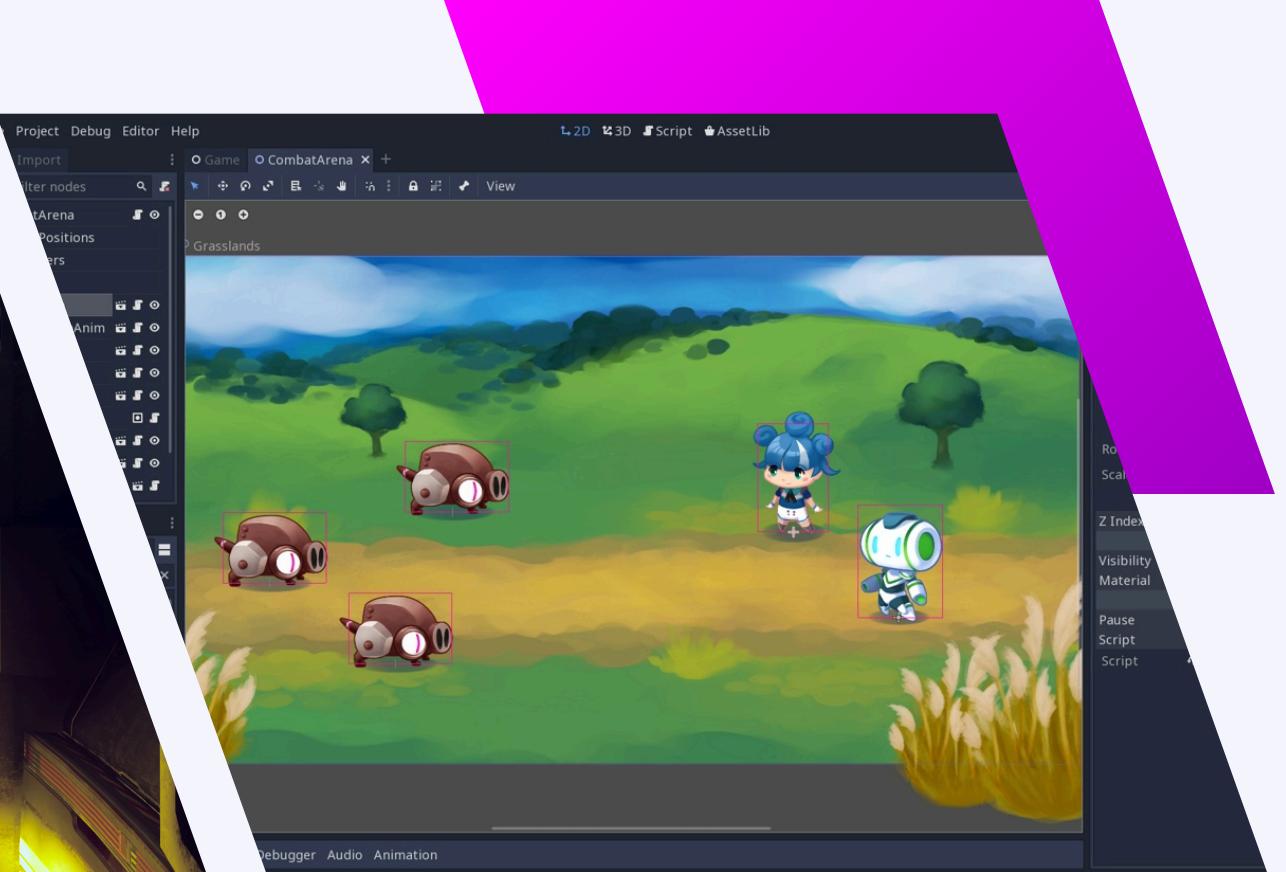
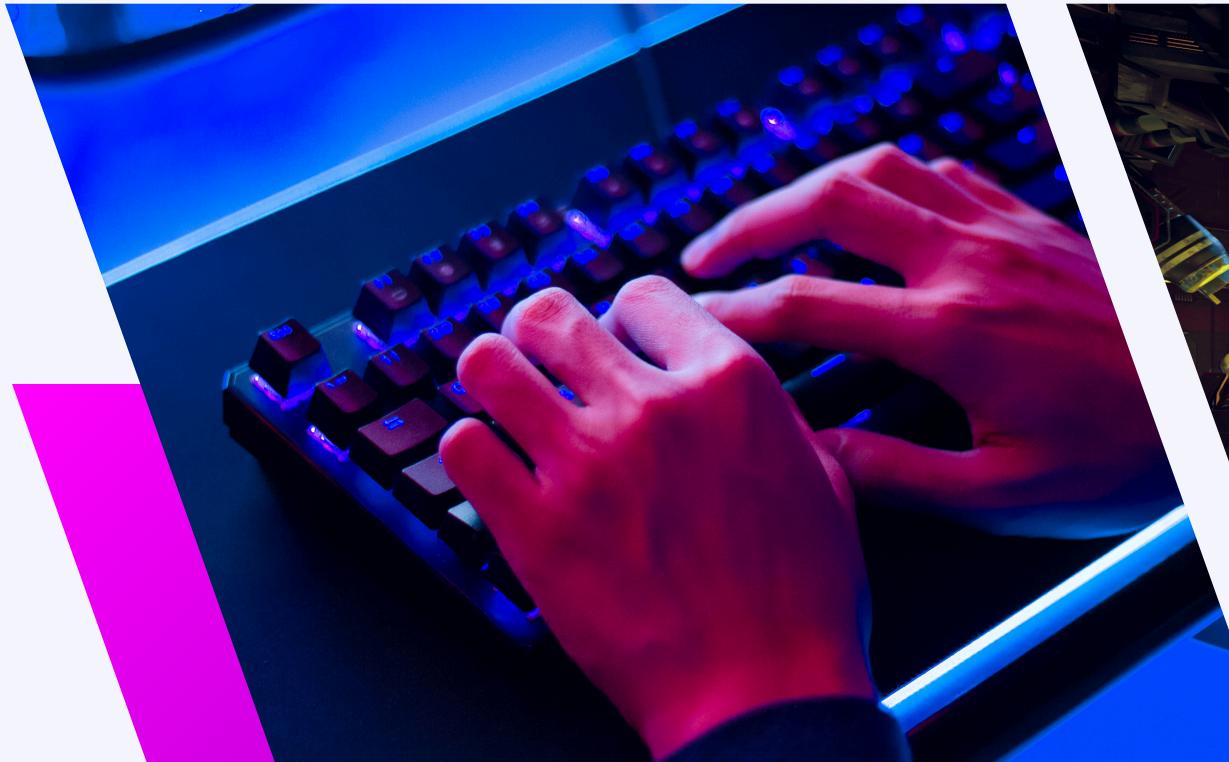


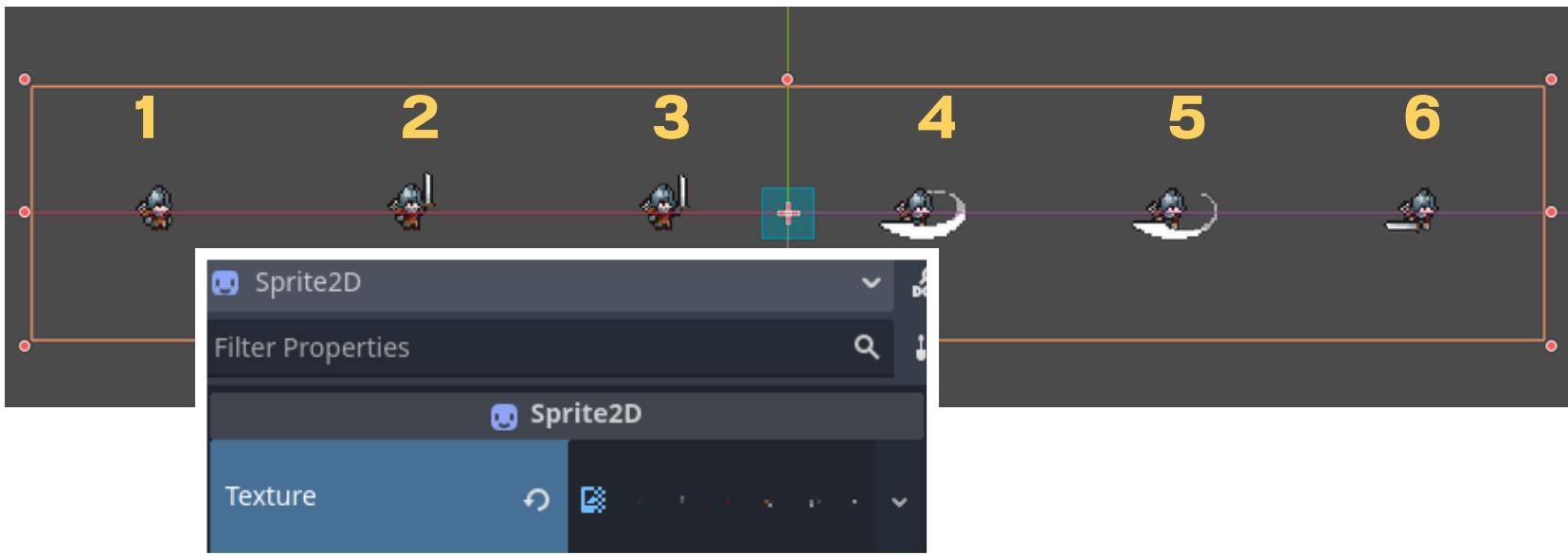
GAME DEVELOPER

College of Creative Design and Entertainment Technology

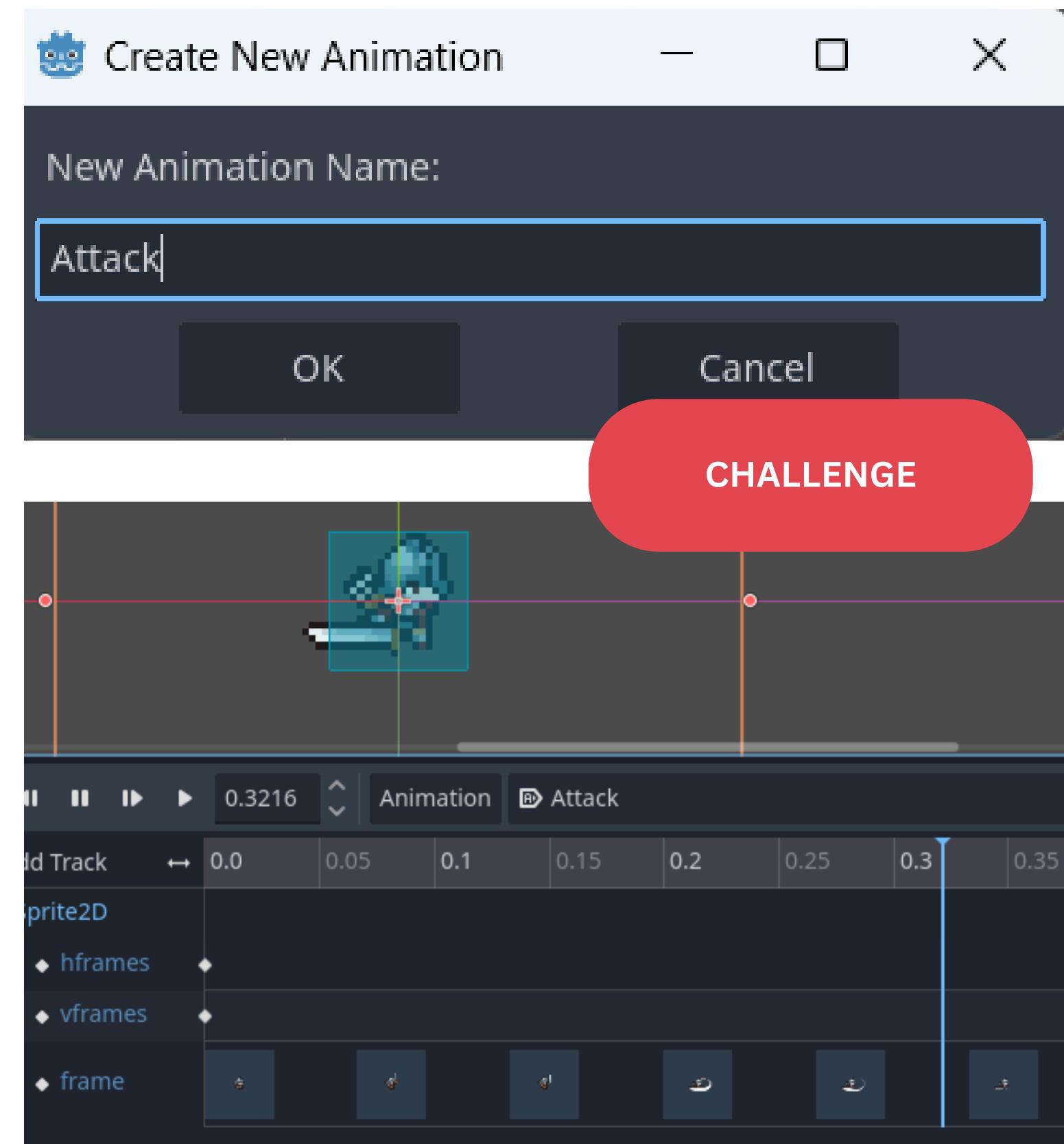
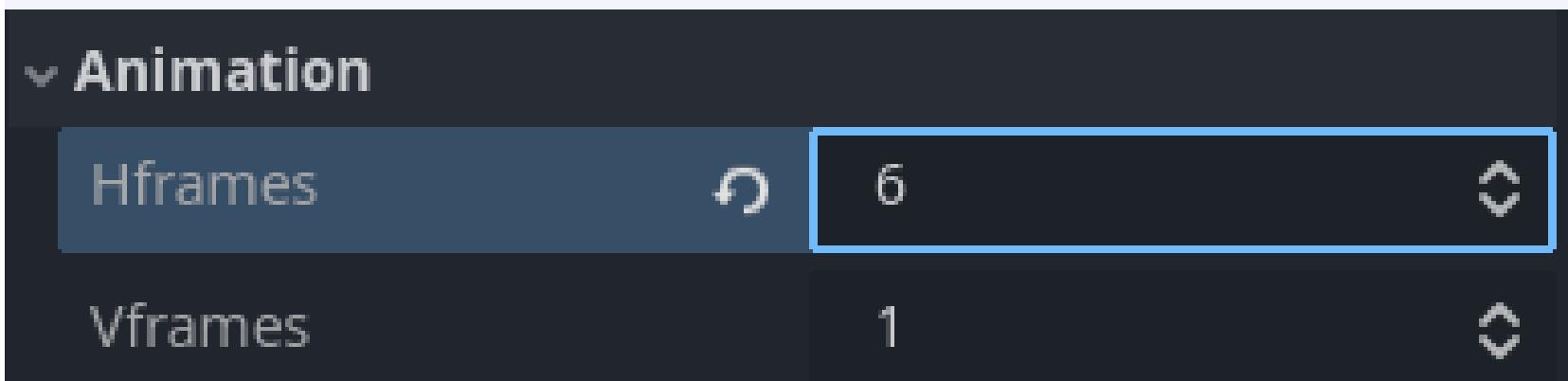


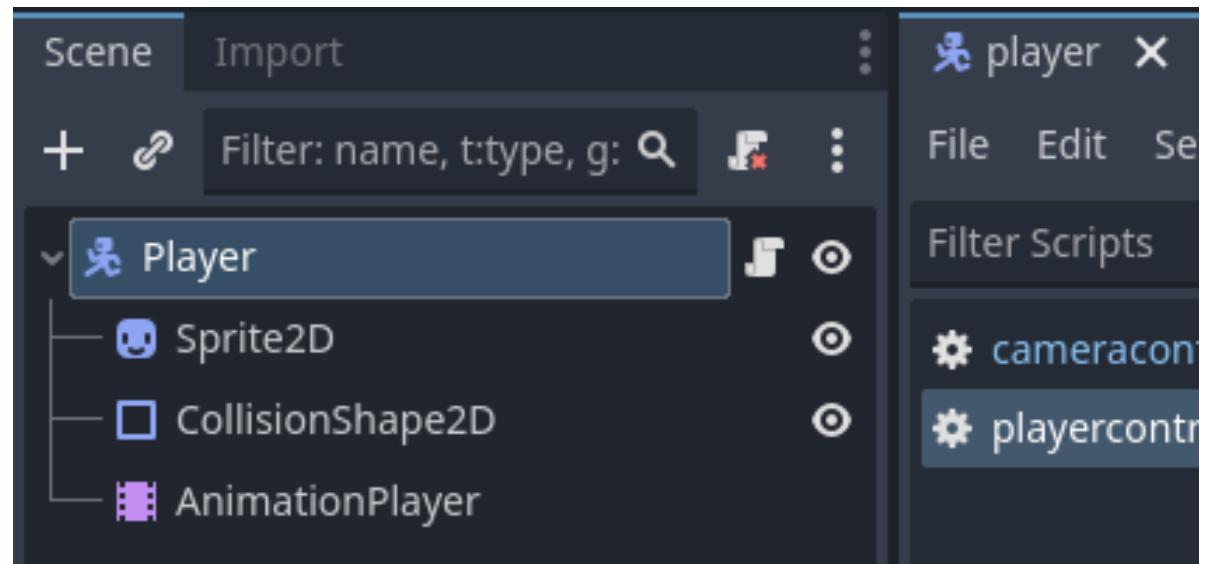
การพัฒนาเกมด้วยโปรแกรม Godot Engine

ແອນັມເບັນໄຈມຕີຂອງຜູ້ເລີນ ໃໃບກເຮີຍບກ່ອນຫຼາ ແລະຄວາມເຫັງໃຈ
ກາຮສ້າງແອນັມເບັນ Sprite2D ແລະ AnimationPlayer ໃໝ່

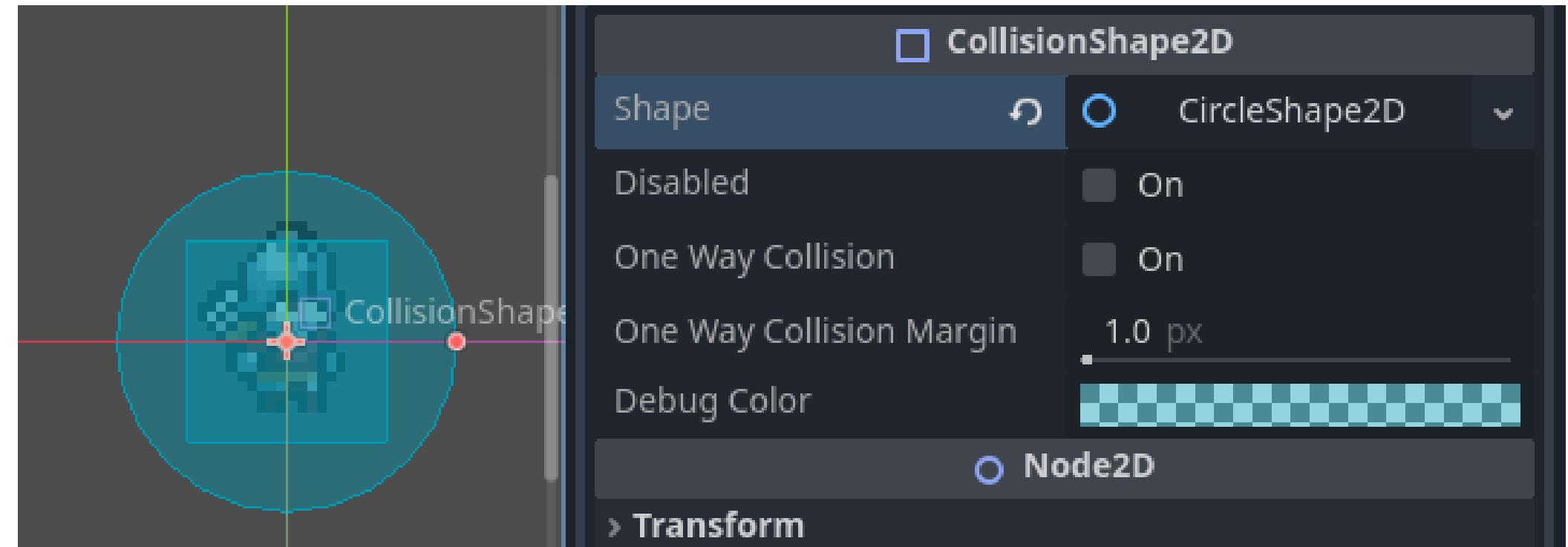
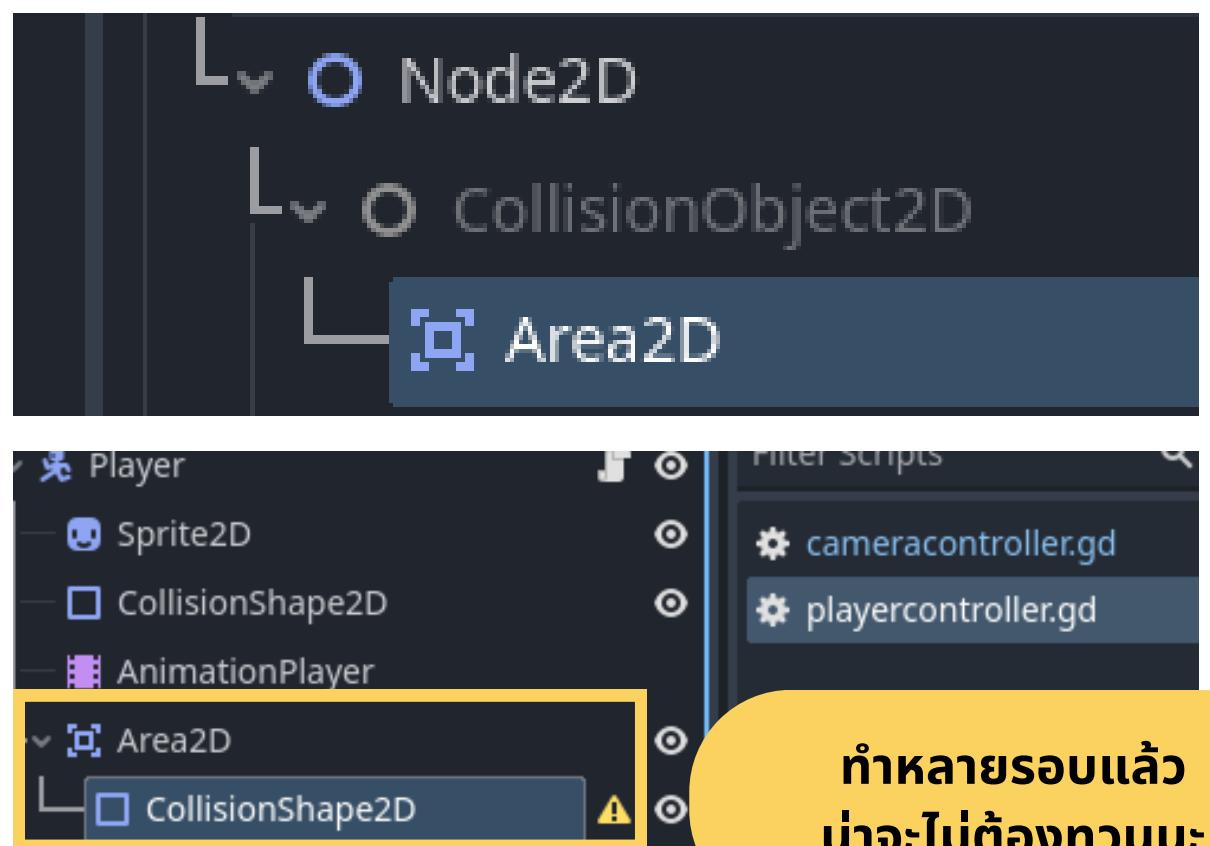


ຂັບຕອນທີ 1 ສ້າງແອນັມເບັນໃໝ່ສໍາຮັບກາຮໄຈມຕີ
ເອົາໄວົງ Soldier-Attack01.png ໂປ່າກປ່າຍກ່ອນ ສາມາດສ້າງ Animation ຊື່ວ່າ “Attack”



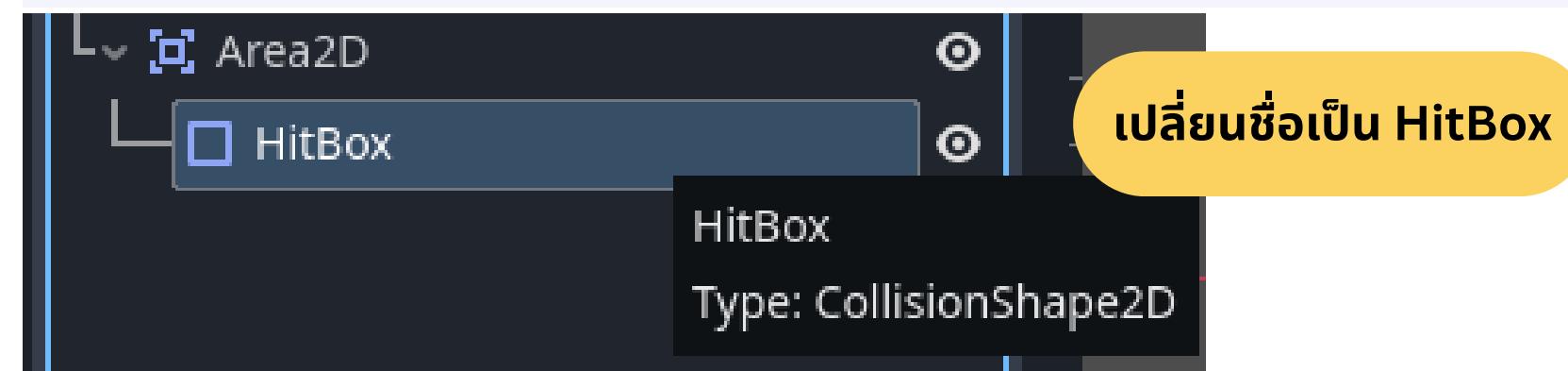


คลิกกี่ไฟล์ **player.tscn** เราจะทำการเพิ่ม **Area2D** และ **Hitbox** ที่เป็น **CollisionShape** เอาไว้โจรตีคัตธุ คลิกขวาที่ Player



ขั้นตอนที่ 2 ปรับ HitBox ของ CollisionShape

เพิ่ม Shape ของ **CollisionShape2D** ตัวที่สองของ Area2D เป็น CircleShape2D หลังจากนั้น Rename เป็น **HitBox** และปรับ Area ให้ครอบคลุม



เมื่อกด **P** หรือคลิกซ้าย → ตัวละครจะเล่นแอบิเมชัน **Attack** และ **Hitbox** จะเปิดระหว่างการโจมตี และปิดเมื่อแอบิเมชันจบ ใช้สำหรับตรวจจับคัตธุที่อยู่ในระยะโจมตี

ขั้นตอนที่ 3 เขียนโปรแกรมควบคุม HitBox บนโจนตี

เปิดไฟล์ playercontroller.gd ขึ้นมาทำการประกาศตัวแปร 2 ตัวคือ animation_player และ hitbox มา กำกับดังนี้:

```
@onready var animation_player: AnimationPlayer = $AnimationPlayer  
@onready var hitbox: CollisionShape2D = $Area2D/Hitbox
```

1. ประกาศตัวแปรชื่อ **animation_player** ซึ่งมีชนิดข้อมูลเป็น AnimationPlayer
2. ใช้ **@onready** เพื่อให้ค่าถูกกำหนด หลังจาก Node ถูกโหลด (ready) แล้ว
3. **\$AnimationPlayer** คือการเข้าถึง Node ซึ่ง AnimationPlayer ที่อยู่ภายใน Node หลัก

```
1  extends CharacterBody2D  
2  @onready var animation_player: AnimationPlayer = $AnimationPlayer  
3  @onready var hitbox: CollisionShape2D = $Area2D/Hitbox
```

เขียนเงื่อนไขใน **func _physic_process()**: ด้านล่างสุดของฟังก์ชัน สำหรับตรวจสอบการกดแป้นพิมพ์ “P” เพื่อเรียกฟังก์ชันว่า **attack()**

```
# ตรวจสอบการโจมตีด้วยปุ่ม P หรือคลิกซ้าย  
if Input.is_action_just_pressed("attack"):  
    attack()
```

```
31  > # เลือกนโยบายกับลูกบัน  
32  > if direction == Vector2.ZERO:  
33  >     $AnimationPlayer.play("Idle")  
34  > else:  
35  >     $AnimationPlayer.play("Walk")  
36  >  
37  > # ตรวจสอบการโจมตีด้วยปุ่ม P หรือคลิกซ้าย  
38  > if Input.is_action_just_pressed("attack"):  
39  >     attack()  
40
```

⟨ Error at (39, 9): Function "attack()" not found in base self.

ระบบจะบอกว่าค้นหาฟังก์ชัน **attack()**; ไม่พบ เพราะว่าเรายังไม่ได้สร้าง (Implements) ขึ้นมา

ขั้นตอนที่ 4 การสร้างฟังก์ชันใหม่

ในการเขียนโปรแกรมเรารสามารถสร้างฟังก์ชันขึ้นมาเองได้ **ฟังก์ชัน** คือ กลุ่มคำสั่งที่รวมกันเป็นหน่วยเดียว เพื่อกำหนดบางอย่าง ช่วยให้โค้ด อ่านง่าย, เรียกใช้ได้, และ จัดการโปรแกรมได้ดีขึ้น

วิธีเขียนฟังก์ชันใน GDScript

```
func ชื่อฟังก์ชัน(พารามิเตอร์):  
    # คำสั่งในฟังก์ชัน  
    return ค่าที่จะส่งกลับ (ถ้ามี)
```

ตัวอย่าง 1: ฟังก์ชันไม่รับค่า ไม่คืนค่า

```
func say_hello():  
    print("สวัสดีชาวโลก!")
```

เรียกใช้:

```
say_hello()
```

ตัวอย่าง 2: ฟังก์ชันรับค่า และคืนค่า

```
func add(a, b):  
    return a + b
```

เรียกใช้:

```
var result = add(5, 10) # result = 15
```

ตัวอย่าง 3: ใช้ควบคุมพฤติกรรมในเกม ซึ่งเราจะ Implement กัน

```
func attack():  
    animation_player.play("Attack")  
    hitbox.disabled = false
```

```
# ปิด hitbox อัตโนมัติหลังจากเล่นแอนิเมชันเสร็จ (หรือตั้งเวลา)  
await animation_player.animation_finished  
hitbox.disabled = true
```

ฟังก์ชันใน GDScript คือคำสั่งที่รวมกันเพื่อกำหนดเฉพาะอย่าง เช่น เดิน โjump เก็บเหรียญ เป็นต้น โดยเราสามารถเรียกใช้ฟังก์ชันเหล่านี้ได้โดยตรง เช่น `player.jump()` หรือ `enemy.collect_coin()`

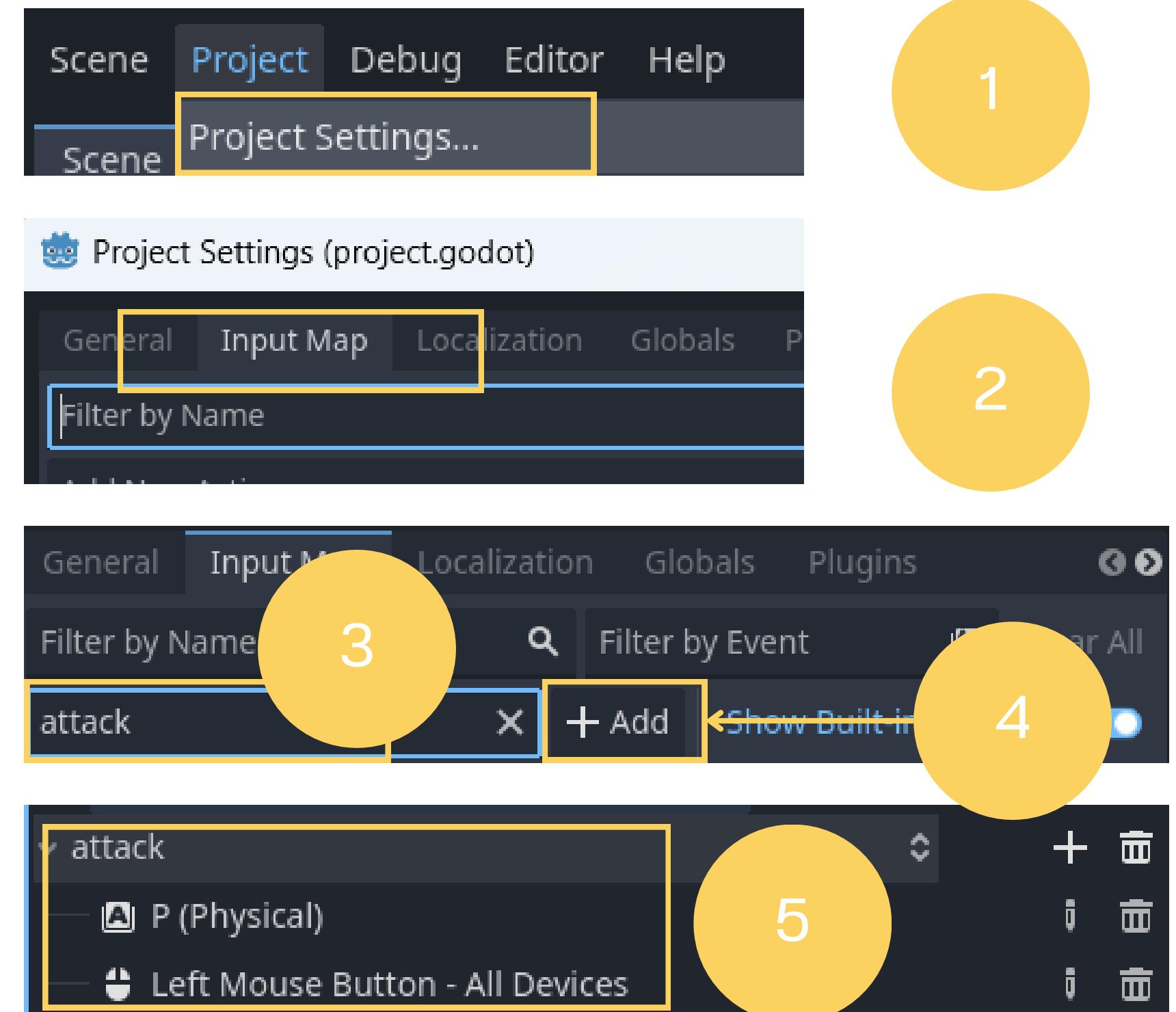
```

38     if Input.is_action_just_pressed("attack"):
39         attack()
40
41     func attack():
42         animation_player.play("Attack")
43         hitbox.disabled = false
44         # Do hitbox วัตถุนี้หลังจากเล่นแอнимชันเสร็จ (หรือตั้งเวลา)
45         await animation_player.animation_finished
46         hitbox.disabled = true
47

```

ชุดคำสั่งเมื่อวงเสร็จแล้วให้หักกวนบทเรียน หลังจากนั้นไปที่ **Project > Project Settings > Input Map** และเพิ่ม:

Action Name	Bindings
attack	P key, Mouse Button Left



ขั้นตอนที่ 5 เพิ่มเงื่อนไข Boolean กับสถานะ

เราต้องแก้ไข playercontroller.gd ใหม่เลือกน้อยในส่วนของการเพิ่มสถานะโจมตีของ Player โดยการใช้งาน Boolean มาช่วย

1

```
var is_attacking = false
```

เก็บสถานะว่า “กำลังโจมตีอยู่หรือไม่” เพื่อป้องกันไม่ให้เคลื่อนไหวหรือเล่นแอนิเมชันอื่นกับในระหว่างการโจมตี

Boolean (บูลีน) คือ ชนิดข้อมูลตรรกะ (Logical Data Type) ที่มีค่าความจริงเพียง 2 ค่าเท่านั้น:

- **true (จริง)**
- **false (เท็จ)**

Boolean ใช้สำหรับ: การตัดสินใจ (เงื่อนไข) ควบคุม flow ของโปรแกรม (เช่น if, while) ตรวจสอบสถานะ เช่น: กำลังเดินอยู่ไหม? กระโดดได้หรือเปล่า?

ไปแก้ไขบางสิ่งในฟังก์ชัน **_physics_process(delta: float)**

ลำดับการทำงาน ให้มีการเช็คตรวจสอบว่าโจมตีอยู่ไหม แก้ไข Code จากเดิมเป็นดังนี้

2

```
if is_attacking:  
    velocity = Vector2.ZERO  
    move_and_slide()  
    return
```

```
9 func _physics_process(delta: float) -> void:  
10     var direction = Vector2.ZERO  
11  
12     # ถ้าโจมตีอยู่ ไม่ให้เคลื่อนที่หรือเปลี่ยนแอนิเมชัน  
13     if is_attacking:  
14         velocity = Vector2.ZERO  
15         move_and_slide()  
16         return
```

เล่นแอนิเมชัน Walk หรือ Idle ให้ปรับไปใช้ ตัวแปรแทน **แก้ไขจาก**

```
if direction == Vector2.ZERO:  
    $AnimationPlayer.play("Idle")  
else:  
    $AnimationPlayer.play("Walk")
```

ปรับแก้เป็น **ตามตัวอย่าง**

```
if direction == Vector2.ZERO:  
    animation_player.play("Idle")  
else:  
    animation_player.play("Walk")
```

1

```
39  if direction == Vector2.ZERO:  
40      animation_player.play("Idle")  
41  else:  
42      animation_player.play("Walk")
```

การตรวจจับการโจมตีด้วยปุ่ม P หรือคลิกซ้าย ในตอนแรก
เราจะเรียกคำสั่งแบบนี้

```
# ตรวจจับการโจมตีด้วยปุ่ม P หรือคลิกซ้าย  
if Input.is_action_just_pressed("attack"):  
    attack()
```

ปรับแก้เป็น **ตามตัวอย่าง**

```
# ตรวจจับการโจมตีด้วยปุ่ม P หรือคลิกซ้าย  
if Input.is_action_just_pressed("attack") and not is_attacking:  
    attack()
```

2

เช่นกันเราจะปรับแก้ฟังก์ชันนี้ใหม่

```
func attack():  
    animation_player.play("Attack")  
    hitbox.disabled = false
```

```
# ปิด hitbox อัตโนมัติหลังจากเล่นแอนิเมชันเสร็จ (หรือตั้งเวลา)  
await animation_player.animation_finished  
hitbox.disabled = true
```

ฟังก์ชัน **Attack()** กี่มีการแก้ไขแล้ว

```
func attack():
    is_attacking = true
    animation_player.play("Attack")
    hitbox.disabled = false

    await animation_player.animation_finished

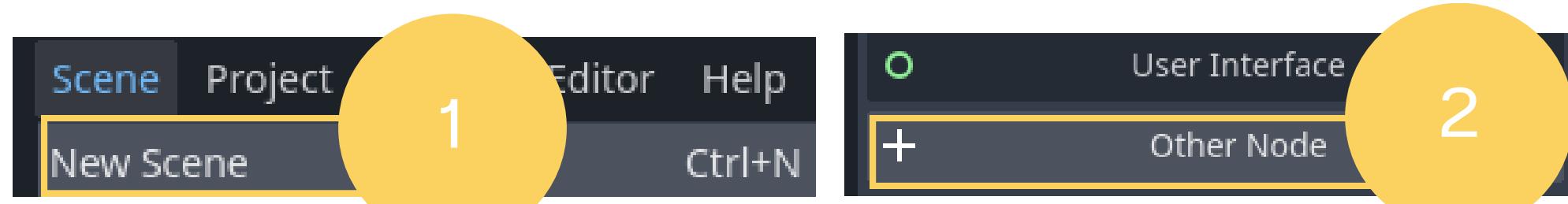
    hitbox.disabled = true
    is_attacking = false
```



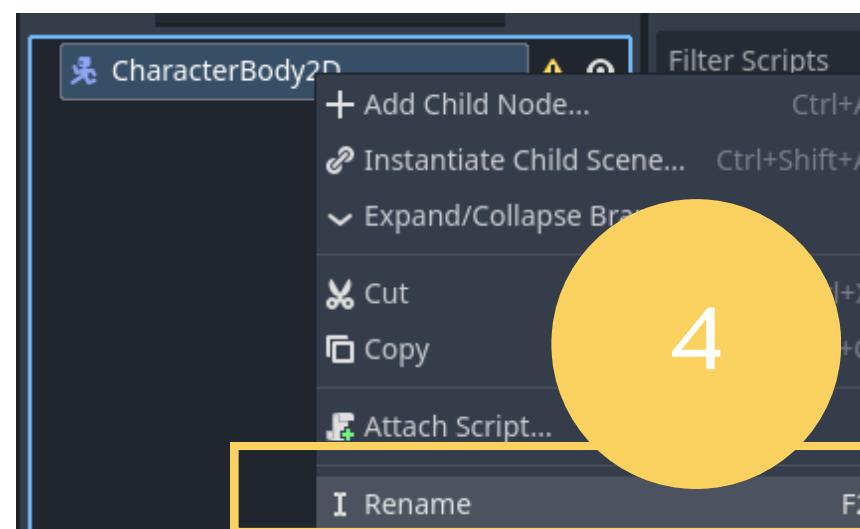
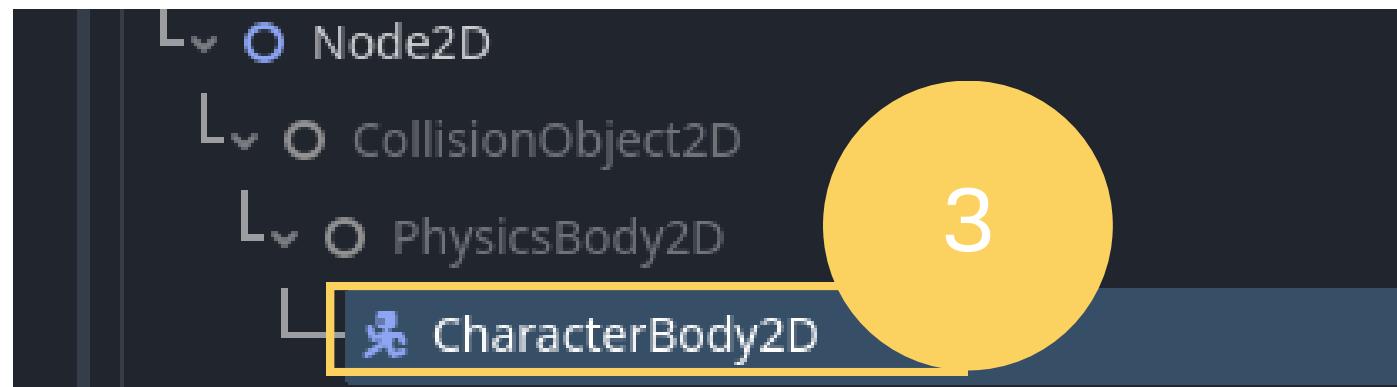
เมื่อเรากด P หรือคลิกซ้ายจะมีการโجمตี และเปิดใช้งาน HitBox ที่ซ่อนไว้ให้ทำงานทำให้เรา CollisionDetect กับศัตรูหรือบางสิ่งได้

ขั้นตอนที่ 6 สร้างศัตรูในเกม (ต้นแบบ)

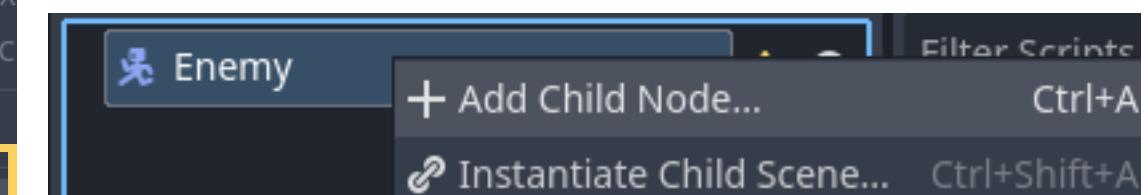
เราจะทำการออกแบบและสร้างศัตรูในเกมกันซึ่งเราจะต้องทำการ New Scene ใหม่ขึ้นมาให้เรียบร้อย **New Scene...** ตั้งชื่อ **enemy.tscn**

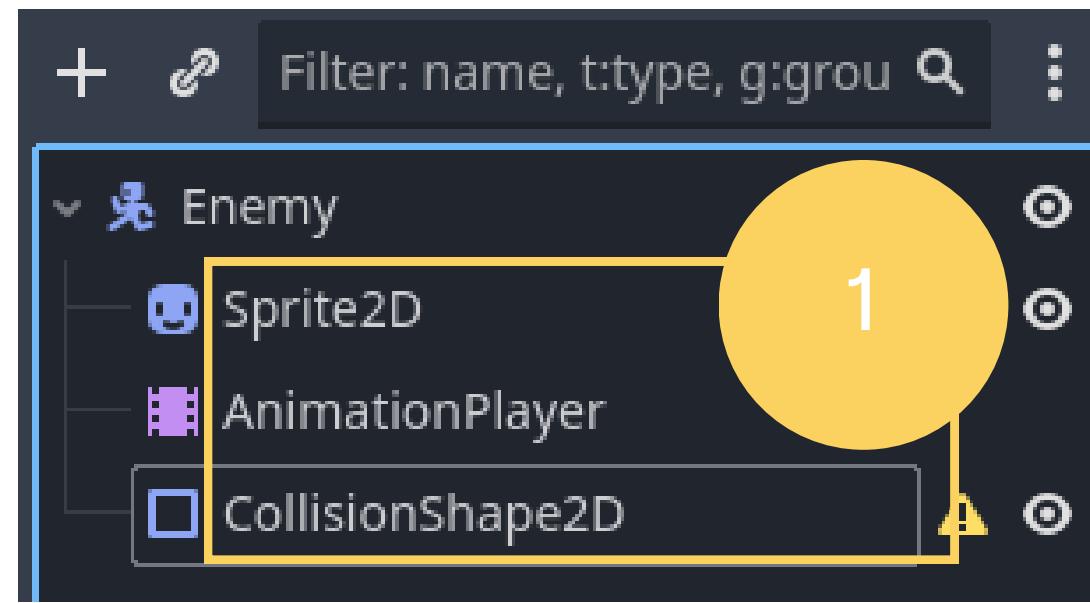


คลิกเมนู **Scene** เลือก **New Scene** เมื่อสร้างแล้วให้ Save As ชื่อ **enemy.tscn** โดยเลือก **Other Node** แล้วเลือก **CharacterBody2D** เป็น Node หลัก

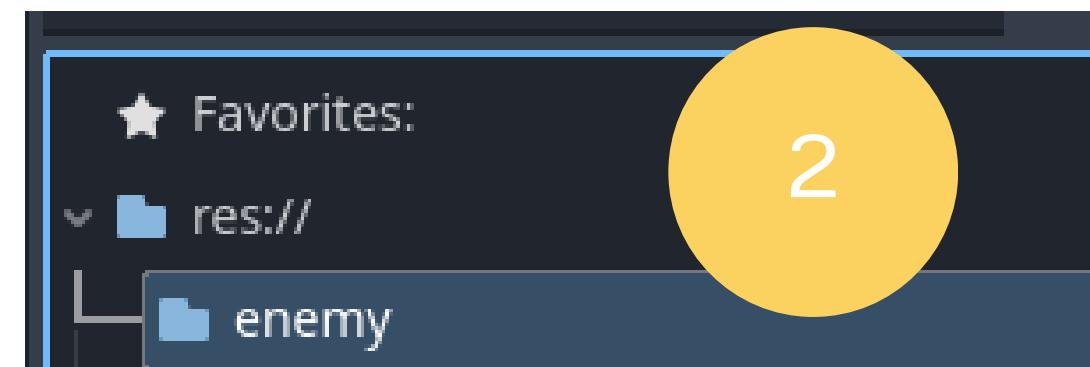


ทำการคลิกขวาที่ แล้วเลือก Rename ตั้งชื่อว่า **Enemy** หลังจากนั้นเลือก Add Child Node ใหม่



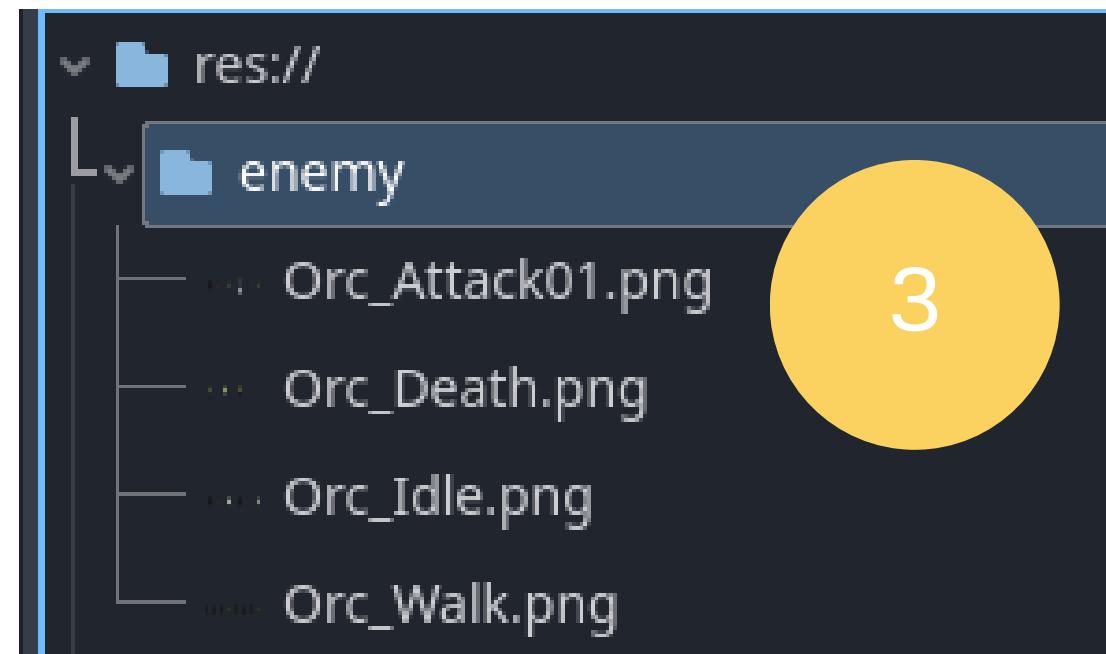


วาง Child Node ให้เป็นไปตามตัวอย่าง
สังเกตคือจะใกล้เคียงกับของ player ทุก
อย่าง เพราะมีคุณลักษณะที่ใกล้เคียงกัน

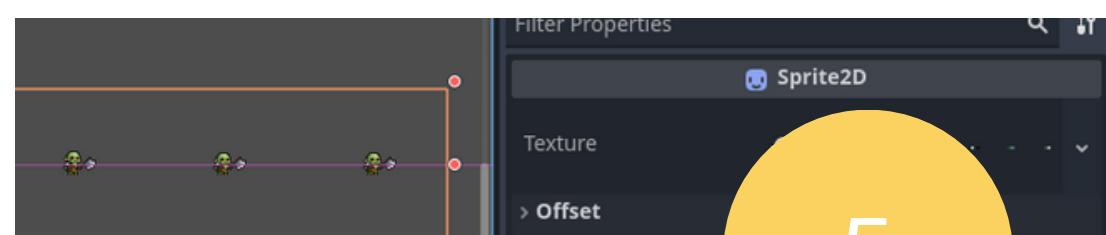
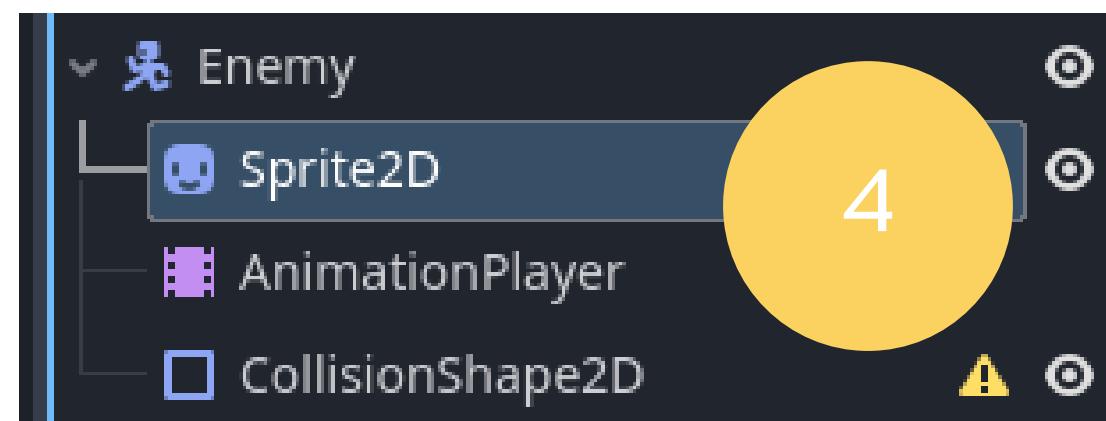


ไปที่ FileSystem ทำการคลิกขวาที่ res://
หลังจากนั้นสร้าง folder ใหม่ชื่อ **enemy**
เพื่อจะได้นำรูปภาพที่เราจะใช้มาใส่ลงใน

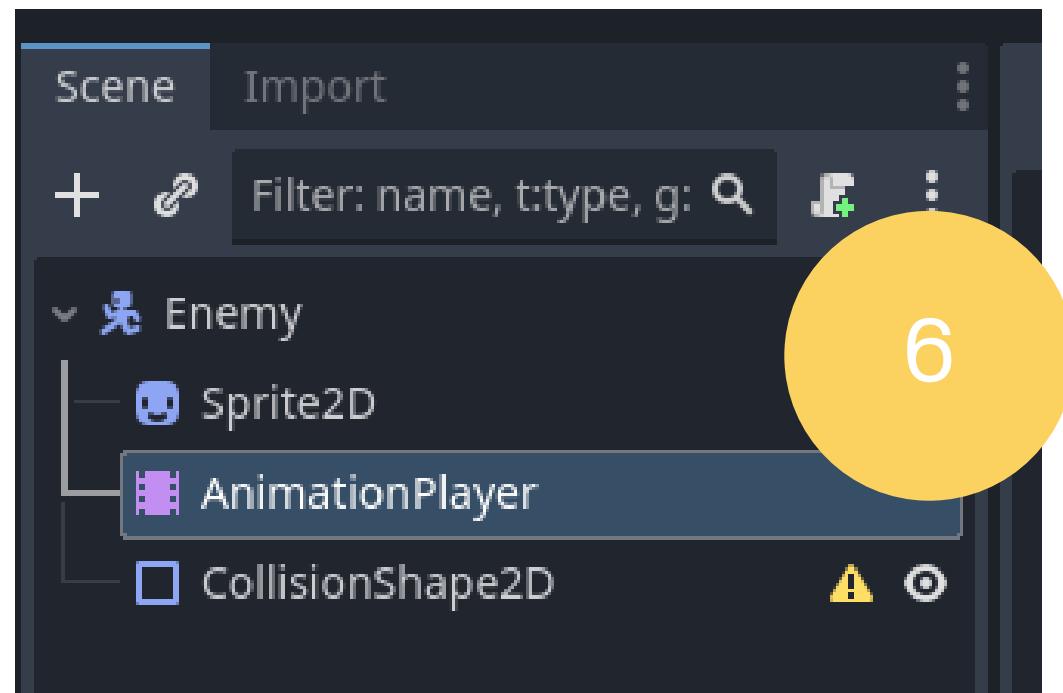
** เป็นการทวนบทเรียนก่อนหน้า



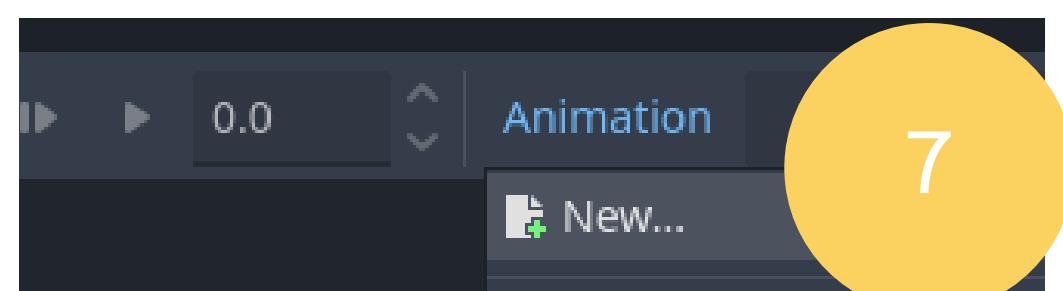
เมื่อบรรจุการจัดวางเป็นที่เรียบร้อยแล้ว ทำการ
คลิกที่ Sprite2D แล้วไปดูที่ Inspector



ทำการลากไฟล์ Orc_Idle เข้าไปใน Texture

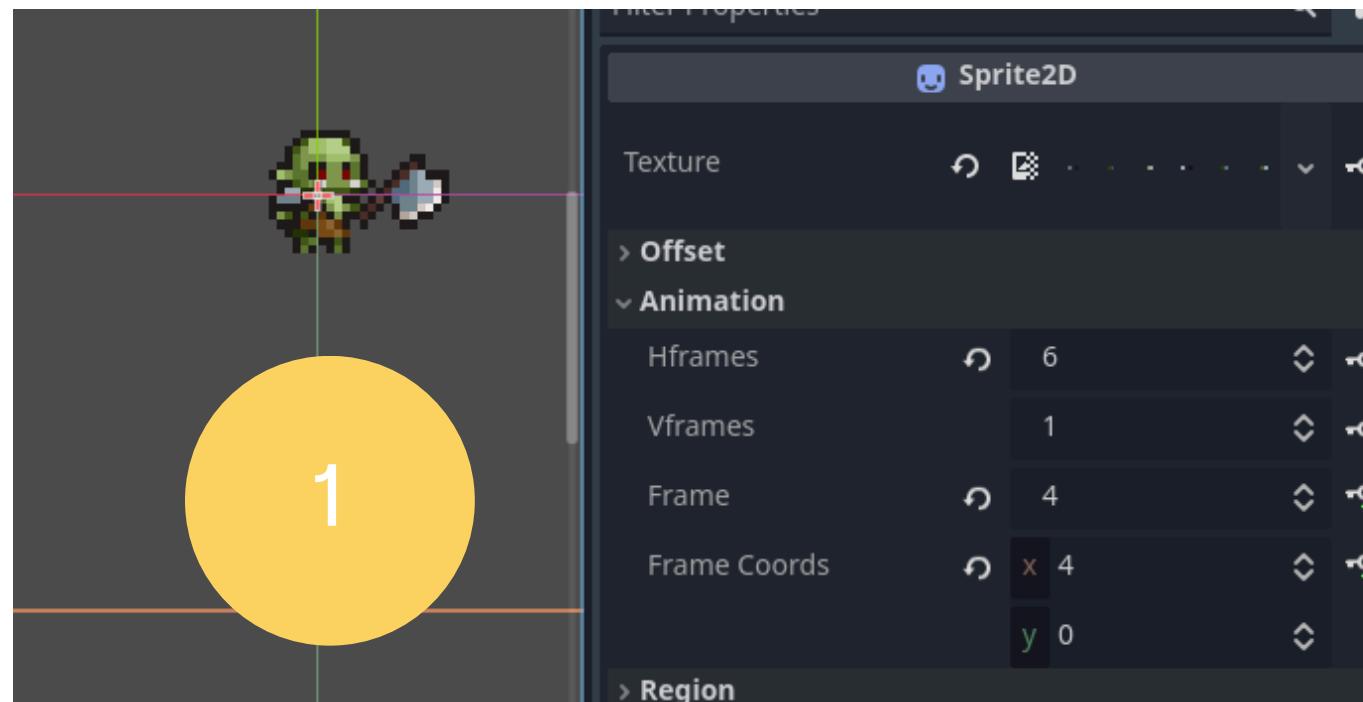


คลิกที่ AnimationPlayer และสร้าง
Animation ใหม่ขึ้นมา



และบิเมชันมี:

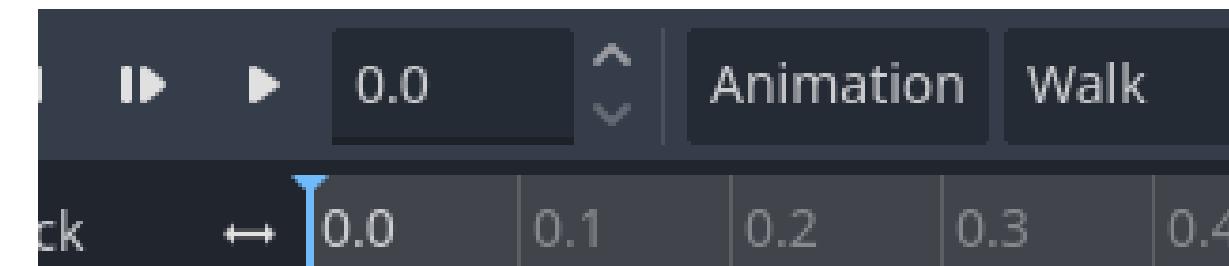
- Idle
- Walk
- Attack
- Death



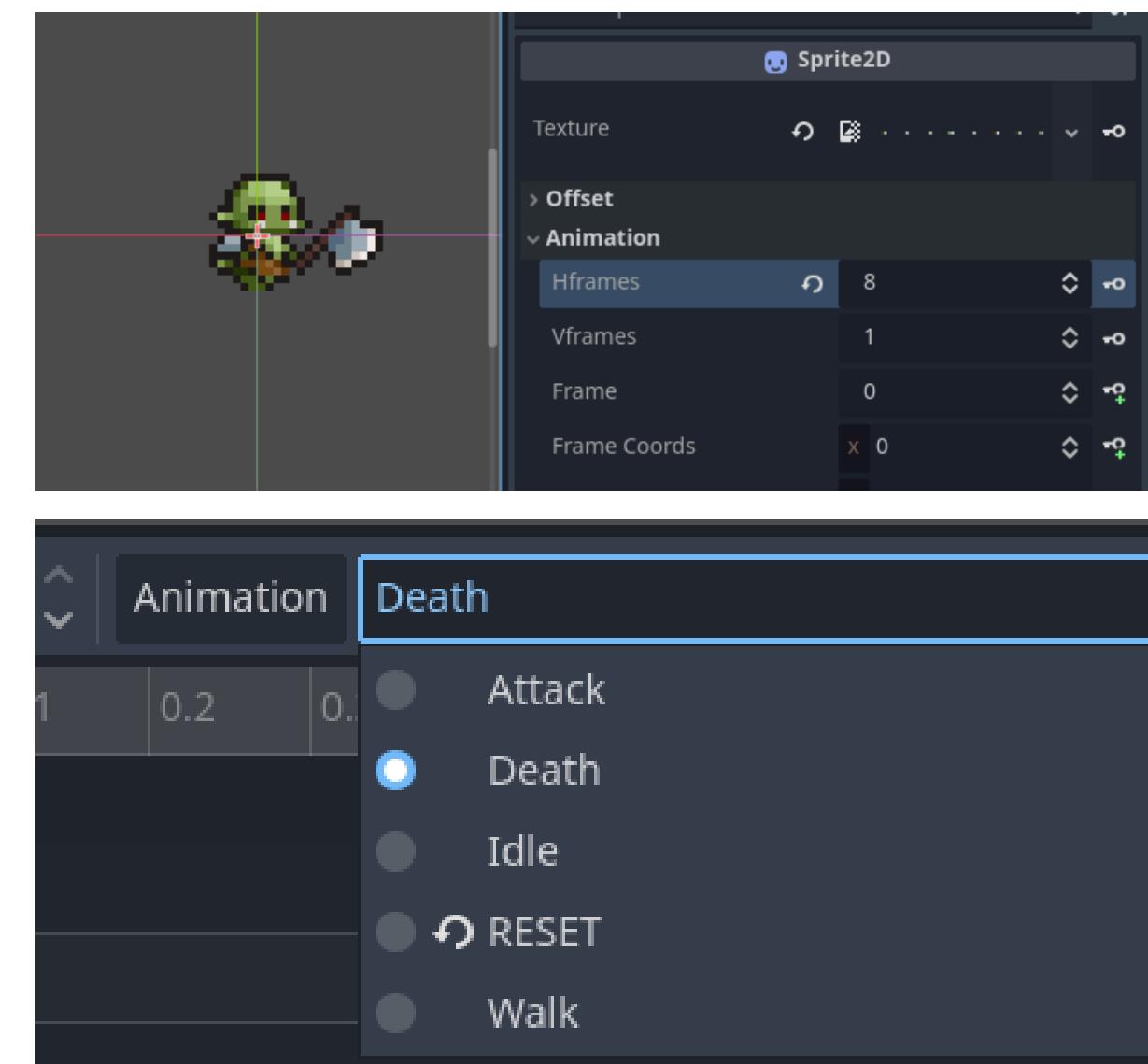
กบกวนบทเรียน: ลากไฟล์ Orc_Idle.png ไปวางใน Texture หลังจากนั้นปรับ HFrames ให้เรียบร้อย



บันทึก KeyFrames ก่อนใช้เข้าไปใช้ครับ



สร้างแอโนเมชัน Walk และ กบกวนขั้นตอนนำเข้า Texture ใน Sprite2D และ AnimationPlayer



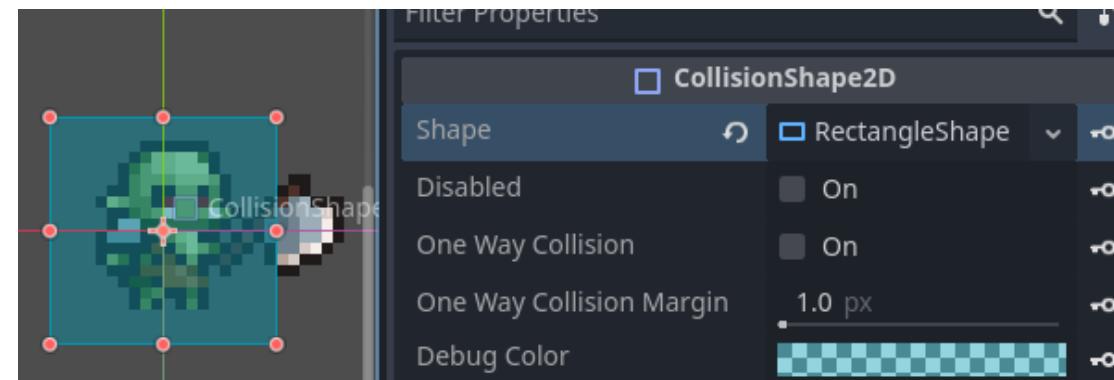
CHALLENGE

กำหนดขั้นตอนเดิมๆ และ
จัดการให้ครบถ้วน
แอโนเมชันก็ง

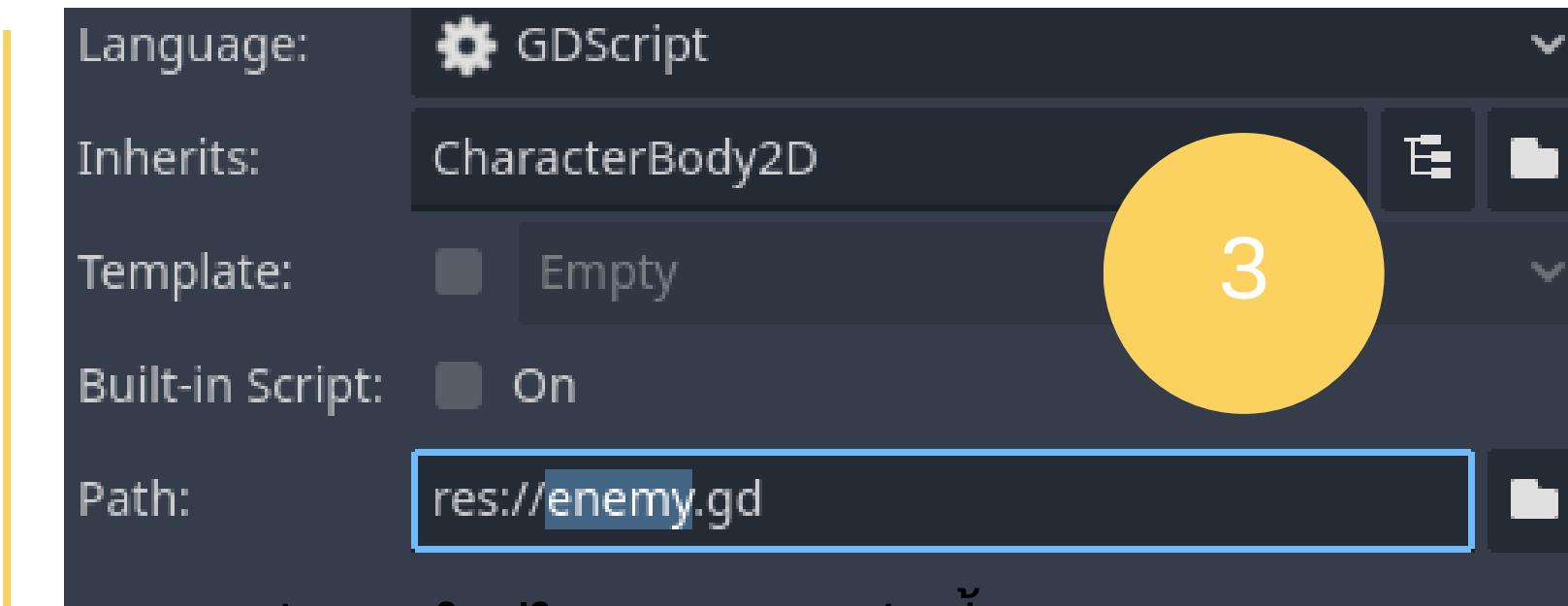
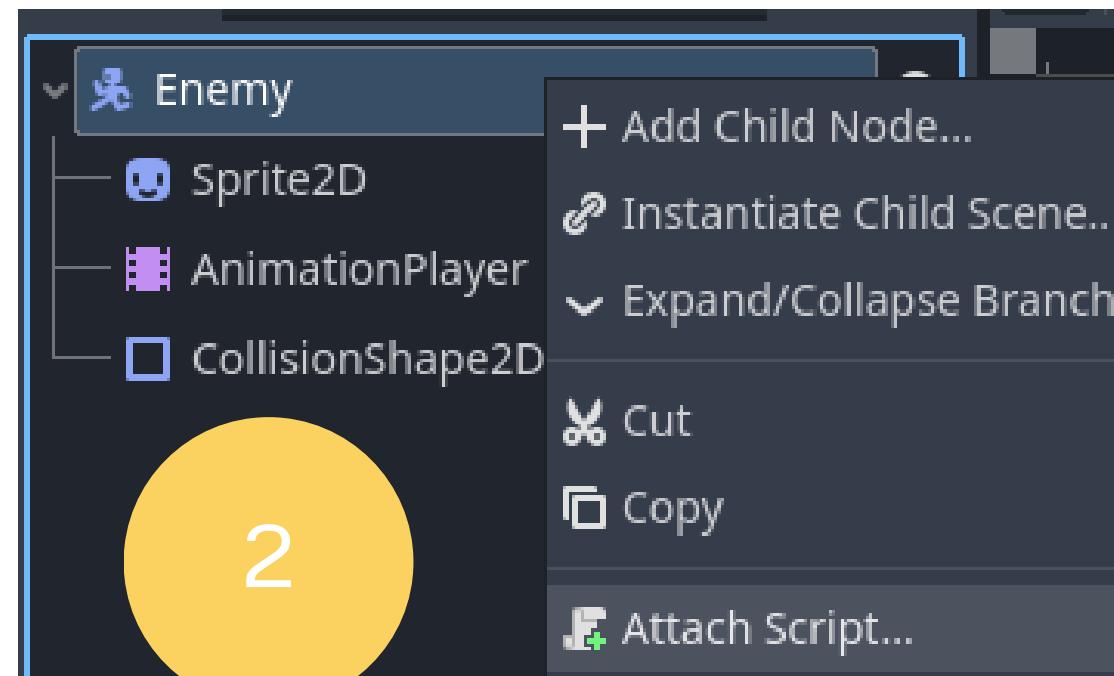
- Idle
- Walk
- Death
- Attack



คลิกเลือก CollisionShape2D แล้วไปที่ Inspector เพื่อปรับค่า Shape ของ Collision



เลือก Shape เป็น **Rectangular2D** ให้กับเม้น
หลังจากนั้นคลิกขวาที่ Enemy แล้วเลือก
Attach Script... สร้างไฟล์ **enemy.gd**



ประกาศตัวแปรใหม่ใน enemy.gd ดังนี้

```
class_name Enemy
```

```
@export var speed: float = 50.0
@export var player_path: NodePath
@onready var player = get_node_or_null(player_path)
```

- ชื่อคลาสว่า Enemy ทำให้สามารถเรียกใช้จากที่อื่นในโปรเจกต์ได้ เช่น `var enemy = Enemy.new()`
- ประกาศตัวแปร speed เป็นความเร็วในการเคลื่อนที่ของศัตรู และสามารถตั้งค่าได้จาก Inspector
- ประกาศตัวแปร player_path เป็น NodePath เพื่อให้ระบุได้ว่า Player อยู่ตรงไหนของฉากผ่าน Inspector เมื่อเกมเริ่ม (ready), ระบบจะค้นหา Node Player ตาม path ที่กำหนดไว้

```

func _physics_process(delta: float) -> void:
    if not player:
        return

    var direction = (player.global_position - global_position).normalized()
    velocity = direction * speed
    move_and_slide()

    # กลับ sprite ตามทิศทาง
    if direction.x != 0:
        $Sprite2D.flip_h = direction.x < 0

```

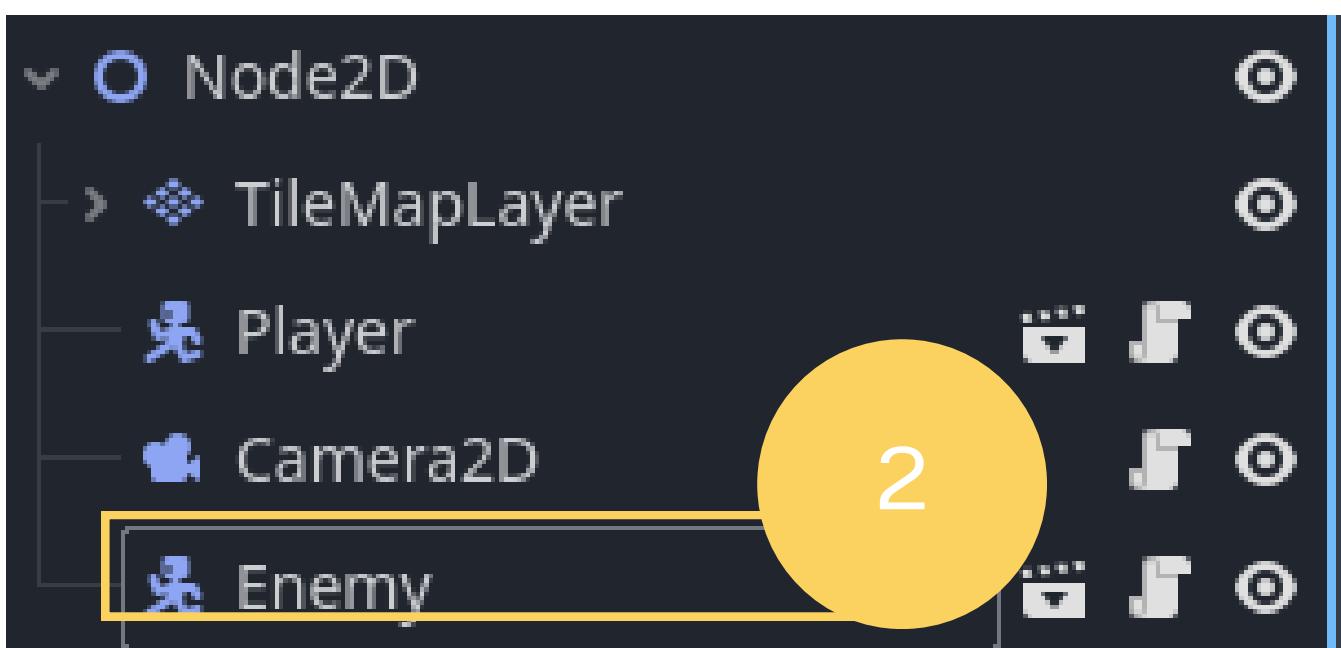
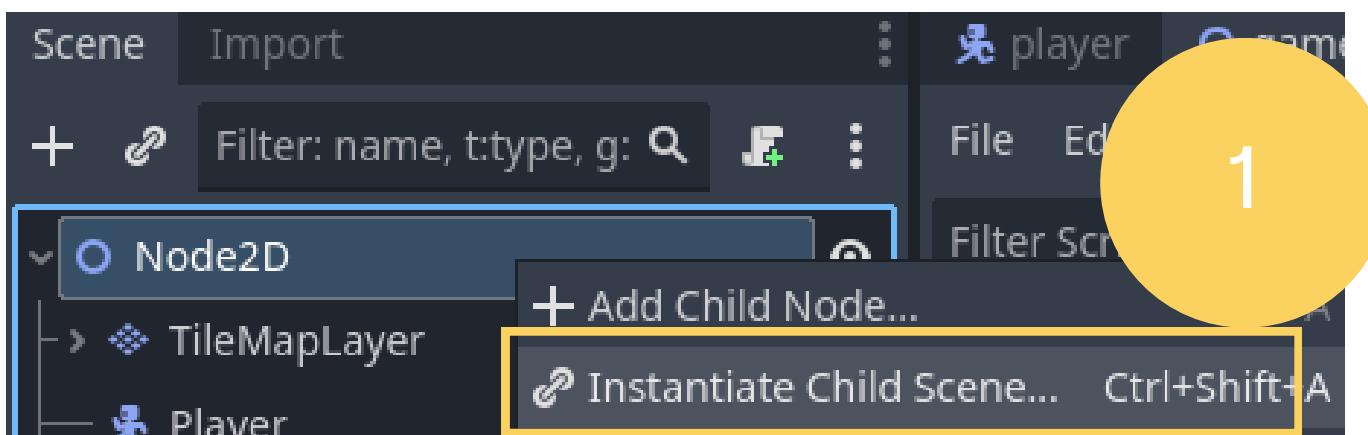
พังก์ชันนี้ถูกเรียกทุกเฟรมโดยระบบฟิสิกส์ของ Godot

คำนวณทิศทางจาก Enemy → ไปยัง Player

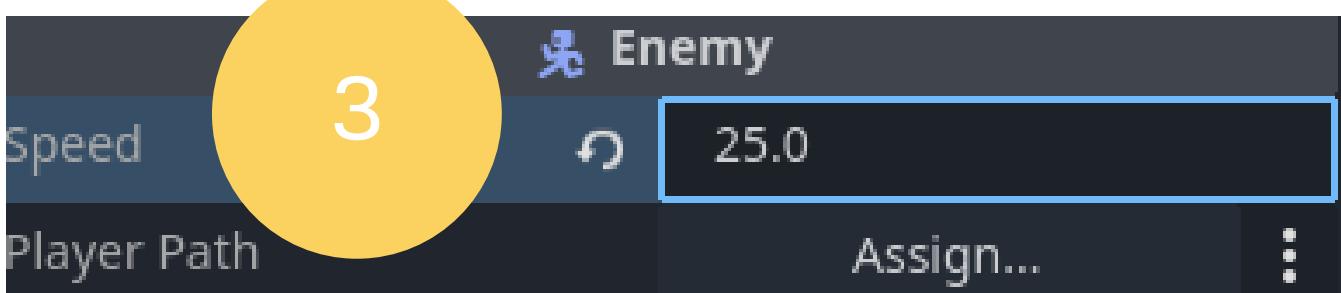
`player.global_position - global_position` = เวกเตอร์ไปยังเป้าหมาย
`.normalized()` ทำให้ได้วेकเตอร์ขนาดยาว 1 (เพื่อคูณกับความเร็ว)

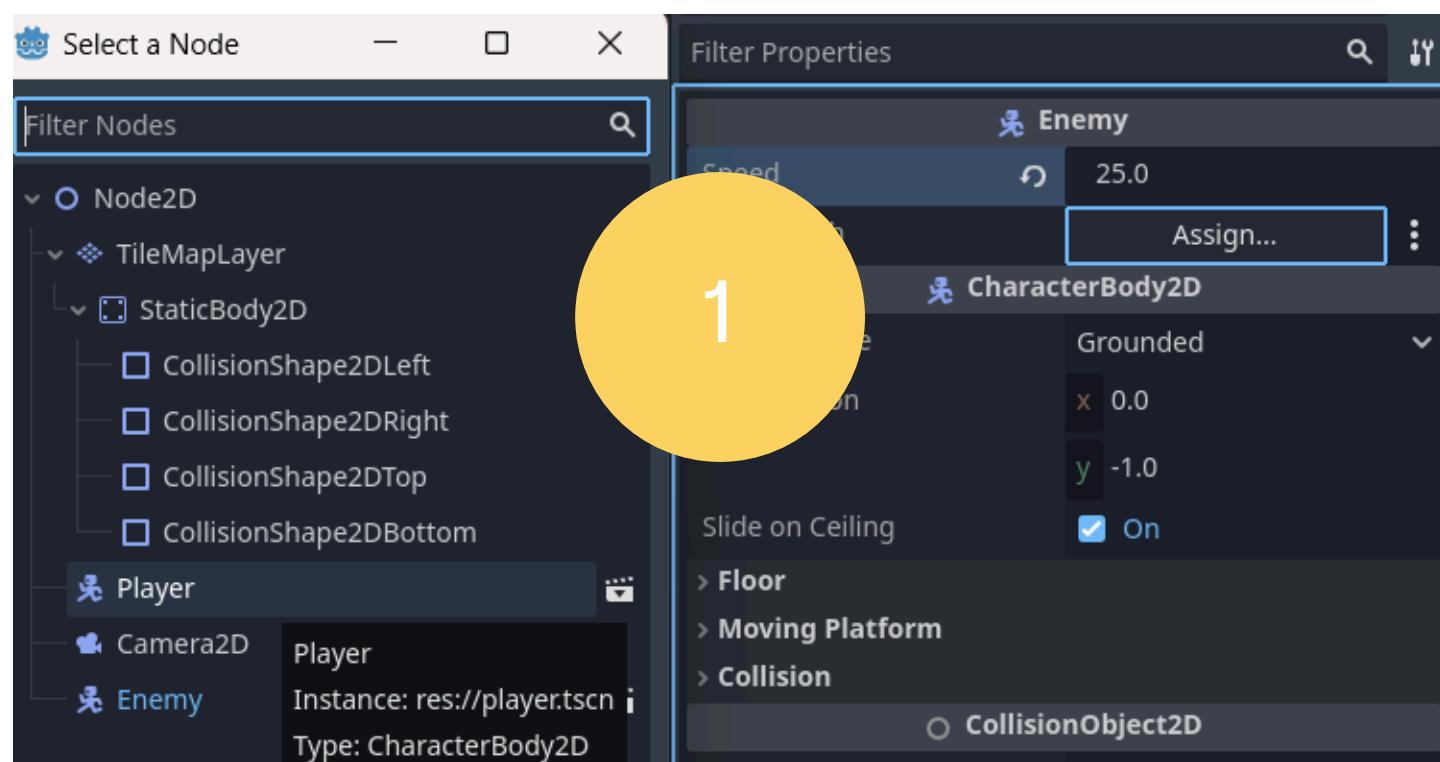
- ถ้าเดินไปทางซ้าย ($x < 0$) → ให้ `flip_h = true` (กลับด้าน)
- ถ้าเดินไปทางขวา ($x > 0$) → ให้ `flip_h = false` (หันหน้าไปทางขวา)

ไปที่ Scene ชื่อ **game.tscn** แล้วคลิกขวาที่ Node2D ทำการเลือก Instantiate Child Scene... แล้วให้เลือก enemy.tscn

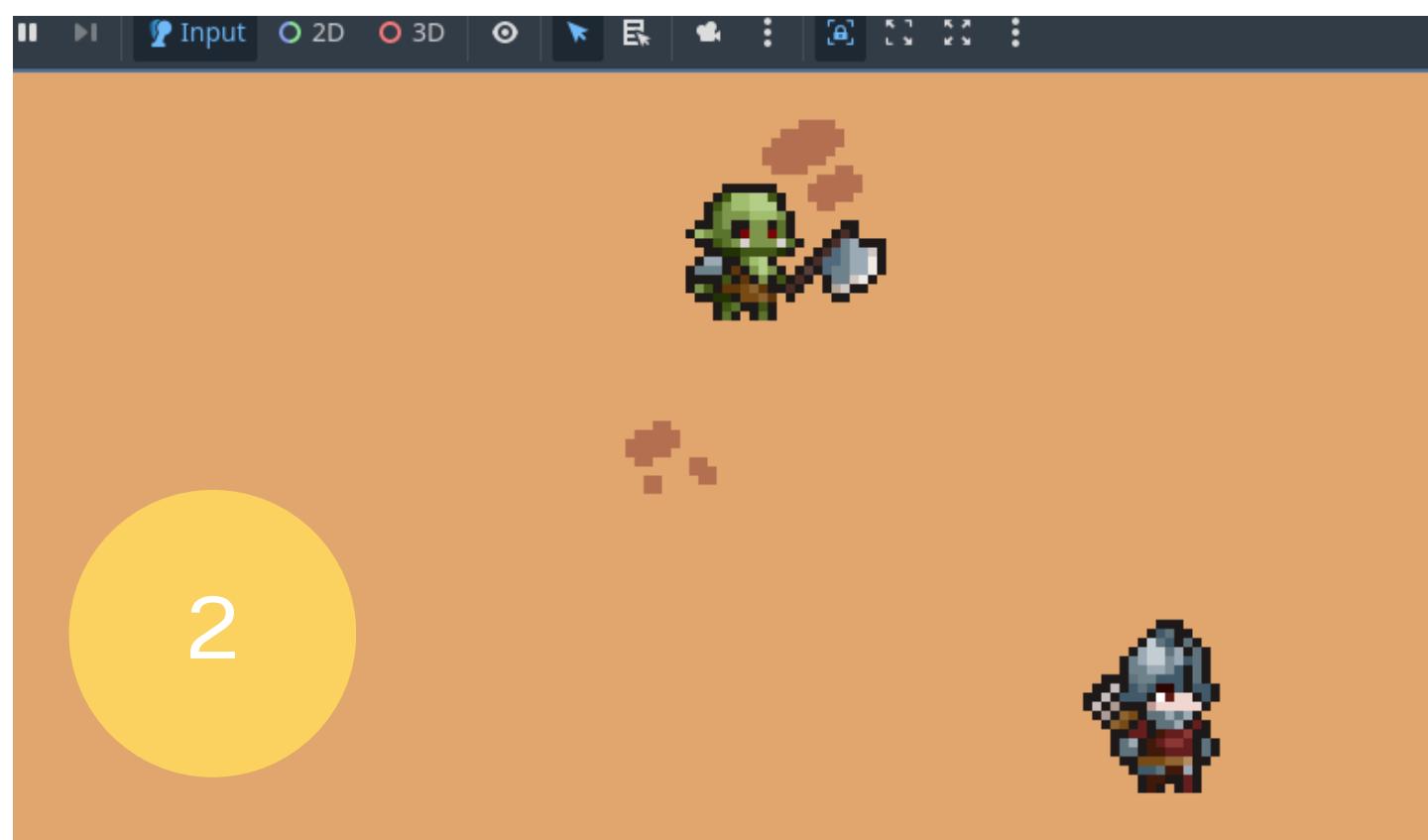


ปรับ Speed ของ `enemy.gd` ให้เป็น 25 และคลิก Assign เลือก Player เท่านี้คัตติรูก็จะเดินตาม Player





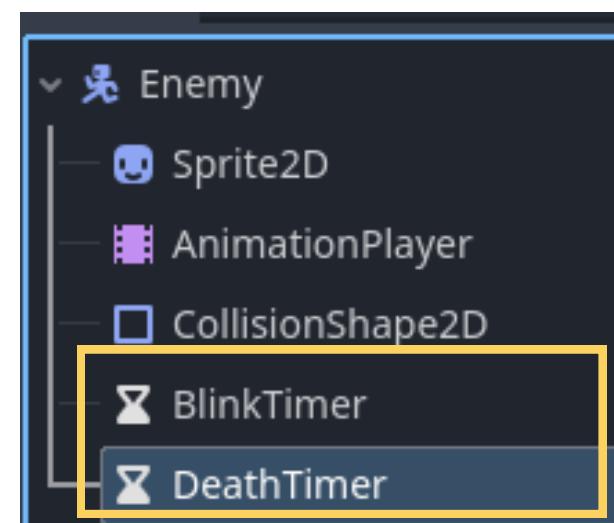
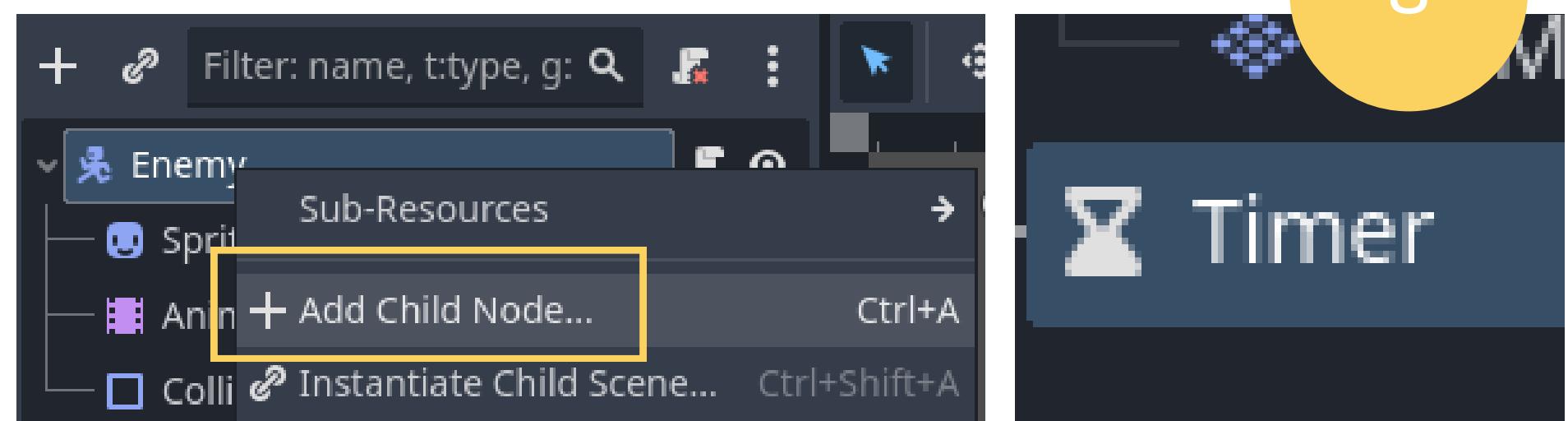
คลิก Assign เลือก Player เท่านี้คัตตูร์ก็จะเดินตาม Player



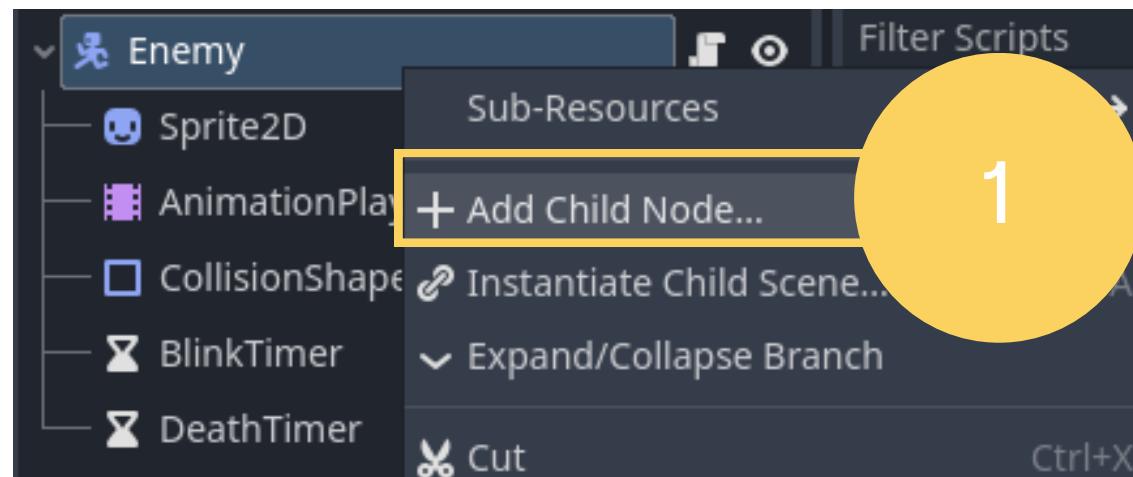
ขั้นตอนที่ 7 สร้างแอนิเมชันให้คัตตูร์

เราจะออกแบบ Behavior (พดติกธรรม) ให้กับคัตตูร์ของเราโดยการกำหนดพดติกธรรม Behavior ดังนี้:

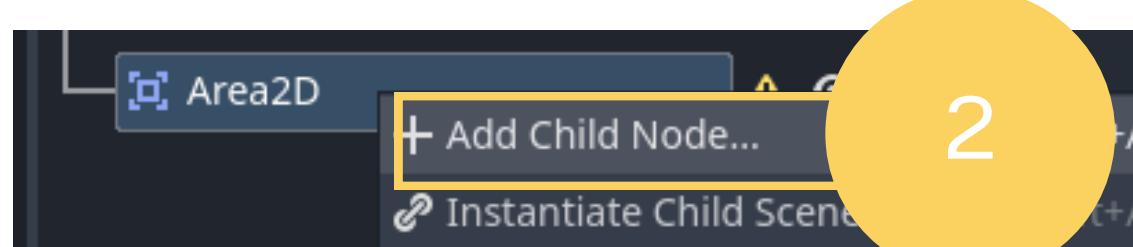
- เดินเข้าหา Player พร้อมแสดงแอนิเมชัน Walk
- ถ้าเข้าใกล้ Player น้อยกว่า 1 → แสดง Attack
- ถ้าใน HitBox → กระเด็นโดยหลัง + แสดงสีแดง
- โดนครับ 2 ครั้ง → เล่นแอนิเมชัน Death 0.8
วินาที → กระพริบ 0.5 วินาที แล้วหายไป



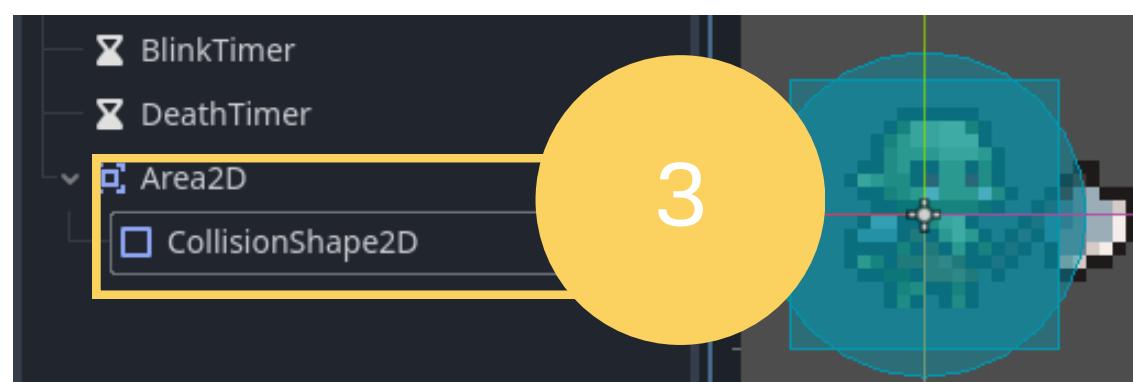
- สร้าง Timer 2 ตัวเปลี่ยนชื่อเป็น
- Timer และ **BlinkTimer**
- Timer ที่สองคือ **DeathTimer**



กลับไปที่ **enemy.tscn** คลิกขวาที่ Enemy
เลือก **+Add Child Node...** หลังจากนั้นเลือก
Area2D

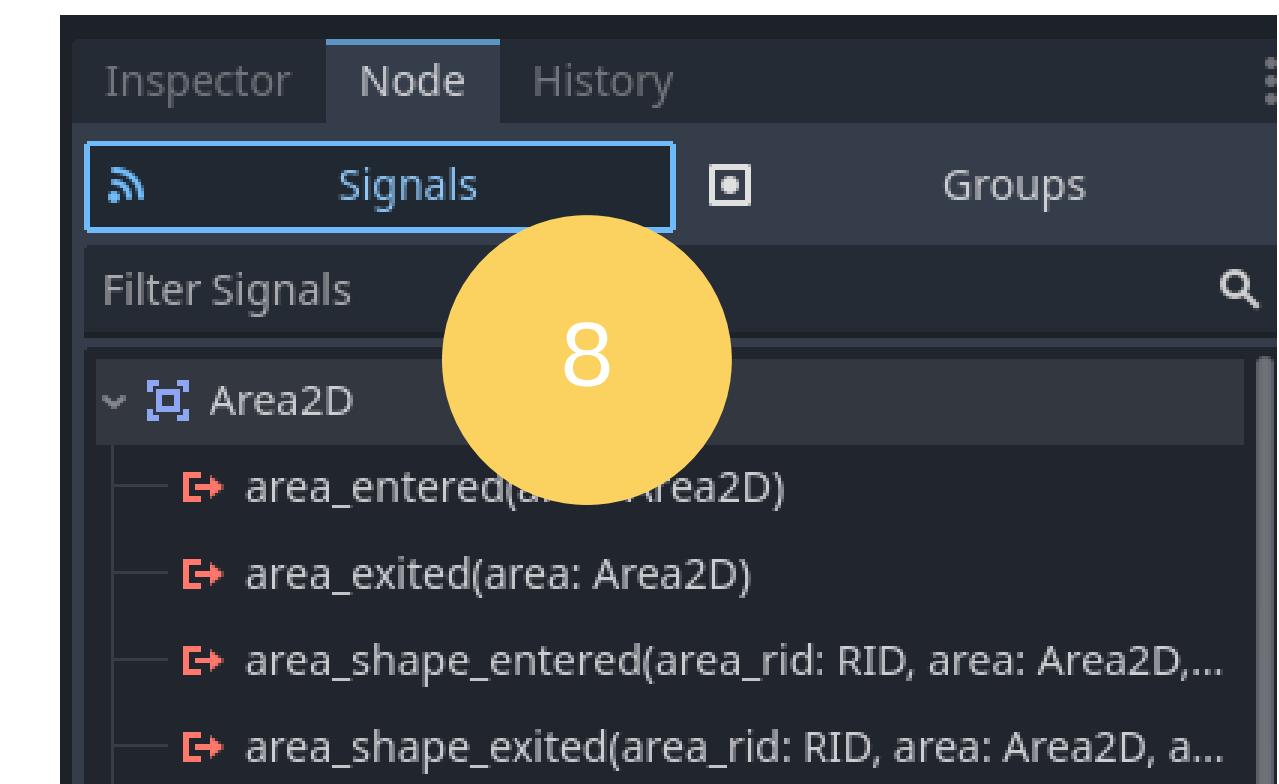
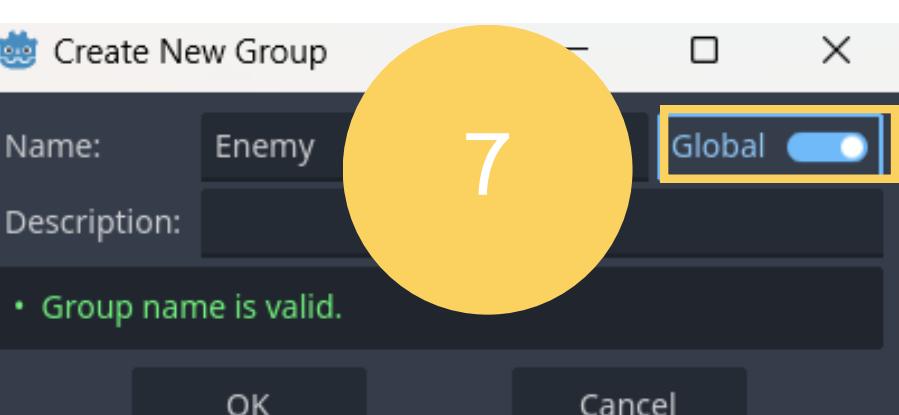
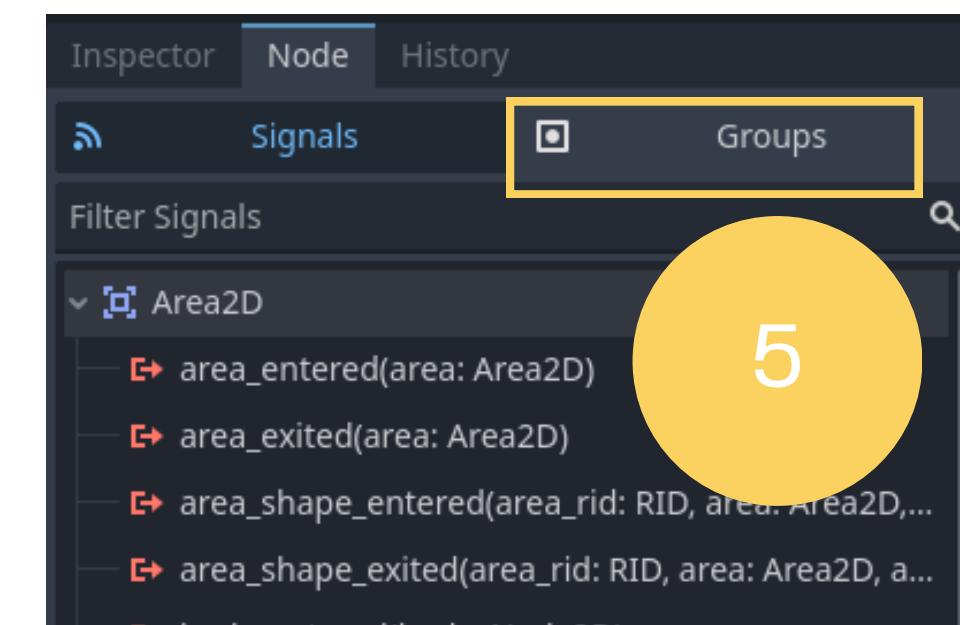
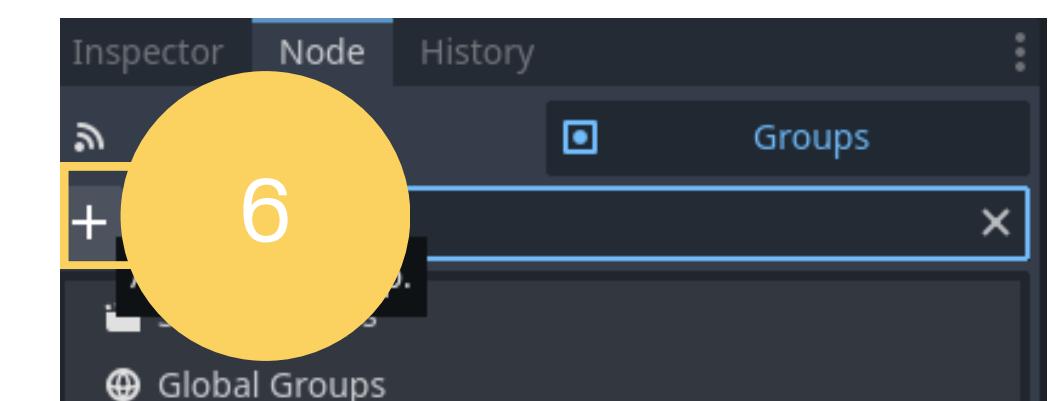


คลิกขวาที่ Area2D หลังจากนั้นเลือก **+Add
Child Node...** หลังจากนั้นเลือก
CollisionShape2D เลือก Rectangle

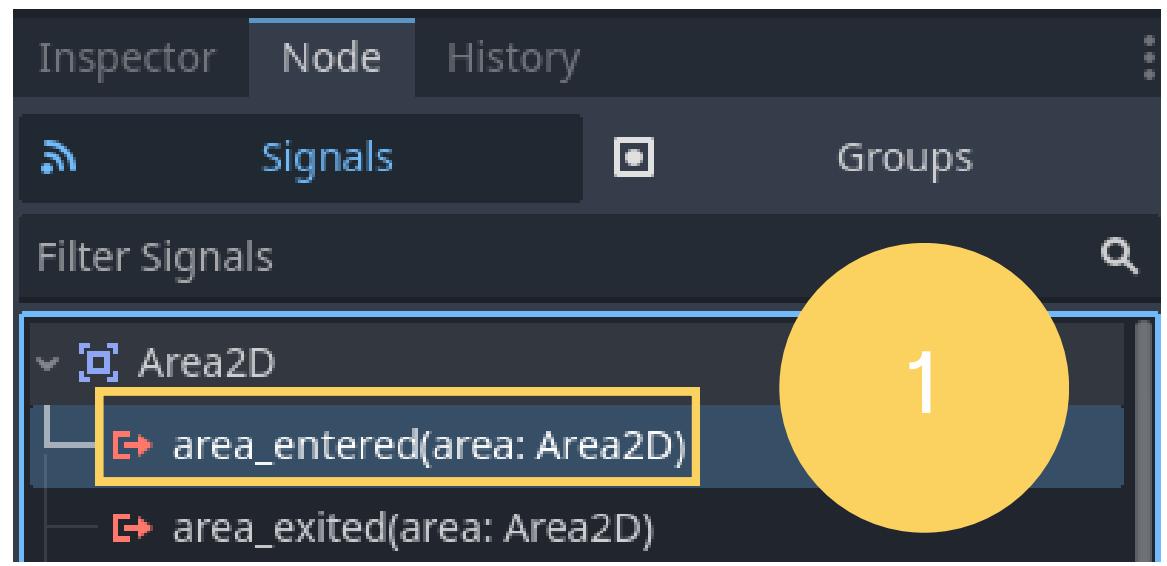


คลิกที่ Area2D หลังจากนั้นไปที่ แท็บ
node ของ Inspector

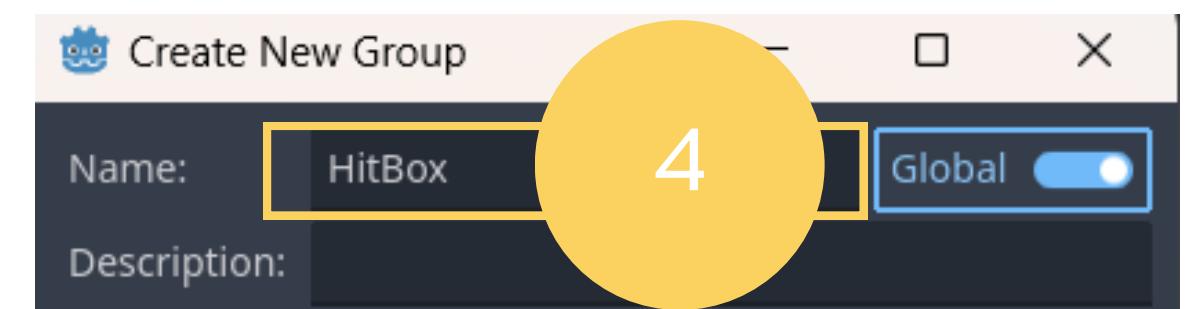
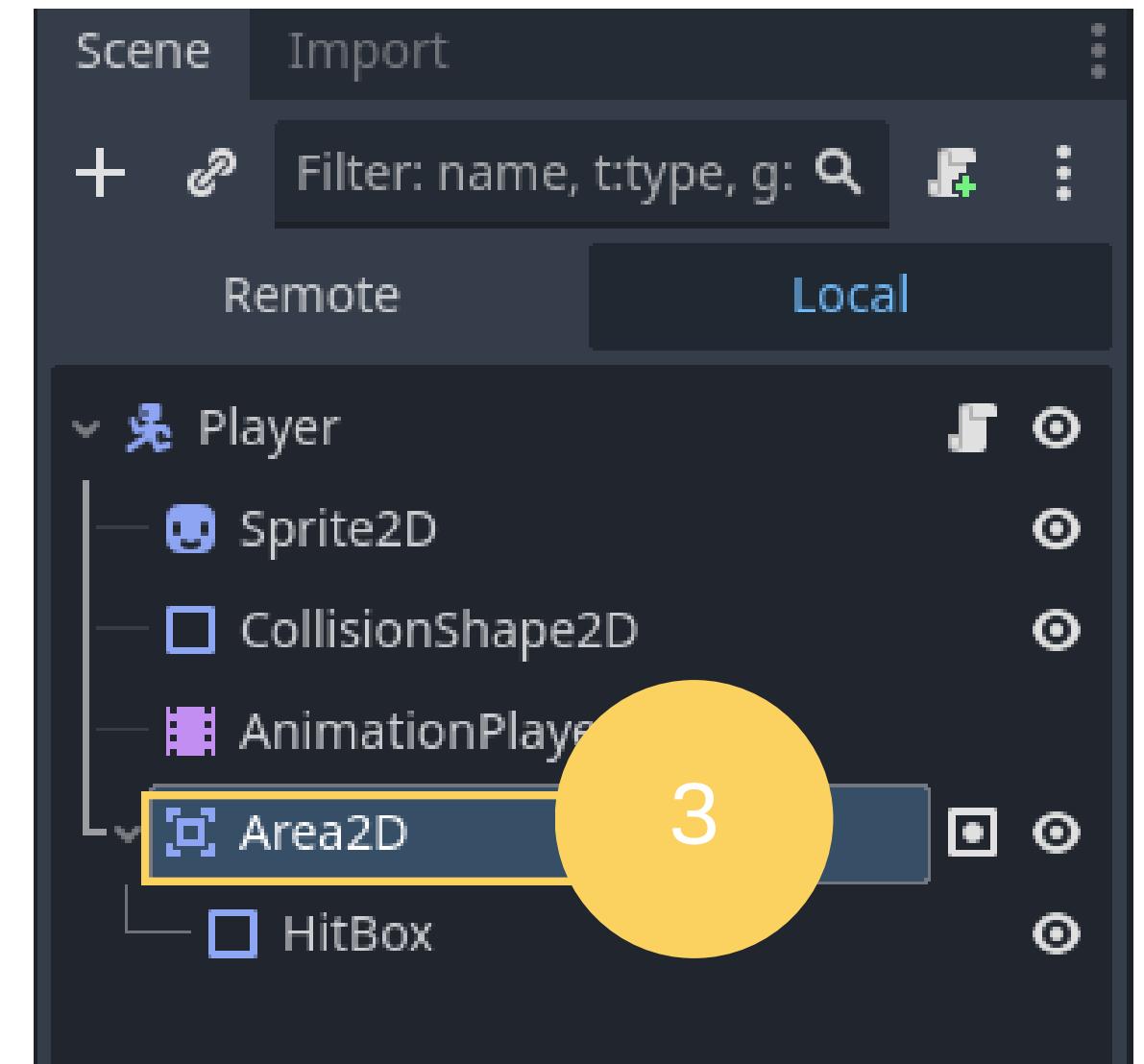
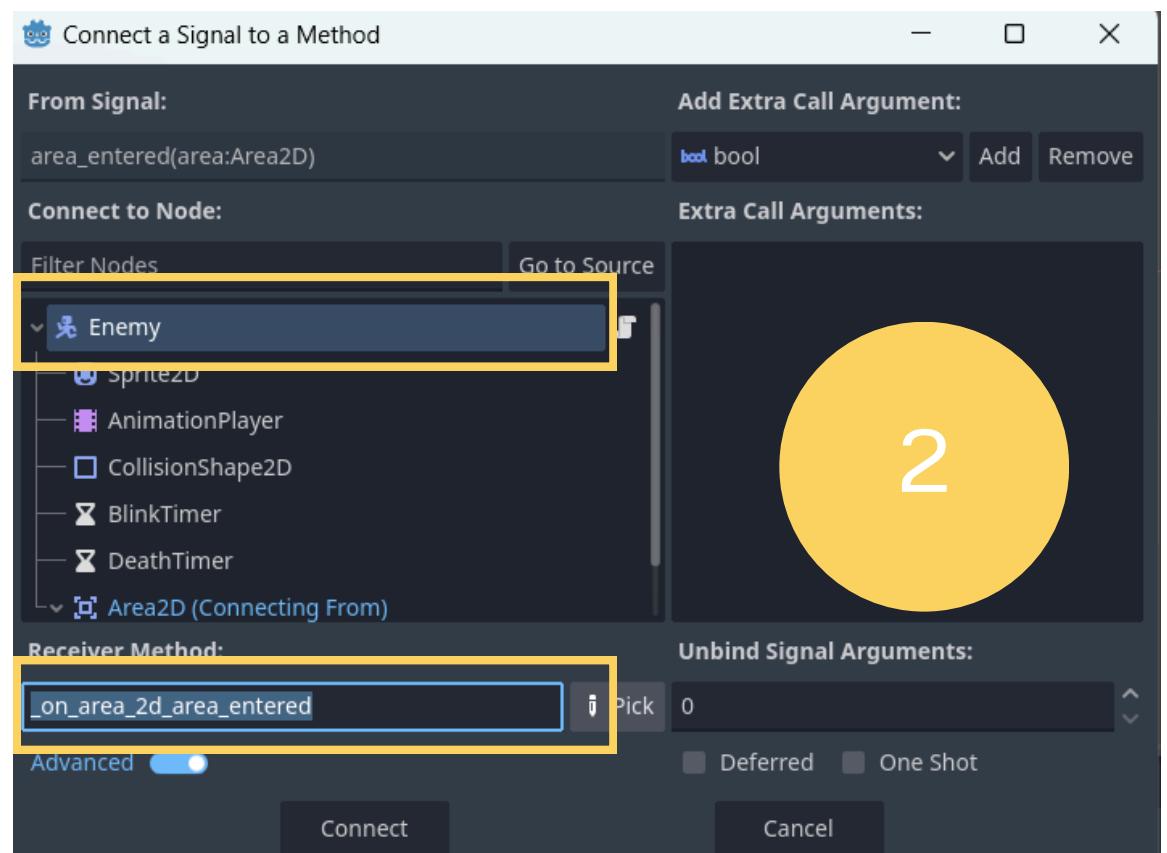
ให้เราเลือก **Groups** ตั้งชื่อว่า **Enemy**



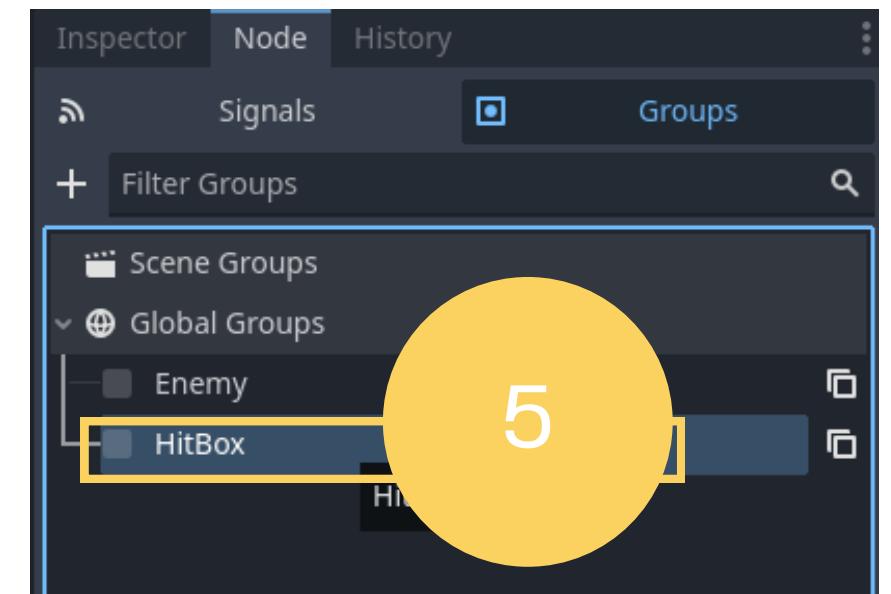
กลับมาที่แท็บที่ซ่อน Node เราจะ
มาทำงานร่วมกับ Signal และ
ต้องเขียนโปรแกรม ร่วมกับมัน



คลิกเลือก Signals หลังจากนั้นเลือก area_entered() ระบบจะทำการเชื่อมโยง Node ของตัวละคร



ไปที่ player.tdcn คลิกที่ Area2D สร้าง Group เป็น Global เหมือน กันตั้งชื่อ HitBox เพื่อให้การโจรต์ ของเรางส่ง Signal ไปยัง Enemy



```
@export var knockback_distance: float = 50.0
@onready var sprite: Sprite2D = $Sprite2D
@onready var animation_player: AnimationPlayer = $AnimationPlayer
@onready var blink_timer: Timer = $BlinkTimer
@onready var death_timer: Timer = $DeathTimer
```

ประกาศตัวแปรใหม่เพื่อเข้าเงื่อนไขตามพดๆกรรมของเรา โดยประกาศตัวแปรใน GDScript เพื่อควบคุมพดๆกรรมของศัตรูในเกม

โดย **@export var knockback_distance** เป็นค่าระยะที่ศัตรูจะถูกกระเด็นโดยหลังเมื่อโดนโจมตี ซึ่งสามารถปรับได้จาก **Inspector** ใน **Editor**

ส่วน **@onready** คือการกำหนดตัวแปรกี่เชื่อมกับ Node ภายใต้ **(Scene)** หลังจากโหลดเรียบร้อย: **sprite** ใช้ควบคุมภาพตัวศัตรู, **animation_player** ใช้เล่นแอนิเมชัน, **blink_timer** และ **death_timer** เป็นตัวจับเวลา (Timer) สำหรับการกระพริบและการลอบศัตรูหลังจากตายตามลำดับ ทำให้เราควบคุมพดๆกรรมของศัตรูได้อย่างเป็นระบบและยึดหยุ่นผ่าน Node กี่ผู้กิว

```
var is_dead = false
var hit_count = 0
var blink_count = 0
var knockback_vector = Vector2.ZERO
```

ตั้งตัวแปร Boolean ไว้เช็คเงื่อนไขการโดน Hitbox จาก Player โฉมตี และกำหนดค่า Timer กระพริบ

แก้ไขฟังก์ชัน **func _physics_process(delta):**

```
if not player:
    return
```

แก้ไขเป็น

```
if not player or is_dead:
    return
```

```

8  export var KnockbackEnemy: Node = null
9  @onready var sprite: Sprite2D = $Sprite2D
10 @onready var animation_player: AnimationPlayer = $AnimationPlayer
11 @onready var blink_timer: Timer = $BlinkTimer
12 @onready var death_timer: Timer = $DeathTimer
13
14 var is_dead = false
15 var hit_count = 0
16 var blink_count = 0
17 var knockback_vector = Vector2.ZERO
18
19 func _physics_process(delta: float) -> void:
20     if not player or is_dead:
21         return

```

ເອົາໄວ້ຕຽບສອບດູການເບື້ອງໃຫ້ເຕີເລະບັບກັດ ລັງຈາກນີ້ໃຫ້ປັບແກ້ໄນ ສ່ວນ
ຂອງ Direction ກີ່ສິກາງຂອງ Enemy

```

var direction = (player.global_position - global_position).normalized()
velocity = direction * speed
move_and_slide()

```

ແກ້ໄນເປັນ

```

var distance = global_position.distance_to(player.global_position)
if distance < 1.0:
    velocity = Vector2.ZERO
    animation_player.play("Attack")
else:
    var direction = (player.global_position -
        global_position).normalized()
    velocity = direction * speed
    move_and_slide()

```

ຍ້າຍສ່ວນນີ້ໄປອູ່ຕ່ອກ້າຍ move_and_slide()

```

if direction.x != 0:
    sprite.flip_h = direction.x < 0
    animation_player.play("Walk")

```

```

19 func _physics_process(delta: float) -> void:
20     if not player or is_dead:
21         return
22
23     var distance = global_position.distance_to(player.global_position)
24     if distance < 1.0:
25         velocity = Vector2.ZERO
26         animation_player.play("Attack")
27     else:
28         var direction = (player.global_position - global_position).normalized()
29         velocity = direction * speed
30         move_and_slide()
31
32     if direction.x != 0:
33         sprite.flip_h = direction.x < 0
34
35     animation_player.play("Walk")

```

```

func _on_area_2d_area_entered(area: Area2D) -> void:
    if is_dead:
        return
    if area.name == "HitBox":
        hit_count += 1
        show_hit_effect()
    # กระเด็นด้วยหลังจากตี Harding player
    var push_dir = (global_position -
                    player.global_position).normalized()
    global_position += push_dir * knockback_distance
    if hit_count >= 2:
        die()

```

เอาไว้ตรวจสอบ การเยื่อง และลำดับของข้อมูลคำสั่ง

ต่อไปเป็นการทำงานของพังก์ชัน แต่ละส่วนเริ่มต้นที่

func _on_HitBox_area_entered(area: Area2D)

พังก์ชันนี้จะถูกเรียกโดยอัตโนมัติเมื่อ Enemy ถูกชนโดย Area2D ของ Player (เช่น HitBox)

- ถ้าคัตตูรพยายามแล้ว ไม่ต้องทำอะไรเลย (ป้องกันซ้ำ)
- ตรวจสอบชื่อของ Area2D กี่ชนว่าชื่อว่า "HitBox" หรือไม่
- เพิ่มจำนวนครั้งที่โดนโจมตี แล้วเรียกพังก์ชัน show_hit_effect() เพื่อแสดงสีแดงชี้ว่าถูกโจมตี
- คำนวณทิศทางกระเด็นด้วยหลังจากจุดชนกับ Player และดันตัว Harding ออกไปตาม knockback_distance
- ถ้าโดนครับ 2 ครั้ง ให้เรียกพังก์ชัน die() เพื่อเข้าสู่กระบวนการตาย

```

37 func _on_HitBox_area_entered(area: Area2D) -> void:
38     if is_dead:
39         return
40
41     if area.name == "HitBox":
42         hit_count += 1
43         show_hit_effect()
44
45         # กระเด็นออกจากตัว主角 player
46         var push_dir = (global_position - player.global_position).normalized()
47         global_position += push_dir * knockback_distance
48
49     if hit_count >= 2:
50         die()

```

เอาไว้ตรวจสอบ การเยื่อง และลำดับของข้อมูลคำสั่ง

ต่อไปเป็นการทำงานของฟังก์ชัน:

func show_hit_effect()

- แสดงผลกราฟิกของการโดนโจมตีด้วยการเปลี่ยนสีศัตรู เป็นสีแดงชี้ว่าคราว
- เปลี่ยนสีของ Sprite เป็นแดง (Red Tint)
- รอ 0.15 วินาที เปลี่ยนกลับเป็นสีปกติ (ขาว)

func show_hit_effect():

```

sprite.modulate = Color(1, 0.2, 0.2) # สีแดง
await get_tree().create_timer(0.15).timeout
sprite.modulate = Color(1, 1, 1)    # คืนสีเดิม

```

ต่อมาให้ปรับฟังก์ชัน **func die()** จัดการลำดับการทำงานของศัตรู

- ตั้งสถานะ is_dead เป็นจริง และหยุดเคลื่อนไหว gdscript คัดลอกแก้ไข
- เล่นアニメชัน "Death"
- เริ่มจับเวลา 0.8 วินาที ก่อนเข้าสู่การกระพริบ

func die():

```

is_dead = true
velocity = Vector2.ZERO
animation_player.play("Death")
collisionShape.disabled = true
collisionShapeArea.disabled = true
if is_in_group("Enemy"):
    remove_from_group("Enemy")
death_timer.start(0.8)

```

ส่วนต่อไป พังก์ชัน **func _on_DeathTimer_timeout()**

ถูกเรียกเมื่อ death_timer หมดเวลา → เริ่มการกระพริบ

- สั่งให้เริ่มกระพริบ Sprite ทุก 0.1 วินาที

```
func _on_DeathTimer_timeout():
```

```
    blink_timer.start(0.1) # เริ่มกระพริบ
```

พังก์ชัน **func _on_BlinkTimer_timeout()** ถูกเรียกทุกครั้งที่ blink_timer หมดเวลา → ทำการกระพริบ และลบตัวเลขเมื่อครบ 5 ครั้ง

```
func _on_BlinkTimer_timeout():
```

```
    sprite.visible = not sprite.visible
```

```
    blink_count += 1
```

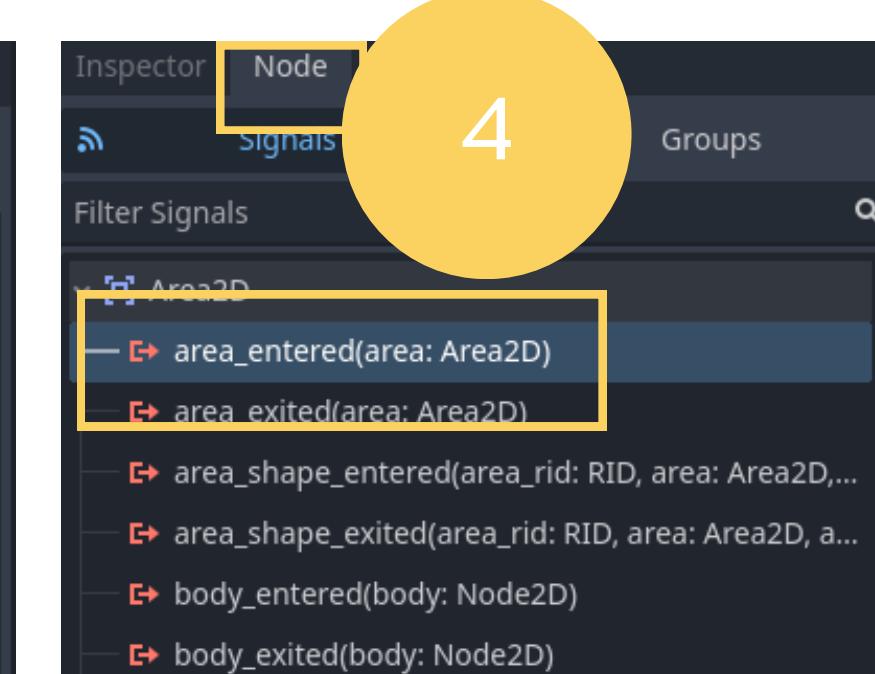
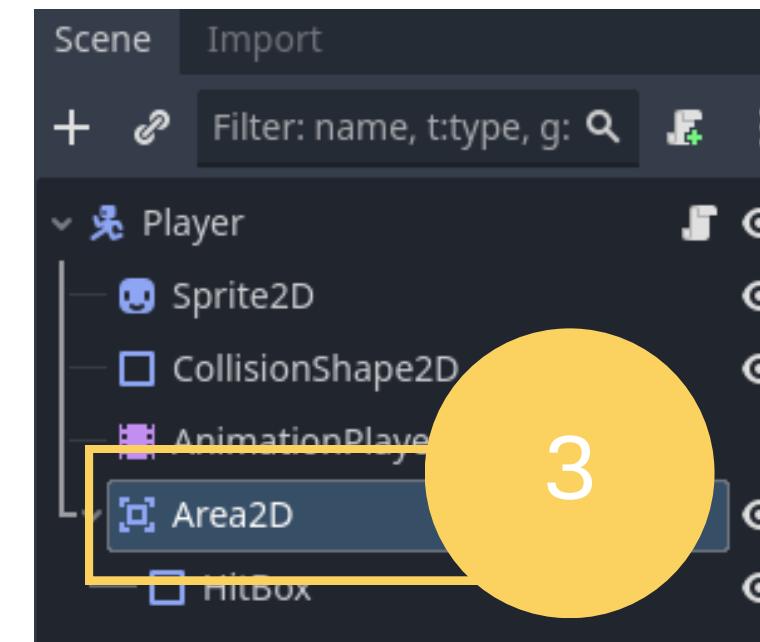
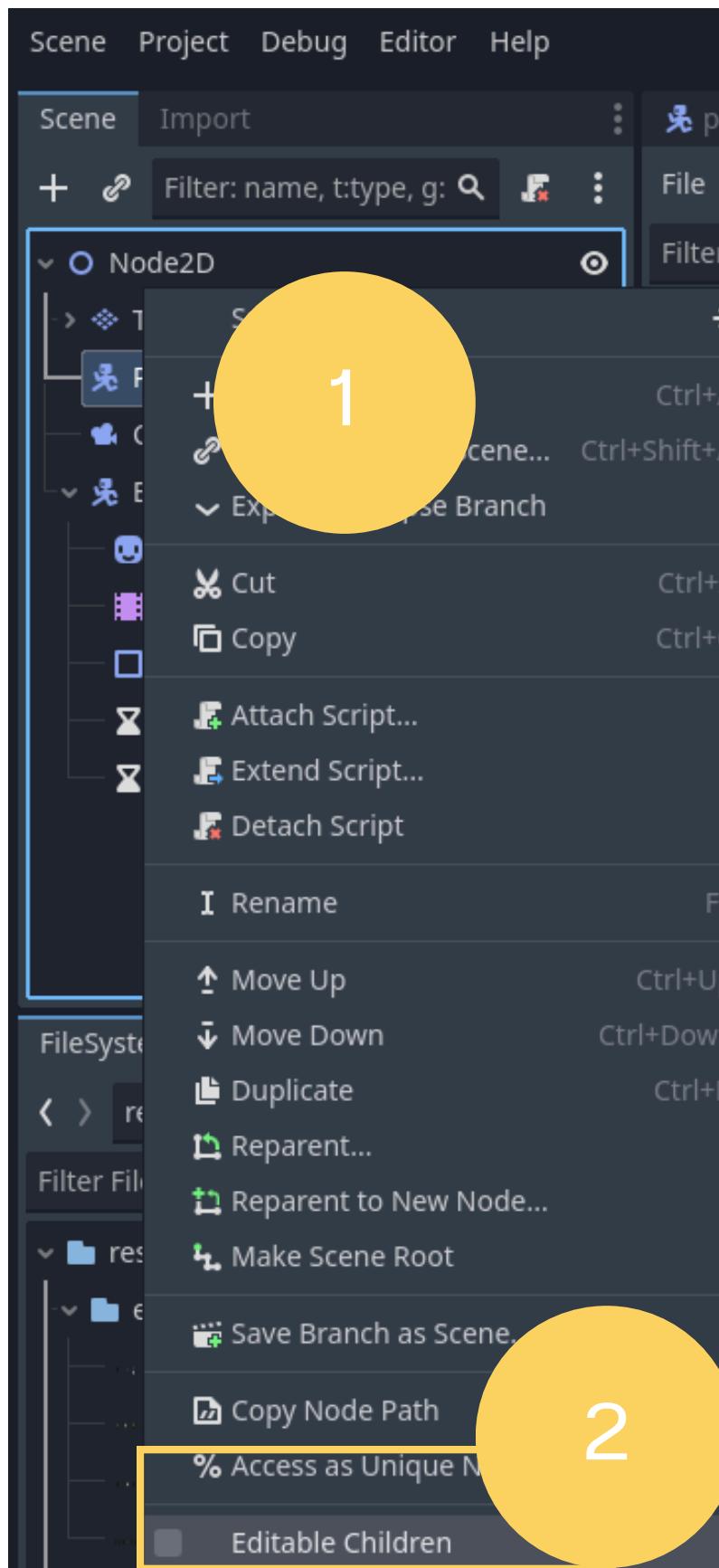
```
    if blink_count >= 5: # กระพริบครบ 0.5 วินาที (5 ครั้ง × 0.1s)
```

```
        queue_free()
```

- สลับการแสดงผล Sprite → กระพริบ

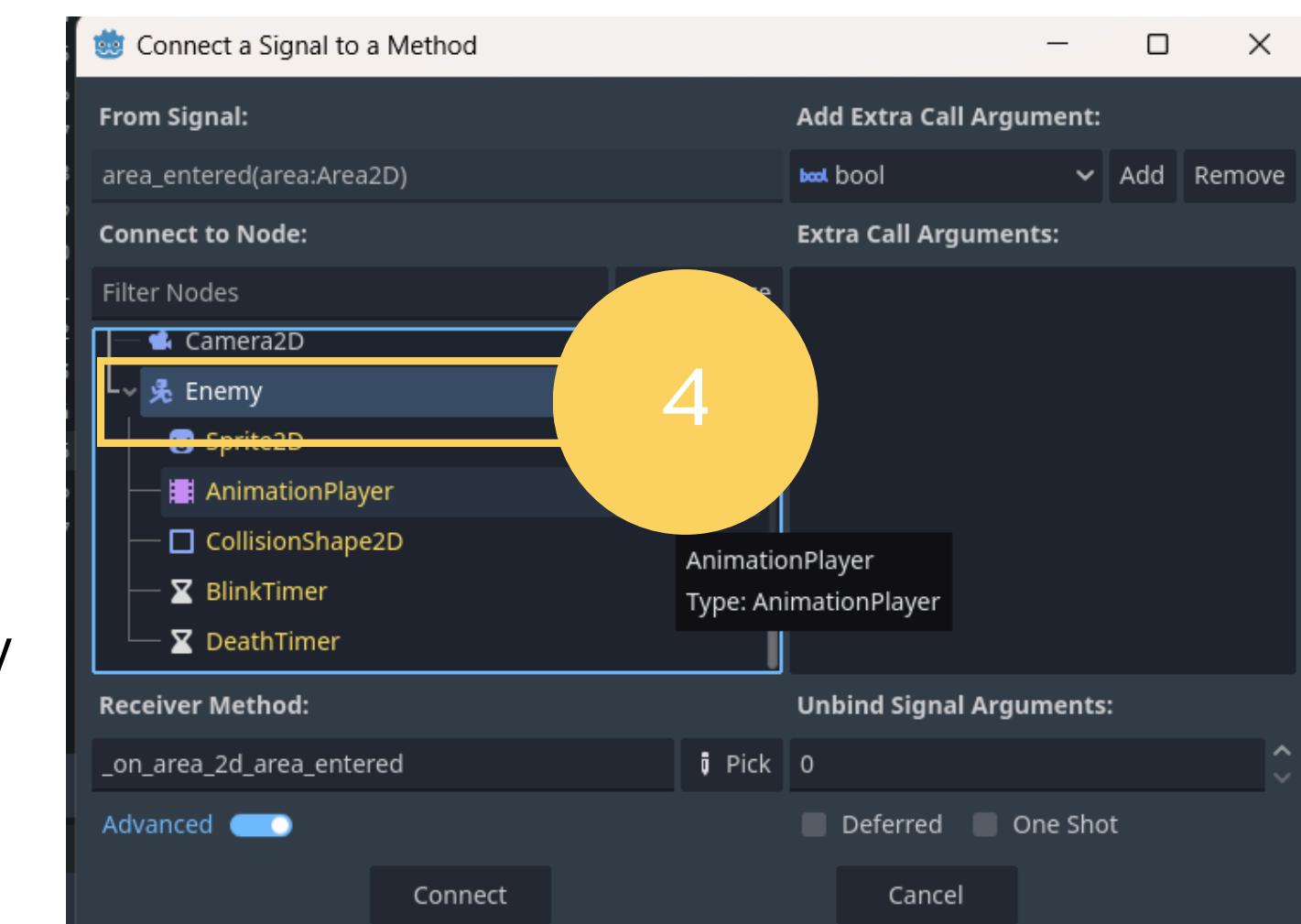
- ด้ากระพริบครบ 5 ครั้ง (0.5 วินาที) → ลบตัวเลขออก
จากจากด้วย queue_free()

```
52     func show_hit_effect():
53         sprite.modulate = Color(1, 0.2, 0.2) # สีแดง
54         await get_tree().create_timer(0.15).timeout
55         sprite.modulate = Color(1, 1, 1) # คืนสีเดิม
56
57     func die():
58         is_dead = true
59         velocity = Vector2.ZERO
60         animation_player.play("Death")
61         death_timer.start(0.8)
62
63     func _on_DeathTimer_timeout():
64         blink_timer.start(0.1) # เริ่มกระพริบ
65
66     func _on_BlinkTimer_timeout():
67         sprite.visible = not sprite.visible
68         blink_count += 1
69
70     if blink_count >= 5: # กระพริบครบ 0.5 วินาที (5 ครั้ง × 0.1s)
71         queue_free()
```

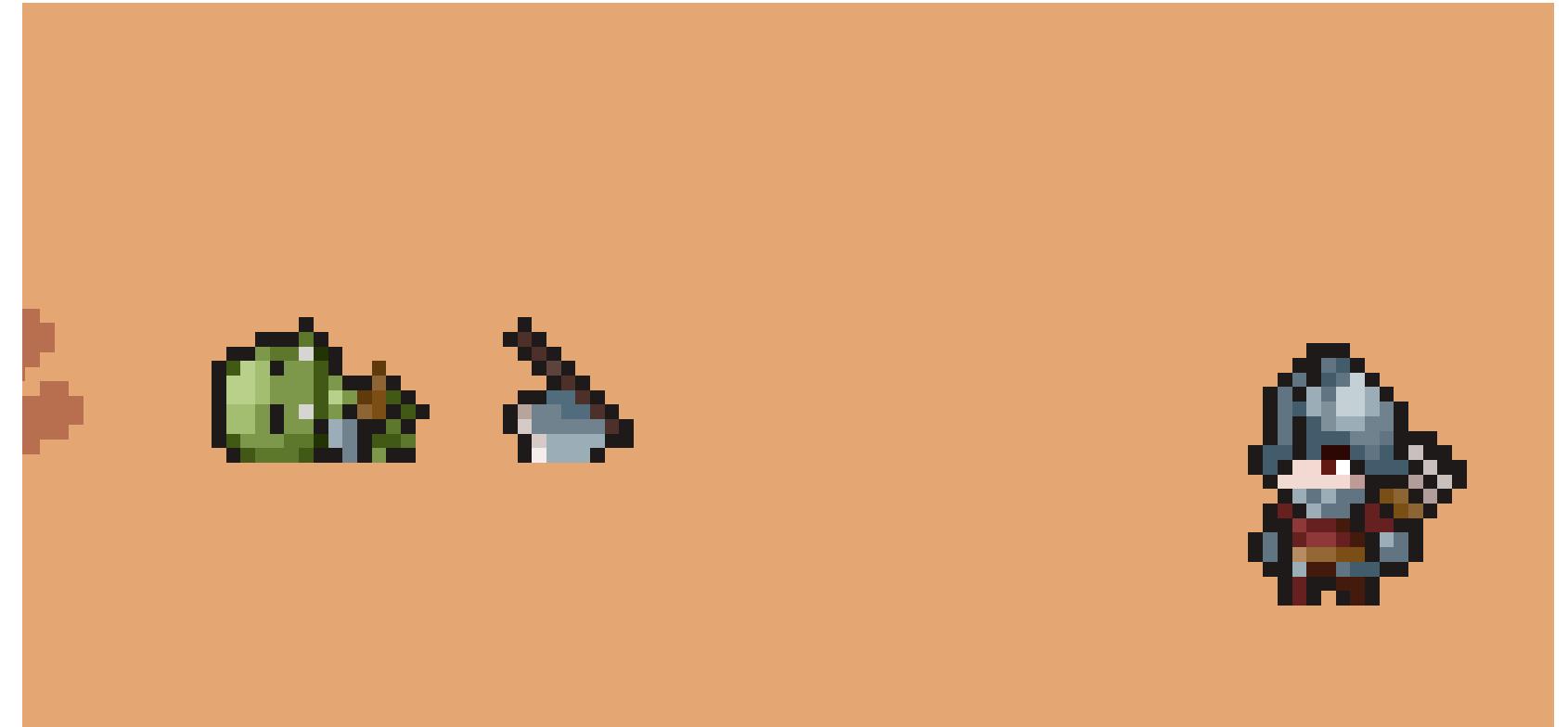
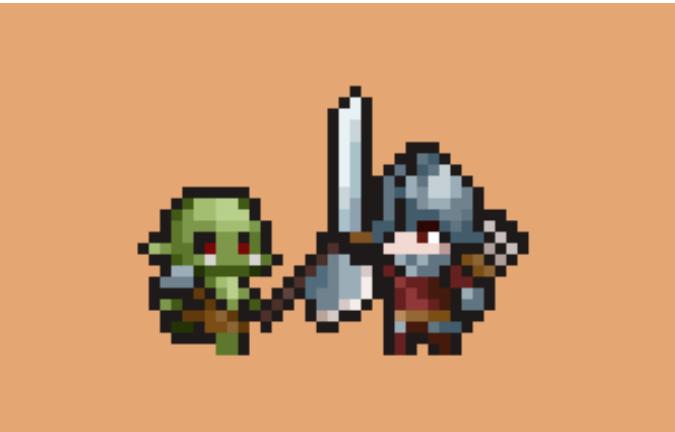


สำหรับการตรวจจับศัตรูโดย HitBox:
ให้ไปที่ **game.tscn** หรือ Scene แล้ว คลิกขวาที่
Player และ Enemy เลือก **Editable Children**
หลังจากนั้นคลิกที่ **Area2D** ของ Player
ผู้เล่น Player ต้องมี:

- คลิกแท็บ **Node** ใน Inspector
- Area2D สำหรับ HitBox เป็น CollisionShape2D ภายใน
- ต้องแนบ signal **area_entered** ไปยัง Enemy



OUTCOME



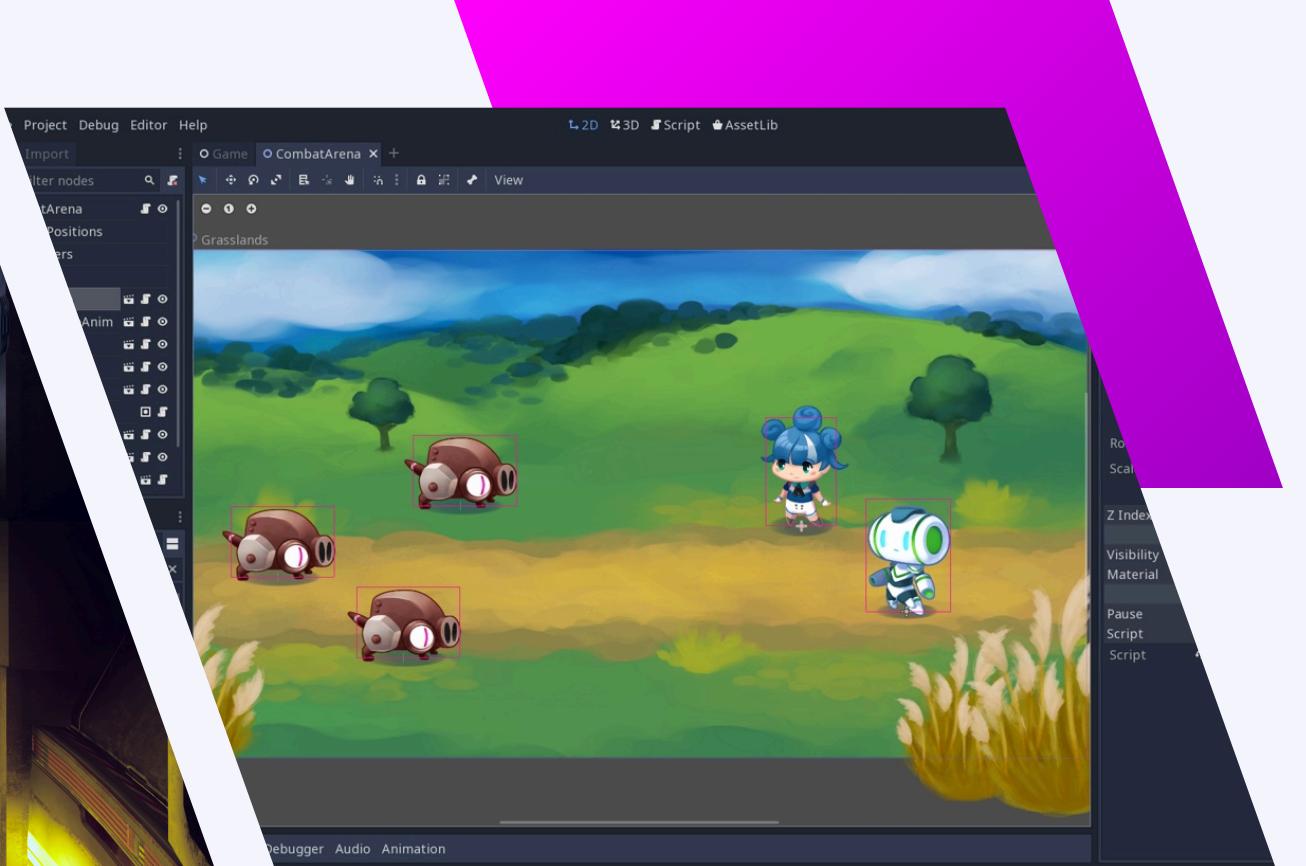
เมื่อคัตติวูปราภูมิจากแล้วตัวละครโจมตี คัตติวู
ตัวคัตติวูจะกระเด็นออกห่างจากตัวละคร และแสดงแอนิเมชัน
“Death” ไม่เคลื่อนที่

Source code: สามารถดาวน์โหลดได้ที่

<https://github.com/banyapon/antdpugodotcourse/tree/main/chapter3>

GAME DEVELOPER

College of Creative Design and Entertainment Technology



การพัฒนาเกมด้วยโปรแกรม Godot Engine