This is the part 2 of "How to create a new character" tutorial.

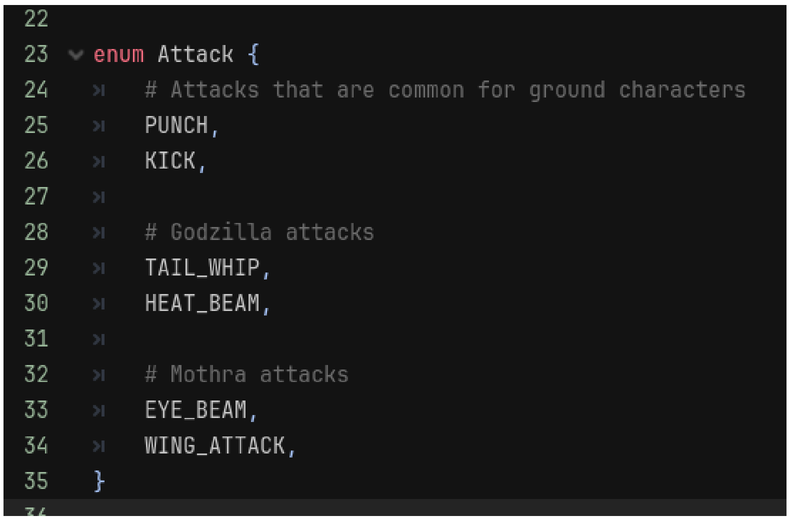
## Contents

## Adding new attacks

I came up with a pretty silly move for Moguera: basically, let's make it move about 30 pixels forward, and then back to its place, all while attacking. I know that the original Moguera didn't have that move, but it might be good enough to show how to add new moves...?

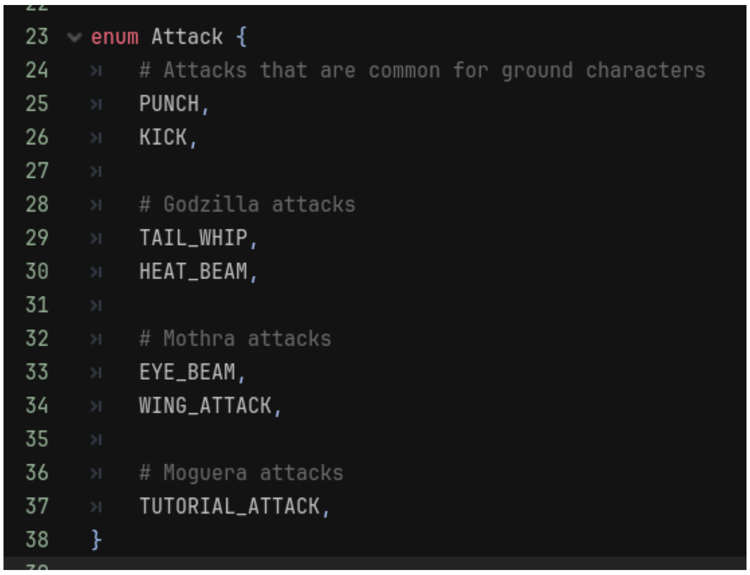
So, let's begin.

Let's add a new attack type first. Open PlayerCharacter.gd, scroll up and find this enum:



```

enum Attack {
	# Attacks that are common for ground characters
	PUNCH,
	KICK,
	
	# Godzilla attacks
	TAIL_WHIP,
	HEAT_BEAM,
	
	# Mothra attacks
	EYE_BEAM,
	WING_ATTACK,
}
```

Add a new entry for our character:

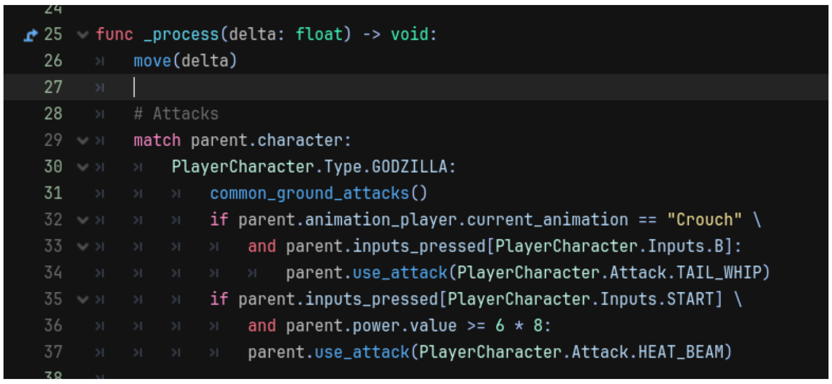


```
enum Attack {
	# Attacks that are common for ground characters
	PUNCH,
	KICK,

	# Godzilla attacks
	TAIL_WHIP,
	HEAT_BEAM,

	# Mothra attacks
	EYE_BEAM,
	WING_ATTACK,

	# Moguera attacks
	TUTORIAL_ATTACK,
}
```

Good enough.

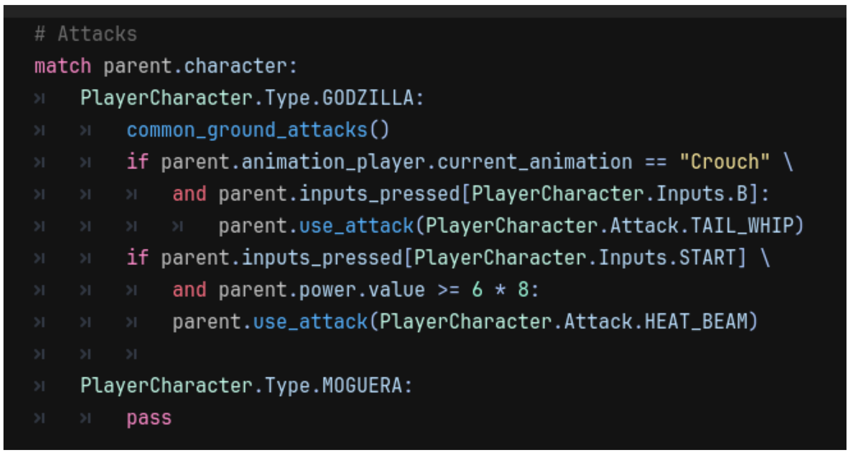
Since Moguera is a walking character and not a flying character (how to do that is one of the topics of this part of the tutorial), we should modify the walking state code, its file is “Scripts/Objects/Characters/Walk.gd”.

If you scroll you can see the “_process” function and Godzilla’s attacks:



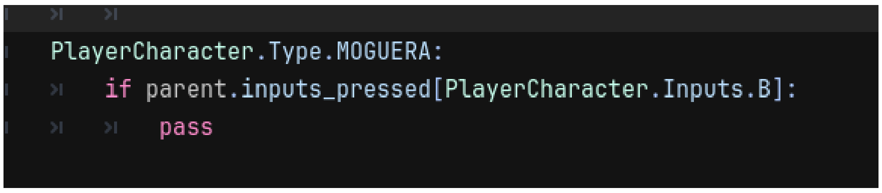
```
func _process(delta: float) -> void:
	move(delta)
	
	# Attacks
	match parent.character:
		PlayerCharacter.Type.GODZILLA:
			common_ground_attacks()
			if parent.animation_player.current_animation == "Crouch" \
				and parent.inputs_pressed[PlayerCharacter.Inputs.B]:
					parent.use_attack(PlayerCharacter.Attack.TAIL_WHIP)
			if parent.inputs_pressed[PlayerCharacter.Inputs.START] \
				and parent.power.value >= 6 * 8:
				parent.use_attack(PlayerCharacter.Attack.HEAT_BEAM)
```

Let’s make a new section for our character:



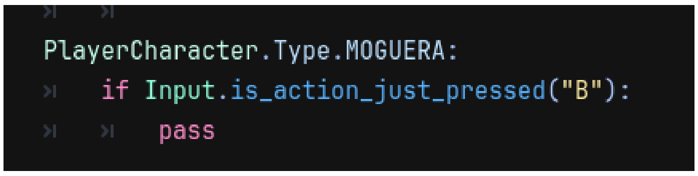
```
# Attacks
match parent.character:
	PlayerCharacter.Type.GODZILLA:
		common_ground_attacks()
		if parent.animation_player.current_animation == "Crouch" \
			and parent.inputs_pressed[PlayerCharacter.Inputs.B]:
				parent.use_attack(PlayerCharacter.Attack.TAIL_WHIP)
		if parent.inputs_pressed[PlayerCharacter.Inputs.START] \
			and parent.power.value >= 6 * 8:
			parent.use_attack(PlayerCharacter.Attack.HEAT_BEAM)
			
	PlayerCharacter.Type.MOGUERA:
		pass
```

Now let’s add a code that checks input like the one used for Godzilla:



```
PlayerCharacter.Type.MOGUERA:
    if parent.inputs_pressed[PlayerCharacter.Inputs.B]:
        pass
```

And also mind that using default Godot’s “Input” singleton will also work

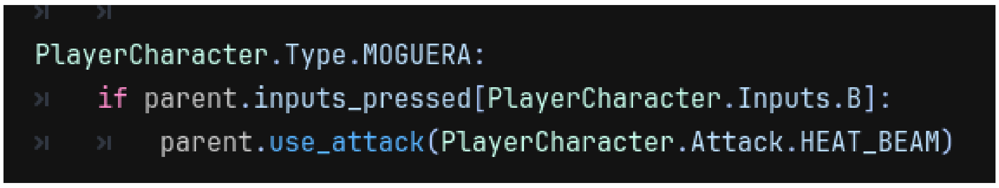


```
PlayerCharacter.Type.MOGUERA:
    if Input.is_action_just_pressed("B"):
        pass
```

But there’s a drawback to this approach: if you want your character to be a boss (also explained in one of the topics here) it won’t be able to attack, because Input singleton only checks the player’s input, it can’t check the boss AI input simulations.

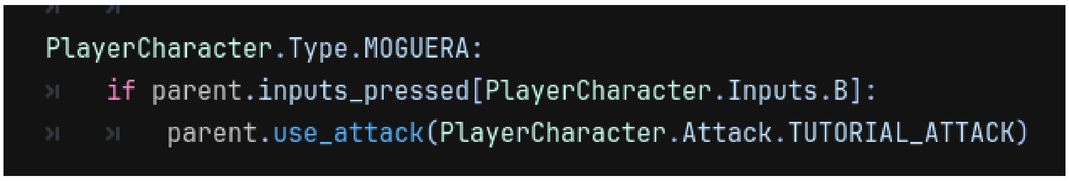
So here I will use the one that uses “parent.inputs_pressed” array.

Do you see that for Godzilla when all the conditions are met the attack is started using “parent.use_attack”? Add a similar line for your character:



```
PlayerCharacter.Type.MOGUERA:
    if parent.inputs_pressed[PlayerCharacter.Inputs.B]:
        parent.use_attack(PlayerCharacter.Attack.HEAT_BEAM)
```

But instead of using Godzilla’s attack type, use the one we created earlier:

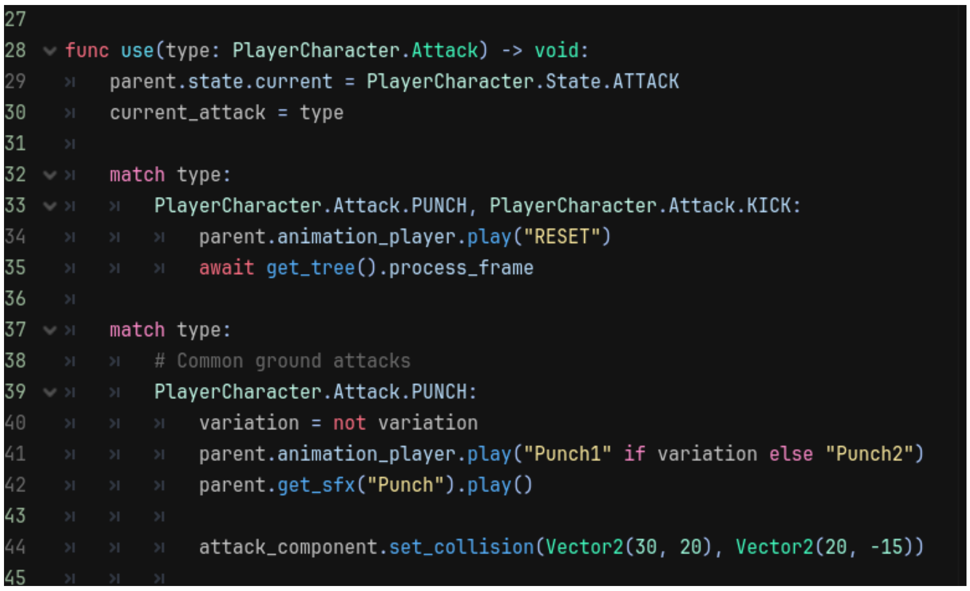


```
PlayerCharacter.Type.MOGUERA:
    if parent.inputs_pressed[PlayerCharacter.Inputs.B]:
        parent.use_attack(PlayerCharacter.Attack.TUTORIAL_ATTACK)
```

Nice. Now let’s add the actual attack.

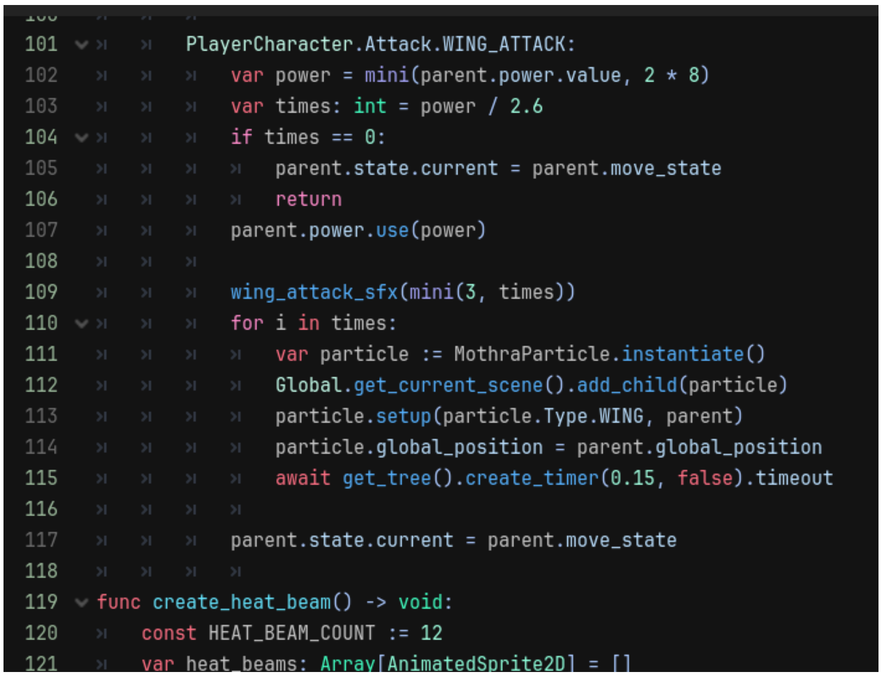
All the attacks are stored in “Scripts/Objects/Characters/Attack.gd” (if you’re an experienced programmer, you can make a separate file for your new attack and reference it in Attack.gd).

If you scroll a bit, you will find a “use” function. That’s where the attacks’ code is stored.



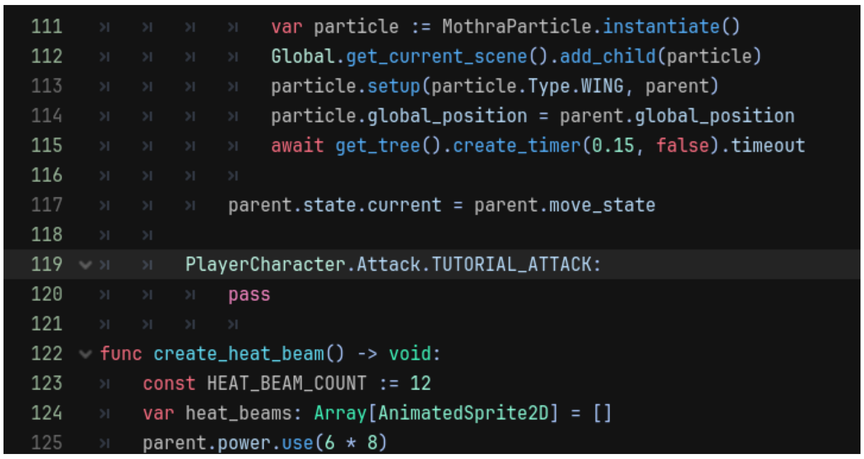
```

func use(type: PlayerCharacter.Attack) -> void:
    parent.state.current = PlayerCharacter.State.ATTACK
    current_attack = type

    match type:
        PlayerCharacter.Attack.PUNCH, PlayerCharacter.Attack.KICK:
            parent.animation_player.play("RESET")
            await get_tree().process_frame

    match type:
        # Common ground attacks
        PlayerCharacter.Attack.PUNCH:
            variation = not variation
            parent.animation_player.play("Punch1" if variation else "Punch2")
            parent.get_sfx("Punch").play()

            attack_component.set_collision(Vector2(30, 20), Vector2(20, -15))

```

Scroll to the bottom of the function where the function ends and a new one starts next:



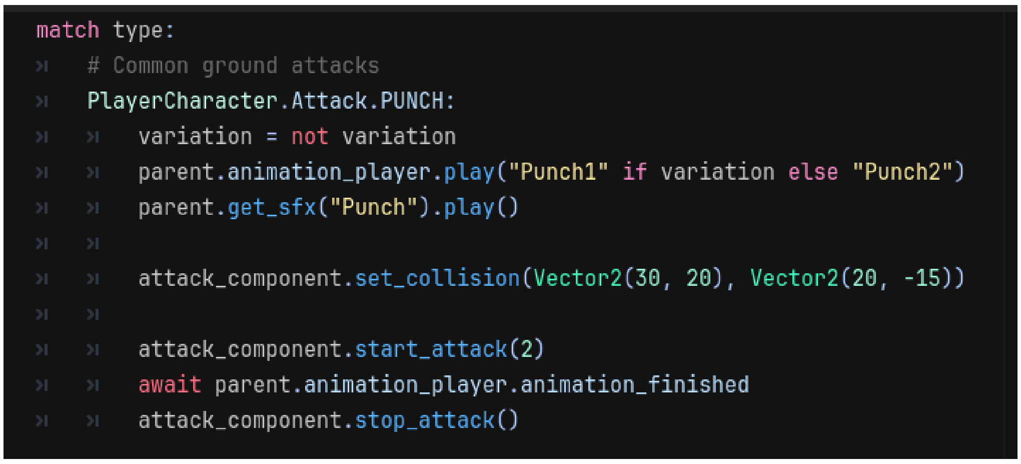
```
        PlayerCharacter.Attack.WING_ATTACK:
            var power = mini(parent.power.value, 2 * 8)
            var times: int = power / 2.6
            if times == 0:
                parent.state.current = parent.move_state
                return
            parent.power.use(power)

            wing_attack_sfx(mini(3, times))
            for i in times:
                var particle := MothraParticle.instantiate()
                Global.get_current_scene().add_child(particle)
                particle.setup(particle.Type.WING, parent)
                particle.global_position = parent.global_position
                await get_tree().create_timer(0.15, false).timeout

            parent.state.current = parent.move_state

func create_heat_beam() -> void:
    const HEAT_BEAM_COUNT := 12
    var heat_beams: Array[AnimatedSprite2D] = []
```

Now let's add a new entry for our tutorial attack:



```
                var particle := MothraParticle.instantiate()
                Global.get_current_scene().add_child(particle)
                particle.setup(particle.Type.WING, parent)
                particle.global_position = parent.global_position
                await get_tree().create_timer(0.15, false).timeout

            parent.state.current = parent.move_state

        PlayerCharacter.Attack.TUTORIAL_ATTACK:
            pass

func create_heat_beam() -> void:
    const HEAT_BEAM_COUNT := 12
    var heat_beams: Array[AnimatedSprite2D] = []
    parent.power.use(6 * 8)
```

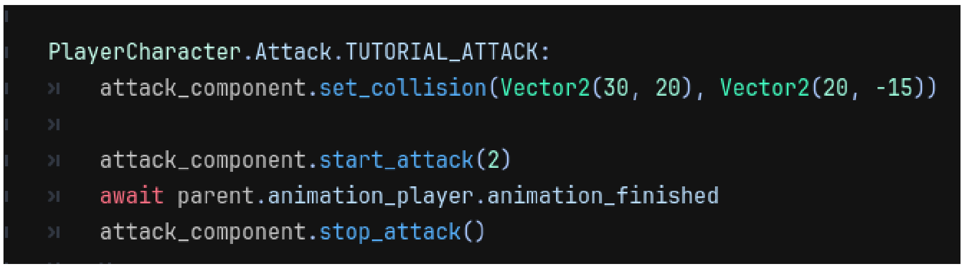
Before coding the actual attack, let's first see how the default punch is coded:



```
match type:
    # Common ground attacks
    PlayerCharacter.Attack.PUNCH:
        variation = not variation
        parent.animation_player.play("Punch1" if variation else "Punch2")
        parent.get_sfx("Punch").play()
        
        attack_component.set_collision(Vector2(30, 20), Vector2(20, -15))
        
        attack_component.start_attack(2)
        await parent.animation_player.animation_finished
        attack_component.stop_attack()
```

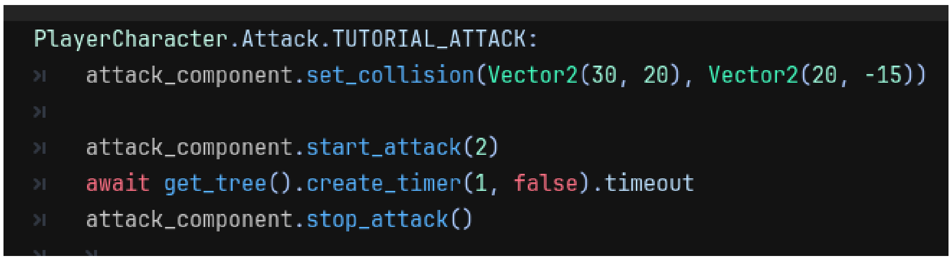
You can see that the attack component's collision changes (the first parameter is the size, the other one is offset that is relative to the character's center position) and also that the attack component starts the attack, the code waits a bit and then the attack stops.

Let's copy these lines of code.



```
PlayerCharacter.Attack.TUTORIAL_ATTACK:
    attack_component.set_collision(Vector2(30, 20), Vector2(20, -15))
    
    attack_component.start_attack(2)
    await parent.animation_player.animation_finished
    attack_component.stop_attack()
```

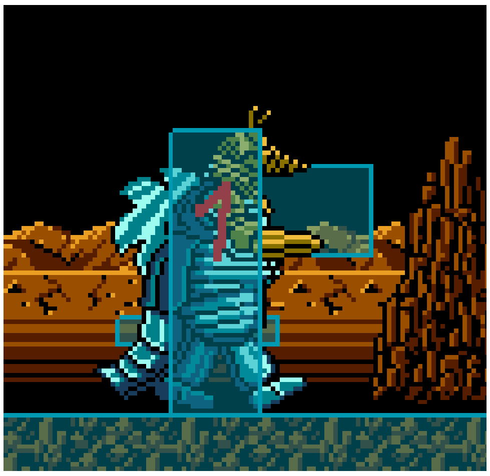
Since we're not playing any animations, we can change the parameter to that await to something different, for example:



```
PlayerCharacter.Attack.TUTORIAL_ATTACK:
    attack_component.set_collision(Vector2(30, 20), Vector2(20, -15))
    
    attack_component.start_attack(2)
    await get_tree().create_timer(1, false).timeout
    attack_component.stop_attack()
```

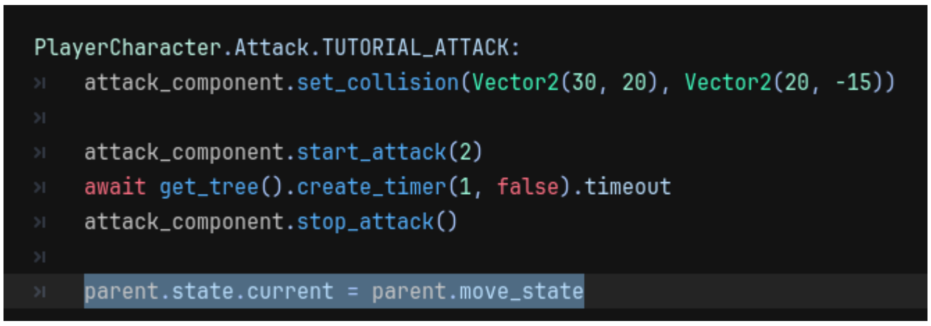
This little line of code waits 1 second before going to the next line, and it also doesn't ignore the game's pausing (because of the "false" parameter).

If you enable visible collision shapes in the Godot editor and try the game, you will see this when you press action B (by default it's keyboard key Z):



This little rectangle shows up for 1 second and then disappears. But after that you can't attack again. Why is that?

My bad, I forgot to tell you to add this line:

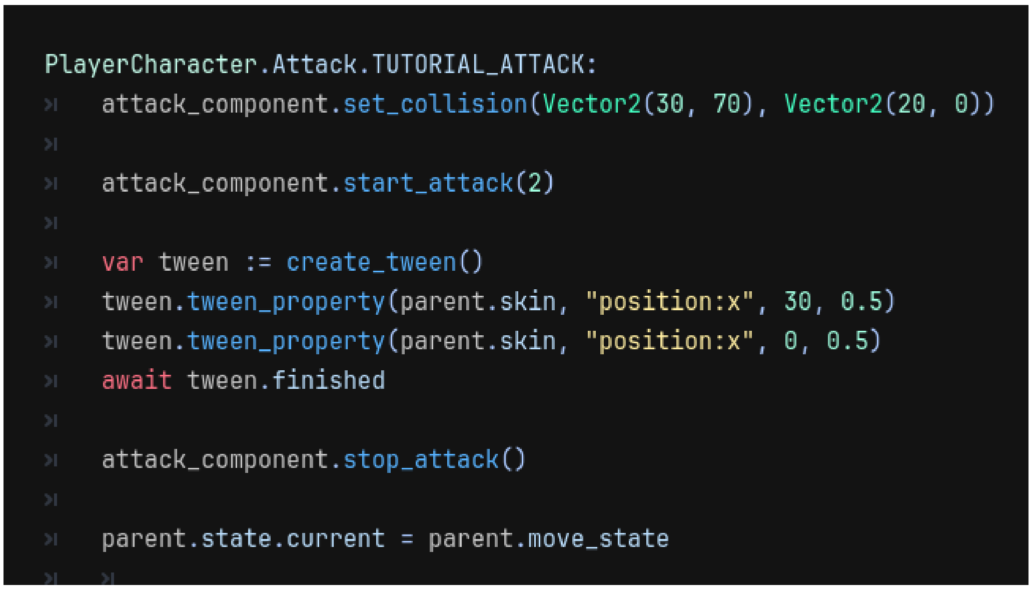


```
PlayerCharacter.Attack.TUTORIAL_ATTACK:
	attack_component.set_collision(Vector2(30, 20), Vector2(20, -15))

	attack_component.start_attack(2)
	await get_tree().create_timer(1, false).timeout
	attack_component.stop_attack()

	parent.state.current = parent.move_state
```

Since we're in attacking state right now, and if we don't change the state back to moving the player will be stuck in attacking state forever.

If you're wondering, how we will be able to move the player's skin from the attack code, worry no more! The "parent" variable is actually referencing the current PlayerCharacter node, i.e. Moguera. In PlayerCharacter class, there's a "skin" property, which is what we need.

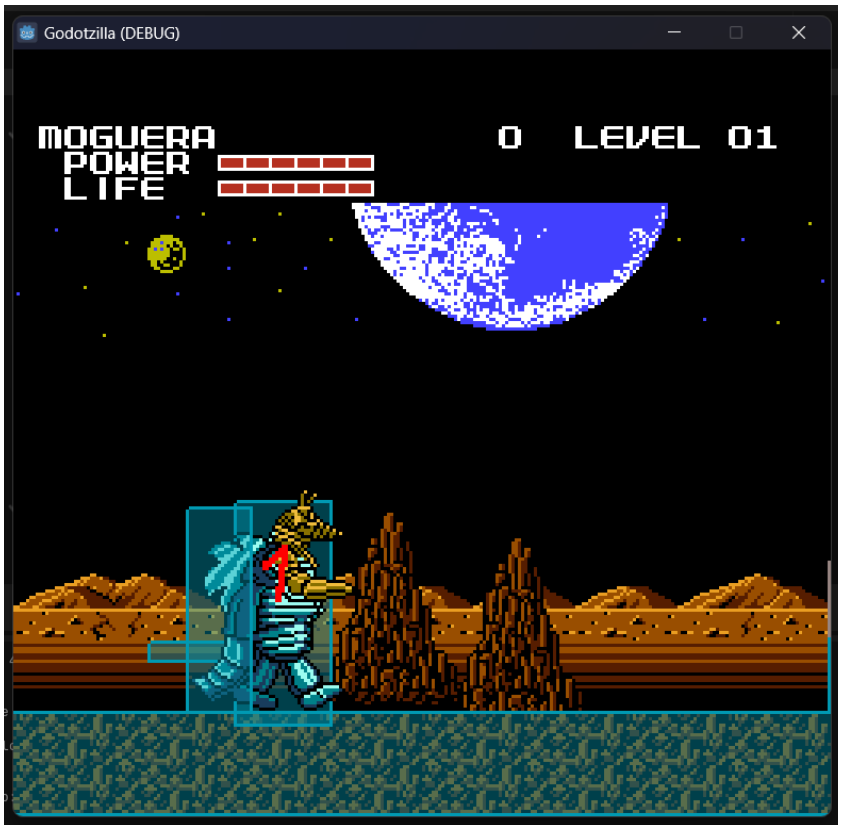
Now you can let your imagination flow :D (Sorry, this is not really a Godot programming tutorial :D)



```
PlayerCharacter.Attack.TUTORIAL_ATTACK:
	attack_component.set_collision(Vector2(30, 70), Vector2(20, 0))
	
	attack_component.start_attack(2)
	
	var tween := create_tween()
	tween.tween_property(parent.skin, "position:x", 30, 0.5)
	tween.tween_property(parent.skin, "position:x", 0, 0.5)
	await tween.finished
	
	attack_component.stop_attack()
	
	parent.state.current = parent.move_state
```

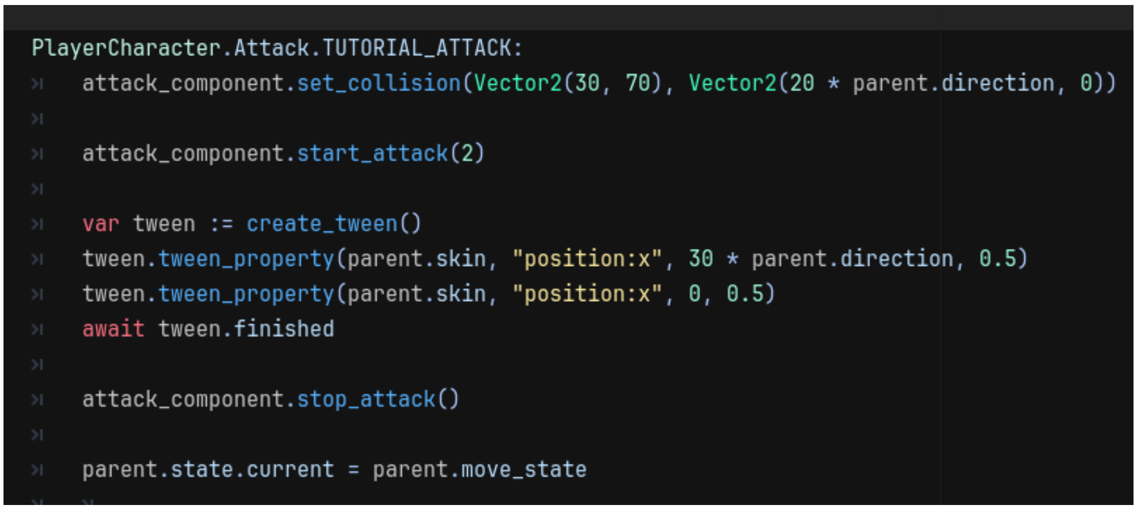
Also notice that I changed the attack component's parameters a bit.

If you try the game again, everything should work as expected:



(I don't know why this move looks so funny in-game)

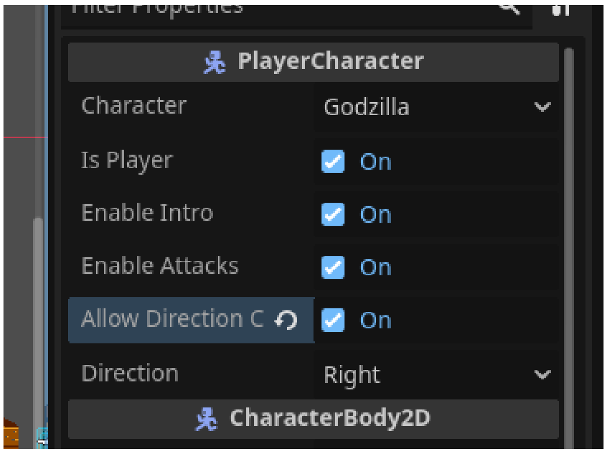
Before we finish this topic, I want you to also think about the bosses. As you know bosses face the left direction, while the players face the right direction. We only coded our attack for the players, if we made a boss out of Moguera this attack will look bugged (it will move backwards, and the attack component's shape will be behind Moguera). So, let's fix that.



```
PlayerCharacter.Attack.TUTORIAL_ATTACK:
	attack_component.set_collision(Vector2(30, 70), Vector2(20 * parent.direction, 0))

	attack_component.start_attack(2)

	var tween := create_tween()
	tween.tween_property(parent.skin, "position:x", 30 * parent.direction, 0.5)
	tween.tween_property(parent.skin, "position:x", 0, 0.5)
	await tween.finished

	attack_component.stop_attack()

	parent.state.current = parent.move_state
```

Notice that I added " * parent.direction" in 2 places: skin position and attack component's offset. Parent.direction is 1 if the player is facing right, -1 otherwise.

You can check that out if you open the Wilderness scene and change this property of the player:

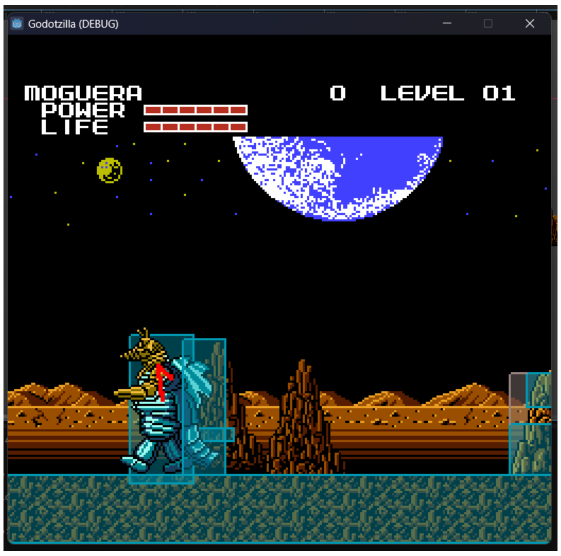


PlayerCharacter

| | |
|---|---|
| Character | Godzilla |
| Is Player | ✓ On |
| Enable Intro | ✓ On |
| Enable Attacks | ✓ On |
| Allow Direction C | ✓ On |
| Direction | Right |

CharacterBody2D

If you try the game again you will be able to change the player's direction when moving left.



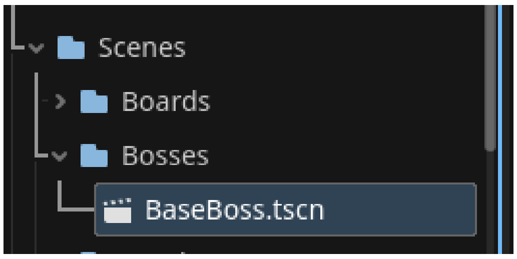
The attack works (the bigger rectangle is the attack component).

## Making a boss from the character

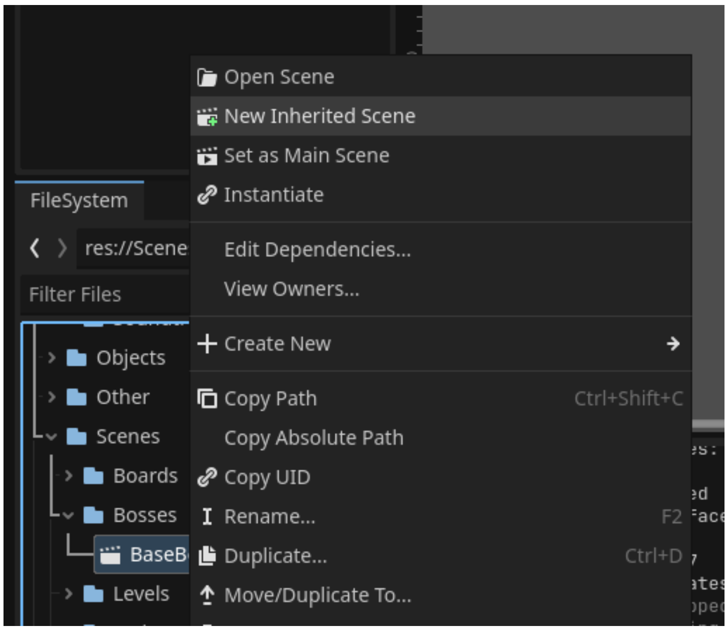
Now let’s make this character a boss!

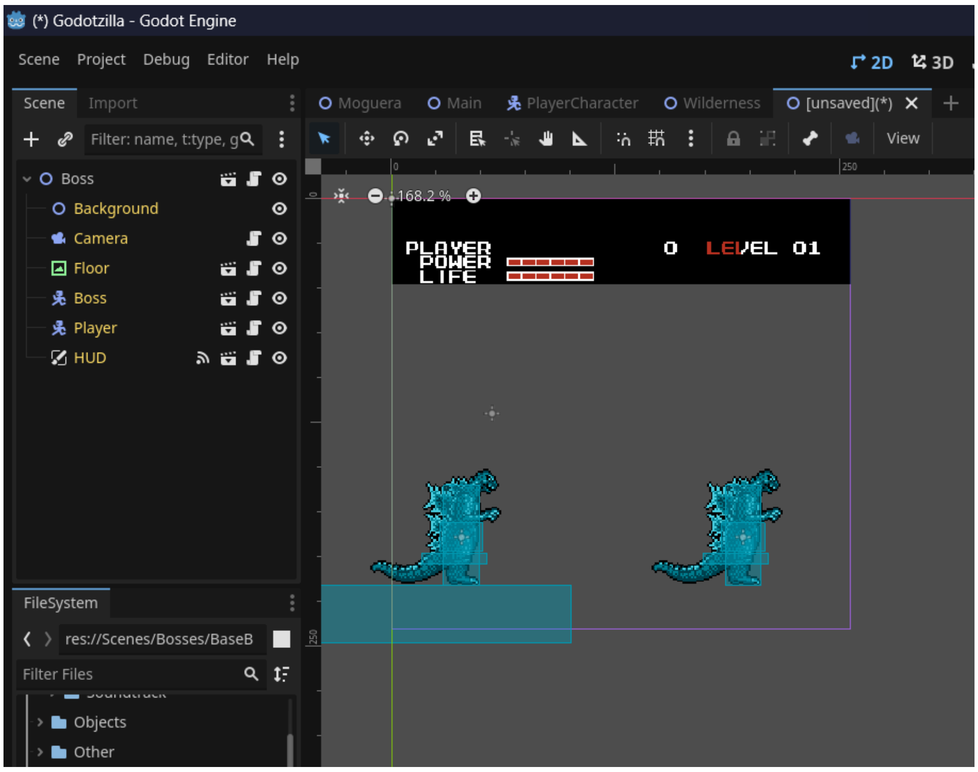
Yes, the characters can be both players and bosses. Yes, that’s how Mothra boss was done :D

Find this scene file: “Scenes/Bosses/BaseBoss.tscn”

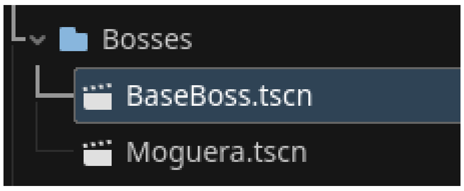


Let’s make a new scene that is based on this scene.

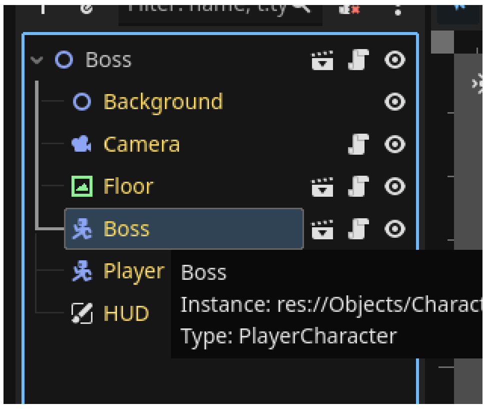




Save this scene to a file.

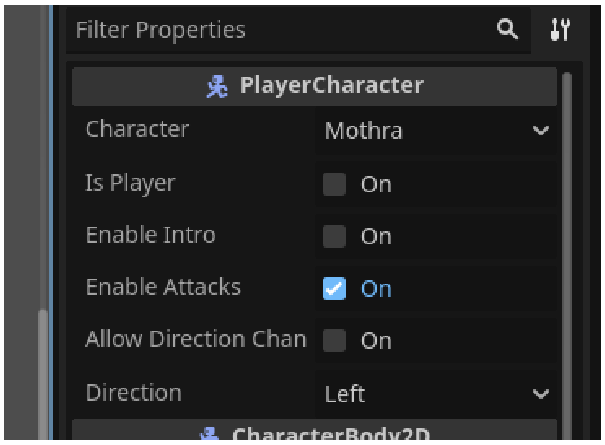


Now let's change the boss here to Moguera. Click on this node:



Boss
Instance: res://Objects/Charact
Type: PlayerCharacter

If you look at its properties, it's an actual PlayerCharacter.



PlayerCharacter

Character Mothra

Is Player ☐ On

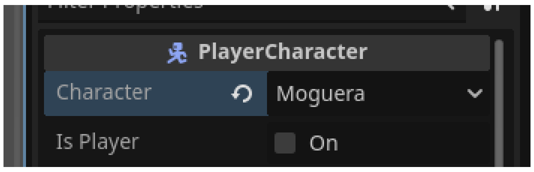
Enable Intro ☐ On

Enable Attacks ✓ On

Allow Direction Chan ☐ On

Direction Left

You know what to do.

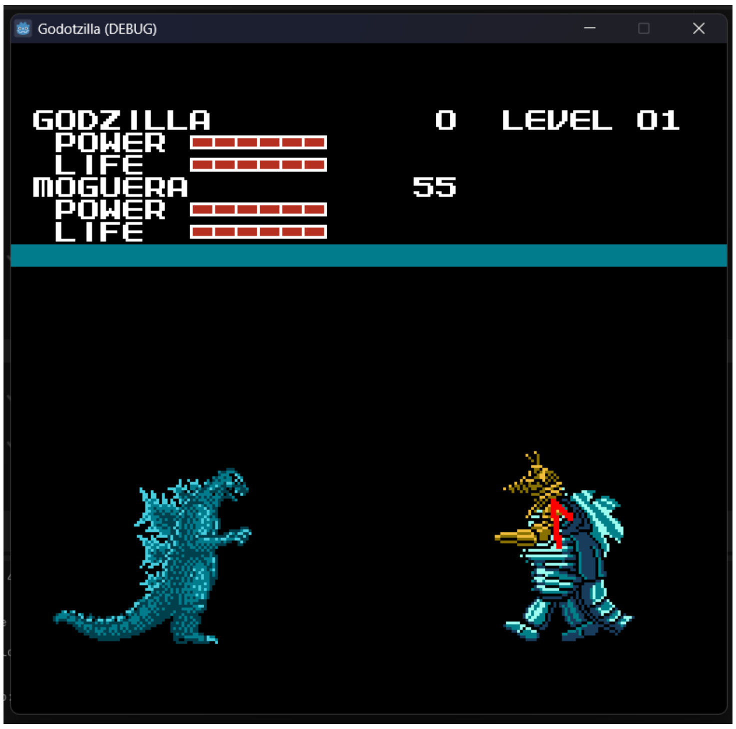


PlayerCharacter

Character Moguera

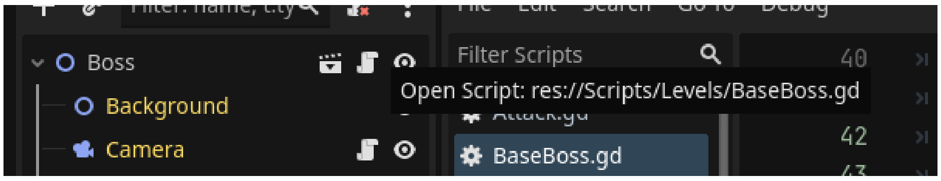
Is Player ☐ On

Now let's try it. Change the game's initial scene, change the level's default character back to Godzilla, and…



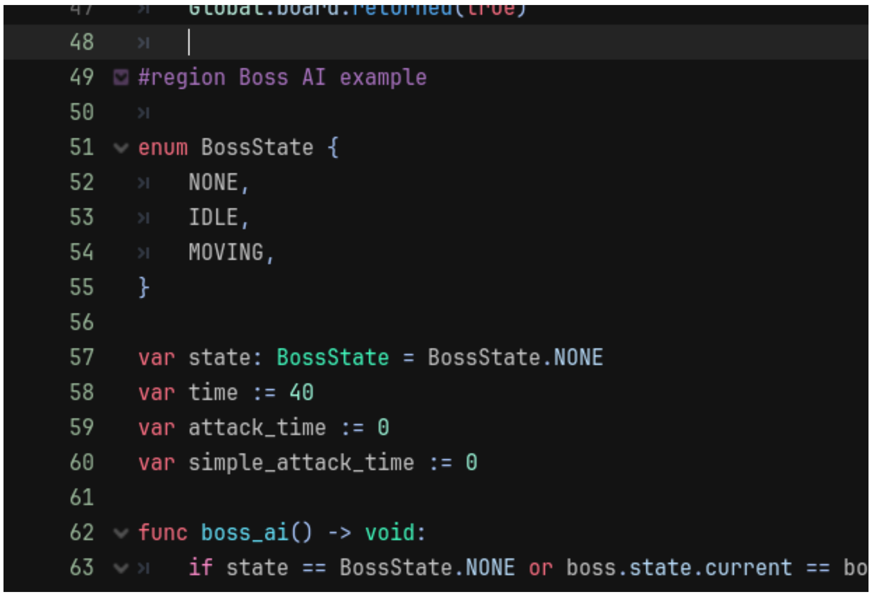
It works! But its AI is a bit buggy because it still uses Mothra's AI.

If you go back to the Moguera boss scene you may notice there's a script attached to it:



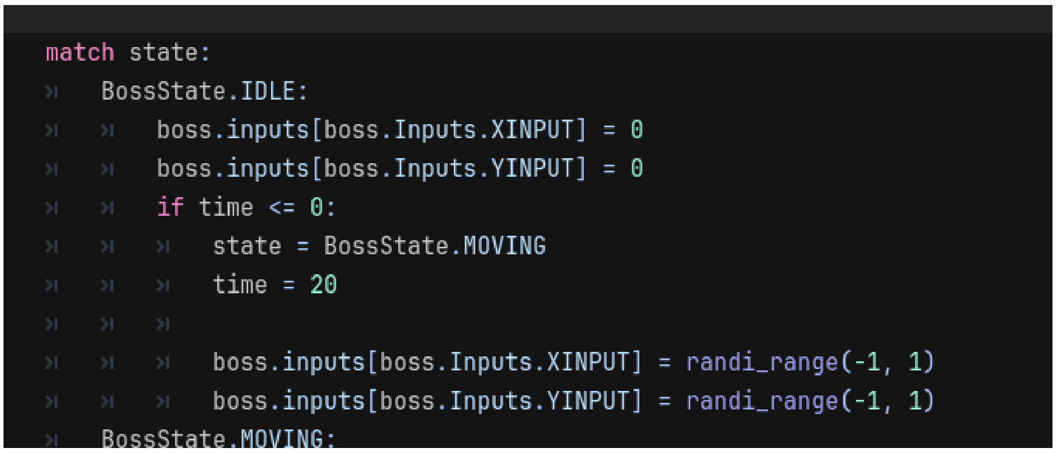
Open Script: res://Scripts/Levels/BaseBoss.gd

If you open it and scroll down, you can see the boss AI example

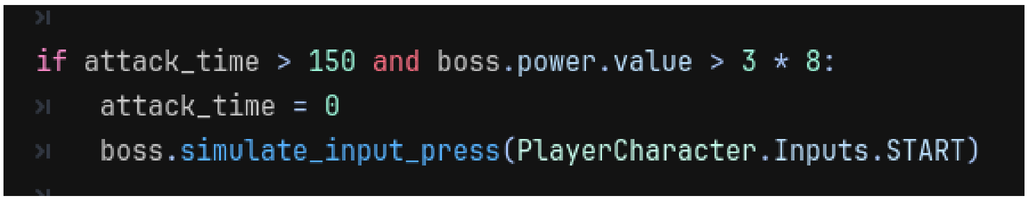


```
    Global.board.returned(true)
    
#region Boss AI example
    
enum BossState {
    NONE,
    IDLE,
    MOVING,
}

var state: BossState = BossState.NONE
var time := 40
var attack_time := 0
var simple_attack_time := 0

func boss_ai() -> void:
    if state == BossState.NONE or boss.state.current == bo
```

As you can see, it works by simulating inputs as if there was someone actually playing with this character and pressing something:

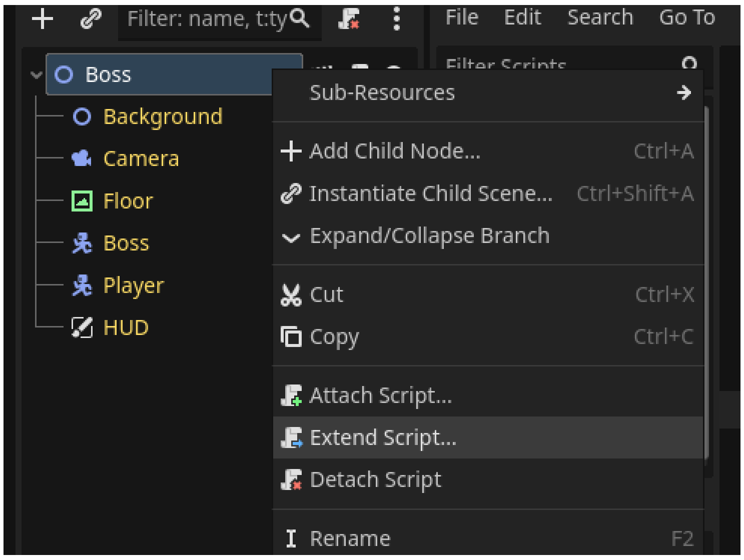


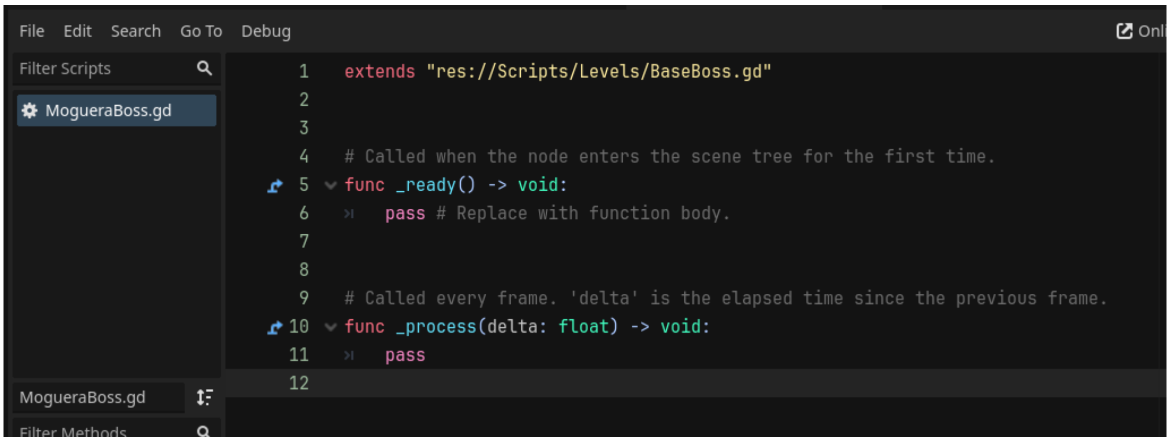
```
match state:
    BossState.IDLE:
        boss.inputs[boss.Inputs.XINPUT] = 0
        boss.inputs[boss.Inputs.YINPUT] = 0
        if time <= 0:
            state = BossState.MOVING
            time = 20
            
            boss.inputs[boss.Inputs.XINPUT] = randi_range(-1, 1)
            boss.inputs[boss.Inputs.YINPUT] = randi_range(-1, 1)
    BossState.MOVING:
```



```

if attack_time > 150 and boss.power.value > 3 * 8:
    attack_time = 0
    boss.simulate_input_press(PlayerCharacter.Inputs.START)
```

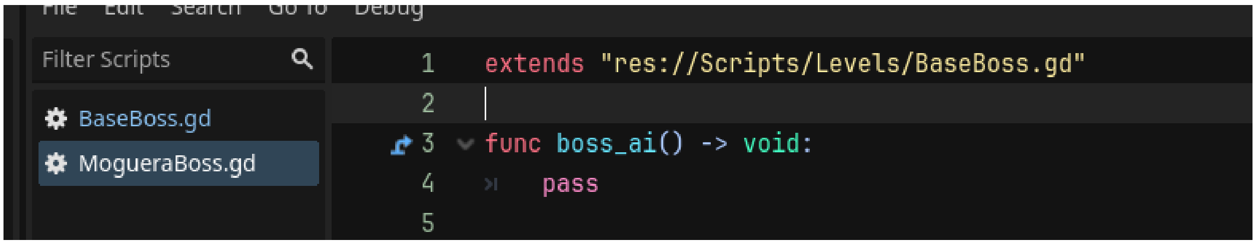
Now you can try to create your own AI. If I were you, I would create a new script that inherits this base boss script:





```
extends "res://Scripts/Levels/BaseBoss.gd"


# Called when the node enters the scene tree for the first time.
func _ready() -> void:
	pass # Replace with function body.


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta: float) -> void:
	pass

```

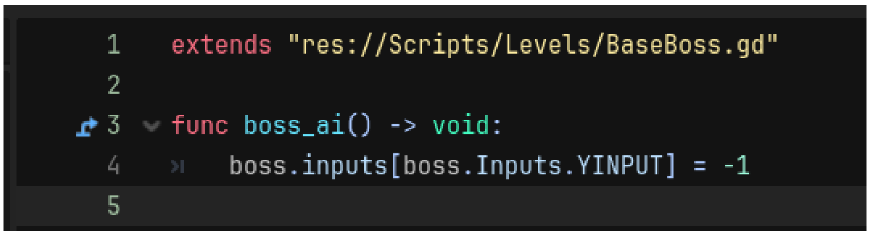
Let’s remove every function and override the boss ai function like this:



```
extends "res://Scripts/Levels/BaseBoss.gd"

func boss_ai() -> void:
	pass

```

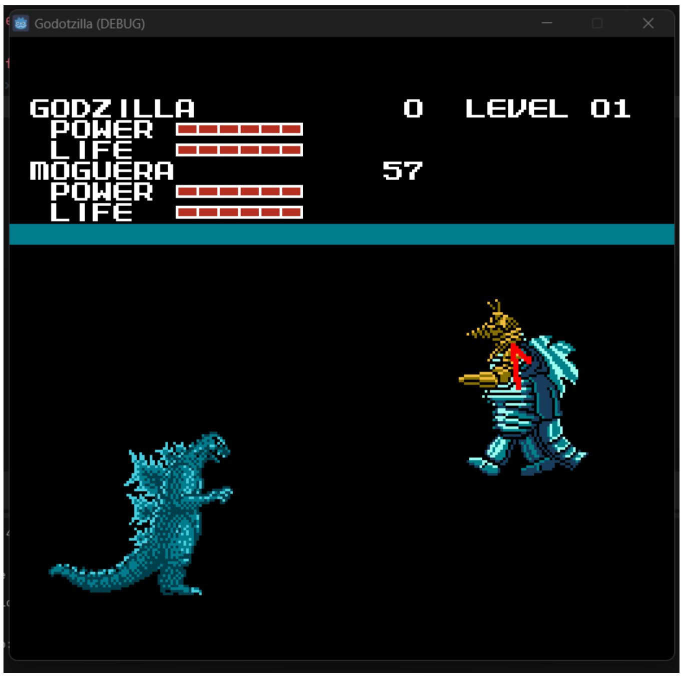
Now you can code your own boss AI!

Check this out:



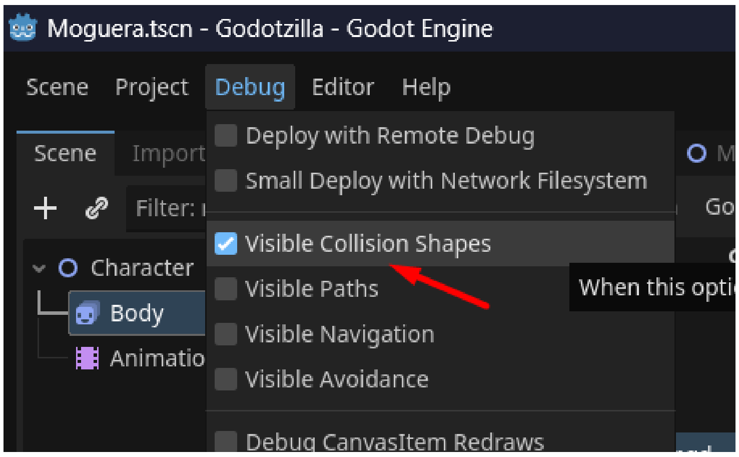
```
extends "res://Scripts/Levels/BaseBoss.gd"

func boss_ai() -> void:
	boss.inputs[boss.Inputs.YINPUT] = -1

```

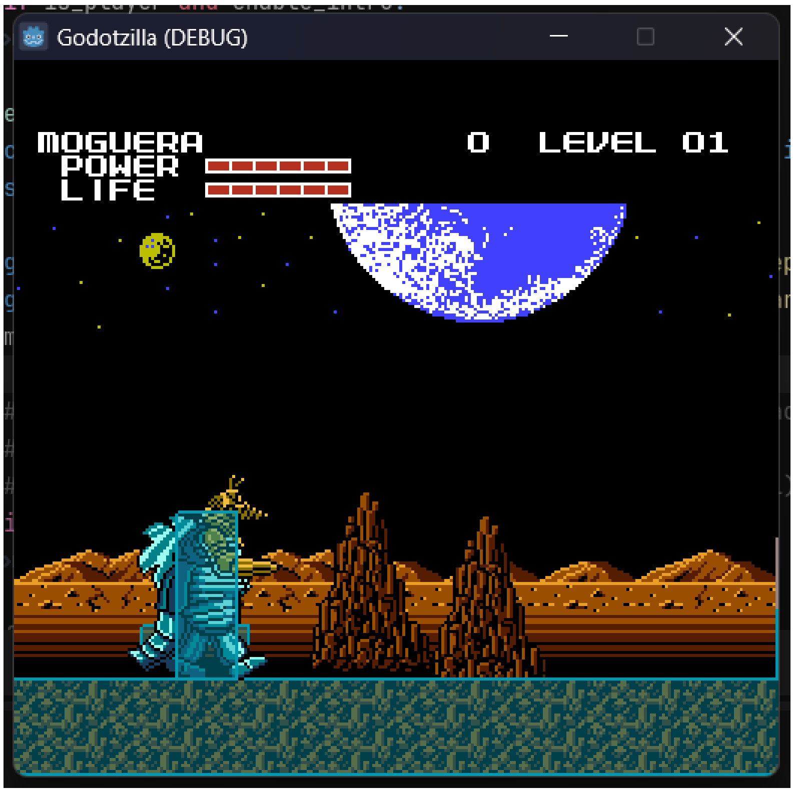
Now Moguera will always be jumping:



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

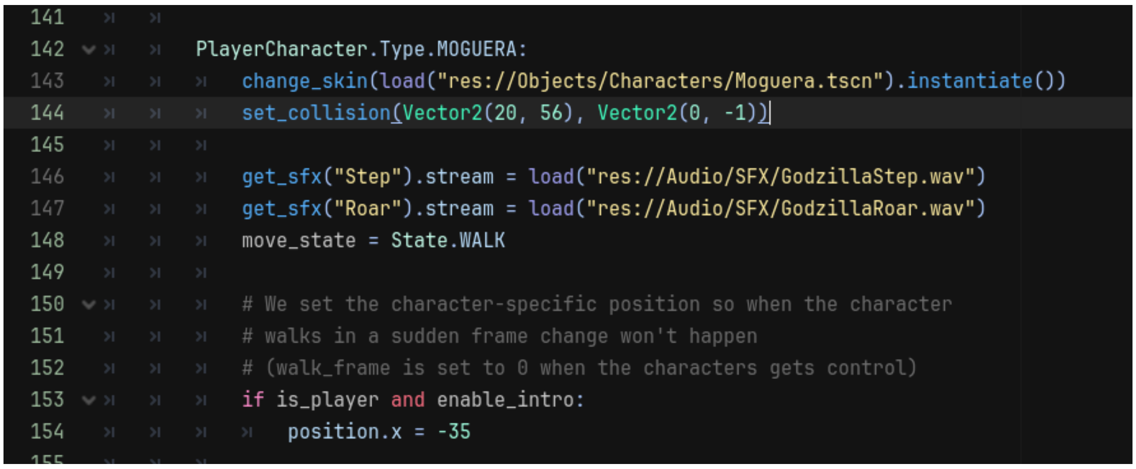
Now it may be subtle, but Moguera is a bit taller than Godzilla, so we may want to change the collision box. But first let’s take a look at our collision box by enabling visible collision shapes:





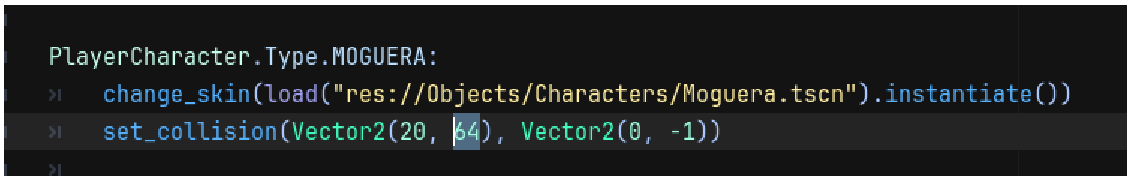
The collision box is indeed a bit too small for Moguera.

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

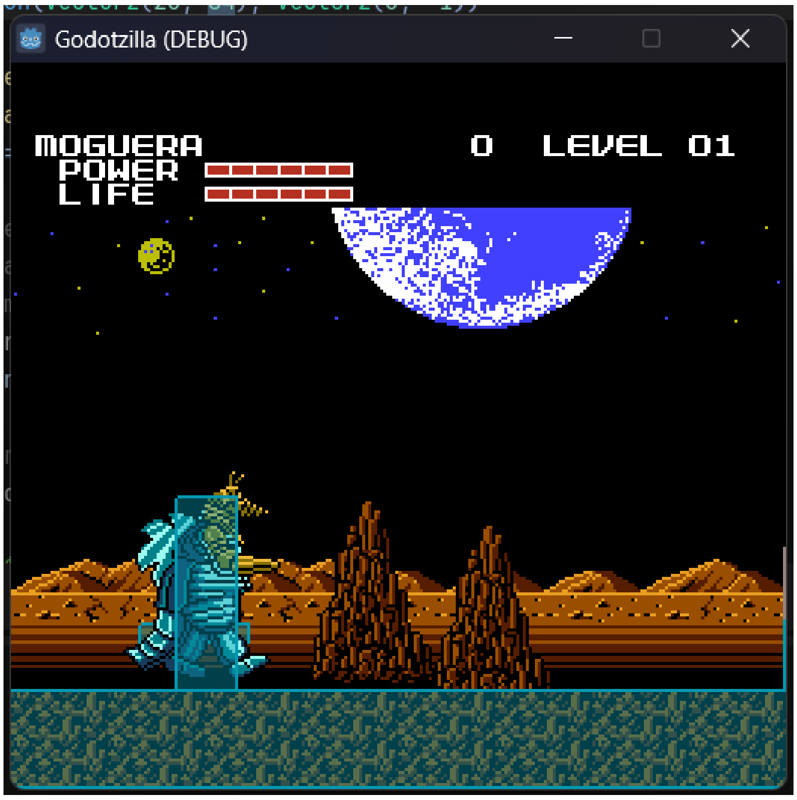


```
		PlayerCharacter.Type.MOGUERA:
			change_skin(load("res://Objects/Characters/Moguera.tscn").instantiate())
			set_collision(Vector2(20, 56), Vector2(0, -1))
			
			get_sfx("Step").stream = load("res://Audio/SFX/GodzillaStep.wav")
			get_sfx("Roar").stream = load("res://Audio/SFX/GodzillaRoar.wav")
			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
```

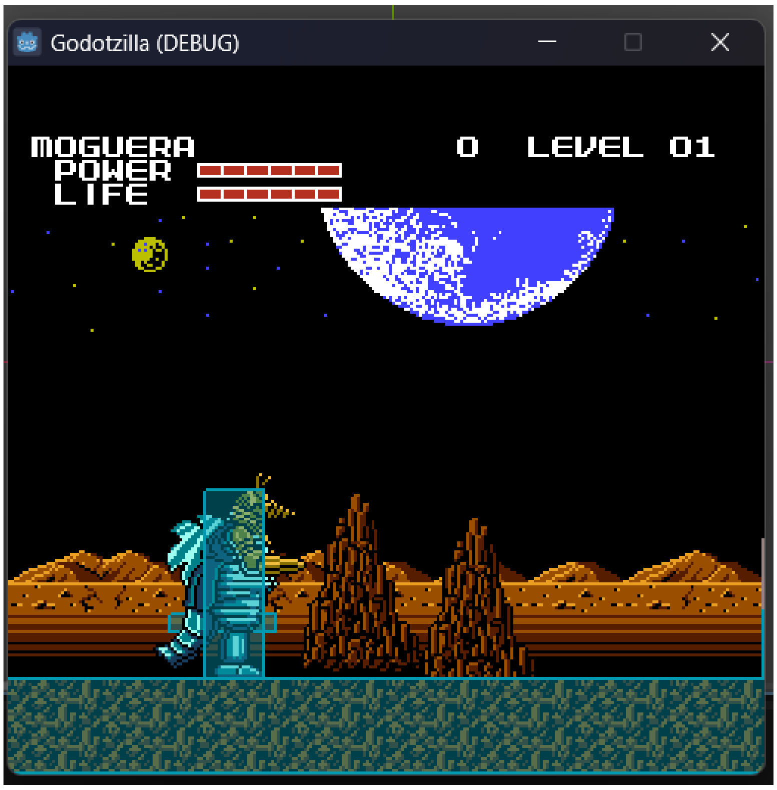
We can see a line that calls the "set_collision" function, it takes the collision box size as the first parameter and offset as the second parameter. Let's change the size to something like Vector2(20, 64):



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



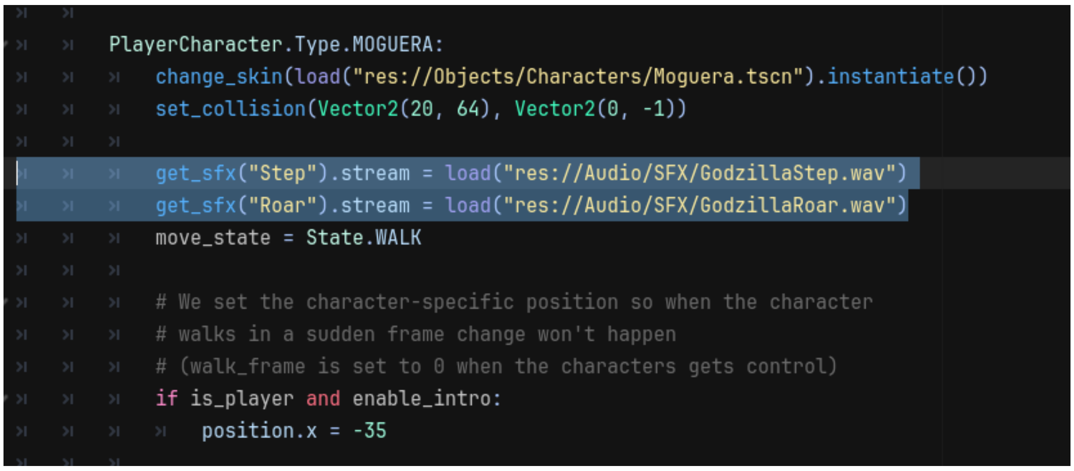
Now the collision box size looks alright, but Moguera’s skin is now floating in the air. We can either offset the collision box or move the sprite in the skin. You can choose either one, but I recommend moving the sprite. So now open the skin scene and move the sprite, like how I showed it before.



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:

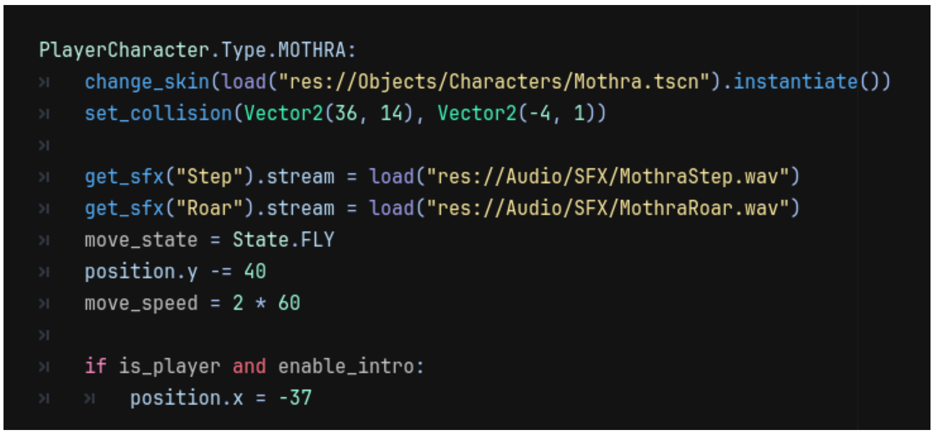


```
		PlayerCharacter.Type.MOGUERA:
			change_skin(load("res://Objects/Characters/Moguera.tscn").instantiate())
			set_collision(Vector2(20, 64), Vector2(0, -1))
			
			get_sfx("Step").stream = load("res://Audio/SFX/GodzillaStep.wav")
			get_sfx("Roar").stream = load("res://Audio/SFX/GodzillaRoar.wav")
			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)

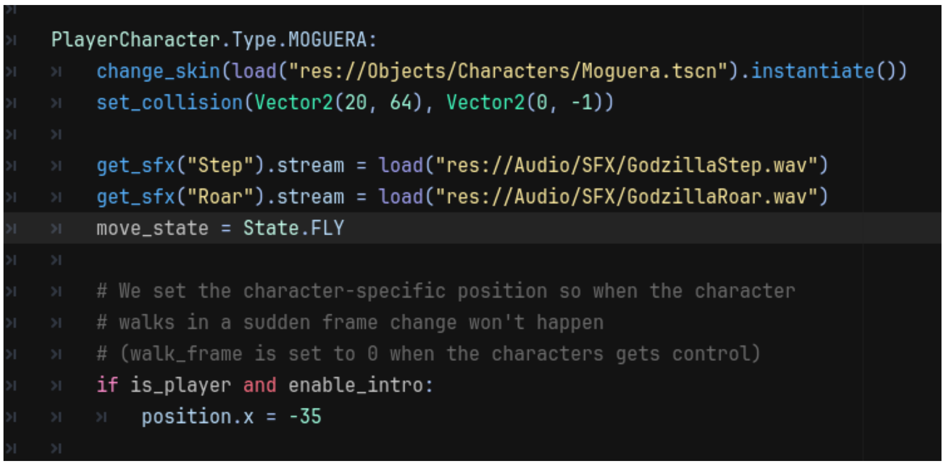
If you want to create a flying character, like Mothra or Rodan, we should do the following:

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



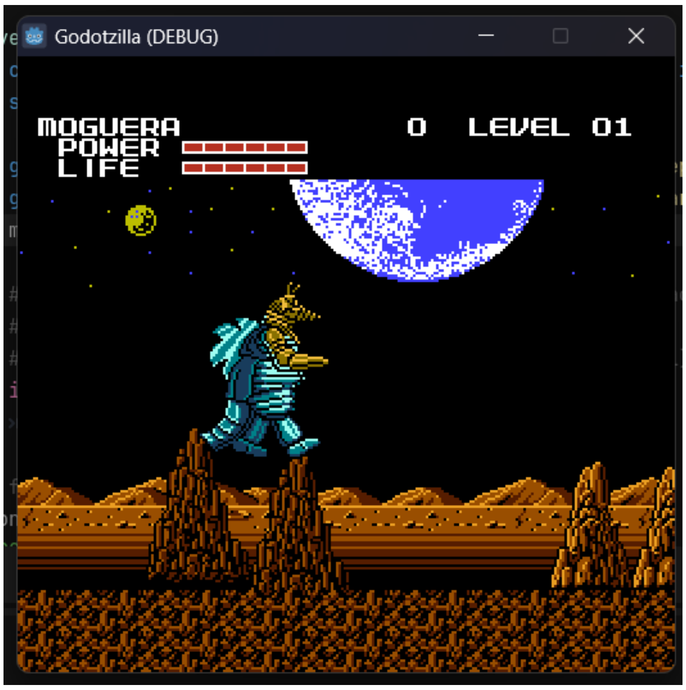
```
PlayerCharacter.Type.MOTHRA:
	change_skin(load("res://Objects/Characters/Mothra.tscn").instantiate())
	set_collision(Vector2(36, 14), Vector2(-4, 1))
	
	get_sfx("Step").stream = load("res://Audio/SFX/MothraStep.wav")
	get_sfx("Roar").stream = load("res://Audio/SFX/MothraRoar.wav")
	move_state = State.FLY
	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:

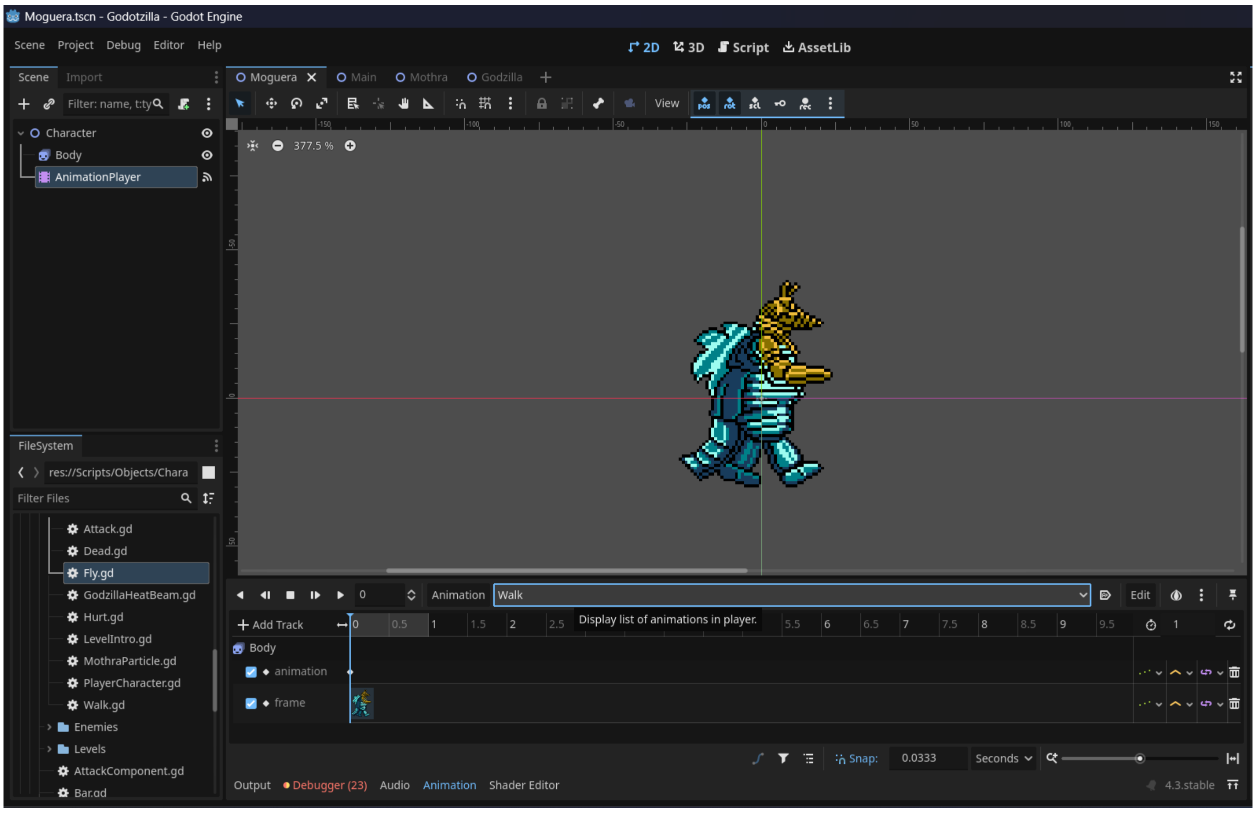


```
PlayerCharacter.Type.MOGUERA:
    change_skin(load("res://Objects/Characters/Moguera.tscn").instantiate())
    set_collision(Vector2(20, 64), Vector2(0, -1))

    get_sfx("Step").stream = load("res://Audio/SFX/GodzillaStep.wav")
    get_sfx("Roar").stream = load("res://Audio/SFX/GodzillaRoar.wav")
    move_state = State.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

```

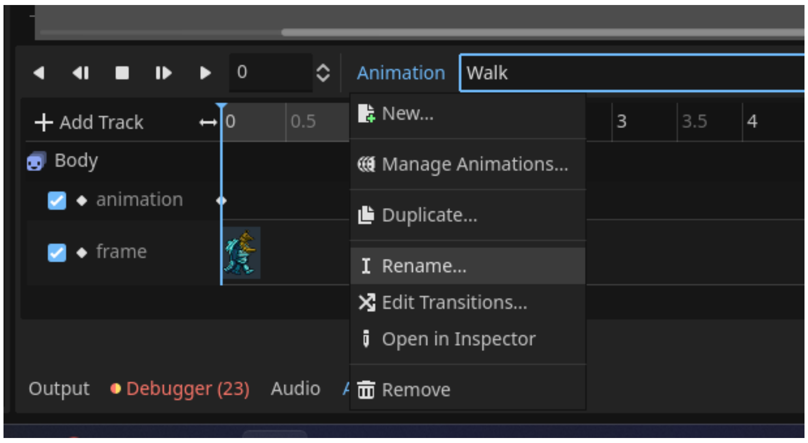
Now let’s test the game.

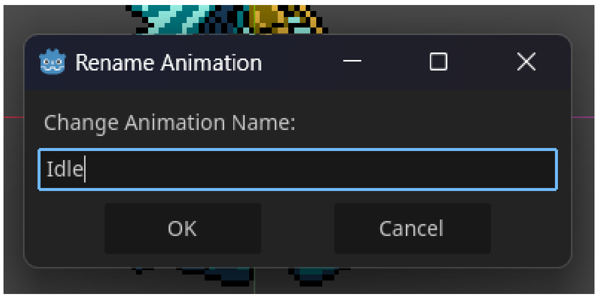


It works! Our new character is flying! But there are new errors that say that animation “Idle” is not found. “Idle” is the equivalent of “Walk” but for flying characters, so let’s rename our walking animation. Open your skin scene and select the walking animation:



Now let’s rename it:



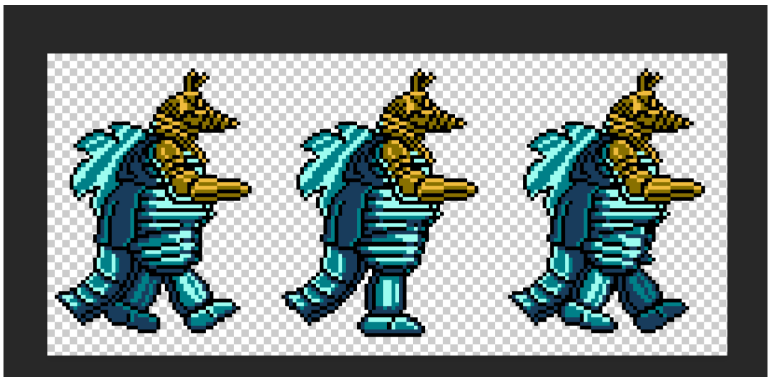


Now let’s try the game again. No errors appear! But you will notice that the animation is stuck again. This time the animation actually works like how it is described in the animation player, not by code. This is because for walking characters the walking frame speed changes depending on the direction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

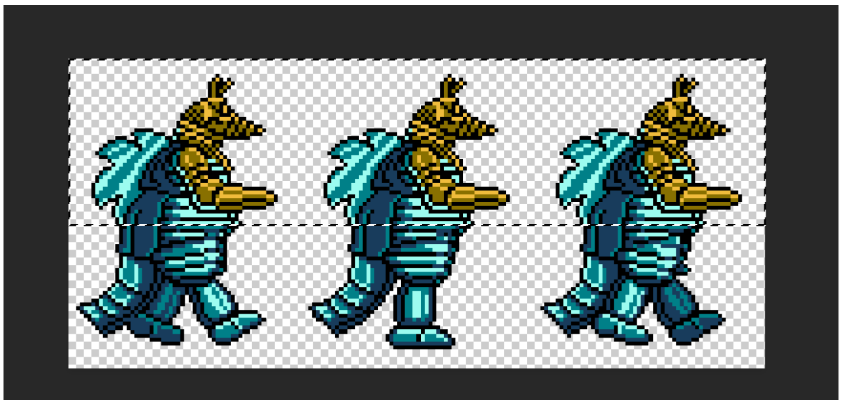
## Separating the top half of the character sprites (optional)

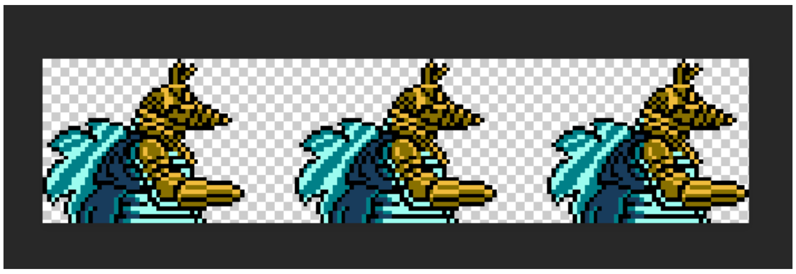
In the "Creating new character skin" chapter I described why you may want to separate the top half of a character's sprites from the bottom half. So now let's implement that for our new character!

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

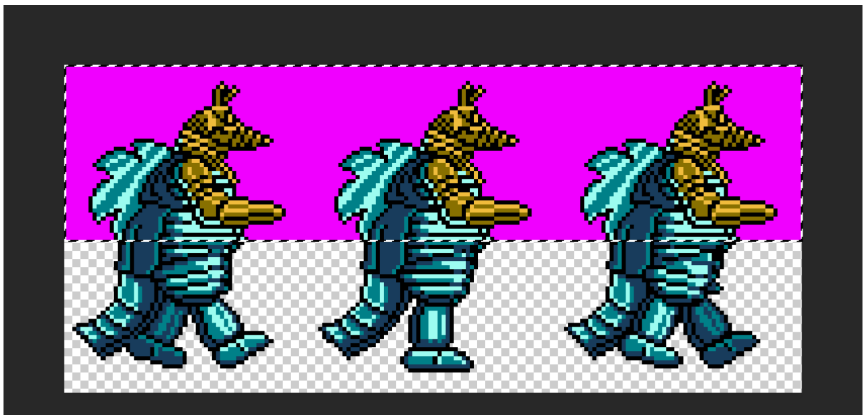


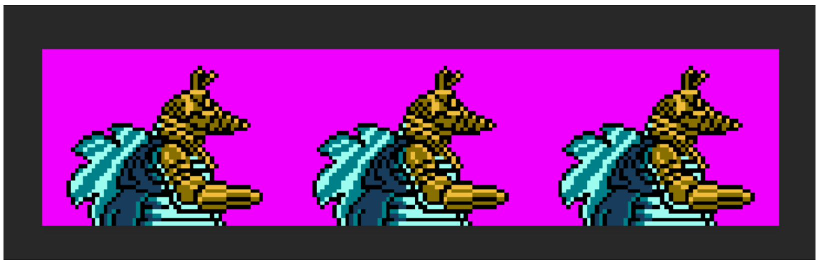
The easiest way to make sprites for the top part is to just copy half of the image and paste it into a new file.



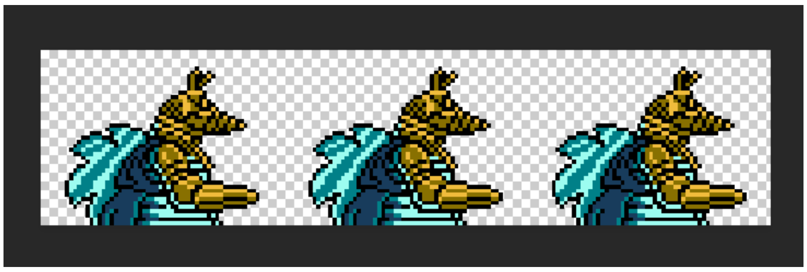


Hm, the new image looks smaller. See, when you copy an image in Photoshop, it ignores the transparent pixels. So you can change the transparent color in your image to something else, like purple, using bucket fill tool.

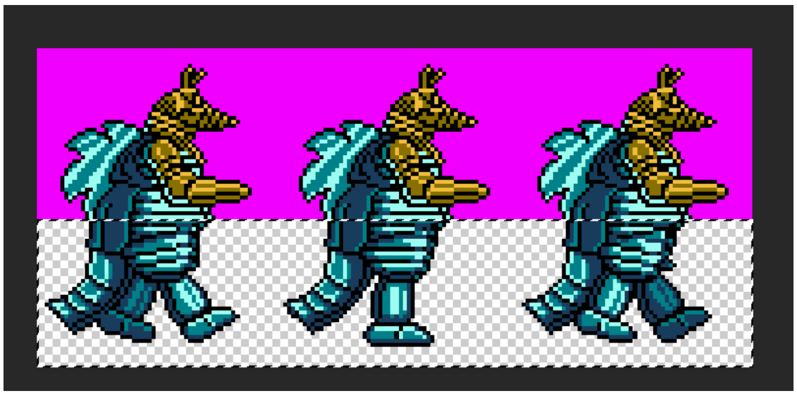


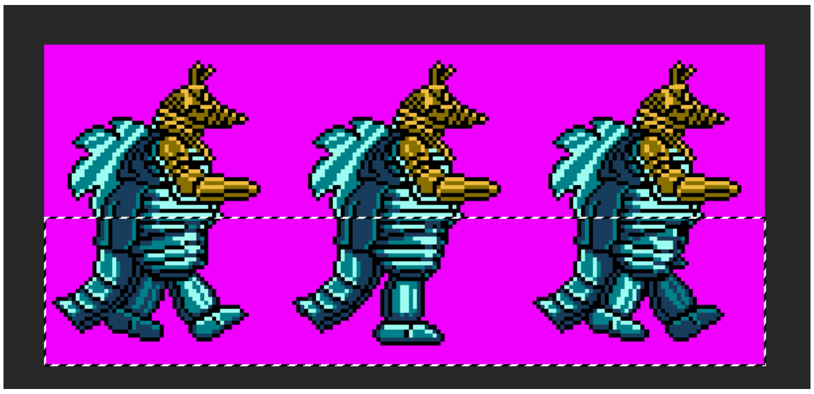


Now we can remove the background color.

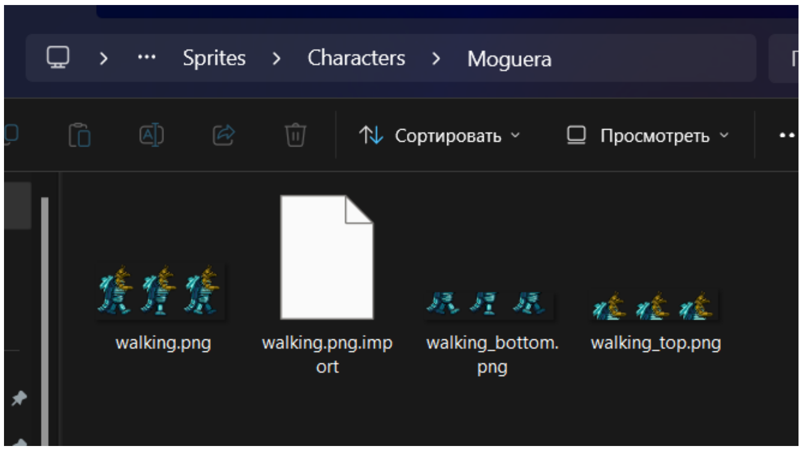


Now we can do the same for the bottom part of the sprites.





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

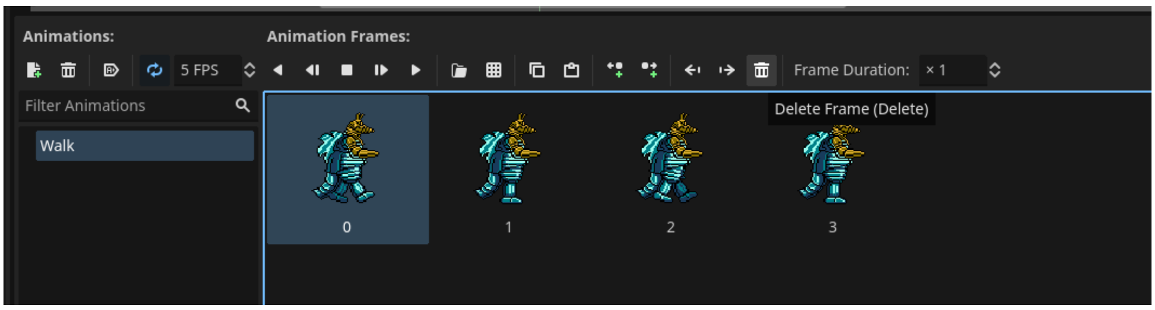


Ignore the .png.import file, it's a special file Godot generated automatically.

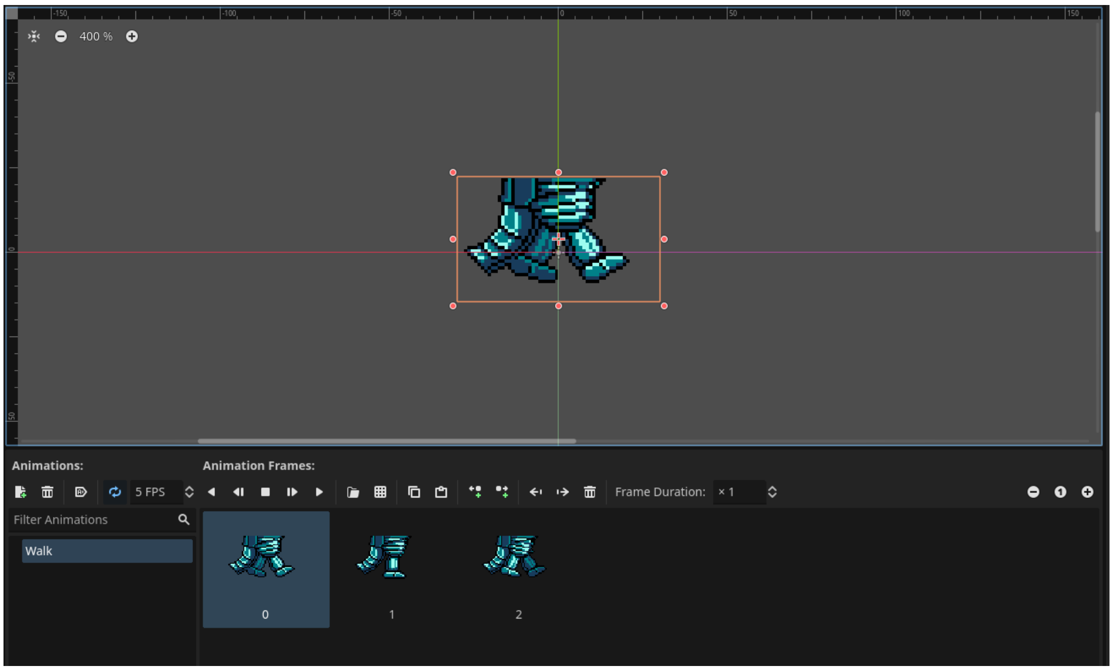
Since Moguera's top half stays the same throughout the animation, I changed the sprites a bit to later indicate what frame is currently shown on screen.

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

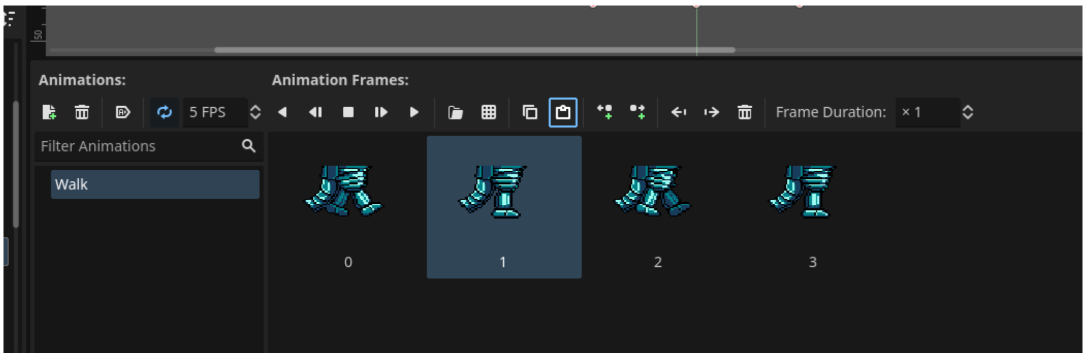
Remember how we imported the sprites in "Importing sprites into Godot" section? We can now do the same, but this time we will load the bottom half sprites into the "Body" node. But first, you will need to delete the previous frames:



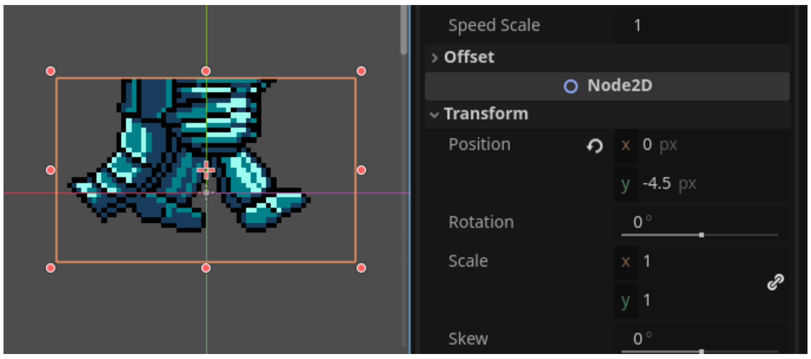
Now let's import the sprites as described in the "Importing sprites into Godot" section.



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

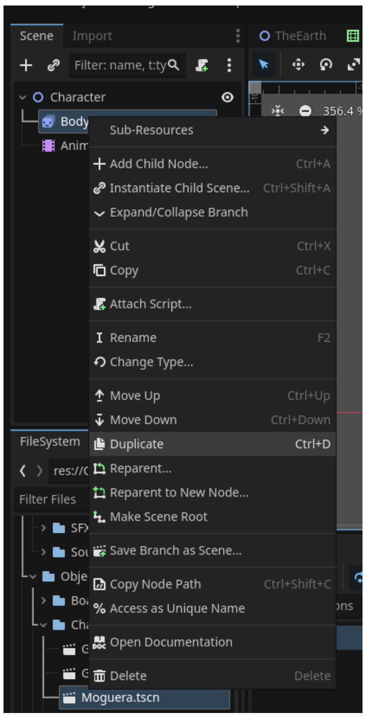


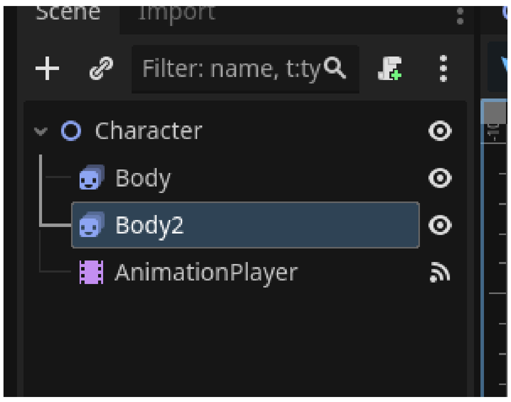
Nice! But in case if the sprites look jittery again if you zoom in/out or move the editor view, the width or height of each sprite became odd if they were even before (or they became even if they were odd before). The reason the sprite becomes jittery is described in the same "Importing sprites into 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.



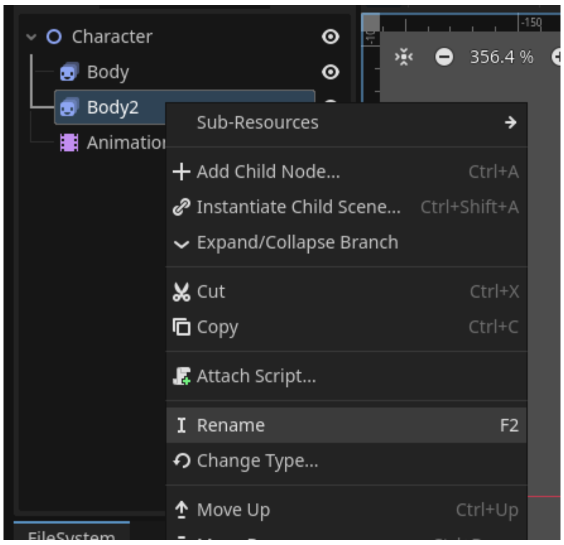
Now let's create a node for the top half. We can create a duplicate of the bottom half node to make the process simpler for this tutorial.

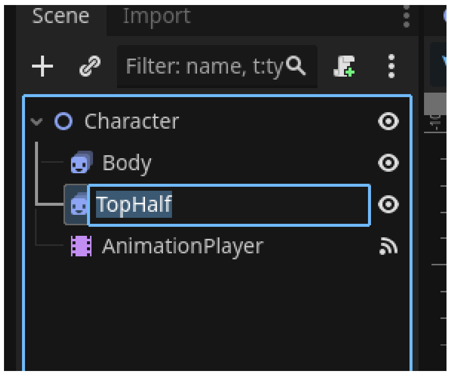
Right click on the "Body" node and select "Duplicate":



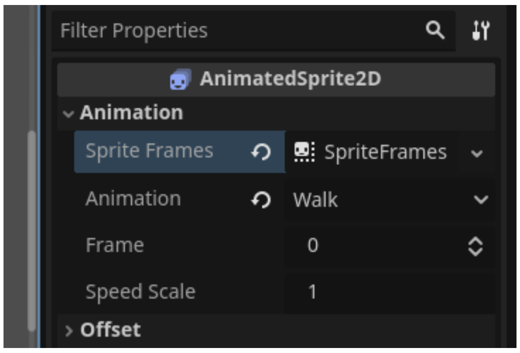


Rename it by right clicking and selecting "Rename":



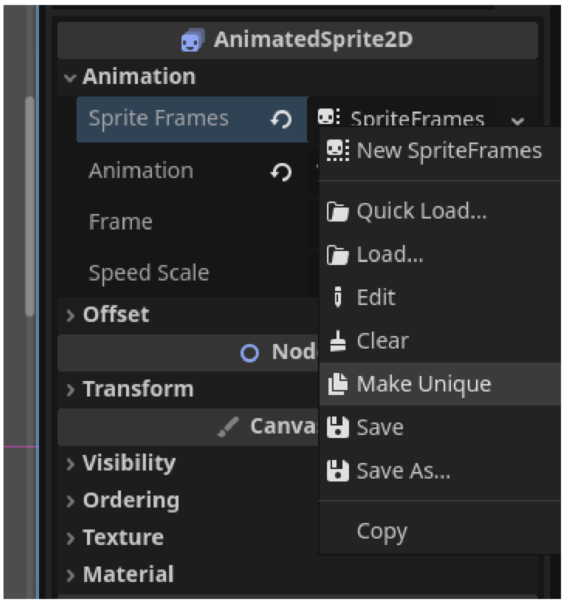


Then press Enter on your keyboard. Nice, now let’s modify it. But before we do that, we need to do one more thing. Select the top half node and look at its properties:

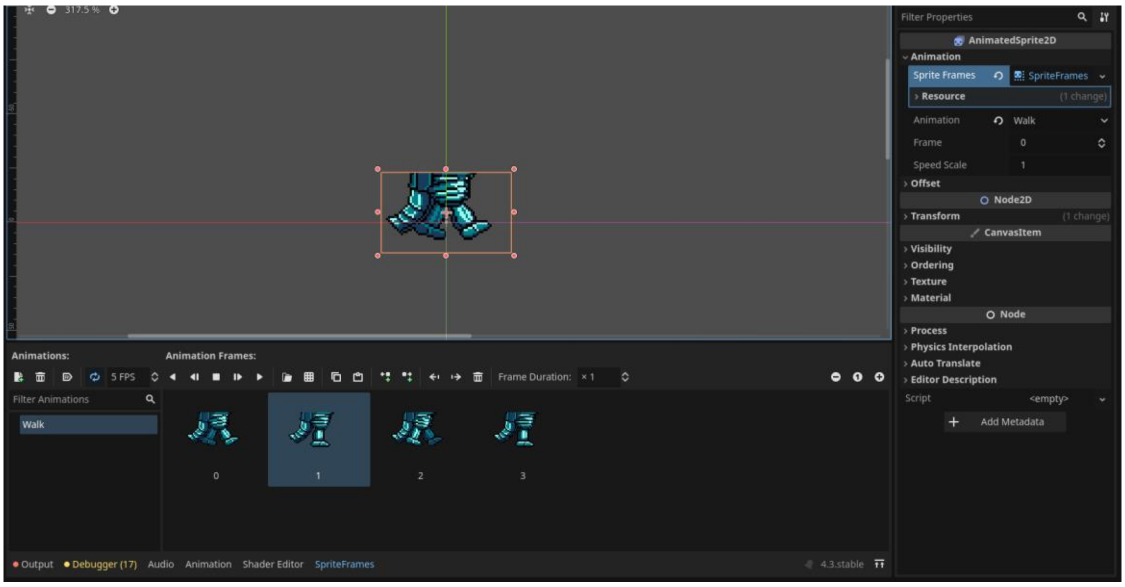


Sprite Frames: SpriteFrames
Animation: Walk
Frame: 0
Speed Scale: 1

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”.



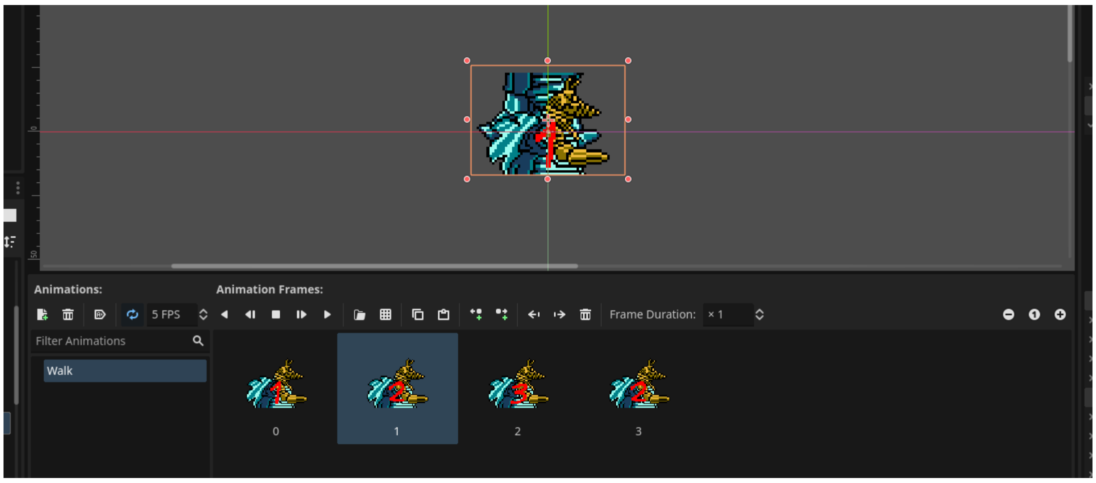
It may look like nothing happened, and after selecting the “SpriteFrames” text again we can see the same sprites of the node.



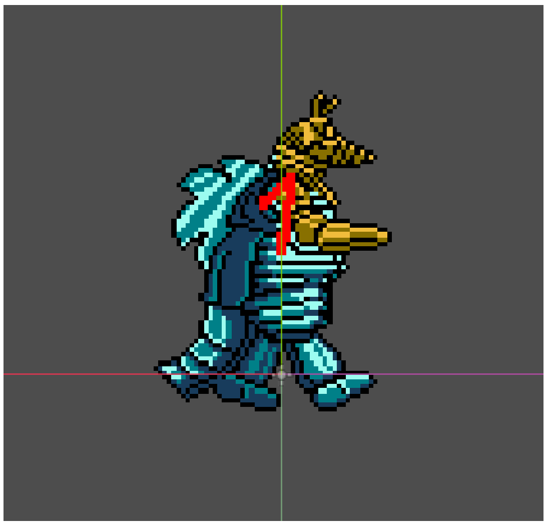
Sprite Frames: SpriteFrames
Animation: Walk
Frame: 0
Speed Scale: 1
Script: <empty>
5 FPS
Frame Duration: × 1

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.

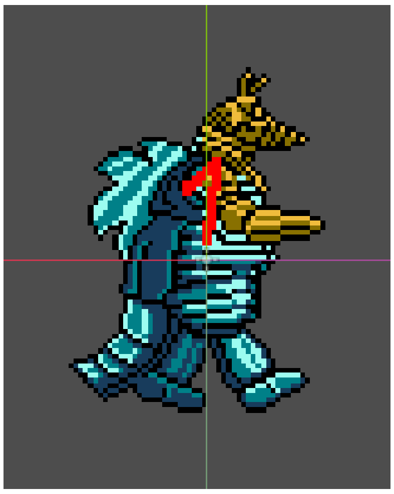
To import the top half sprites, you can use the same technique we used for bottom half.



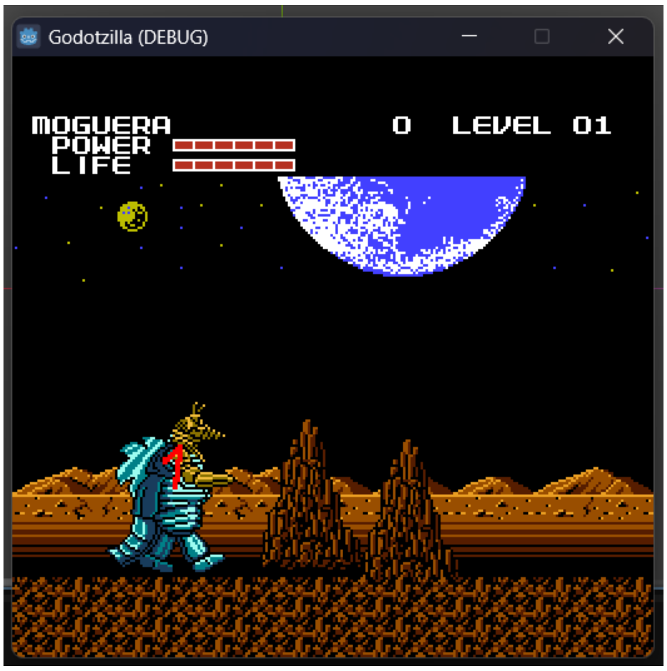
Nice, they're imported. Now let's move that node until it looks alright. And if it looks jittery again, do the same thing as we did above: add half a pixel for the node's position.



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

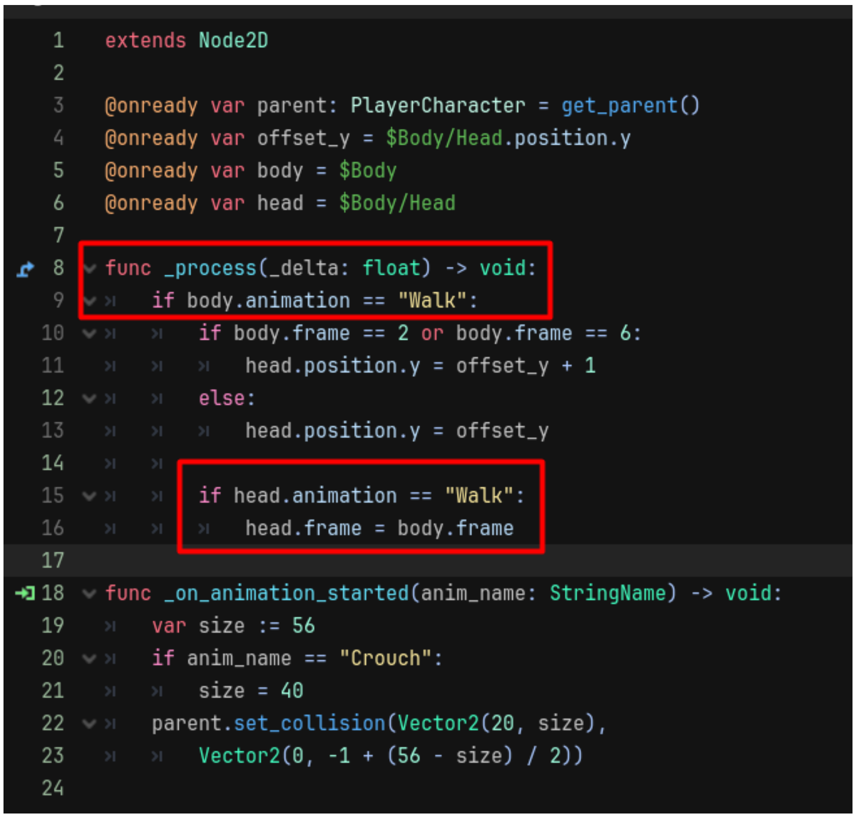


That should do it for now, let's test it in-game! If the character's sprite is a bit in the ground again, you can always move it in the skin scene.



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?

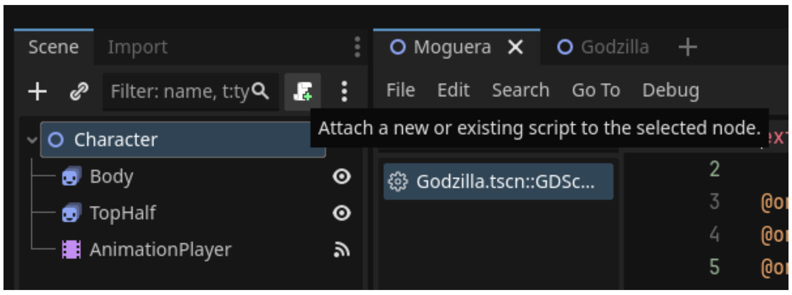
Do you remember how we removed a script from the top character skin node (named "Character") that was a leftover from Godzilla's skin? That script also controls the top part's frame id to be the same as the bottom part when possible (it doesn't do that if the top part's animation isn't set to walking).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



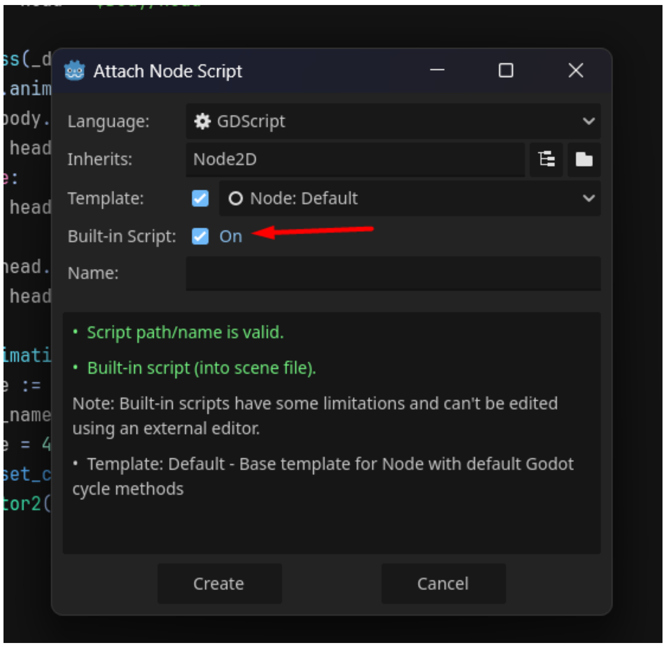
```
extends Node2D

@onready var parent: PlayerCharacter = get_parent()
@onready var offset_y = $Body/Head.position.y
@onready var body = $Body
@onready var head = $Body/Head

func _process(_delta: float) -> void:
	if body.animation == "Walk":
		if body.frame == 2 or body.frame == 6:
			head.position.y = offset_y + 1
		else:
			head.position.y = offset_y

		if head.animation == "Walk":
			head.frame = body.frame

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))

```

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:



Attach a new or existing script to the selected node.



Language: GDScript

Inherits: Node2D

Template: ✓ Node: Default

Built-in Script: ✓ On

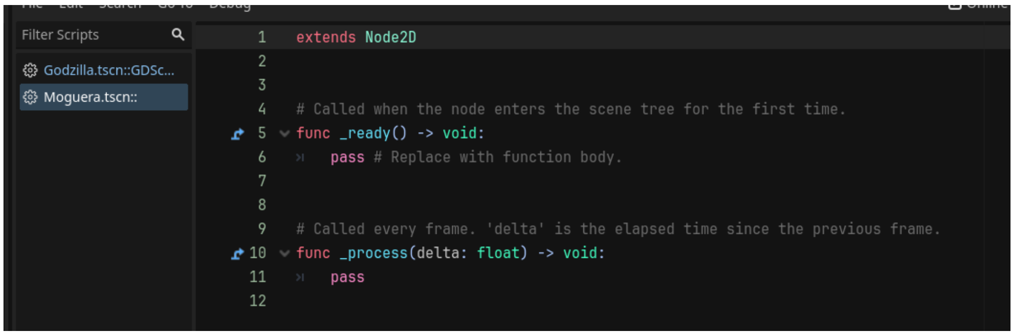
Name:

- Script path/name is valid.
- Built-in script (into scene file).

Note: Built-in scripts have some limitations and can't be edited using an external editor.

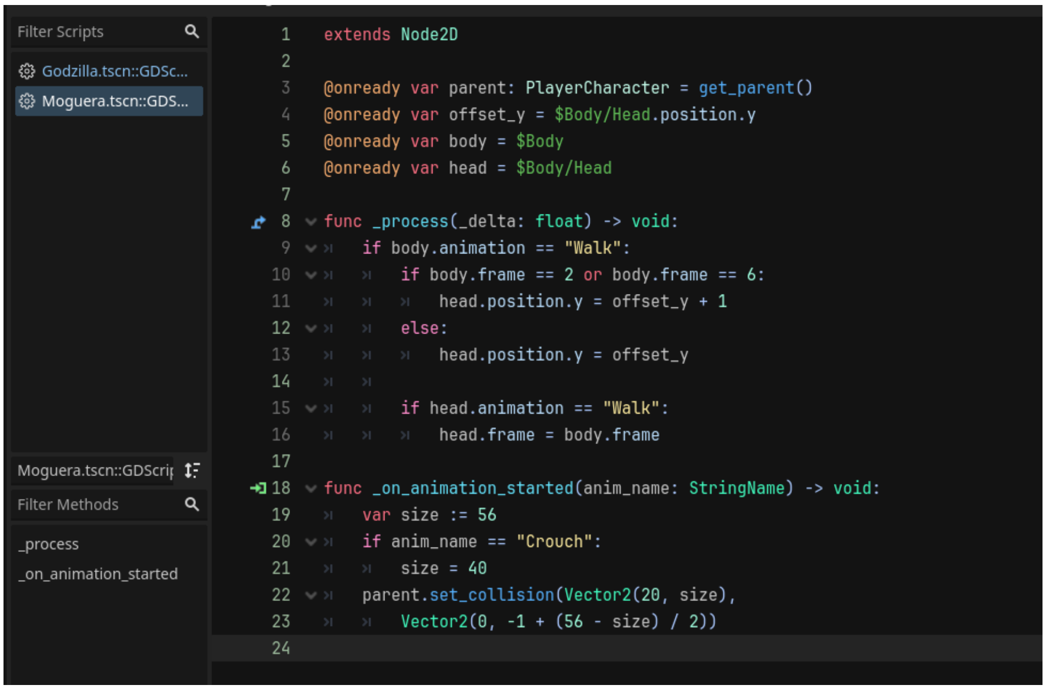
- Template: Default - Base template for Node with default Godot cycle methods

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).



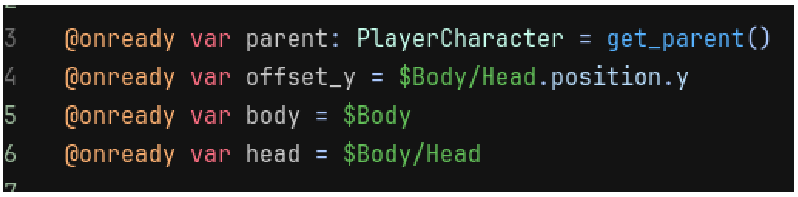
```
extends Node2D


# Called when the node enters the scene tree for the first time.
func _ready() -> void:
	pass # Replace with function body.


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta: float) -> void:
	pass

```

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:



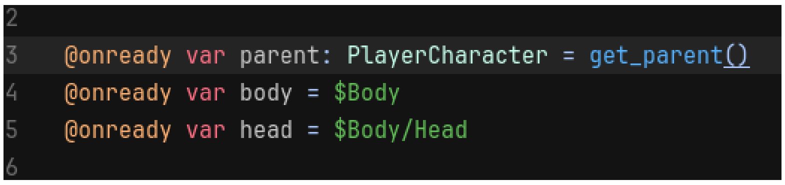
```
extends Node2D

@onready var parent: PlayerCharacter = get_parent()
@onready var offset_y = $Body/Head.position.y
@onready var body = $Body
@onready var head = $Body/Head

func _process(_delta: float) -> void:
	if body.animation == "Walk":
		if body.frame == 2 or body.frame == 6:
			head.position.y = offset_y + 1
		else:
			head.position.y = offset_y
		
		if head.animation == "Walk":
			head.frame = body.frame

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))

```

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



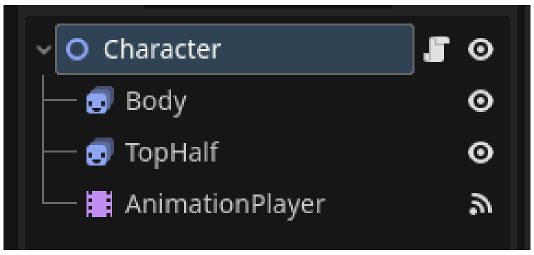
```
@onready var parent: PlayerCharacter = get_parent()
@onready var offset_y = $Body/Head.position.y
@onready var body = $Body
@onready var head = $Body/Head
```

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

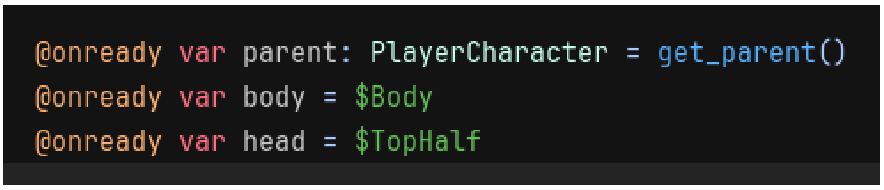


```
@onready var parent: PlayerCharacter = get_parent()
@onready var body = $Body
@onready var head = $Body/Head
```

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

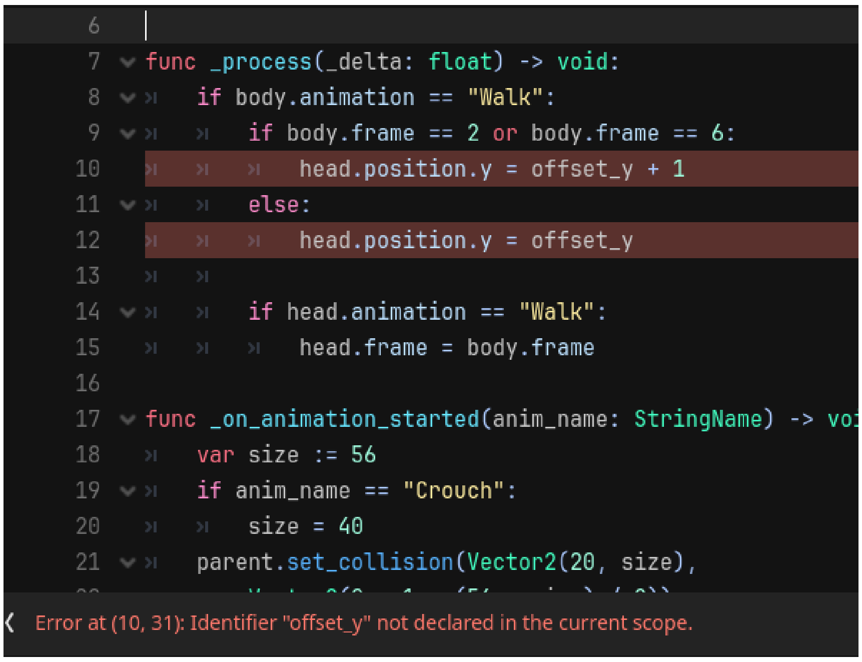


There is a node named "Body" so the body variable will work. And then there's "TopHalf" node, so let's use that name instead of "Body/Head":



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

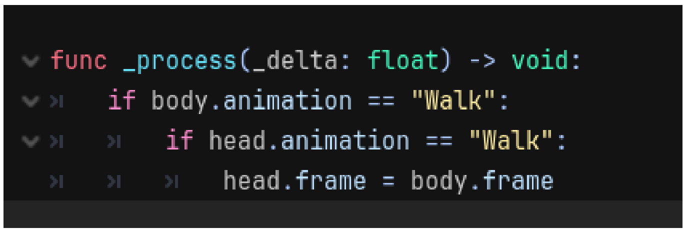
You’ll notice some errors that Godot will show you in the script:



```

func _process(_delta: float) -> void:
	if body.animation == "Walk":
		if body.frame == 2 or body.frame == 6:
			head.position.y = offset_y + 1
		else:
			head.position.y = offset_y

		if head.animation == "Walk":
			head.frame = body.frame

func _on_animation_started(anim_name: StringName) -> voi
	var size := 56
	if anim_name == "Crouch":
		size = 40
	parent.set_collision(Vector2(20, size),
```

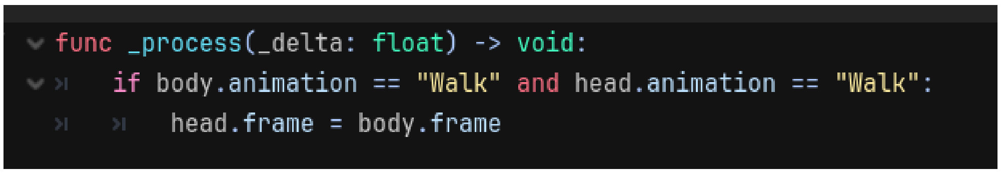
```
Error at (10, 31): Identifier "offset_y" not declared in the current scope.
```

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



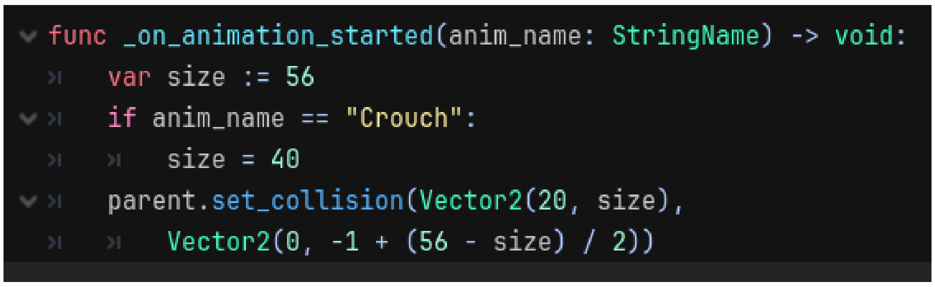
```
func _process(_delta: float) -> void:
	if body.animation == "Walk":
		if head.animation == "Walk":
			head.frame = body.frame
```

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



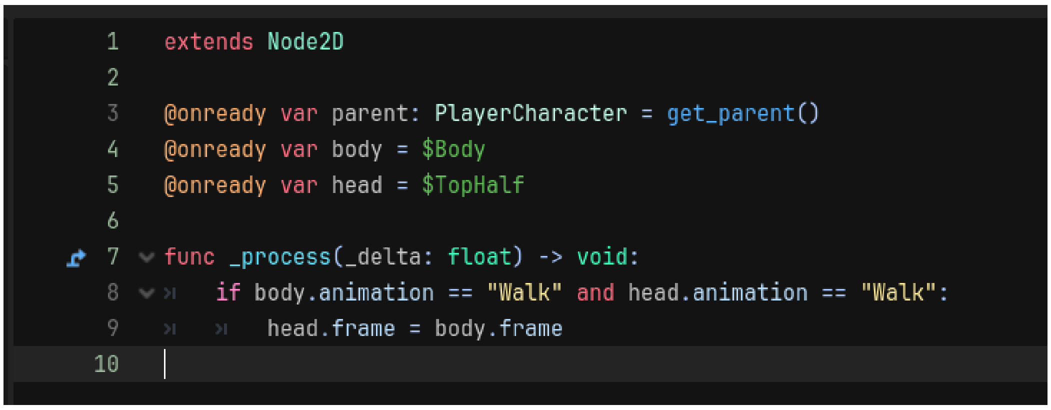
```
func _process(_delta: float) -> void:
	if body.animation == "Walk" and head.animation == "Walk":
		head.frame = body.frame
```

There’s also this leftover piece of code:

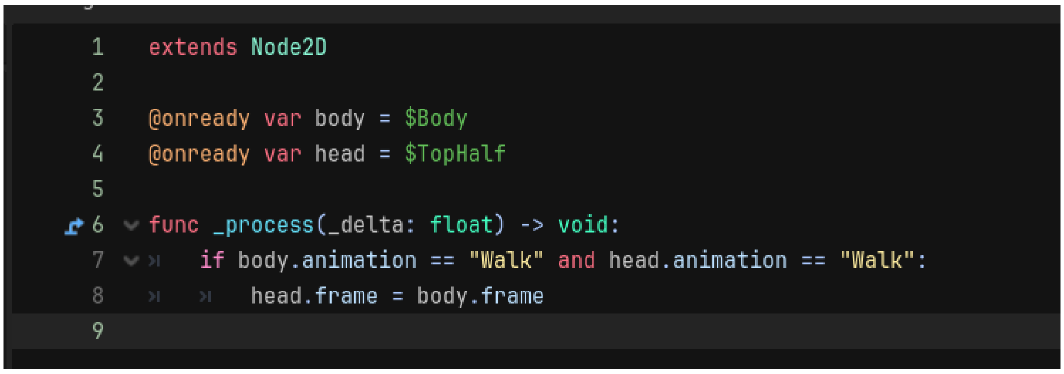


```
func _on_animation_started(anim_name: StringName) -> void:
	var size := 56
	if anim_name == "Crouch":
		size = 40
	parent.set_collision(Vector2(20, size),
		Vector2(0, -1 + (56 - size) / 2))
```

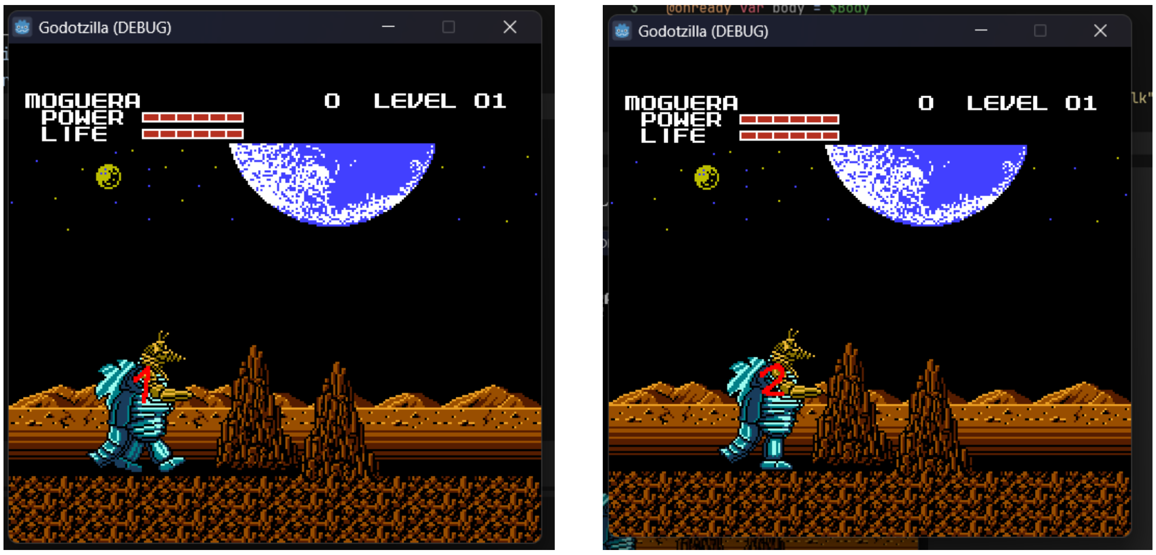
Our character doesn't have a crouch animation, so you can remove this whole function.


```
extends Node2D

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

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

```

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


```
extends Node2D

@onready var body = $Body
@onready var head = $TopHalf

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

```

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



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