A little RPG Battle System

Gaëtan Blaise-Cazalet - Responsable de la section programmation

A small map

- Create a 2d node Map
- Save the scene into Map.tscn
- Create a TileMap in the map

We need a tileset to fill the map

A tileset

- Create a new scene, create a node2d called Tileset
- Create five sprite as Tileset children
- Create fixe 64 * 64 images figuring Grass, Sand, Path, Wather and Rock. Rock will block the player move.
- Change the snap settings 64 * 64, activate snapping
- Add the textures to the sprites, untick the centered parameter in the Inspector / Offset and place the sprites near each other

We want to add collisions to the Rock tile :

- Set the snap grid to 32 * 32
- Add a StaticBody2D as a child of Rock sprite
- Add a CollisionShape2D as a chlid of this last node
- Create a rectangle shape in the inspector
- Set the right size using the snap grid

Same thing for the water.

Tile drawing

To Export the tileset :

- Use Scene / Convert to / Tileset, call the file tileset.tres
- Save the scene to keep it

Go back to your map scene.

- In your map, go to the Tilemap inspector
- Load the tileset.tres file you created
- Paint your background on the map
- Save your map

Map Hero

Create the hero scene :

- Create a scene, add a KinematicBody2D inside, add a CollisionShape2D and set it up
- Add also an Area2d that will be used to detect enemy collision
- Add a sprite and create the image for the hero

Your hero needs a classic script for its move logic :

```
extends KinematicBody2D

export (int) var speed = 500

var velocity = Vector2()

func get_input():
    velocity = Vector2()
    if Input.is_action_pressed('ui_right'):
        velocity.x += speed
    if Input.is_action_pressed('ui_left'):
        velocity.x -= speed
    if Input.is_action_pressed('ui_down'):
        velocity.y += speed
    if Input.is_action_pressed('ui_up'):
        velocity.y -= speed
    velocity = velocity.normalized() * speed

func _physics_process(delta):
    get_input()
    move_and_slide(velocity)
```

Map Enemy

Create the map enemy scene :

- Create a scene, add a Area2D inside, add a CollisionShape2D and set it up
- Add a sprite and create the image for the enemy
- The collision shape must be bigger than the sprite

Fight Scene

Create a scene, add a Node2D, name it Fight and save it :

- Add a sprite for the Enemy and a sprite for the Hero. Call them "Enemy" and "Hero".
- Add and item list. Use the central menu to Add a "Attack" and a "Defense" element in the item list.

Add a Battle script to the Fight node.
The battle will be divided with phases : player phase, enemy phase, eventual victory or loss.

Fight Scene

The code structure will be :

```
enum BattlePhase {
	PLAYER_PHASE, ENEMY_PHASE, VICTORY, LOSS
}

var phase

func _ready():
	phase = BattlePhase.PLAYER_PHASE
	$ItemList.visible = true

func _process(delta):
	match(phase):
		BattlePhase.PLAYER_PHASE:
			update_player_phase(delta)
		BattlePhase.ENEMY_PHASE:
			update_enemy_phase(delta)
		BattlePhase.VICTORY:
			update_victory(delta)
		BattlePhase.LOSS:
			update_loss(delta)

func update_player_phase(delta):
	pass

func update_enemy_phase(delta):
	pass

func update_victory(delta):
	pass

func update_loss(delta):
	pass
```

Fight Scene

Now we want to choose an action for the player. We will use the yield function to stop a function until the player's action is chosen.

```
func update_player_phase(delta):
	var action = yield($ItemList, "item_selected")
	$ItemList.visible = false
	if(action == 0):
		print("attack")
	else:
		print("defense")
	phase = BattlePhase.ENEMY_PHASE
```

Fight Animations

We will launch a serie of animation when the player choose attack.

First, we need to add an AnimationPlayer to our scene.

Then select the Hero in the hierarchy. Add a new animation called "hero_move_forward".

Add a starting key using the middle key button. Move the animation cursor then move the sprite to its end position. Add a key. You can test your animation.

Create two animations for "hero_hit" and "hero_move_backward".

```
func update_player_phase(delta):
	var action = yield($ItemList, "item_selected")
	$ItemList.visible = false
	if(action == 0):
		$AnimationPlayer.play("hero_move_forward")
		yield($AnimationPlayer, "animation_finished")
		$AnimationPlayer.play("hero_hit")
		yield($AnimationPlayer, "animation_finished")
		$AnimationPlayer.play("hero_move_backward")
		yield($AnimationPlayer, "animation_finished")
	else:
		print("defense")
	phase = BattlePhase.ENEMY_PHASE
```

Fight Animations

Now, make the same for the enemy turn.

At the end of the enemy turn, don't forget to make the action ItemList visible and to get back to PLAYER_PHASE.

Victory and Defeat panels

Add a VictoryPanel and a DefeatPanel, with a Label and a Button inside.

Make them invisible at the start of the scene by modifying _ready() :

```
func _ready():
	$VictoryPanel.visible = false
	$LossPanel.visible = false
	phase = BattlePhase.PLAYER_PHASE
	$ItemList.visible = true
```

Hurting the enemy

We will add four new variables in our fighting system :

- one for the player hp
- one for the enemy hp.
- one for the player damages
- one for the enemy damages

Now, we can update the hp when the battler hits its enemy. If the enemy has 0 hp, the player wins. And vice versa. Here is the code for the player.

```
func update_player_phase(delta):
    ...
    if(action == 0):
        ...
        $AnimationPlayer.play("hero_hit")
        yield($AnimationPlayer, "animation_finished")
        enemy_hp = enemy_hp - player_damage
        if(enemy_hp <= 0):
            $Enemy.visible = false
        $AnimationPlayer.play("hero_move_backward")
        yield($AnimationPlayer, "animation_finished")
        if(enemy_hp <= 0):
            phase = BattlePhase.VICTORY
        else:
            phase = BattlePhase.ENEMY_PHASE
```

You can add 1 hp to the player when it defends itself.

Hurting the enemy - Bug

Our code is quite simple, but it has a bug. The player will always kill the enemy in one shot, even if the damage should no suffice.

We can use the debugger to see what happens.

A simple fix would be so set an "execution" boolean that would prevent the action to be executed more than once.

We have a second bug : the ItemSelect keep the same item selected. We have to deselect it.

```
func update_player_phase(delta):
	var action = yield($ItemList, "item_selected")
	$ItemList.visible = false
	if(action == 0):
		$AnimationPlayer.play("hero_move_forward")
		yield($AnimationPlayer, "animation_finished")
		$AnimationPlayer.play("hero_hit")
		yield($AnimationPlayer, "animation_finished")
		if not action_executed:
			action_executed = true
			enemy_hp = enemy_hp - player_damage
			if(enemy_hp <= 0):
				$Enemy.visible = false
		$AnimationPlayer.play("hero_move_backward")
		yield($AnimationPlayer, "animation_finished")
		$ItemList.unselect(0)
		if(enemy_hp <= 0):
			phase = BattlePhase.VICTORY
		else:
			action_executed = false
			phase = BattlePhase.ENEMY_PHASE
	else:
		player_hp = player_hp + 1
		$ItemList.unselect(1)
		phase = BattlePhase.ENEMY_PHASE
```

Note for future programmers :

This code is note a "good" code from the software engineering point of view. A good code would abstract the notion of action, the action queue , the phases etc. But good code is not required to make a prototype. When you prototype, you want to go fast.

If you want to extend your game after your prototype, you will have to refactor your code to make it "good", or at least "good enough".

Victory or defeat announcement

Now, we just have to set the game behaviour when we are in VICTORY or in LOSS state.

For our basic system, we just have to wait for an action on the panel button.

Code for VICTORY state :

```
func update_victory(delta):
	$VictoryPanel.visible = true
	yield($VictoryPanel/Button, "pressed")
	get_tree().change_scene("res://Map.tscn")
```

Write the code for the LOSS state. Create a new Game Over scene and switch to hit when player_hp is inferior or equal to zero.

Map to battle transition

Add an Area2D on the hero.

Now, on the Map scene, when the player's Area2d collides the enemy Area2D, trigger the battle scene.

Effect of the battle on the map 1/3

We want the enemy to disappear after it is defeated.

We will use an Autoload, which is a node loaded automatically by the game, to store the state of the map before the battle, and reload it after the battle, with the consequences of the battle.

Go to the script editor. File / New and create a "global.gd" script

Go to Project / Preferences / Autoload. Browse to the file you created with the ".." button. Give it the "global" name.

From now, we can call the autoloaded script from any other script. We will add in it a function to save the current scene and switch to an other, and an other function to load the previous scene.

Effect of the battle on the map 2/3

Add this script in global.gd:

```
var saved_scene = null

func switch_scene(temp_scene):
    # save scene and remove it from the tree
    saved_scene = get_tree().get_current_scene()
    get_tree().get_root().remove_child(saved_scene)
    # instance and add temporary scene as current scene
    var new_scene = load(temp_scene).instance()
    get_tree().get_root().add_child(new_scene)
    get_tree().set_current_scene(new_scene)

func load_scene():
    if saved_scene != null:
        # free temporary scene
        get_tree().get_current_scene().queue_free()
        # add saved scene back to the tree
        get_tree().get_root().add_child(saved_scene)
        saved_scene = null
```

Effect of the battle on the map 3/3

Create a new script in the MapEnemy. Connect the area_entered event to this script. Delete the enemy and save/switch to fight scene with this script.

```
func _on_MapEnemy_area_entered(area):
	queue_free()
	global.switch_scene("res://Fight.tscn")
```

Change the battle end script to get back to map scene :

```
var map_loaded = false

func update_victory(delta):
	$VictoryPanel.visible = true
	yield($VictoryPanel/Button, "pressed")
	if not map_loaded:
		map_loaded = true
		global.load_scene()
```