Institut Puig Castellar
Santa Coloma de Gramenet

Shadow Traveler

(Proyecto de desarrollo)

CFGM Administración de Sistemas Microinformáticos y Redes

Autor: Eric Aguilar Pulido
Grupo: A
Curso académico: 2SMX

B) GNU Free Documentation License (GNU FDL)

C) Copyright

Resumen del proyecto

Desarrollé un videojuego de aventura y plataformas usando el motor gráfico de Godot Engine 4 desde cero, usando tutoriales y aprendiendo sobre la marcha el lenguaje de GDScript. El objetivo es obtener un videojuego funcional, estable y con una experiencia jugable satisfactoria.

Este proyecto tiene como temática principal el desarrollo de un videojuego de tipo plataformero en 2D, centrado en la acción y con la presencia de jefes finales que suponen un reto para el jugador. El objetivo del proyecto es crear un juego completamente jugable utilizando el motor Godot Engine 4.6, aplicando sus herramientas y funcionalidades para el desarrollo en dos dimensiones.

Para alcanzar el objetivo que deseo, se ha seguido una metodología basada principalmente en el aprendizaje autodidacta. Se han utilizado recursos disponibles en Internet, como documentación oficial, foros y vídeos tutoriales en YouTube, que han hecho comprender cómo funciona este motor gráfico, la programación de mecánicas básicas, el diseño de niveles y la implementación de enemigos y jefes.

La conclusión es que el proyecto ha permitido adquirir conocimientos prácticos sobre el desarrollo de videojuegos, mejorar la capacidad de resolución de problemas y demostrar que, con los recursos adecuados y constancia, es posible crear un juego funcional desde cero.

Palabras clave:

Plataformero, jefes finales, parkour, aventura, metroidvania

Abstract:

This project focuses on the development of a 2D platformer video game centered on action, featuring boss battles that pose a challenge to the player. The main objective of the project is to create a fully playable game using the Godot Engine 4, applying its tools and features for two-dimensional game development.

To achieve this objective, a methodology based mainly on self-directed learning was followed. Various resources available on the Internet were used, such as official documentation, forums, and tutorial videos on YouTube, which helped in understanding how this game engine works, including basic mechanic programming, level design, and the implementation of enemies and bosses.

In conclusion, the project made it possible to acquire practical knowledge of video game development, improve problem-solving skills, and demonstrate that

with the right resources and perseverance, it is possible to create a functional game from scratch.

Keywords:

Platformer, bosses, parkour, adventure, metroidvania

Índice

Lista de figuras

1 Introducción

Quiero hacer un juego usando Godot Engine 4.6 de formato plataformas con jefes finales.

1.1 Contexto

Los videojuegos 2D se basan en un control preciso del personaje, donde acciones como saltar, atacar o esquivar deben responder de forma rápida para ofrecer una experiencia fluida al jugador. El mayor atractivo de estas obras radica en la arquitectura de sus entornos, concebida específicamente para poner a prueba los reflejos y la destreza manual sin necesidad de recurrir a sistemas enrevesados. Las aventuras que sitúan la cámara fuera del protagonista, especialmente en un plano plano, son auténticos pilares fundacionales en la creación de videojuegos. Estas entregas llevan cautivando al público desde los orígenes del medio gracias a una exposición de los acontecimientos sumamente nítida y frontal.

Street Fighter y Hollow Knight Silksong

1.2 Justificación

1. Permite aplicar los conocimientos adquiridos en programación y desarrollo de software.
2. Introduce al desarrollo de videojuegos
3. Utiliza una tecnología moderna, gratuita y ampliamente utilizada en el sector de videojuegos indie.
4. Da como resultado un producto funcional y evaluable, que puede ser usado en un futuro.
5. Fomenta el aprendizaje autónomo y la resolución de problemas técnicos complejos.

1.3 Objetivos

1.3.1 Objetivo general

Desarrollar un videojuego de plataformas 2D completamente funcional, con un gran diseño de nivel y jefe final, utilizando Godot Engine 4 y aplicando buenas prácticas de desarrollo de software.

1.3.2 Objetivos específicos

1. Diseñar la mecánica principal de movimiento del personaje.
2. Implementar un sistema de nivel.
3. Desarrollar enemigos comunes y jefes finales con patrones de ataque.
4. Implementar un sistema de vida, daño y derrota.
5. Crear una interfaz de usuario básica.
6. Documentar correctamente el proceso de desarrollo.

1.4 Estrategia y planificación del proyecto

La estrategia elegida consiste en el desarrollo de un producto nuevo, diseñado específicamente para este proyecto. No se parte de un videojuego existente, sino de una idea original adaptada al alcance y tiempo que tenga disponible.

Esta estrategia es la más adecuada porque:

- Permite controlar totalmente el diseño y las funcionalidades.
- Evita dependencias externas.
- Facilita ajustar el alcance en función del progreso del proyecto.

Desde el punto de vista de la viabilidad, el proyecto es asumible por una sola persona gracias al uso de herramientas gratuitas.

1.5 Metodología de trabajo

Se siguió una metodología ágil, concretamente una adaptación de Scrum, ya que permite dividir el desarrollo en iteraciones cortas y revisar el progreso de forma continua.

- Control de versiones mediante Git.

Esta metodología es adecuada porque el desarrollo de videojuegos suele requerir ajustes constantes en la jugabilidad y el diseño.

1.6 Estudio económico y presupuestario

Producto	Descripción	Precio Final
Ordenador Gaming	Usado para desarrollar el proyecto desde casa	950€
Ordenador portátil	Usado para desarrollar el proyecto desde casa	150€
Ordenador de clase	Usado para programar en clase	600€ aprox
Godot Engine 4	Motor gráfico libre	0€
Teclado	Periférico	25€
Ratón	Periférico	10€
Github	Plataforma de almacenamiento de código	0€
Recursos gráficos y sonoros	Sprites y sonidos	0€

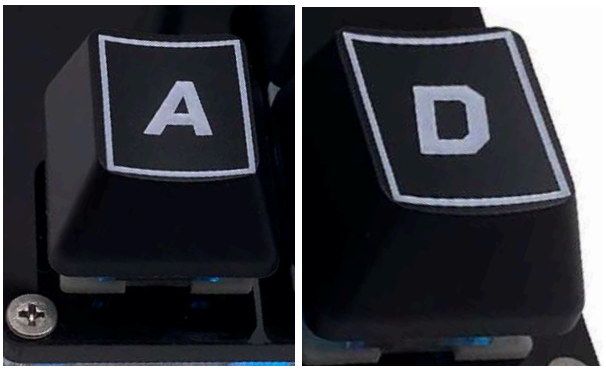
2 Descripción del proyecto

2.1 Análisis de requisitos

Terminar el juego de manera correcta, sin errores críticos de funcionamiento, buenas mecánicas, enemigos variados y con comportamientos diferenciados , y buenos jefes.

2.1.1 Requisitos funcionales

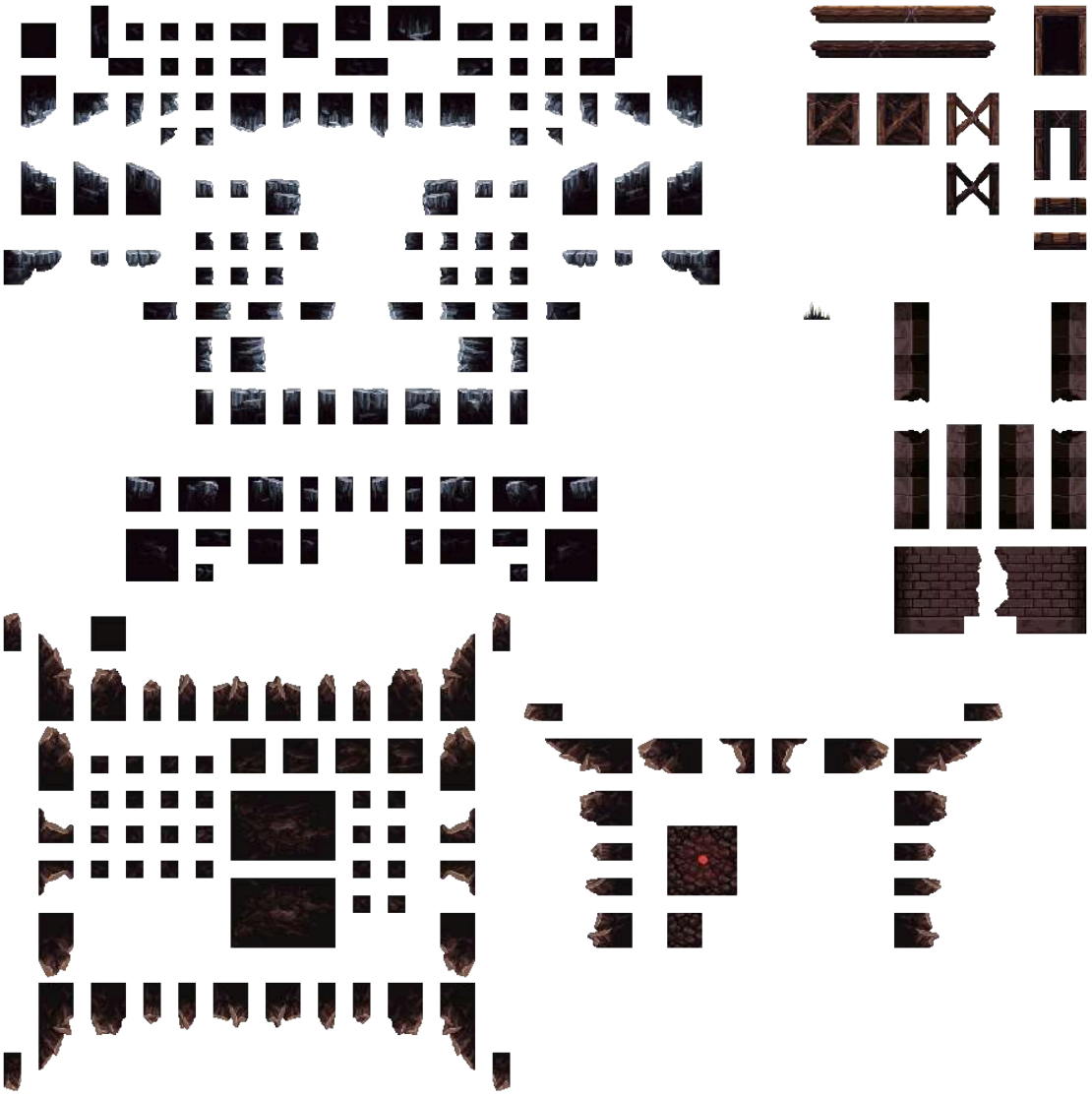
El jugador se moverá hacia adelante y hacia atrás con las teclas “A,D”



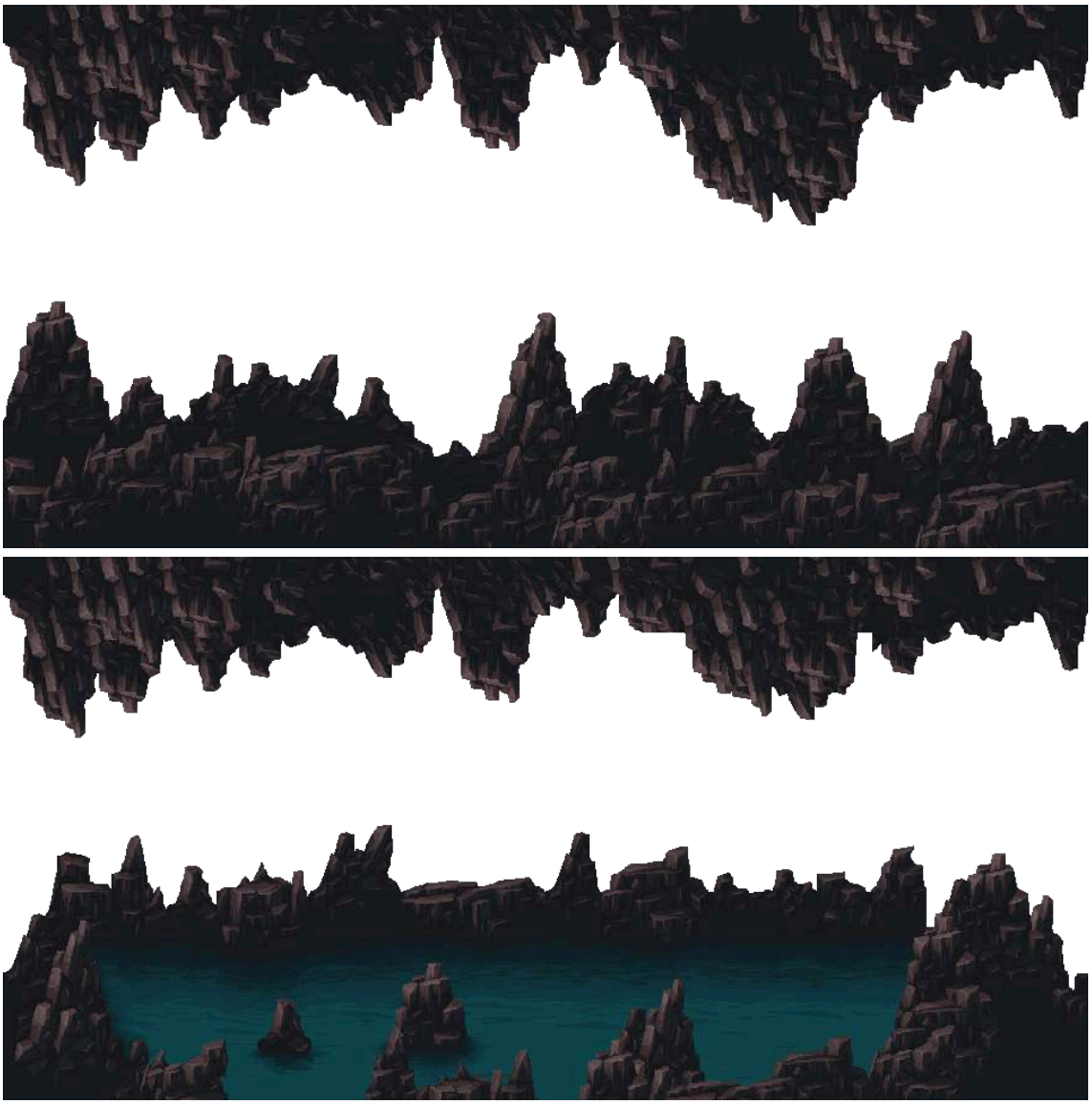
Podremos hacer un salto normal o doble salto con la tecla “ESPACIO”

Disparar rayos mágicos con la tecla “CONTROL”

Imagen Tile del mapa



Fondos decorativos para el mapa



2.1.2 Requisitos no funcionales

Las especificaciones de calidad y rendimiento dictaminan las condiciones bajo las cuales opera el software interactivo. Es decir, evalúan el comportamiento del sistema en lugar de describir sus mecánicas concretas.

1. **Tasa de refresco ininterrumpida:** El motor gráfico debe asegurar un rendimiento constante que nunca descienda de la barrera de los sesenta fotogramas por segundo, garantizando un dinamismo visual impecable.
2. **Accesibilidad e intuición:** La curva de aprendizaje ha de ser sumamente accesible. Esto implica diseñar un esquema de manejo tan amigable que permita a cualquier perfil de usuario disfrutar de la obra sin lidiar con frustraciones iniciales.
3. **Solidez del código:** La arquitectura del programa requiere una robustez absoluta para asegurar una sesión continúa, erradicando por completo la posibilidad de bloqueos críticos o salidas repentinas al escritorio.
4. **Compatibilidad de ecosistema:** El producto final necesita contar con optimización nativa para operar de manera eficiente en entornos informáticos basados en la plataforma de Microsoft.

5. **Transiciones ágiles:** Los periodos de procesamiento necesarios para cargar nuevos escenarios deben minimizarse al extremo, evitando así mermar la atención y el nivel de inmersión del espectador.
6. **Legibilidad del HUD:** Los indicadores visuales en pantalla precisan una distribución limpia y directa, permitiendo una lectura instantánea de las métricas vitales, el inventario de riquezas y los datos clave de la sesión.
7. **Persistencia de datos fiable:** El mecanismo de almacenamiento tiene que respaldar de forma infalible el historial de la partida.
8. **Latencia de entrada mínima:** La comunicación entre los periféricos y la pantalla debe ser prácticamente instantánea, asegurando que las órdenes manuales se traduzcan en acciones virtuales sin retardo perceptible (bajo input lag).
9. **Claridad compositiva:** La dirección de arte técnica debe estar enfocada en facilitar la identificación del entorno, logrando que el cerebro distinga de un solo vistazo las amenazas, las recompensas y las barreras arquitectónicas.

2.2 Tecnologías

2.2.1 Comparativa de las tecnologías valoradas

Godot Engine es un motor de desarrollo de videojuegos que permite crear tanto juegos en 2D como en 3D. En mi proyecto lo utilice para diseñar los niveles, programar cómo se mueve el personaje y añadir elementos como enemigos y demás.

Ventajas:

- Es gratuito y de código abierto.
- Resulta sencillo de aprender para proyectos pequeños.
- Está muy bien optimizado para el desarrollo en 2D.
- Incluye su propio lenguaje de programación, fácil de usar.

Desventajas:

- Tiene menos recursos y tutoriales que otros motores más conocidos.
- Puede quedarse corto en proyectos muy grandes o complejos.

GDScript es el lenguaje de programación propio de Godot, y es el que utilice para desarrollar el juego.

Ventajas:

- Está totalmente integrado en Godot y funciona de forma eficiente.
- Permite crear juegos sencillos con poco código.
- Facilita hacer pruebas rápidas durante el desarrollo.
- Incluye autocompletado, lo que ayuda a reducir errores.

Desventajas:

- Solo se utiliza dentro de Godot, por lo que tiene menos aplicaciones externas.
- Ofrece menos funcionalidades avanzadas que otros lenguajes más completos.
- Puede ser limitado en proyectos de gran escala.
- No es fácilmente reutilizable si se cambia de motor de desarrollo.

GitHub es una plataforma que sirve para almacenar proyectos y gestionar los cambios en el código.

Ventajas:

- Permite guardar copias del proyecto en la nube, evitando pérdidas de información.
- Da la opción de volver a versiones anteriores si ocurre algún error.
- Mantiene un historial completo de todas las modificaciones.
- Ayuda a organizar el proyecto mediante carpetas y archivos.
- Se integra fácilmente con editores de código y con Godot.

Desventajas:

- Al principio puede resultar complicado porque hay que aprender nuevos comandos.
- Es necesario tener conexión a internet para sincronizar los cambios.
- Si no se gestiona bien, pueden surgir conflictos entre archivos.
- Corregir errores al subir archivos puede requerir conocimientos adicionales.
- Puedes perder varios archivos si se interrumpe el proceso de la subida de archivos

2.2.2 Tecnologías escogidas

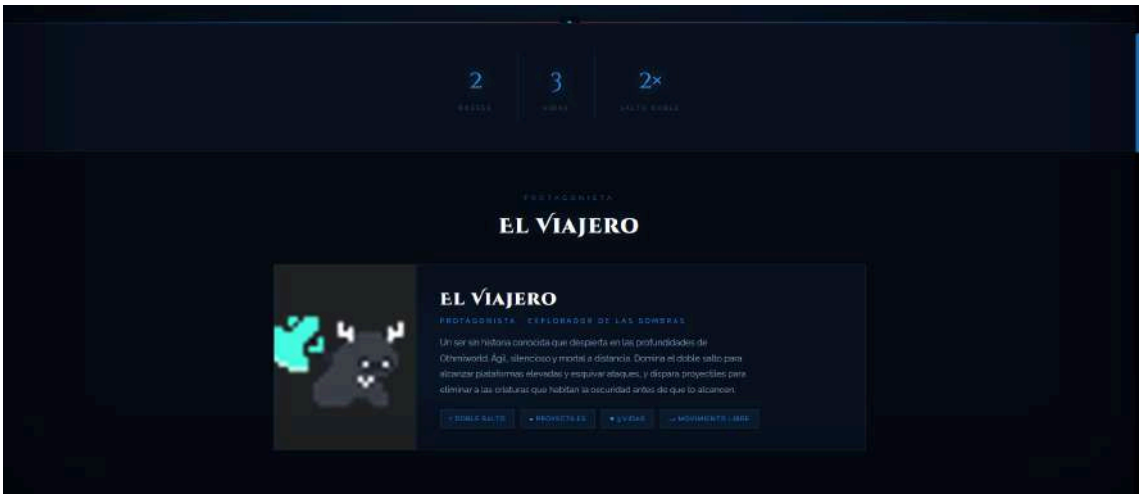
He utilizado el motor gráfico Godot Engine, en su versión 4.6, para el desarrollo del proyecto, ya que me ha permitido crear de forma eficiente tanto la jugabilidad como los distintos elementos del videojuego.

GDScript, el lenguaje que usa Godot

GitHub para almacenar la página web y el videojuego

Página web. Que contiene una explicación y contexto del videojuego, además de incluir un enlace de la página de itch.io para poder probar el videojuego en el navegador, sin necesidad de descargar nada

Página web:


2
3
2×
EL VIAJERO
EL VIAJERO


MECÁNICAS DE JUEGO
DOBLE SALTO
DISPARO DE PROYECTILES
SISTEMA DE VIDAS
DETECCIÓN ENEMIGA
APARICIÓN DE BOSSES
EXPLORACIÓN LIBRE


ENEMIGOS
ERRANTES DEL VACÍO
EL GÓLEM DE HIELO
EL ERIZO DE SOMBRA


LOS BOSSES
EL HERALDO
LA SOMBRA FINAL

2.3 Estructura del proyecto

El proyecto del videojuego Shadow Traveler está desarrollado en distintas secciones dentro de Godot Engine, donde cada una cumple una función específica para dar forma a una experiencia oscura, envolvente y coherente.

Personaje principal:
En este apartado se define al protagonista del juego, incluyendo su diseño visual, animaciones y la lógica que controla sus movimientos. El personaje puede desplazarse, saltar y explorar el entorno mientras interactúa con elementos del mapa. Además, sus habilidades están pensadas para adaptarse a un estilo de juego tipo metroidvania, donde el dominio del movimiento es clave.

Jefes y enemigos especiales:
En lugar de centrarse en objetos coleccionables, el juego introduce enfrentamientos contra enemigos desafiantes, especialmente jefes que poseen habilidades únicas. Algunos de estos jefes finales pueden teletransportarse, cambiar de posición inesperadamente o atacar desde distintos puntos del escenario, obligando al jugador a reaccionar rápidamente y aprender sus patrones.

Mundo y exploración:
El juego se desarrolla en un mapa amplio e interconectado con una estética oscura y misteriosa. No hay niveles separados como tal, sino un mundo continuo que el jugador puede recorrer libremente. A medida que avanza, desbloquea nuevas rutas y accede a zonas previamente inaccesibles, reforzando el enfoque metroidvania basado en la exploración y el backtracking.

Diseño Metroidvania:
Shadow Traveler se basa en una estructura no lineal, donde el progreso no depende de superar niveles concretos, sino de explorar, descubrir habilidades y abrir caminos dentro del mundo. Esto permite que cada jugador tenga una experiencia distinta según su forma de jugar.

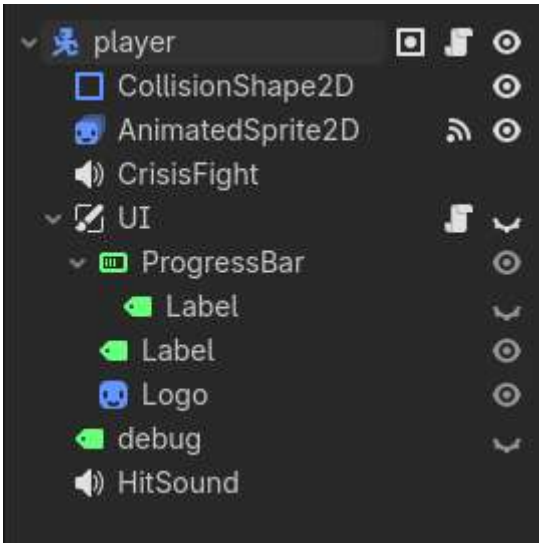
Menús del juego:
El juego cuenta con una interfaz clara y acorde a su ambientación. Incluye un menú principal desde donde iniciar la partida, continuar una sesión o salir del juego. También dispone de menús secundarios como pausa, inventario o mapa, que ayudan al jugador a orientarse dentro del mundo.

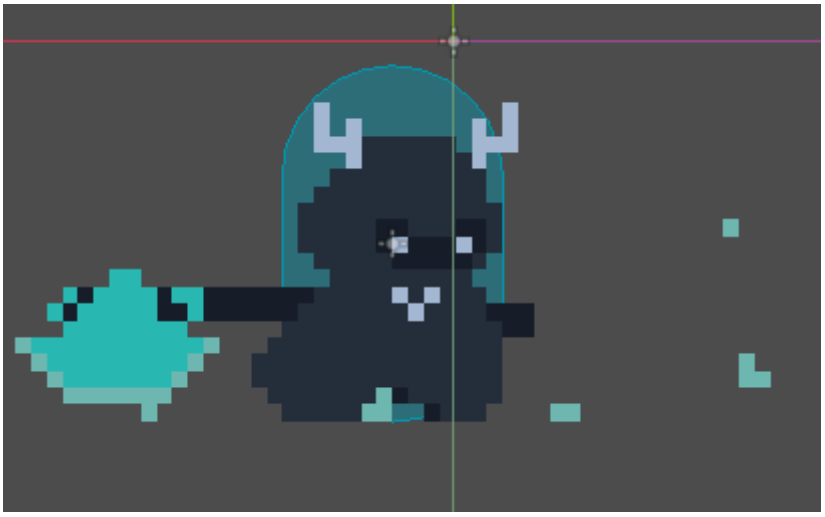
Ajustes:
Dentro de las opciones, el jugador puede configurar distintos aspectos como el volumen del sonido, la música, los controles o la calidad gráfica. Estos ajustes permiten adaptar la experiencia a las preferencias de cada usuario, manteniendo siempre la esencia inmersiva del juego.

2.4 Descripción de los componentes

2.4.1 Componente 1 Jugador

El jugador controla al personaje principal, que podrá moverse libremente por el mapa, saltar y realizar un doble salto para acceder a zonas elevadas o esquivar ataques enemigos. Además, contará con la capacidad de atacar disparando proyectiles para enfrentarse a los enemigos a distancia. Dispondrá de 3 vidas, por lo que deberá avanzar con cuidado, evitando daños y aprovechando sus habilidades de movimiento y ataque para superar los distintos obstáculos y enfrentamientos del juego.



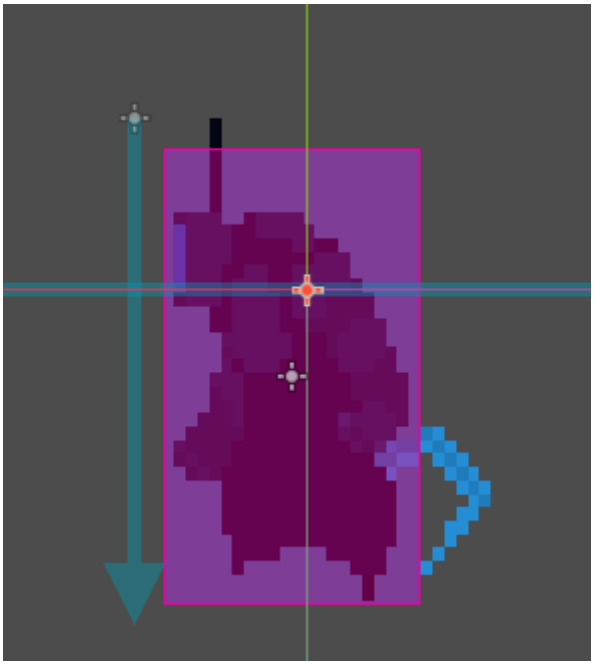


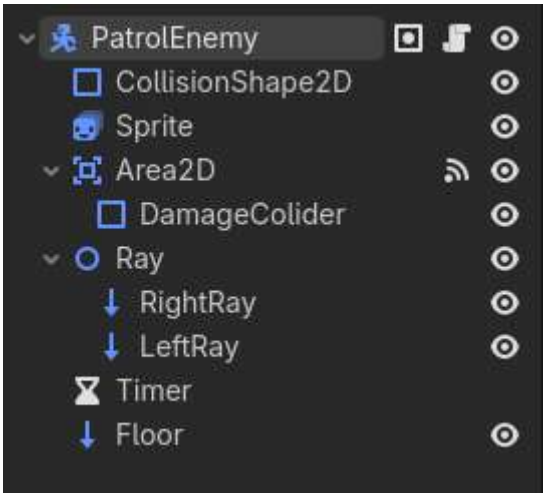
Estructura del jugador

2.4.2 Componente 2 Enemigos casuales

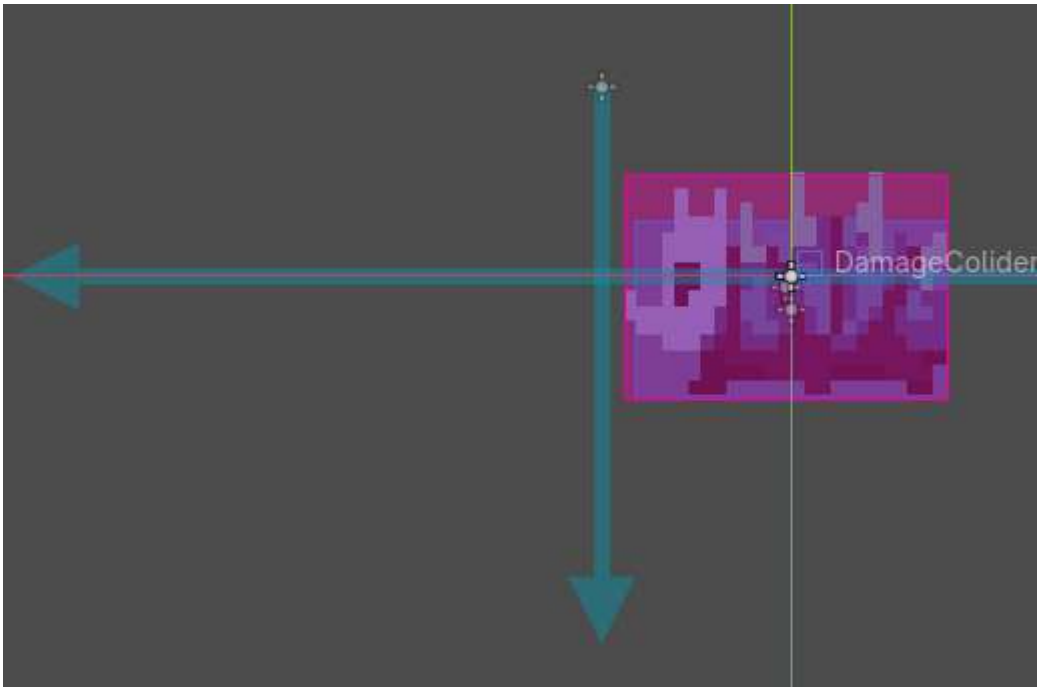
Estos enemigos atacarán al jugador y estarán repartidos por todo el mapa, apareciendo en diferentes zonas para dificultar la exploración. Habrá variedad de personajes, como un erizo, un golem o murciélagos, cada uno con comportamientos y ataques distintos, haciendo que el jugador tenga que adaptarse según el enemigo al que se enfrente.

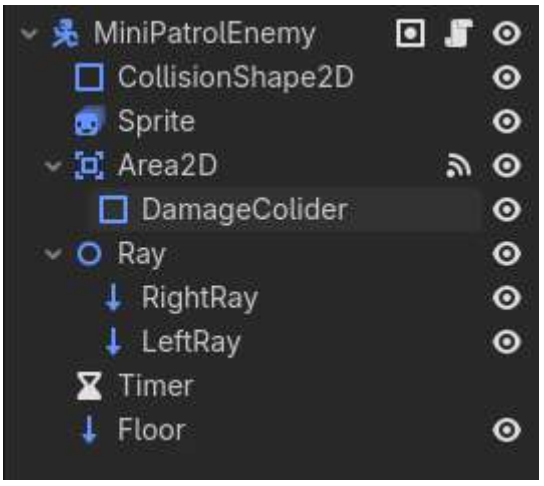
Golem:



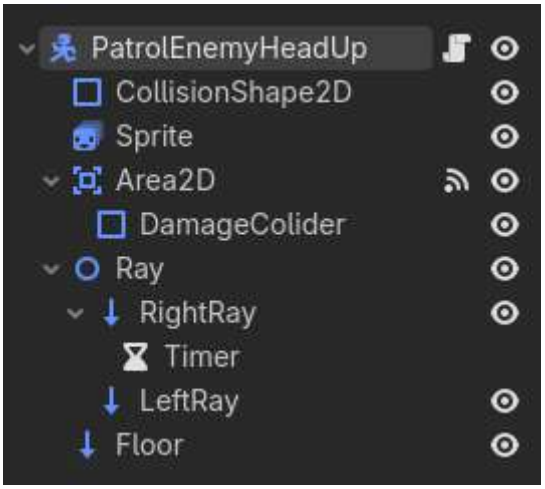


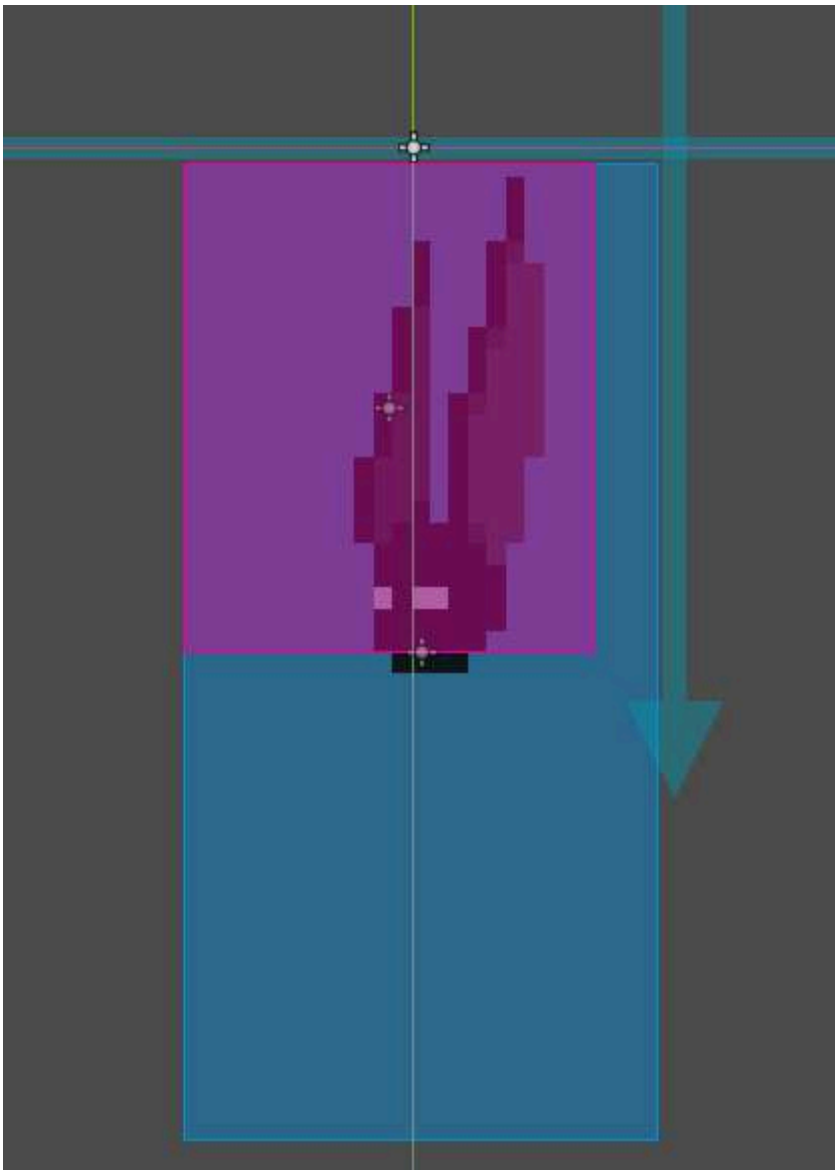
Erizo:


DamageColider


MiniPatrolEnemy
CollisionShape2D
Sprite
Area2D
DamageColider
Ray
RightRay
LeftRay
Timer
Floor

Murciélago:

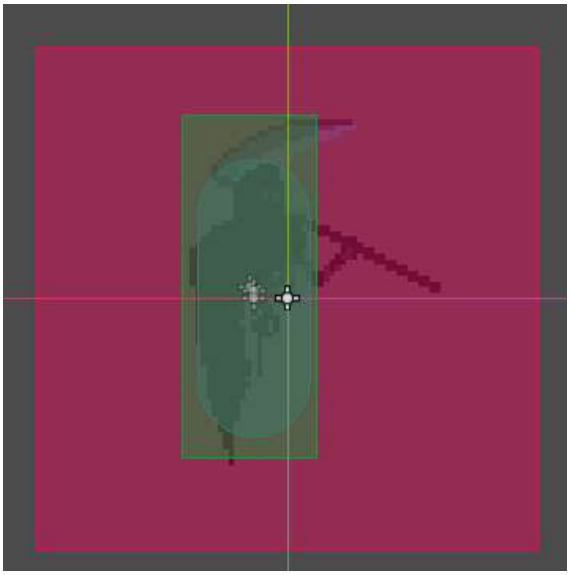

PatrolEnemyHeadUp
CollisionShape2D
Sprite
Area2D
DamageColider
Ray
RightRay
Timer
LeftRay
Floor

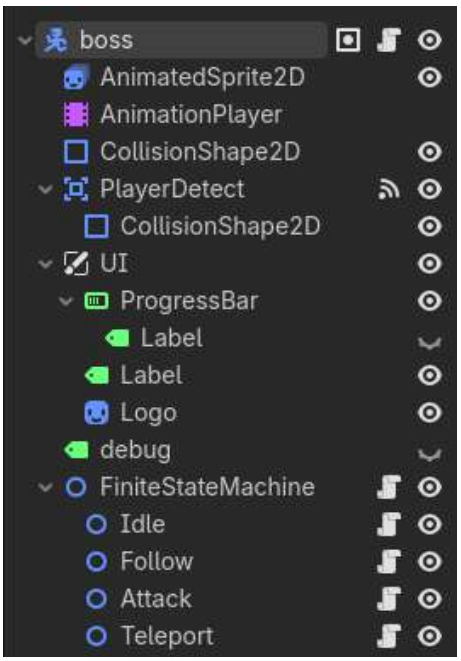


2.4.3 Componente 3 Jefes finales

Habrá 2 jefes finales que el jugador se encontrará por sorpresa mientras recorre el mapa. Aparecerán de forma inesperada en distintos momentos de la partida, haciendo que la exploración sea más tensa y dinámica. Cada uno tendrá características y formas de ataque diferentes, ofreciendo encuentros variados y obligando al jugador a adaptarse rápidamente a cada situación.

Boss 1:





Boss 2:


CollisionShape2D
DamageColider

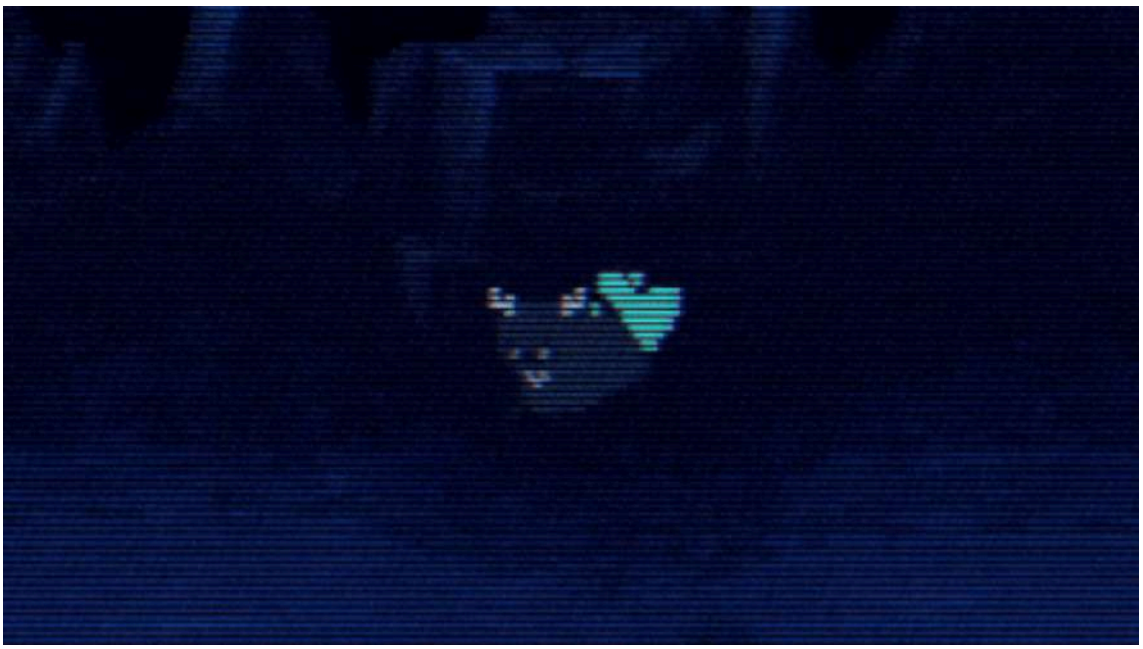

boss2
CollisionShape2D
AnimatedSprite2D
DamagePlayerArea
DamageColider
HitSound

2.5 Definición de las funcionalidades

El videojuego contará con diferentes funcionalidades diseñadas para ofrecer una experiencia dinámica, entretenida y con cierta dificultad para el jugador. El personaje principal podrá desplazarse libremente por el mapa, saltar y realizar un doble salto, lo que le permitirá acceder a zonas más elevadas, esquivar ataques enemigos y superar distintos obstáculos presentes durante la partida. Además, el jugador podrá atacar disparando proyectiles para derrotar a los enemigos desde la distancia. También dispondrá de 3 vidas, que irán disminuyendo cada vez que reciba daño. Si estas se agotan, la partida finalizará y el jugador deberá reiniciar desde el punto establecido. En cuanto a los enemigos, estarán distribuidos por todo el mapa y atacarán al jugador al detectarlo. Habrá variedad de personajes, como un erizo, un golem o murciélagos, cada uno con comportamientos y patrones de ataque diferentes, haciendo que el jugador tenga que adaptarse a cada enfrentamiento. Por último, el juego incluirá 2 jefes finales que aparecerán por sorpresa durante la exploración. Estos enfrentamientos serán más desafiantes que los combates normales y pondrán a prueba la habilidad del jugador, combinando estrategia, movimiento y precisión para poder superarlos.

2.5.1 Funcionalidad 1: Doble salto

Una de las habilidades principales del personaje es la capacidad de realizar un doble salto, lo que le permite desplazarse con mayor libertad por el escenario. Para ejecutar esta acción, el jugador debe pulsar la tecla espacio dos veces seguidas en un corto intervalo de tiempo. Al realizar el primer salto, el personaje se eleva normalmente, y al presionar espacio por segunda vez mientras está en el aire, activa un impulso adicional que le permite alcanzar una mayor altura o distancia. Esta mecánica resulta muy útil para superar obstáculos, cruzar zonas peligrosas, acceder a plataformas elevadas y explorar áreas que no serían accesibles con un salto normal. Además, el doble salto añade dinamismo al movimiento del personaje, haciendo que el control sea más ágil y estratégico durante toda la partida.



2.5.2 Funcionalidad 2: Morir

El personaje comienza cada partida con un total de 3 vidas, que representan las oportunidades que tiene antes de que la partida termine. Cada vez que el jugador entra en contacto con un enemigo, recibe daño de un ataque, pierde la mitad de una vida. El sistema está diseñado para aumentar la dificultad y obligar al jugador a actuar con cuidado, evitando errores y planificando mejor sus movimientos. Cuando las tres vidas se agotan por completo, el personaje muere automáticamente y se muestra el final de la partida, obligando al jugador a reiniciar desde el punto establecido o comenzar de nuevo. Este sistema añade tensión y emoción al juego, ya que cada vida perdida acerca al jugador al fracaso, haciendo que cada decisión durante la partida tenga una mayor importancia.

2.5.3 Funcionalidad 3: Disparar rayos mágicos

El personaje tiene la habilidad especial de lanzar rayos mágicos para atacar a los enemigos a distancia. Esta acción se activa pulsando la tecla Ctrl, lo que genera un proyectil que se desplaza en la dirección indicada y puede impactar contra los enemigos presentes en el escenario. El uso de rayos mágicos permite al jugador defenderse sin necesidad de acercarse demasiado al enemigo, ofreciendo una ventaja estratégica durante el combate. Esto resulta especialmente útil contra enemigos que se mueven rápidamente o que atacan cuerpo a cuerpo. Además, los enemigos cuentan con una barra de vida representada mediante corazones divididos por mitades, lo que permite visualizar de forma precisa el daño recibido. Cada impacto del rayo mágico reduce media unidad de corazón, hasta que el enemigo pierde toda su vida y es derrotado. Esta mecánica hace que los enfrentamientos sean más claros y permite al jugador calcular mejor cuántos ataques necesita para vencer a cada enemigo.

3 Otros capítulos

Durante el desarrollo de Shadow Traveler tuve que tomar varias decisiones importantes antes de empezar a programar. Para cada una valoré algunas opciones que tenía en mente y elegí la más adecuada según mi nivel y el tiempo disponible.

Estuve decidido desde el principio a usar Godot Engine 4.6 para este videojuego, ya que es un motor sencillo de usar y funciona bien en ordenadores menos potentes como los de clase.

Además, su lenguaje resulta sencillo de aprender. Me habría gustado crear un juego en 3D, pero al final lo desarrollé en 2D porque es más manejable y me permitirá terminar el proyecto correctamente.

En cuanto al contenido, al principio quería hacer varios niveles, pero al final decidí hacer un mapa bastante grande con una estética. Sobre las mecánicas, valoré la idea de añadir habilidades complejas, pero preferí simplificar el juego, centrándome en esquivar enemigos y avanzar hacia el siguiente boss, para garantizar que todo funcione bien.

Para el almacenamiento del proyecto, sabía que usaría GitHub, lo que me permitió organizar mejor el proyecto y guardar los avances de forma segura y fácil. Finalmente, decidí que al perder se vuelva al menú. Todas estas decisiones me ayudaron a crear un juego sencillo, funcional y acorde al tiempo disponible.

4 Conclusiones

4.1 Conclusiones generales del proyecto

Al haber hecho este videojuego en todo este curso, he podido aprender conocimientos bastante importantes a la hora de desarrollar un videojuego. Me han ayudado a entender mejor el motor gráfico, saber desarrollar videojuegos 2D, me puede ser útil para un futuro, por si me gustaría dedicarme a la creación de ellos ya que anteriormente he diseñado 3D usando Unreal Engine.

4.2 Consecución de los objetivos

- Diseñar la mecánica principal de movimiento del personaje. (Superado)
- Implementar un sistema de nivel. (Superado)
- Desarrollar enemigos comunes y jefes finales con patrones de ataque. (Superado)
- Implementar un sistema de vida, daño y derrota. (Superado)
- Crear una interfaz de usuario básica. (Superado)
- Documentar correctamente el proceso de desarrollo. (Superado)

4.3 Valoración de la metodología y planificación

El desarrollo del proyecto se ha desarrollado bastante estructurado. En la base inicial se establecieron las mecánicas principales que deberían salir en el proyecto, y a partir de ahí se fue construyendo el juego paso a paso. Comencé por el sistema de movimiento del personaje, seguido de los enemigos, posteriormente la interfaz y, por último, el diseño de los niveles.

Durante el proceso fue necesario realizar algunos cambios y ajustes, como por ejemplo al equilibrar la dificultad o al incorporar nuevos peligros. Estas modificaciones no supusieron un inconveniente, sino que forman parte habitual del desarrollo de un videojuego.

La metodología utilizada resultó adecuada para un proyecto de estas características, ya que permitió avanzar de forma progresiva y facilitó la detección de errores en fases tempranas.

4.4 Visión de futuro

El juego podría perfectamente ser muchísimo más largo, con más contenido adicional que me permitiría publicarlo en alguna plataforma grande de videojuegos. Se pueden añadir más cosas a este videojuego para que llegue a ser grande.

- **Más mapas y entornos**

  Se añadirán nuevos mapas con diseños diferentes, aumentando la variedad de escenarios y la dificultad de forma progresiva.

- **Nuevos enemigos y peligros**

  Se crearán enemigos con comportamientos más variados y trampas nuevas, haciendo cada nivel más único y desafiante.

- **Mejoras visuales y sonoras**

  Se trabajará en una mejor ambientación con efectos de sonido, música y animaciones más elaboradas para hacer la experiencia más inmersiva

5. Glosario

Gameplay o jugabilidad: Hace referencia a la experiencia del jugador dentro del juego, abarcando aspectos como el control del personaje, las diferentes acciones, la evasión de enemigos y la sensación general de fluidez durante la partida.

GDScript: Lenguaje de programación propio de Godot empleado para desarrollar la lógica del juego, incluyendo el movimiento del personaje, las colisiones y el sistema de vidas.

Git: Herramienta de control de versiones utilizada durante el desarrollo para registrar cambios en el proyecto y facilitar una organización eficiente del trabajo.

GitHub: Plataforma online donde se aloja el repositorio del juego, permitiendo su descarga, la colaboración entre desarrolladores y la gestión de versiones.

Godot Engine: Motor de desarrollo de videojuegos gratuito y de código abierto con el que se ha creado el videojuego, especialmente enfocado en el desarrollo de juegos en 2D.

Colisión: Sistema del motor Godot que detecta el contacto entre objetos, como cuando el jugador interactúa con monedas, enemigos o trampas, activando las acciones correspondientes.

Scene: Elemento fundamental de organización en Godot. Cada nivel, el menú principal y la interfaz del juego se estructuran como escenas independientes que se conectan entre sí.

Tilemap: Herramienta utilizada para diseñar los niveles mediante la colocación de pequeñas piezas gráficas, formando el terreno, plataformas y escenarios.

Hitbox: Zona invisible que rodea a personajes, enemigos y obstáculos, y que permite al motor detectar interacciones y aplicar efectos como daño.

AnimatedSprite: Nodo de Godot utilizado para representar animaciones en el juego, como las del personaje principal, los enemigos o elementos peligrosos como sierras, fuego o pinchos.

6. Bibliografía

A lo largo del desarrollo de este videojuego, he tenido que investigar en diferentes fuentes de información, apartados necesarios que debo saber para lograr mi objetivo. Como usar el motor gráfico, conseguir los sprites que aparecen en mi videojuego, sonidos, música, y demás.

Efectos visuales de mi videojuego:
https://godotshaders.com/shader/?orderby=date&order=DESC

Sonidos (ambientes, efectos…)
https://freesound.org/

Sprites (escenario, personaje principal, enemigos…)
https://itch.io/

Documentacion oficial del motor:
https://docs.godotengine.org/es/4.x/

Tu Primer PERSONAJE 2D en GODOT | TUTORIAL:
https://www.youtube.com/watch?v=6hj6PwndLJE&t=141s

Domina Godot 4: TileMaps en 10 min - Tutorial
https://www.youtube.com/watch?v=UFjmoGevs7s

Boss Fight with Finite State Machine - Godot 4
https://www.youtube.com/watch?v=zh5bwenMn9o&list=PLDvxSFN380vAdsbBU2dY8jlaQNgBIRxy9&index=2

Godot 4.3 Double Jump Tutorial || Weekly Godot Challenge #17
https://www.youtube.com/watch?v=DW4CQoYddXQ&t=1s

IAs usadas:

Google Gemini:
https://gemini.google.com/app?hl=es-ES

OpenAI ChatGPT:
https://chatgpt.com/es-ES/

7 Anexo

Scripts GD

Player:

```gdscript
extends CharacterBody2D

@export var speed = 150
@export var jump = -330
@export var bullet_node: PackedScene
@export var atacar = false
@onready var animated_sprite_2d = $AnimatedSprite2D

signal health_changed(current_health)
var max_health: float = 3.0
var current_health: float = 3.0

var jump_count = 0
var gravity = 950

func _ready() -> void:
	animated_sprite_2d.flip_h = true
	add_to_group("Player")
	current_health = max_health
	await get_tree().process_frame
	var hud = get_tree().current_scene.find_child("HUD", true, false)
	if hud:
		health_changed.connect(hud.actualizar_interfaz)
		hud.actualizar_interfaz(current_health)

func _physics_process(delta):
	# DISPARO
	if Input.is_action_just_pressed("shoot"):
		shoot()

	# GRAVEDAD
	if not is_on_floor():
		velocity.y += gravity * delta
	else:
		jump_count = 0
```

...

Boss de la Guadaña:

```gdscript
extends CharacterBody2D

@export var max_health: float = 35
var health: float = max_health

@onready var player = get_tree().get_first_node_in_group("Player")
@onready var animated_sprite = $AnimatedSprite2D
@onready var progress_bar = $UI/ProgressBar
@onready var hit_sound = $HitSound

var direction: Vector2

func _ready():
    if not player:
        player = get_tree().get_first_node_in_group("Player")
    set_physics_process(true)

func _process(_delta):
    if player:
        direction = player.global_position - global_position
        animated_sprite.flip_h = direction.x < 0

func _physics_process(_delta):
    if player:
        velocity = direction.normalized() * 80
        move_and_slide()

func take_damage(amount := 2.0):
    health -= amount
    print("Vida del boss:", health, "/", max_health)

    if progress_bar:
        progress_bar.value = health

    if hit_sound:
        hit_sound.play()
```

…

Boss de los cuernos:

```gdscript
extends CharacterBody2D

@export var max_health: float = 35
var health: float = max_health

@onready var player = get_tree().get_first_node_in_group("Player")
@onready var animated_sprite = $AnimatedSprite2D
@onready var health_control = $CanvasLayer/Control
@onready var hit_sound = $DamagePlayerArea/HitSound

var direction: Vector2

signal health_changed(value: float)

func _ready():
    if not player:
        player = get_tree().get_first_node_in_group("Player")
    connect("health_changed", Callable(health_control, "set_health"))
    emit_signal("health_changed", health / max_health)

func _process(_delta):
    if player:
        direction = player.global_position - global_position
        animated_sprite.flip_h = direction.x < 0

func _physics_process(_delta):
    if player:
        velocity = direction.normalized() * 60
        move_and_slide()

func take_damage(amount: float):
    health -= amount
    print("Vida del boss:", health, "/", max_health)

    emit_signal("health_changed", health / max_health)

    if hit_sound:
```

...

Enemigos casuales:

```gdscript
extends CharacterBody2D

const enemyrun = 60
const gravedad = 98

var esta_atacando: bool = false

@onready var area: Area2D = $Area2D
@onready var sprite: AnimatedSprite2D = $Sprite
@onready var ray_suelo: RayCast2D = $Floor

func _ready():
	velocity.x = -enemyrun
	sprite.play("Run")
	sprite.animation_finished.connect(_on_animation_finished)

func _physics_process(_delta):
	velocity.y += gravedad

	if not esta_atacando:
		if is_on_wall() or !ray_suelo.is_colliding():
			velocity.x *= -1

		if velocity.x < 0:
			sprite.flip_h = false
			area.scale.x = 1
			ray_suelo.position.x = -8
		elif velocity.x > 0:
			sprite.flip_h = true
			area.scale.x = -1
			ray_suelo.position.x = 8

		sprite.play("Run")
	else:
		velocity.x = 0
```

```
37      move_and_slide()
38
39  func _on_area_2d_body_entered(body: Node2D) -> void:
40      if body.is_in_group("Player") and not esta_atacando:
41          atacar(body)
42
43  func atacar(objetivo):
44      esta_atacando = true
45      sprite.play("Attack")
46
47      if objetivo.has_method("take_damage"):
48          objetivo.take_damage(0.5)
49
50  func _on_animation_finished():
51      if sprite.animation == "Attack":
52          esta_atacando = false
53          velocity.x = enemyrun if sprite.flip_h else -enemyrun
```