Autores: José Serrano Flores, Pau Duque Martínez, Íñigo Jiménez Díaz.

Curso Académico: Sistemas Microinformáticos y Redes.

Grupo: 2 SMX

Resum del projecte (màxim 250 paraules):

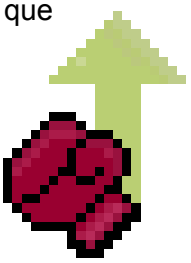
Nuestro proyecto de síntesis consiste en la creación de un videojuego RPG desarrollado con el motor Godot, en colaboración con dos compañeros. Entre los tres, hemos diseñado y programado el juego desde cero, encargándonos tanto de la parte visual como de la lógica interna.

Uno de los aspectos más destacados del proyecto es que todos los **sprites** han sido creados por nosotros. Diseñamos los personajes, enemigos, escenarios y elementos del entorno en estilo pixel art, buscando darle al juego una identidad visual coherente y original. Esto nos permitió trabajar el apartado artístico y entender la importancia del diseño visual en la experiencia del jugador.

Además, programamos todos los **scripts** utilizando GDScript, el lenguaje propio de Godot. Implementamos funciones como combates a tiempo real, pociones de salud, y subida de nivel. Cada línea de código fue escrita por nosotros, permitiéndonos adaptar completamente el juego a nuestra idea.

Este proyecto nos ayudó a poner en práctica los conocimientos adquiridos durante el curso, trabajar en equipo y enfrentarnos a los retos reales del desarrollo de videojuegos. El resultado es un RPG jugable, original y completamente desarrollado por nosotros, que refleja nuestro esfuerzo y creatividad.

Paraules clau (entre 4 i 8):
Videojoc, RPG, Godot, Sprites, GDScript, Programació, Pixel art, Treball en equip



Resum en anglès

Our synthesis project consists of creating an RPG video game developed with the Godot engine, in collaboration with two classmates. Together, we designed and programmed the game from scratch, taking care of both the visual and internal logic aspects.

One of the highlights of the project is that all the sprites were created by us. We designed the characters, enemies, environments, and elements of the setting in pixel art style, aiming to give the game a coherent and original visual identity. This allowed us to work on the artistic aspect and understand the importance of visual design in the player's experience.

Additionally, we programmed all the scripts using GDScript, Godot's native language. We implemented features such as real-time combat, health potions, and level-ups. Every line of code was written by us, allowing us to fully tailor the game to our vision.

This project helped us put into practice the knowledge gained during the course, work as a team, and face the real challenges of game development. The result is a playable, original RPG, entirely developed by us, reflecting our effort and creativity.

Índex:

1. Introducción

1.1 Contexto y justificación del trabajo.

Teniendo en el grupo como pasatiempo jugar videojuegos, desarrollar un proyecto como este nos ha permitido adentrarnos en una disciplina que fusiona nuestra motivación y ganas de aprender. El objetivo principal fue crear un videojuego RPG funcional usando el motor Godot, desarrollando desde cero tanto su aspecto visual como su programación.

Elegir el género RPG no fue una decisión aleatoria; este tipo de juego nos ha brindado la oportunidad de explorar mecánicas como el combate en tiempo real y la gestión de recursos. Además, trabajar con Godot nos ha permitido emplear herramientas profesionales que ganan cada vez más terreno en la industria independiente.

Este proyecto lo justificamos como una forma práctica y creativa de aplicar los conocimientos adquiridos durante nuestra formación, especialmente en áreas clave como la programación y el trabajo en equipo. Crear desde cero todos los sprites y scripts nos ha obligado a resolver problemas reales de desarrollo, lo que ha mejorado nuestras habilidades y nos ha preparado para futuros desafíos en el ámbito profesional.

1.2 Objetivos

- Jugable
- Variedad mapas y enemigos
- Comprender el lenguajes de script
- Usar una plataforma de código libre y que no sea tan intuitiva.

1.3 Estrategia y planificación del proyecto.

Desde el principio supimos que para sacar adelante un proyecto como este necesitábamos organizarnos bien y tener claro qué pasos seguir. Seguimos estas fases.

Fase 1 Idear el juego
Nos juntamos a compartir ideas y acordamos el estilo de juego, qué queríamos que pasara en la historia, cómo se iba a ver el mundo y qué mecánicas nos gustaría incluir. Esta parte fue muy creativa y nos ayudó a estar todos en la mismo conjunto desde el inicio.

Fase 2 Tareas
Cada uno se encargó de lo que más le gustaba o en lo que tenía más experiencia: uno se centró más en godot, otro en los sprites y el otro se dedicó en ambas cosas. De esta manera, pudimos trabajar rápidamente.

Fase 3 Desarrollar el juego
Nos pareció más fácil trabajar por partes, como por ejemplo el sistema de combate, el movimiento del personaje,objetos consumible

Nos aseguramos de que todo funcionara bien junto, ajustamos el equilibrio del juego, corregimos bugs y mejoramos lo que no terminaba de encajar. Fue una fase de mucha prueba y error, pero muy útil para pulir los detalles.

Fase 6 Documentar
Documentamos todo el proceso, hicimos capturas del juego, y dejamos bien explicado todo lo que hicimos y aprendimos. También hablamos sobre qué nos gustaría seguir mejorando en el futuro.

En resumen, nuestra estrategia fue simple pero efectiva: buena comunicación, trabajo en equipo, y muchas ganas de aprender y disfrutar el proceso.

1.4 Metodología del trabajo.

Desde el principio tuvimos claro que, para que esto funcionara, teníamos que organizarnos bien, pero sin volvernos locos. No usamos una metodología súper estricta, más bien armamos una forma de trabajo que nos sirviera a nosotros: práctica, flexible y con buena comunicación.

Lo que hicimos fue dividir el proyecto en partes pequeñas y ponernos metas. Cada vez que completabamos una meta deciamos: Que hacemos ahora. Si algo salía mal o no ibamos a poder implementarlo, no pasaba nada, lo cambiábamos sobre la marcha.

1.5 Enlace al proyecto.

https://github.com/Spino77/Vorkuta

Este es el enlace de nuestro proyecto en github, aquí creamos un “árbol” con cada versión diferente, la más nueva es la 0.20.

https://drive.google.com/file/d/1f-6Fzxh1nsL4hyStBvOVEc4qfVyyIbvP/view?usp=sharing

Y este es el enlace del ejecutable del juego.

2. Descripción del proyecto

El proyecto consiste en la creación de un videojuego de tipo RPG (Role-Playing Game) desarrollado con el motor Godot, utilizando el lenguaje de programación GDScript.
Ha sido diseñado e implementado por nosotros, también hemos trabajado conjuntamente en la programación, el apartado artístico (pixel art) y el diseño de las mecánicas del juego, con el objetivo de aplicar los conocimientos aprendidos durante el ciclo de sistemas microinformáticos y redes.

2.1 Análisis de requisitos.

- **Funcionalidades básicas :** El juego tiene que ser jugable con un sistema de combate a tiempo real.

- **Gestión de los elementos:** queremos meter un sistema de puntuación en base a los enemigos matados que se pueda usar como consumible para subirte atributos como salud y armas.

- **Area2D:** Hace que los enemigos sean capaces de detectar el jugador y moverse hacia él.

- **Interfície gráfica:** Indicaciones de las cosas que pasan entorno al juego (vida, puntuación y mejoras.)

- **Disseny visual:** Todos los sprites tenian que ser creados por nosotros con un estilo adecuado al juego en forma de pixel art.

- **Escalabilidad:** Posibilidad de añadir más funcionalidades en el futuro, como realidad virtual (VR) o la narrativa e historia.

2.2 Tecnologías

Tecnologías que usamos

Godot para desarrollar el juego.

GDScript Lenguaje de programación de Godot.

Google Drive para compartir cosas entre todos.

WhatsApp o Discord para hablar rápido y coordinarnos.

Github Para añadir la nueva versión del juego

Pixilart Herramienta que usamos para crear los sprites en pixel art.

2.3 Estructura del proyecto

El proyecto está organizado en varias escenas dentro de Godot, agrupadas en las siguientes categorías:

- **Jugador y equipamiento:** Jugador, arma, antorcha, scripts de movimiento del jugador y del ataque.
- **Enemigos:** Sombra, fuego fatuo y boss, con el script que se encarga de su movimiento, IA simple que sigue al jugador y daño.
- **Mapas:** Dos mapas con las colisiones, objetos y zonas de colisiones.
- **Spawner:** Escena que se encarga de generar enemigos cada cierto tiempo dentro del mapa.
- **Menus:** Pantalla de menú principal, de opciones, selección de mapa y la pantalla de muerte.
- **GlobalScript:** Script global que se encarga de gestionar variables y el estado de varias cosas del juego.

2.4 Definición de las funcionalidades

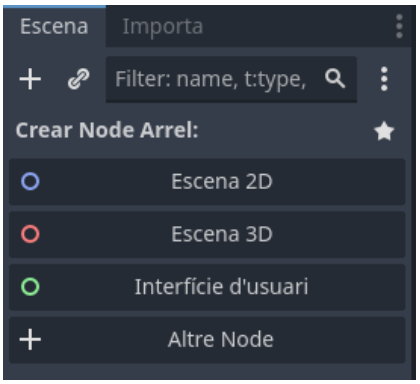
Las funcionalidades clave implementadas són:

- **Combate a tiempo real**, con animaciones y daño calculado.
- **Sistema de puntos y mejoras** (vida, velocidad, daño y cooldown).
- **Pociones de curación**.
- **Desbloqueo de mapa** y acceso al boss final con la puntuación.
- **Varios tipos de enemigos diferentes**.
- **Cambio de escenas** mediante menús.
- **Iluminación dinámica** con las antorchas y efectos de luz.
- **Escalabilidad de dificultad** según la puntuación del jugador.
- **Sistema de muerte** y reinicio del juego.

3. Godot

En este apartado explicaremos qué escenas forman nuestro juego y explicaremos cada una detalladamente y que nodos utilizan, también describiremos que hace todo el código que tienen algunos nodos de cada escena, que hacen que funcionen todas entre sí para formar el juego.

Escenas:



Godot funciona con escenas, hay varios tipos, pero las que se usan normalmente són:

- **Escena 2D**
- **Escena 3D**
- **Interfaz de usuario**

En nuestro caso, como el juego será en 2D, usaremos solo **Escenas 2D** para formar el juego en sí, e **Interfaz de usuario** para toda la interfaz (ex: vida, puntuación, menú principal, etc).

Las escenas que contiene nuestro proyecto són:

- **Mapas** (Escena 2D)
- **Jugador** (Escena 2D)
- **Boss** (Escena 2D)
- **Enemigos** (Escena 2D)
- **Arma** (Escena 2D)
- **Antorcha** (Escena 2D)
- **Spawner** (Escena 2D)
- **Menú principal** (Interfaz de usuario)
- **Opciones, elegir mapa, pantalla de muerte** (Interfaz de usuario)

Cada escena contiene diferentes nodos, cada nodo tiene una función en específico y algunos necesitan un script (GDScript) para funcionar de la manera que se quiere, explicaremos todo esto en los siguientes puntos.

Scripts:

Los scripts utilizan el lenguaje de programación propio de Godot llamado GDScript, para resumir su funcionamiento, cada script está adherido a un nodo, y usa funciones para realizar acciones, dentro del script solo se pueden referenciar otros nodos si forman parte de la misma escena, aunque hay maneras de referenciar nodos de otras escenas, como exportando o usando nombres únicos.

También se pueden realizar funciones del script usando entradas del usuario, esto se puede poner en la configuración del proyecto.

Ahora pondré un script simple para explicar su funcionamiento más detalladamente.

extends Label

#Esta línea declara que el script se hace en un nodo de tipo Label.

func _physics_process(_delta):

#La función “physics_process” se repite constantemente.

if Input.is_action_just_pressed("click"):

#Aquí define que cada vez que se pulse la acción llamada “click”, realice las siguientes líneas.

numero_epico()

#Esta es la línea que se realizará si se pulsa la acción “click”, simplemente llama a otra función que ya está creada en el mismo script.

func numero_epico()

#Esta es la función llamada cuando se pulsa la acción “click”.

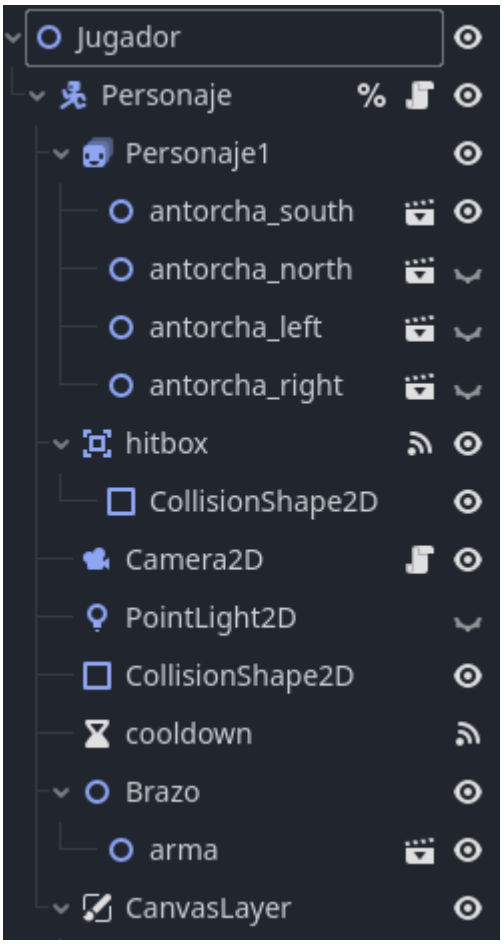
print(“Hola”)

#Y esto simplemente escribe la palabra “Hola” en la terminal cada vez que se realice la función anterior.

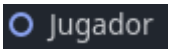
```
1  func _physics_process(_delta):
2
3      if Input.is_action_just_pressed("click"):
4          numero_epico()
5
6  func numero_epico():
7      print("Hola")
```

3.1 El jugador

El jugador es la parte más importante del juego, ya que sin él, no se podría jugar ni interactuar con nada.

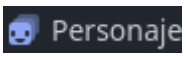


Nuestro personaje está formado por las siguientes partes:

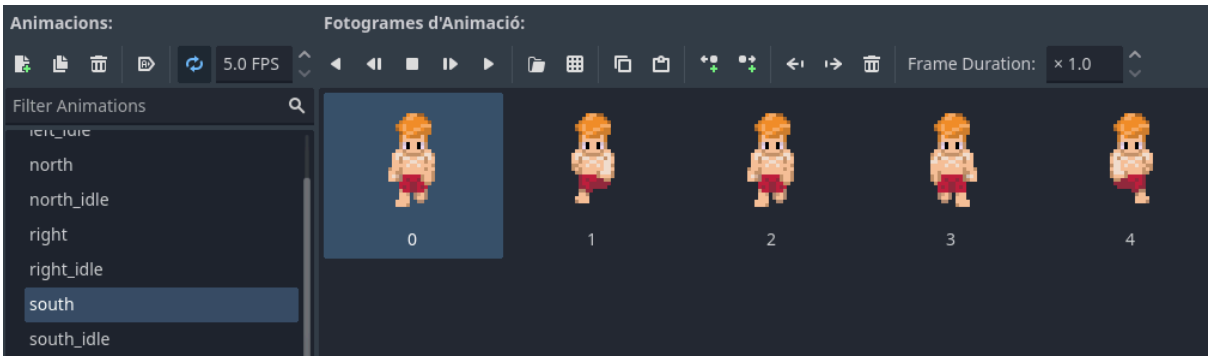


Node2D: Es la base de la escena del jugador.

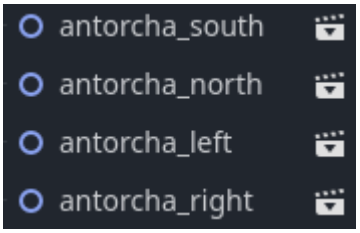
CharacterBody2D: Es controlado por el jugador y contiene el script que afecta a todos los otros nodos del personaje, los cuales son “hijos” de este nodo.



Sprite2D: Este nodo simplemente contiene todas las animaciones e imágenes que usa el personaje, cada vez que el personaje se mueve hacia una dirección, el Sprite2D pone la animación correspondiente.

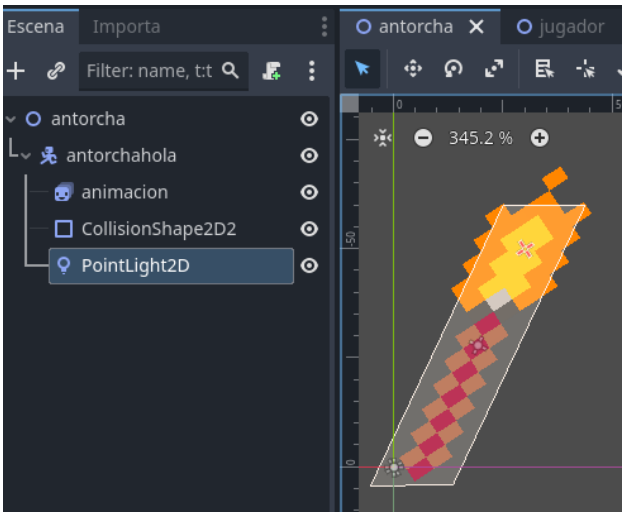


Podemos ver que el Sprite2D contiene una animación para cuando el personaje está caminando hacia cada dirección y otra para cuando está quieto.

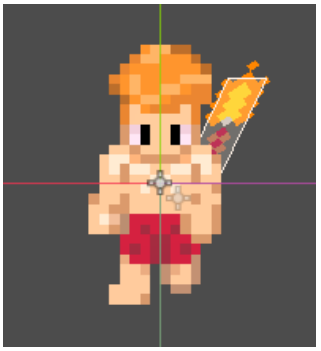


Node2D: Estos cuatro nodos són una escena entera importada dentro de la escena del jugador, estas escenas actúan como la antorcha que tiene el personaje en la espalda, que sirve para iluminar.

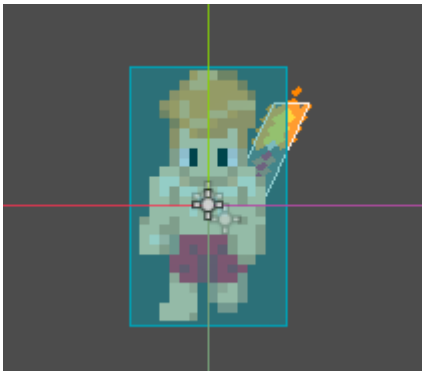
La escena de la antorcha contiene los siguientes nodos:



- **Nodo2D**: La base.
- **Characterbody2D**: Contiene todos los otros nodos.
- **AnimatedSprite2D**: Es la animación del fuego de la antorcha.
- **CollisionShape2D**: Es la colisión de la antorcha pero al final no lo usamos ya que daba errores.

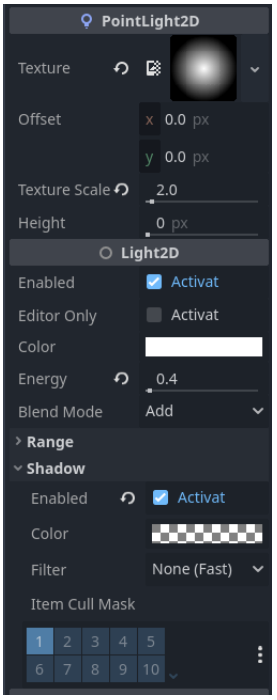


En el jugador, cada vez que el personaje mira hacia cada dirección, la antorcha de la dirección correcta se vuelve la única visible.



hitbox **Area2D:** Es un área que detecta cuando un CharacterBody2D entra en contacto, en este caso se usa para detectar cuando un enemigo le hace daño al personaje.

CollisionShape2D **CollisionShape2D:** En el caso del Area2D, se necesita una colisión que defina su forma, en este caso es el cuadrado azul que podemos ver en la imagen, cada vez que algún enemigo entre en el cuadrado, el enemigo recibe daño.

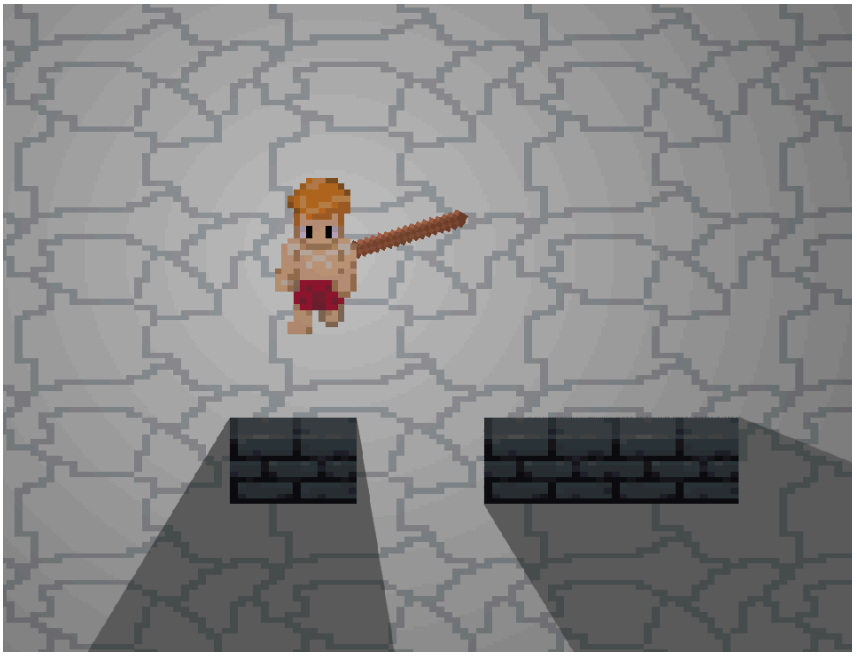


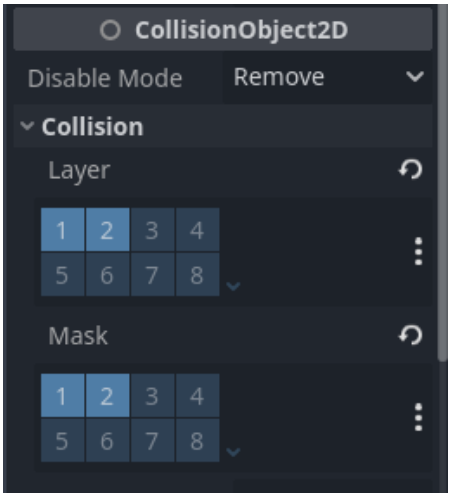
Camera2D **Camera2D:** Es lo que ve el jugador, como la cámara es un "hijo" de personaje, esta seguirá la posición del personaje todo el rato.

PointLight2D **PointLight2D:** Esta era la luz que el jugador usaba para poder ver en la oscuridad, la cual desactivamos cuando añadimos la antorcha.

Un PointLight2D usa una imagen que usa para poner sombras a su alrededor, en nuestro caso, usamos una imagen de luz que encontramos online.

Así es como se veía antes de añadir las antorchas:





CollisionShape2D **CollisionShape2D:** Se encarga de darle una colisión al jugador para que no atraviese ni paredes del mapa ni enemigos.

Se tienen que aplicar qué capas de colisión tiene cada CharacterBody2D.

cooldown **Timer:** El timer sirve para realizar una función de un script cada vez que el contador acabe, en este caso se usa para indicar cuando el personaje puede atacar y cuando no.

Brazo
arma

Node2D:

- **Brazo:** Se encarga de tener el arma como un nodo hijo para que pueda girar alrededor del jugador.
- **arma:** Es una escena importada que se encarga de tener todos los nodos necesarios para el arma y para poder dañar a los enemigos.

CanvasLayer **CanvasLayer:** Se encarga de renderizar objetos de forma independiente dentro de una escena, en este caso lo usamos para añadir toda la interfaz del jugador y de tener el script que gestiona la mayoría de la interfaz.

3.1.1 Script del jugador

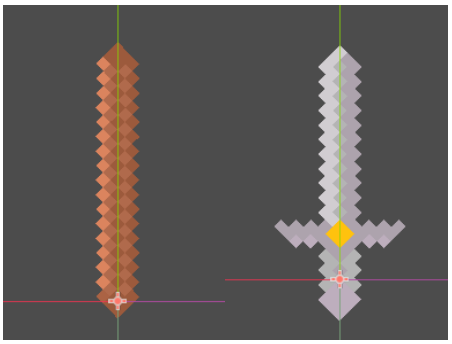
Este es el script del jugador, el cual explicaremos detalladamente que hace y que funciones tiene.

@onready var brazo : Node2D = $Brazo **@onready var barra_vida = $CanvasLayer/barra_vida_jugador**	Declara que las dos variables són dos nodos hijos del jugador.
var enemy_in_range = false **var fuego_in_range = false** **var boss_in_range = false** **var cooldown = true** **var alive = true** **var perimeter_radius:float** **var direccion = "no"** **var vida = 10**	Crea las variables necesarias para que los enemigos le hagan daño y otras como la vida o la dirección.
func jugador(): **pass**	Crea una función que usan los enemigos para saber qué nodo es el personaje.
func _process(delta): **if Input.is_action_just_pressed("pausar"):** **get_tree().change_scene_to_file("res://escenas/menu.tscn")** **barra_vida.value = vida** **var target_position = get_global_mouse_position()** **brazo.look_at(target_position)** **player_movement(delta)** **ataque_enemigo()** **ataque_fuego()** **ataque_boss()** **curacion()**	Esta es la función bucle. Tiene las funciones: - Apuntar el arma hacia la posición del ratón. - De volver al menú principal cuando se pulse el botón para salir. - Realizar otras funciones creadas en el script.
if vida <= 0: **alive = false** **vida = 0** **print("has muerto bobo")** **self.queue_free()** **get_tree().change_scene_to_file("res://escenas/game_over.tscn")**	Se encarga de comprobar constantemente si el personaje está muerto (0 en la variable vida) y si lo está, cambia la escena a la pantalla de muerte.
func set_antorcha(direction: String): **var directions = ["right", "left", "north", "south"]** **for dir in directions:** **var antorcha = $Personaje1.get_node_or_null("antorcha_" + dir)** **if antorcha:** **antorcha.visible = (dir == direction)** **func antorcha_derecha():** **set_antorcha("right")** **func antorcha_izquierda():** **set_antorcha("left")** **func antorcha_norte():** **set_antorcha("north")** **func antorcha_sur():** **set_antorcha("south")**	Su función es cambiar la antorcha visible dependiendo de la dirección.

func player_movement(_delta): **if Input.is_action_pressed("right"):** **direccion = "right"** **play_anim(1)** **velocity.x = GlobalScript.velocidad** **velocity.y = 0** **antorcha_derecha()** **elif Input.is_action_pressed("left"):** **direccion = "left"** **play_anim(1)** **velocity.x = -GlobalScript.velocidad** **velocity.y = 0** **antorcha_izquierda()** **elif Input.is_action_pressed("down"):** **direccion = "down"** **play_anim(1)** **velocity.x = 0** **velocity.y = GlobalScript.velocidad** **antorcha_sur()** **elif Input.is_action_pressed("up"):** **direccion = "up"** **play_anim(1)** **velocity.x = 0** **velocity.y = -GlobalScript.velocidad** **antorcha_norte()** **else:** **velocity.x = 0** **velocity.y = 0** **play_anim(0)** **move_and_slide()**	Se encarga simplemente del movimiento del jugador y de comprobar la dirección del jugador.
func play_anim(movement): **var direccion_actual = direccion** **var anim = $Personaje1** **if direccion == "right":** **anim.flip_h = false** **if movement == 1:** **anim.play("right")** **elif movement == 0:** **anim.play("right_idle")** **if direccion == "left":** **anim.flip_h = false** **if movement == 1:** **anim.play("left")** **elif movement == 0:** **anim.play("left_idle")** **if direccion == "down":** **anim.flip_h = false** **if movement == 1:** **anim.play("south")** **elif movement == 0:** **anim.play("south_idle")** **if direccion == "up":** **anim.flip_h = false** **if movement == 1:** **anim.play("north")** **elif movement == 0:** **anim.play("north_idle")**	Se encarga de cambiar la animación del jugador dependiendo de la dirección actual.

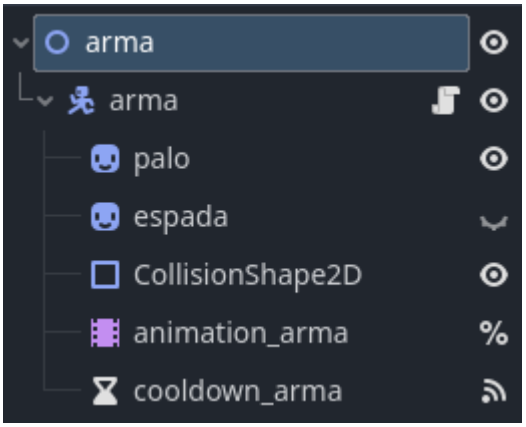
func _on_hitbox_body_entered(body: CharacterBody2D): **if body.has_method("enemigoreal"):** **enemy_in_range = true** **if body.has_method("fuegofatuo"):** **fuego_in_range = true** **if body.has_method("boss_nieve"):** **boss_in_range = true**	Hace que cuando un enemigo entra en el área 2D del jugador, modifica unas variables para definir que el enemigo está dentro de la hitbox y puede hacer daño.
func _on_hitbox_body_exited(body: CharacterBody2D): **if body.has_method("enemigoreal"):** **enemy_in_range = false** **if body.has_method("fuegofatuo"):** **fuego_in_range = false** **if body.has_method("boss_nieve"):** **boss_in_range = false**	Igual que la anterior pero cuando el enemigo sale del área.
func ataque_enemigo(): **if enemy_in_range and cooldown == true:** **vida = vida - GlobalScript.damage_sombra** **cooldown = false** **$cooldown.start()** **func ataque_fuego():** **if fuego_in_range and cooldown == true:** **vida = vida - GlobalScript.damage_fuego** **cooldown = false** **$cooldown.start()** **func ataque_boss():** **if boss_in_range and cooldown == true:** **vida = vida - 4** **cooldown = false** **$cooldown.start()**	Funciones para quitarle vida al jugador cuando detecte que el enemigo esté dentro de la hitbox y el cooldown haya acabado, cada función es para un enemigo diferente.
func curacion(): **if Input.is_action_just_pressed("pocion_vida") and GlobalScript.pociones_vida > 0 and vida < GlobalScript.vida_maxima:** **vida = vida + 1** **GlobalScript.pociones_vida = GlobalScript.pociones_vida - 1**	La función que se encarga de la curación del jugador usando la variable de pociones de vida.
func _on_cooldown_timeout(): **cooldown = true**	Se encarga de modificar la variable del cooldown cada vez que el Timer(Temporizador) acabe.

3.1.2 El arma del jugador



En nuestro juego hemos añadido dos armas, un palo y una espada, aunque dentro del juego en el personaje solo hay un nodo de arma, ahora explicaremos que forma la escena de ese nodo importado, como funciona el arma y como hace daño.

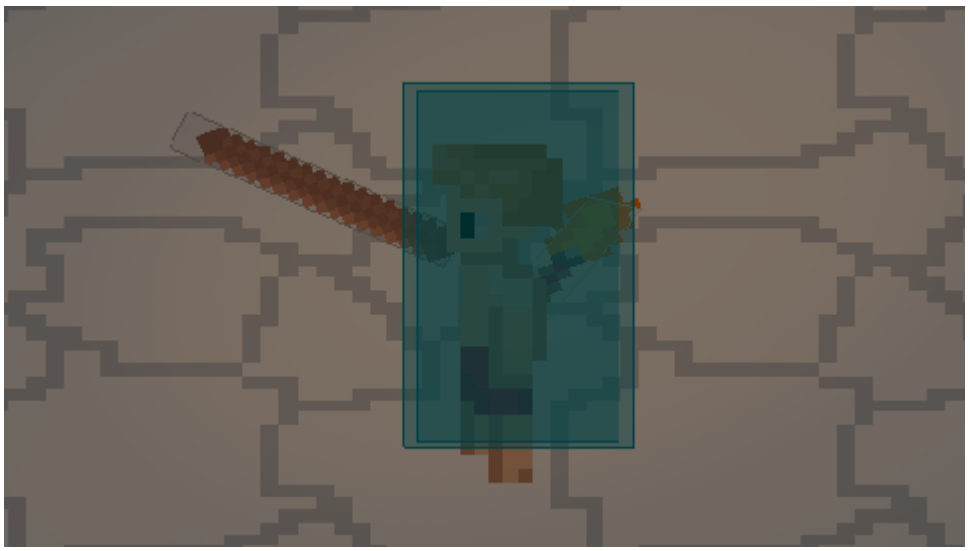
La escena del arma está formada por:



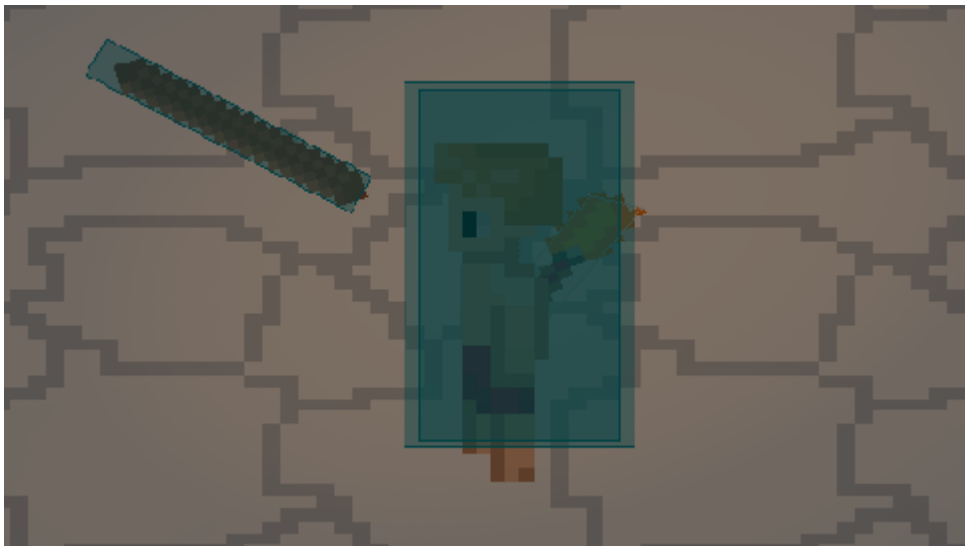
- **Nodo2D**: Es el nodo base del arma.
- **CharacterBody2D**: Se encarga del movimiento del arma y de tener su script.
- **Sprite2D**: Hay dos Sprites, uno para el palo y uno para la espada, cada vez que se usa la mejora que desbloquea la espada, el sprite del arma se cambia del palo a la espada y se modifica la variable que se encarga de las animaciones.
- **CollisionShape2D**: Es la colisión del arma.
- **AnimationPlayer2D**: Se encarga de las animaciones de cada una de las dos armas.
- **Timer**: Es el cooldown del arma.

La colisión del arma está desactivada por defecto, pero para que haga daño hace falta que esté activada, así que nosotros tuvimos la idea de modificar la colisión con la animación del arma, haciendo que al final de la animación, la colisión se active durante unos milisegundos para que pudiera hacer daño.

En la animación podemos ver que al principio el arma tiene la colisión desactivada y la posición inicial, pero cuando llega a la posición final, la colisión está activada, haciéndo así el ataque de nuestro personaje.



El arma quieta, se puede ver que no tiene ninguna colisión.

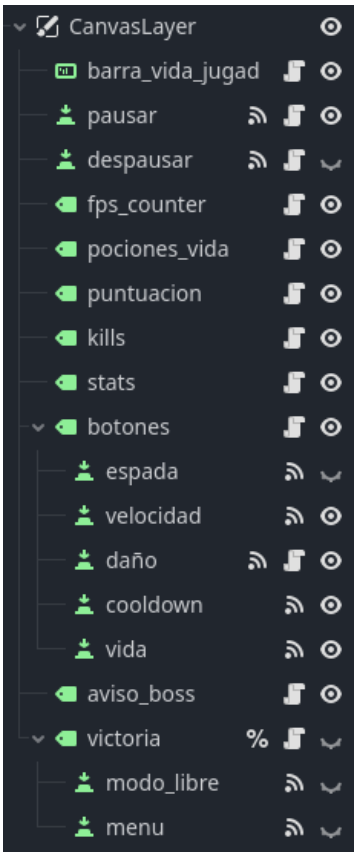


El arma atacando, esta vez sí que tiene una colisión que se encarga de ser detectada por los enemigos para que sepan cuándo recibir daño.

Script del arma:

var cooldown = false	Declara la variable del cooldown.
func _physics_process(_delta): **if GlobalScript.weapon == 1:** **$palo.show()** **$espada.hide()** **if GlobalScript.weapon == 2:** **$palo.hide()** **$espada.show()** **$cooldown_arma.wait_time = GlobalScript.cooldown_arma**	Hace que si la variable arma, la cual se modifica a 2 si se usa la mejora de espada, si está en número 1, se cambia el sprite al del palo y si está en 2 al de la espada.
if Input.is_action_just_pressed("atacar") and GlobalScript.weapon == 1 and cooldown == false: **atacar()** **await $animation_arma.animation_finished == ("atacar")** **cooldown = true** **if Input.is_action_just_pressed("atacar") and GlobalScript.weapon == 2 and cooldown == false:** **atacar_espada()** **await $animation_arma.animation_finished == ("atacar_espada")** **cooldown = true**	Hace que cuando se pulse el botón de atacar, si el cooldown está completado, llame a la función de atacar correspondiente.
func _on_cooldown_arma_timeout(): **cooldown = false**	Cuando el timer (temporizador) acaba hace que la variable cooldown sea false.
func arma(): **pass**	Función necesaria para que los enemigos puedan detectar el arma en su hitbox.
func atacar(): **$animation_arma.play("atacar")** **$animation_arma.play("desatacar")** **func atacar_espada():** **$animation_arma.play("atacar_espada")** **$animation_arma.play("desatacar_espada")**	Funciones que realizan las animaciones de ataque de cada arma.

3.1.3 La Interfaz del jugador



La interfaz del jugador está hecha usando **CanvasLayer**, está formada por varios botones y etiquetas que muestran información o sirven para modificar esta información u otras variables.

Toda la interfaz está en la misma escena que es jugador, y todos los nodos son hijos del nodo CanvasLayer, así siguen constantemente la cámara de forma estática.

Está formada por:

- **TextureProgressBar** (Barra de progreso): Usada para la barra de vida del jugador.

- **Button** (Botón): Usados para los botones de mejora que sirven para mejorar las estadísticas del jugador como la vida usando puntos que se consiguen matando enemigos, y también usado para la pantalla de victoria.

  Finalmente usamos botones para pausar y despausar el juego.

- **Label** (Etiqueta): Usados para mostrar botones o la siguiente información:

  - Contador de fps: Para medir el rendimiento de nuestro juego.
  - Número de opciones de vida disponibles.
  - Puntuación y puntos disponibles para mejoras.
  - Estadísticas del jugador (Vida, velocidad, daño y velocidad de ataque).
  - Aviso de cuando has desbloqueado la sala del boss.

Scripts de la interfaz:

Hemos hecho que cada estadística del jugador sea una variable de **GlobalScript**, así se pueden acceder desde cualquier script o escena.

El script creado para los botones de mejor es el siguiente:

```
extends Label

func _physics_process(_delta):
	botones()

func botones():
	if GlobalScript.enemigos_muertos >= 10 and GlobalScript.weapon == 1:
	$espada.show()
	if GlobalScript.enemigos_muertos < 10 and GlobalScript.weapon == 1:
	$espada.hide()
```

```
func _on_espada_pressed():
	if GlobalScript.enemigos_muertos >= 10 and GlobalScript.weapon == 1:
	GlobalScript.weapon = 2
	GlobalScript.attack_damage = GlobalScript.attack_damage + 1
	GlobalScript.enemigos_muertos = GlobalScript.enemigos_muertos - 10
	$espada.hide()

func _on_velocidad_pressed():
	if GlobalScript.enemigos_muertos >= 1:
	GlobalScript.velocidad = GlobalScript.velocidad + 5
	GlobalScript.enemigos_muertos = GlobalScript.enemigos_muertos - 1

func _on_daño_pressed():
	if GlobalScript.enemigos_muertos >= GlobalScript.coste_dano:
	GlobalScript.attack_damage = GlobalScript.attack_damage + 1
	GlobalScript.enemigos_muertos = GlobalScript.enemigos_muertos - GlobalScript.coste_dano
	GlobalScript.coste_dano = GlobalScript.coste_dano * 2

func _on_cooldown_pressed():
	if GlobalScript.enemigos_muertos >= 2:
	GlobalScript.cooldown_arma = GlobalScript.cooldown_arma * 0.75
	GlobalScript.enemigos_muertos = GlobalScript.enemigos_muertos - 2

func _on_vida_pressed():
	if GlobalScript.enemigos_muertos >= 4:
	GlobalScript.vida_maxima = GlobalScript.vida_maxima + 1
	GlobalScript.enemigos_muertos = GlobalScript.enemigos_muertos - 4
```

3.2 Enemigos

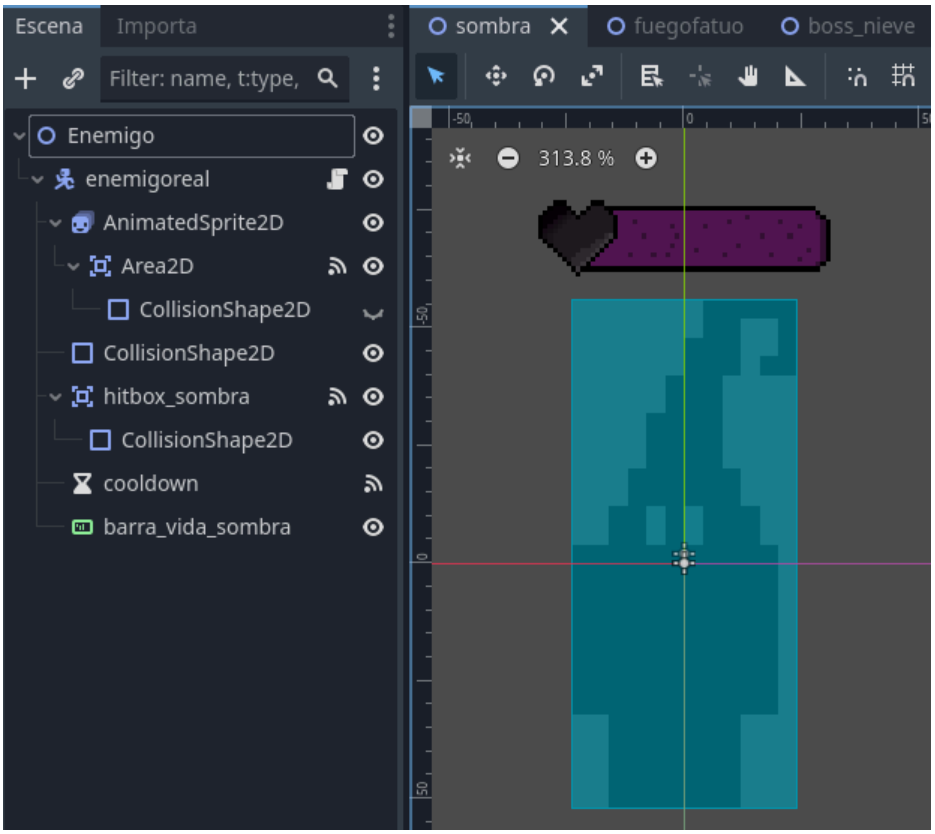
En nuestro juego hay 3 enemigos diferentes:

- **Sombras**: Son el primer enemigo que añadimos y el más débil.

- **Fuegos fatuos**: Són más rápidos que la sombra y hacen más daño, pero con menos vida.

- **Boss**: Es el jefe final de nuestro juego, al cual puedes acceder cuando consigues 100 puntos.

Las escenas de los tres enemigos son prácticamente iguales, solo cambian las variables como la vida o el daño, el sprite y algunas cosas del script.



Los enemigos están formados por los siguientes nodos:

- **Node2D**: Es la base del enemigo.

- **CharacterBody2D**: Se encarga del movimiento y de tener el script del enemigo.

- **AnimatedSprite2D**: Su función es poner la animación del enemigo correspondiente, hay una animación para cuando está quieto, una para cuando se mueve y otra para cuando muere, esto es así para todos los enemigos.

- **Area2D**: Esta es el área que se encarga de detectar al jugador, esto hace que el enemigo pueda perseguir al jugador para poder tocarle y quitarle vida.

- **CollisionShape2D**: Es la colisión necesaria para el area2d que detecta al enemigo.

- **CollisionShape2D**: Esta colisión es la que se encarga de hacer que el enemigo no pueda atravesar paredes ni al enemigo, es la colisión que tiene con otras escenas.

- **Area2D**(Hitbox): Es la colisión que se encarga de hacer que el enemigo pueda recibir daño cuando recibe la colisión del arma del jugador.

- **CollisionShape2D**: Colisión necesaria para el area2d de la hitbox.

- **Timer**(Cooldown): Es el cooldown del enemigo, hace que no haga daño constantemente cuando toca al jugador.

- **TextureProgressBar**: Esta es la barra de vida de la sombra, la cual se puede ver en la imagen anterior encima del enemigo.

3.2.1 Scripts de los enemigos

Script de los enemigos:

var speed = 125 **var player_chase = false** **var player = null** **var arma_en_enemigo = false** **var health = 6** **var player_attack_area = false** **var cooldown_enemigo = true** **var atacando = false**	Declara todas las variables que usa el enemigo.
@onready var barra_vida = **$barra_vida_sombra**	Se encarga de declarar que la variable “barra_vida” es el nodo definido con “$” de la escena del enemigo.
func enemigoreal(): **pass**	Función necesaria para que el jugador pueda detectar el enemigo que lo está atacando.
func _ready(): **scale_stats()** **func _physics_process(_delta):** **barra_vida.value = health** **get_damaged()**	Hace que cuando empieza el juego, se repiten constantemente las funciones de subir las estadísticas del enemigo (dificultad) y la de recibir daño, también hace que la variable de barra de vida sea la otra variable creada llamada “health”.

if health <= 0: **if $AnimatedSprite2D.animation != "muerte":** **$AnimatedSprite2D.play("muerte")** **speed = 0** **player_chase = false** **cooldown_enemigo = false** **player_attack_area = false** **atacando = false** **$hitbox_sombra.set_deferred("monitorable", false)** **$hitbox_sombra.set_deferred("monitoring", false)** **await get_tree().create_timer(0.5).timeout** **self.queue_free()** **GlobalScript.enemigos_muertos = GlobalScript.enemigos_muertos + 1** **GlobalScript.puntuacion_total = GlobalScript.puntuacion_total + 1** **if GlobalScript.pociones_vida < 9:** **GlobalScript.pociones_vida = GlobalScript.pociones_vida + 1**	Se encarga de cuando muere el enemigo hacer que reproduzca la animación de muerte antes de desaparecer y de subir la puntuación y puntos del jugador.
if player_chase and player: **var direction = (player.global_position - global_position).normalized()** **velocity = direction * speed** **$AnimatedSprite2D.play ("movimiento")** **if (player.global_position.x - global_position.x) < 0:** **$AnimatedSprite2D.flip_h = true** **else:** **$AnimatedSprite2D.flip_h = false** **else:** **velocity = Vector2.ZERO** **$AnimatedSprite2D.play ("idle")** **move_and_slide()**	Se encarga de hacer que el enemigo persiga al jugador y de su movimiento.
func _on_area_2d_body_entered(body): **player = body** **player_chase = true** **func _on_area_2d_body_exited(_body):** **player = null** **player_chase = false**	Funciones que se encargan de hacer que el enemigo sepa si el personaje está en su área de perseguir.
func _on_hitbox_sombra_body_entered(body: Node2D): **if body.has_method("arma"):** **arma_en_enemigo = true**	Estas funciones hacen que el enemigo reciba daño cuando detecta que el arma del jugador está dentro de su hitbox.

func get_damaged(): **if arma_en_enemigo == true and cooldown_enemigo == true:** **health = health - GlobalScript.attack_damage** **cooldown_enemigo = false** **$cooldown.start()** **func _on_hitbox_sombra_body_exited(body: Node2D):** **if body.has_method("arma"):** **arma_en_enemigo = false**	
func _on_cooldown_timeout(): **cooldown_enemigo = true**	Es el cooldown del enemigo.
func scale_stats(): **var multiplier = int(GlobalScript.puntuacion_total / 20)** **speed = 125 * (1 + 0.05 * multiplier)** **health = 6 + (1 * multiplier)** **$barra_vida_sombra.max_value = health** **$barra_vida_sombra.value = health**	Función encargada de cambiar las variables cada vez que la puntuación llega a un cierto punto (en este caso cada vez que la puntuación sube 20 puntos, hace que le suba la vida y la velocidad al enemigo).

Tambien se sube el daño cada vez que la puntuacion sube 50 puntos, aunque esto cómo usa variables Globales, se hace en el script Global, el script es el siguiente:

```
func scale_stats():
	var multiplier_damage = int(GlobalScript.puntuacion_total / 50)
	damage_sombra = 1 + multiplier_damage
	damage_fuego = 2 + multiplier_damage
```

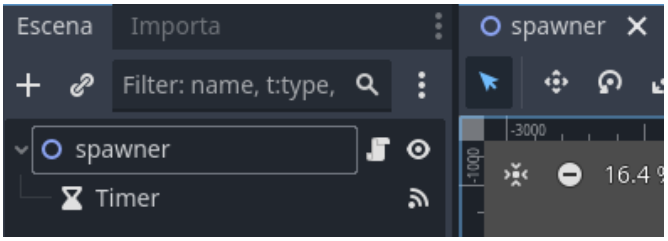
El script del enemigo de antes es prácticamente igual a los de los otros dos enemigos, solo cambian algunas variables y nombres.

3.2.2 Spawner de enemigos

Para el juego, como es de oleadas de enemigos, hacía falta una manera de hacer aparecer enemigos constantemente en el mapa, esto lo hicimos creando una escena de un “spawner” que se encargaría de aparecer enemigos en el mapa cada cierto tiempo, usando un contador para determinar el tiempo que queríamos.

Ahora explicaremos qué nodos forman la escena del spawner, la configuración necesaria para el nodo principal y el funcionamiento del script.

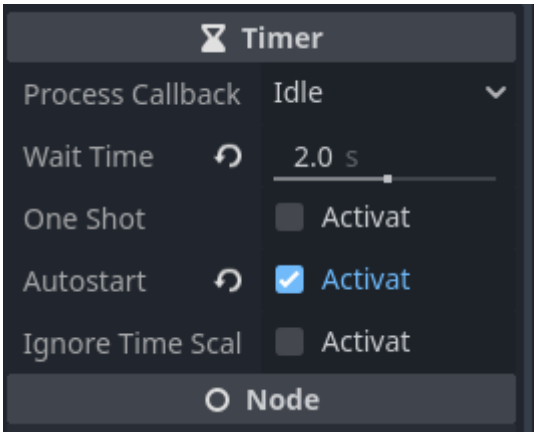
Nodos:



El spawner solo usa dos nodos, estos son:

- **Node2D**: Se encarga de determinar qué escenas hará aparecer en el mapa y de tener el script para que puedan aparecer.
- **Timer**: El contador que determina cada cuantos segundos aparecen enemigos, en este caso són cada 2 segundos.

  También puse que el contador empezará automáticamente.



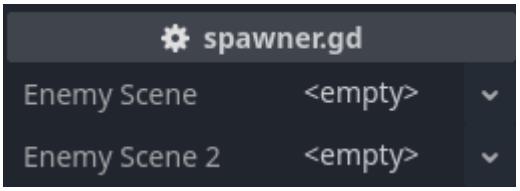
Script:

@export var enemy_scene : PackedScene **@export var enemy_scene_2 : PackedScene** **@onready var timer = $Timer**	Define que las variables “enemy_scene” y “enemy_scene_2” serán las escenas determinadas en otro sitio, esto lo usamos para que el script pueda saber que escenas tiene que hacer aparecer. También se encarga de determinar que la variable del contador es el nodo Timer de la misma escena.
func _physics_process(_delta): **timer.wait_time = GlobalScript.timer_spawn_enemigo**	Determina que el contador durara el tiempo definido por la variable Global. Esto lo teníamos planeado usar para subir la dificultad, pero al final lo dejamos simplemente a 2 segundos por defecto.
func _on_timer_timeout():	Se encarga de hacer aparecer los

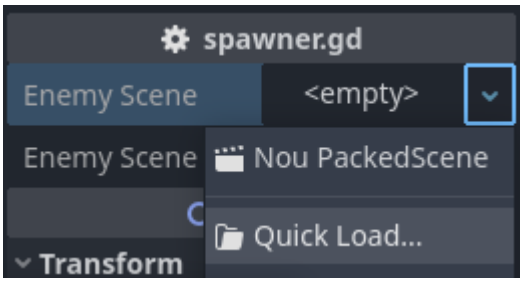
var enemy = null **if randf() < 0.75:** **enemy = enemy_scene.instantiate()** **else:** **enemy = enemy_scene_2.instantiate()** **enemy.global_position = get_random_offscreen_position()** **add_child(enemy)** **func get_random_offscreen_position() -> Vector2:** **var space_state = get_world_2d().direct_space_state** **var screen = get_viewport().get_visible_rect()** **var margin = 20** **var max_attempts = 10** **for i in range(max_attempts):** **var x = 0.0** **var y = 0.0** **match randi() % 4:** **0:** **x = randf_range(screen.position.x, screen.end.x)** **y = screen.position.y - margin** **1:** **x = randf_range(screen.position.x, screen.end.x)** **y = screen.end.y + margin** **2:** **x = screen.position.x - margin** **y = randf_range(screen.position.y, screen.end.y)** **3:** **x = screen.end.x + margin** **y = randf_range(screen.position.y, screen.end.y)** **var pos = Vector2(x, y)** **var query = PhysicsPointQueryParameters2D.new()** **query.position = pos** **query.collide_with_areas = false** **query.collide_with_bodies = true** **var result = space_state.intersect_point(query)** **if result.is_empty():** **return pos** **return Vector2(screen.position.x - margin, screen.position.y - margin)**	enemigos cada vez que el contador acabe, también hace todos los cálculos necesarios para que los enemigos no aparezcan o fuera del mapa o dentro de colisiones o demasiado cerca del jugador. También hace que haya un 25% de porcentaje de aparecer el enemigo **"fuego fatuo"** y un 75% de aparecer una **“sombra”**.

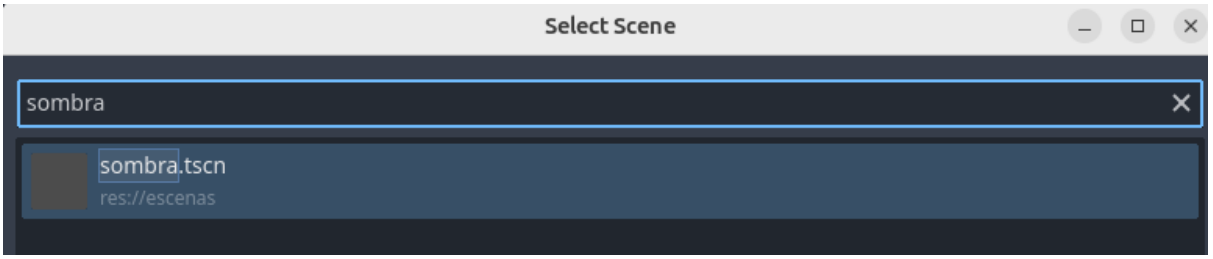
Configuración del nodo2d:

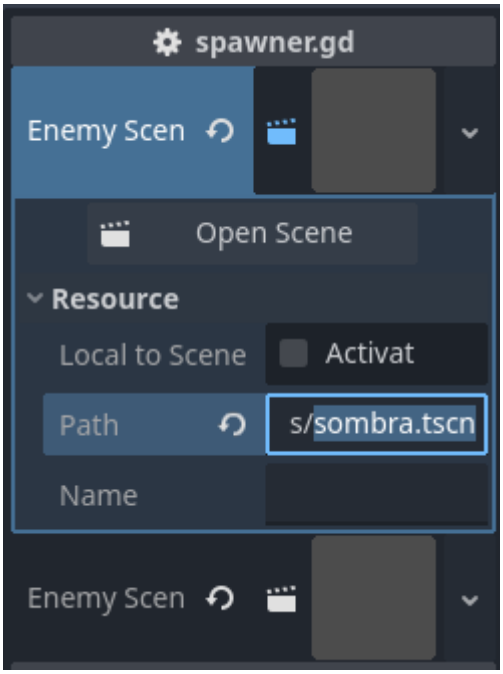
Hacía falta determinar qué escenas eran las variables creadas en el script, en las que la variable era “**PackedScene**”, esto se hace seleccionando el Nodo 2D y eligiendo las escenas de los enemigos en la parte necesaria.



Aquí es donde se tienen que elegir las escenas de los enemigos.



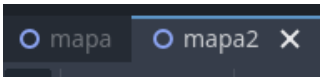


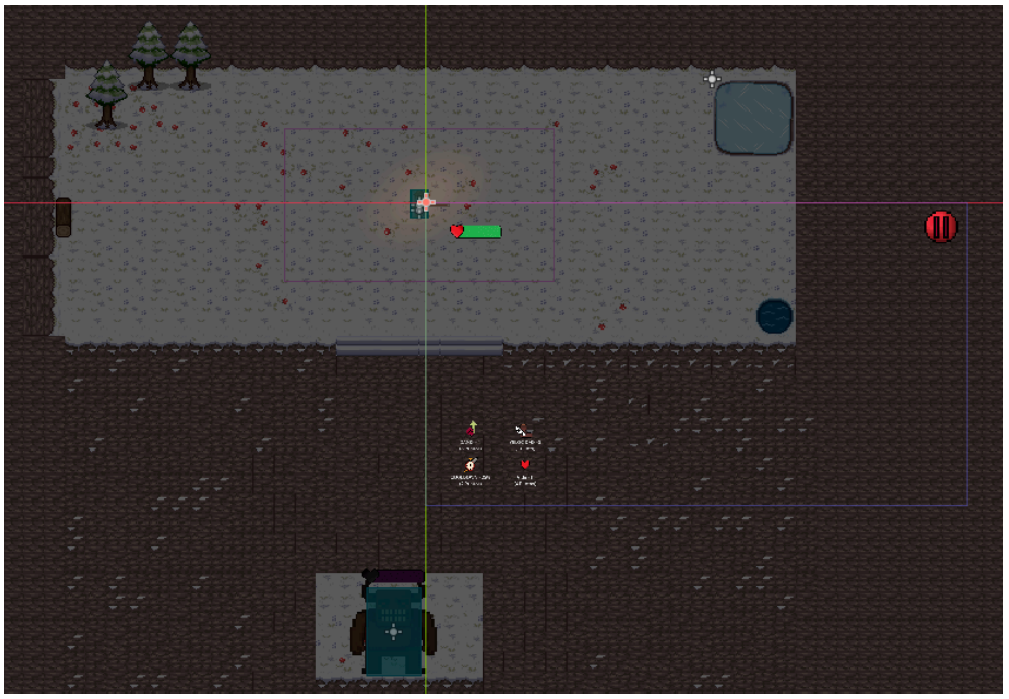


Aquí ya podemos ver que se han seleccionado las escenas de los dos enemigos, definiendo qué “Enemy_scene” es la sombra y “Enemy_scene_2” es el "fuego fatuo".

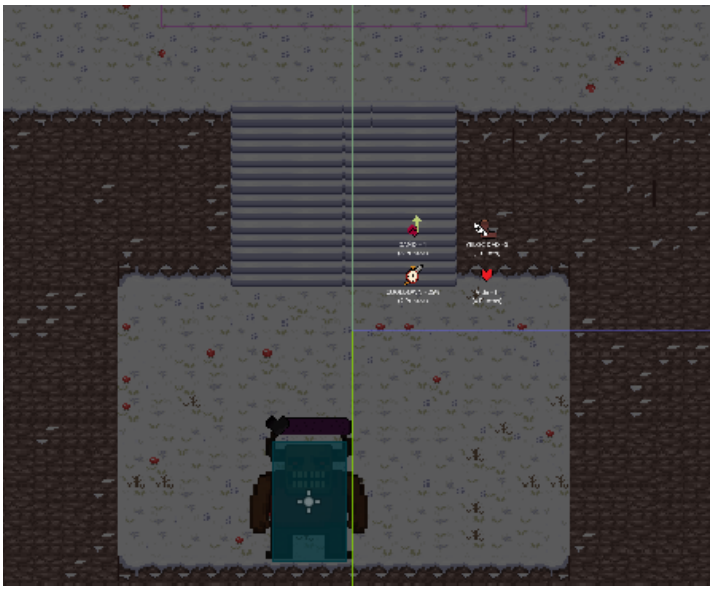
3.3 El mapa

En vorkuta de momento hay solo 2 mapas, uno es el que creamos al principio, el cual fue muy simple ya que lo usamos para empezar a hacer el juego, y el otro es un mapa más detallado y bien hecho.

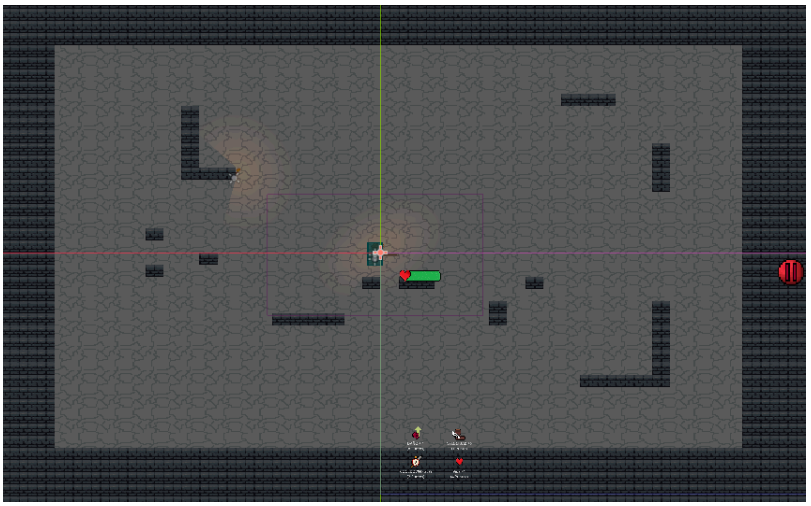




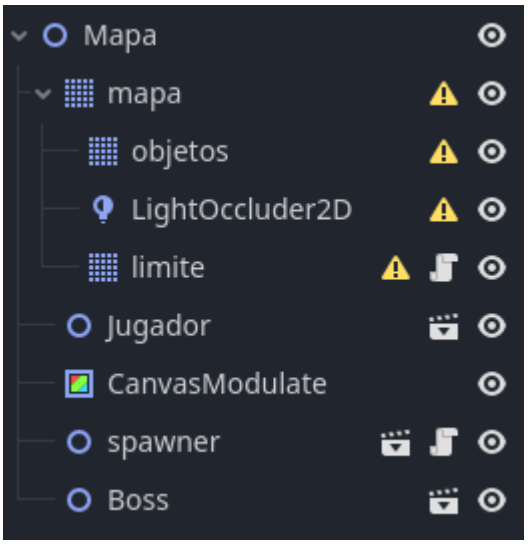
Este es el mapa principal, es el único que contiene un boss, que se puede acceder cuando el personaje llega a 100 puntos, cuando eso pasa, la parte del mapa que bloquea al boss desaparece.



El mismo mapa con la parte que bloquea al boss desbloqueada.



Y este es el primer mapa que creamos, se puede jugar como el otro aunque no tiene ningún boss, lo hemos puesto para el modo libre.

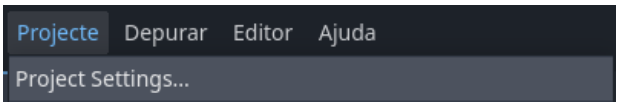


Nodos:

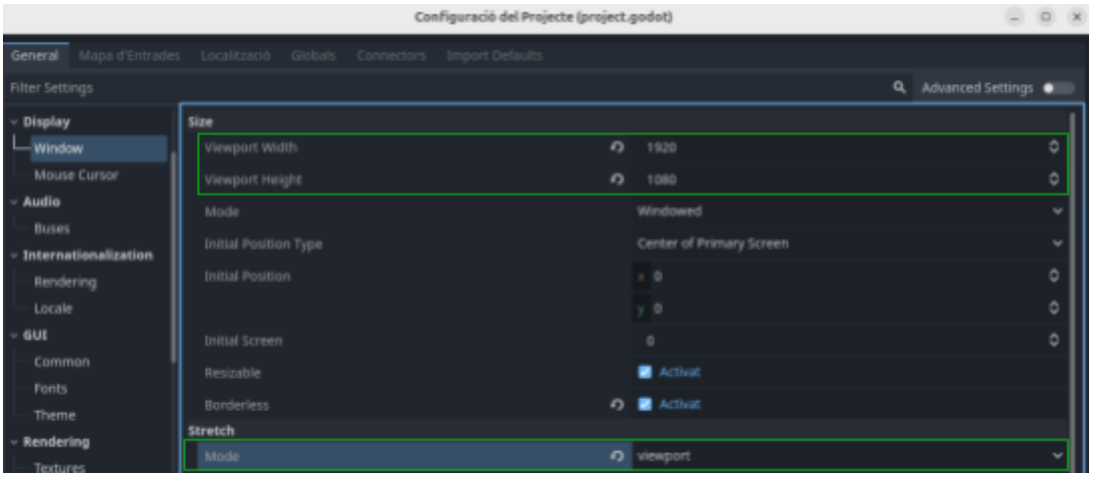
Las escenas de los dos mapas són muy similares, ahora vamos a explicar qué nodos forman la escena del mapa principal:

- **Node2D**: Es la base del mapa.
- **TileMap (mapa)**: Es la primera capa del mapa, podríamos decir que es todo el fondo y las paredes.
- **TileMap (objetos)**: Contiene todos los objetos del mapa, como los árboles o el agua.
- **LightOccluder2D**: Se encarga de las sombras que dan todos los objetos de luz.
- **TileMap (límite)**: Esta es la parte del mapa que bloquea el acceso al boss, podemos ver que tiene un script, el cual se encarga de hacer que desaparezca el nodo cuando la puntuación llegue a 100.
- **Node2D (Jugador)**: Esta es la escena del jugador importada dentro del mapa.
- **CanvasModulate**: Se encarga de añadir una oscuridad al mapa con el color que nosotros elegimos.
- **Node2D (spawner)**: Es la escena del spawner importada en el mapa.
- **Node2D (Boss)**: Igual que la anterior pero para la escena del jefe final.

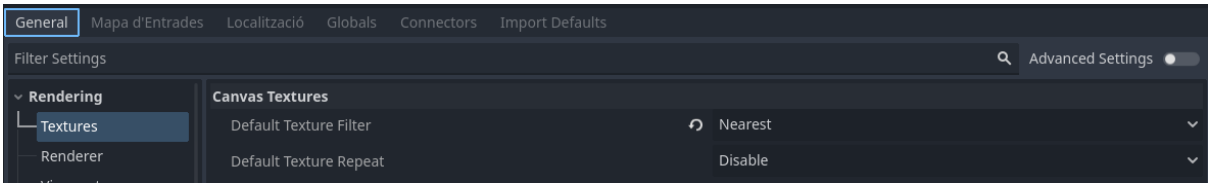
3.4 Configuración del juego



Para que el juego funcionase de la manera que queríamos, hacía falta hacer algunos cambios en la configuración general del juego, los cambios qué cambios són los siguientes:



Primero cambiamos la resolución del juego, en donde pusimos la resolución estándar que usan la mayoría de juegos, 1920x1080, también pusimos el modo “viewport”, para evitar problemas con la interfaz que causaban los otros modos.



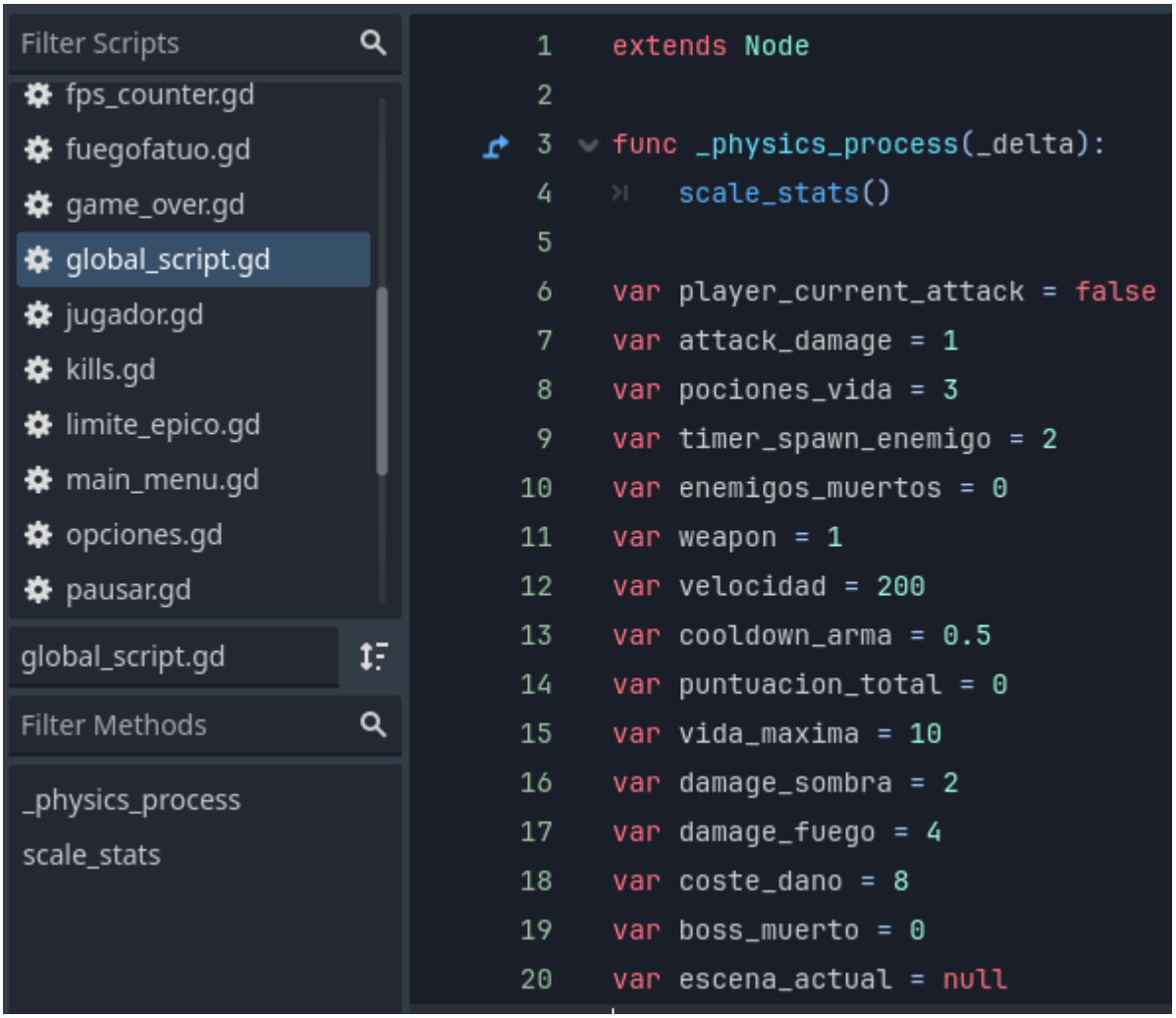
También hacia falta cambiar el filtro de las texturas a “Nearest” (cercano), esto era importante ya que sin eso, todas las texturas se veían borrosas.

En la izquierda está en modo por defecto, se ve todo muy borroso.
Y en la derecha está en modo “Nearest”, ahora todas las texturas pixeladas se ven correctamente.

Estas són todas las configuraciones del juego que hicieron falta hacer, algo explicaremos que es el script global del juego, y para que diferentes funciones lo usamos.

GlobalScript:

El GlobalScript se puede referenciar desde cualquier otro script, lo usamos para crear variables que se modifican desde otras escenas o nodos, este es el script global que nosotros creamos:



También lo usamos para crear la función que se encargaría de subir el daño de los enemigos cada vez que el jugador consiga 50 de puntuación, esto lo hicimos ya que la variable del daño de los enemigos la creamos dentro del script global.

Esta es la función que sube el daño a los enemigos:

```
func scale_stats():
    var multiplier_damage = int(GlobalScript.puntuacion_total / 50)
    damage_sombra = 1 + multiplier_damage
    damage_fuego = 2 + multiplier_damage
```

3.5 Los menús

En nuestro juego usamos solo 4 menús diferentes, estos són:

- **Menú principal:** Es lo primero que ve el jugador cuando empieza el juego, y desde aqui tiene las opciones: Jugar; Opciones; Salir.

- **Opciones:** Esta es la pantalla en la que el jugador podrá modificar aspectos del juego, teníamos pensado añadir la opción de desactivar o activar musica y de subir o bajar el volumen, aunque no nos dio tiempo de añadir eso, por esta razón el menu de opciones no contiene nada aparte del botón de volver al menú principal.

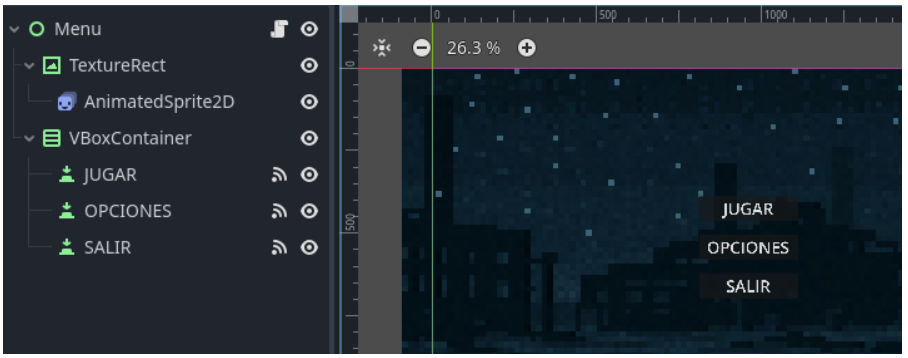
- **Elegir mapa:** Este es el menú que ve el jugador al darle a Jugar, aquí puede elegir entre los dos mapas cual quiere jugar.

- **Game over:** Esta es la pantalla de muerte del jugador, la cual aparece cada vez que el jugador muera, y tiene un botón para volver al menú principal.

Funcionamiento de los menús:

Los menús tienen un funcionamiento muy simple, solo tienen que cambiar la escena a la necesaria dependiendo del botón que se pulse, en este caso, como todos los menús són muy similares, explicaremos solo que nodos forman el menú principal y su script.

Nodos:



El menú principal está formado por:

- **Node2D**: Es la base del menú.

- **TextureRect**: Se encarga de mostrar la imagen del menú principal, esto lo usamos al principio, hasta que lo reemplazamos por el siguiente nodo, el cual era una animación que dibujamos nosotros.

- **AnimatedSprite2D**: Esta es la animación actual del menú principal.

- **VBoxContainer**: Se encarga de organizar los botones.

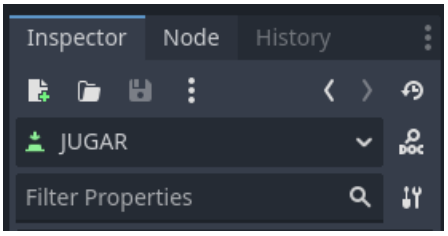
- **Button**: Su función es cambiar la escena a la determinada cada vez que se pulse un botón.

Script:

func _physics_process(delta): **$TextureRect/AnimatedSprite2D.play("default")**	Se encarga de hacer que se este reproduciendo la animación del fondo del menú principal todo el tiempo.
func _on_jugar_pressed(): **get_tree().change_scene_to_file("res://escenas/elejir_mapa.tscn")** **GlobalScript.pociones_vida = 3** **GlobalScript.enemigos_muertos = 0** **GlobalScript.weapon = 1** **GlobalScript.velocidad = 200** **GlobalScript.attack_damage = 1** **GlobalScript.cooldown_arma = 0.5** **GlobalScript.puntuacion_total = 0** **GlobalScript.vida_maxima = 10**	Función que cambia la escena a la de elegir mapas cada vez que se pulse el botón de jugar, también reinicia todas las variables que se pueden haber modificado en la partida anterior.
func _on_opciones_pressed(): **get_tree().change_scene_to_file("res://escenas/opciones.tscn")**	Cambia la escena cada vez que se pulse el botón de opciones del menú.
func _on_salir_pressed(): **get_tree().quit()**	Sale del juego cada vez que se pulse el botón de salir.

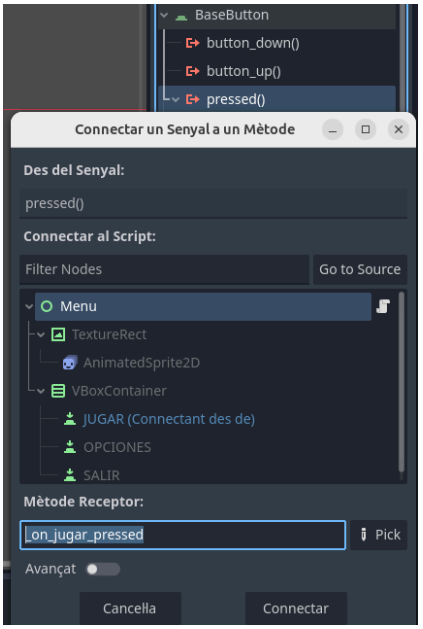
Funcionamiento de señales:

Para hacer que el script del menu funcionase correctamente, se tenia que hacer que pudiera recibir entradas de otros nodos, en este caso de los botones.



Esto se hace seleccionando el nodo del cual se quiere recibir entradas y dándole a “Node” al lado del inspector.

Dentro de “Node” se busca la señal que se quiere tener como entrada, en este caso “pressed()”, qué significa presionado.



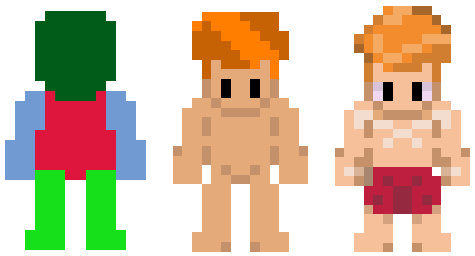
Una vez encontrada la señal, se hace doble click y se selecciona a donde se quiere llegar la señal, en este caso como el script lo tiene el Nodo2D, se selecciona ese.

Y ahora ya tendríamos la señal añadida en el script.

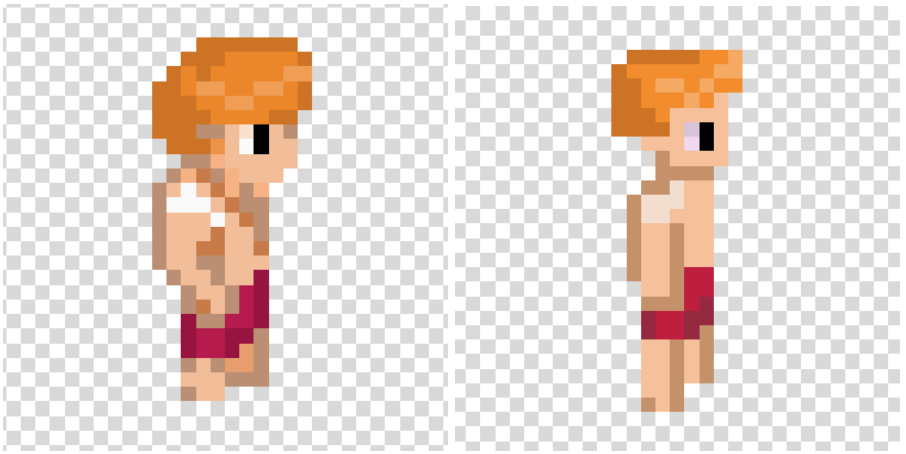
```
6  func _on_jugar_pressed():
7      print("hola")
```

4. Dibujo inicial

Para la creación de nuestro personaje protagonista tomamos de inspiración a nosotros mismos, ya que los tres somos aficionados de los videojuegos y nos gustaba la idea de aparecer en algún juego.Escogimos primero a Pau, no obstante no teníamos tiempo para incluirnos y decidimos por nuestra propia cuenta añadirnos en próximos juegos.



Aquí se puede apreciar el primer boceto que usamos como base para hacer a nuestro protagonista, poco después fuimos agregando color y detallando en el dibujo. También tenemos un boceto de Pau estando de lado y su versión final.



4.1 Creación de los personajes/armas

Una mención honorífica para estos sprites que no pudimos añadir

Teniamos pensado en que fuera como un especie de mercader donde nos vendiera armas y pociones.

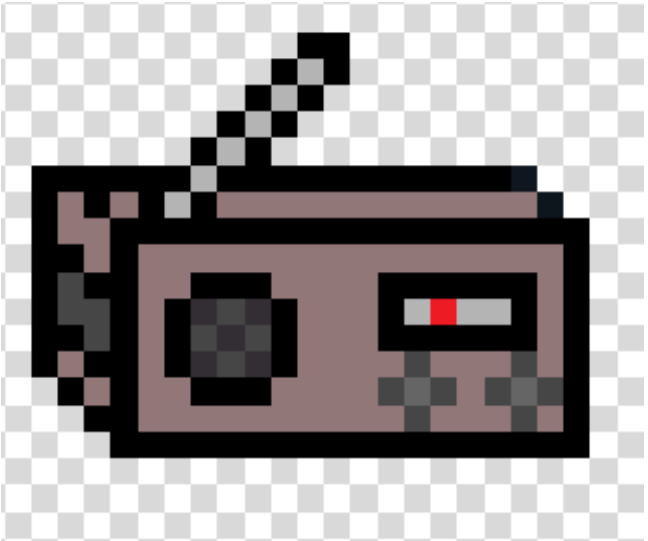
Un ogro rey que teníamos pensado meter como boss final al final fue reemplazado.

Un zombie que al final fue reemplazado por la sombra ya que pensamos que no tendría sentido argumental.(Aunque no hayamos añadido historia)

Estas son armaduras que habíamos pensado implementar la de la izquierda seria de cuero, siendo la que menos protección te da pero la mas rápida de adquirir.La imagen de la derecha sería una de hierro con más protección y la evolución directa a la cuero.

Esto es una cabeza de prueba que hicimos.

Esto de aquí son una cereza, que teníamos planeado hacer como un item para recuperar vida o mejorar el daño temporalmente, y una radio con la que queriamos implementar una canción propia

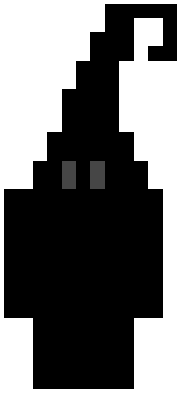


4.1.1 Inspiración

https://www.slynyrd.com/blog/2019/10/21/pixelblog-22-top-down-character-sprites

vorkuta/ubicación

  fuego fatuo/pokemon

  berserk,sombra

Boss/Doom,baron of hell

4.1.2 Creación

Para la creación de nuestros sprites, utilizamos una página web que nos proporcionaba muchas herramientas de diseño. Esta página se llama Pixilart y a continuación daré el enlace a la misma. https://www.pixilart.com/

A continuación explicare las herramientas que más utilizamos y que nos parecieron interesantes:

El lápiz: Es la más básica, sirve para pintar un cuadrado del color que hayas seleccionado.

Goma: La segunda más básica sirve borrar un cuadrado pintado selecionado.

Cubo: Si formamos una figura donde no hubiese salida podíamos pintar una gran superficie de un mismo color.

Selector de color:su utilidad es la de copiar el color por ejemplo:necesitas el color que has usado en otro lado pero no lo tienes seleccionado,solo lo tocas para que seleccione el color sin necesidad de estar buscando el que mas se parezca en toda la gama de colores

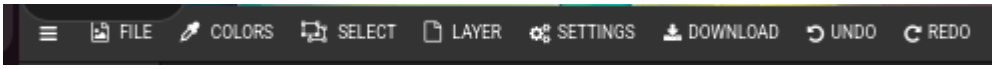
Seleccionar: Con esta herramienta podemos seleccionar una parte de nuestro dibujo y decidir si queremos hacerla mas pequeña,grande estirarla desde punto. etc, etc

Aclarar/oscurecer: Con esta herramienta podemos, como su nombre indica, aclarar o oscurecer un cuadrado previamente pintado. Es muy útil hacer sombras en tus dibujos.

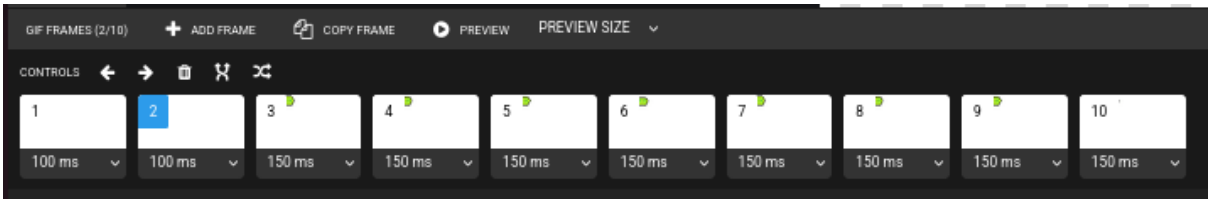
Pintura de aereosol: Se utiliza para adrle el efecto de como se hubiera pintado a traves de un bote. No la llegamos a utilizar mucho puesto que nos era dificil hacer un buen uso.

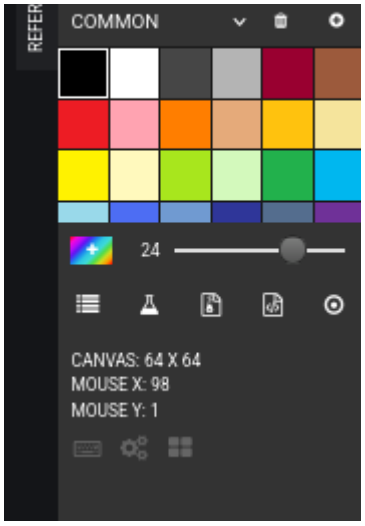
Herramienta de filtro:sirve para barrejar los colores entre ellos como una mancha. Hicimos las letras del título de nuestro juego de este modo

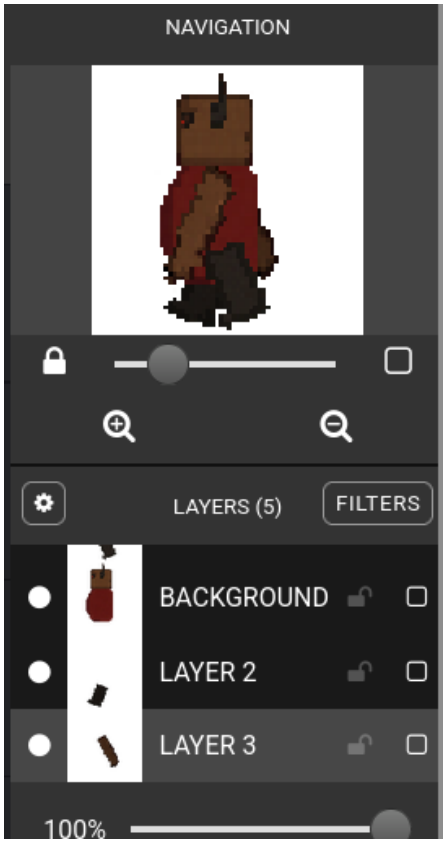
En la zona superior de la pagina estaran las siguientes opciones.
Que explicaremos mas las utilizadas:
En file podremos hacer varias acciones con nuestro dibujo o uno ya anteriormente creado.
Undo y Redo con esto podremos rebobinar y volver a hacer nuestra última acción.



Esto de aquí se utiliza para hacer frames, utilizamos esta herraminta para la crecaion de la animacion del game over

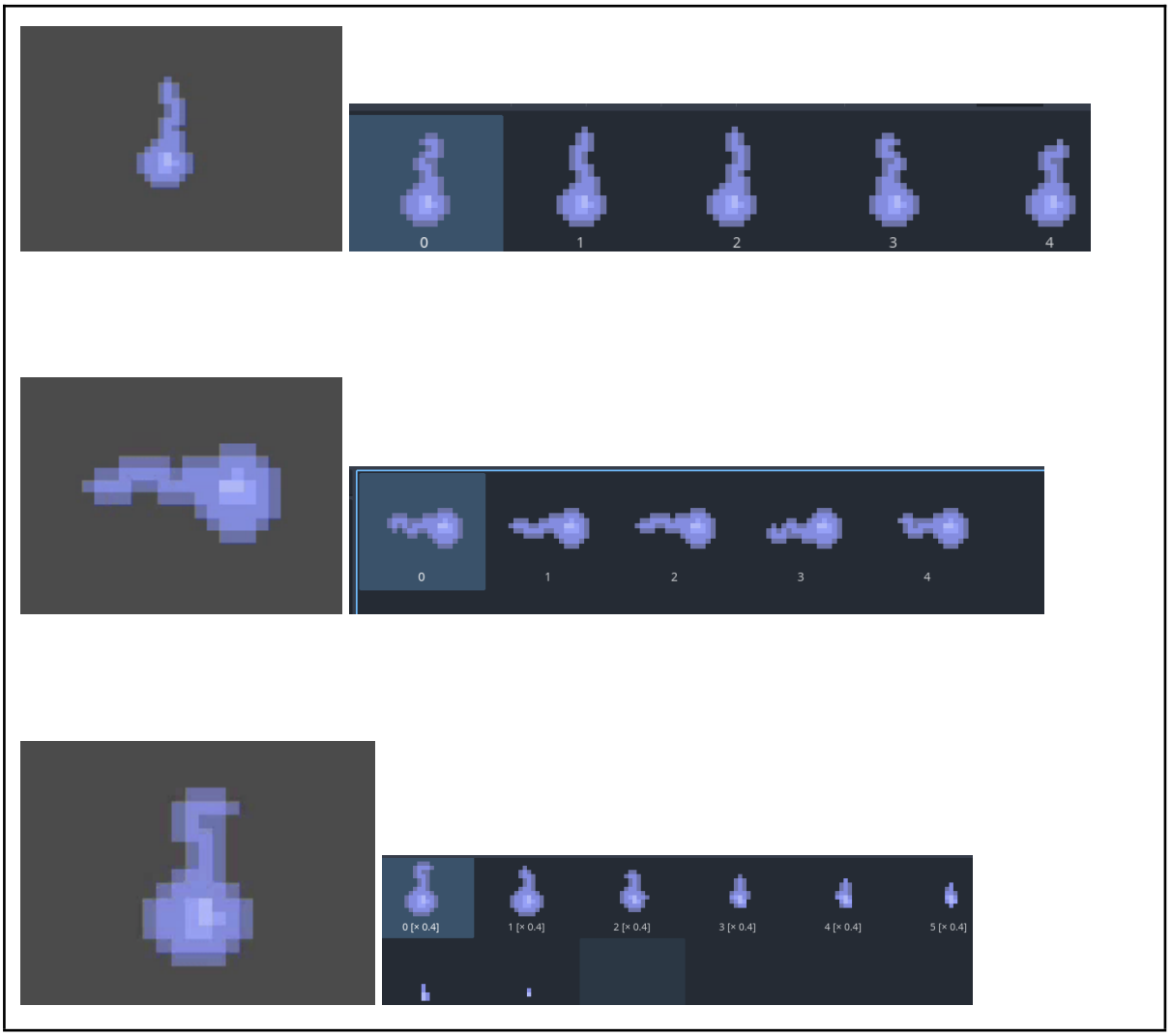


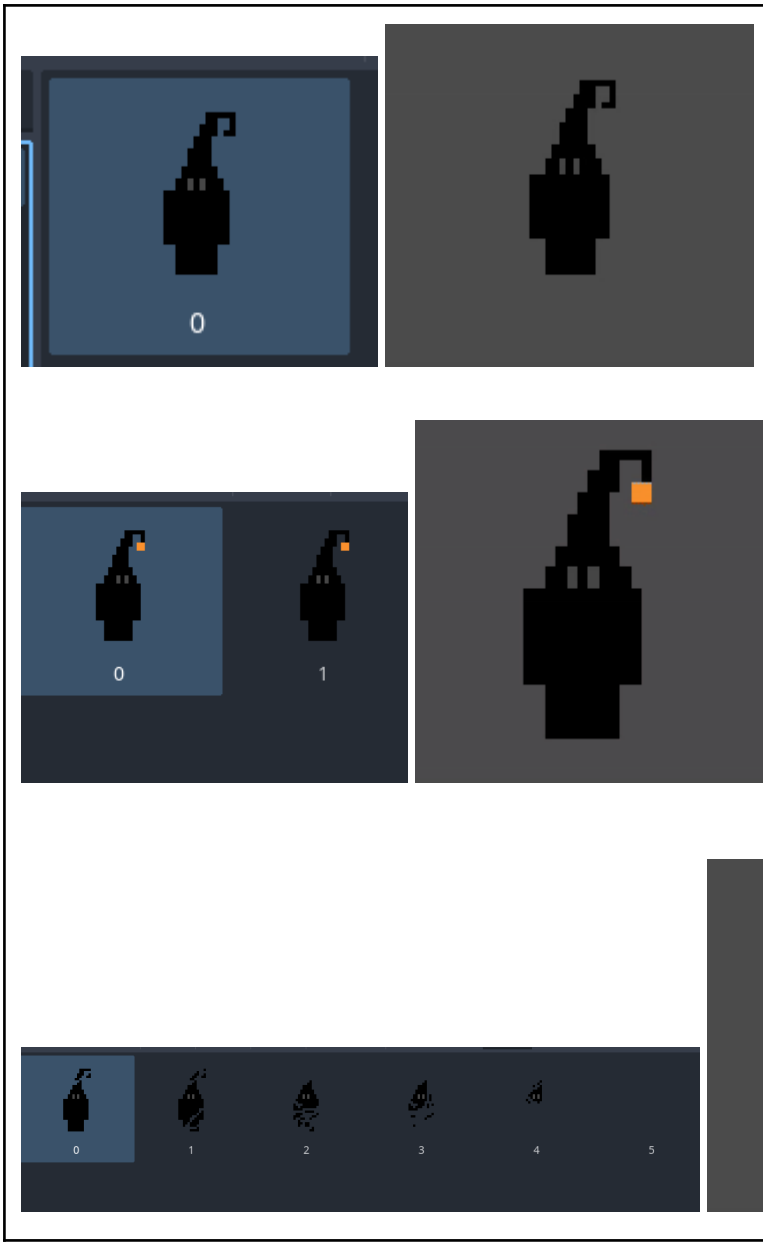
	Para escoger los colores más usados. Para escoger el tamaño de dibujo goma u otras herramientas utilizables. Resolución de pixel art (canvas 64x64x) Mouse X o Y:cuando pintas un punto te dice en la posición que está horizontal el eje x,vertical el eje y.

	**Capas** :las capas se usan para poder crear una profundidad a una imagen sin tener que sobrepintar una misma imagen por ejemplo en un bosque en al que la capa del final pones como si fuera de noche y en otra de dia con eso solo tendrias que cambia una imagen separada sin tan molesto ni manchar lo demás. En mi caso lo he usado para hacer el movimiento de lado del boss haciendo por partes su cuerpo asi no puedo agarrar una pierna sin que se agarre lo demas y girarla simulando su movimiento caminando.

5. Pixelarts

Para crear el idle/movimiento/muerte de los elementos que se ven en pantalla , se ha necesitado añadirle frames. Un frame es una imagen concreta dentro de una sucesión de imágenes en movimiento.
Todo junto quedaría de esta manera.



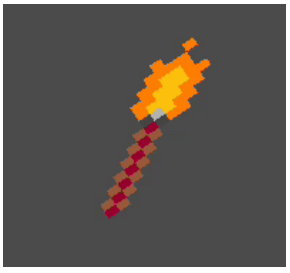

0
0
1
0
1
2
3
4
5


0

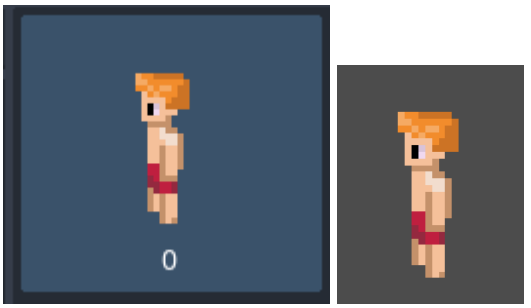

0
1
2
3
4
5
6
7
8
9
10
11

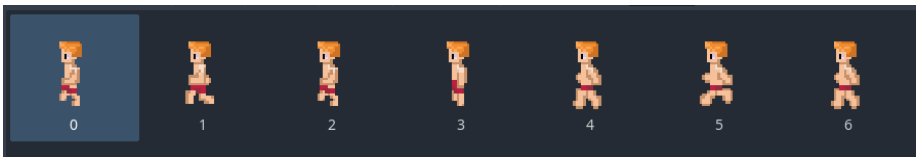

0
1
2
3
4
5
6
7
8
9
10
11

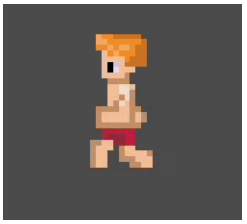

0
1
2
3
4
5
6



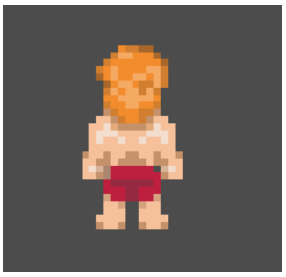

0 [× 0.6]
1 [× 0.6]
2 [× 0.6]
3 [× 0.6]
4 [× 0.6]
5 [× 0.6]
6 [× 0.6]
7 [× 0.6]
8 [× 0.6]
9 [× 0.6]
11 [× 0.6]
12 [× 0.6]
13 [× 0.6]
14 [× 0.6]
15 [× 0.6]
16 [× 0.6]
17 [× 0.6]
18 [× 0.6]

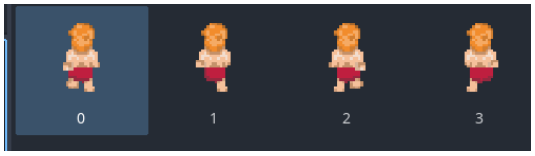

0

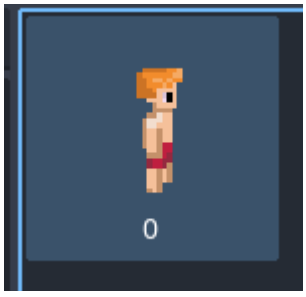

0
1
2
3
4
5
6

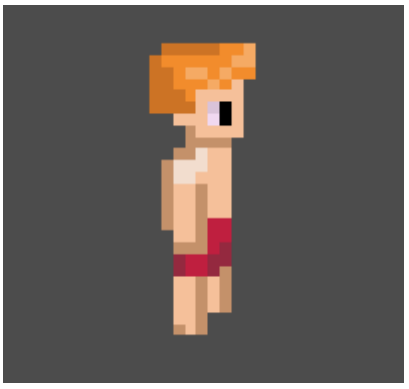



0

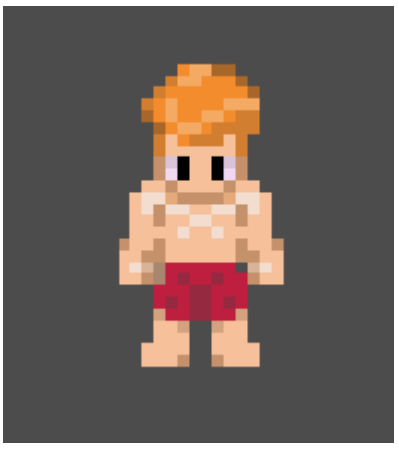


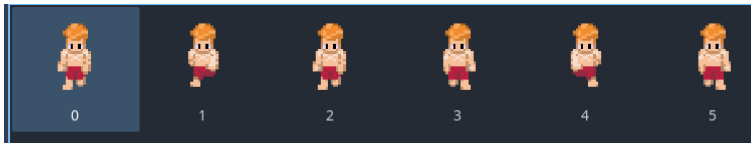

0
1
2
3

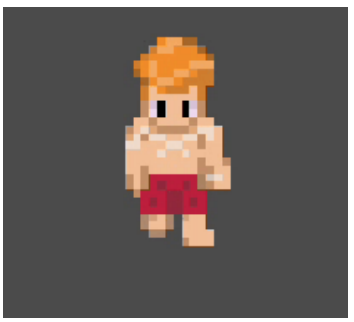

0

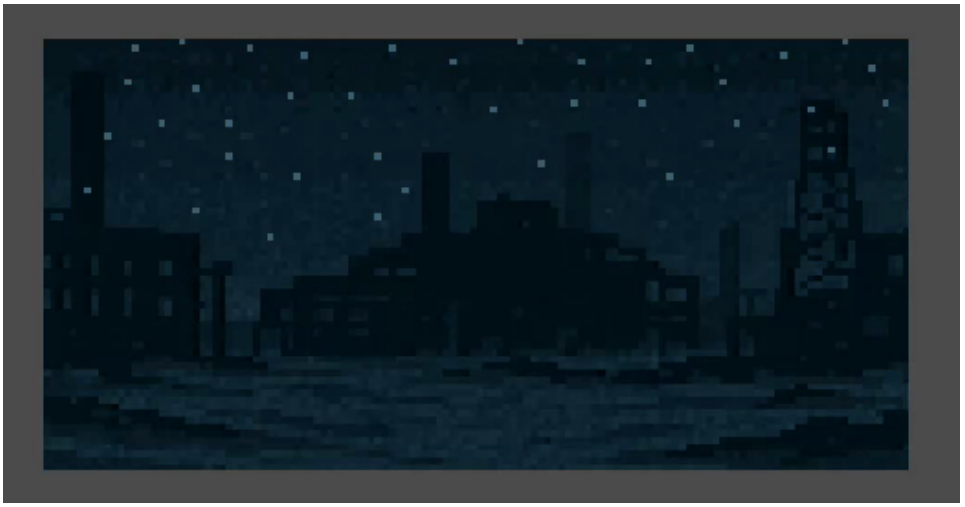



0




0
1
2
3
4
5





0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35

36 37 38 39 40 41 42

6. Errores y problemas

Nosotros tuvimos algunos errores y problemas en el desarrollo del juego, esto se debe principalmente a que empezamos a crear el juego con casi nada de idea del funcionamiento de godot ni de sus scripts.

Por esa razón hubieron muchos errores dentro de nuestros scripts, cosa que pudimos al final corregir buscando en internet gente que hubiera tenido un error similar, al final esto no era tan problema ya que teníamos mejor conocimiento sobre godot y programación en GDScript.

Otro problema que tuvimos fue que cuando buscabamos algun tutorial para aprender a utilizar godot y para empezar nuestro juego, la mayoría de veces era un tutorial que usaba una version desactualizada de godot, así que nos costó seguir estos tutoriales ya que algunas cosas eran diferentes.

En la parte artística, nuestra inexperiencia en pixel art nos hizo pasar malas jugadas cuando quisimos hacer movimiento a los personajes.
Más tarde aprendimos a hacerlo por capas facilitandonos el movimiento, no obstante también tuvimos algunos problemas por que la web que usábamos de pixelart no nos permitia meter más de 9 frames así que cada 9 tuvimos que descargarlos aparte y luego juntarlos todos siendo un problema de organización.

7. Trabajo a futuro

Gafas de Realidad Virtual

Queremos introducir las gafas VR para que parezca que el juego sea un poco más inmersivo a pesar de ser un juego 2D y que sea dificil de contrastar las vr ,poder usarlo adentro del juego los mandos que suelen traer las gafas para hacer acciones y con los botones de ese mando subirte salud,daño etc…

Gráficos 3D o Efectos 3D

Aunque el estilo principal seguirá siendo 2D pixel art, planeamos:

- Posiblemente integrar cinematográficas con animaciones en 3D que contrasten con el arte 2D.

Escudo-Resistencia

Queremos añadir un sistema de defensa que incluya escudos o mayas, lo que permitirá:

- Bloquear ataques enemigos o que te peguen menos
- Mejorar la estrategia del combate, haciendo que no todo sea ofensivo.

Configuración del volumen de la música que queremos añadir en un futuro dentro de la pantalla de opciones del menu principal.

Tutorial que explique las mecánicas base del juego y el objetivo.

Historia

Queremos integrar una narrativa sólida en el juego para que los jugadores se sientan más conectados con el mundo y los personajes. Esto incluye:

- Un argumento principal con misiones o capítulos.
- Escenas cinemáticas o diálogos que desarrollen el universo del juego.
- Eventos o decisiones que afecten el curso de la historia.

8. Conclusiones

Después de este proyecto, además de trabajar en equipo y enfrentarnos a los problemas, todos en el equipo podemos concluir que volveríamos a repetir algo semejante a esto. Pues aunque fuera frustrante, alguna parte más que otra, nos divertimos y gozamos en la creación de nuestro juego. Además hemos adquirido diversas habilidades que nos servirán en el dia de mañana en nuestros empleos. Como ser mas pacientes, y ser mas perspicaz y tenaces.

9. Glossario

Godot: Motor de desarrollo de videojuegos libre y de código abierto que permite crear juegos en 2D y 3D. Es la herramienta principal que hemos utilizado para desarrollar nuestro juego 2D.

GDScript: Es el lenguaje de programación que usa Godot, permite programar la lógica de todos las partes del juego.

Pixel Art: Estilo gráfico a base de cuadrados como píxeles visibles coloreables que a conjunto crea una imagen.. Es el estilo artístico que hemos elegido para nuestro juego por su estética retro y su simplicidad visual aparte de tenerlo de inspiración de otros juegos indie como terraria o stardew valley.

Sprites: En nuestro caso un conjunto de dibujos que utilizamos en nuestro proyecto para representar acciones, como correr entre otros.

Scene (Escena): Grupo de nodos de Godot relacionados entre sí para formar partes del juego, como puede ser los mapas, enemigos o personajes, etc.

Node (Nodo): Son el elemento básico de Godot. Cada escena está compuesta por nodos, y cada uno cumple una función en concreto .

Cooldown: Tiempo de espera entre una acción y la posibilidad de repetirla. En este caso lo usamos para el ataque de los enemigos y jugador.

Spawner: Se encarga de generar enemigos automáticamente en el mapa cada cierto tiempo. Es esencial en nuestro juego, ya que está basado en oleadas.

Interfaz de usuario (UI): Es el conjunto de elementos visuales que dan información a la persona que está jugando o le permite interactuar con botones que tienen una función. Algunos ejemplos són las barras de vida, el daño, o los botones de mejora.

Hitbox: Es el área de colisión que se encarga de detectar si un personaje, enemigo, objeto o pared ha sido golpeado, se usa para gestionar el daño y los contactos entre los objetos y personajes.

10. Bibliografía

Autor	Fecha	Título	Editor,revista	URL
Godot forum	No hay	Object follow mouse in radius	Forum Godot	Enlace
floringame(YT)	2022,7 mayo	Easy Health Bars in Godot 4	Youtube	Enlace
DevWorm(YT)	2022	How to Create an RPG in Godot 4	Youtube	Enlace
Slynyrd	2019,21 octubre	Pixelblog #22: Top-down character sprites.	Slynyrd Blog.	Enlace
Craftpix	No hay	Free fantasy enemies pixel art sprite pack.	Craftpix.net.	Enlace

Godot:

https://forum.godotengine.org/t/object-follow-mouse-in-radius/8185/2
Para el ataque del arma.

https://www.youtube.com/watch?v=GPYSACim_P4
Para las barras de vida del personaje y los enemigos.

https://www.youtube.com/watch?v=pBoXqW4RykE&list=PL3cGrGHvkwn0zoGLoGorwvGj6dHCjLaGd
Lista de reproducción con videos de como crear un juego RPG en godot, usamos algunos videos de esta lista para aprender cosas básicas de godot, aunque estaban un poco desactualizados asi que no pudimos seguir todo completamente.

Pixel Art:

Algunas guias para dibujar bien a los personajes, nos fue bien menos en las piernas.

https://www.slynyrd.com/blog/2019/10/21/pixelblog-22-top-down-character-sprites

https://craftpix.net/freebies/free-fantasy-enemies-pixel-art-sprite-pack/