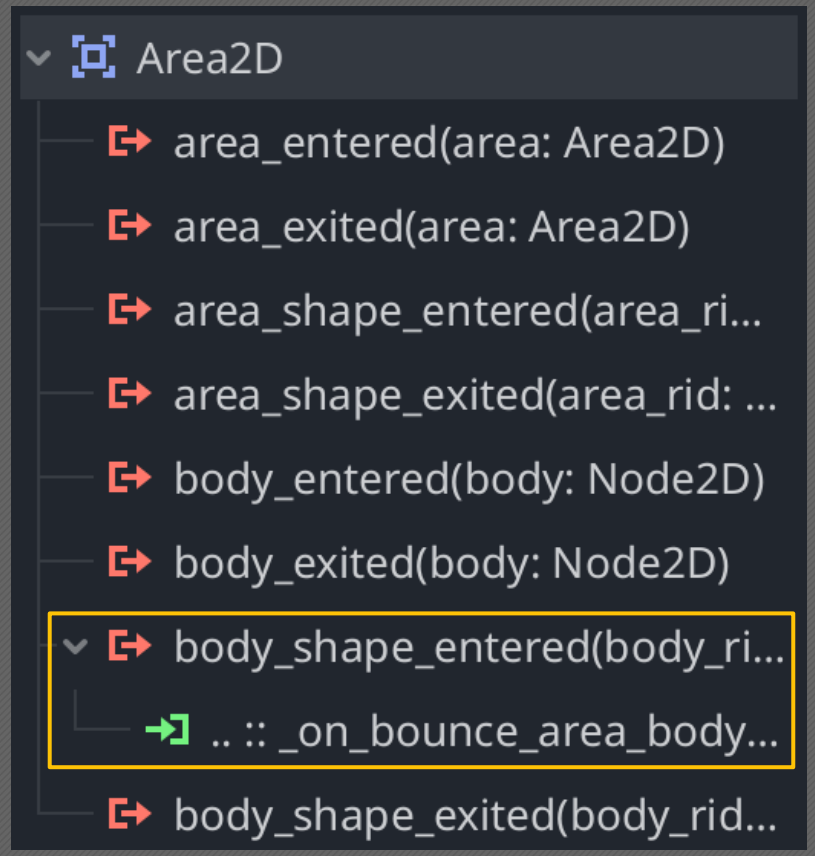


Adding Player Bounce to MushroomEnemy



The MushroomEnemy has various Area2Ds for which collisions can happen. The BounceArea is the Area2D that will provide this behavior.

- In the MushroomEnemy script, connect the BounceArea body_shape_entered signal to the MushroomEnemy script.



Adding Player Bounce to MushroomEnemy

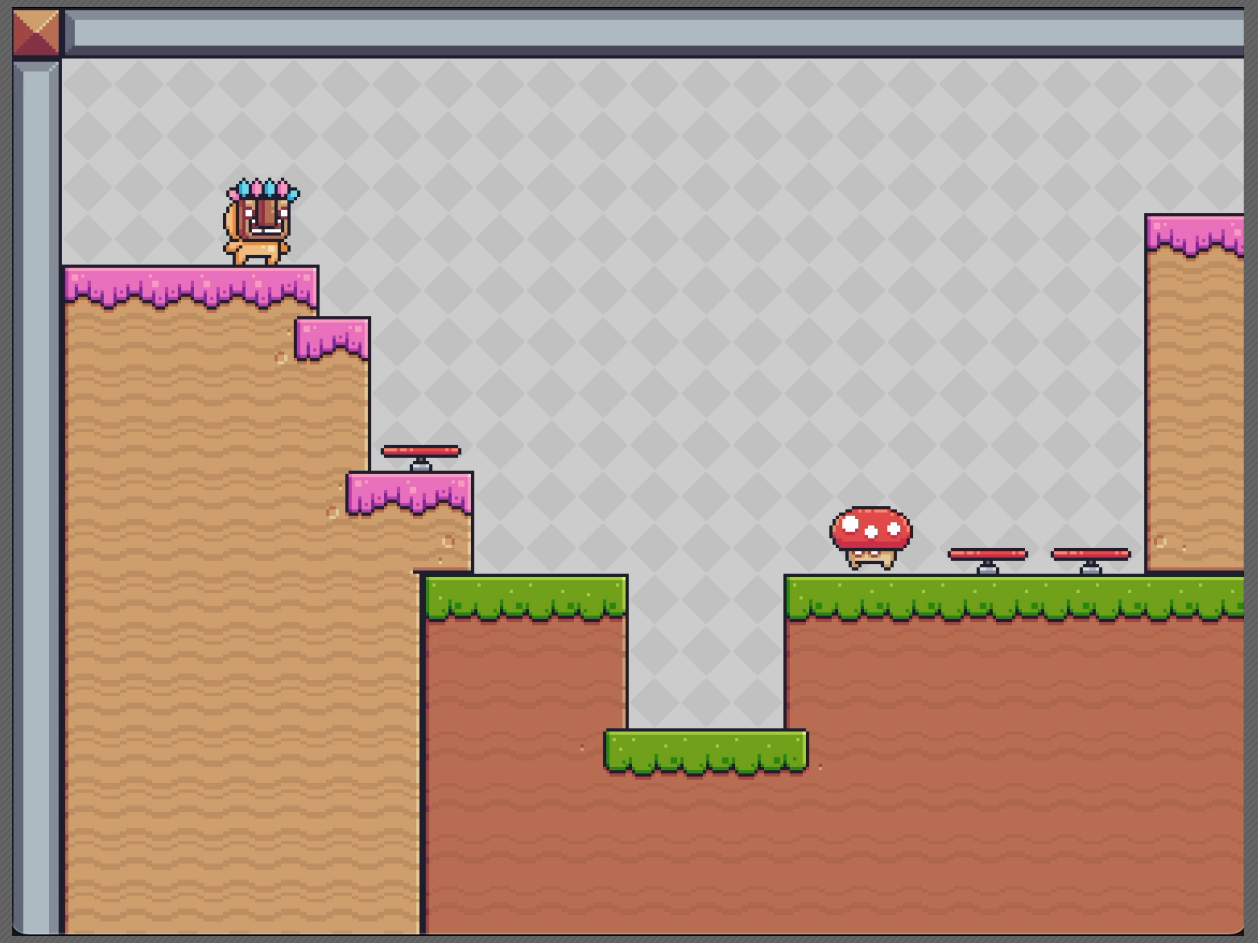
- The MushroomEnemy's bounce behavior is essentially the same thing as the Trampoline behavior.

- In the generated func, add the same kind of type-check implementation that was done for the Trampoline logic.

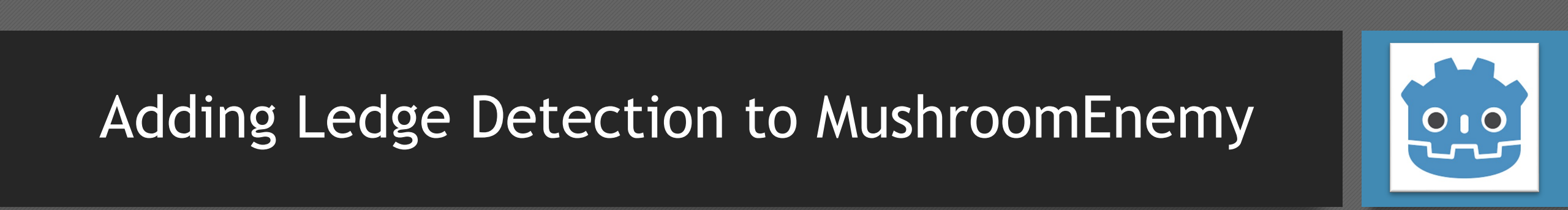
Don't forget to create the BOUNCE_STRENGTH const.

```
26  func _on_bounce_area_body_shape_entered(body_rid: RID,
27      if body is CharacterBody2D:
28          $BounceAudioStream.play()
29          body.velocity.y = -BOUNCE_STRENGTH
```

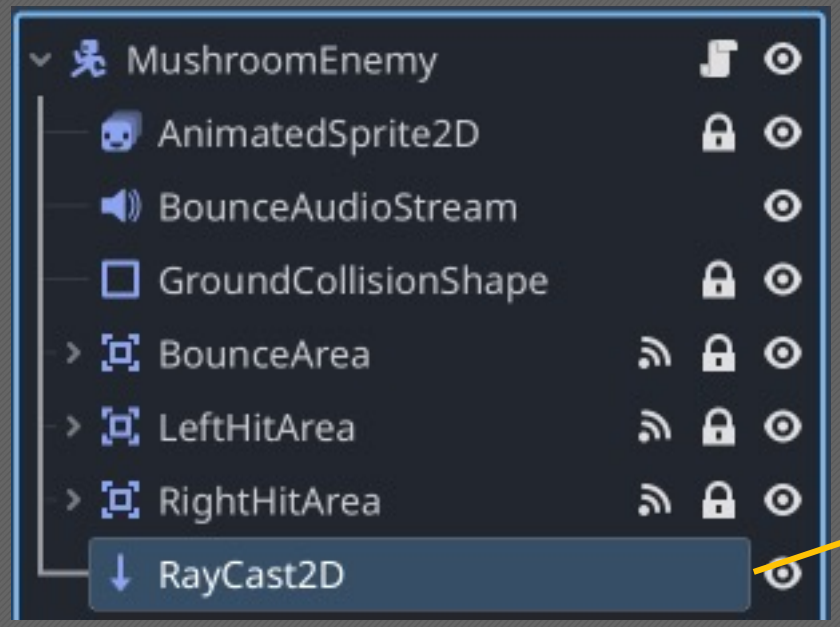
Still not very smart when it comes to ledges...

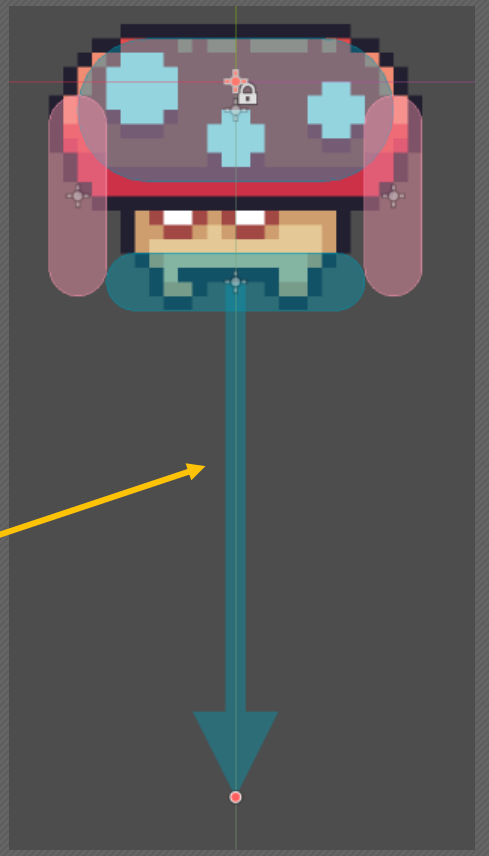


Adding Ledge Detection to MushroomEnemy



Add a RayCast2D to the MushroomEnemy





Adding Ledge Detection to MushroomEnemy

A RayCast2D is a 2-point line that performs collision detection along its length. A RayCast has a collision layer and mask.

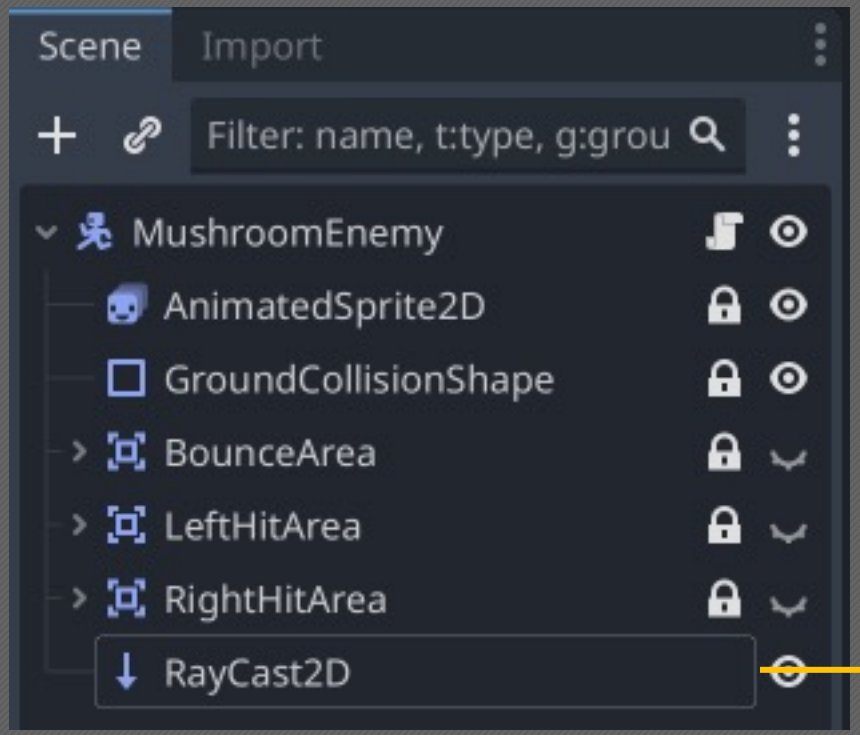
The enemy should turn around when the RayCast does NOT detect a collision (aka – there's no ground in front of the enemy).

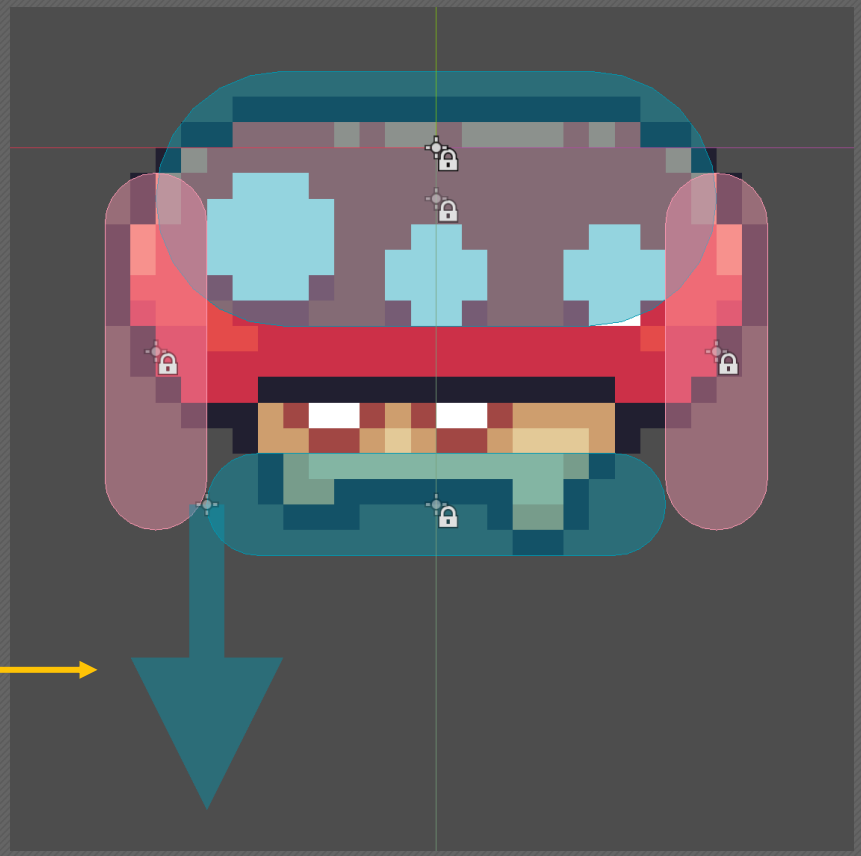
```
8  func _physics_process(delta: float) -> void:
9      if is_on_wall() or not $RayCast2D.is_colliding():
10         turn_around()
```

Update the MushroomEnemy script logic to check if the RayCast2D is not colliding.

Adding Ledge Detection to MushroomEnemy

Reposition the new RayCast2D to be a little bit in front of the GroundCollisionShape.

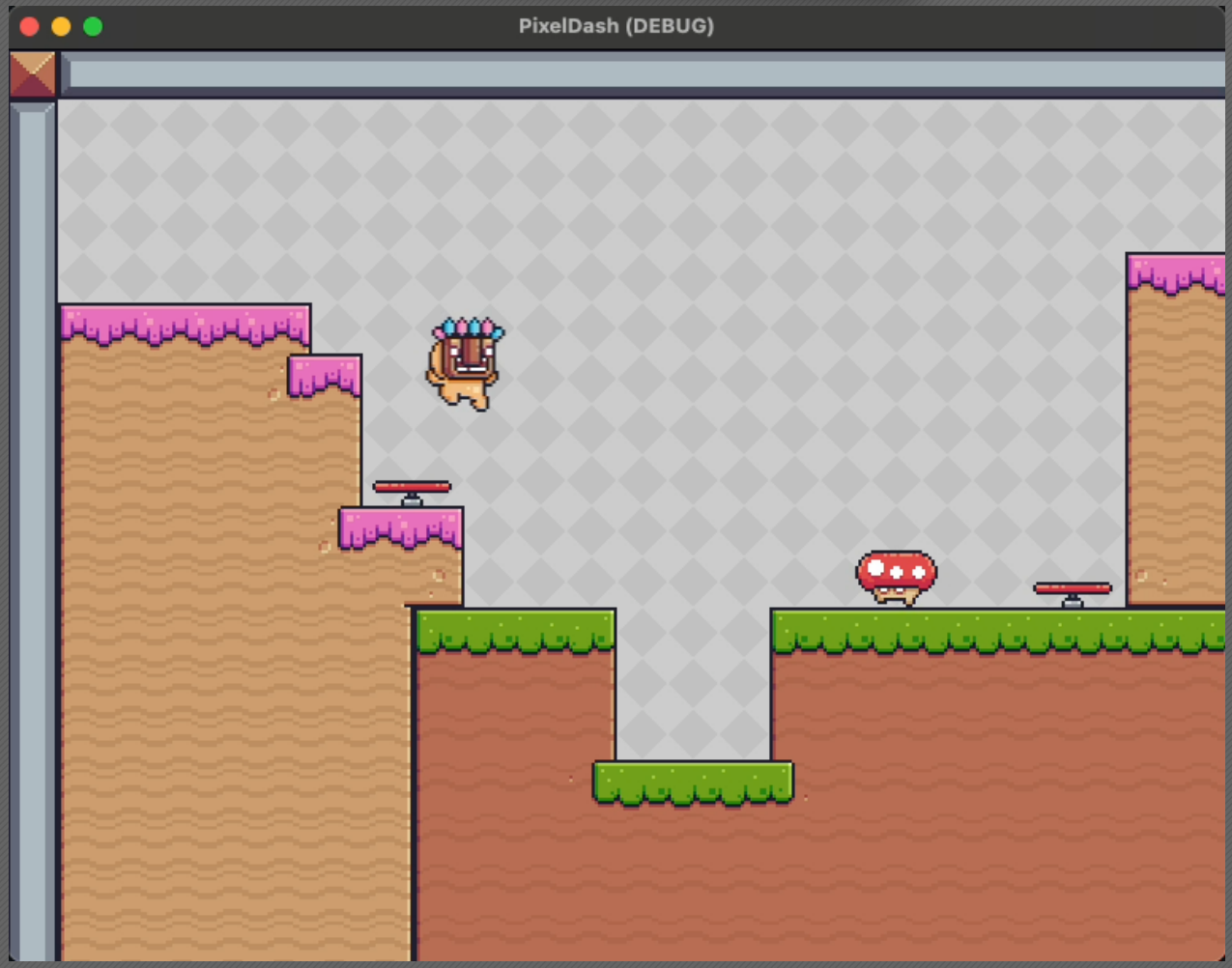




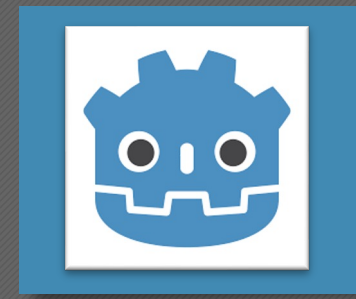
Tip: if locking nodes isn't useful, nodes can also be hidden (but remember to turn them on before running!)

Lab Time (~15 Minutes)

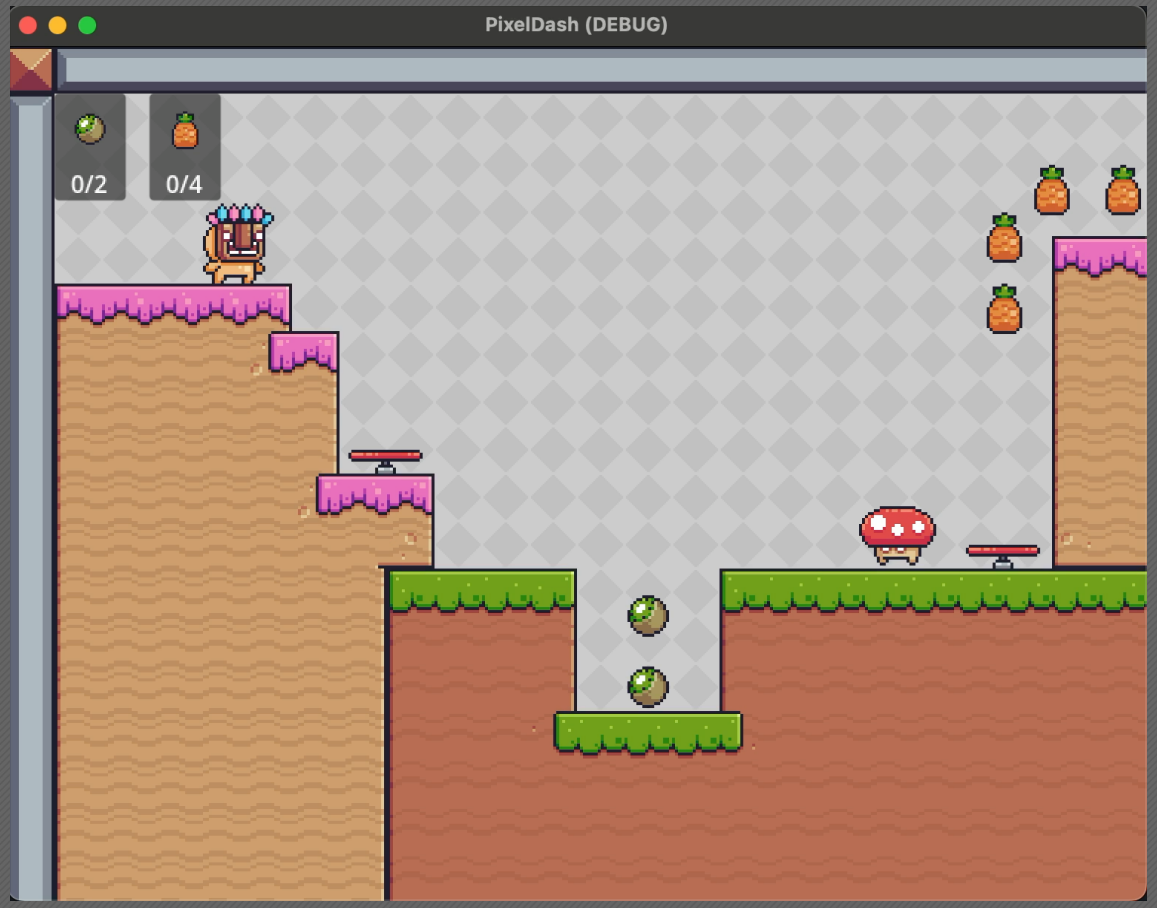
- Create a new MushroomEnemy scene, with a root node of CharacterBody2D.
- Animate the enemy and add its areas and collision shapes.
- Implement the code to move the enemy and turn around on collision walls and near ledges.
- Make the Player bounce on the enemy.



Workshop Goal #6

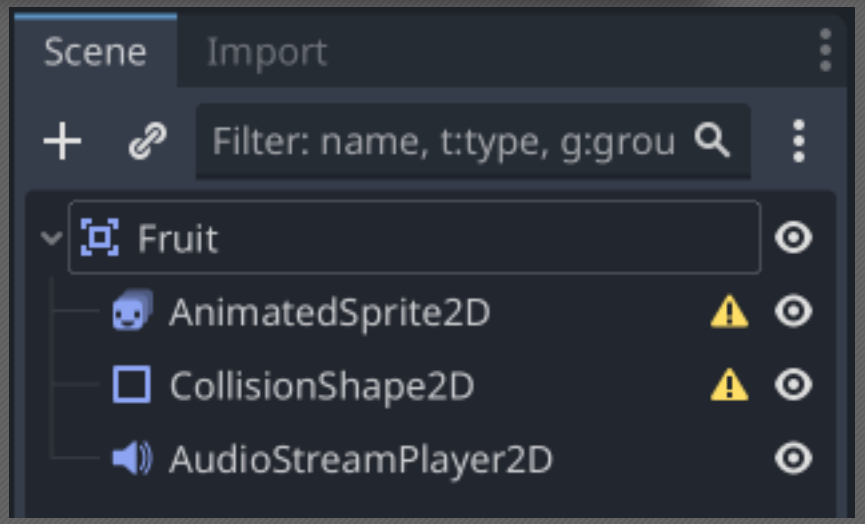


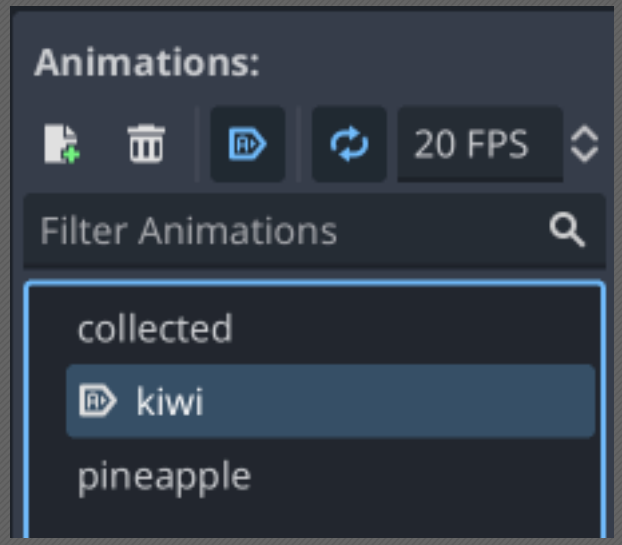
1. ✅ Control & Animate a Character
2. ✅ Build a Level with Tiles
3. ✅ Add Player & Level Polish
   (Camera, Gravity, Sound, Background)
4. ✅ Detect Collisions with Trampolines
5. ✅ Create a basic Mushroom Enemy
6. ➡️ Add Collectible Fruit and HUD Display
7. Respawn Character when Defeated
8. Create a Main Menu & Change Scenes
9. Open Workshop, Tinker Time

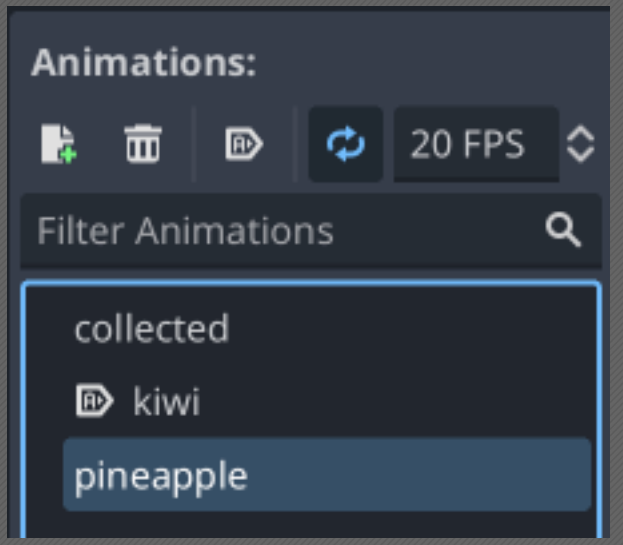


Create a Fruit Scene

- Create a new Area2D scene named Fruit with the scene tree structure shown right.
- Create two animations of your choice, using the fruit options under: Assets/Art/Pixel Adventure 1/Items/Fruit
- Create a third 20 FPS animation called “collected” using the sprite sheet of the same name.



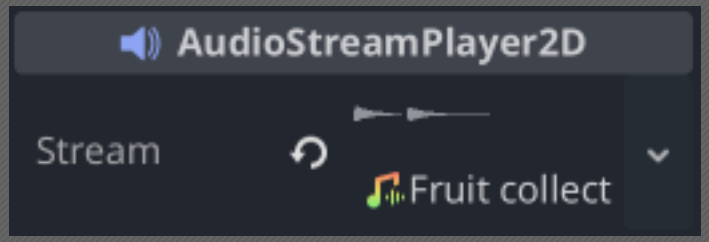




Give the Fruit a Shape and “Collected” Sound

The CollisionShape should Encompass the whole fruit.

Use the following file for the AudioStreamPlayer2D:
Assets/Audio/jdwasabi_Sounds/Fruit collect 1.wav



GDScript Export Var Options

See also PROPERTY_HINT_ENUM.

```
@export_enum("Warrior", "Magician", "Thief") var character_class: int
@export_enum("Slow:30", "Average:60", "Very Fast:200") var character_speed: int
@export_enum("Rebecca", "Mary", "Leah") var character_name: String

@export_enum("Sword", "Spear", "Mace") var character_items: Array[int]
@export_enum("double_jump", "climb", "dash") var character_skills: Array[String]
```

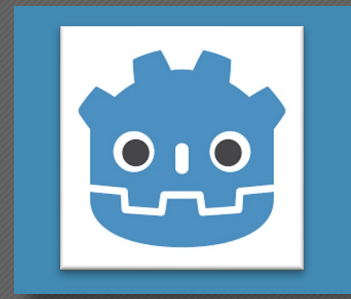
If you want to set an initial value, you must specify it explicitly:

```
@export_enum("Rebecca", "Mary", "Leah") var character_name: String = "Rebecca"
```

If you want to use named GDScript enums, then use `@export` instead:

```
enum CharacterName {REBECCA, MARY, LEAH}
@export var character_name: CharacterName

enum CharacterItem {SWORD, SPEAR, MACE}
@export var character_items: Array[CharacterItem]
```

GDScript Export Var Options



See also PROPERTY_HINT_ENUM.

```
@export_enum("Warrior", "Magician", "Thief") var character_class: int
@export_enum("Slow:30", "Average:60", "Very Fast:200") var character_speed: int
@export_enum("Rebecca", "Mary", "Leah") var character_name: String

@export_enum("Sword", "Spear", "Mace") var character_items: Array[int]
@export_enum("double_jump", "climb", "dash") var character_skills: Array[String]
```

If you want to set an initial value, you must specify it explicitly:

```
@export_enum("Rebecca", "Mary", "Leah") var character_name: String = "Rebecca"
```

If you want to use named GDScript enums, then use `@export` instead:

```
enum CharacterName {REBECCA, MARY, LEAH}
@export var character_name: CharacterName

enum CharacterItem {SWORD, SPEAR, MACE}
@export var character_items: Array[CharacterItem]
```

This is the style we will use for Fruit collectibles

Declaring Collectible Fruit Types

- Attach a script to the Fruit scene.
- Add an @onready var to create a reference to the animated sprite node.

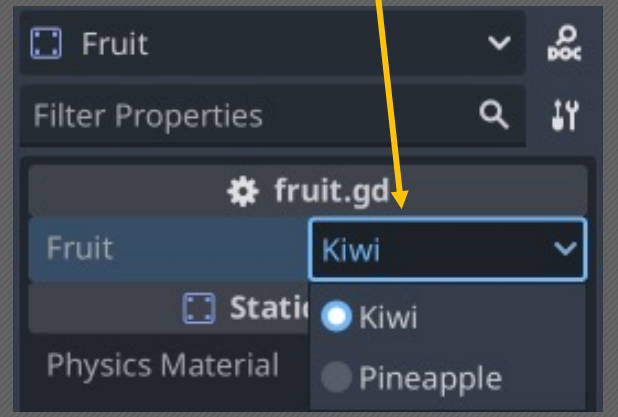
```
1   extends Area2D
2
3   @onready var ANIMATED_SPRITE := $AnimatedSprite2D
4   @export_enum("Kiwi", "Pineapple") var fruit := "Kiwi"
5
6   func _ready() -> void:
7       ANIMATED_SPRITE.play(fruit.to_lower())
8
```

The @onready var can be used as shorthand for initializations that you might do in _ready. All @onready vars are initialized *just before* _ready is invoked.

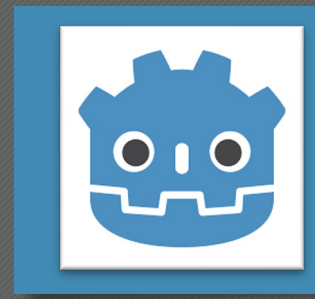
Declaring Collectible Fruit Types

- Create an export_enum var for the types of fruit added to the animated sprite.
- This exported enum var will be accessible as a list of options in the Inspector tab, limited to the list of declared enum options.

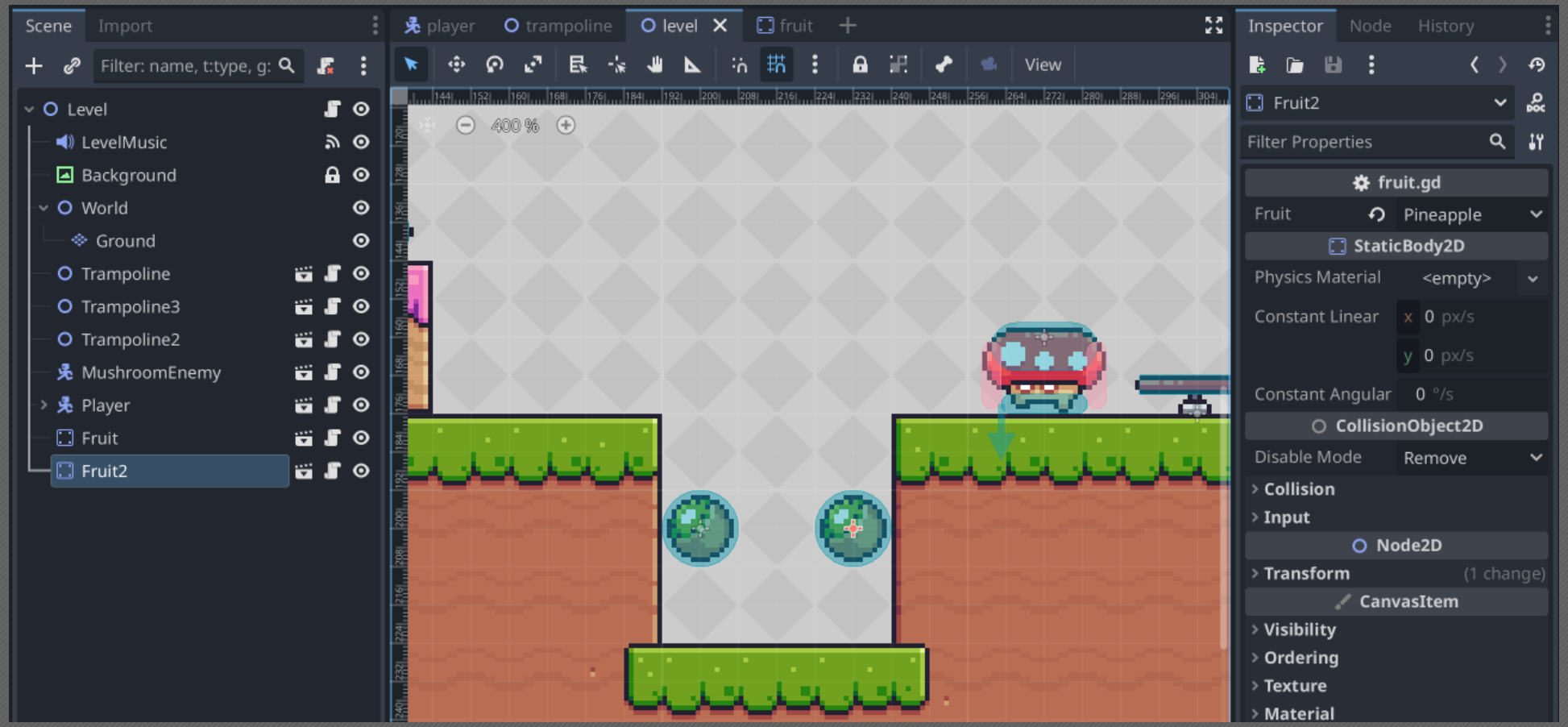
```
1   extends Area2D
2
3   @onready var ANIMATED_SPRITE := $AnimatedSprite2D
4   @export_enum("Kiwi", "Pineapple") var fruit := "Kiwi"
5
6   func _ready() -> void:
7       ANIMATED_SPRITE.play(fruit.to_lower())
8
```



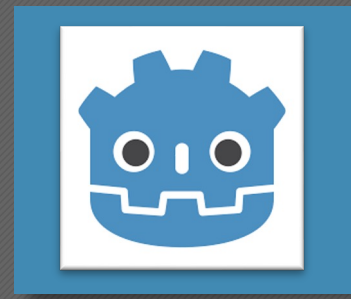
Only @tool Scripts run in the Editor



Updating the Fruit property value for an instantiated node will be correct in-game, but the Fruit does not update in the editor. This can be fixed by making the script a Tool.



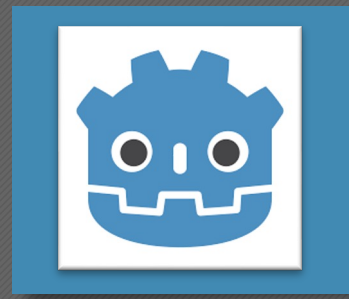
Declare the Fruit script to be a tool.



The `@tool` annotation allows the script to be loaded and executed by the editor. This can be dangerous, so be careful what scripts are made tools!

```
1   @tool
2   extends Area2D
3
4   @onready var ANIMATED_SPRITE := $AnimatedSprite2D
5   @export_enum("Kiwi", "Pineapple") var fruit := "Kiwi" : set = set_fruit
6
7
8   func set_fruit(new_value:String) -> void:
9       fruit = new_value
10      # Since ANIMATED_SPRITE is an onready var, it doesn't become available until just before _ready.
11      # Node values are set prior to the scene being ready, so trying to use it would produce Nil errors.
12      $AnimatedSprite2D.play(fruit.to_lower())
13
14
15  func _ready() -> void:
16      ANIMATED_SPRITE.play(fruit.to_lower())
17
```

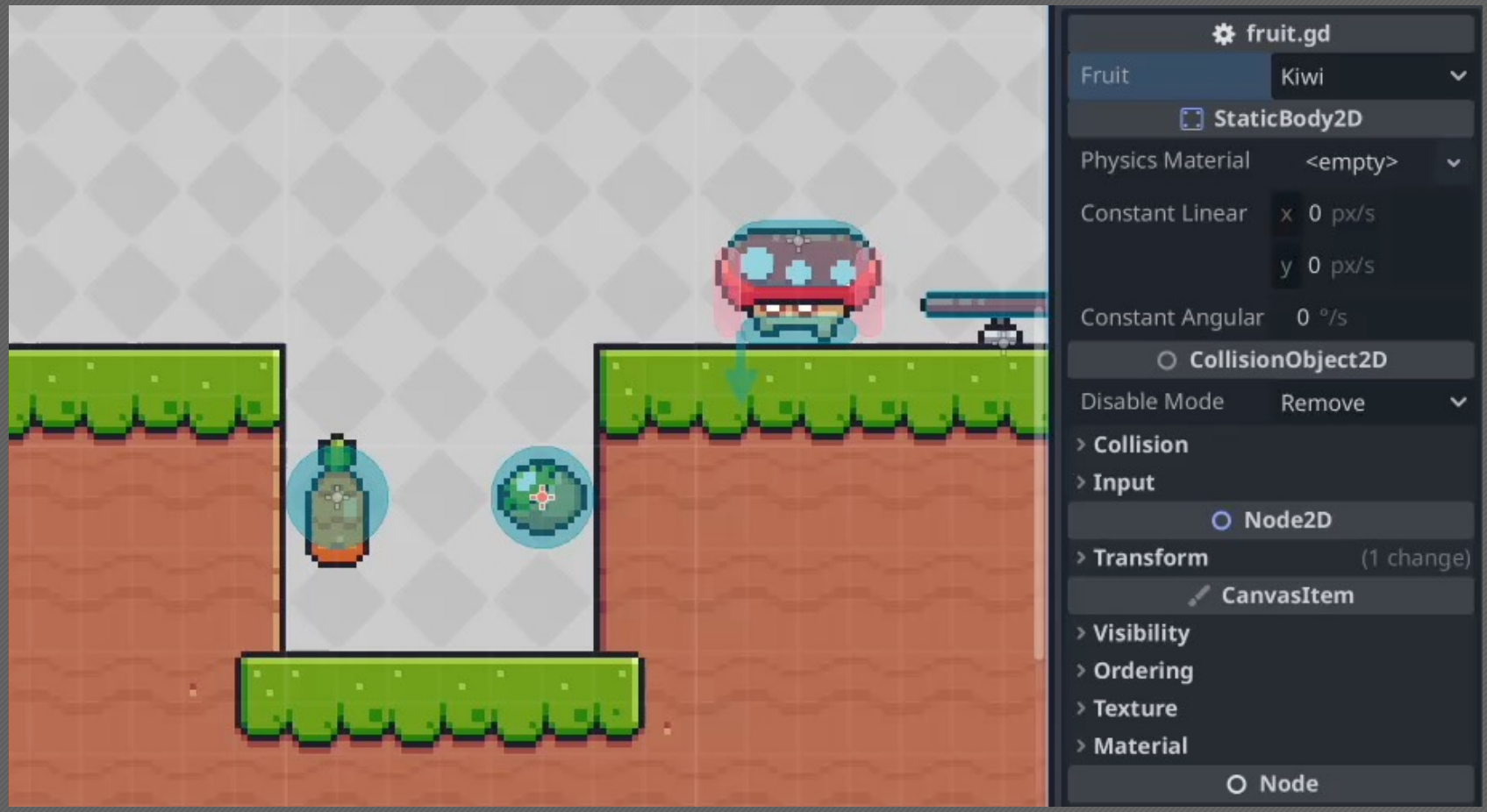
Create a set_fruit Setter



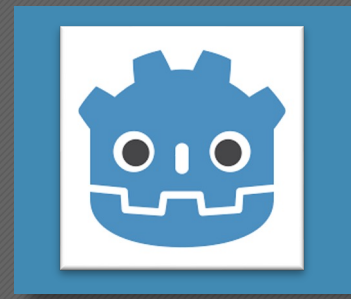
Variables can have set or get functions defined as part of the declaration. The AnimatedSprite2D node should change what animation is played **any time** the fruit value is changed, so it makes sense to use a setter in this case.

```gdscript
@tool
extends Area2D

@onready var ANIMATED_SPRITE := $AnimatedSprite2D
@export_enum("Kiwi", "Pineapple") var fruit := "Kiwi" : set = set_fruit


func set_fruit(new_value:String) -> void:
    fruit = new_value
    # Since ANIMATED_SPRITE is an onready var, it doesn't become available until just before _ready.
    # Node values are set prior to the scene being ready, so trying to use it would produce Nil errors.
    ANIMATED_SPRITE.play(fruit.to_lower())


func _ready() -> void:
    ANIMATED_SPRITE.play(fruit.to_lower())
```

Results of the Fruit Setter



Updating the Fruit to “Be Collected”



- Connect the Fruit’s Area2D body_entered signal to its own script.

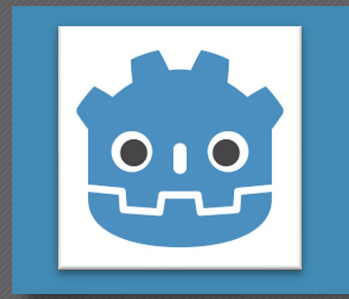
```
19 func _on_body_entered(body: Node2D) -> void:
20     if body.has_method("collect_fruit"):
21         body.collect_fruit(fruit)
22         despawn()
```

- The has_method func is an Object-inherited method that will return true if the given method name exists for the object on which it’s called.

```
25 func despawn() -> void:
26     $CollisionShape2D.disabled = true
27     $AudioStreamPlayer2D.play()
28     ANIMATED_SPRITE.play("collected")
```

- If anything collides with the Fruit which has the “collect_fruit” method, call collect_fruit and despawn.

Destroying the Fruit scene after Despawn



The despawn function, as it exists, will “get stuck” on the last frame of the collected animation. It requires a similar treatment as the Trampoline, except the Fruit needs to be set to visible = false to hide the fruit from the scene.

Hiding the scene does not remove it, however. The queue_free() func marks the node as ready and safe to be removed from the active scene tree and will eventually be cleaned up from memory in garbage collection.

```
31  func _on_animated_sprite_2d_animation_finished() -> void:
32      if $AnimatedSprite2D.animation == "collected":
33          visible = false
34          queue_free()
```

Adding collect_fruit to the Player

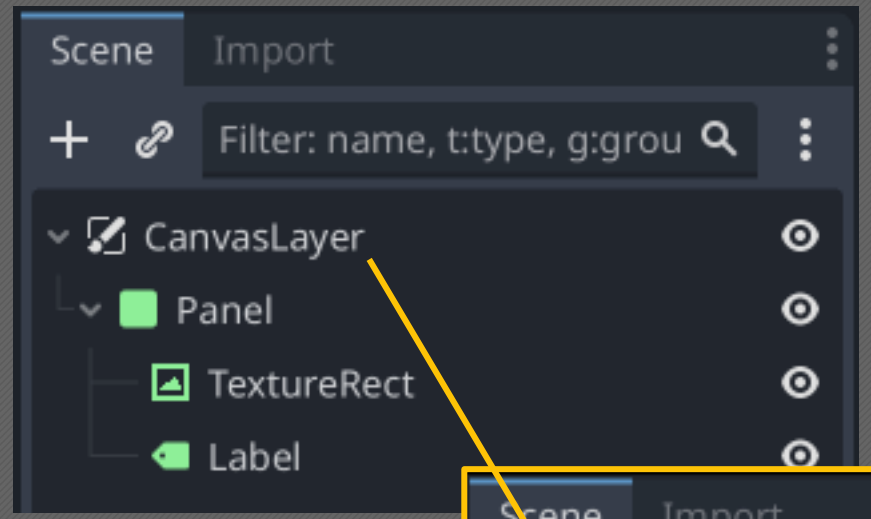
- The Player script needs to have the collect_fruit func on it for the Fruit scene to invoke it.

- Add this basic implementation, which will print to the console and serve a primary purpose of verifying the signals and functions are connected properly.

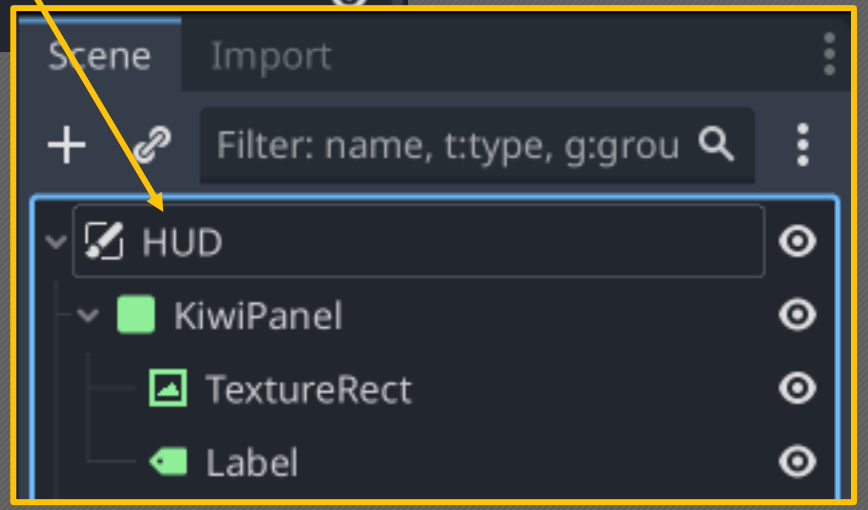
```
43  func collect_fruit(fruit:String) -> void:
44      print("Player collected %s" % fruit)
45
```

Creating a HUD to display Fruit Counts

- Create a new scene with a CanvasLayer as the root node type.
- A CanvasLayer is used for independent rendering of objects within a 2D scene, and it follows the viewport.
- Create the scene tree shown to the right and give the nodes appropriate names for the fruit you've chosen to use.



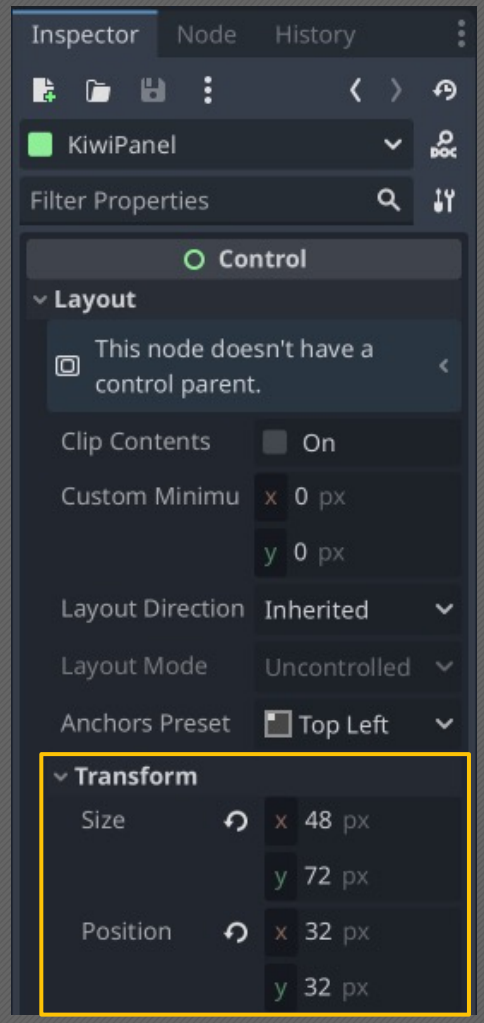
Note: Green icons represent a Control node



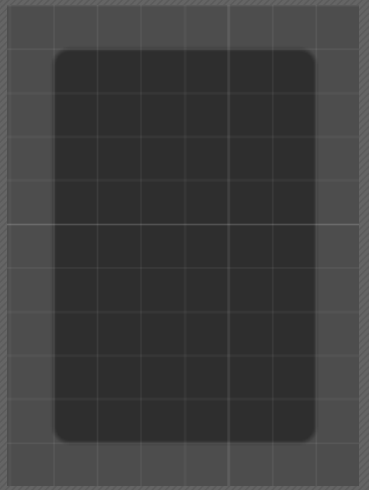
Creating a HUD to display Fruit Counts

- The Panel component is a shaded container for other Control-inherited components.

- Set the Size and Position elements are shown to the right (or position the panel to your liking).



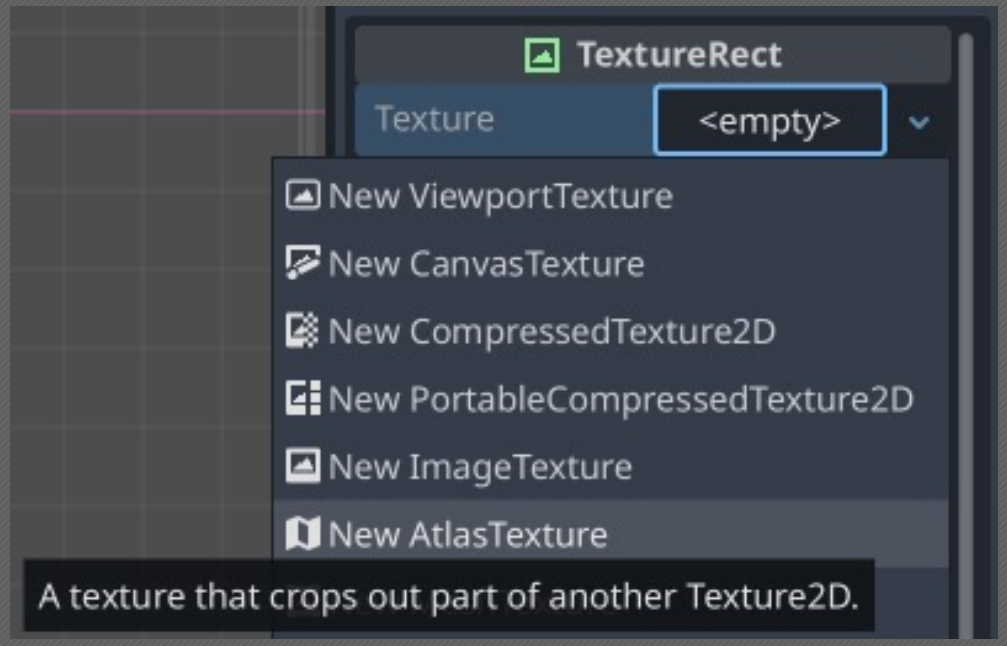
Result



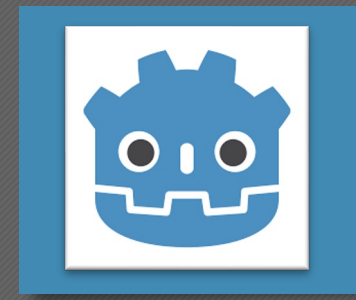
Creating a HUD to display Fruit Counts

An AtlasTexture is a node for which only part of a larger image (or sprite sheet) can be used.

- Create a new AtlasTexture as the Texture value for the TextureRect node.

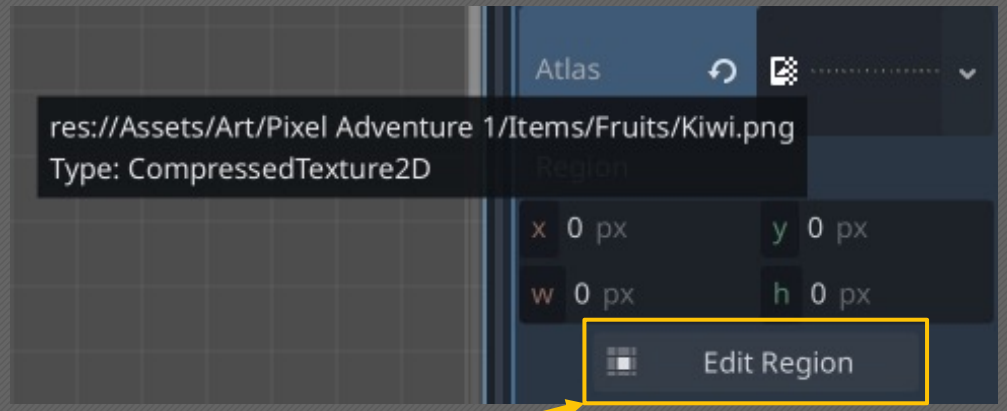


Creating a HUD to display Fruit Counts



An AtlasTexture requires a source image file (the Atlas property) to be able to choose which part of the Atlas is used as the texture.

- Assign an Atlas image using the file for the appropriate fruit this panel will represent.
- After an Atlas is assigned, click the Edit Region button.



Creating a HUD to display Fruit Counts

The Region Editor allows us to select a specific region of the atlas image. The region is what will be used as the TextureRect's displayed image.

The default Snap Mode is None, which makes it hard to be pixel perfect.

- Change the Snap Mode to Grid Snap.

Creating a HUD to display Fruit Counts

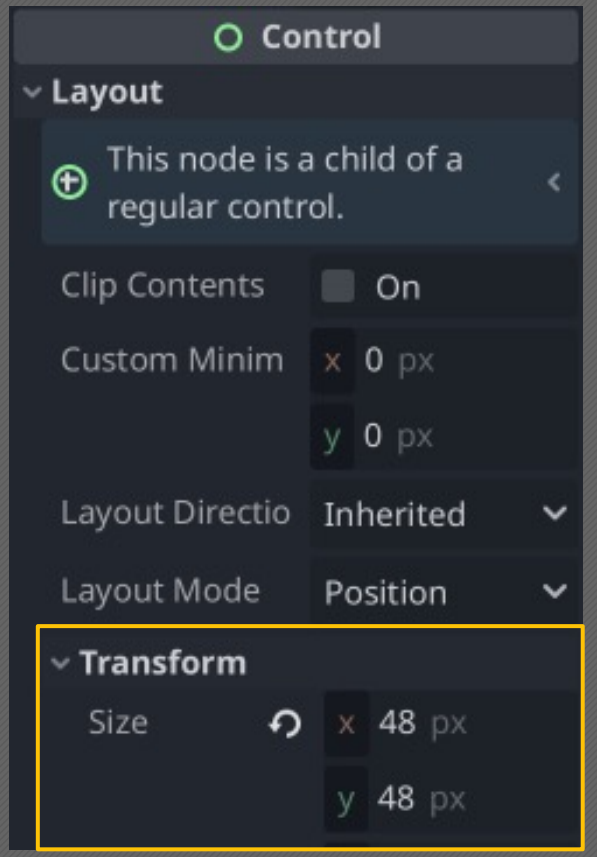
In Grid Snap mode, the offset and step size of grid snaps can be changed for better snapping based on sprite sizes.

- Resize the region to include only the first frame of the fruit sprite sheet (32x32 px) and close the prompt.

Creating a HUD to display Fruit Counts

Lastly, the size of the texture needs to be larger, so it looks like it fills the panel width.

- Update the Transform Size of the TextureRect to be 48px in both dimensions.

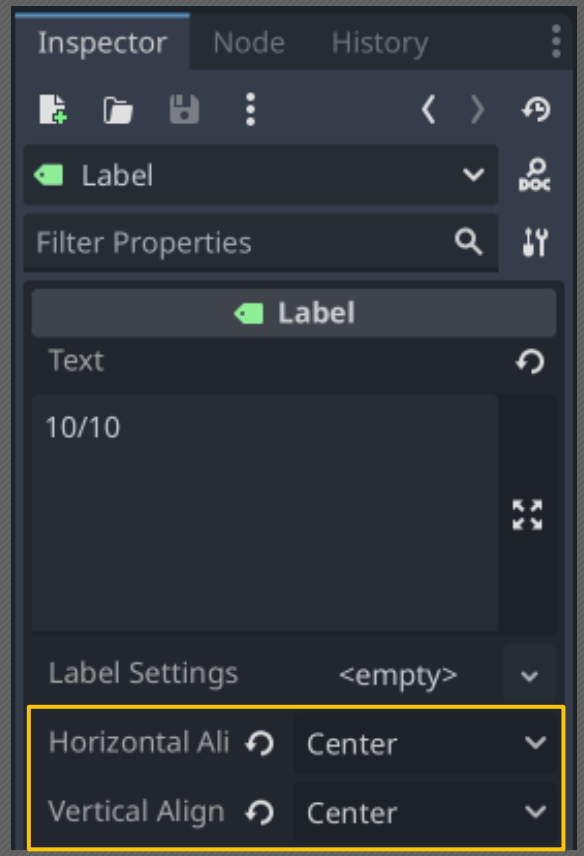


Result

Creating a HUD to display Fruit Counts

A Label is a basic line of text.

- Update the Label node to have a basic placeholder value, such as “10/10”.
- Set the Horizontal and Vertical alignments to be Center.



Result

Control Node Anchors

By default, most Control nodes are anchored to the Top Left corner of the container the control exists within (represented by the green pins).

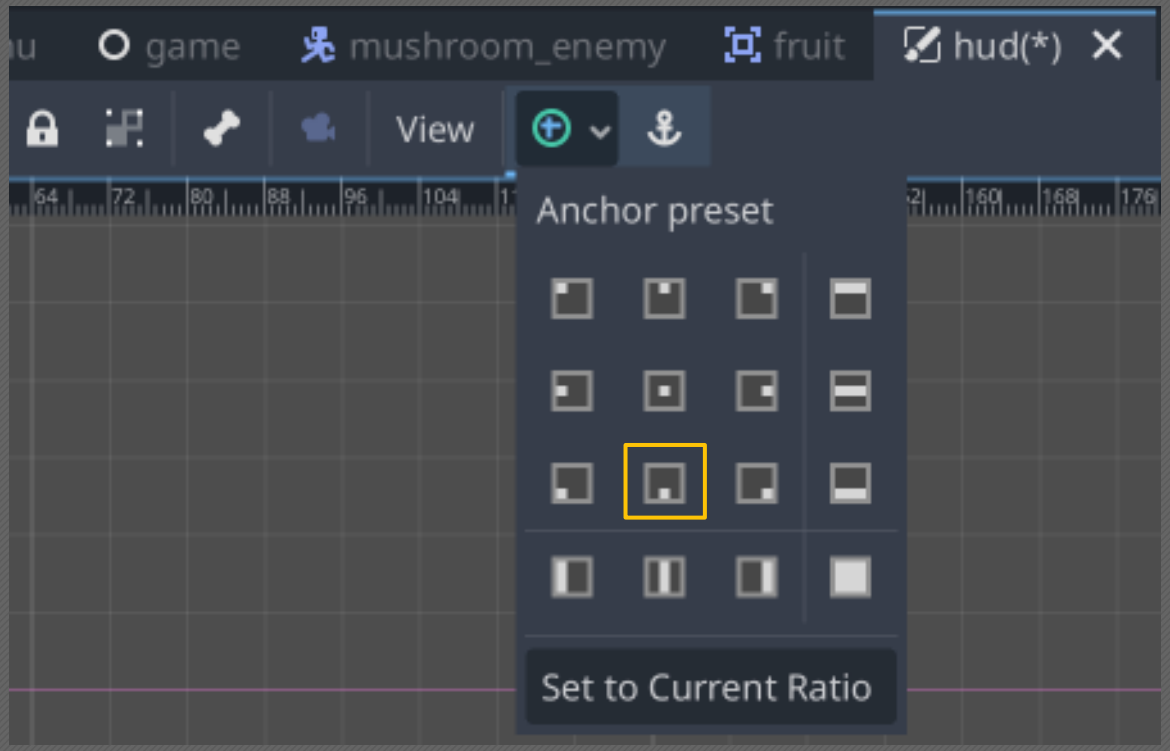
An Anchor influences how a control node resizes and repositions itself relative to its container.

Control Node Anchors

When a control node is selected in the scene view or scene tree, an anchor option becomes available in the scene view options row.

The anchor dropdown offers different anchoring options.

- With the Label node selected, select the Center Bottom anchor.



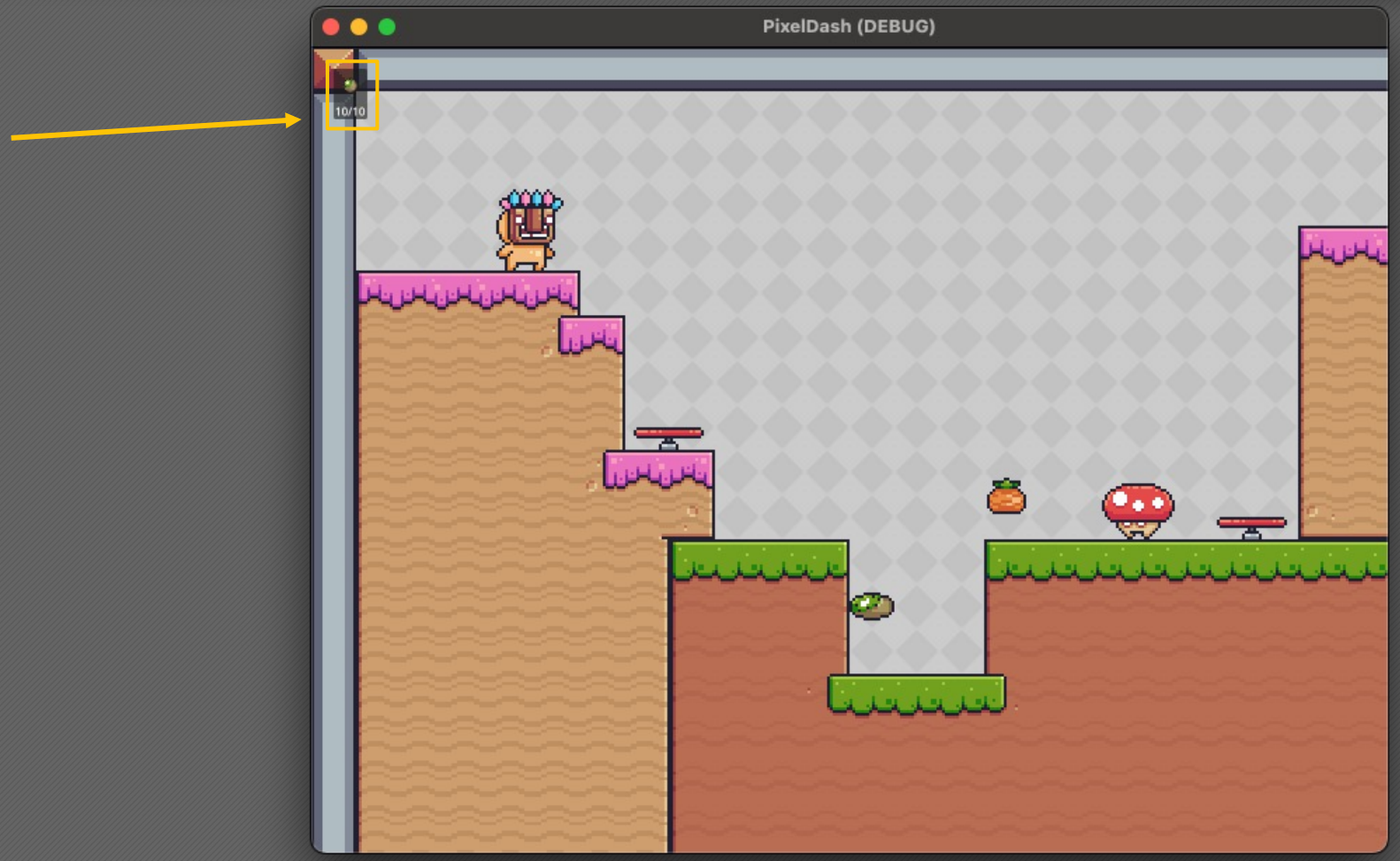
Control Node Anchors

Notice that the green pins representing the anchor for the Label node have been repositioned to the bottom of the panel.

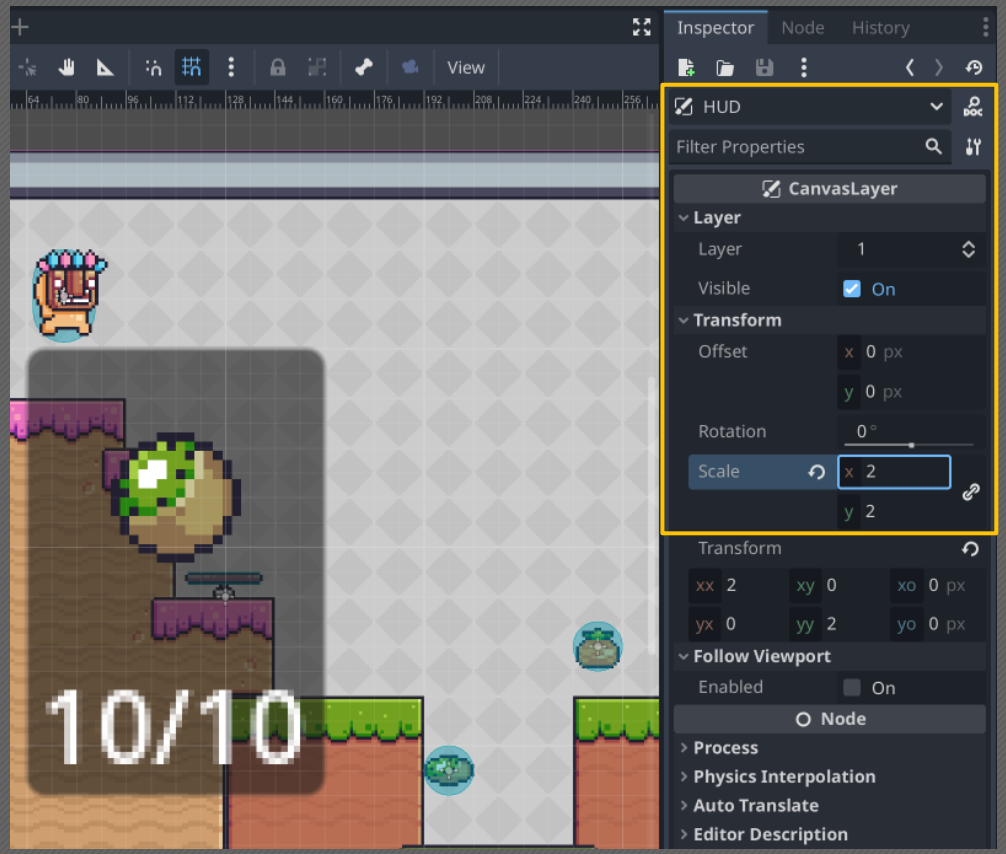
We now have something that can be used to represent how many of that fruit has been collected!

Result

But this is a little awkward...



Upscale the HUD



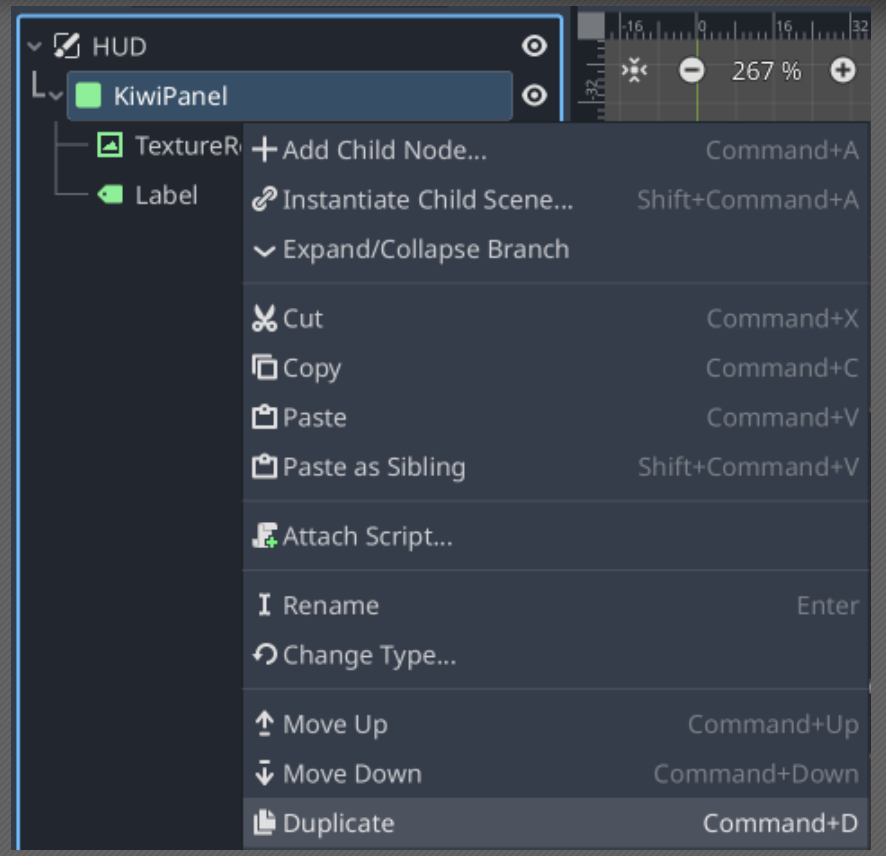
Duplicating Nodes

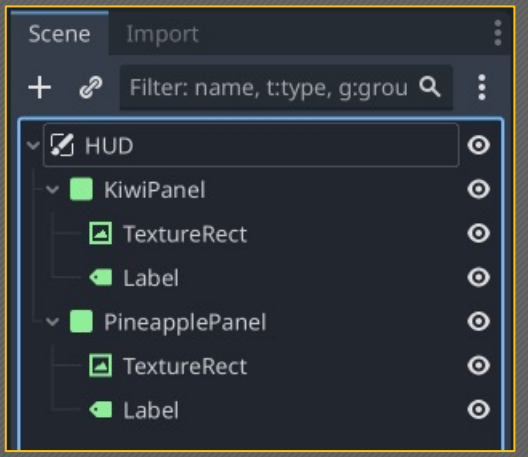
Nodes can be duplicated via the right-click context menu.

The duplication is referential, meaning resources (like the Atlas texture) are duplicated as references to the original resource.

- Duplicate the first Panel and rename the new panel to whatever other fruit was chosen for your game.

- Reposition the panel to sit to the right of the first panel.





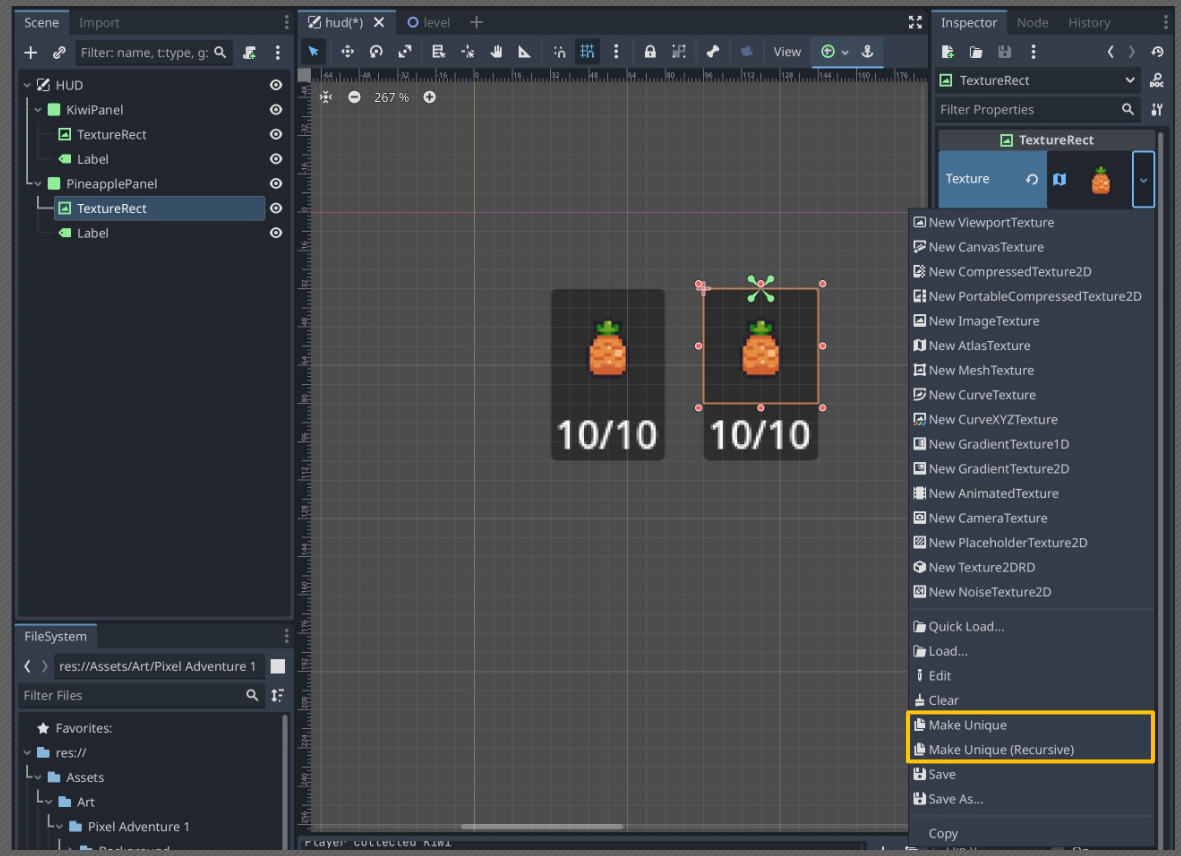
Making Resources Unique

Since they're referential, changing one atlas changes both nodes. No good!

To fix this, a resource can be made unique via the dropdown for that resource.

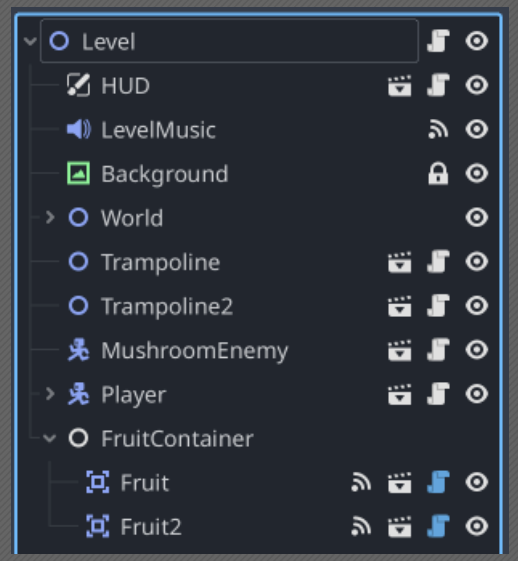
- Make the duplicated panel's TextureRect Texture unique.

- Update the AtlasTexture of the second panel to be for the fruit of your choice.

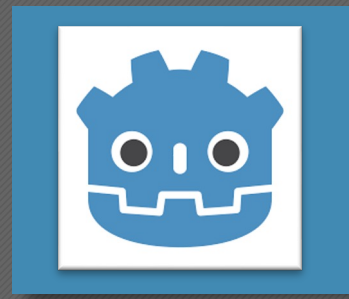


Creating a HUD to display Fruit Counts

- Add the HUD scene to the Level as the first child of the Level.
- Add a plain-old Node named FruitContainer, and instantiate some fruit into the scene.
- A Node is like a Node2D, except it has no position.



Wiring Up the Fruit



The Fruit and the Player need some way to interact with each other.

- Add a class_name of Fruit to the Fruit script.
- Custom signals can be defined with the signal keyword; add a signal named collected.

A signal definition follows the same syntax as a method header.

```
1  @tool
2  class_name Fruit
3  extends Area2D
4
5  signal collected(fruit_name:String)
6
7  @onready var ANIMATED_SPRITE := $AnimatedSprite2D
8  @export_enum("Kiwi", "Pineapple") var fruit := "Kiwi" : set = set_fruit
```

Wiring Up the Fruit

Custom signals can be emitted thru code via the emit function.

The call to emit must have args that match the parameters of the signal definition.

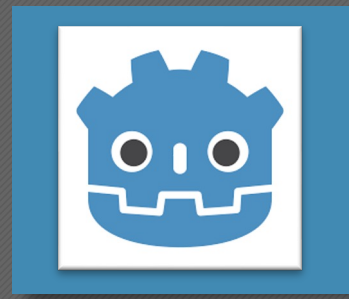
- Update the _on_body_entered signal handler in the Fruit script to emit the collected signal.

```
1  @tool
2  class_name Fruit
3  extends Area2D
4
5  signal collected(fruit_name:String)
6
7  @onready var ANIMATED_SPRITE := $AnimatedSprite2D
8  @export_enum("Kiwi", "Pineapple") var fruit := "Kiwi" : set = set_fruit
```

Takes a single string

```
22 func _on_body_entered(body: Node2D) -> void:
23     if body.has_method("collect_fruit"):
24         body.collect_fruit(fruit)
25         collected.emit(fruit)
26         despawn()
```

Listen for Fruit Collected in the Level Script



The Level needs to connect to the signals of all the Fruit in the Level.

- Add a _ready func to the Level script, which loops through all child nodes of the FruitContainer node which are of type Fruit (matching the class_name).

- The connect function takes a Callable arg, which is a reference to a method which has a method header that matches the signal parameters.

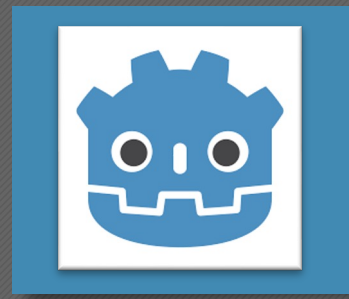
The Level script connects to the Fruit signals.

```
4  func _ready() -> void:
5      for child in $FruitContainer.get_children():
6          if child is Fruit:
7              child.collected.connect(handleFruitCollected)
8
9
10 # When any of the fruit emit their collected signal, we handle that signal by updating our fruit
11 # totals count and send the total for that fruit to the HUD for updating the UI/control components.
12 func handleFruitCollected(fruitName:String) -> void:
13     print("Collected %s" % fruitName)
```

The Fruit's collected signal.

```
1  @tool
2  class_name Fruit
3  extends Area2D
4
5  signal collected(fruit_name:String)
6
7  @onready var ANIMATED_SPRITE := $AnimatedSprite2D
8  @export_enum("Kiwi", "Pineapple") var fruit := "Kiwi" : set = set_fruit
```

Update the HUD Script to update fruit counts

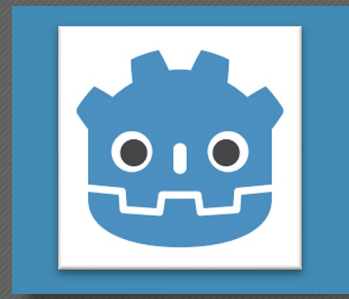


The HUD will need a way to be told how to update the fruit totals by the Level script.

- Create the following script for the HUD scene using var names representing your chosen fruit.

```
1   class_name HUD
2   extends CanvasLayer
3
4   @onready var kiwiCountLabel := $KiwiPanel/Label
5   @onready var pineappleCountLabel := $PineapplePanel/Label
6
7   func update_kiwi_count(count:int, total:int) -> void:
8       kiwiCountLabel.text = "%d/%d" % [count, total]
9
10  func update_pineapple_count(count:int, total:int) -> void:
11      pineappleCountLabel.text = "%d/%d" % [count, total]
```

Update the Level to track Fruit totals

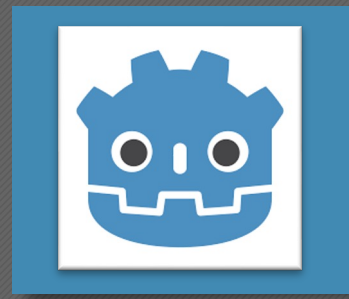


The Level needs to keep track of fruit totals to tell the HUD how to update with the right values.

- Add a new fruit_totals and fruit_maximums Dictionary (a key-value pair container) to store the fruit of your choosing.

```
3  var fruit_totals := {
4      "Kiwi": 0,
5      "Pineapple": 0
6  }
7
8  var fruit_maximums := {
9      "Kiwi": 0,
10     "Pineapple": 0
11 }
```

Update the Level to track Fruit totals



Update the Level script to populate and update the fruit count dictionaries on ready and when the fruit collected signals are emitted. Lastly, the HUD needs to be updated with the totals.

```
15  func _ready() -> void:
16      for child in $FruitContainer.get_children():
17          if child is Fruit:
18              fruit_maximums[child.fruit] += 1
19              child.collected.connect(handleFruitCollected)
20      hud.update_kiwi_count(0, fruit_maximums["Kiwi"])
21      hud.update_pineapple_count(0, fruit_maximums["Pineapple"])
22
23
24  # When any of the fruit emit their collected signal, we handle that signal by updating our fruit
25    # totals count and send the total for that fruit to the HUD for updating the UI/control components.
26  func handleFruitCollected(fruitName:String) -> void:
27      fruit_totals[fruitName] += 1
28      hud.update_kiwi_count(fruit_totals['Kiwi'], fruit_maximums["Kiwi"])
29      hud.update_pineapple_count(fruit_totals['Pineapple'], fruit_maximums["Pineapple"])
```

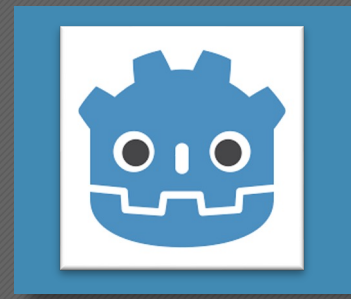
Don’t forget to add an onready node reference to the HUD.

Fruit Collected Race Condition

Since the fruit is technically still in the scene with active collision shapes, a single fruit can be collected more than once if you move fast enough.

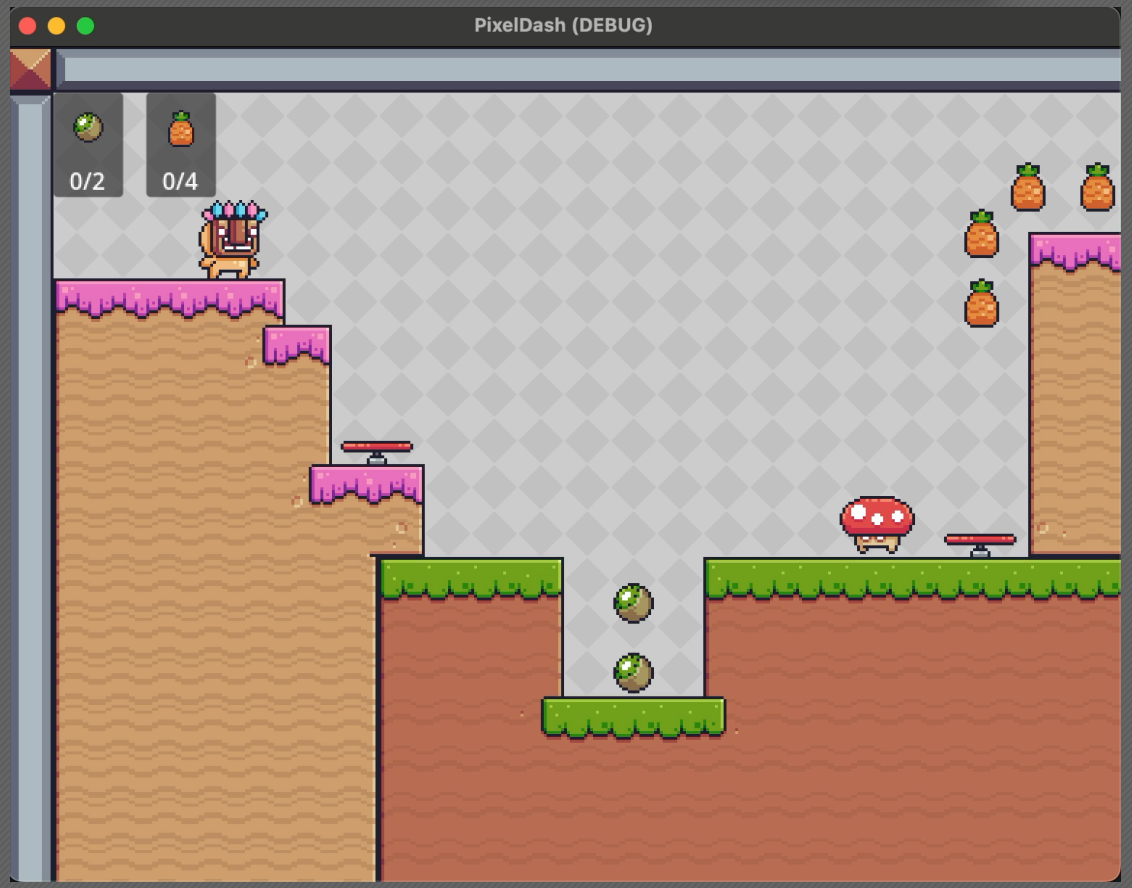
We need a mechanism to prevent undesirable double-collection of individual fruit.

Fruit Collected Race Condition

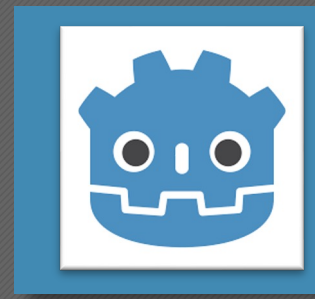


Add a new is_collected bool to prevent being able to collect it more than once.

```
7   var is_collected := false
8
9   @onready var ANIMATED_SPRITE := $AnimatedSprite2D
10  @export_enum("Kiwi", "Pineapple") var fruit := "Kiwi" : set = set_fruit
11
12
13  func set_fruit(new_value:String) -> void:
18
19
20  func _ready() -> void:
22
23
24  func _on_body_entered(body: Node2D) -> void:
25      if not is_collected and body.has_method("collect_fruit"):
26          body.collect_fruit(fruit)
27          collected.emit(fruit)
28          despawn()
29
30
31  func despawn() -> void:
32      is_collected = true
33      $AudioStreamPlayer2D.play()
34      ANIMATED_SPRITE.play("collected")
```

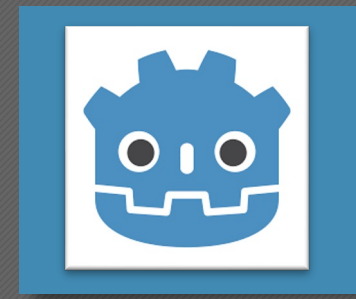


Lab Time (~25 Minutes)

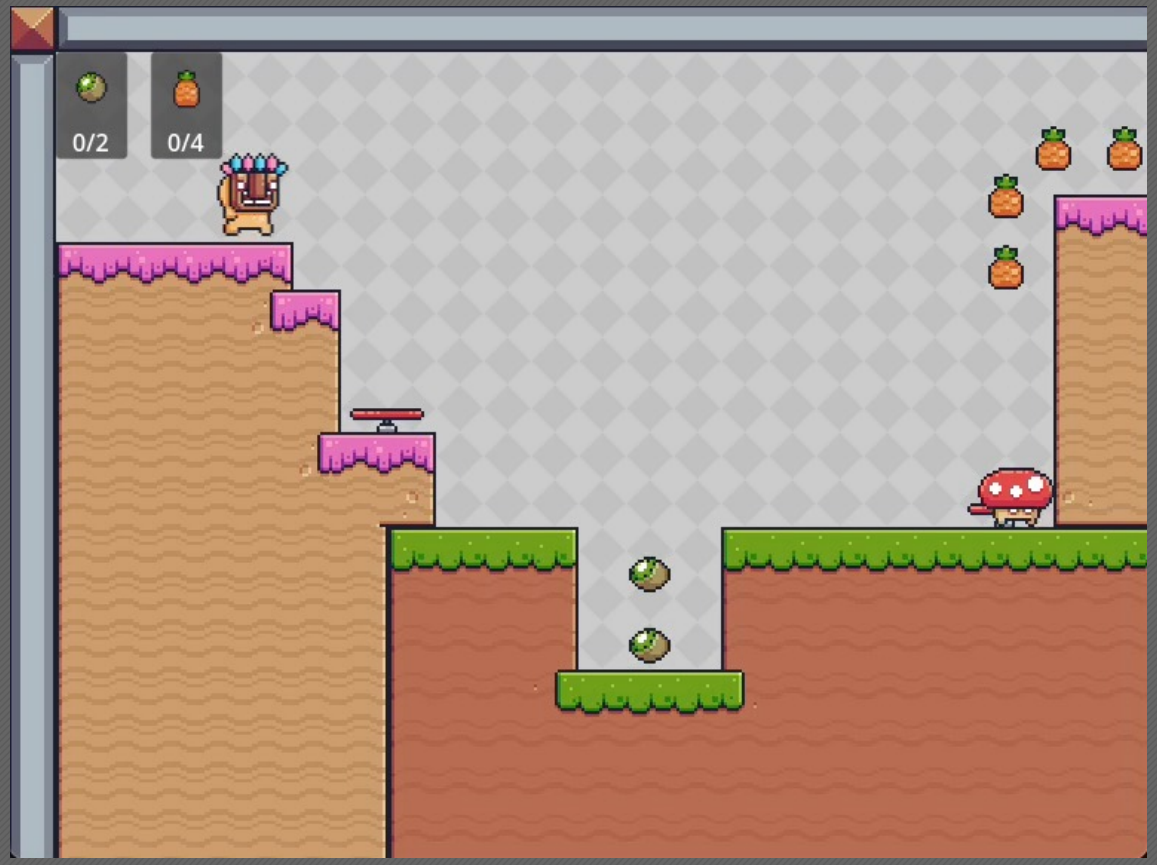


- Create a Fruit scene and script.
- Make the Player able to collect the fruit.
- Create a CanvasLayer scene and script named HUD.
- Update the Fruit, Level, Player, and HUD scripts to handle updating fruit counts.

Workshop Goal #7



1. ✅ Control & Animate a Character
2. ✅ Build a Level with Tiles
3. ✅ Add Player & Level Polish
   (Camera, Gravity, Sound, Background)
4. ✅ Detect Collisions with Trampolines
5. ✅ Create a basic Mushroom Enemy
6. ✅ Add Collectible Fruit and HUD Display
7. ➡️ Respawn Character when Defeated
8. Create a Main Menu & Change Scenes
9. Open Workshop, Tinker Time



Update Player to Emit a defeated Signal

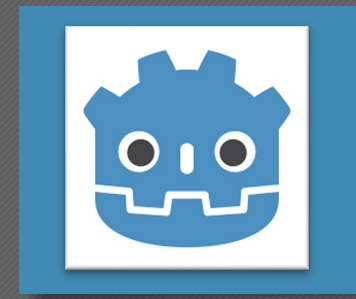
- Update the Player script to have class_name Player, and to have a defeated signal that takes no parameters.

```
1   class_name Player
2   extends CharacterBody2D
3
4   signal defeated
```

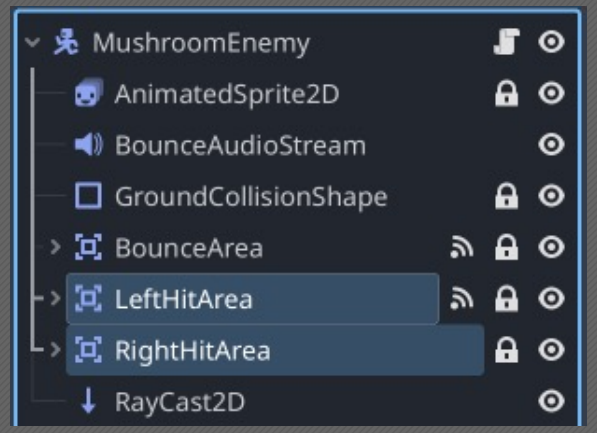
- Add a take_damage(int) func, which emits the defeated signal when invoked.

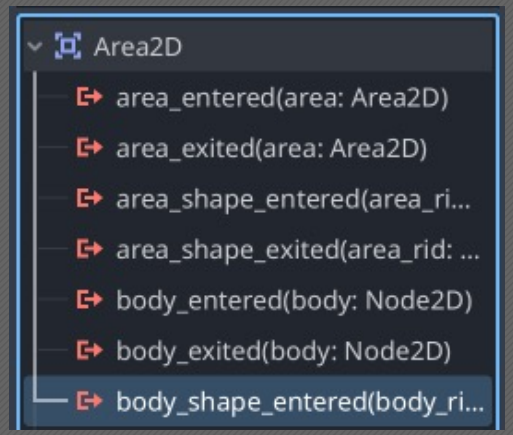
```
49  func take_damage(amount:int) -> void:
50      defeated.emit()
```

Update MushroomEnemy with Area Handlers



- Add signal handlers for the body_shape_entered signal of LeftHitArea and RightHitArea areas on the MushroomEnemy to the MushroomEnemy script.

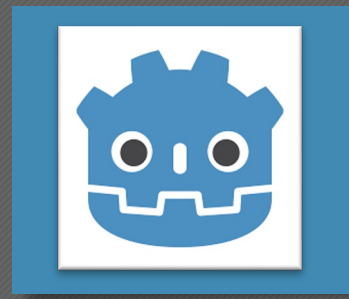




- Add a Player type check for the body parameter of both handler funcs, and call take_damage on the body if true.

```
32 func _on_left_hit_area_body_shape_entered(body_rid: RID,
33     if body is Player:
34         body.take_damage(1)
35
36
37 func _on_right_hit_area_body_shape_entered(body_rid: RID,
38     if body is Player:
39         body.take_damage(1)
```

Update the Level to Restart itself



The Level needs a reference to the Player so it can listen for the Player's defeated signal.

- Add an onready var to reference the Player node.
- Connect the Level to the Player defeated signal, using restart_level as the Callable.
- Implement restart_level to call get_tree().change_scene_to_file().

```
13    @onready var hud : HUD = $HUD
14    @onready var player := $Player
```

```
19  func _ready() -> void:
20      for child in $FruitContainer.get_children():
21          if child is Fruit:
22              fruit_maximums[child.fruit] += 1
23              child.collected.connect(handleFruitCollected)
24      player.defeated.connect(restart_level)
```

```
38  func restart_level() -> void:
39      get_tree().change_scene_to_file("res://Scenes/Level/level.tscn")
40
```

get_tree and change_scene_to_file functions

```
38 func restart_level() -> void:
39     get_tree().change_scene_to_file("res://Scenes/Level/level.tscn")
40
```

The get_tree function returns the scene tree (SceneTree object) that contains the node on which get_tree() is called.

The change_scene_to_file function changes the running scene to the one at the given path, after loading it into a PackedScene and creating a new instance of that PackedScene.

The implementation above effectively re-instantiates the Level scene in the scene tree, which functions as a “reset” for the scene.

Player Respawns after Colliding with Enemy

Add Polish to the Player being Defeated

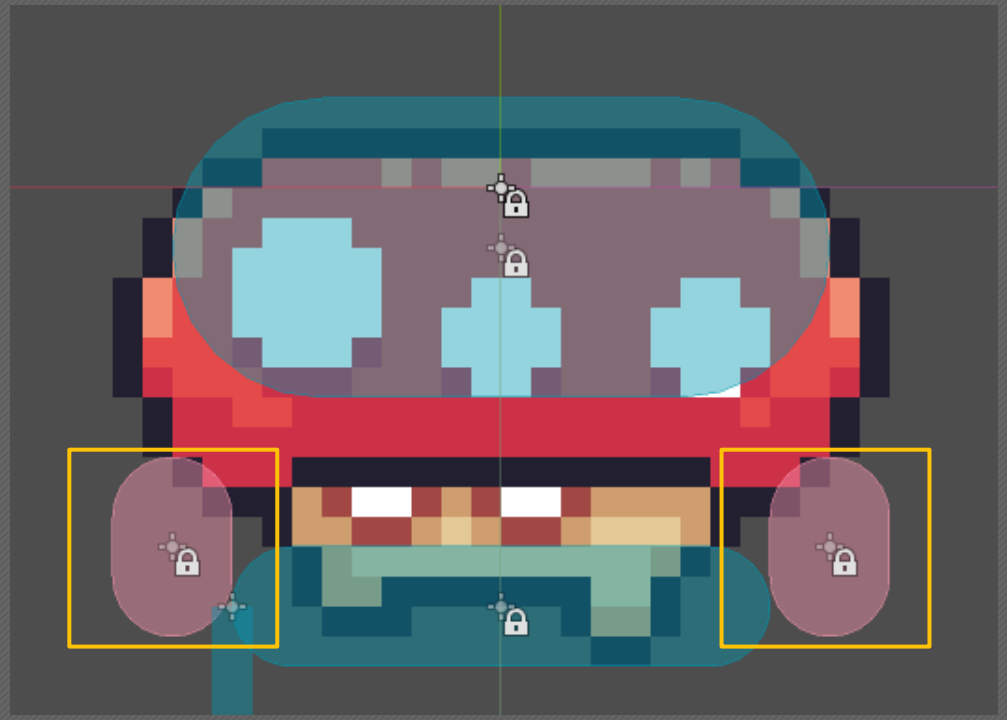
- Update the Player animated sprite to contain a “defeat” animation, playing the Disappearing sprite sheet frames in the Main Characters folder (20 FPS, no loop).
- Use a new is_defeated bool to prevent multiple calls to defeat the Player.
- Use the is_defeated variable to prevent allowing the Player to move if the Player is defeated.
- Emit the defeated signal when the new “defeat” animation completes.

```
 9   var is_defeated := false
10
11   func _physics_process(delta: float) -> void:
12       if not is_defeated: handle_movement()
13
14
15   func _input(event: InputEvent) -> void:
19
20
21   func handle_movement() -> void:
50
51   func collect_fruit(fruit:String) -> void:
53
54
55   func take_damage(amount:int) -> void:
56       if not is_defeated:
57           is_defeated = true
58           $AnimatedSprite2D.play("defeat")
59
60
61   func _on_animated_sprite_2d_animation_finished() -> void:
62       if $AnimatedSprite2D.animation == 'defeat':
63           defeated.emit()
```

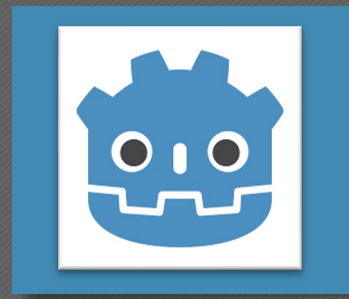
Tweaking MushroomEnemy Collisions

The Area2D does not prevent bodies from entering the area, so it's common for the hit areas to detect collision in the same frame the Player should bounce.

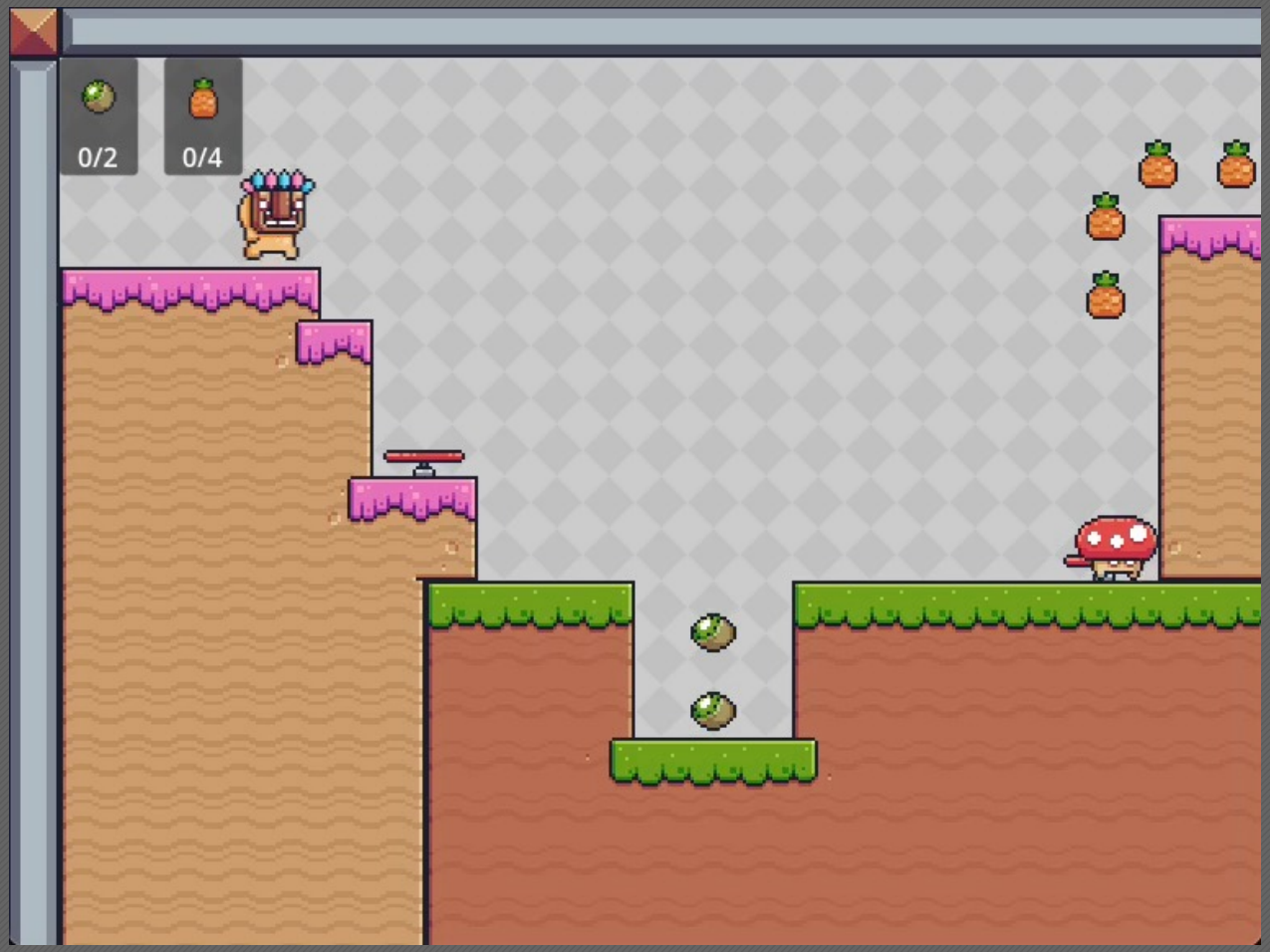
- Resize the MushroomEnemy's left and right hit area collision shapes to prevent defeating the Player when the Player bounces on the enemy.



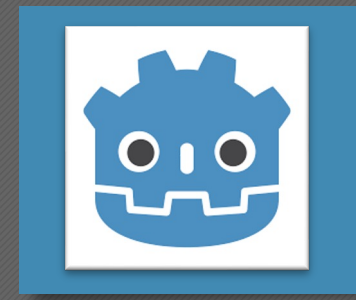
Lab Time (~15 Minutes)



- Update the Player script to be defeated and play an accompanying animation.
- Update the MushroomEnemy to defeat the Player when the Player enters the HitArea Area2D nodes.
- Update the Level to change to the Level scene when the Player emits the defeated signal.

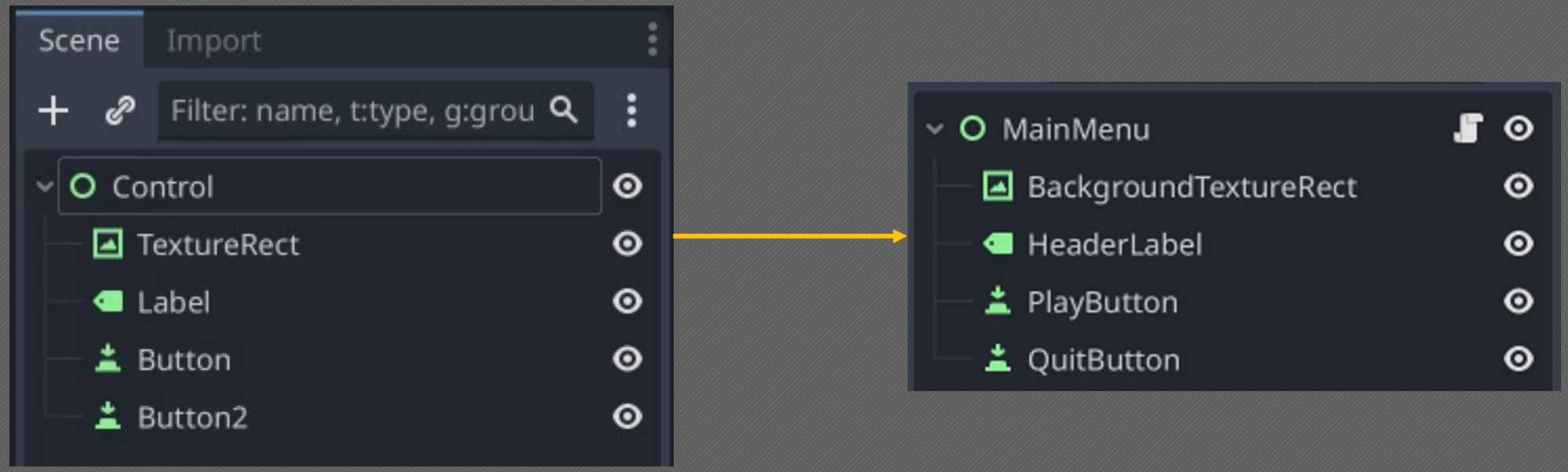


Workshop Goal #8



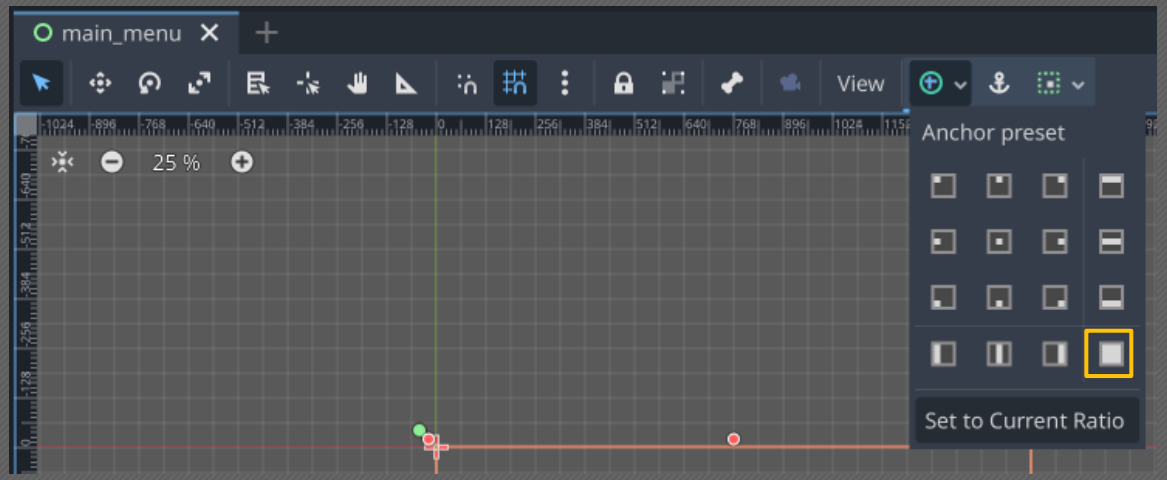
1. ✅ Control & Animate a Character
2. ✅ Build a Level with Tiles
3. ✅ Add Player & Level Polish
   (Camera, Gravity, Sound, Background)
4. ✅ Detect Collisions with Trampolines
5. ✅ Create a basic Mushroom Enemy
6. ✅ Add Collectible Fruit and HUD Display
7. ✅ Respawn Character when Defeated
8. ➡️ Create a Main Menu & Change Scenes
9. Open Workshop, Tinker Time

New MainMenu Scene & Attach a Script

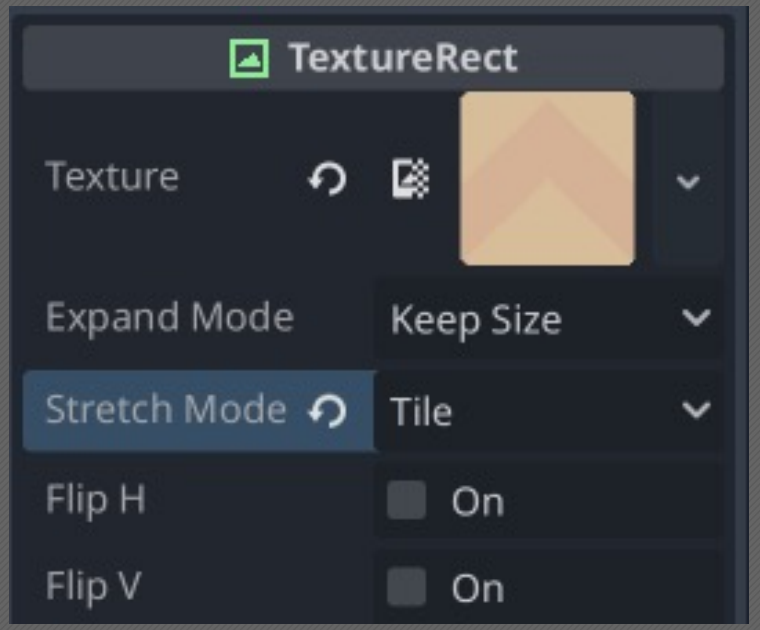


Anchor the MenuMenu & Background

Set the root MainMenu and BackgroundTextureRect nodes to have a Full Rect anchoring.

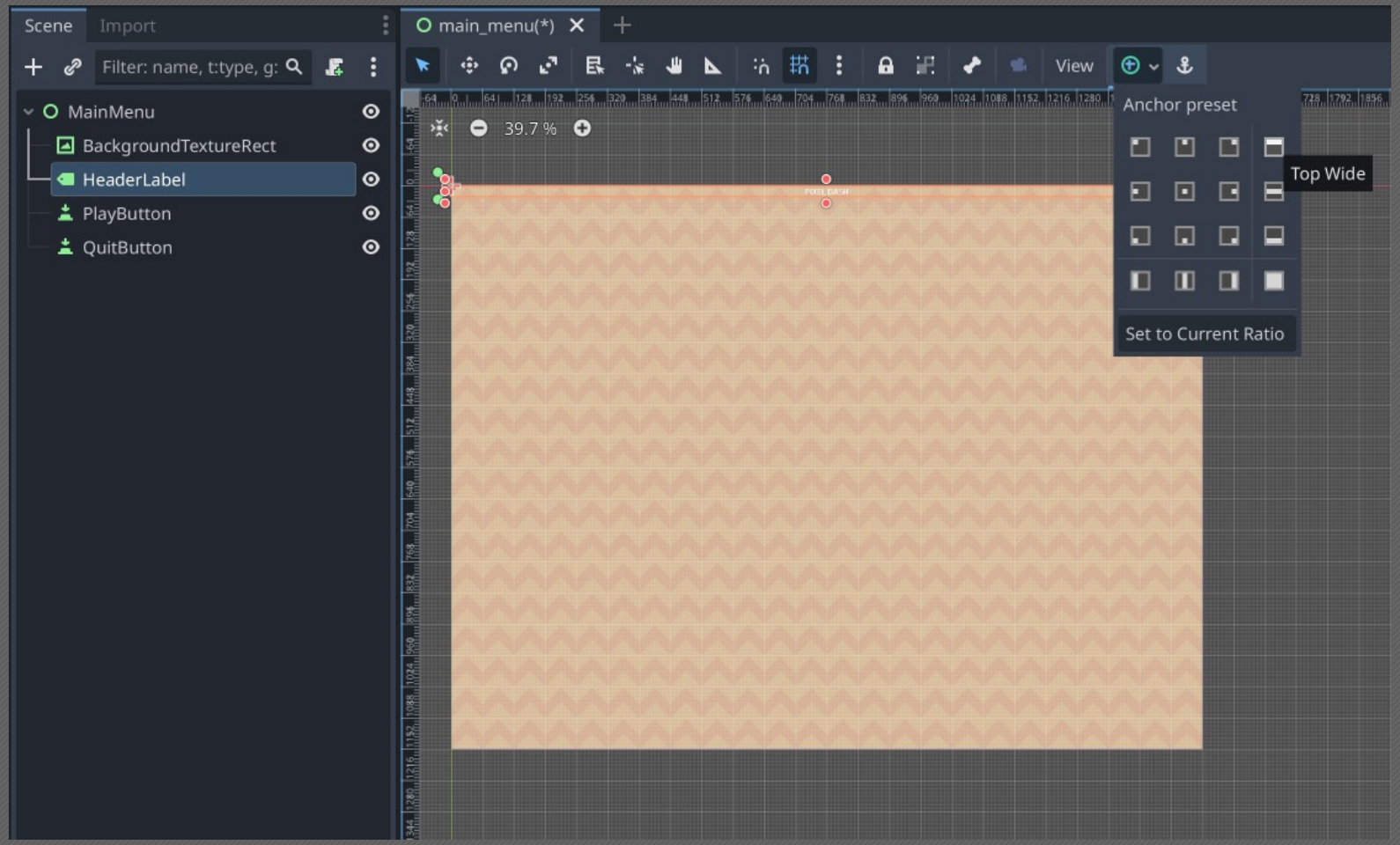


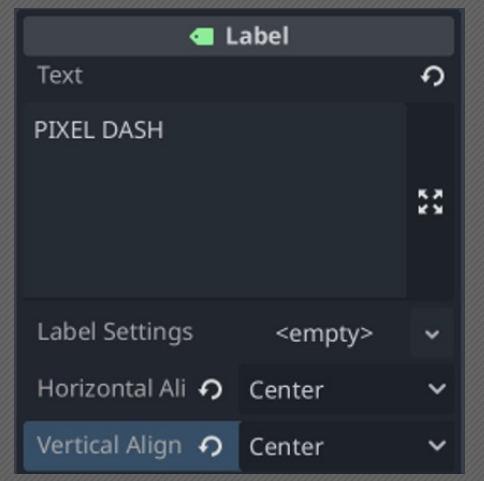
Assign a background image to the BackgroundTextureRect and set the Stretch Mode to Tile



Set up the HeaderLabel

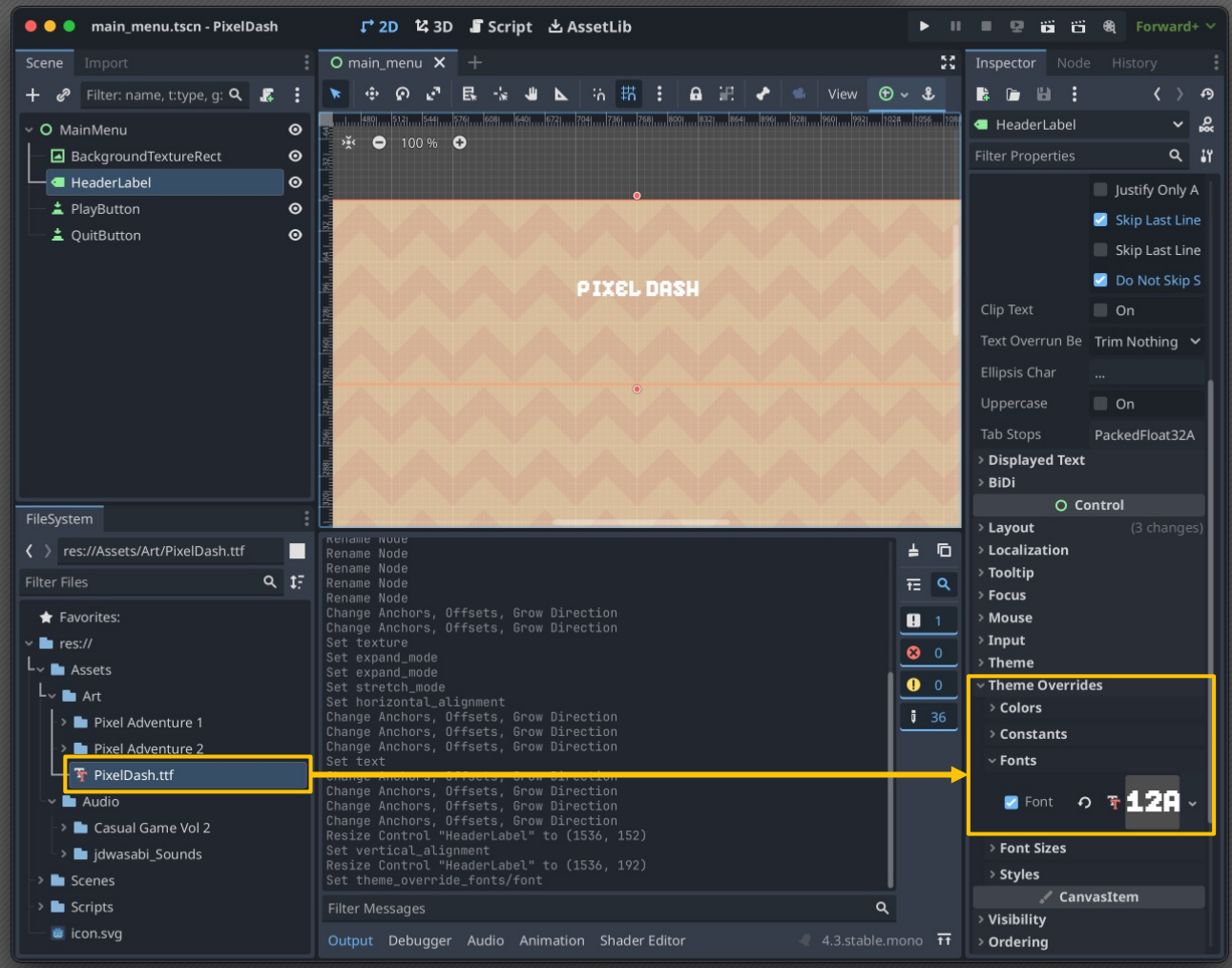
Set the HeaderLabel Text to “PIXEL DASH” and give it a Top Wide anchoring.





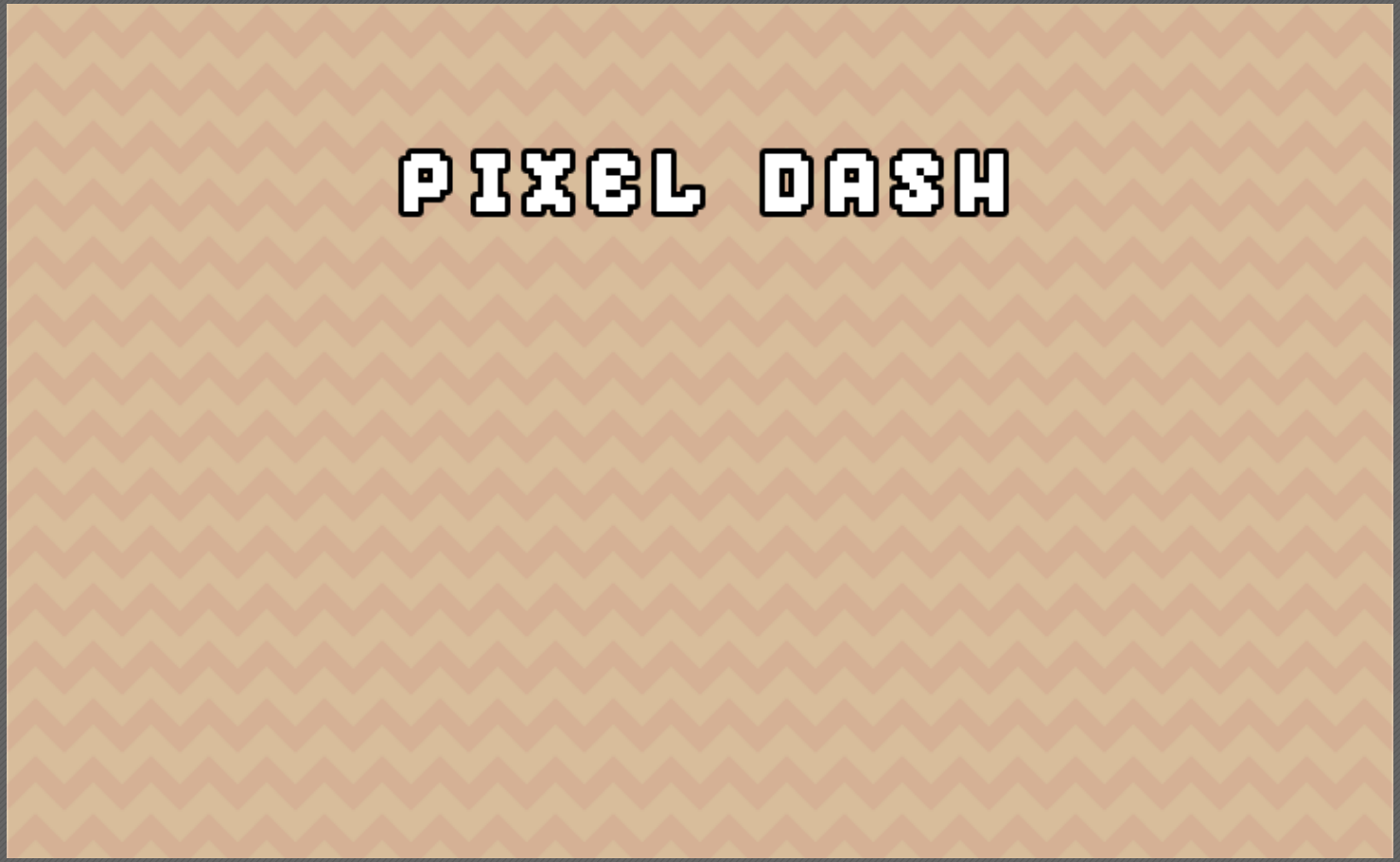
Set up the HeaderLabel

- Expand the height of the HeaderLabel so the words sit comfortably under the top.
- Set the HeaderLabel Font to use the PixelDash.ttf file.
- Increase the Font Size to 48px or to some other size of your liking.



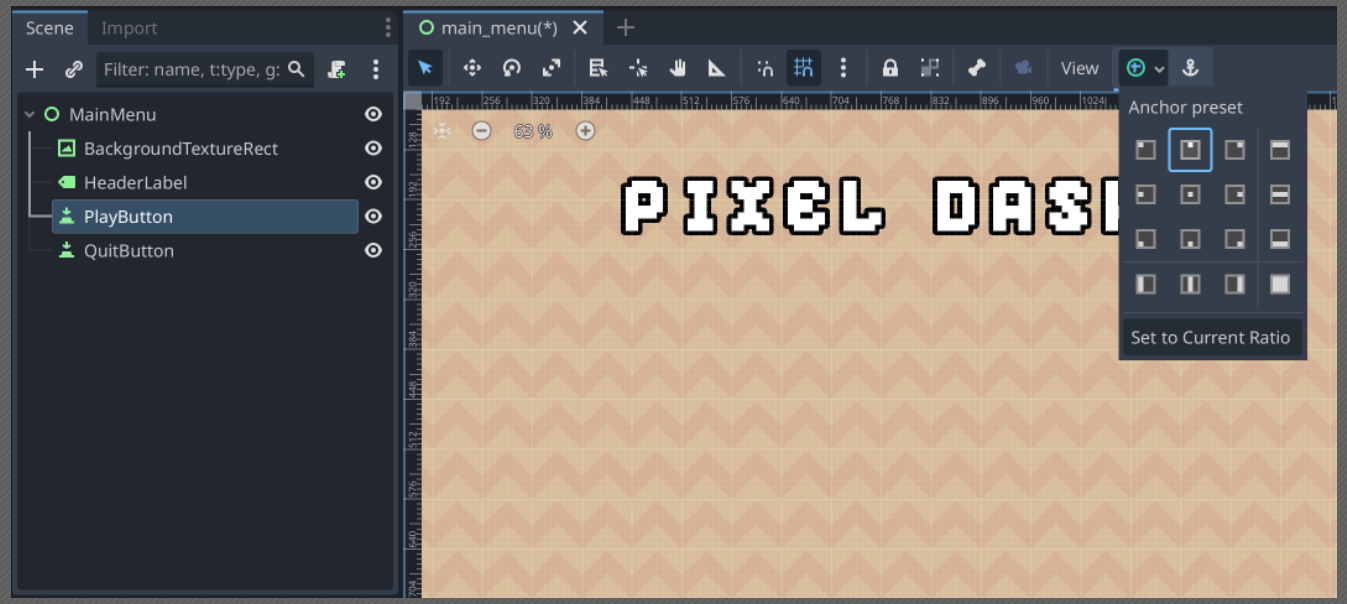
Set up the HeaderLabel

Optionally, add some font shadows or outlines with the other Theme Overrides options.



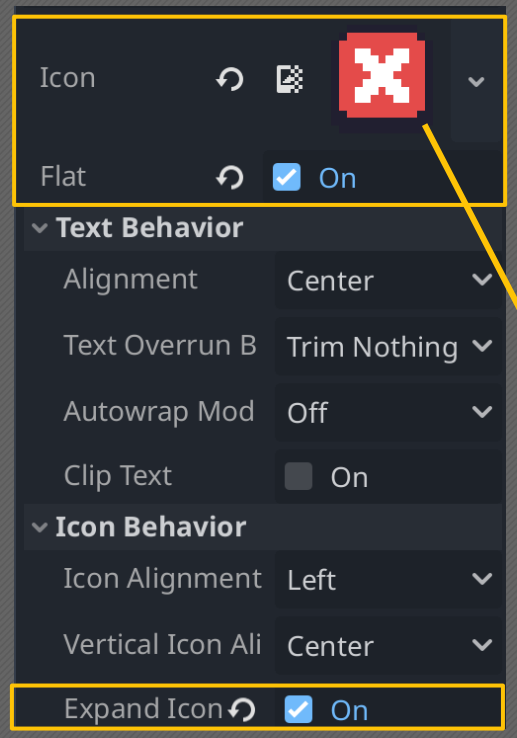
Set up the PlayButton

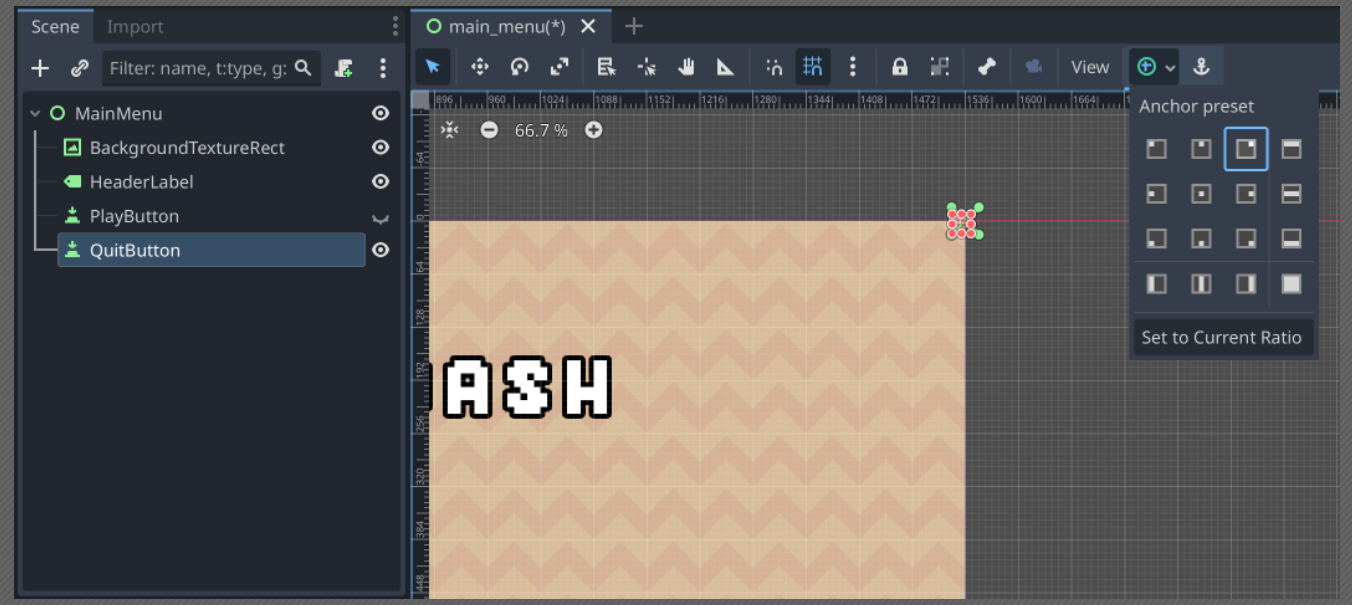
- Anchor the PlayButton to the Center Top.
- Set the icon texture to be the Play.png texture.
- Set the Flat and Expand Icon properties to be true.
- Increase the button size and reposition the PlayButton to sit “comfortably” under the title.



Set up the QuitButton

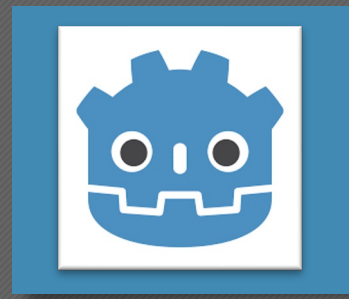
- Anchor the QuitButton to the Top Right.
- Set the Icon, Flat, and Expand Icon settings so the button fits nicely in the corner.





Main Menu Look

Set up the PlayButton Signal Handler

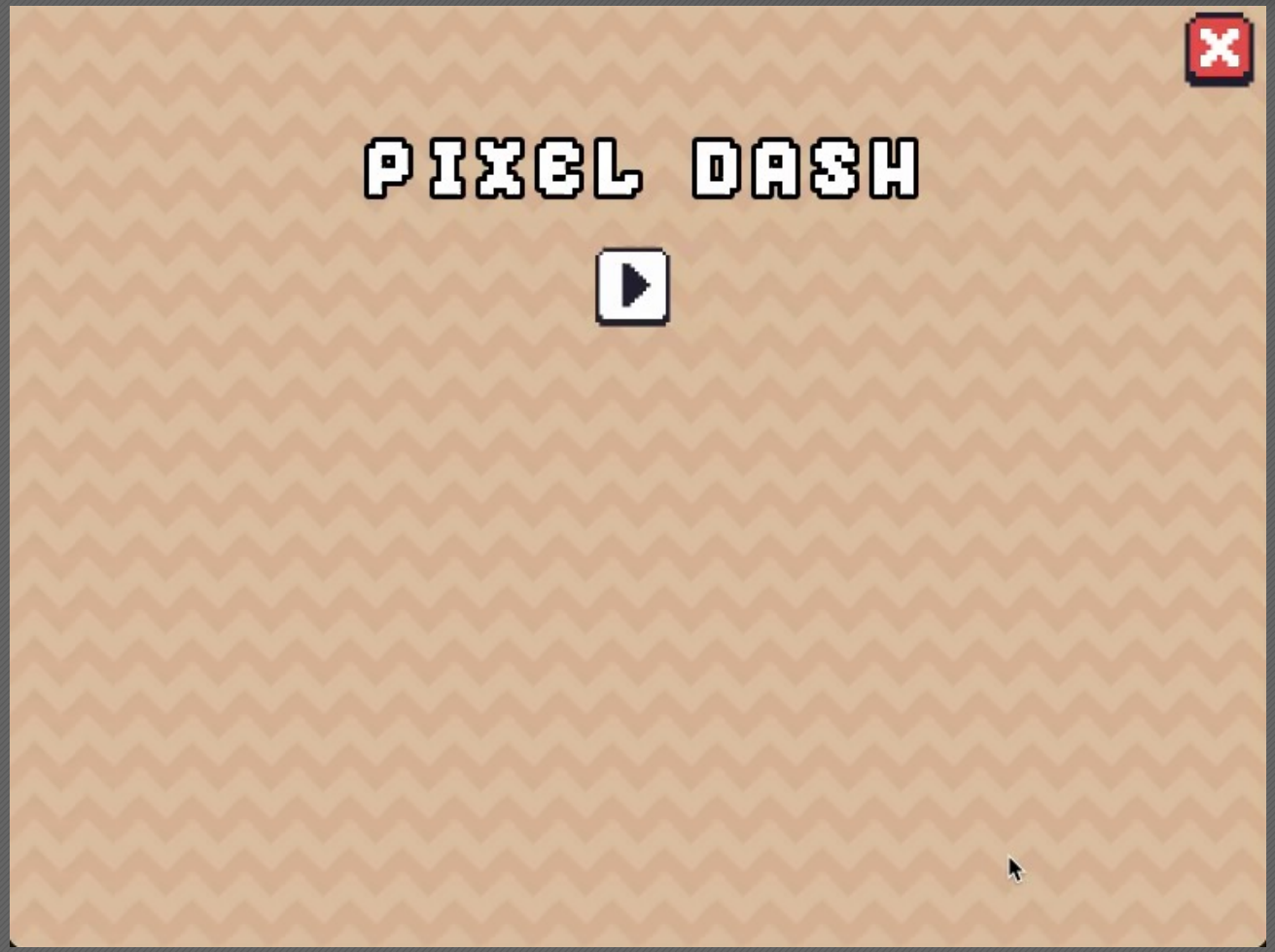


- Connect the PlayButton's pressed() signal with the so-far-empty MainMenu script.
- The change_scene_to_packed func is similar to the change_scene_to_file func except it takes a proper PackedScene type (which must be preloaded) instead of a String file path.

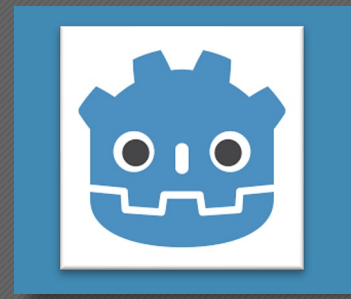
*Make sure you preload the .tscn scene file, **not** the .gd script file!*

```
1  extends Control
2
3  const level : PackedScene = preload("res://Scenes/Level/level.tscn")
4
5  func _on_play_button_pressed() -> void:
6      get_tree().change_scene_to_packed(level)
```

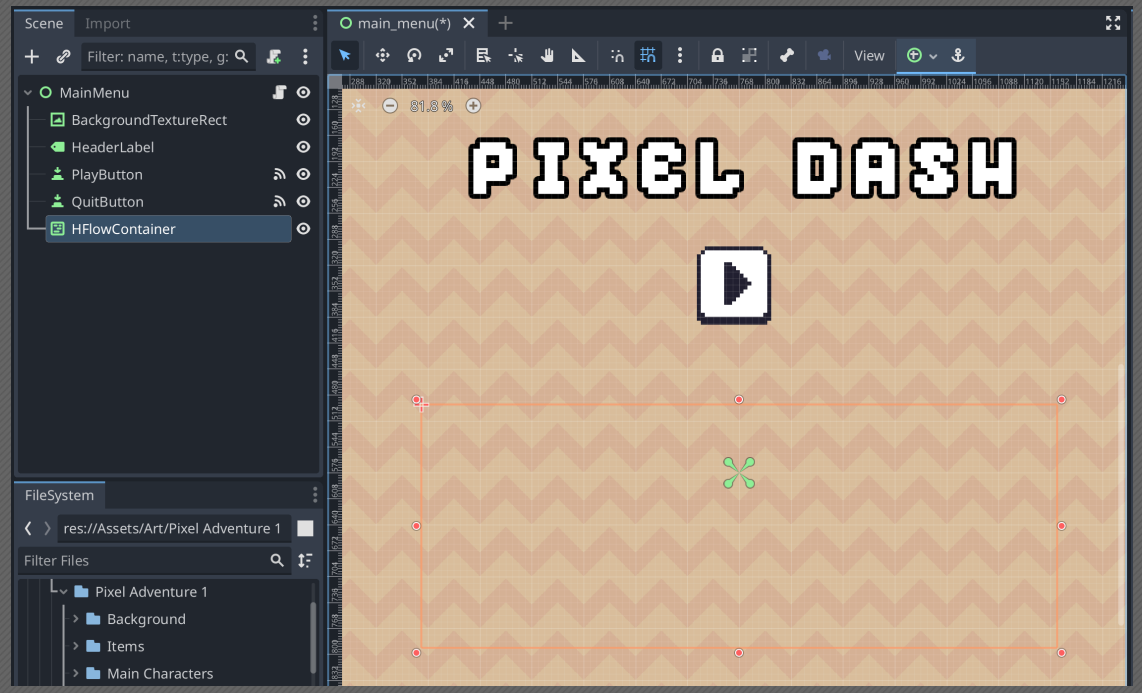
Test the PlayButton



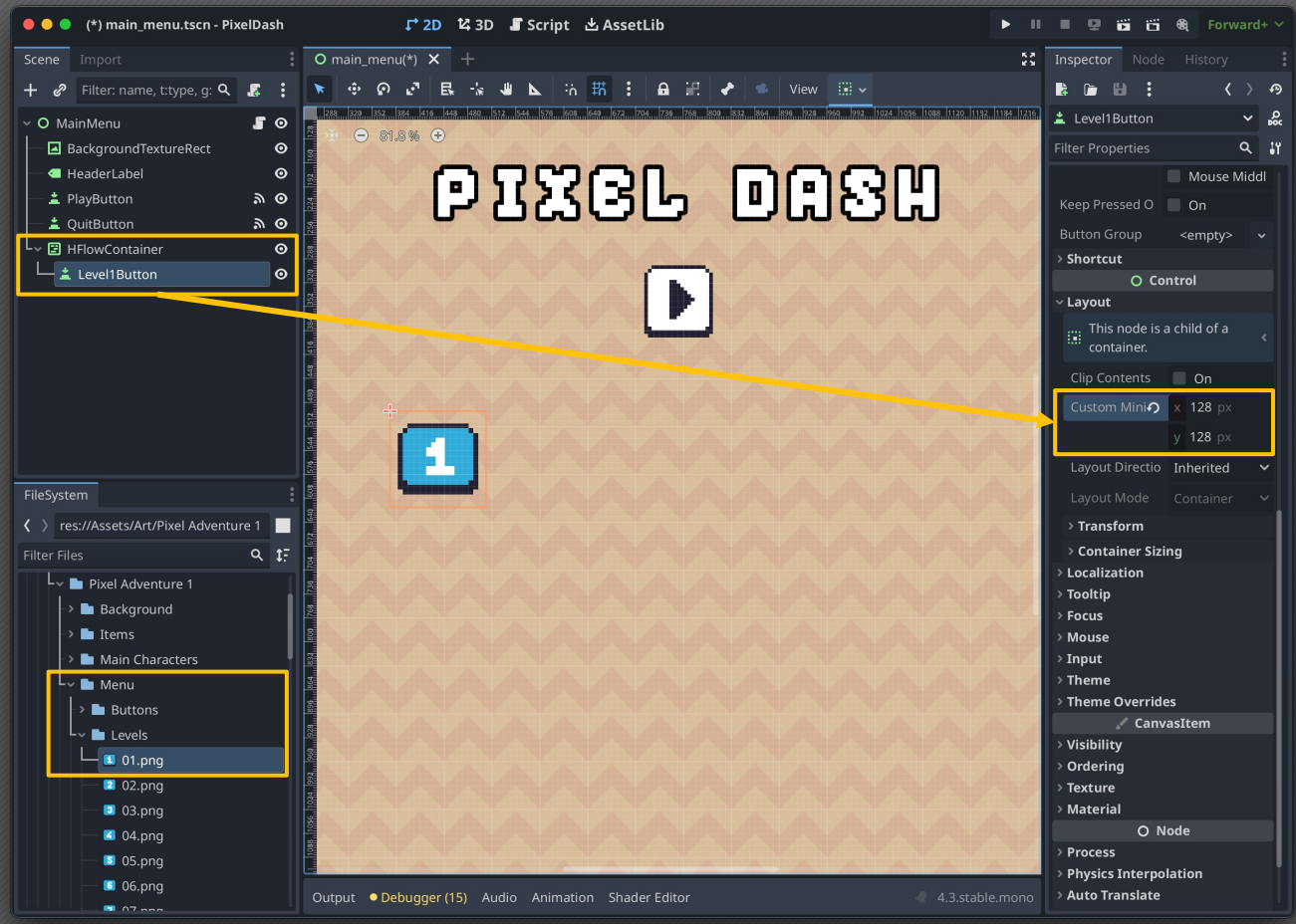
Adding Level Select Buttons



Add an HFlowContainer node, Anchored to Center and positioned comfortably under the PlayButton.

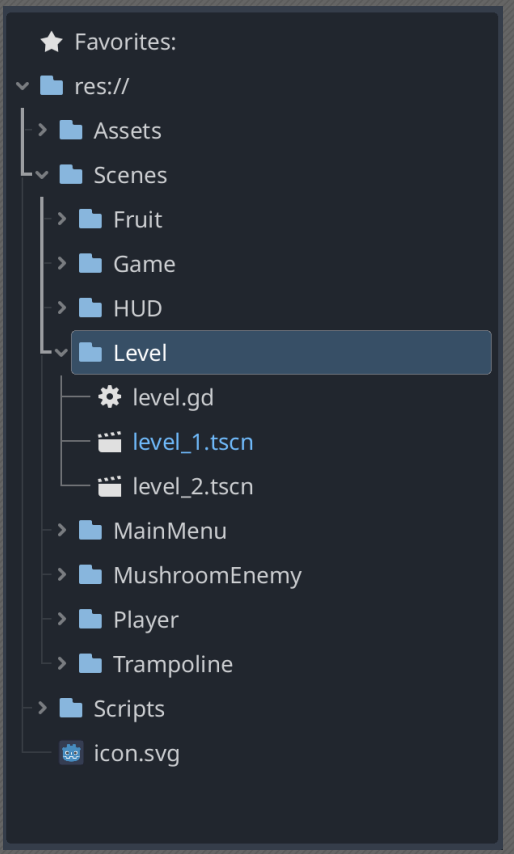


Add a button named Level1Button as a child of the HFlowContainer. Give the button a texture and min size.



Rename level and Duplicate to make level_2

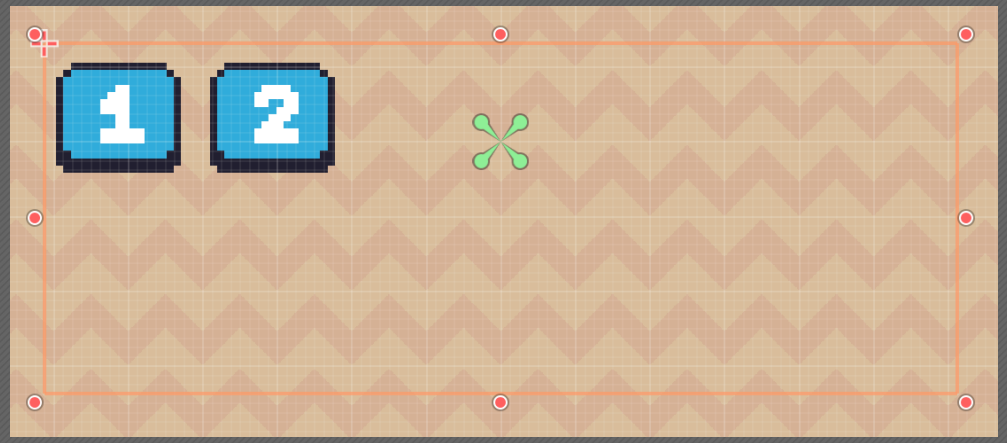
- Rename the “level” scene to “level_1”.
- Right-click the level_1.tscn scene file in the FileSystem and select “Duplicate”.
- Name the duplicated scene “level_2.tscn” and change the tiles, enemies, and trampolines in level_2 to be different than level_1.



Add a Level 2 Button to Main Menu

- Add a second button for Level 2 to the HFlowContainer in the MainMenu scene.

Notice the HFlowContainer automatically controls the position of the button.

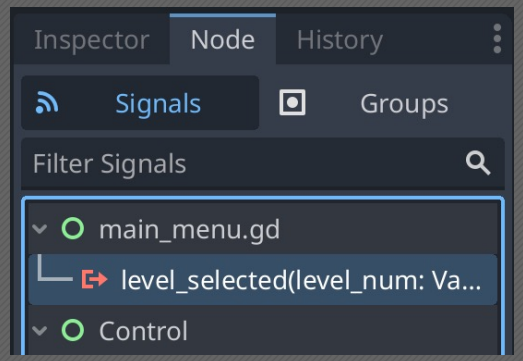


Add MainMenu Button Pressed Signal Handlers

- In the MainMenu script, define a new level_selected signal that takes a level_num arg.

- Add signal handlers to the level_1 and level_2 buttons for the pressed() signal, of which both functions emit the level_selected signal for its appropriate button level.

```
13  signal level_selected(level_num)
14
15  func _on_level_1_button_pressed() -> void:
16      level_selected.emit(1)
17
18
19  func _on_level_2_button_pressed() -> void:
20      level_selected.emit(2)
21
```

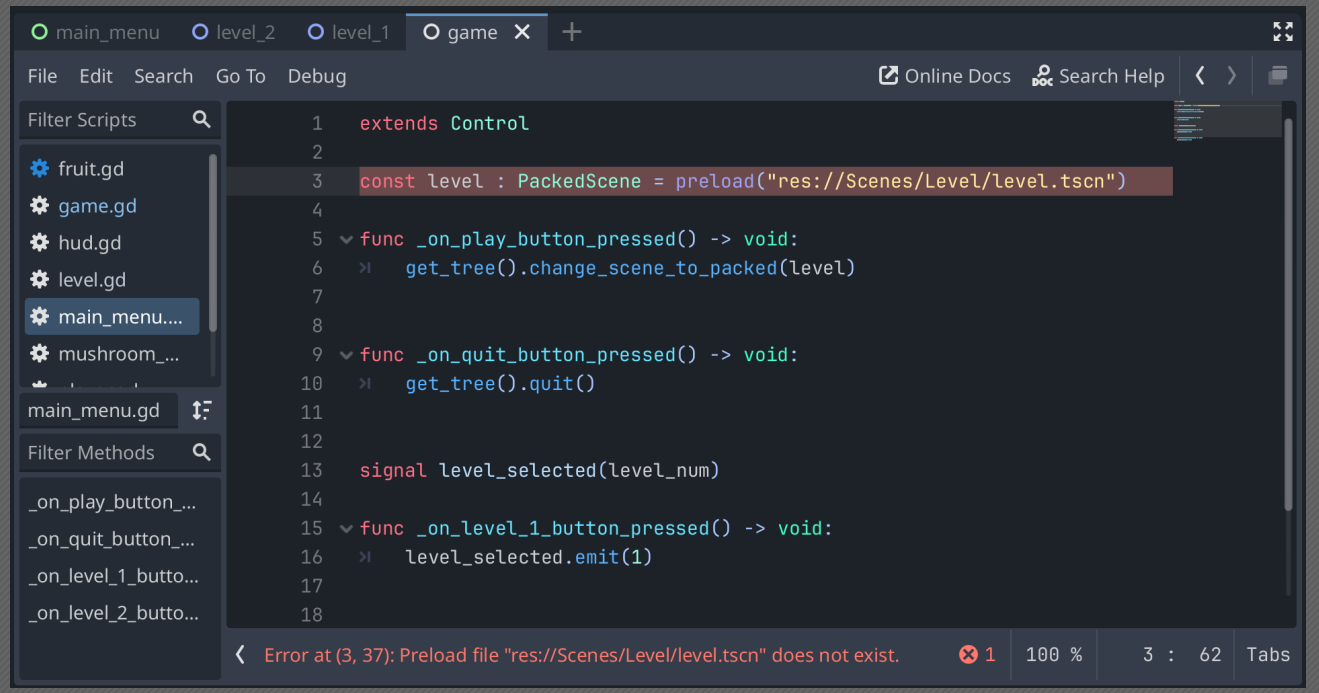


Custom signals are Accessible in the Node tab.

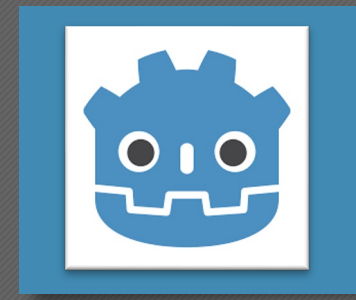
Fix MainMenu Script Error

Since the level scene was renamed to level_1, the preload for that file is no longer correct in the MainMenu script.

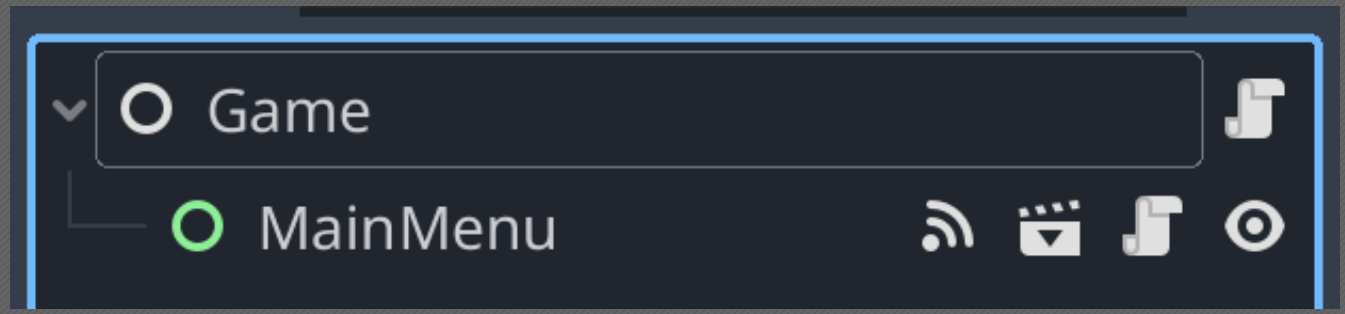
- Update the PackedScene preload call to preload the level_1 file instead of level.



Creating a Main “Game” Scene & Script



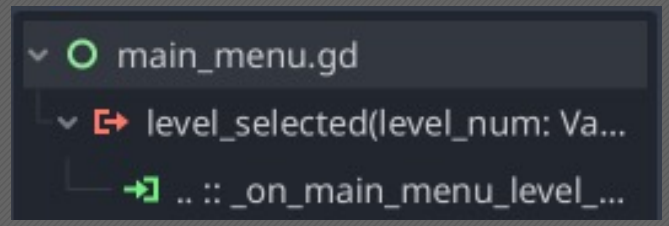
It would be helpful to have a main scene that can coordinate when to show/hide the main menu vs specific level scenes.



- Create a new Node scene named Game.

```
1  extends Node
2
3  var current_level_instance
4
5  func _on_main_menu_level_selected(level_num: Variant) -> void:
6      var next_level_scene = load("res://Scenes/Level/level_%d.tscn" % level_num)
7      current_level_instance = next_level_scene.instantiate()
8      add_child(current_level_instance)
9      $MainMenu.visible = false
```

- Add the MainMenu scene to the Game scene tree.



- Connect the MainMenu’s level_selected signal to the Game script.

Creating a Main “Game” Scene & Script

The load function is like preload, except it happens at runtime instead of on load, so it is less performant but good for loading more dynamic content.

```
1  extends Node
2
3  var current_level_instance
4
5  func _on_main_menu_level_selected(level_num: Variant) -> void:
6      var next_level_scene = load("res://Scenes/Level/level_%d.tscn" % level_num)
7      current_level_instance = next_level_scene.instantiate()
8      add_child(current_level_instance)
9      $MainMenu.visible = false
```

Levels load now, but don’t reset properly.

Update the Level Script

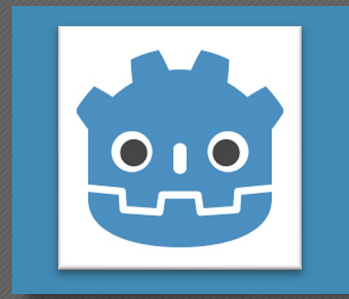
- Update the Level script to have a proper class_name.

- Add a level_done signal to the Level script.

The player_won variable won't be used immediately but can be used to "unlock the next level" as a stretch goal.

```
1  class_name Level
2  extends Node2D
3
4  signal level_done(player_won:bool)
5
6  var fruit_totals := {
7      "Kiwi": 0,
8      "Pineapple": 0
9  }
```

Update the Level Script



The current restart_level func reloads the level scene into the scene tree via String file path, which is no longer appropriate behavior.

- Update the Level script's restart_level func to instead emit the level_done signal.

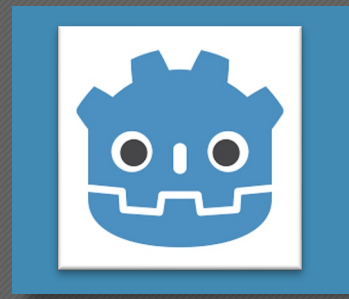
Old code

```
38 func restart_level() -> void:
39     get_tree().change_scene_to_file("res://Scenes/Level/level.tscn")
40
```

New code

```
36 func _on_level_music_finished() -> void:
37     $LevelMusic.play()
38
39
40 func restart_level() -> void:
41     level_done.emit(false)
42
```

Update the Game Script to handle level_done



The Game needs to handle the level_done signal, so that it knows to remove the current level and show the main menu.

- Update the Game script to keep track of the current level number.
- Connect Game script to handle the current level's finish_level signal, which will remove the level from the scene tree and show the MainMenu.

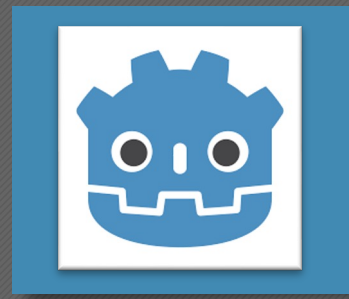
```
3   var current_level_instance : Level
4   var current_level_num : int
5
6   func _on_main_menu_level_selected(level_num: Variant) -> void:
7       var next_level_scene = load("res://Scenes/Level/level_%d.tscn" % level_num)
8       current_level_instance = next_level_scene.instantiate()
9       current_level_num = level_num
10      current_level_instance.connect('level_done', finish_level)
11      add_child(current_level_instance)
12      $MainMenu.visible = false
13
14
15  func finish_level(player_won:bool) -> void:
16      remove_child(current_level_instance)
17      $MainMenu.visible = true
```

The Final Product!

Ideas for Continued Tinkering

- Add “level complete” behavior that allows a player to properly “win” a level.
- Update the Game to only playing the “next level” when the current level is beaten (e.g. to unlock level 4, the player must beat level 3).
- Create more enemies! The Art assets folder has over a dozen unique enemies for you to get creative with.
- Create more traps, such as falling platforms or a Trampoline that bounces you into the air when you land on it.

Continued Learning



https://docs.godotengine.org/en/stable/index.html

The Godot docs are *actually* quite good! The Manual section is rich with descriptive use case and coding samples for common problems in game development (such as animations, save data, AI navigation, and networking).

YouTube

The HeartBeast and GDQuest are two very popular and high-quality channels dedicated to learning how to develop games in Godot.

https://www.youtube.com/@uheartbeast

https://www.youtube.com/@Gdquest

Final Lab & Tinkering Time

- Create a Main Menu scene that allows the Player to select which level is played.
- Create a Game scene which controls when the MainMenu shows, and which level is currently active in the scene tree.
- Congratulations for making it this far and creating your first game! Give yourself a pat on the back.