Game Programming 2 – Week 2 lab

Target practice

Goals:

- Practice using Godot interface
- Practice creating scenes and nodes
- Practice using collision bodies and simple physics
- Practice using rotation and mouse position

Top down character

Steps:

1. Create a new scene. The root node shall be a CharacterBody2D. Call it “Scout”, save it as scout.tscn
2. Download the asset pack called “TopDownShooter” in Teams
3. Add a Sprite2D to the scout. Use the **player.png** found in the player folder for the image.
4. Add a CollisionPolygon2D to create a collision shape around the player
5. Add a script to the root node. Use the basicMovement template provided. Modify the initial to:
    a. Remove the gravity completely
    b. Adjust the movement so that the four inputs (up, down, left, right) can move the character in four directions. (This should be very similar to what we did with the butterfly in class)

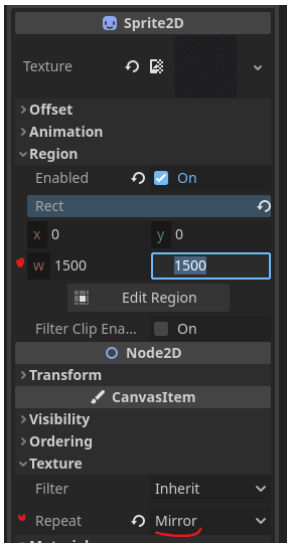
Rotation

1. We will rotate the player to look at the camera. In your player script, inside the _process method, add this line:

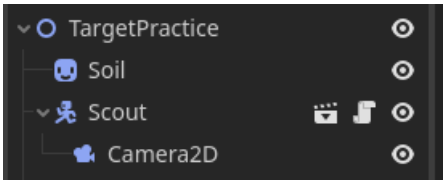
```
# rotate
look_at(get_global_mouse_position())
```

2. For this to work properly with our sprite, the sprite needs to face right.
    a. Add a 90 degrees rotation on the sprite2D node
    b. Add a 90 degrees rotation on the collision polygon node

Target practice Scene



1. Create a new scene. Use Node2D as root. Call it “TargetPractice” and save it.
2. Add a Sprite2D node, call it “Soil”
    a. Add the sprite “dirt.png” in the texture, found in the ground folder
    b. *We will create a large area background to play around.*
    Find the region section of the Sprite2D node. Enable it. Expand it and assign 1500px in the **w** and **h** field. This will create a large section, but the sprite will be distorted outside of the image range.
    c. To fix the distortion, find the texture section and apply the “mirror” technique in the *repeat* field.
    d. Keep the sprite position centered on (0,0)
3. Add the scout player as an instance scene to the root of TargetPractice.
4. Add a Camera2D node as a child of the scout node.

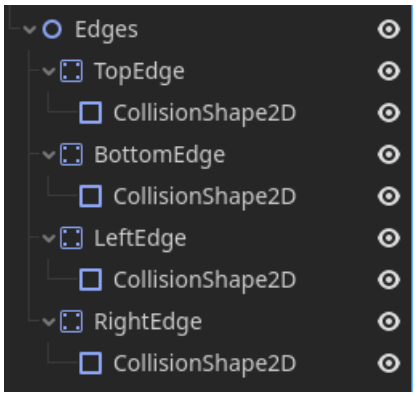


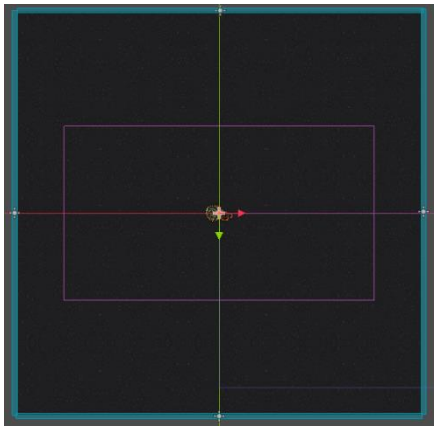
Now test your character and map. The player should be able to move around and look at the mouse position.
The character should be able to see the background edges and behind. We will fix that next.

Camera limits and map edges

1. On the camera2D node, spot the limit property in the inspector.
    a. Assign proper limits to the camera. They should match the limit of the background image. *Figure out the limits value yourself*.
    Hint: Remember, the background sprite is **1500px** wide and tall and is centered.
2. Add a Node2D, child of the root node. Call it *Edges*.
3. Add one StaticBody2D node, child to Edges. Call it “TopEdge”.
    a. Add a CollisionShape2D, child to “TopEdge”. Add a rectangle shape (or you can try a *WorldBoundary* shape) and move it to the top of the background.

4. Repeat step 3 for the three other sides to create edges that the character will not be able to cross. Result should look like this:



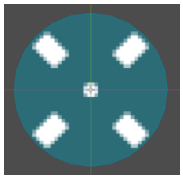


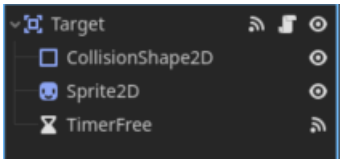
Now test your character and map. The player should be able to move around and look at the mouse position and be confined to the play area.

Creating targets to reach

We will now create some target that the player must get to. When the player reaches the target, we will destroy it after one second and try to spawn a new one elsewhere.

1. Create a new scene.
    a. Name: Target
    b. Root node: Area2D
2. Add a Sprite2D, use the crosshair texture, found in the UI folder.
3. Add a collision shape (circle) to match the shape of the sprite
4. Add a Timer node, child of the root
    a. Name: TimerFree
    b. Duration: 1s
    c. One shot
5. Add a script to the root node





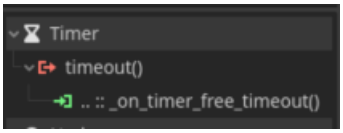
We want the target to be destroyed, when the player reaches it, so we will use signals.

6. Select the root node and add a signal method for “on_body_entered” event. Link it with our main script.
    a. We will start the timer and modify the color to show that the target was reached. Add this code:

```
func _on_body_entered(body):
    $Sprite2D.self_modulate = Color(1,1,1,.5)
    $TimerFree.start()
```

7. Now, on the timer, se will use the *timeout* signal. On the timer node, add a signal for the timeout in our main script:

```
func _on_timer_free_timeout():
    queue_free()
```



8. Save your scene. For testing purposes, add a target instance in the "target_practice" scene and see if the player can make it disappear by reaching it.

Now test your character and map. The player should be able to move around and look at the mouse position and be able to make targets disappear.

Generating new targets

Back in the "target_practice" scene, we will generate new targets when one is reached.

We will have to create a custom signal, that we will name "spawn_new_target"

Inside the "target" script, add a new custom signal and fire on destruction:

1. Inside target.gd, declare a new signal at the top:

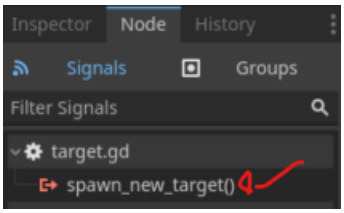
```
1   extends Area2D
2
3   signal spawn_new_target
```

2. When the timer runs out, emit the new signal:

```
func _on_timer_free_timeout():
    spawn_new_target.emit()
    queue_free()
```

Back in the *target_practice* scene, make sure that you placed one instance of the target in the tree, if you have not already.

3. Create a new script attached to the root node TargetPractice.
4. By selecting the Target node, spot the new custom signal that we created and bind it to a callback method in the new script.




```
func _on_target_spawn_new_target():
    print("signal received")
```

5. Let's create a new target instance and place it in a random position in the map. Add this code to the callback method:

```gdscript
var target_scene = preload("res://Scenes/target.tscn")

func _on_target_spawn_new_target():
    print("signal received")

    # Create a new target
    var new_target = target_scene.instantiate() as Area2D

    # Assign a random position in the map
    var random_position = Vector2(randf_range(-750,750),randf_range(-750,750))
    new_target.position = random_position

    # Add the node to the scene tree
    $".".add_child(new_target)
```

Now test your character and map. The player should be able to move around and be able to make the first target disappear, but a second one should be created in a random position.

Try to reach the second target, you might notice that a third target never appears.

=> This is because our first target is connected to the callback function, but the second target is created at runtime and is not connected in the same manner, so the new signal is not listened to.

6. We can finalize our script by connecting our new instance of target to the callback method using code. Add this line

```gdscript
new_target.spawn_new_target.connect(_on_target_spawn_new_target)
```

The final method should look like this:

```gdscript
func _on_target_spawn_new_target():
    print("signal received")

    # Create a new target
    var new_target = target_scene.instantiate() as Area2D
    new_target.spawn_new_target.connect(_on_target_spawn_new_target)

    # Assign a random position in the map
    var random_position = Vector2(randf_range(-750,750),randf_range(-750,750))
    new_target.position = random_position

    # Add the node to the scene tree
    $".".add_child(new_target)
```

Test it. It should generate new targets infinitely now.

You have completed your second lab. Congratulations.