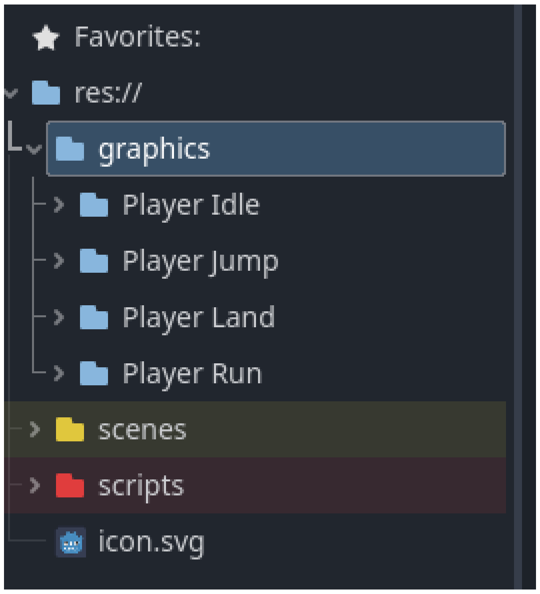
In Week3 we get familiar with working on graphics and animations.

Updating the Character

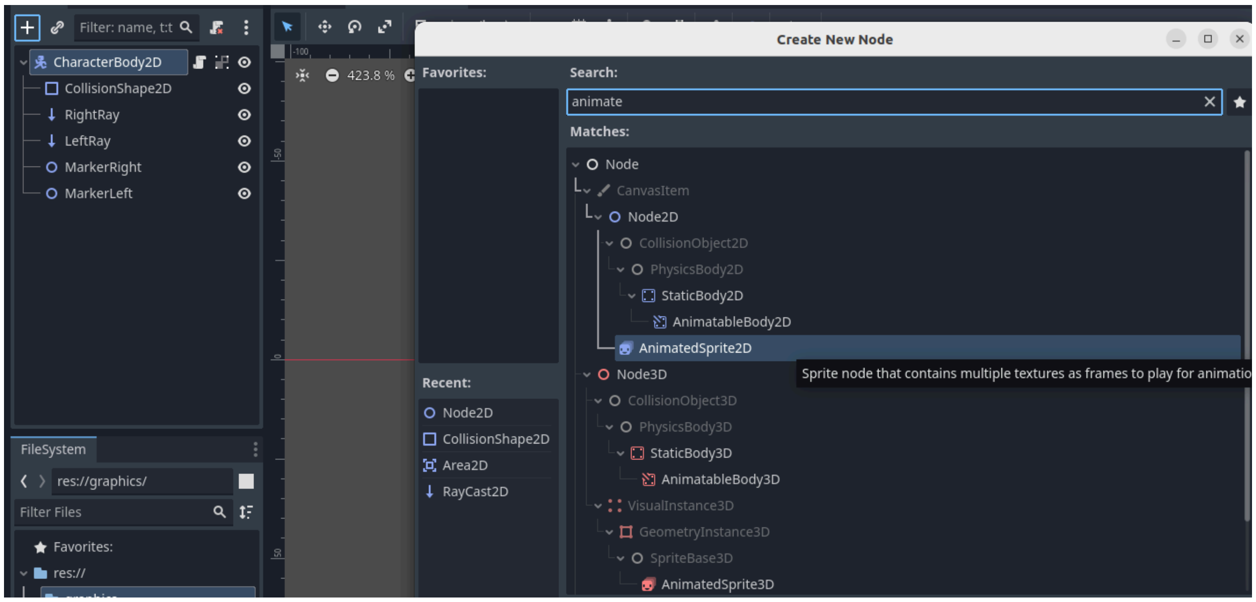
Up to now we have just had a simple collider for our character, but that will not do! In this update, we are going to employ animated sprite sheets to give our character some style.

Animated Sprites



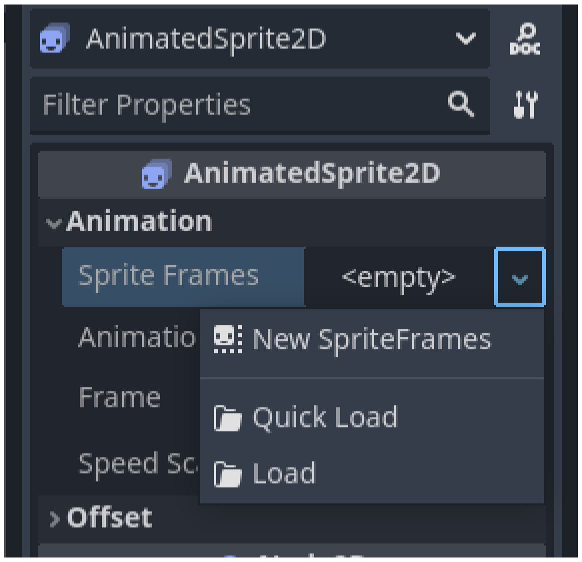
In our graphics folder, we can drag in our sprite sheets.

We then add an AnimatedSprite2D to our player.



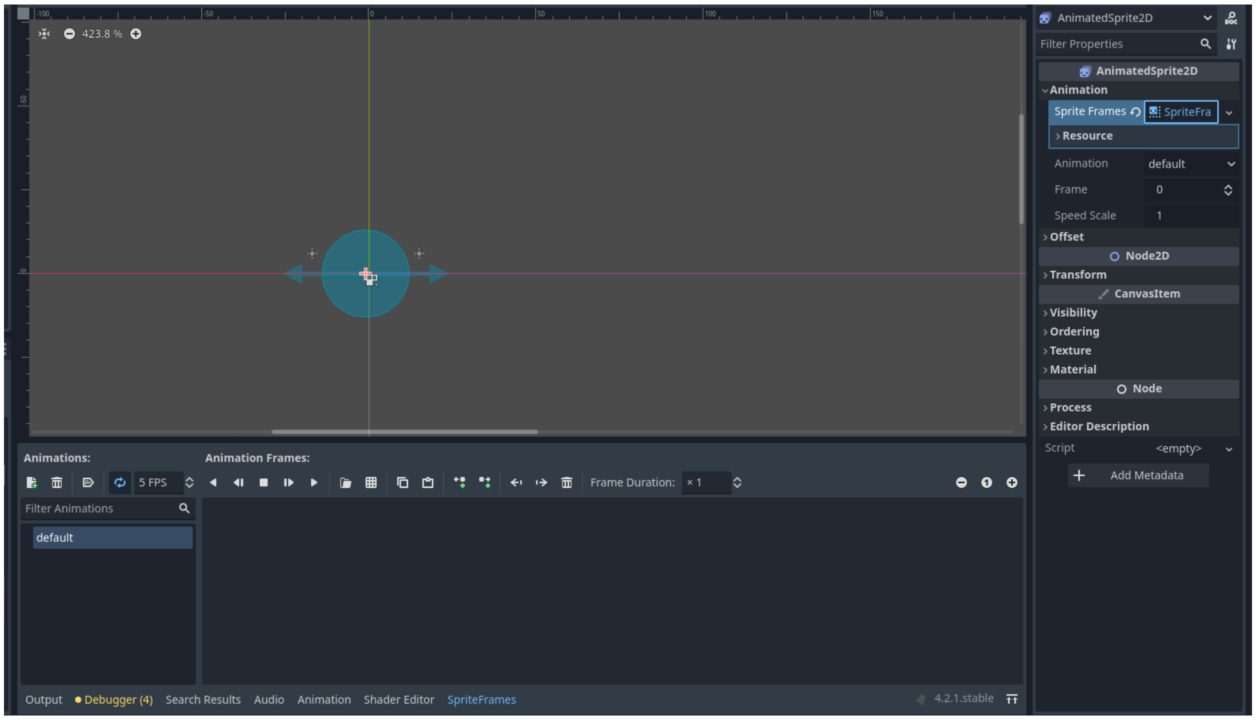
Sprite node that contains multiple textures as frames to play for animatio

Click on the AnimatedSprite2D and look for the “Animation” section in the inspector.

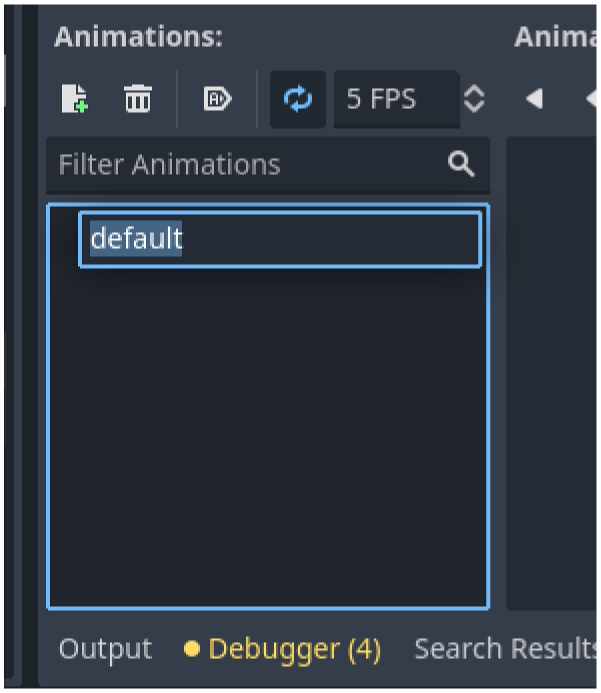


Expand “Animation” and click to add new spriteframes to the Sprite Frames property.

Then click on that new created SpriteFrames, and you’ll see a new editor appear in the workspace at the bottom of the screen:

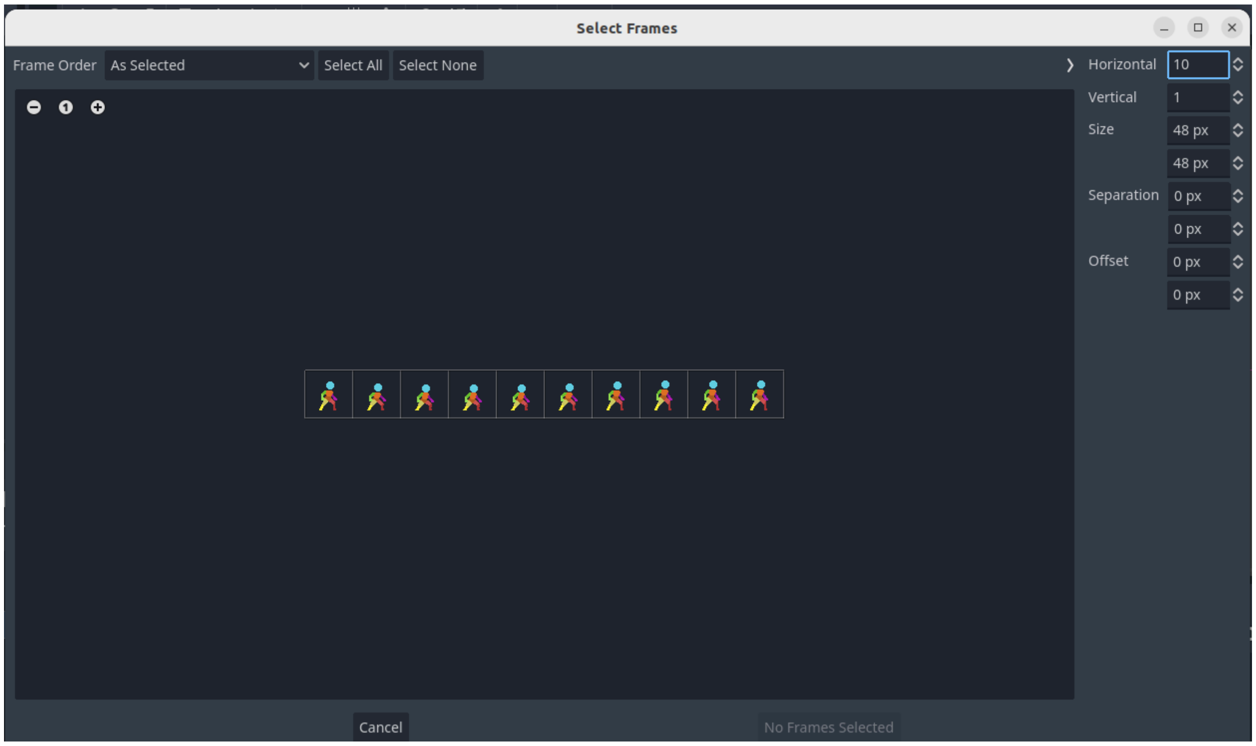


From here you can add a sprite sheet, or individual sprite to your animation.



Rename the default animation to “idle”. Then click on the sprite sheet importer, and select the idle animation from your graphics folder.

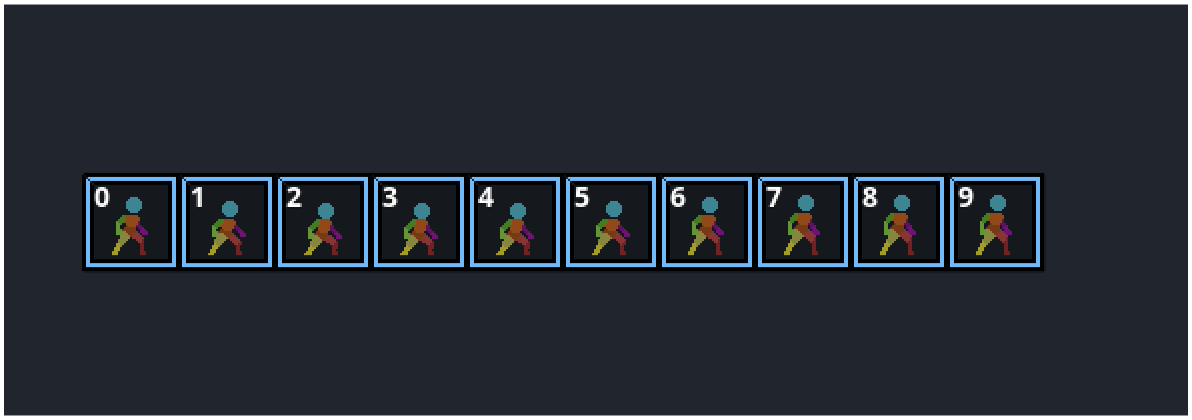
Update the horizontal and vertical properties until all the copies of your player animation are nicely framed.



Horizontal 10
Vertical 1
Size 48 px
48 px
Separation 0 px
0 px
Offset 0 px
0 px

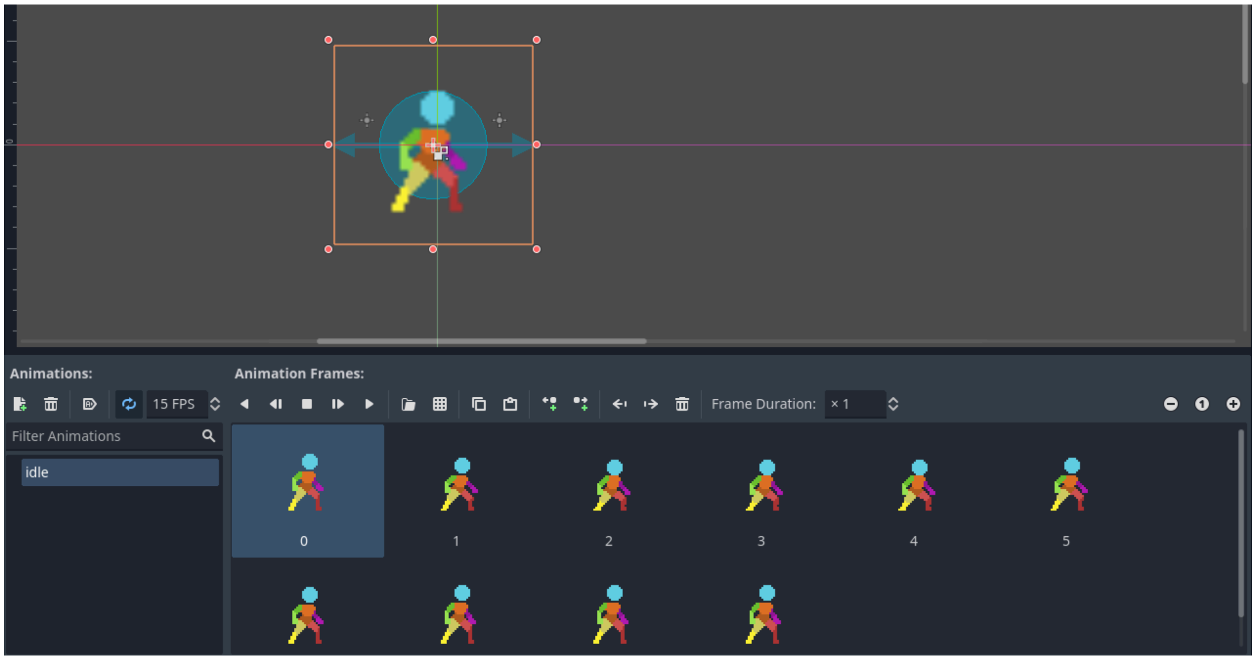
You’ll notice when you do so that the size properties will match the 48 pixels your character has been designed with.

Now select all the frames you would like to include in the “idle” animation.

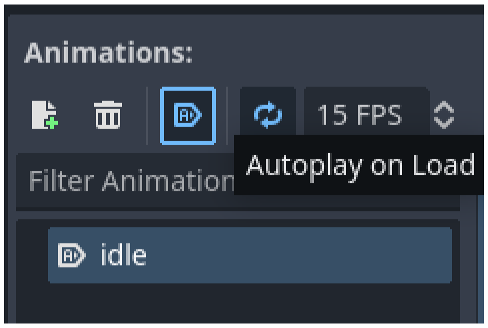


Select “add frames” to include them.

You will see them appear in the editor!

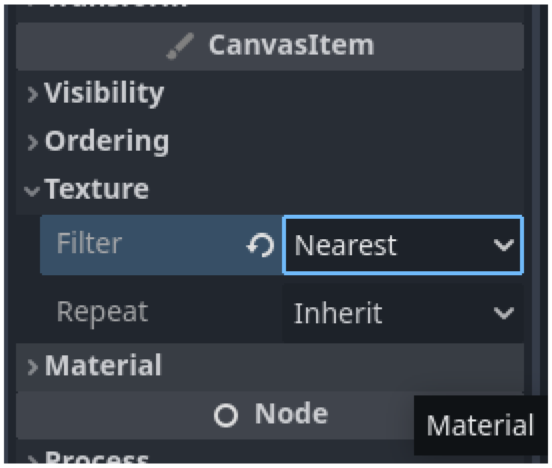


Set autorun to true, and the fps to 15:



You will notice that your graphics may look a bit fuzzy. This is because the graphics renderer is using a setting preferred for non-pixel graphics that interpolates the edges. We can change that at the level of the graphic, or as a project setting.

With the animated sprite 2D selected, look in the inspector under “canvas item” and change the renderer’s texture filter to “nearest”



Filter: Nearest
Repeat: Inherit

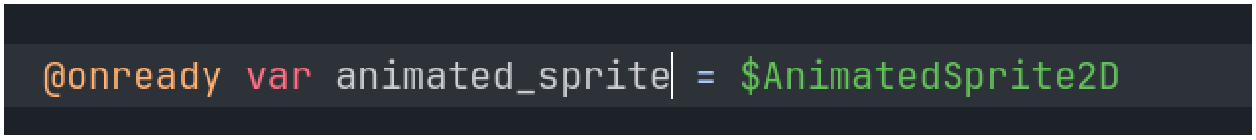
Test your game to see how the animation looks!

Exercise: Add animations for “run” and “jump”.

UPDATING THE CODE

The AnimatedSprite2D provides a simple way to control animation by playing the various states we have created (the “idle”, “run” and “jump” animations)

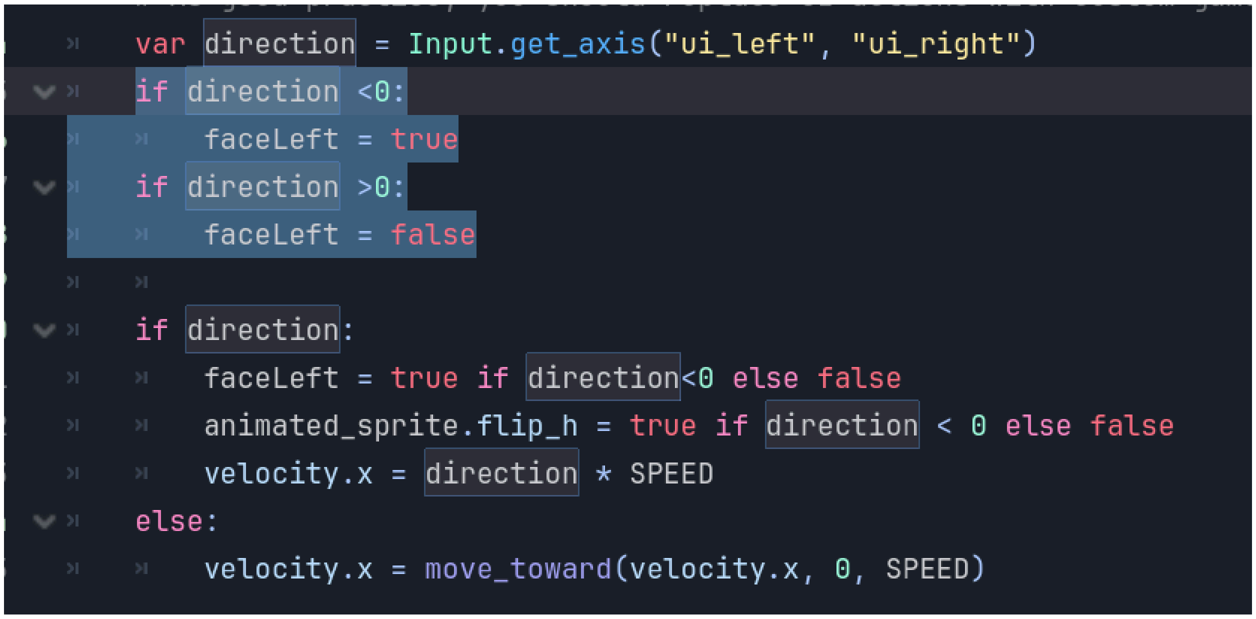
First we need to get a reference to the sprite in our player script, by dragging and control-dropping it into our code.



```
@onready var animated_sprite = $AnimatedSprite2D
```

Now let’s get our character facing the correct direction. A common way to do this is to flip the sprite horizontally based on what direction the player has moved in.

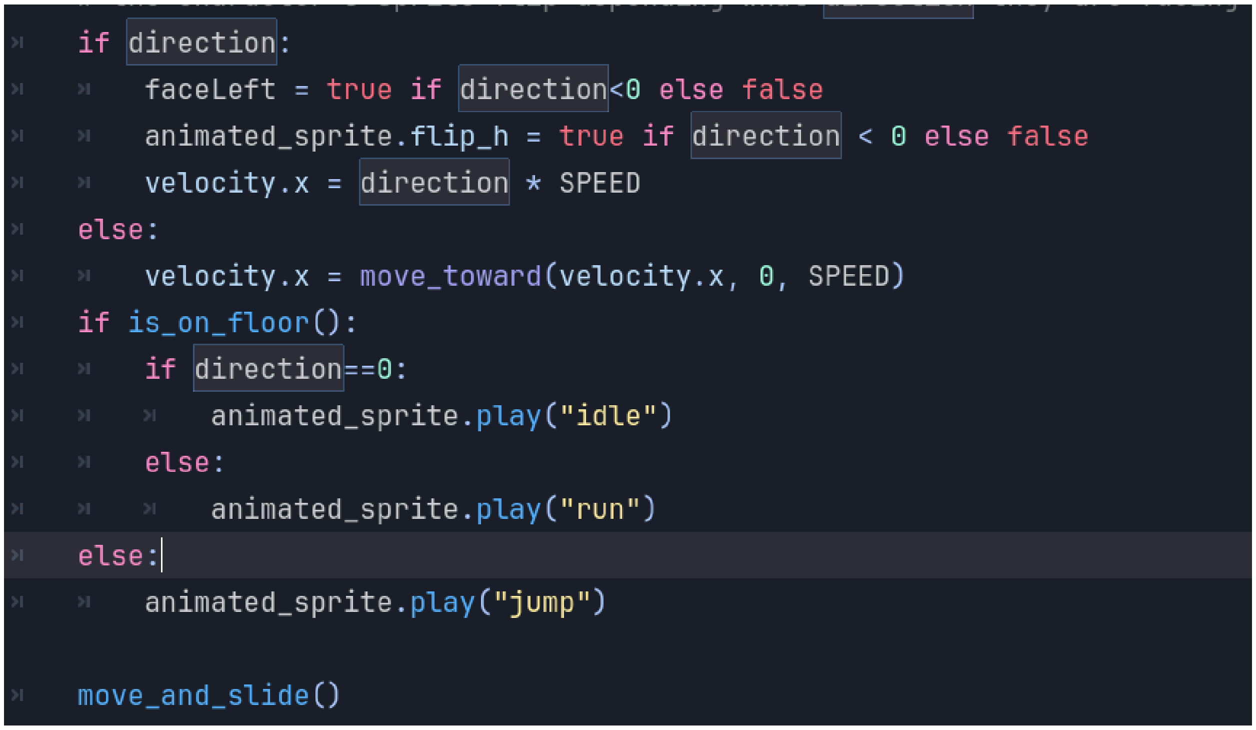
We can replace the highlighted code:



```
var direction = Input.get_axis("ui_left", "ui_right")
if direction <0:
    faceLeft = true
if direction >0:
    faceLeft = false

if direction:
    faceLeft = true if direction<0 else false
    animated_sprite.flip_h = true if direction < 0 else false
    velocity.x = direction * SPEED
else:
    velocity.x = move_toward(velocity.x, 0, SPEED)
```

This is already looking better! Let’s get the other animation states in.



```
if direction:
    faceLeft = true if direction<0 else false
    animated_sprite.flip_h = true if direction < 0 else false
    velocity.x = direction * SPEED
else:
    velocity.x = move_toward(velocity.x, 0, SPEED)
if is_on_floor():
    if direction==0:
        animated_sprite.play("idle")
    else:
        animated_sprite.play("run")
else:
    animated_sprite.play("jump")

move_and_slide()
```

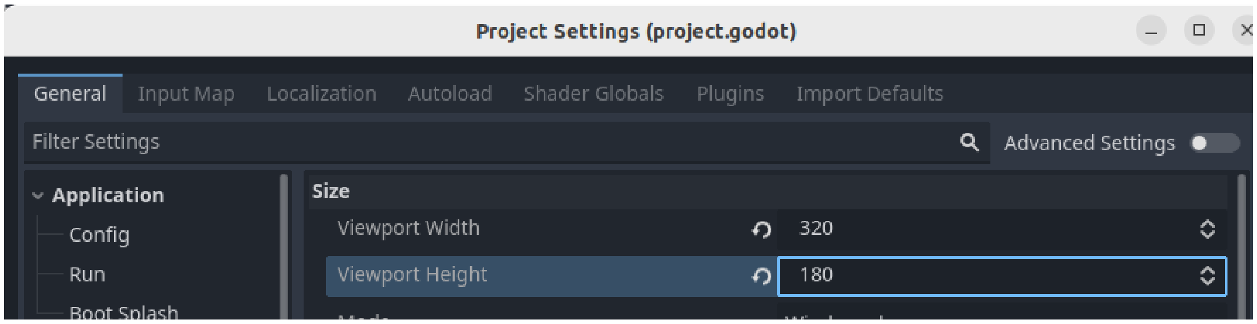
This code looks at whether the player is on the floor. If they are, they must be either in the “idle” or “run” states. If they are not on the floor, they are in the air and we can go to the “jump” state.

Great! We now have animated sprites that reflect the state of the player. You could continue on to include any number of animations, such as an attack with either their melee or ranged.

Before we leave the player character, there are a couple more things we could do.

Adjusting The Screen Sizing

It is very common in pixel based games to render the screen smaller, then scale it up to the play size we want.


Project Settings (project.godot)

General | Input Map | Localization | Autoload | Shader Globals | Plugins | Import Defaults

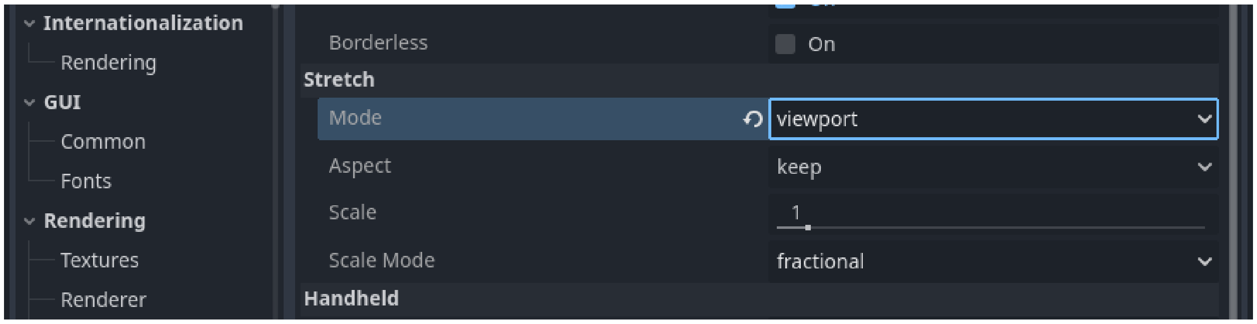
Filter Settings | Advanced Settings

Application
- Config
- Run
- Boot Splash

Size

Viewport Width	320
Viewport Height	180

Set Stretch Mode to "viewport"


Internationalization
- Rendering

GUI
- Common
- Fonts

Rendering
- Textures
- Renderer

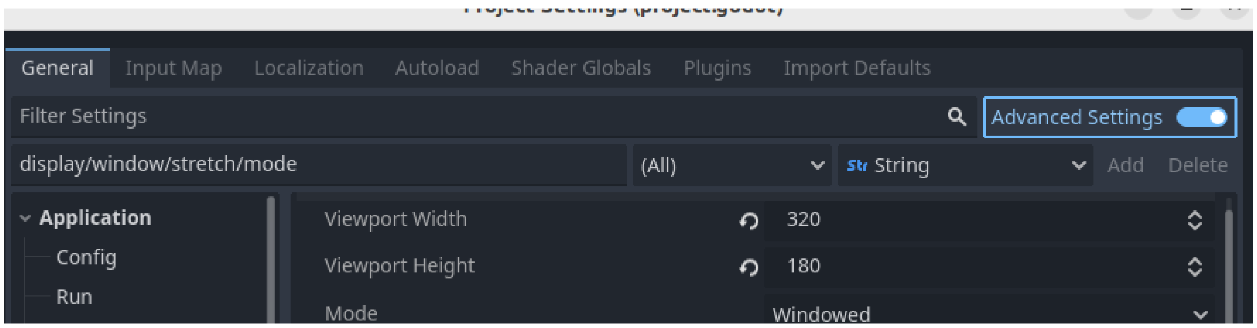
Borderless	On

Stretch

Mode	viewport
Aspect	keep
Scale	1
Scale Mode	fractional

Handheld

Turn on "advanced settings"


General | Input Map | Localization | Autoload | Shader Globals | Plugins | Import Defaults

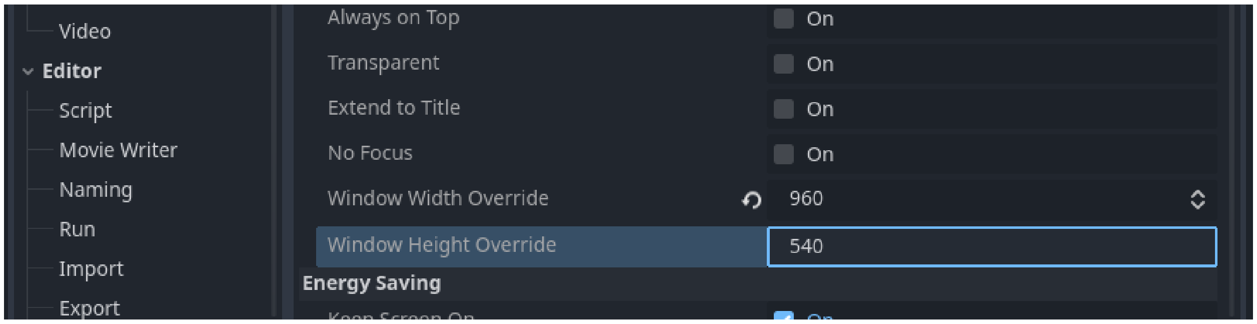
Filter Settings | Advanced Settings

display/window/stretch/mode | (All) | String | Add | Delete

Application
- Config
- Run

Viewport Width	320
Viewport Height	180
Mode	Windowed

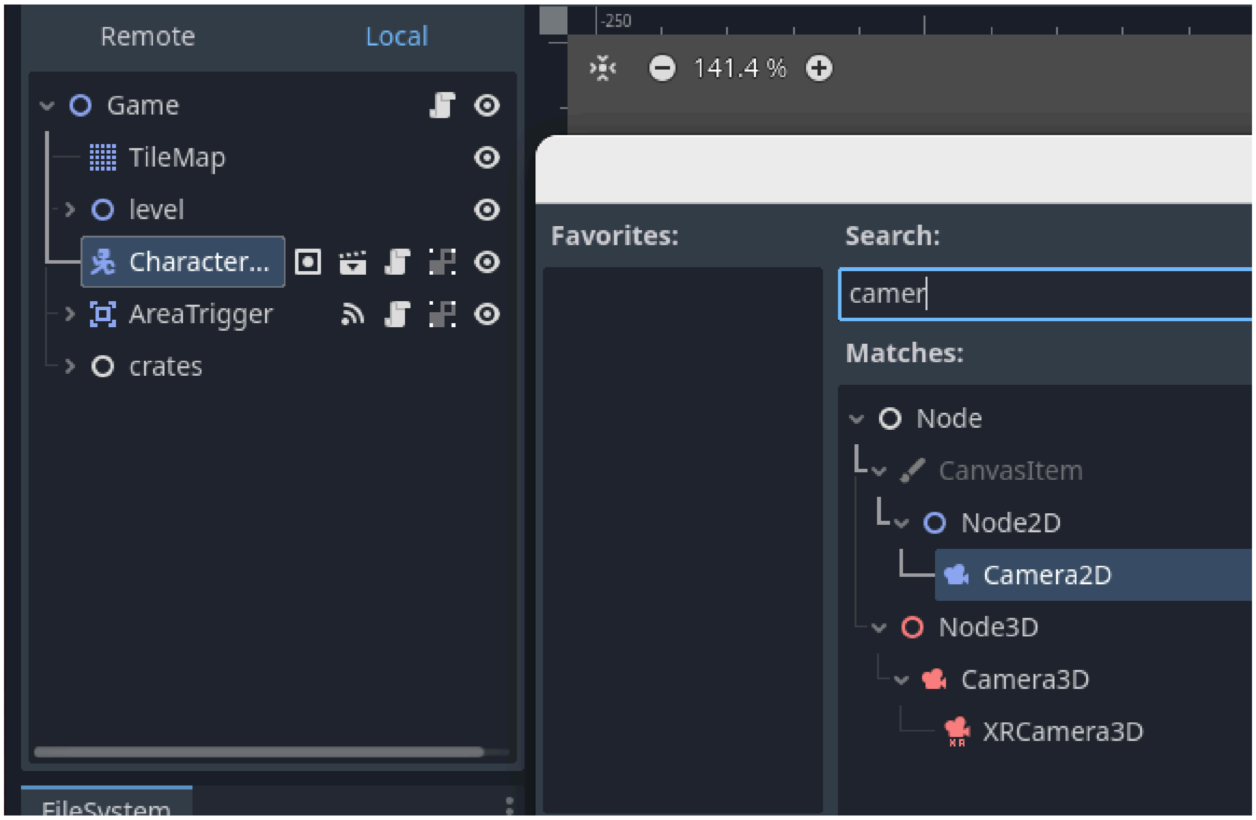
Set the desired screen size (960x540):



Always on Top	On
Transparent	On
Extend to Title	On
No Focus	On
Window Width Override	960
Window Height Override	540

Energy Saving

Now we can add a camera to the player that will follow their movement, and our world will really start to become interesting.



By default the only camera added to the game will become the camera that renders the screen.