

So you want to create a new character for your game? Here's how to do that!

I chose Moguera as an example for this tutorial.

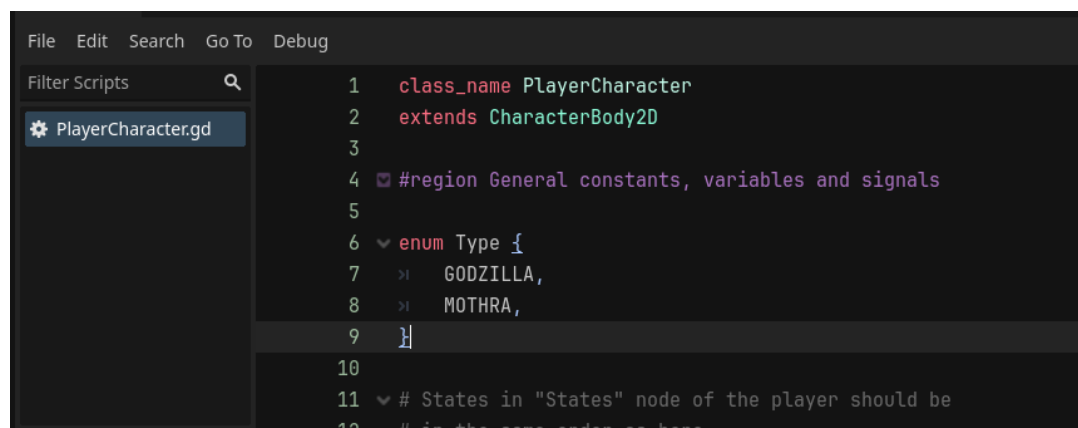
## Contents

New character slot.....	1
Creating new character skin .....	3
Preparing sprites.....	8
Importing sprites into Godot.....	11
Make the skin load .....	19
Other character parameters .....	21
Testing our new character .....	23
Fixing skin position and animation .....	25
Board piece .....	29
Changing the character's collision box (optional) .....	33
Using different SFX for level intro (optional).....	35
Making the character fly (optional).....	36
Separating the top half of the character sprites (optional) .....	38

## New character slot

First, let's add a slot for our new character. The character types are stored in the characters' script, "PlayerCharacter.gd", it's contained in "Scripts/Objects/Characters" folder.

If you open the script in the Godot script editor and scroll to the very top you will immediately notice the enumeration that contains every character type:

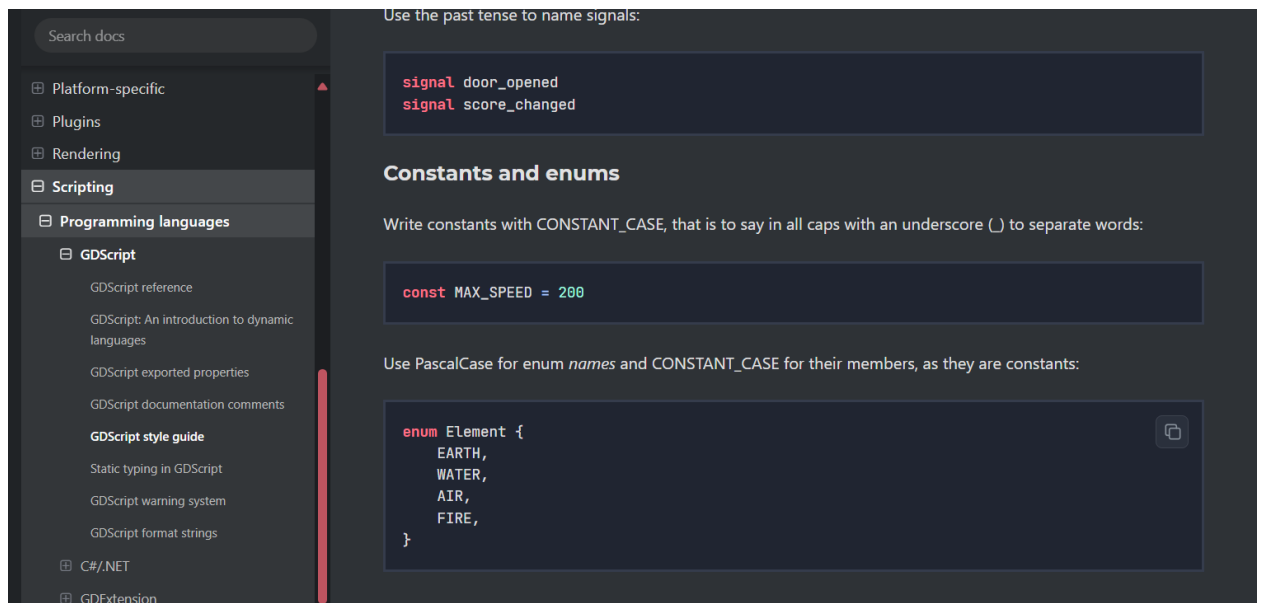


Let's add a new character type! Create a new line after the last character ("MOTHRA," in this case) and put in similar text but for your character's name (if your character's name has multiple words, like "Mecha Godzilla", it's

recommended to use underscores, like “MECHA\_GODZILLA”, don’t use spaces because Godot would be confused), in our case it’s “MOGUERA,”, like so:

```
4  ☒ #region General constants
5
6  ☒ enum Type {
7      >|  GODZILLA,
8      >|  MOTHRA,
9      >|  MOGUERA,
10 }
11
```

You’re probably wondering why we are using uppercase letters for character names, well, it’s just a convention: if we declare something constant (in our case, the character type will be the same throughout the entire game, so it’s constant, in other constants can be something like the value of pi, or how much damage should an attack do) we use uppercase letters and underscores (“\_”) if we want to separate several words. This convention is applicable to many languages, including GDScript (screenshot from Godot documentation):



The screenshot shows the Godot documentation website. On the left is a sidebar with a search bar and a list of categories: Platform-specific, Plugins, Rendering, Scripting, and Programming languages. Under Programming languages, GDScript is selected, showing a list of links including GDScript reference, GDScript: An introduction to dynamic languages, GDScript exported properties, GDScript documentation comments, GDScript style guide (which is highlighted), Static typing in GDScript, GDScript warning system, and GDScript format strings. The main content area is titled 'Constants and enums' and includes instructions on naming conventions. It shows examples of signal names, constants, and enums.

Search docs

- Platform-specific
- Plugins
- Rendering
- Scripting
- Programming languages
  - GDScript
    - GDScript reference
    - GDScript: An introduction to dynamic languages
    - GDScript exported properties
    - GDScript documentation comments
    - GDScript style guide**
    - Static typing in GDScript
    - GDScript warning system
    - GDScript format strings
  - C#/NET
  - GDExtension

Use the past tense to name signals:

```
signal door_opened
signal score_changed
```

### Constants and enums

Write constants with CONSTANT\_CASE, that is to say in all caps with an underscore (\_) to separate words:

```
const MAX_SPEED = 200
```

Use PascalCase for enum *names* and CONSTANT\_CASE for their members, as they are constants:

```
enum Element {
    EARTH,
    WATER,
    AIR,
    FIRE,
}
```

And also Java, for example:

## 2. Basics

A constant is a variable whose value won't change after it's been defined.

Let's look at the basics for defining a constant:

```
private static final int OUR_CONSTANT = 1;
```

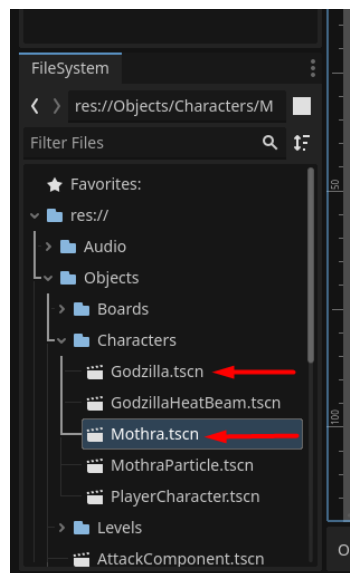


Some of the patterns we'll look at will address the *public* or *private* access modifier decision. We make our constants *static* and *final* and give them an appropriate type, whether that's a Java primitive, a class, or an *enum*. **The name should be all capital letters with the words separated by underscores**, sometimes known as screaming snake case. Finally, we provide the value itself.

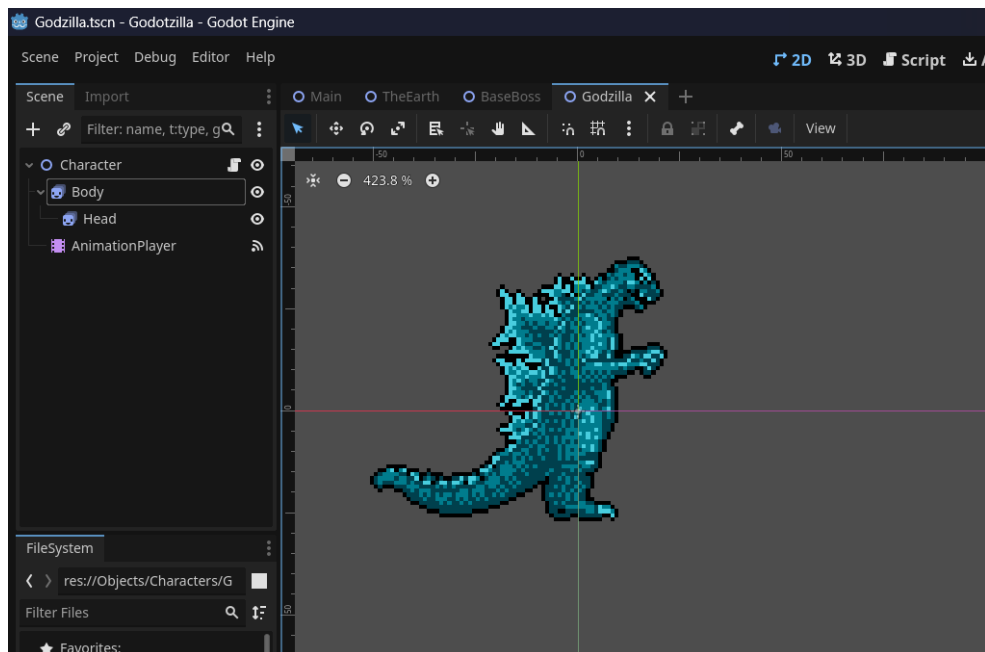
## Creating new character skin

Alright, we made our new slot. What now? Let's create a new character skin for our character! How are we going to do that? Easy, but first let's find the existing skins for Godzilla and Mothra and see how they work.

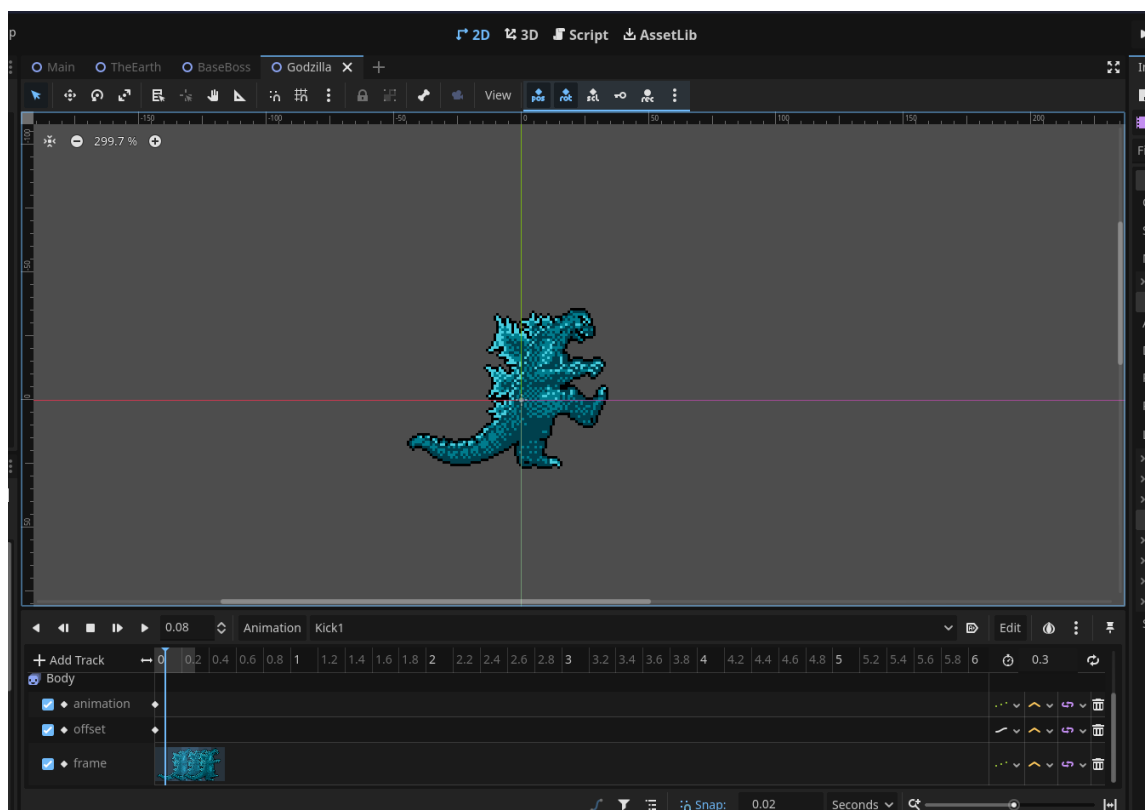
The character skins are in “Objects/Characters” folder:



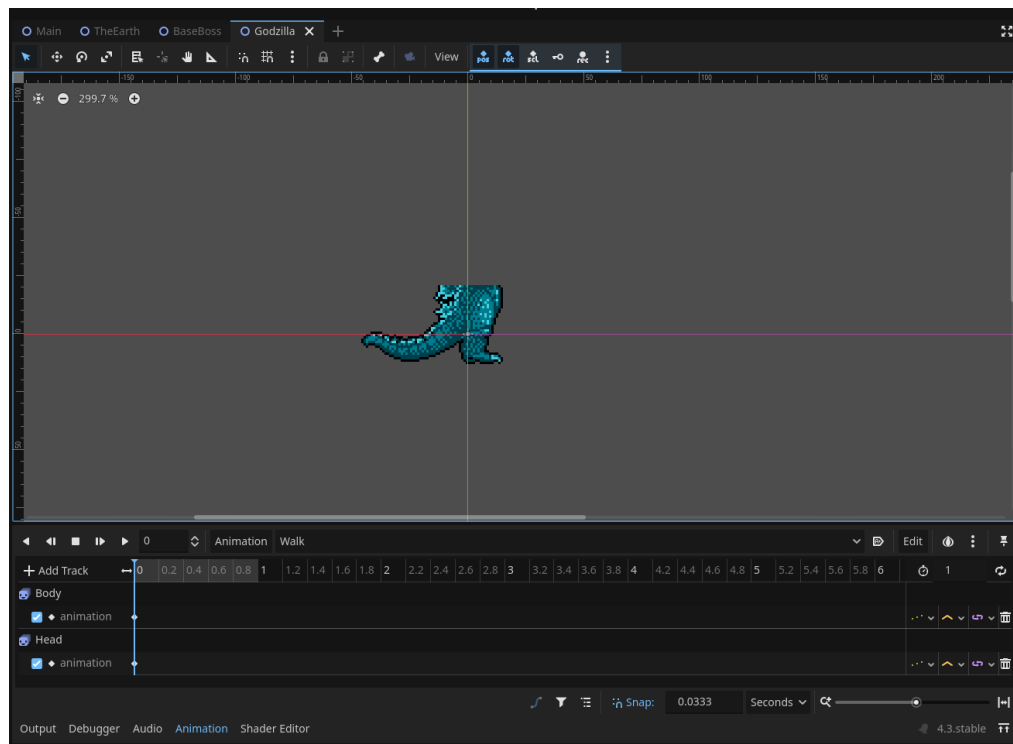
Now let's open “Godzilla.tscn”



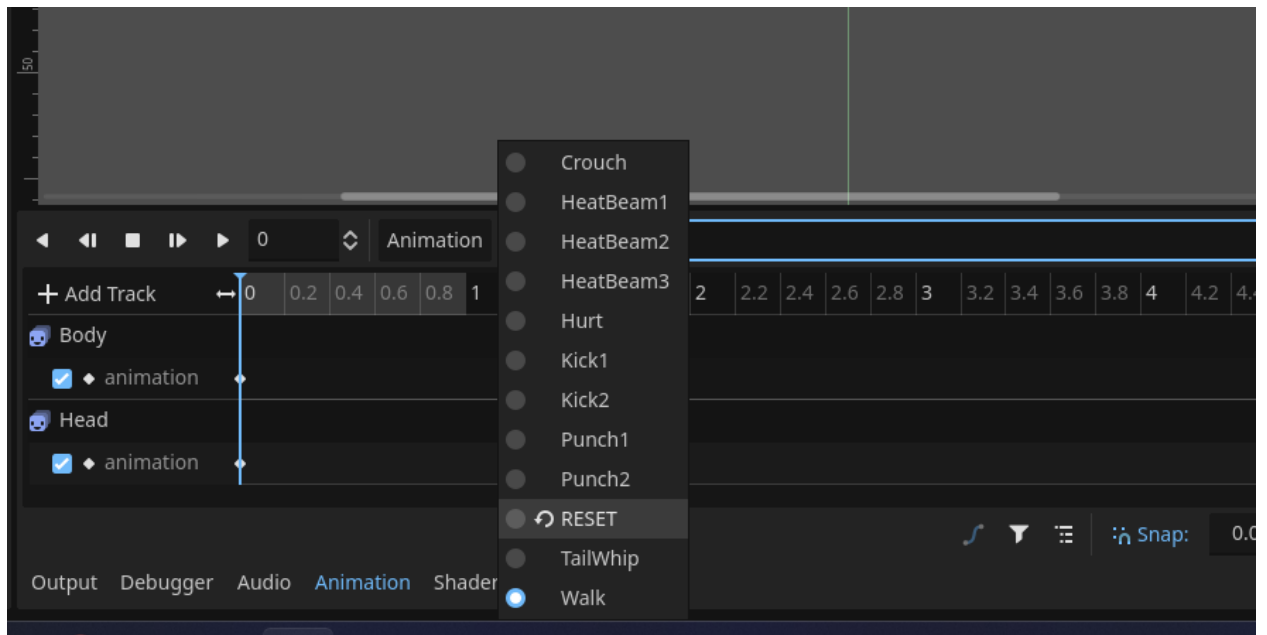
We can see that there's a root node called "Character", several sprite nodes and an animation player. There are 2 sprite nodes because they need to be separate (think about attacking while walking or while standing, in that case the sprites would contain a combination of every top part of Godzilla's sprites attacking and the bottom one walking if they weren't split, that sounds like a mess, so we separate them). The animation player, as you may guess, is responsible for Godzilla's animations:



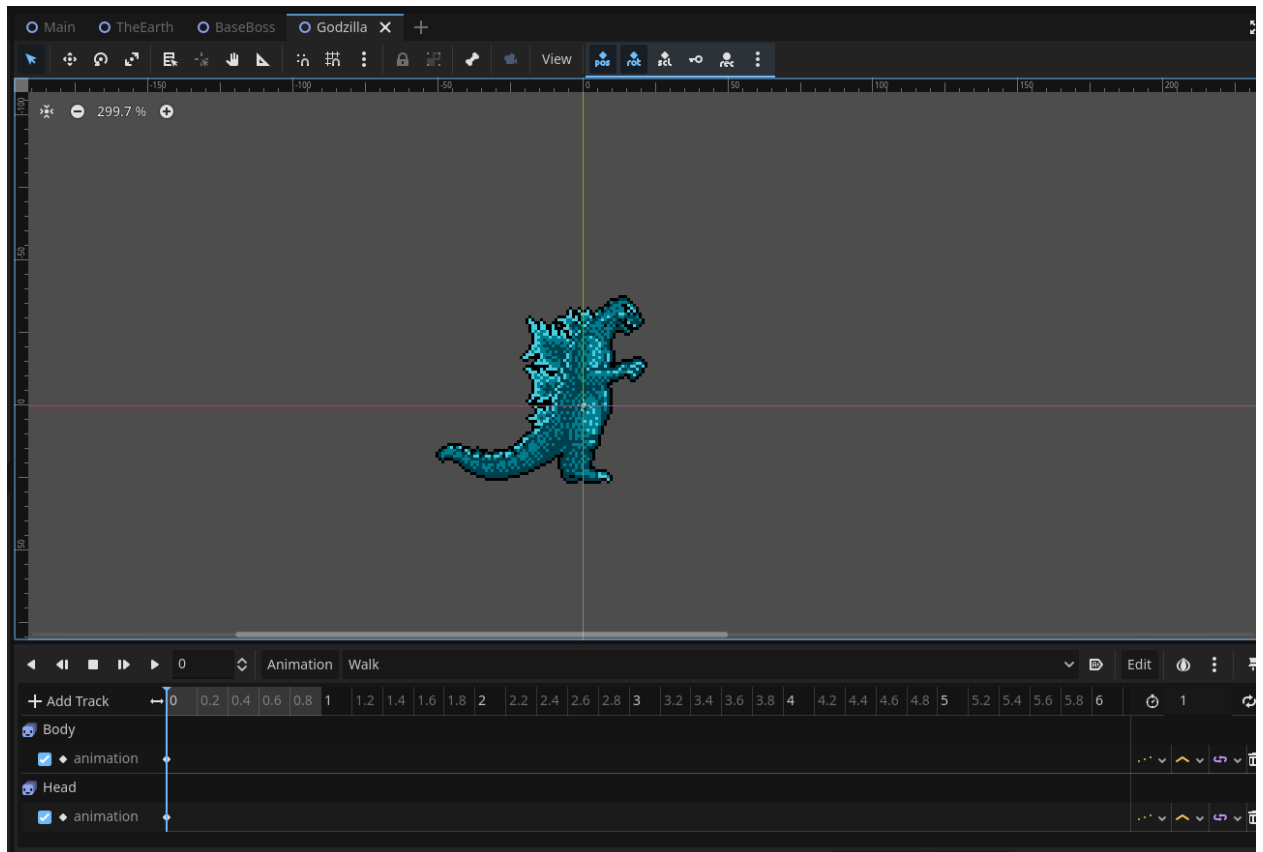
Sometimes if you keep playing with that by selecting various animations you may notice that the top part of the sprites becomes invisible



But don't worry, you can reset everything by selecting "RESET" animation:

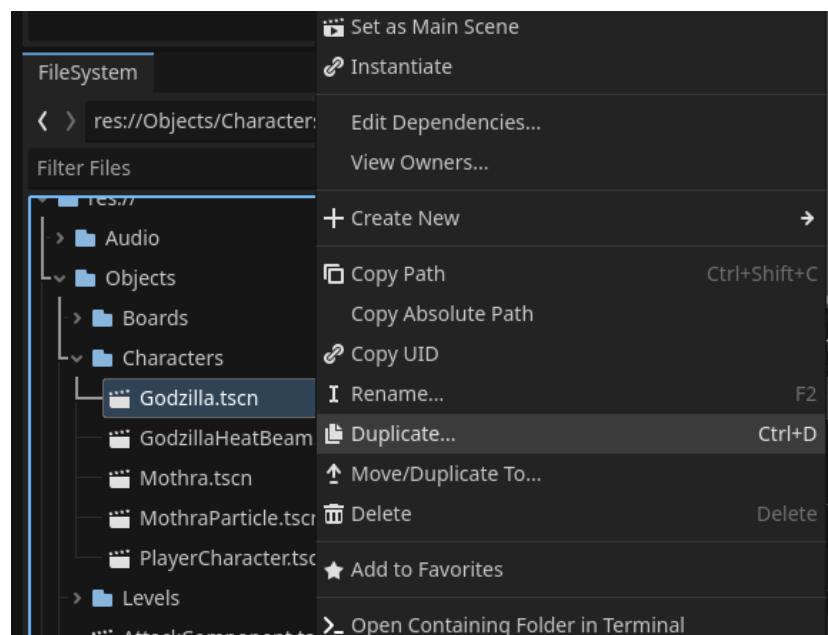


You can also notice that the walking animation doesn't have any frames besides the first one:

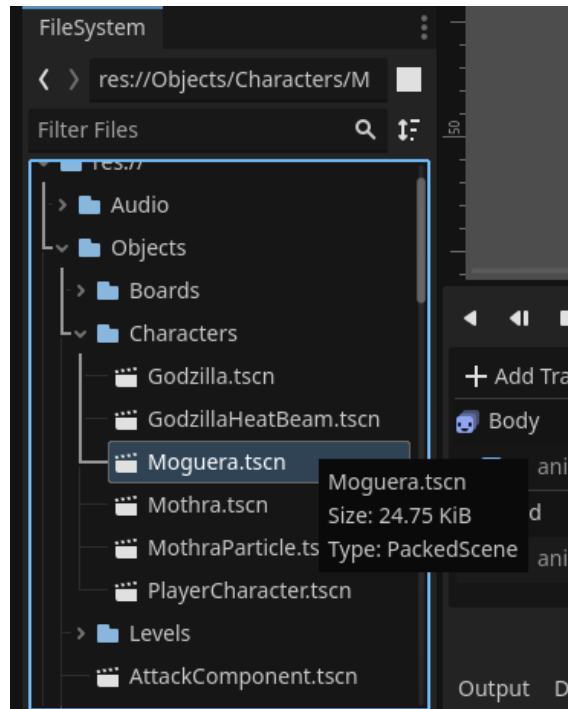


That's intentional, we change the walking frame using code, not with animations (if you're curious about how it works, you can check out line 52 in "Scripts/Objects/Characters/Walk.gd").

So, how do we actually create a new skin? Well, we can duplicate the already existing one and start from there:



There we go:



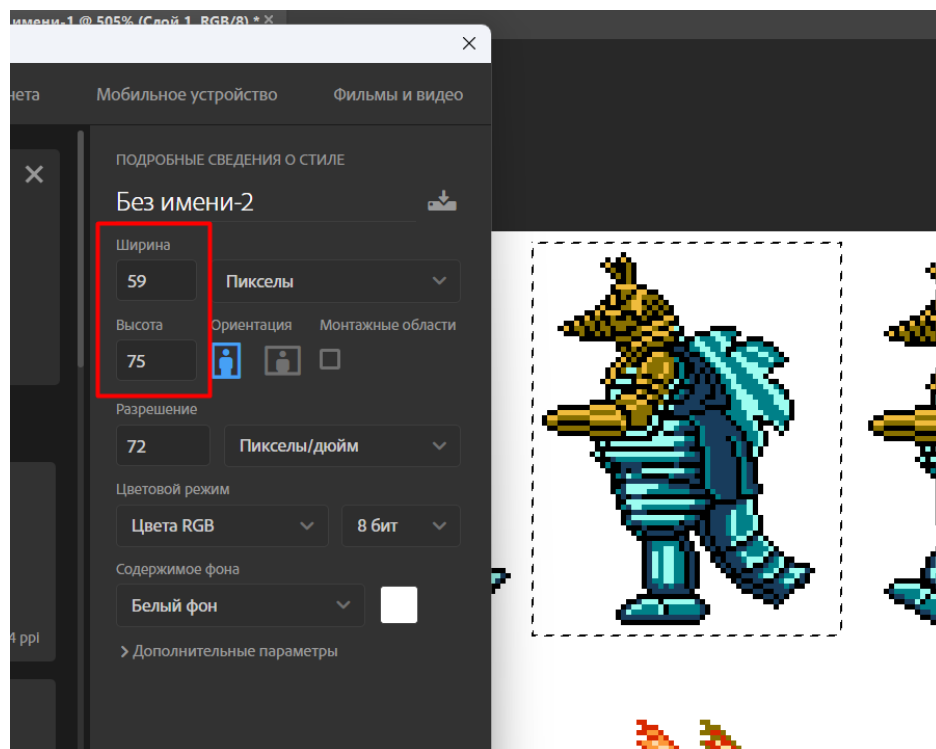
## Preparing sprites

Meanwhile we can prepare the sprites for our character. I will be using Photoshop for that, and thanks to FreKay Planet for ripping the Moguera sprites from Godzilla: Monster of Monsters.



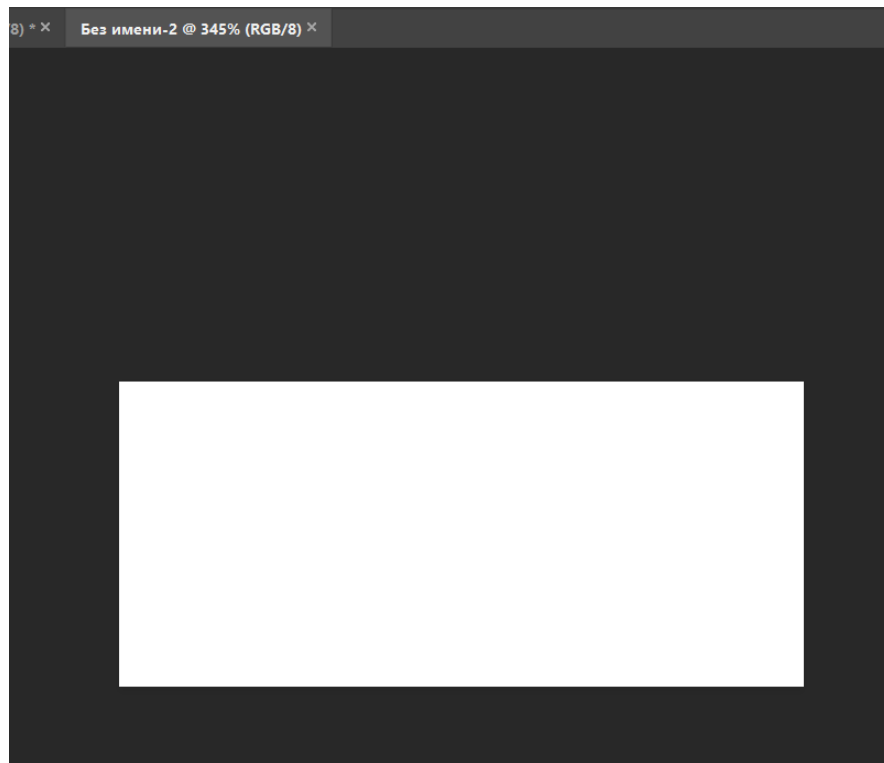
Since Moguera doesn't have any special animations with its top part of the sprites we can omit the step of preparing separate sprites for top and bottom parts of the sprites.

Now let's figure out the bounding box for the sprites. I roughly made a selection over a sprite and copied it.

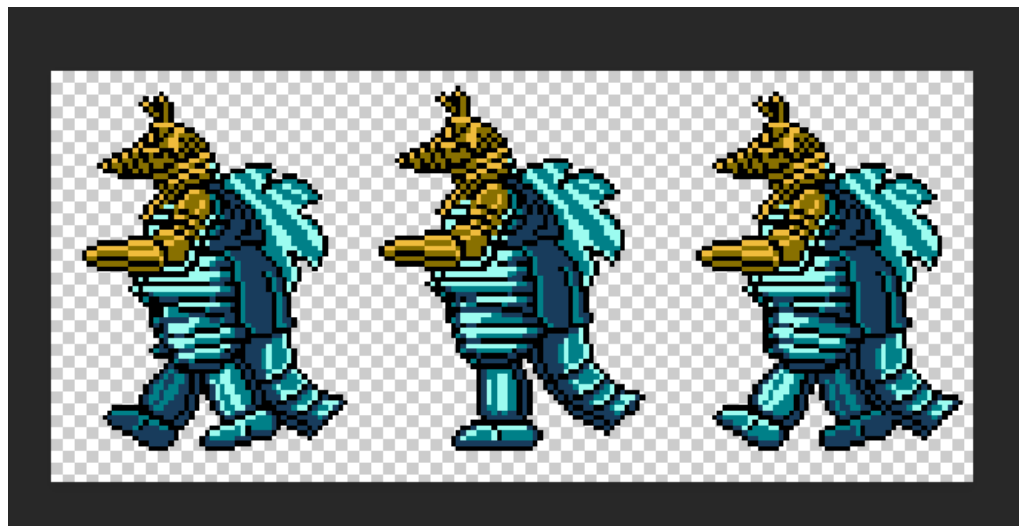




As we can see, the bounding box of 59 x 75 is fine, but I like pretty numbers so I will go with 60 x 80, but since there are 3 sprites we will make a new spritesheet with the size of 180 x 80 ( $180 = 60 * 3$ ).

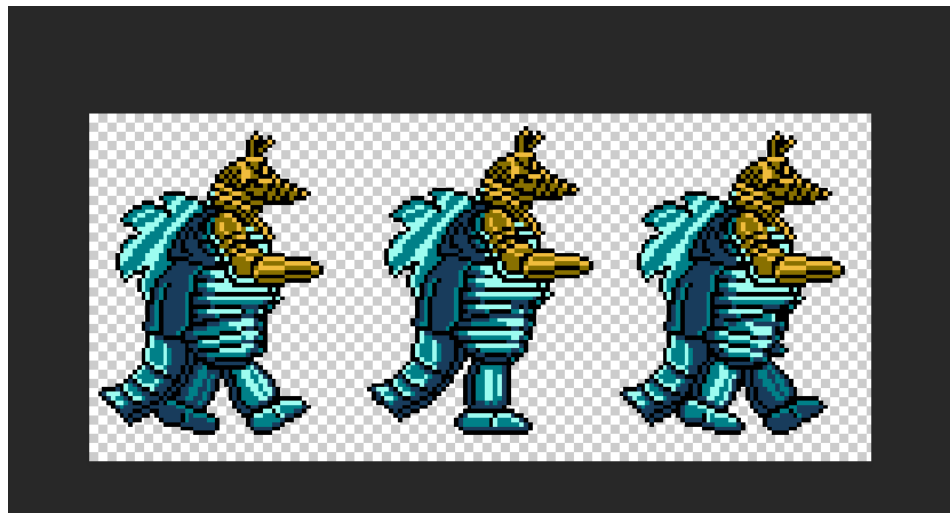


Now let's copy our sprites and delete the white background



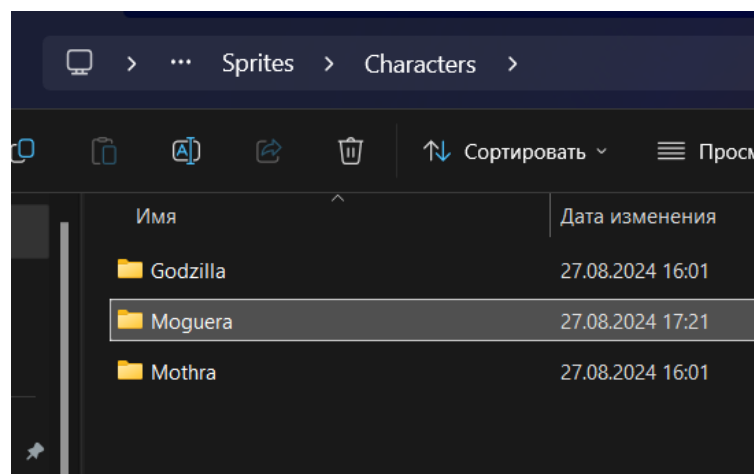
If we use the sprites in this state the animations will be a bit shaky (we haven't aligned each sprite correctly in their respective bounding boxes yet, you will see what I mean later) but we can fix that later.

Now let's mirror them since the player is facing right by default

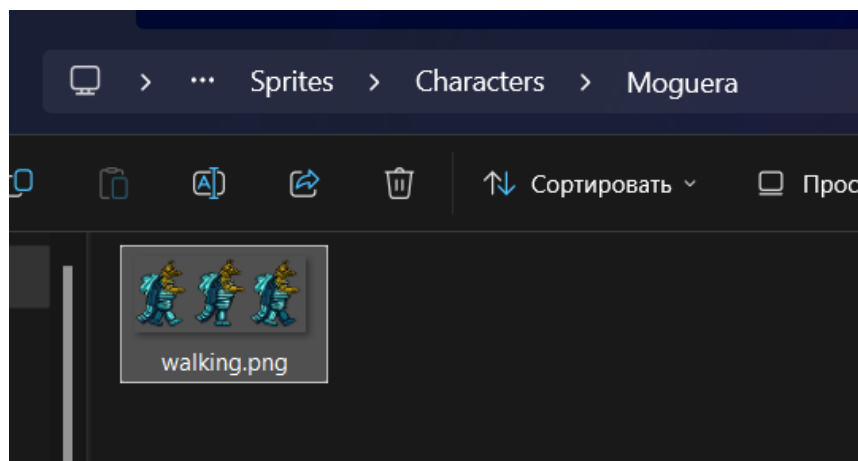


Great! Now let's save it. The folder for character sprites is "Sprites/Character", and you will notice that the sprites for each character are separated into folders. If you look into both folders you will know why: it would be messy and hard to manage if they were in the same folder.

So now let's create our new folder for our character:



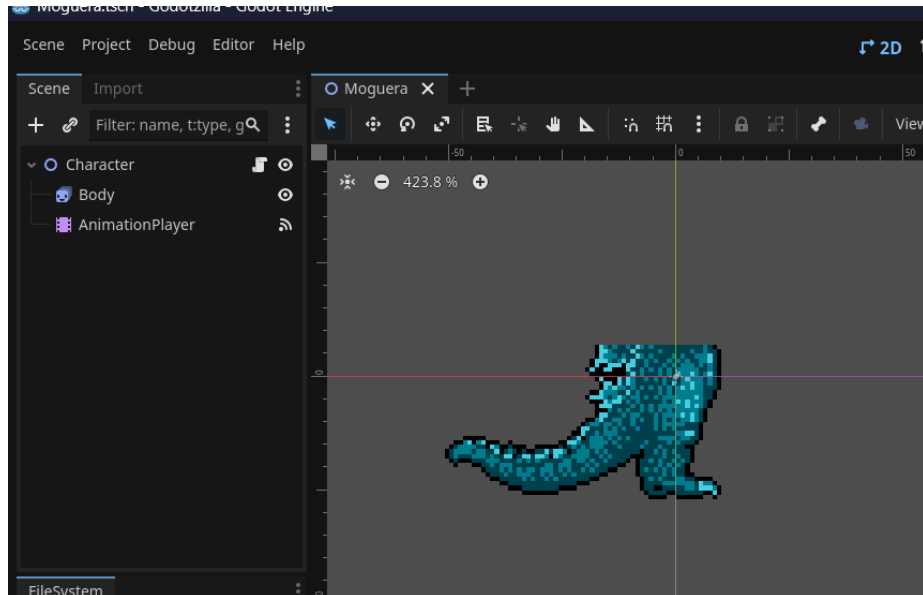
And now let's save our sprites there:



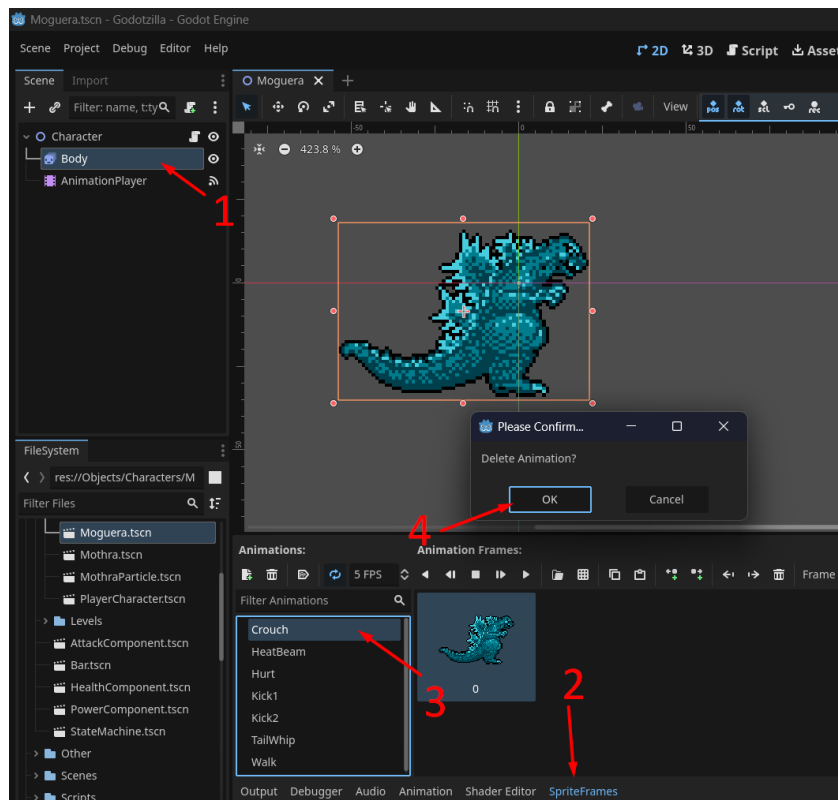
Great, now we're ready to import them into Godot!

## Importing sprites into Godot

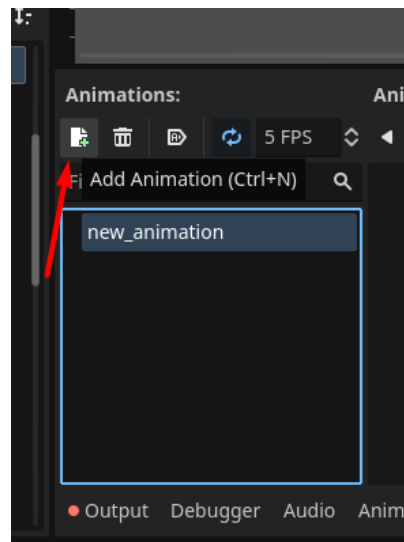
Since we don't have a separate top part of the sprites we can delete the "Head" node in our skin scene (.tscn file is a scene file in Godot). You will be prompted to delete related animation tracks, press OK.



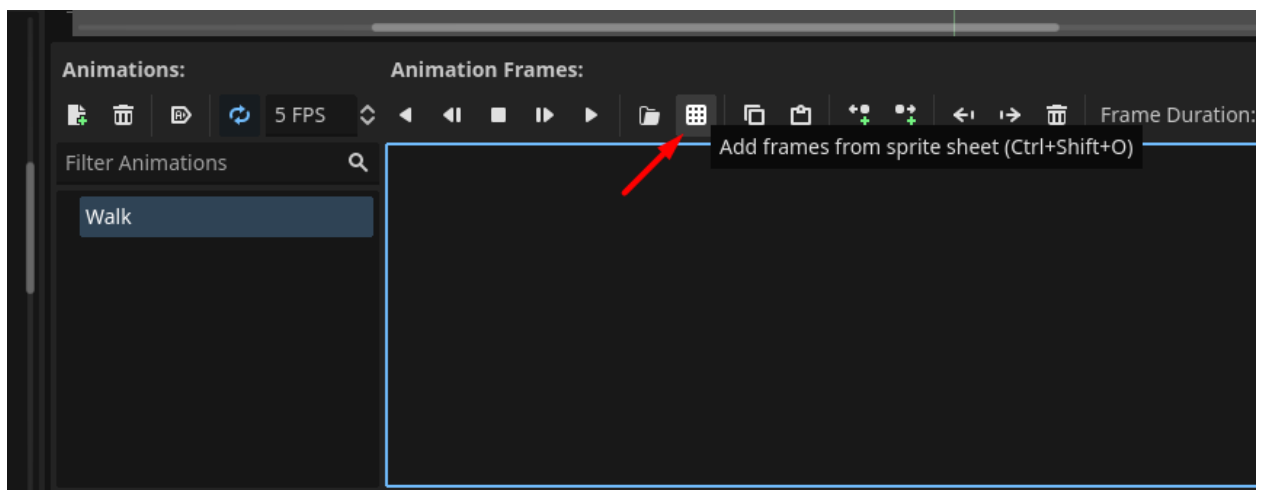
Now you can select the sprite node (1 on the picture) and delete each sprite animation it has. Select the SpriteFrames section (2), select an animation (3), press Delete on your keyboard, you will be prompted for confirmation, press OK (4). Do that for every animation.



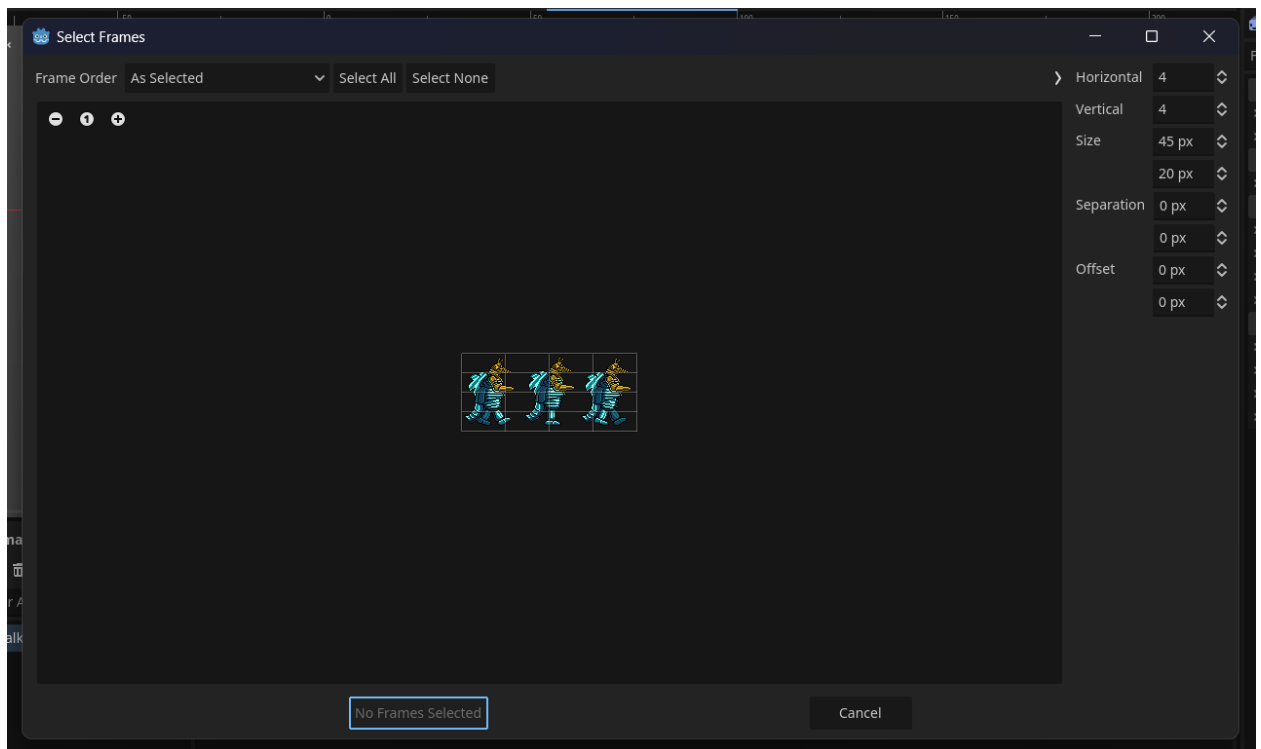
Now there are no animations. Let's add one:



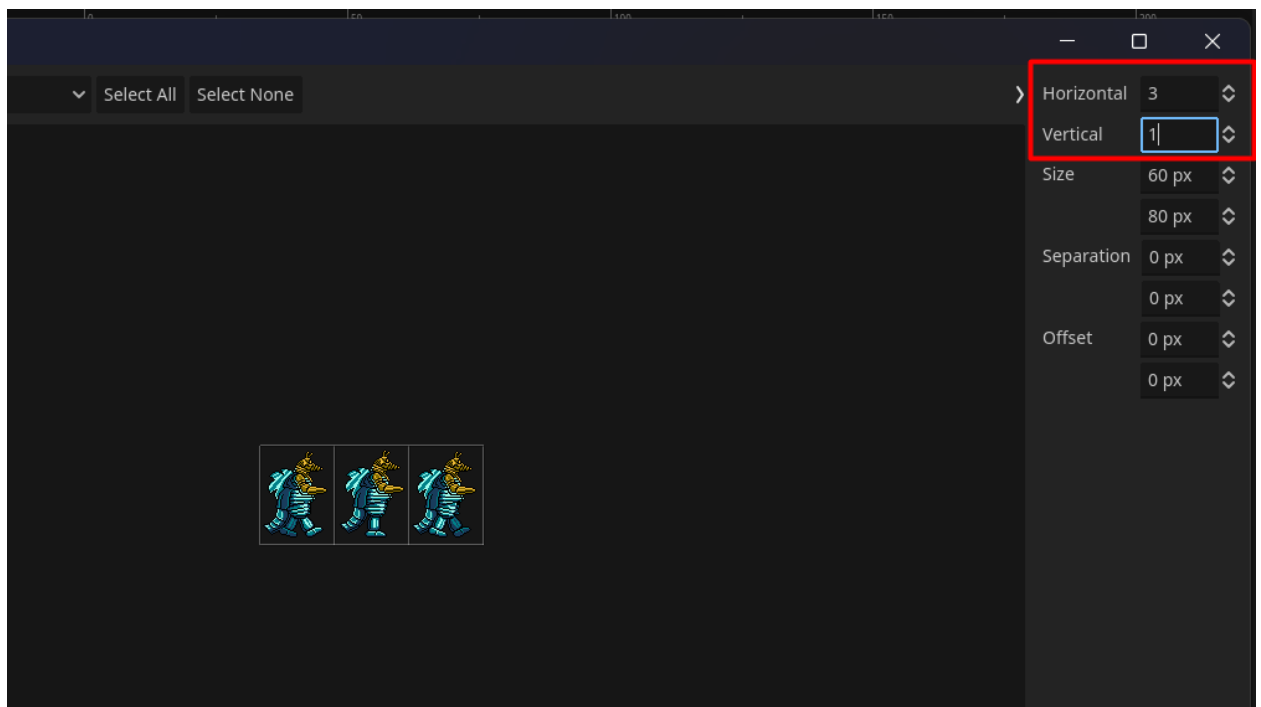
While selected, you can press it again with the left mouse button to rename it. I will use “Walk” for the name. Now let’s add our sprites:



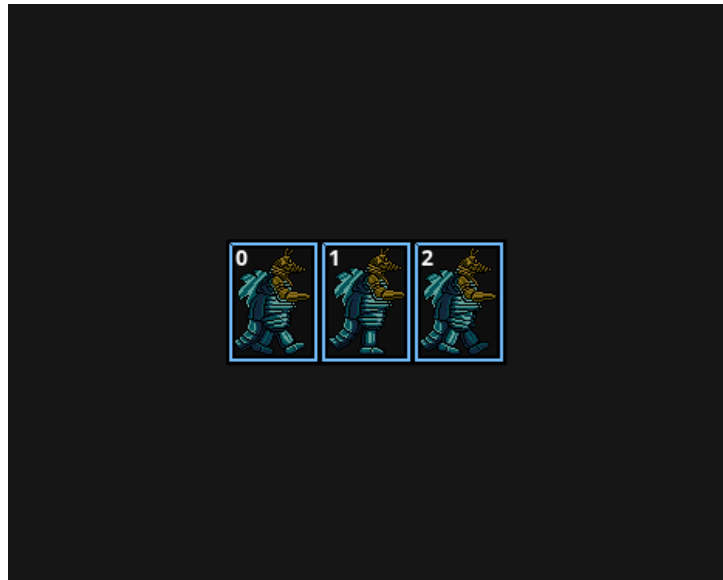
A file dialog will appear, select your sprites file. A new menu will open:



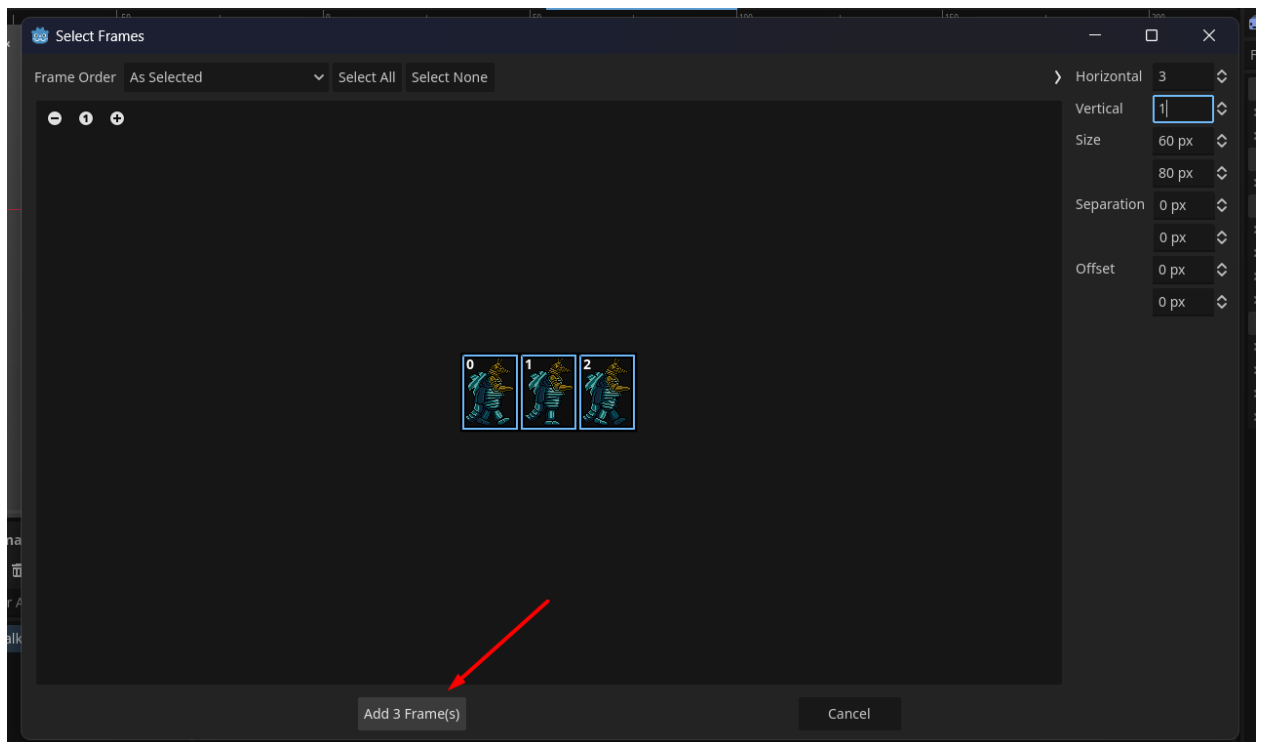
This is our sprite importer. Since there are 3 sprites arranged horizontally in just 1 vertical row, put 3 and 1 in “Horizontal” and “Vertical” options respectively:



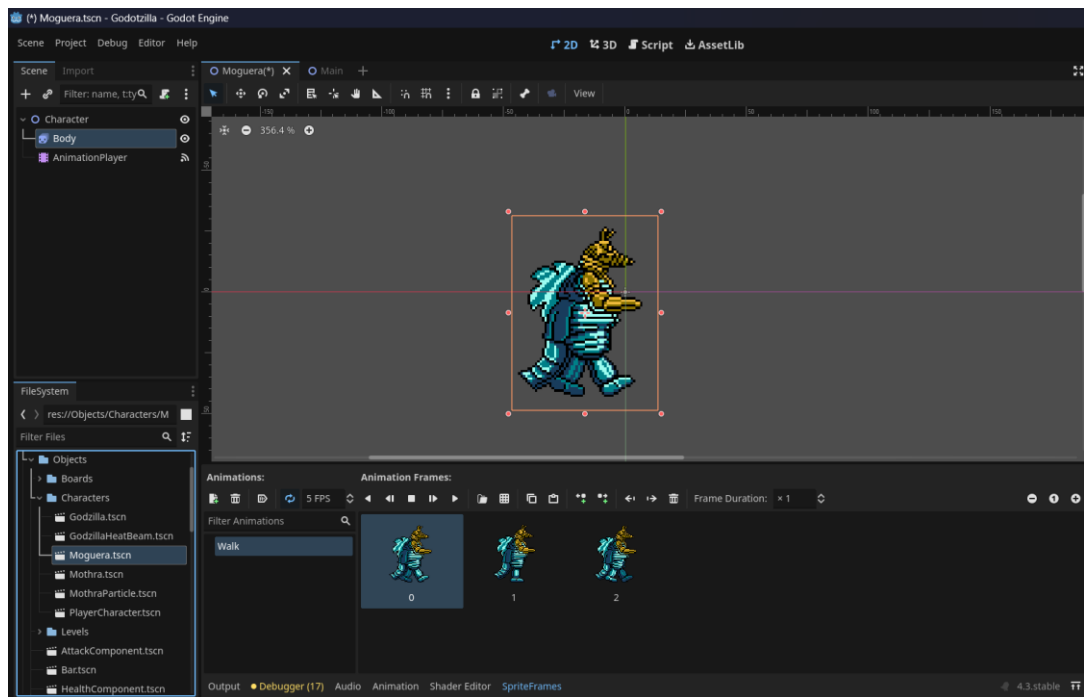
Now you can select every sprite



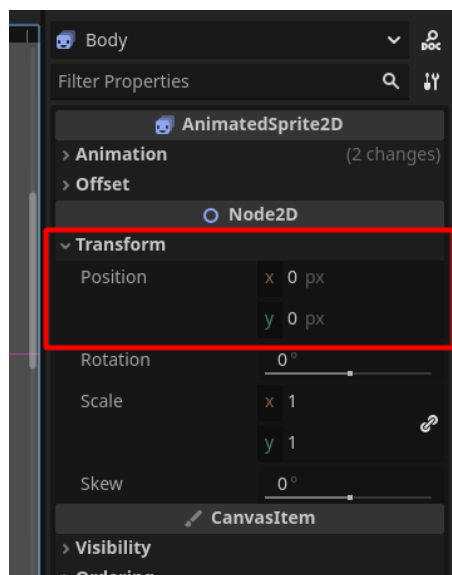
Press “Add 3 Frame(s)” to add the sprite frames.



Wow, we can now see our new character in Godot editor!

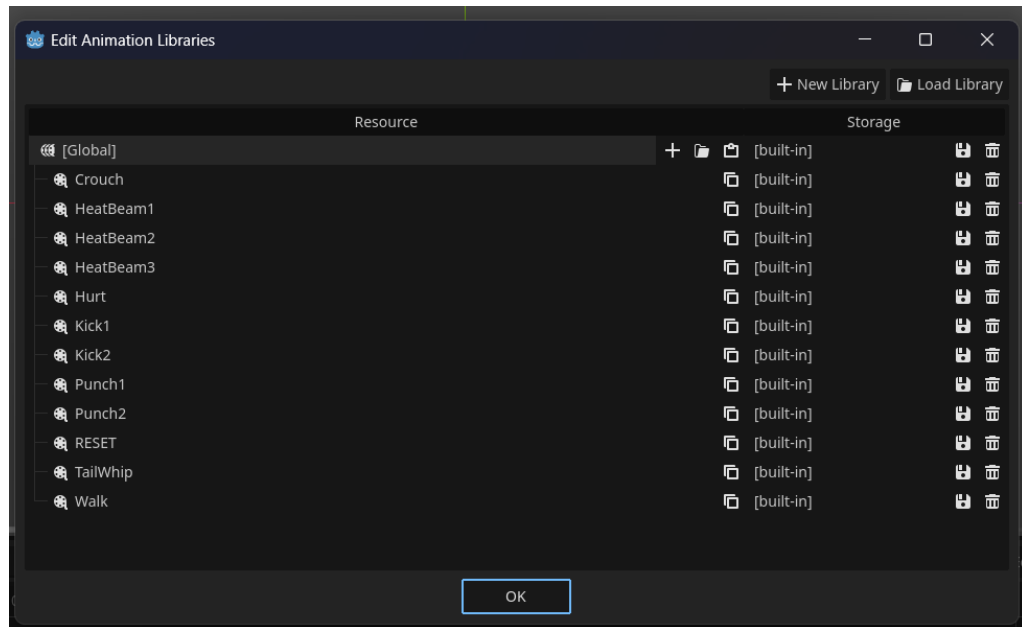
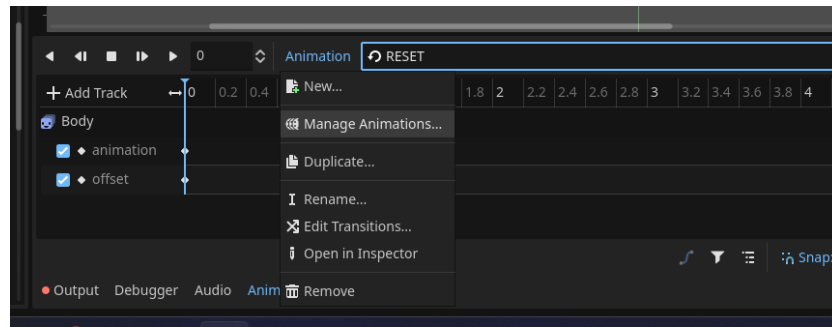


But you may notice it's a bit shaky if you look around or zoom in/out, so now let's fix that. Move the sprite into the middle of the scene, so that the "Body" node's position becomes 0,0

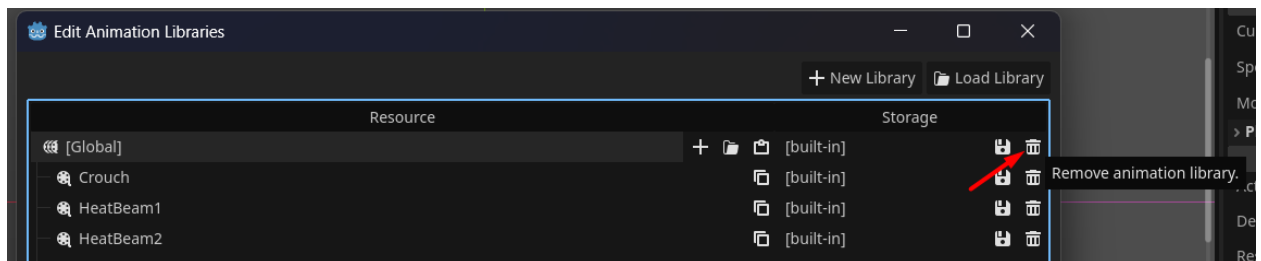


The shake happens because the position before was a decimal number, and the middle of our sprite is an integer number since the width of a sprite is 60, so the resulting position becomes a decimal number, and since pixel snapping is enabled in the project settings, Godot tries to position the object to the nearest integer position, and sometimes it doesn't work as expected.

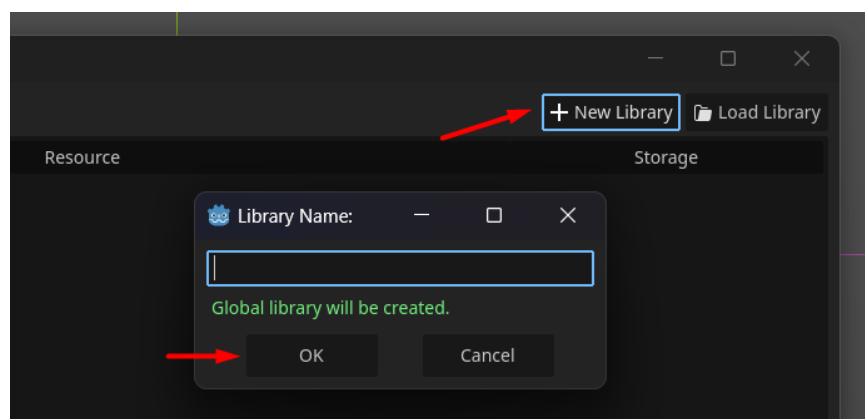
Now let's create a walking animation in the animation node. Select the node and use the "Manage Animations" button:



Now delete every animation by pressing this button:

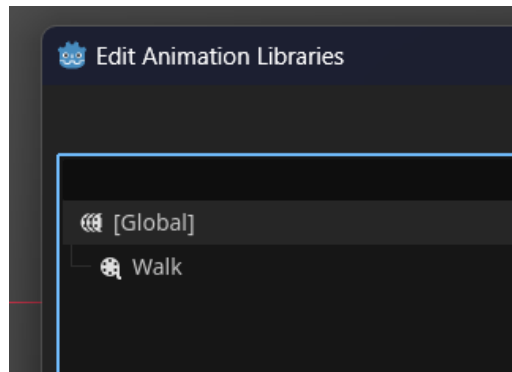
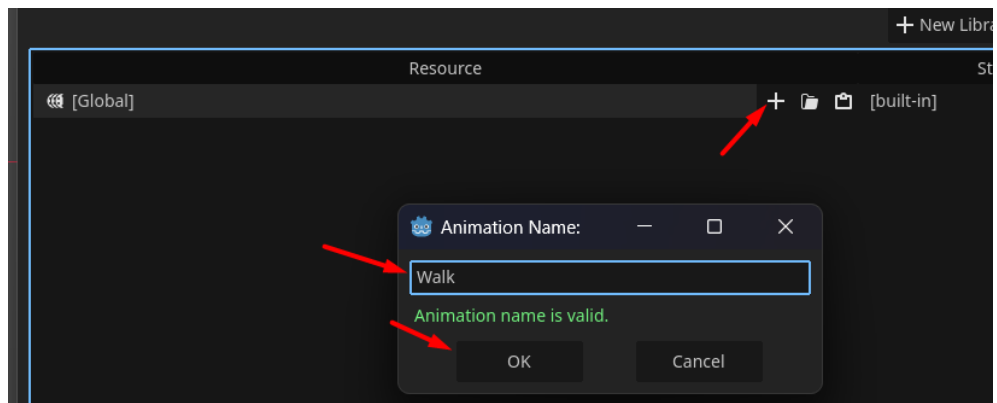


Now create a new global library:



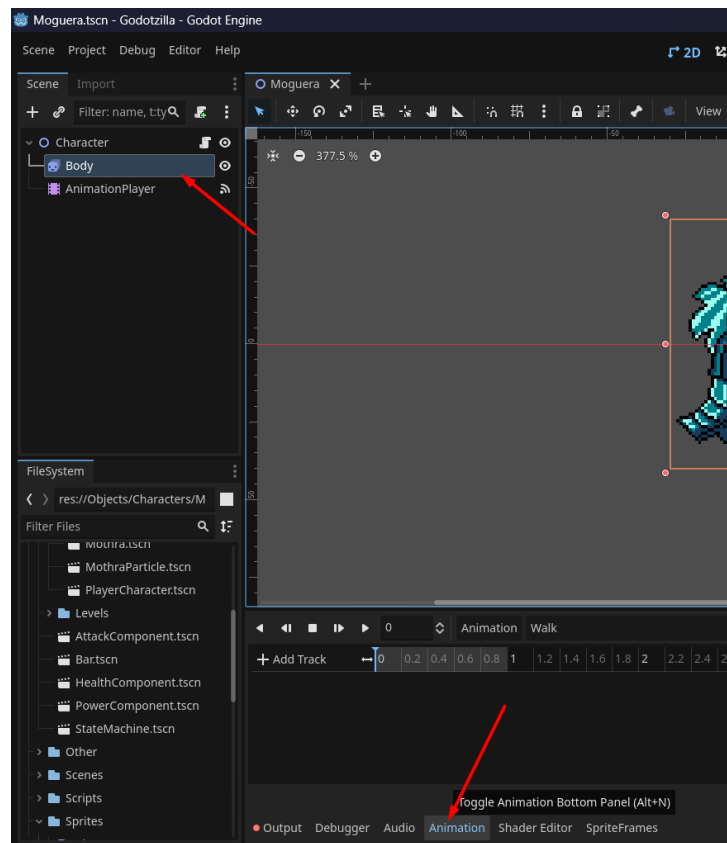
Now let's add a new animation:



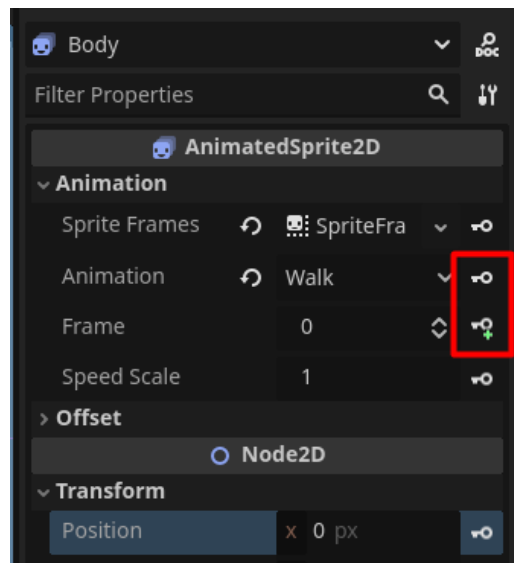


Great! Now press OK.

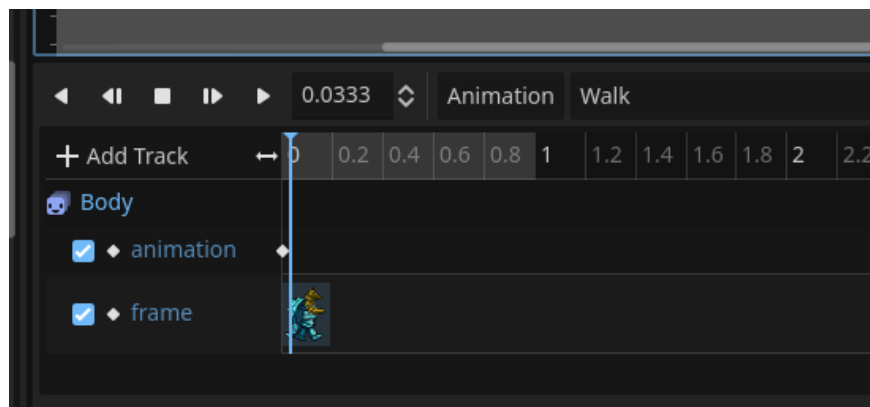
Let's setup the walking animation. It should already be selected in your animation player node. Now select the body node and go to the "Animation" tab:



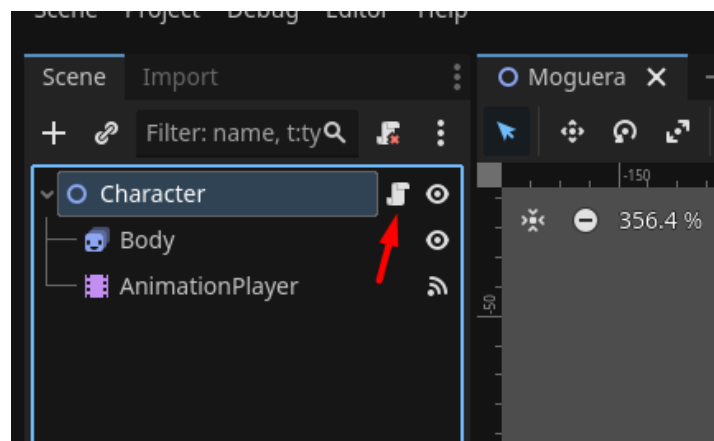
You will notice that the node properties in the right part of the screen now have key buttons. Select the keys for sprite's animation and frame properties to make an animation frame. You will be prompted to confirm the creation of a new animation track, a new animation key and a “RESET” track, just press “Create”.



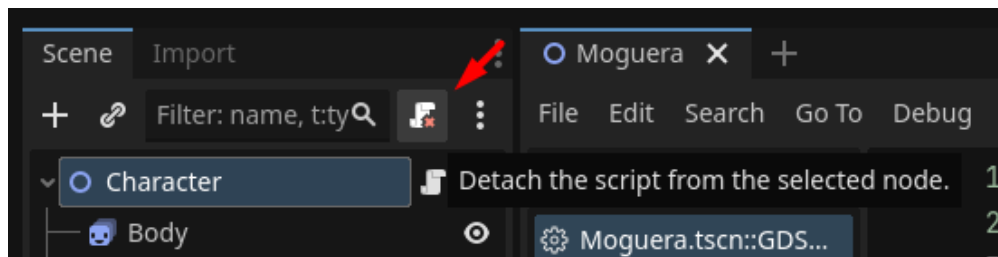
That should do it.



You may also notice that the “Character” node has a script attached to it:

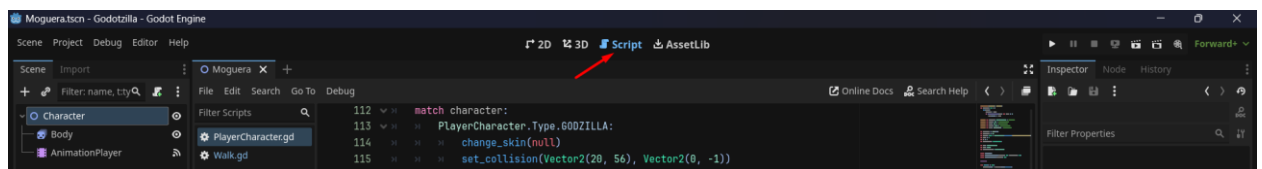


You can look at it but it's just Godzilla-specific small fixes regarding animation and collision box size, so we're going to remove it

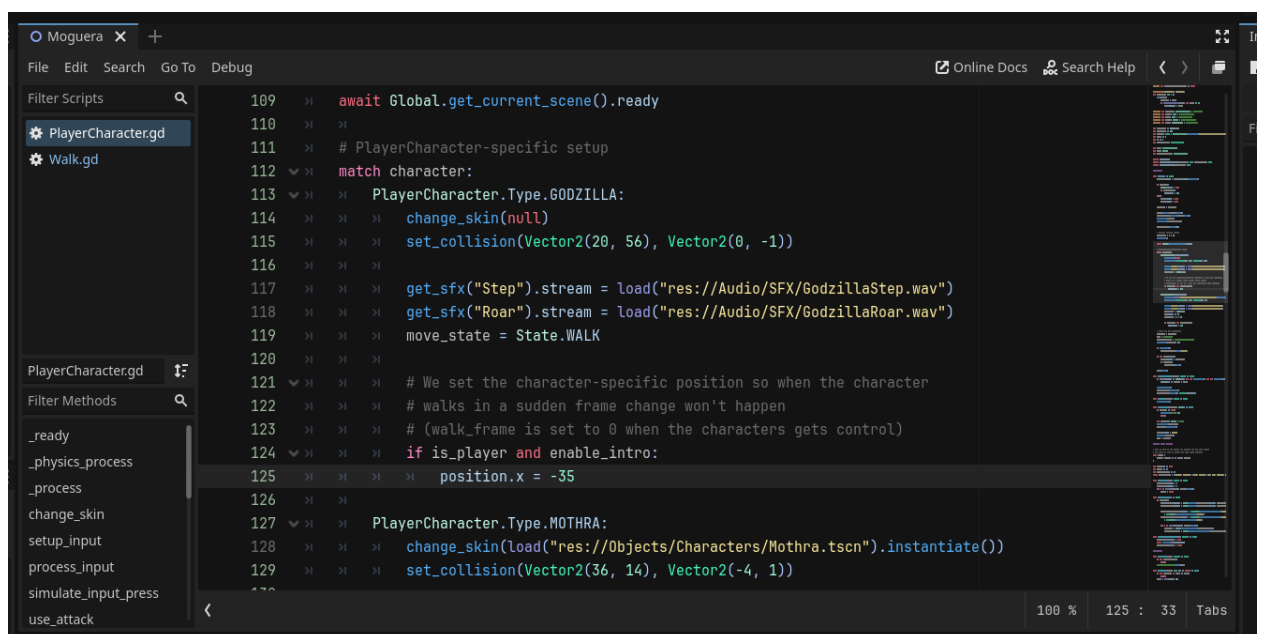


## Make the skin load

Our character skin should be ready now! Let's make the game load it. Open the "Script" tab:



If you haven't closed any script files in the tab you should still have the PlayerCharacter.gd file opened. Select that file and scroll down until you find the code that looks like this:



That's where we have our characters-specific setup. Copy the Godzilla's setup code for now:

```

110  |> |> |> |>
111  |> |> |> |> # PlayerCharacter-specific setup
112  |> |> |> |> match character:
113  |> |> |> |> PlayerCharacter.Type.GODZILLA:
114  |> |> |> |> |> change_skin(null)
115  |> |> |> |> |> set_collision(Vector2(20, 56), Vector2(0, -1))
116  |> |> |> |> |>
117  |> |> |> |> |> get_sfx("Step").stream = load("res://Audio/SFX/GodzillaStep.wav")
118  |> |> |> |> |> get_sfx("Roar").stream = load("res://Audio/SFX/GodzillaRoar.wav")
119  |> |> |> |> |> move_state = State.WALK
120  |> |> |> |> |>
121  |> |> |> |> |> # We set the character-specific position so when the character
122  |> |> |> |> |> # walks in a sudden frame change won't happen
123  |> |> |> |> |> # (walk_frame is set to 0 when the characters gets control)
124  |> |> |> |> |> if is_player and enable_intro:
125  |> |> |> |> |> |> position.x = -35
126  |> |> |> |> |>
127  |> |> |> |> |> PlayerCharacter.Type.MOTHRA:

```

And paste it below Mothra's setup code:

```

127  |> |> |> |> |> PlayerCharacter.Type.MOTHRA:
128  |> |> |> |> |> |> change_skin(load("res://Objects/Characters/Mothra.tscn").instantiate())
129  |> |> |> |> |> |> set_collision(Vector2(36, 14), Vector2(-4, 1))
130  |> |> |> |> |> |>
131  |> |> |> |> |> |> get_sfx("Step").stream = load("res://Audio/SFX/MothraStep.wav")
132  |> |> |> |> |> |> get_sfx("Roar").stream = load("res://Audio/SFX/MothraRoar.wav")
133  |> |> |> |> |> |> move_state = State.FLY
134  |> |> |> |> |> |> position.y -= 40
135  |> |> |> |> |> |> move_speed = 2 * 60
136  |> |> |> |> |> |>
137  |> |> |> |> |> |> if is_player and enable_intro:
138  |> |> |> |> |> |> |> position.x = -37
139  |> |> |> |> |> |>
140  |> |> |> |> |> |> PlayerCharacter.Type.GODZILLA:
141  |> |> |> |> |> |> |> change_skin(null)
142  |> |> |> |> |> |> |> set_collision(Vector2(20, 56), Vector2(0, -1))
143  |> |> |> |> |> |> |>
144  |> |> |> |> |> |> |> get_sfx("Step").stream = load("res://Audio/SFX/GodzillaStep.wav")
145  |> |> |> |> |> |> |> get_sfx("Roar").stream = load("res://Audio/SFX/GodzillaRoar.wav")
146  |> |> |> |> |> |> |> move_state = State.WALK
147  |> |> |> |> |> |> |>

```

And change “GODZILLA” to “MOGUERA”:

```

|> |> |> |> |> position.x = -37
|> |> |> |> |>
|> |> |> |> |> PlayerCharacter.Type.MOGUERA:
|> |> |> |> |> |> change_skin(null)
|> |> |> |> |> |> set_collision(Vector2(20, 56), Vector2(0, -1))
|> |> |> |> |> |>

```

Nice, now let's tell the game to load our skin file. You can see how the game loads Mothra's skin in the code above our new Moguera setup:

```

|> |> |> |> |> if is_player and enable_intro:
|> |> |> |> |> |> position.x = -35
|> |> |> |> |> |>
|> |> |> |> |> |> PlayerCharacter.Type.MOTHRA:
|> |> |> |> |> |> |> change_skin(load("res://Objects/Characters/Mothra.tscn").instantiate())
|> |> |> |> |> |> |> set_collision(Vector2(36, 14), Vector2(-4, 1))
|> |> |> |> |> |> |>
|> |> |> |> |> |> |> get_sfx("Step").stream = load("res://Audio/SFX/MothraStep.wav")

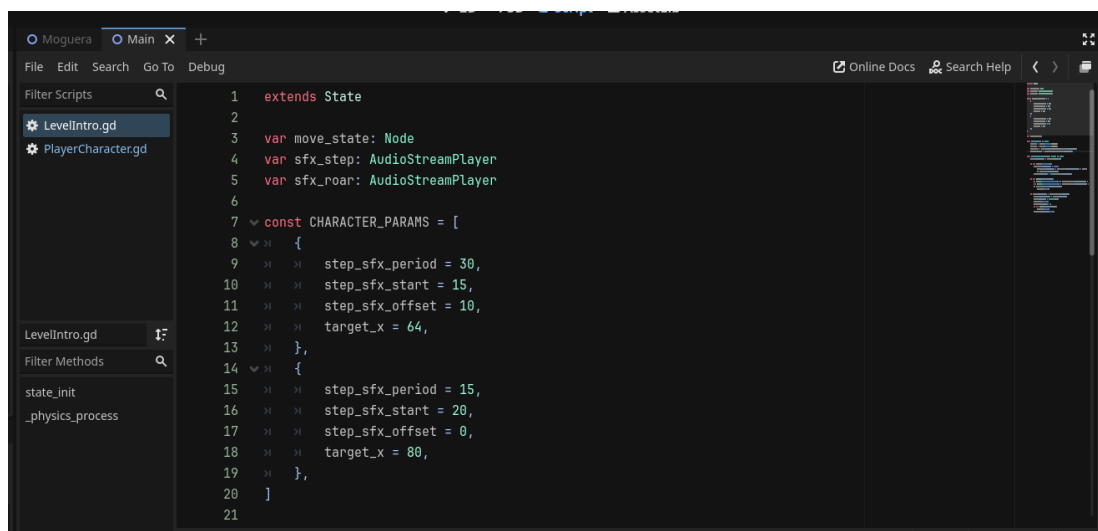
```

So let's copy this line and modify it to point to our new skin:

```
>|
>| PlayerCharacter.Type.MOGUERA:
>| >| change_skin(load("res://Objects/Characters/Moguera.tscn").instantiate())
>| >| set_collision(Vector2(20, 56), Vector2(0, -1))
>| >|
```

## Other character parameters

You will probably want to test the character now, but we still have some more stuff that we need to do before we can test our character. Open “Scripts/Objects/Characters/LevelIntro.gd”:




You now see characters parameters for their level intro (when you enter a level the character walks out from the left side of the screen, unable to be controlled and after some time they make a sound and you get the control over the character, that's called “level intro”). So now let's copy Godzilla's parameters for now:

```
6
7 const CHARACTER_PARAMS = [
8     {
9         step_sfx_period = 30,
10        step_sfx_start = 15,
11        step_sfx_offset = 10,
12        target_x = 64,
13    },
14    {
15        step_sfx_period = 15,
16        step_sfx_start = 20,
17        step_sfx_offset = 0,
18        target_x = 80,
19    },
20 ]
21
```

And paste it below Mothra's, like so:

```
6
7  ▼ const CHARACTER_PARAMS = [
8  ▼ | {
9  | |   step_sfx_period = 30,
10 | |   step_sfx_start = 15,
11 | |   step_sfx_offset = 10,
12 | |   target_x = 64,
13 | | },
14 ▼ | {
15 | |   step_sfx_period = 15,
16 | |   step_sfx_start = 20,
17 | |   step_sfx_offset = 0,
18 | |   target_x = 80,
19 | | },
20 ▼ | {
21 | |   step_sfx_period = 30,
22 | |   step_sfx_start = 15,
23 | |   step_sfx_offset = 10,
24 | |   target_x = 64,
25 | | },
```



You can play with these parameters by the way, you can change them however you want and it may work better depending on your character.

The game HUD shows the character's name, so let's add one for Moguera. Open the PlayerCharacter.gd again and on the line 37 there's this list:

```
36
37  ▼ const CHARACTER_NAMES: Array[String] = [
38  |   "Godzilla",
39  |   "Mothra",|
40  ]
41
```

Add a new entry in it:

```
36
37  ▼ const CHARACTER_NAMES: Array[String] = [
38  |   "Godzilla",
39  |   "Mothra",
40  |   "Moguera",
41  ]
42
```

Every character has a base power/life bar count (the amount of bars if the character is on level 1), for Godzilla it's 6 and for Mothra it's 8. Right below the "CHARACTER\_NAMES" list there's "BaseBarCount":

```

42
43 ▼ const BaseBarCount: Array[int] = [
44   » 6, # Godzilla
45   » 8, # Mothra
46   1

```

For now Godzilla's base amount should work, so add a new entry containing "6":

```

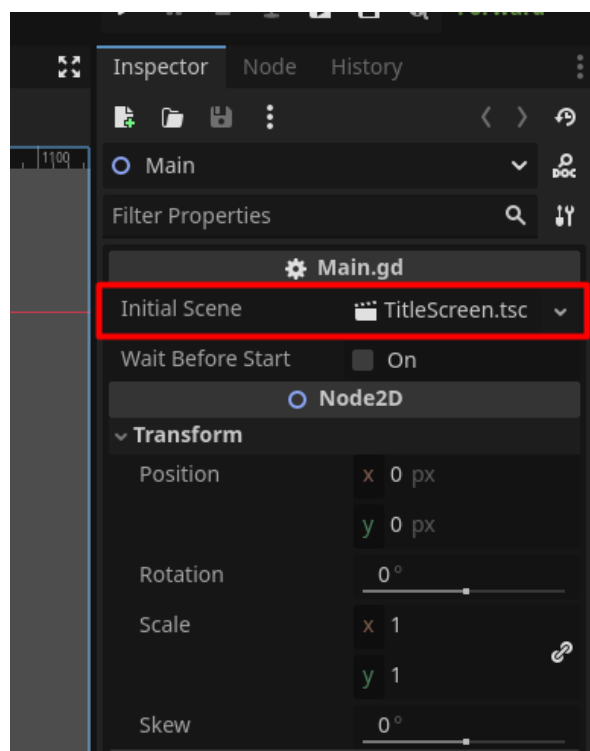
42
43 ▼ const BaseBarCount: Array[int] = [
44   » 6, # Godzilla
45   » 8, # Mothra
46   » 6, # Moguera|
47   ]
48

```

Of course, you can use a different number if you want.

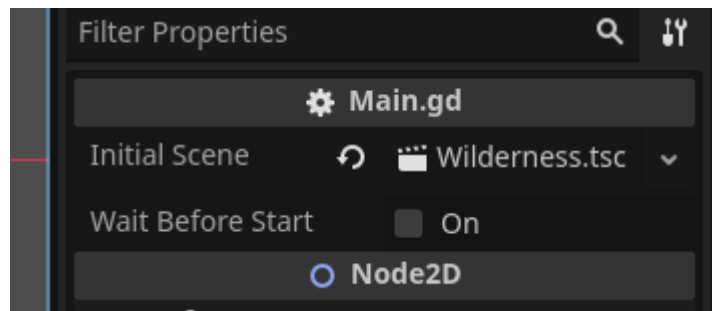
## Testing our new character

Now let's look at our character in-game! First, let's make the game to start on a level. In the Godot file system (bottom-left corner of the screen), find the file "Scenes/Main.tscn", now double-click it and look at the properties in the right part of the screen:



Now we know where we can change the initial scene the game starts at. Now where's the level scene? Let's choose the wilderness level. In the Godot file

system, find “Scenes/Levels/Wilderness.tscn”, now click and hold it with left mouse button, drag it onto the “Initial Scene” property value (where it says “TitleScreen.tscn” like on the picture above), now stop holding the left mouse button. You will notice that the initial scene property was changed:



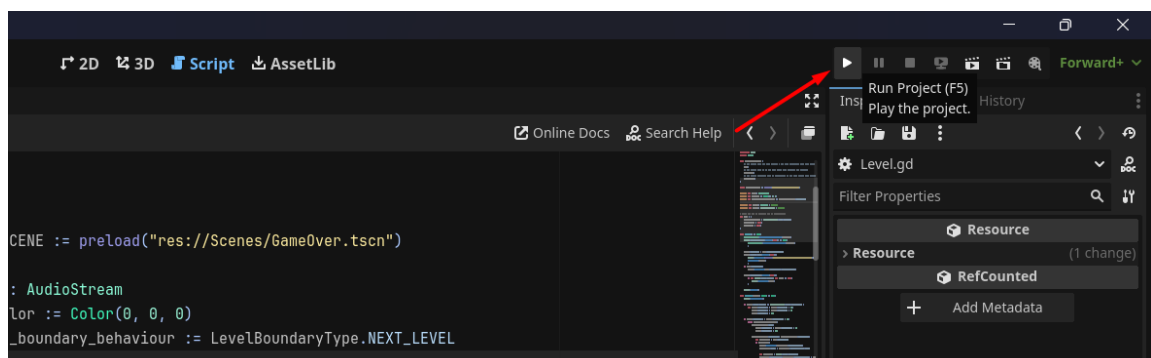
But if you run the game, Godzilla will be the default character, so let's change that. In the Godot file system, find “Scripts/Levels/Level.gd”, that's the level scene code, and on the line 24 you will see the default value for the character ID:

```
21
22 # These are set in Board.gd and in next_level()
23 var data = {
24     current_character = PlayerCharacter.Type.GODZILLA,
25     board_piece = null,
26     boss_piece = null,
27 }
28
```

That's our type enum that we added a new entry in! You can now change “GODZILLA” to “MOGUERA”:

```
21
22 # These are set in Board.gd and in next_level()
23 var data = {
24     current_character = PlayerCharacter.Type.MOQUERA,
25     board_piece = null,
26     boss_piece = null,
27 }
```

Now you can start the game by either pressing F5 or pressing this button in the top-right corner:



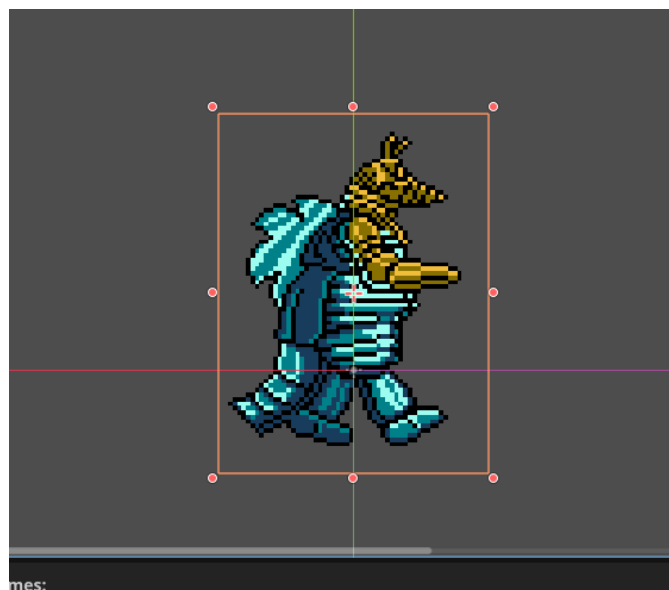




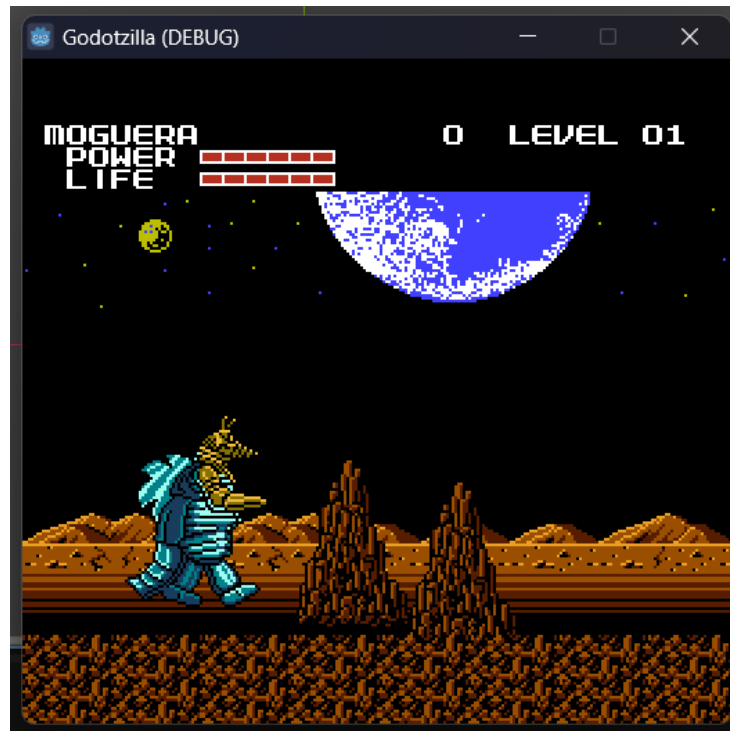
We can now see our character on the screen! But you will notice that Moguera's legs are partially in the ground, and the animation is stuck on the first frame. So now let's fix that.

## Fixing skin position and animation

Do you remember where's your skin file? If not, it's in "Objects/Characters" folder, double click on "Moguera.tscn". Now you can move the sprite upper, you can just drag it.



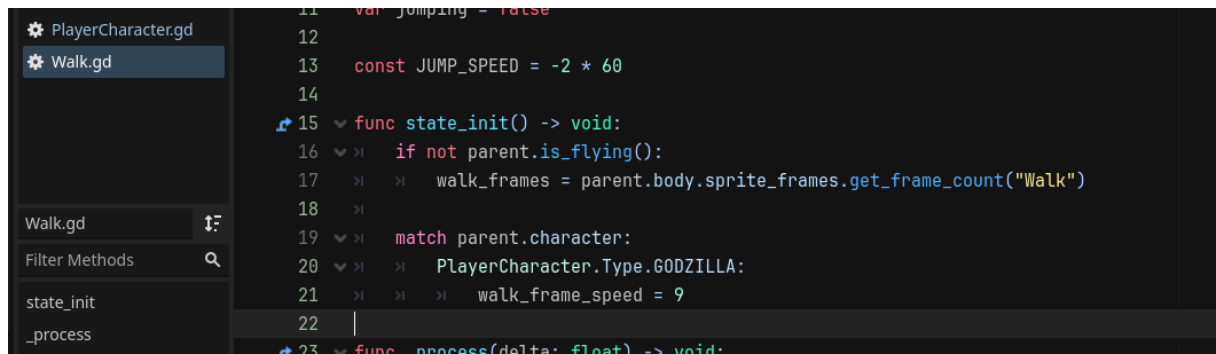
Now let's test again



Keep moving the sprite until it looks alright:



Perfect. Now let's fix the walking animation problem. In the file system, find "Scripts/Objects/Characters/Walk.gd" and double click it.



So that's where we setup our walking animation speed. Do the same thing we've done with Moguera's skin setup: copy Godzilla's specific code and modify it to fit Moguera:

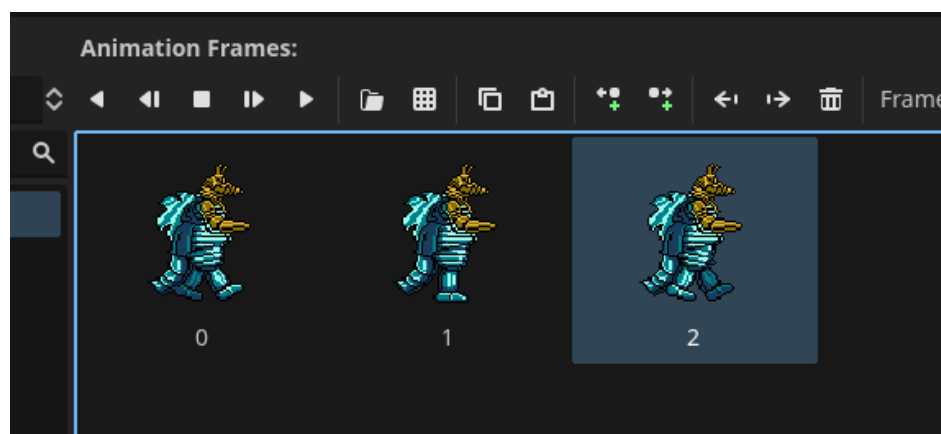
```

18  >|
19  >| match parent.character:
20  >| >| PlayerCharacter.Type.GODZILLA:
21  >| >| >| walk_frame_speed = 9
22  >| >| PlayerCharacter.Type.MOGUERA:
23  >| >| >| walk_frame_speed = 9
24

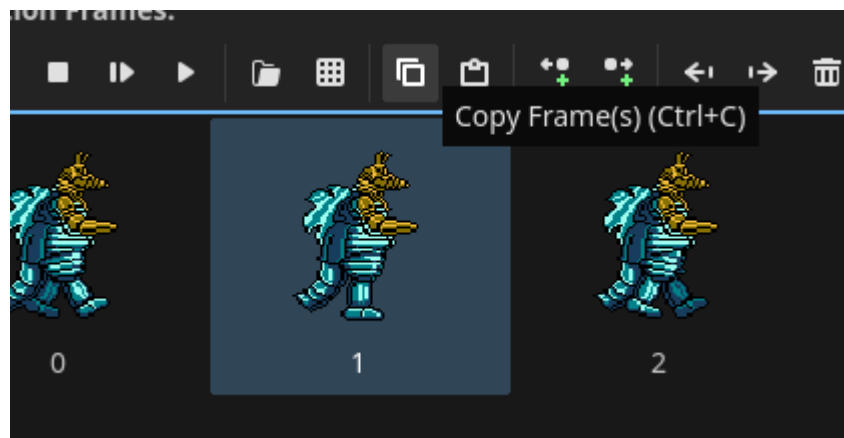
```

Now let's test the game! Moguera looks a bit shaky and the animation looks weird... Almost the same thing as the skin position, you can change the position of each sprite on the spritesheet by separating them and moving them individually. In my case, the standing sprite looks like it has an offset, so I will separate it and move it until it looks alright.

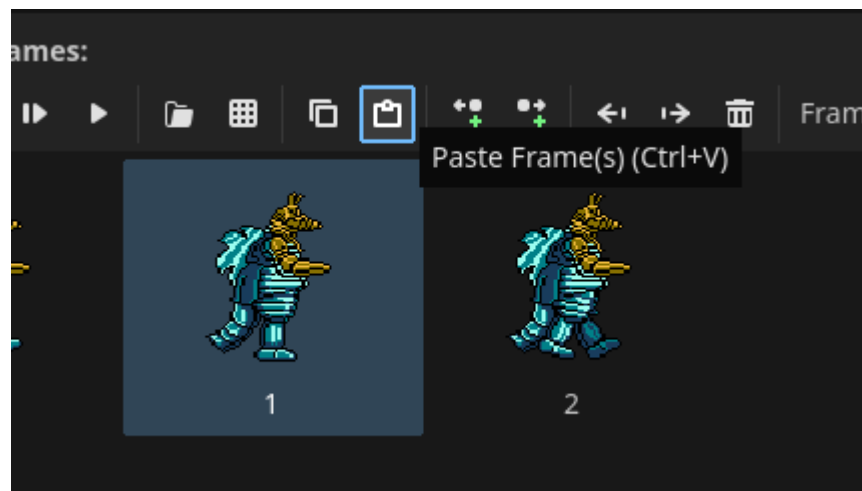
Another problem fixed, but the animation still looks weird. Let's take a look at our sprites in the "Body" node:



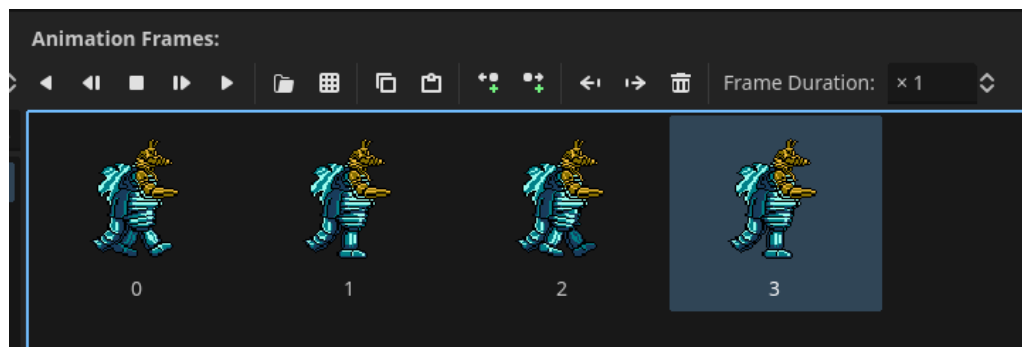
The walking animation loops when it reaches the final frame, so when the sprites reach the "right leg is behind", the game just loops right into the "right leg is in front" sprite, skipping the "right leg is in the middle" sprite, so let's make a copy of that sprite and paste it into the end of the animation. Click on the sprite 1 and press "Copy Frame(s)":



Now when it's copied, press "Paste Frame(s)":



Oh, look, it's pasted at the end of our animation, nice!



Let's try the game again. Wow, the animation frames look alright, but the animation itself is a bit too fast for that number of sprites. Remember the "walk\_frame\_speed" code we setup above? You can change that "9" in there until the animation looks fine.

```

18 >|
19 >| match parent.character:
20 >| >| PlayerCharacter.Type.GODZILLA:
21 >| >| >| walk_frame_speed = 9
22 >| >| PlayerCharacter.Type.MOQUERA:
23 >| >| >| walk_frame_speed = 5|
24

```

“5” should work. Test the game again. For me the animation started to look nice.

## Board piece

Don’t forget that there also should be a specific board piece that can be used to control where the character goes on the board! First, let’s find out where are the board pieces’ sprites are located: it’s “Sprites/BoardSprites.png”, and you will notice that there are sprites for both Godzilla and Mothra pieces as well as other board sprites, such as the selector and message window.



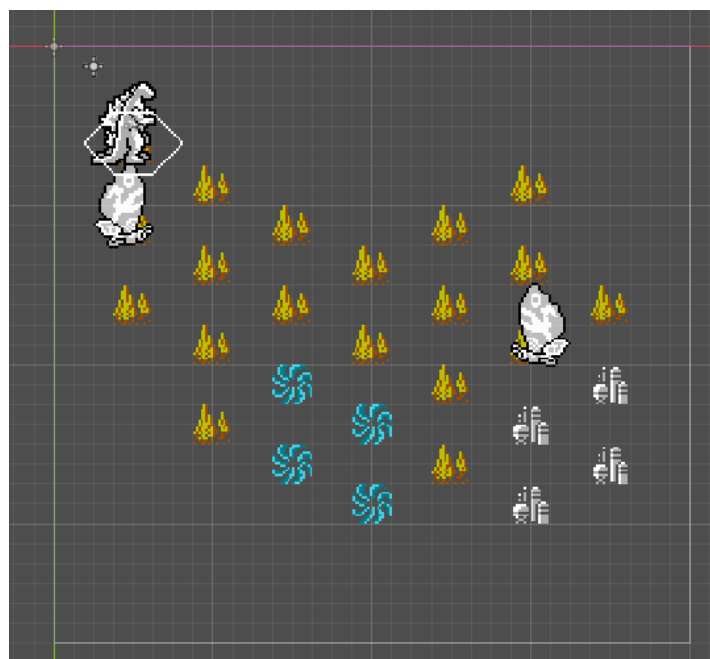
Let’s make some space for our new character. Every piece sprite here is 48x48 in size, so let’s add a new 48-pixel-high row.



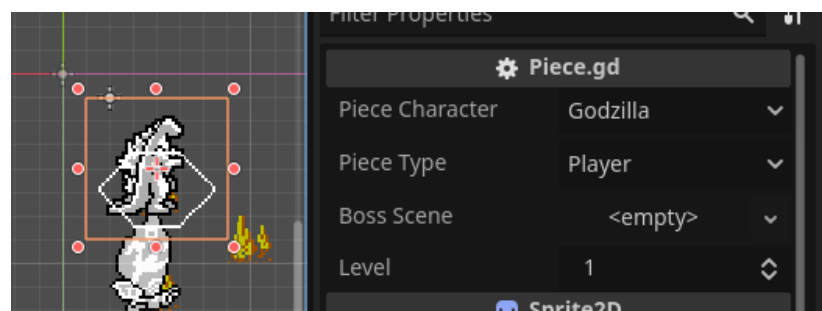
Nice, now let’s add our sprites.



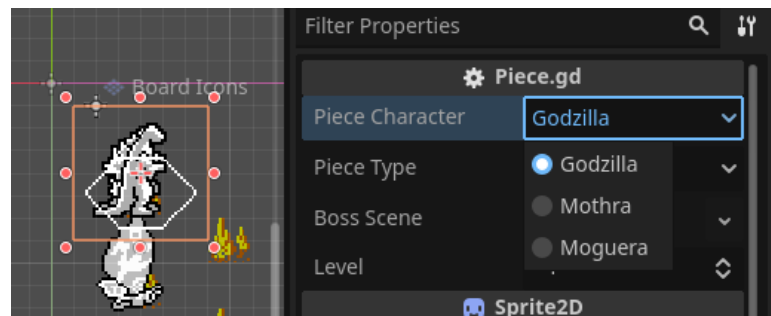
Great! Now let's actually make a new board piece. First, open a board scene, for example, "Scenes/Boards/TheEarth.tscn".



Now click on any piece and look at its properties.



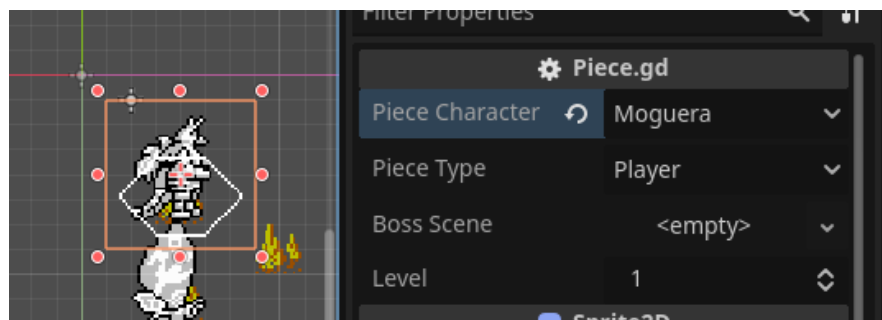
We can set its character with “Piece Character” property, click on the dropdown box next to it.



Wait, what? Moguera is already there? I mean, we didn't even have to add any texts ourselves to make it work in the editor.

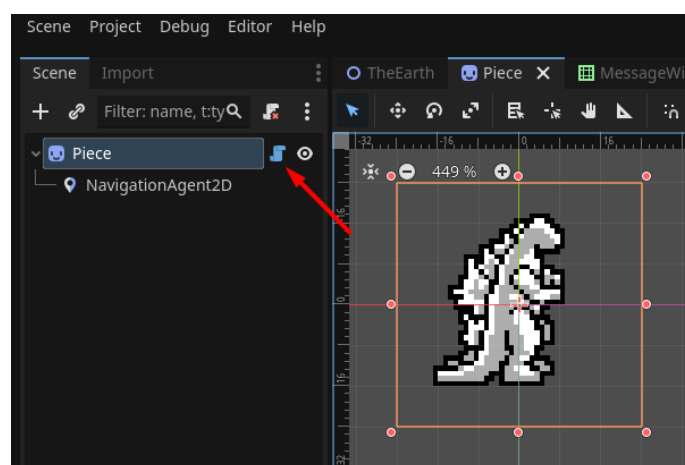
Yes! Godot is smart. See, the piece character property actually has the type of our character type enum that we added a new entry in in the first chapter. Godot automatically gets every enum entry and changes its name to something more readable. Amazing!

Now click on our new character's name in that dropdown box.



The piece immediately changed its sprite to show our new character, neat!

There are still some piece properties we haven't changed yet, so open the piece scene again.



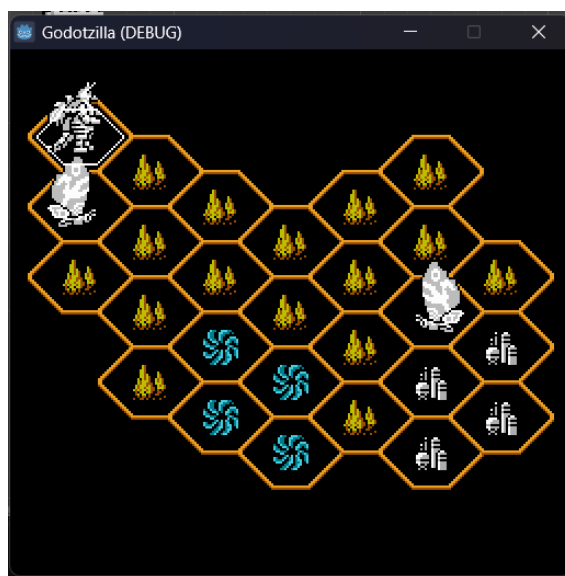
Click on this button to open up the piece script, and scroll up to the very top of the script.

```
File Edit Search Go To Debug
Filter Scripts
Piece.gd
1 @tool
2 extends Sprite2D
3
4 const PIECE_STEPS := [
5     2, # Godzilla
6     4, # Mothra
7 ]
8 const FRAME_COUNT := 3 # White piece and 2 colored walking sprites
9 const FRAME_SPEED := [
10    0.13, # Godzilla
11    0.2, # Mothra
12 ]
13
```

Here are some piece parameters: piece steps and frame speed. Piece steps is how much levels on the board can the character move through on the player's turn, and frame speed is actually how much time (in seconds) is required for a piece to proceed to its next frame while moving. I'm going to use the same values as Godzilla for this tutorial.

```
3
4 const PIECE_STEPS := [
5     2, # Godzilla
6     4, # Mothra
7     2, # Moguera
8 ]
9 const FRAME_COUNT := 3 # Whi
10 const FRAME_SPEED := [
11    0.13, # Godzilla
12    0.2, # Mothra
13    0.13, # Moguera
14 ]
```

Now, remember the chapter about testing our character, where we set the initial scene to be a level? You can do the same thing but change the level to the board we're currently on in the editor, "Scenes/Boards/TheEarth.tscn".



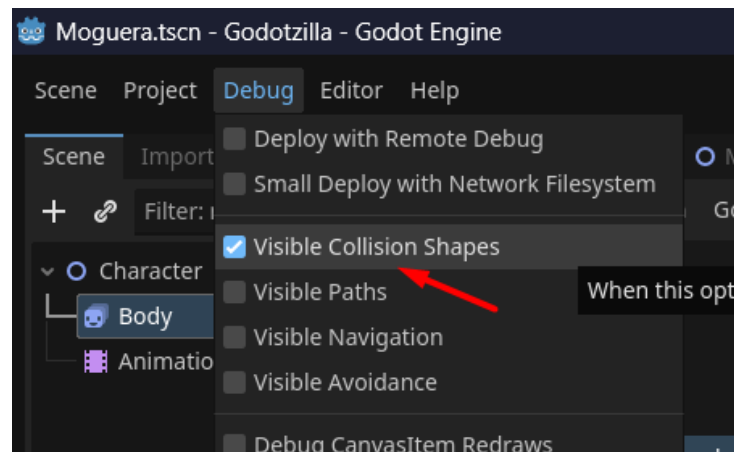
Wow, it works! We now have our character on the board!



If you select the character and it looked like the sprite moved, you can change the sprite positions on the board sprites spritesheet.

## Changing the character's collision box (optional)

Now it may be subtle, but Moguera is a bit taller than Godzilla, so we may want to change the collision box. But first let's take a look at our collision box by enabling visible collision shapes:



The collision box is indeed a bit too small for Moguera.

Remember our character setup code, the one that also setups the character's skin in `PlayerCharacter.gd`?

```

141 >| >|
142 ▾ >| >| PlayerCharacter.Type.MOGUERA:
143 >| >| >| change_skin(load("res://Objects/Characters/Moguera.tscn").instantiate())
144 >| >| >| set_collision(Vector2(20, 56), Vector2(0, -1))
145 >| >| >|
146 >| >| >| get_sfx("Step").stream = load("res://Audio/SFX/GodzillaStep.wav")
147 >| >| >| get_sfx("Roar").stream = load("res://Audio/SFX/GodzillaRoar.wav")
148 >| >| >| move_state = State.WALK
149 >| >| >|
150 ▾ >| >| >| # We set the character-specific position so when the character
151 >| >| >| # walks in a sudden frame change won't happen
152 >| >| >| # (walk_frame is set to 0 when the characters gets control)
153 ▾ >| >| >| if is_player and enable_intro:
154 >| >| >| >| position.x = -35
155 >| >| >|

```

We can see a line that calls the “set\_collision” function, it takes the collision box size as the first parameter and offset as the second parameter. Let’s change the size to something like Vector2(20, 64):

```

PlayerCharacter.Type.MOGUERA:
>| change_skin(load("res://Objects/Characters/Moguera.tscn").instantiate())
>| set_collision(Vector2(20, 64), Vector2(0, -1))
>|

```



Now the collision box size looks alright, but Moguera’s skin is now floating in the air. We can either offset the collision box or move the sprite in the skin. You can choose either one, but I recommend moving the sprite. So now open the skin scene and move the sprite, like how I showed it before.



Perfect!

## Using different SFX for level intro (optional)

Now, if you want our new character to use different sound effect for walking and “roaring” (the sound that plays when the player gets in control of the character after level intro) while level intro is playing, you can add them into the “Audio/SFX” folder and reference them in the last file we had open:

```
>| >|
>| >| PlayerCharacter.Type.MOGUERA:
>| >| >| change_skin(load("res://Objects/Characters/Moguera.tscn").instantiate())
>| >| >| set_collision(Vector2(20, 64), Vector2(0, -1))
>| >| >|
>| >| >| get_sfx("Step").stream = load("res://Audio/SFX/GodzillaStep.wav")
>| >| >| get_sfx("Roar").stream = load("res://Audio/SFX/GodzillaRoar.wav")
>| >| >| move_state = State.WALK
>| >| >|
>| >| >| # We set the character-specific position so when the character
>| >| >| # walks in a sudden frame change won't happen
>| >| >| # (walk_frame is set to 0 when the characters gets control)
>| >| >| if is_player and enable_intro:
>| >| >| >| position.x = -35
>| >| >|
```

## Making the character fly (optional)

If you want to create a flying character, like Mothra or Rodan, we should do the following:

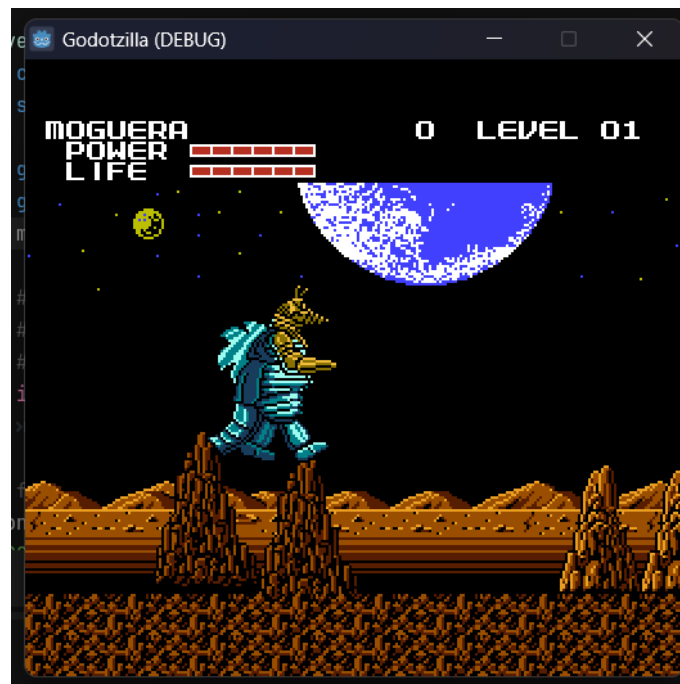
First let's take a look at Mothra's setup code in PlayerCharacter.gd:

```
PlayerCharacter.Type.MOTHRA:
>|  change_skin(load("res://Objects/Characters/Mothra.tscn").instantiate())
>|  set_collision(Vector2(36, 14), Vector2(-4, 1))
>|
>|  get_sfx("Step").stream = load("res://Audio/SFX/MothraStep.wav")
>|  get_sfx("Roar").stream = load("res://Audio/SFX/MothraRoar.wav")
>|  move_state = State.FLY
>|  position.y -= 40
>|  move_speed = 2 * 60
>|
>|  if is_player and enable_intro:
>|    position.x = -37
```

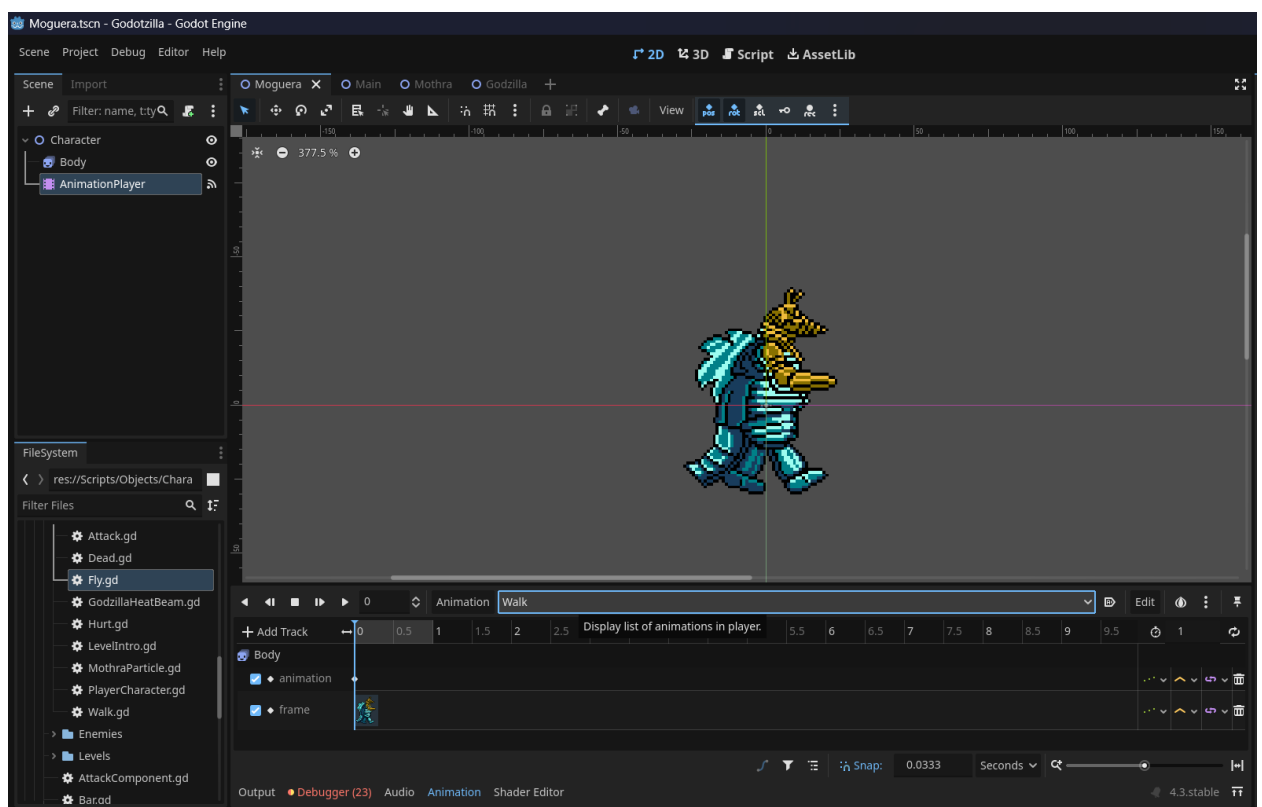
Do you notice something? We set the move\_state variable to the flying state. The move\_state variable, as you may guess, refers to the state that the character uses for moving. So we can copy this line for Moguera to make it fly:

```
>|
>|  PlayerCharacter.Type.MOQUERA:
>|  >|  change_skin(load("res://Objects/Characters/Moguera.tscn").instantiate())
>|  >|  set_collision(Vector2(20, 64), Vector2(0, -1))
>|  >|
>|  >|  get_sfx("Step").stream = load("res://Audio/SFX/GodzillaStep.wav")
>|  >|  get_sfx("Roar").stream = load("res://Audio/SFX/GodzillaRoar.wav")
>|  >|  move_state = State.FLY
>|  >|
>|  >|  # We set the character-specific position so when the character
>|  >|  # walks in a sudden frame change won't happen
>|  >|  # (walk_frame is set to 0 when the characters gets control)
>|  >|  if is_player and enable_intro:
>|  >|    position.x = -35
>|  >|
```

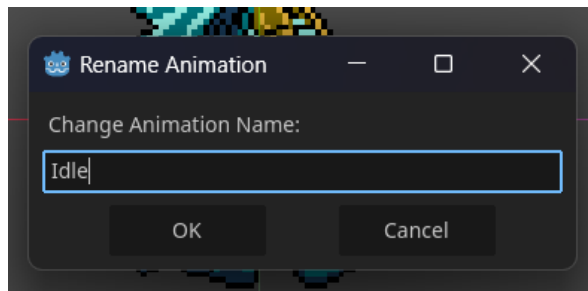
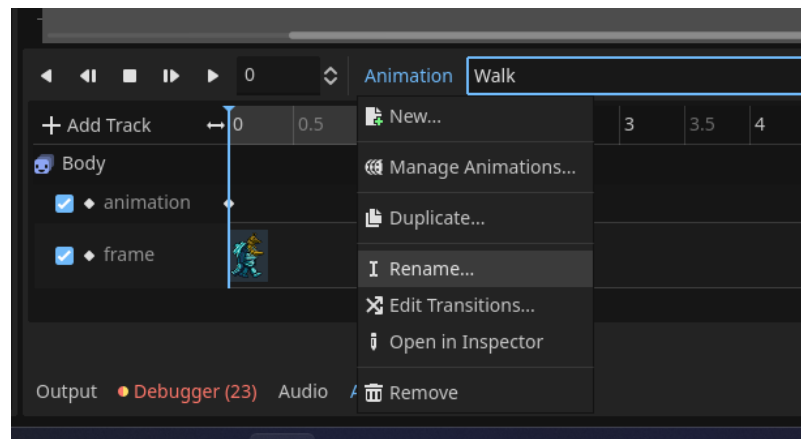
Now let's test the game.



It works! Our new character is flying! But there are new errors that say that animation “Idle” is not found. “Idle” is the equivalent of “Walk” but for flying characters, so let’s rename our walking animation. Open your skin scene and select the walking animation:



Now let’s rename it:

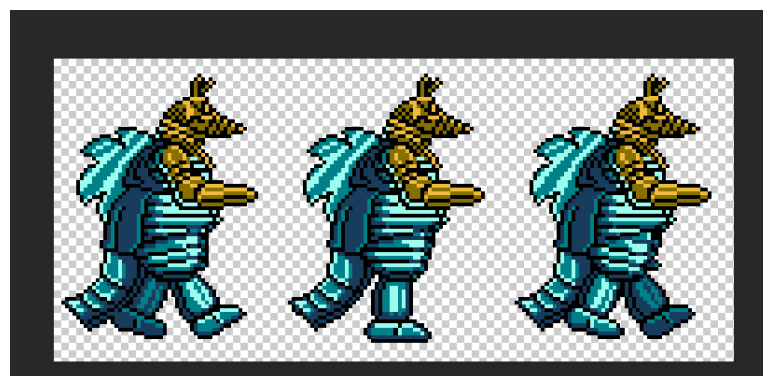


Now let's try the game again. No errors appear! But you will notice that the animation is stuck again. This time the animation actually works like how it is described in the animation player, not by code. This is because for walking characters the walking frame speed changes depending on the direction of their movement, but since Mothra's wings don't depend on the movement flying characters use a different behavior.

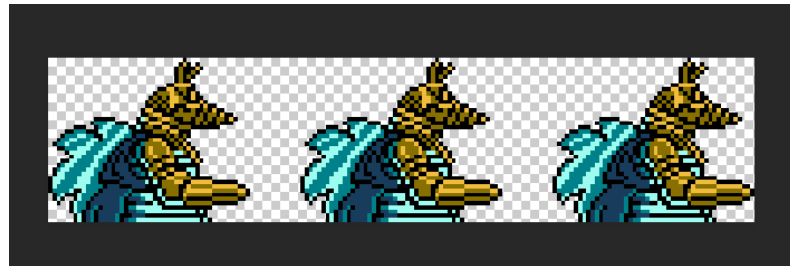
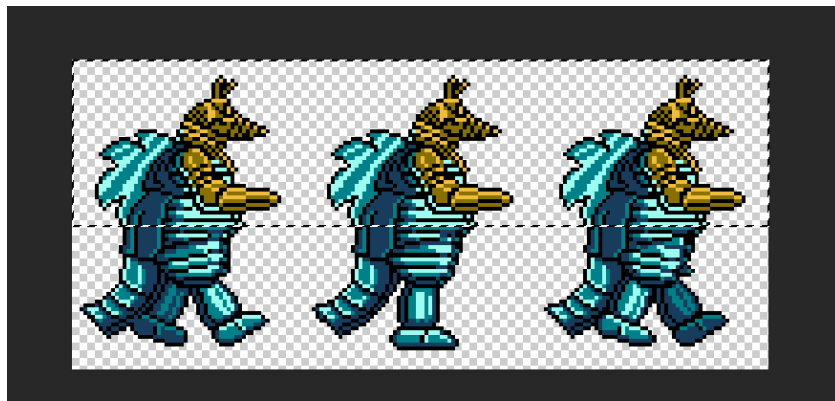
## Separating the top half of the character sprites (optional)

In the "Creating new character skin" chapter I described why you may want to separate the top half of a character's sprites from the bottom half. So now let's implement that for our new character!

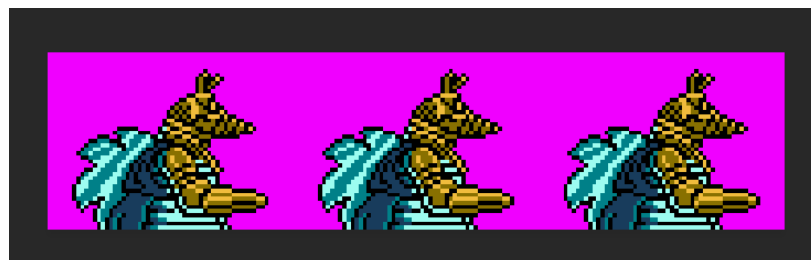
First, let's take a look at our sprites again.



The easiest way to make sprites for the top part is to just copy half of the image and paste it into a new file.



Hm, the new image looks smaller. See, when you copy an image in Photoshop, it ignores the transparent pixels. So you can change the transparent color in your image to something else, like purple, using bucket fill tool.



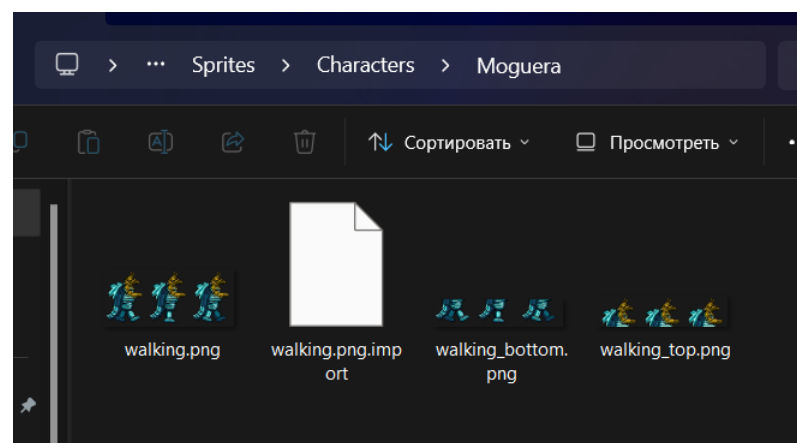
Now we can remove the background color.



Now we can do the same for the bottom part of the sprites.



Now save both parts into separate files in the character's sprites folder.





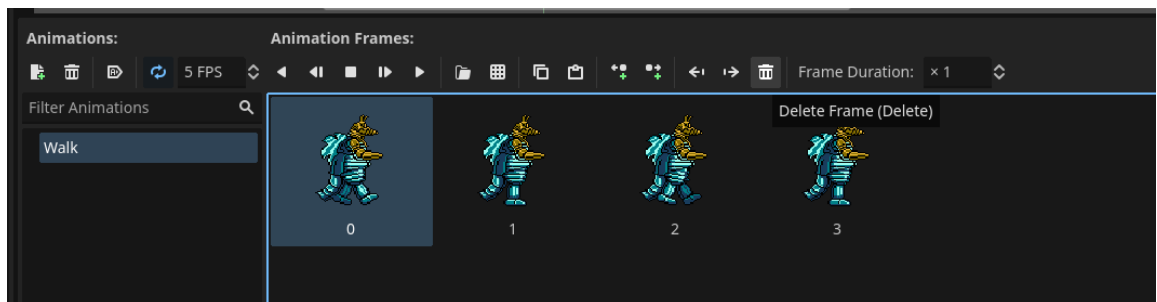
Ignore the .png.import file, it's a special file Godot generated automatically.

Since Moguera's top half stays the same throughout the animation, I changed the sprites a bit to later indicate what frame is currently shown on screen.

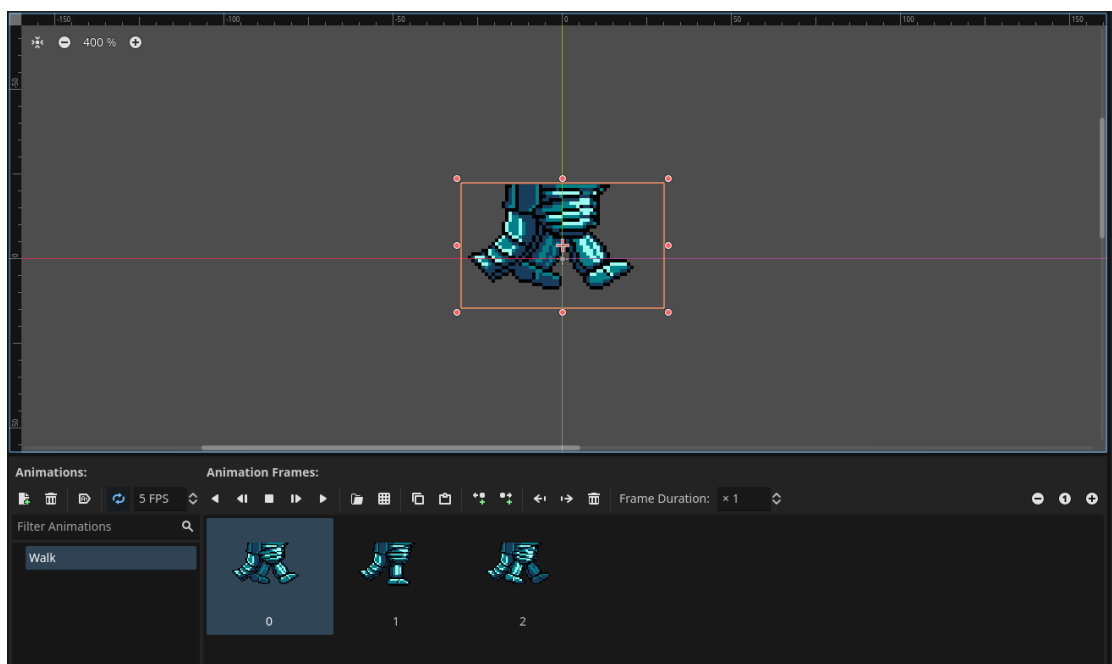


Don't judge, I'm not an artist, I'm just a programmer :D

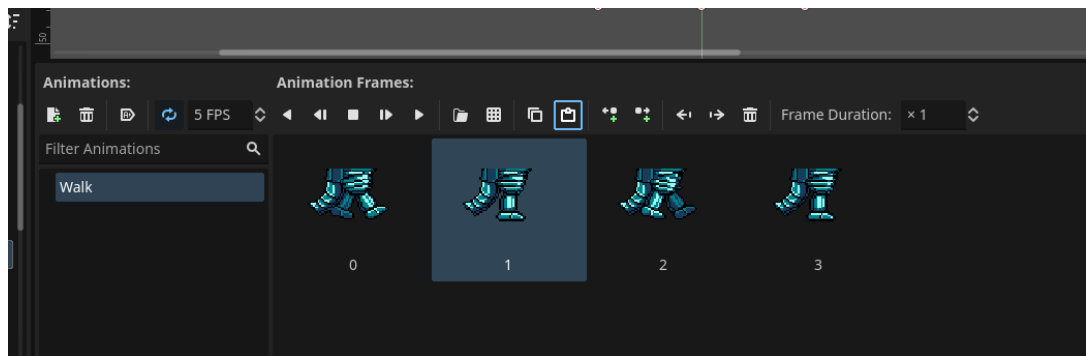
Remember how we imported the sprites in "Importing sprites into Godot" section? We can now do the same, but this time we will load the bottom half sprites into the "Body" node. But first, you will need to delete the previous frames:



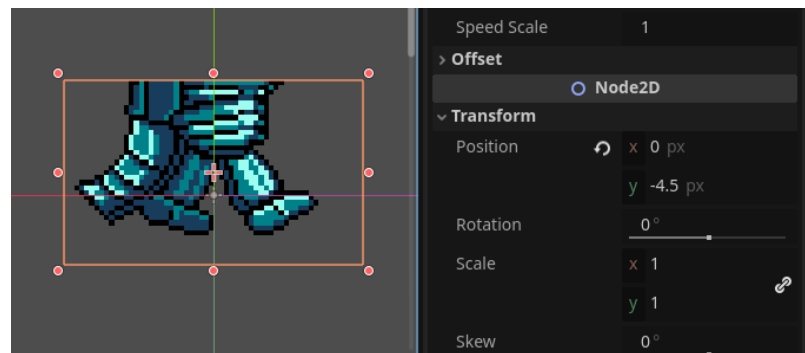
Now let's import the sprites as described in the "Importing sprites into Godot" section.



Remember how we duplicated the "standing" sprite to make the animation look alright when walking? You can do the same thing here again.

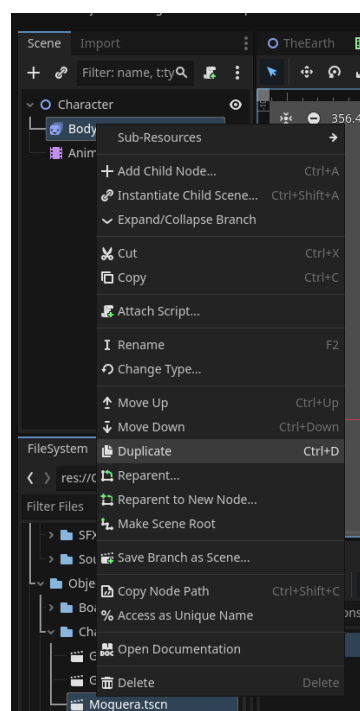


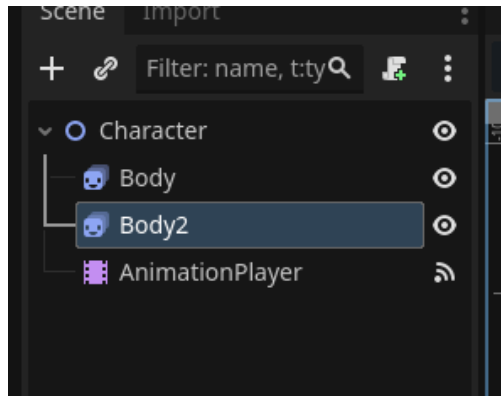
Nice! But in case if the sprites look jittery again if you zoom in/out or move the editor view, the width or height of each sprite became odd if they were even before (or they became even if they were odd before). The reason the sprite becomes jittery is described in the same “Importing sprites into Godot” section. In that case you can move the sprite’s X or Y position by half a pixel and see if the shakiness stops.



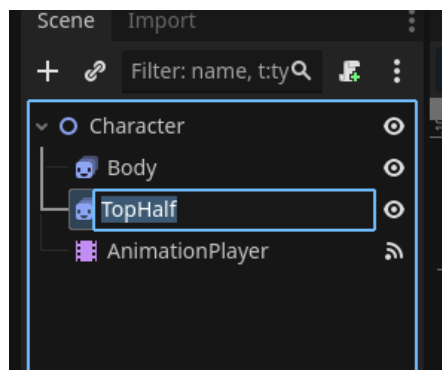
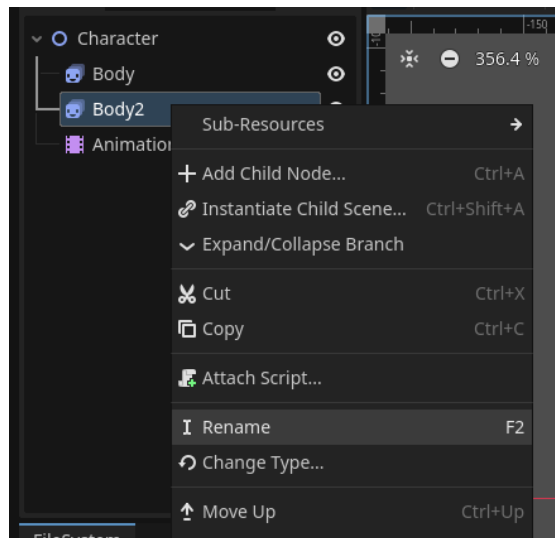
Now let’s create a node for the top half. We can create a duplicate of the bottom half node to make the process simpler for this tutorial.

Right click on the “Body” node and select “Duplicate”:

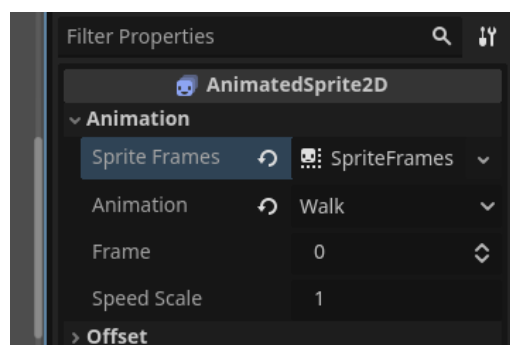




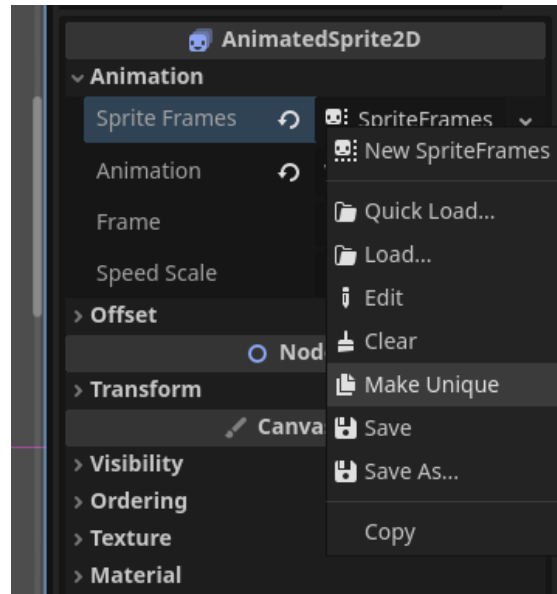
Rename it by right clicking and selecting “Rename”:



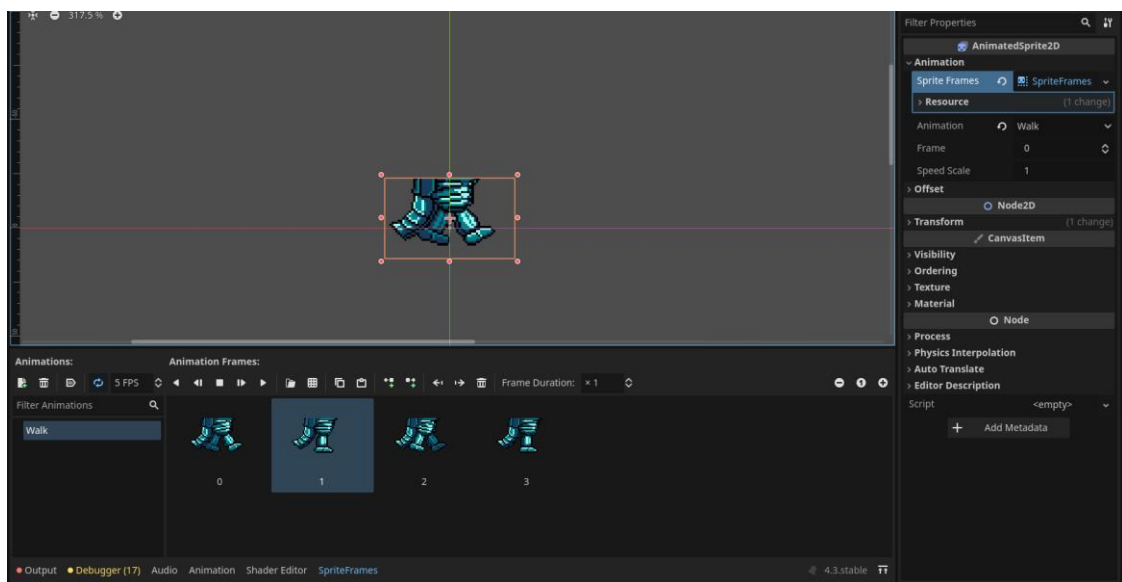
Then press Enter on your keyboard. Nice, now let’s modify it. But before we do that, we need to do one more thing. Select the top half node and look at its properties:



There's a "Sprite Frames" property with a special resource type, "SpriteFrames". Since we duplicated the node, both top half and bottom half share that resource, so they use the same sprites at the moment. How to change that? We need to make the top half's sprite frames unique. Right click on "SpriteFrames" text and select "Make unique".

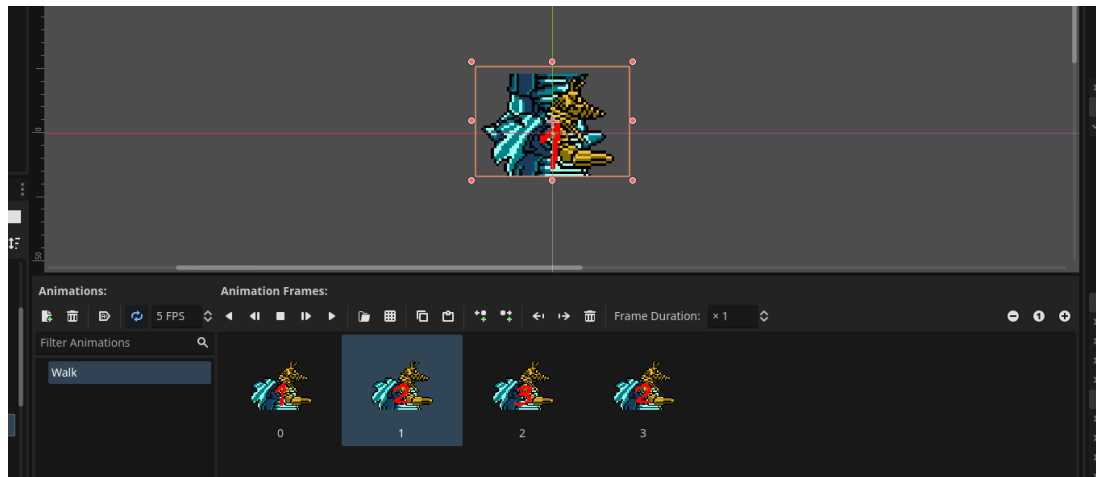


It may look like nothing happened, and after selecting the "SpriteFrames" text again we can see the same sprites of the node.



But now that resource is unique, so we can now change the sprites here without consequences. You can try importing the sprites without making that resource unique and you will see why we did that.

To import the top half sprites, you can use the same technique we used for bottom half.



Nice, they're imported. Now let's move that node until it looks alright. And if it looks jittery again, do the same thing as we did above: add half a pixel for the node's position.



Great, but the character is not in the middle of the scene, so now let's move both nodes again until it looks alright.



That should do it for now, let's test it in-game! If the character's sprite is a bit in the ground again, you can always move it in the skin scene.



We now have bottom half and top half separate! But how would we make the top half change its frame when the character is moving?

Do you remember how we removed a script from the top character skin node (named “Character”) that was a leftover from Godzilla’s skin? That script also controls the top part’s frame id to be the same as the bottom part when possible (it doesn’t do that if the top part’s animation isn’t set to walking). Let’s take a look at that script:

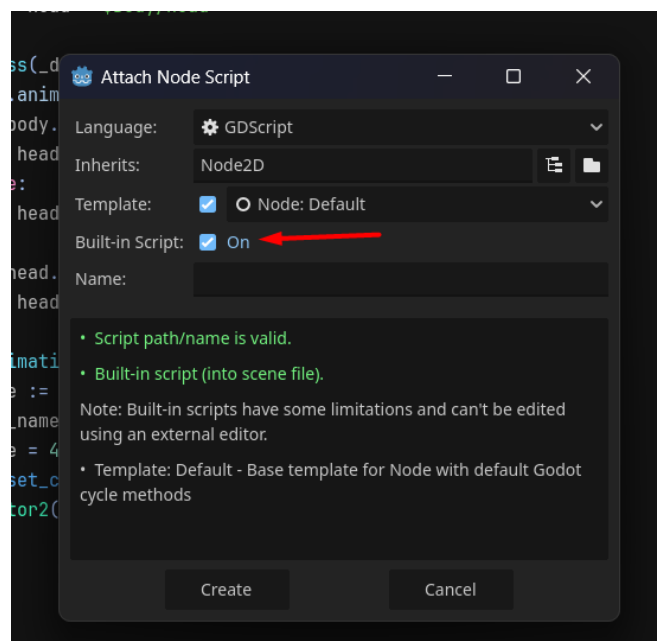
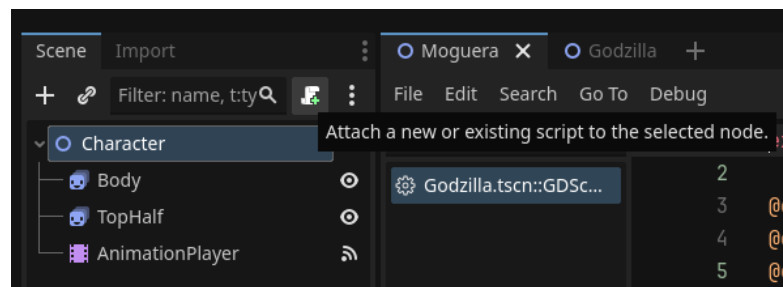
```

1  extends Node2D
2
3  @onready var parent: PlayerCharacter = get_parent()
4  @onready var offset_y = $Body/Head.position.y
5  @onready var body = $Body
6  @onready var head = $Body/Head
7
8  func _process(_delta: float) -> void:
9      if body.animation == "Walk":
10         if body.frame == 2 or body.frame == 6:
11             head.position.y = offset_y + 1
12         else:
13             head.position.y = offset_y
14
15         if head.animation == "Walk":
16             head.frame = body.frame
17
18  func _on_animation_started(anim_name: StringName) -> void:
19      var size := 56
20      if anim_name == "Crouch":
21          size = 40
22      parent.set_collision(Vector2(20, size),
23          Vector2(0, -1 + (56 - size) / 2))
24

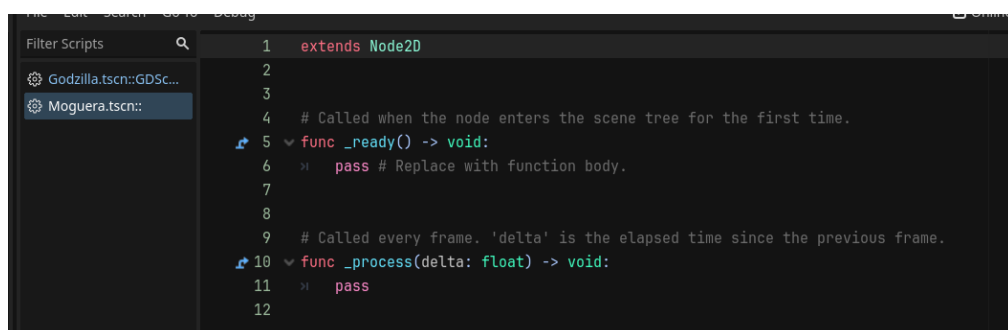
```

We can see that this script sets the top part's (which is called "head" here) frame property to be the same as the bottom part's ("body") frame property every frame (because it's in a special "\_process" function) under the conditions that both parts are in walking state. (There's also other code between those events but that only changes the top part's Y position when Godzilla is walking, it doesn't really apply to our character here)

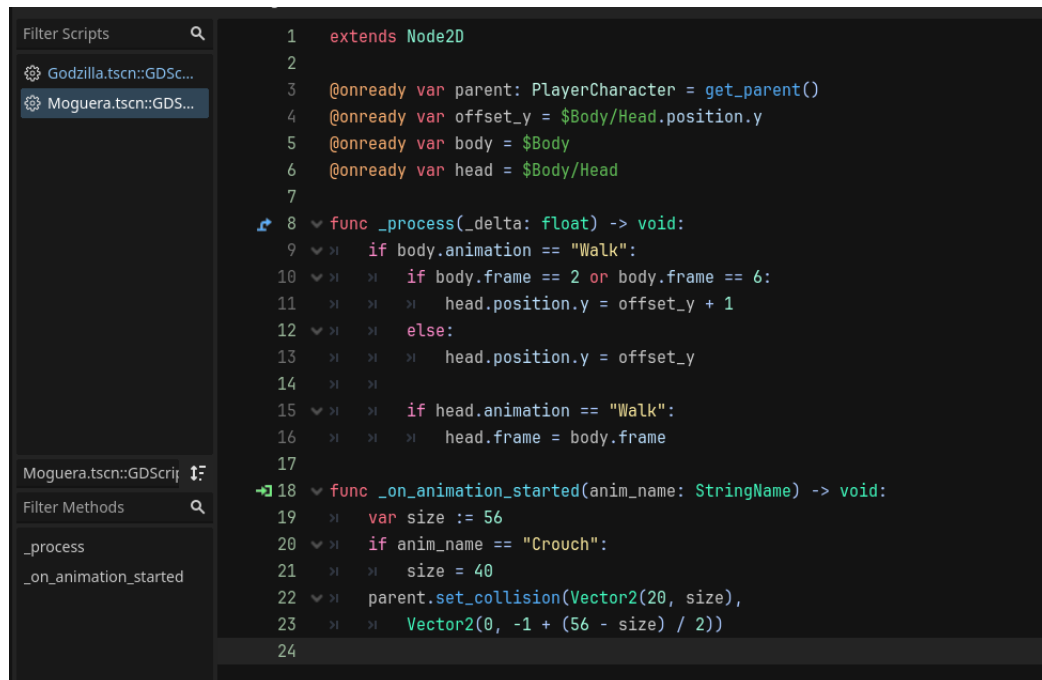
After you copied the code, go to your character's skin scene and press this button to add a new script:



Make sure it's a built-in script (the code is so small so I don't think it's too practical for it to be in a separate file, so this script is going to be saved inside of the scene's text).

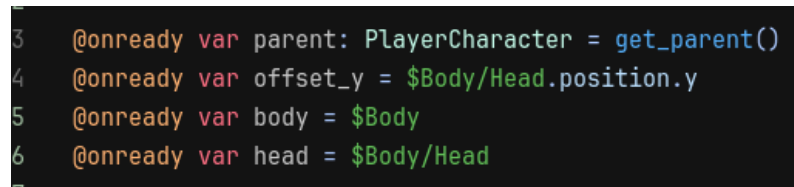


You'll probably already have `_ready` and `_process` functions made for you in that script by the way. But you can delete everything in that script and paste what you copied earlier:



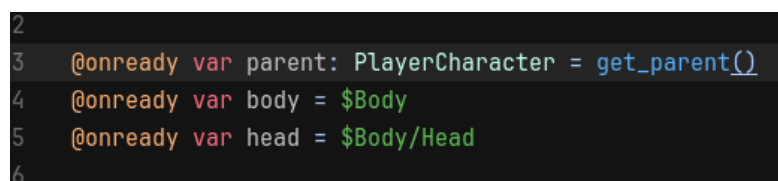
```
1 extends Node2D
2
3 @onready var parent: PlayerCharacter = get_parent()
4 @onready var offset_y = $Body/Head.position.y
5 @onready var body = $Body
6 @onready var head = $Body/Head
7
8 func _process(delta: float) -> void:
9     if body.animation == "Walk":
10         if body.frame == 2 or body.frame == 6:
11             head.position.y = offset_y + 1
12         else:
13             head.position.y = offset_y
14
15     if head.animation == "Walk":
16         head.frame = body.frame
17
18 func _on_animation_started(anim_name: StringName) -> void:
19     var size := 56
20     if anim_name == "Crouch":
21         size = 40
22     parent.set_collision(Vector2(20, size),
23         Vector2(0, -1 + (56 - size) / 2))
24
```

Now let's start fixing. We'll start with `@onready` variables at the top:



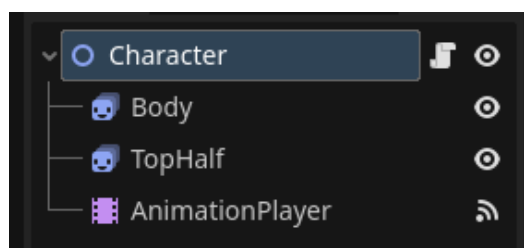
```
3 @onready var parent: PlayerCharacter = get_parent()
4 @onready var offset_y = $Body/Head.position.y
5 @onready var body = $Body
6 @onready var head = $Body/Head
7
```

Since we don't need to modify the top part's Y position we can remove "`offset_y`"



```
2
3 @onready var parent: PlayerCharacter = get_parent()
4 @onready var body = $Body
5 @onready var head = $Body/Head
6
```

We also changed the top part's node name so "`$Body/Head`" won't work, let's look at our scene nodes:



There is a node named "Body" so the body variable will work. And then there's "TopHalf" node, so let's use that name instead of "Body/Head":



```
@onready var parent: PlayerCharacter = get_parent()
@onready var body = $Body
@onready var head = $TopHalf
```

You'll notice some errors that Godot will show you in the script:

```
6 |
7 ▼ func _process(_delta: float) -> void:
8 ▼> if body.animation == "Walk":
9 ▼> > if body.frame == 2 or body.frame == 6:
10 > > > head.position.y = offset_y + 1
11 ▼> > else:
12 > > > head.position.y = offset_y
13 > >
14 ▼> > if head.animation == "Walk":
15 > > > head.frame = body.frame
16
17 ▼ func _on_animation_started(anim_name: StringName) -> void:
18 > var size := 56
19 ▼> if anim_name == "Crouch":
20 > > size = 40
21 ▼> parent.set_collision(Vector2(20, size),
22 > > Vector2(0, -1 + (56 - size) / 2))
< Error at (10, 31): Identifier "offset_y" not declared in the current scope.
```

Remember that code that uses top part's Y position? We don't need it for our character, so remove it:

```
▼ func _process(_delta: float) -> void:
▼> if body.animation == "Walk":
▼> > if head.animation == "Walk":
> > > head.frame = body.frame
```

Nested if-statements like this can be merged using "and" operator like this:

```
▼ func _process(_delta: float) -> void:
▼> if body.animation == "Walk" and head.animation == "Walk":
> > head.frame = body.frame
```

There's also this leftover piece of code:

```
▼ func _on_animation_started(anim_name: StringName) -> void:
> var size := 56
▼> if anim_name == "Crouch":
> > size = 40
▼> parent.set_collision(Vector2(20, size),
> > Vector2(0, -1 + (56 - size) / 2))
```

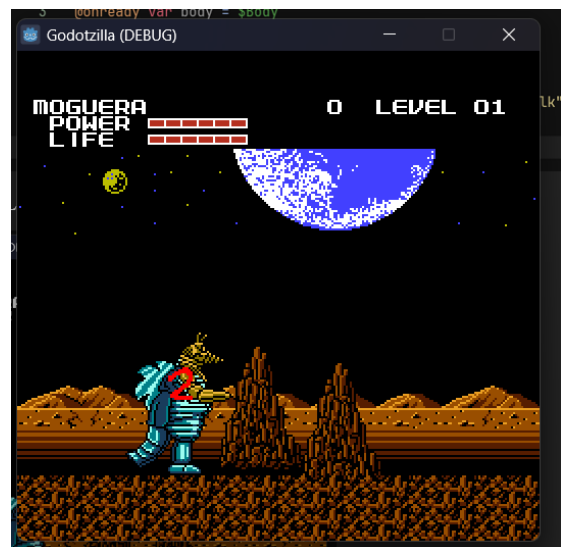
Our character doesn't have a crouch animation, so you can remove this whole function.

```
1 extends Node2D
2
3 @onready var parent: PlayerCharacter = get_parent()
4 @onready var body = $Body
5 @onready var head = $TopHalf
6
7 func _process(_delta: float) -> void:
8     if body.animation == "Walk" and head.animation == "Walk":
9         head.frame = body.frame
10
```

Almost perfect! Now you can notice that the “parent” variable is unused, so you can remove that one as well:

```
1 extends Node2D
2
3 @onready var body = $Body
4 @onready var head = $TopHalf
5
6 func _process(_delta: float) -> void:
7     if body.animation == "Walk" and head.animation == "Walk":
8         head.frame = body.frame
9
```

Perfect! Let's try it out in-game.



It works! Now our character has separate top and bottom parts.