PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE HÁBITAT, INFRAESTRUCTURA Y CREATIVIDAD

INGENIERÍA DE SISTEMAS DE INFORMACIÓN

Trabajo de Titulación

Desarrollo de una Demo de Videojuego de Aventura de Plataformas en 2D en el Motor de Videojuego Godot y Manual del Proceso de Elaboración del Juego

AUTOR:

Richard Darío Enríquez Pérez

QUITO DM, 2025

Dedicatoria

El presente trabajo de titulación se encuentra dedicado especialmente a mis padres y mi hermana, que me han apoyado de manera incondicional para que obtuviera la mejor educación, inspirándome a no rendirme en mi proceso de aprendizaje y enseñanzas, luchar para conseguir las metas que busco obtener.

También dedico este proyecto a los amigos y profesores que conocí durante todo mi proceso de estudios, que han llegado a contribuir con su conocimiento en mi formación como profesional y como persona.

Agradecimiento

Quiero agradecer a todos los miembros mi familia, desde mis padres, mi hermana, mis tios, tias y primos, hasta los miembros que ya no se encuentran conmigo en esta vida, todos ellos han llegado a formar una parte importante de mi vida y sin su apoyo, no llegaría a ser la persona que soy en este momento, por lo que siempre estaré agradecido con ellos y buscare no decepcionarlos.

También agradezco a los profesores que han compartido su conocimiento y experiencias conmigo en sus clases y que me han impulsado a querer seguir aprendiendo sobre la ingeniería en sistemas.

Finalmente, también agradezco a los compañeros que he conocido a lo largo de la carrera y que algunos han llegado a convertirse en grandes amigos con los que se que puedo contar en todo momento.

Abstract

This thesis project aimed to develop a demo of a 2D adventure-platform video game using the Godot Engine, as well as to create a manual that documents the game development process in a practical and structured way. The project focused on the use of open-source tools to demonstrate their technical feasibility and accessibility for independent developers and students interested in video game development.

For development management, the Scrum agile methodology was applied, adapted to a one-person work environment. This allowed for organizing activities into iterations, prioritizing essential functionalities, and progressively achieving playable increments. Based on the requirements analysis, the main mechanics of the video game were defined, including character control, interaction with the environment, object collection, level management, and the implementation of a functional graphical interface.

The video game was developed entirely in two dimensions, taking into account technical and design advantages that facilitated the implementation of physics, collisions, and animations, as well as allowing for more efficient development in line with the project's scope. As a result, a functional demo was exported for Windows operating systems, which meets the established functional and non-functional requirements.

Finally, the game development manual documents the development, configuration, and export processes step by step, supported by screenshots and clear descriptions, serving as a reference guide for new developers who wish to begin developing 2D video games using the Godot Engine.

Tabla de contenido

Capítulo 1: Introducción

1.1. Justificación

El crecimiento del desarrollo independiente de videojuegos exige recursos formativos que sean a la vez prácticos y acotados al tiempo y capacidades de los creadores indie. Aunque Godot Engine y otras herramientas open-source proporcionan funcionalidades potentes y gratuitas, la información disponible suele encontrarse fragmentada entre documentación oficial, foros y tutoriales de variable calidad, lo que dificulta la asimilación rápida por parte de desarrolladores principiantes. Ante esta realidad, resulta pertinente generar materiales que demuestren de forma concreta cómo superar obstáculos técnicos frecuentes en proyectos de dos dimensiones básicos.

Por ello, este trabajo plantea la construcción de una demo de un videojuego de aventura de plataformas en 2D que implemente mecánicas esenciales: control de personaje, físicas básicas y detección de colisiones, recolección de objetos como condición de avance, y una interfaz mínima, que incluye un menú de inicio, contador de objetos y pantalla de nivel completado. La elección de estas funcionalidades responde a su presencia recurrente en prototipos indie y a su utilidad pedagógica para enseñar patrones de diseño y solución de problemas técnicos en Godot.

Como aporte académico y práctico, el demo irá acompañado de un manual que documente, paso a paso, el proceso de desarrollo: desde la creación del proyecto y la configuración inicial en Godot hasta la implementación de sus principales sistemas. Este documento servirá como documentación de referencia que permita comprender las etapas y decisiones técnicas del proyecto, y a la vez funcione como material de apoyo para usuarios que deseen replicar o adaptar la experiencia.

En conjunto, este proyecto busca demostrar la viabilidad del desarrollo de un videojuego 2D completo utilizando únicamente herramientas y recursos libres, al mismo tiempo que ofrece un material formativo concreto para nuevos desarrolladores. Así, el trabajo contribuye a fortalecer la adopción de tecnologías open-source dentro del ámbito educativo y profesional del desarrollo de videojuegos independientes.

1.2. Planteamiento del Problema

El desarrollo de videojuegos independientes ha adquirido una relevancia creciente en la industria de los videojuegos, al convertirse en un espacio donde la creatividad y la innovación se manifiestan sin las restricciones de grandes estudios en cuanto al contenido que deben desarrollar, equipo que manejan y las limitaciones que enfrentan, tanto en tiempo de desarrollo, como en recursos económicos. No obstante, los desarrolladores independientes suelen enfrentarse a su manera a estas limitaciones, tanto económicos como técnicas, que dificultan la creación de proyectos funcionales y de calidad. Si bien existen motores de videojuegos de código abierto, que ofrecen las herramientas necesarias para que los usuarios puedan trabajar libremente y de la misma manera de como si se utilizaran software privado, su aprovechamiento óptimo requiere en ciertos casos de conocimientos previos y de documentación práctica que oriente a los desarrolladores en la implementación de mecánicas básicas, especialmente cuando se busca construir juegos de diversas categorías en dos dimensiones de manera estructurada y eficiente.

En la actualidad, la información disponible sobre esta clase de motores de videojuegos se encuentra dispersa entre documentación técnica, foros y tutoriales no oficiales, lo que genera una curva de aprendizaje elevada y procesos poco estandarizados para quienes quieren iniciar en el desarrollo de videojuegos, buscan ampliar sus conocimientos en esta clase de motores o migrar sus proyectos realizados a nuevos motores para mejorar su rendimiento. Esta situación limita la adopción de motores open-source en contextos académicos y en proyectos personales de bajo presupuesto. Por tanto, se identifica la necesidad de contar con un material aplicado que demuestre, de forma clara y documentada, cómo construir un prototipo jugable completo utilizando únicamente herramientas libres.

En respuesta a esta problemática, el presente trabajo planteo el desarrollo de un demo de videojuego de aventura de plataformas en 2D utilizando el motor de videojuegos de código libre Godot Engine, junto con un manual del proceso de elaboración que documente su proceso de construcción, permitiendo a nuevos desarrolladores comprender y replicar las bases del desarrollo indie con recursos accesibles y software libre, lo que permitirá a los desarrolladores aprovechar plenamente el potencial de esta clase de motores y mejorar significativamente sus proyectos sin la necesidad de gastar recursos económicos que a su momento no puedan permitirse.

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar una demo de un videojuego de aventura de plataformas en 2D para sistemas Windows utilizando el motor de videojuegos Godot Engine, acompañado de manual del proceso de elaboración que documente el proceso de diseño e implementación de sus principales componentes, con el fin de facilitar el aprendizaje y la replicación del desarrollo del videojuego en entornos de software libre.

1.3.2. Objetivos Específicos:

- Analizar las características y herramientas principales que ofrece Godot Engine para el desarrollo de videojuegos 2D
- Determinar la metodología adecuada para el desarrollo de la demo.
- Diseñar la estructura general de la demo para que funcione en la plataforma de Windows.
- Diseñar el contenido de la demo con 3 niveles con un límite de tiempo de 5 minutos por nivel, con la mecánica de recolección de objetos para finalización de cada nivel y con los recursos visuales necesarios.
- Documentar el proceso de desarrollo mediante un manual que describa los pasos esenciales para la elaboración de la demo en Godot.

1.4. Metodología

1.4.1. Tipo de Investigación

Se realizo una investigación preliminar para determinar metodologías adecuadas para el desarrollo de proyectos similares en el ámbito de los videojuegos, encontrando un capítulo de un libro llamado “Agile Game Development with SCRUM” escrito por el entrenador certificado de SCRUM Clinton Keith, donde explica todas sus experiencias trabajando con esta metodología ágil dentro del ámbito del desarrollo de un videojuego de Microsoft, como manejo la división de tareas, la colaboración entre los miembros del equipo, el manejo de las entregas y las situaciones donde miembros del equipo abandonan el proyecto, de esa misma manera, también se fue encontrando enfoques de varias compañías de videojuegos conocidas en la industria,, como es el caso de Valve Corporation, conocido por la saga de videojuego que crearon con el nombre de “Half-Life” y Riot Games, conocido mundialmente

por el juego multijugador de “League of Legends”, ambas empresas crearon su propio modelo de trabajo basándose en metodologías agiles para la entrega incremental y flexible de resultados de productos que cumplan con sus objetivos.

En base a lo recopilado anteriormente, la utilización de una metodología ágil favorece a la elaboración de esta clase de proyectos, donde lo que se busca son los siguientes puntos:

1. Flexibilidad: La metodología ágil se enfoca en la adaptación rápida a los cambios en los requerimientos del proyecto, lo cual resulta fundamental en entornos creativos y técnicos donde las ideas, mecánicas y elementos visuales pueden evolucionar durante el proceso de producción. Esta capacidad de adaptación permite ajustar las funcionalidades, la jugabilidad o el diseño de niveles conforme se identifican mejoras o nuevas oportunidades durante el desarrollo.
2. Enfoque en la colaboración: Al involucrar a integrantes de distintas áreas, como programación, diseño, arte y sonido, se fomenta la comunicación efectiva y el trabajo interdisciplinario. Este enfoque colaborativo asegura que cada componente del proyecto se integre de manera coherente, alineando los objetivos técnicos y creativos para lograr una experiencia de juego sólida y equilibrada.
3. Iteración rápida: La metodología ágil se caracteriza por ciclos de trabajo cortos y constantes, lo que facilita la creación de versiones jugables o prototipos en etapas tempranas. Estas iteraciones permiten obtener retroalimentación inmediata sobre las mecánicas, el rendimiento o la experiencia del jugador, posibilitando realizar ajustes precisos antes de avanzar a fases más complejas.
4. Mejora continua: A través de la evaluación constante y la revisión de cada entrega parcial, la metodología ágil impulsa la mejora progresiva del proyecto. Esto garantiza que el producto final evolucione con base en la retroalimentación de pruebas internas o de los usuarios, optimizando tanto la calidad técnica como la experiencia interactiva que ofrece el juego.

En el caso de este proyecto, se realizó un análisis respectivo entre las metodologías agiles SCRUM y Programación Extrema (XP) antes del inicio del desarrollo de la demo para determinar cuál metodología funcionará de mejor manera en este caso en especial y como llegará a ser aplicada durante todo el ciclo de desarrollo del proyecto.

1.5. Alcance

El presente trabajo de titulación comprende el diseño, desarrollo e implementación de una demo de un videojuego de aventura de plataformas en 2D utilizando el motor Godot Engine en Windows, junto con la elaboración de un manual de elaboración que documente los principales pasos del proceso de construcción. El demo incluirá entre tres niveles de los 7 previstos con mecánicas de movimiento, salto, detección de colisiones, junto a una mecánica de recolección de objetos para la progresión entre niveles, priorizando la funcionalidad sobre la complejidad artística o narrativa. El manual se limitará a describir los procedimientos técnicos esenciales para replicar el proyecto, empleando únicamente recursos de libre uso. El trabajo no contempla la creación de cinemáticas, inteligencia artificial avanzada, integración multijugador ni optimización comercial, centrándose exclusivamente en la demostración técnica y educativa del desarrollo 2D en Godot.

Capítulo 2: Marco Teórico

2.1. Historia y Evolución de los Videojuegos en 2D

La aparición del primer videojuego se lo puede atribuir al año de 1952 con el juego de tres en raya, denominado “OXO” desarrollado por Alexander S. Douglas para la computadora EDSAC, de igual manera, en 1958 “sirviéndose de un programa para el cálculo de trayectorias y un osciloscopio” (Facultat d'Informàtica de Barcelona, 2008) William Higginbotham creo el juego de Tennis for Two, considerado el primer juego que permitía a dos personas jugar entre sí. Debido a ese avance, en años posteriores se fueron creando juegos similares, como el caso de Spacewar, que utilizo gráficos vectoriales para la simulación del movimiento de naves espaciales que se enfrentaban entre ellas. Luego de estos avances, en la década de los 70, la empresa estadounidense Atari lanzo su versión comercial para máquinas de arcades el juego de Pong, que para algunas personas fue considerada la versión comercial de Tennis for Two e incluso causo problemas de demandas por consideraciones de plagio, pero para el fundador de la empresa Nolan Bushnell fue el éxito que marco a su empresa y la ubico en el mapa, para Bushnell “No pensé que sería un éxito comercial…básicamente preví el advenimiento de la industria de los videojuegos. Sabía que iba a ser enorme” (Baig, 2022).

Para la década de los 80, los videojuegos avanzaron con su imagen de jugabilidad en 8 bits y generaron gran popularidad para las máquinas recreativas y los negocios que manejaban estos equipos, el primer juego destacado de esta época es de la empresa japones Namco, con el juego de Pac-Man, que gano mucha popularidad por la simplicidad de su jugabilidad y diseño, creado por el diseñador japones Toru Iwatani, ha declarado en varias entrevistas que la intención de crear este juego fue que “A fines de la década de 1970, había muchos juegos en las salas de juegos que presentaban matar alienígenas u otros enemigos que en su mayoría atraían a los niños para jugar... Quería cambiar eso introduciendo máquinas de juego en las que aparecieran personajes lindos con controles más simples que no intimidaran a las clientas y parejas para que las probaran ... y las parejas que visitan las salas de juegos aumentarían” (Matt, 2015). Para el año de 1985, la popularidad de las máquinas recreativas había caído lentamente debido a la popularización de las consolas de videojuegos para el hogar, empresas como Nintendo y Sega fueron las empresas dominantes en empezar a generar ganancias en la industria de los videojuegos, y el juego que gano más popularidad que Pac-Man en esa época fue Super Mario Bros, lanzado en 1985 por Nintendo, revoluciono los

juegos al pasar de escenarios repetitivos en bucles sin objetivos fijos a mundos variados que contaban con un final después de varias horas de juego.

2.2. Motores de Videojuegos

Se puede definir a los motores de Videojuego como una suite de herramientas y librerías de programación que permiten a los desarrolladores crear, diseñar y programar videojuegos de manera más eficiente y efectiva (Grupo ARGOsoft - Consultores en Tecnología y Marketing Digital en El Salvador, 2024). Llegando a proporcionar una vasta gama de funciones que permiten a los desarrolladores enfocarse en aspectos creativos y en la jugabilidad que tiene planeado crear para los jugadores en lugar de tener que crear todo lo que utilizaría desde cero.

Estos motores suelen estar compuestos de la siguiente manera:

- Motor gráfico o de renderizado. Dependiente para mostrar las imágenes en la pantalla en diferentes dimensiones, pero también en todo lo relacionado con las texturas y la iluminación.
- Motor de sonidos. Mantiene una importancia similar a la del motor gráfico y se encarga de sincronizar el sonido, cargar las diferentes pistas o modificarlas a su gusto.
- Física. Imprescindible a la hora de dar naturalidad al juego y de que sus elementos interaccionen entre ellos de forma realista. También controla las colisiones que se producen en el juego.
- Scripting. Se centra en el funcionamiento tanto de los personajes del juego como del resto de objetos que aparecen. El programador debe contar con la opción de poder crear sus propios scripts o modificar los del motor, para poder aplicar todas las ideas que tiene (UNIR - Universidad Internacional de La Rioja, 2023).
- Inteligencia artificial. Empieza a estar presente en la inmensa mayoría de los videojuegos que apuestan por incluirla como un valor añadido.

En la industria del desarrollo de videojuegos, los motores se dividen principalmente en dos categorías: motores privados o comerciales y motores de código abierto. Ambos modelos ofrecen ventajas y limitaciones que influyen directamente en la accesibilidad, el costo y la libertad de desarrollo, siendo aspectos que muchas personas consideran al momento de seleccionar un motor de videojuego, tanto para el desarrollo del mismo como para el diseño

gráfico. A continuación, se presenta un cuadro comparativo que resume las principales diferencias entre ambos enfoques:

Tabla 1

Cuadro comparativo entre motores de videojuegos privados y código libre

Criterio de comparación	Motores de videojuegos privados	Motores de videojuegos de código libre
Licencia y acceso al código fuente	El uso está sujeto a licencias comerciales o planes de suscripción.	Los usuarios pueden modificar, adaptar y redistribuir el software según su licencia
Costo de uso	Requieren pagos por licencia, regalías o suscripción mensual/anual.	Generalmente gratuitos, sin regalías ni costos de licencia.
Soporte técnico	Cuentan con soporte profesional oficial y asistencia directa al cliente.	El soporte se basa en la comunidad de usuarios, foros y desarrolladores voluntarios.
Flexibilidad y personalización	Las modificaciones profundas pueden estar restringidas por la licencia.	Los desarrolladores pueden adaptar el motor a sus necesidades
Rendimiento y estabilidad	Suelen ofrecer un alto rendimiento, probado en producciones comerciales de gran escala.	Varía según la madurez del proyecto; algunos alcanzan un rendimiento comparable a los motores comerciales.
Comunidad y ecosistema	Amplio ecosistema comercial con recursos oficiales.	Comunidad abierta y colaborativa que comparte recursos de forma gratuita.
Actualizaciones y mantenimiento	Actualizaciones regulares controladas por la empresa propietaria.	Las actualizaciones dependen del aporte de colaboradores y fundaciones.
Propiedad intelectual de los juegos creados	En algunos casos, el desarrollador debe compartir ingresos o atribuir el uso del motor.	El creador conserva la totalidad de los derechos sobre el juego.

Nota. La tabla muestra las diferencias entre los motores de videojuegos privados y los motores de videojuego de código libre para comprender cual categoría escoger.

2.2.1. Godot Engine

Godot Engine es un motor de juego versátil, multifuncional y multiplataforma diseñado para crear juegos 2D y 3D desde una interfaz unificada. Proporciona una amplia gama de herramientas comunes, lo que permite a los usuarios centrarse en la creación de sus juegos sin necesidad de reinventar la rueda. Los juegos pueden exportarse de manera sencilla a una variedad de plataformas, incluidas las principales plataformas de escritorio como lo son Linux, macOS y Windows, plataformas móviles siendo el caso de Android, iOS, así como plataformas basadas en la web y consolas.

Godot es completamente gratuito y de código abierto bajo la permisiva licencia MIT. Sin ataduras, sin regalías, nada. Los juegos creados son propiedad total de los usuarios, hasta la última línea de código del motor. El desarrollo de Godot es completamente independiente y está impulsado por la comunidad, lo que empodera a los usuarios para ayudar a moldear el motor según sus expectativas. Es respaldado por la organización sin fines de lucro Godot Foundation (Juan et al., 2014).

La decisión de utilizar Godot Engine viene en base a las características provenientes del sitio web oficial como:

- Diseño intuitivo basado en escenas: Godot organiza los proyectos mediante un sistema jerárquico de escenas y nodos, lo que permite estructurar los elementos del juego de manera lógica y reutilizable.
- Herramientas de programación: El motor permite programar utilizando su propio lenguaje, GDScript, inspirado en Python, además de ofrecer soporte para C#, C++ y visual scripting, brindando flexibilidad a los desarrolladores a elegir el método que mejor se adapte a su nivel técnico o al tipo de proyecto.
- Motor 3D simple pero potente: Godot incluye un sistema 3D optimizado que incorpora renderizado físico, luces dinámicas, sombras en tiempo real y compatibilidad con materiales avanzados.
- Flujo de trabajo especializado para 2D: A diferencia de otros motores que adaptan el entorno 3D para el trabajo 2D, Godot cuenta con un motor 2D completamente independiente. Esto garantiza un rendimiento eficiente, colisiones precisas, efectos

visuales suaves y herramientas específicas para animación y física bidimensional, ideales para juegos de plataformas o aventura.

- Compatibilidad multiplataforma: Los juegos creados en Godot pueden exportarse a distintas plataformas, incluyendo Windows, Linux, macOS, Android, iOS y Web. El sistema de exportación simplifica la distribución, manteniendo la integridad del proyecto original sin necesidad de configuraciones adicionales.

Estas características permiten mostrar como Godot Engine es una muy buena opción para desarrollar proyectos de desarrollo de videojuegos.

2.2.2. Unity Engine

Unity Engine es una plataforma de desarrollo de videojuegos ampliamente utilizada en la industria, reconocida por su versatilidad, escalabilidad y enfoque multiplataforma. Permite crear juegos 2D, 3D y experiencias interactivas desde una interfaz unificada, con herramientas integradas que cubren desde la programación hasta el diseño visual y la simulación física. Unity facilita la exportación de proyectos a una amplia gama de plataformas, incluyendo Windows, macOS, Linux, Android, iOS, consolas y navegadores web mediante WebGL.

Unity se distribuye bajo un modelo de licencia comercial con opciones gratuitas para desarrolladores independientes y estudiantes, y planes de suscripción para estudios profesionales. Aunque no es de código abierto, ofrece acceso a una gran cantidad de documentación, recursos oficiales y extensiones a través de su Asset Store. Su comunidad global y su ecosistema de aprendizaje lo convierten en una herramienta accesible y poderosa para desarrolladores de todos los niveles (Unity Technologies, 2025).

La decisión de utilizar Unity Engine se basa en las siguientes características destacadas:

- Entorno de desarrollo visual y modular: Unity organiza los proyectos mediante una jerarquía de objetos en escenas, lo que permite estructurar los elementos del juego de forma lógica y reutilizable.
- Lenguaje de programación moderno: Unity utiliza C# como lenguaje principal, ampliamente documentado y compatible con prácticas de programación orientada a objetos.

- Motor 3D robusto: Incluye sistemas avanzados de renderizado, iluminación global, sombras dinámicas, simulación física y efectos visuales compatibles con estándares de la industria.
- Herramientas para desarrollo 2D: Aunque originalmente centrado en 3D, Unity ha incorporado herramientas específicas para 2D como Tilemap, Sprite Renderer, Animator y Cinemachine adaptado a 2D, lo que permite crear juegos bidimensionales con precisión y eficiencia.
- Compatibilidad multiplataforma: Unity permite exportar proyectos a más de 20 plataformas, incluyendo Windows, Android, iOS, WebGL, PlayStation, Xbox y Nintendo Switch, con configuraciones adaptables para cada entorno.
- Integración de inteligencia artificial: En versiones recientes, Unity ha incorporado herramientas de IA generativa para acelerar la creación de entornos, personajes y animaciones (Petrov, 2024).

Estas características posicionan a Unity como una opción sólida para el desarrollo de videojuegos tanto en entornos académicos como profesionales, especialmente cuando se busca una solución con amplio soporte, documentación y capacidad de expansión.

2.2.3. Unreal Engine

Unreal Engine es un motor de videojuegos desarrollado por Epic Games, reconocido por su potencia gráfica, capacidad de renderizado en tiempo real y uso extendido en la industria de gran renombre. Está diseñado para crear experiencias inmersivas en 3D, simulaciones complejas y mundos abiertos de gran escala, aunque también permite el desarrollo de juegos 2D mediante herramientas específicas. Este motor es compatible con múltiples plataformas, incluyendo Windows, macOS, Linux, Android, iOS, consolas y sistemas de realidad virtual.

El motor se distribuye de forma gratuita para desarrolladores, con un modelo de regalías del 5% sobre ingresos comerciales que excedan un umbral determinado. Su código fuente está disponible en GitHub para usuarios registrados, lo que permite personalizaciones profundas en proyectos avanzados. Unreal Engine destaca por su enfoque en calidad visual, rendimiento y herramientas de producción profesional (Epic Games, 2025).

La decisión de utilizar Unreal Engine se fundamenta en las siguientes características:

- Sistema de renderizado de última generación: Unreal Engine 5.6 incorpora tecnologías como Nanite (geometría virtualizada) y Lumen (iluminación global

dinámica), que permiten crear entornos altamente detallados sin comprometer el rendimiento (Tereshchuk, 2025).

- Lenguajes de programación y scripting visual: El motor permite programar en C++ para control total del rendimiento, y ofrece Blueprints, un sistema visual de scripting que facilita el prototipado sin necesidad de escribir código.
- Simulación física avanzada: Incluye sistemas de colisiones, físicas de cuerpos rígidos, partículas, fluidos y destrucción dinámica, ideales para juegos con interacción compleja.
- Herramientas para animación y cinematografía: Unreal ofrece herramientas como Sequencer, Control Rig y MetaHuman para crear animaciones realistas, cinemáticas y personajes detallados.
- Compatibilidad multiplataforma: Los proyectos pueden exportarse a Windows, consolas, dispositivos móviles, navegadores y entornos de realidad aumentada y virtual.
- Flujo de trabajo profesional: El motor está optimizado para equipos grandes, integración con pipelines de producción, control de versiones y colaboración en tiempo real.

Aunque Unreal Engine está más orientado a proyectos de gran escala y alto presupuesto, sus herramientas visuales y su rendimiento lo convierten en una opción viable para desarrolladores que buscan calidad gráfica y control técnico avanzado.

2.3. Géneros de Juegos

Los videojuegos fueron categorizados según el estilo, interacción, objetivos y mecánicas que manejan los juegos, permitiendo clasificarlos según las experiencias y desafíos que enfrentaran los jugadores, desde el foco en la exploración, la lógica o la toma de decisiones hasta la actuación rápida del jugador por medio de sus reflejos, llegando a haber combinaciones entre géneros, innovando el panorama de los videojuegos y las opciones de los desarrolladores al llevar a cabo este tipo de proyectos.

De los géneros existentes, los que son conocidos por la mayoría de personas son las siguientes:

- Acción
- Aventura
- Estrategia
- Rol
- Simulación
- Plataforma
- Terror

Para comprender de mejor manera las diferencias que existen entre estas categorías de videojuegos se procede a realizar una tabla donde se muestren dichas diferencias:

Tabla 2

Diferencias entre Géneros de Videojuegos

Género	Objetivo principal	Mecánicas clave	Ejemplos
Acción	Superar enemigos y desafíos mediante reflejos rápidos.	Combate, disparos, saltos y esquivas.	*Doom, Bayonetta*.
Aventura	Explorar y resolver acertijos dentro de una narrativa.	Interacción con objetos y personajes.	*The Legend of Zelda, Life is Strange*.
Estrategia	Planificar y tomar decisiones para vencer o administrar recursos.	Gestión de unidades y recursos.	*Age of Empires, StarCraft*.

Género	Objetivo principal	Mecánicas clave	Ejemplos
Plataforma	Superar niveles con saltos y precisión.	Movimiento lateral y control de ritmo.	*Super Mario Bros.*, *Celeste*.
Rol (RPG)	Desarrollar personajes mediante experiencia y decisiones.	Combate, progresión y personalización.	*Skyrim*, *Final Fantasy*.
Simulación	Recrear experiencias reales o ficticias con detalle.	Control de sistemas o vida cotidiana.	*The Sims*, *Flight Simulator*.
Terror	Generar miedo o tensión en el jugador.	Exploración limitada y supervivencia.	*Resident Evil*, *Outlast*.

Nota: En esta tabla se muestra las características principales de cada categoría de videojuego y como ambas llegan a funcionar bien juntas.

2.3.1. Juego de Acción

son juegos diseñados para ejercitar los músculos del jugador de naturaleza física, la velocidad de reacción o la coordinación mano-ojo. Uno de sus principales objetivos es presentar una experiencia de juego dinámica y desafiante de tal manera que los jugadores interactúen con obstáculos, enemigos y escenarios de toma de riesgos de manera dinámica, utilizando reflejos y precisión, y con control sobre el personaje en tiempo real.

Un ritmo rápido, mecánicas de juego simples y una dificultad progresiva hacen de estos videojuegos una experiencia donde el jugador está continuamente aprendiendo nuevas habilidades. Tienen elementos narrativos, pero el enfoque principal en estos juegos es cómo realizar lo que se requiere instantáneamente y reaccionar rápidamente a los estímulos ambientales.

Los aspectos que distinguen a un videojuego de acción son los siguientes:

- Confrontaciones en tiempo real: el jugador debe reaccionar rápidamente a enemigos, trampas o eventos del entorno.
- Control preciso del personaje: se requiere dominio del movimiento, ataques, esquivas o habilidades especiales.

- Desafíos de reflejos y coordinación: El éxito del juego depende del movimiento y la precisión en la ejecución.
- Progresión basada en habilidad: el avance se logra al superar niveles, jefes o secuencias de combate.
- Estética intensa y dinámica: efectos visuales, música rápida y animaciones fluidas refuerzan la sensación de urgencia.

2.3.2. Juegos de Aventura

Los juegos de aventura son videojuegos basados en personajes que presentan exploración, narrativa y desafíos intelectuales en lugar de acción directa. Su propósito principal es permitir al jugador perderse en una simulación inmersiva donde necesita avanzar mediante la toma de decisiones, recolectar cosas, interactuar con otros personajes y misiones basadas en acertijos o misiones que desarrollan la historia.

Se enfatiza la experiencia narrativa y la inmersión en el mundo virtual a medida que las acciones del jugador alteran su paisaje. En general, el jugador es el protagonista en una misión o propósito que lo hace navegar a través de laberintos y obstáculos que requieren pensamiento lógico, exploración extensa y, a veces, alternativas morales o emocionales.

Los aspectos que distinguen a un videojuego de aventura son los siguientes:

- Enfoque narrativo: la historia es el eje central del juego, guiando las acciones y motivaciones del jugador.
- Exploración del entorno: los escenarios suelen ofrecer múltiples caminos, secretos o áreas ocultas que incentivan la curiosidad.
- Interacción con personajes y objetos: el jugador debe dialogar, recolectar elementos o combinarlos para resolver enigmas.
- Progresión basada en la historia: el avance depende de cumplir objetivos narrativos o resolver situaciones clave.
- Ambiente inmersivo: la música, los gráficos y el guion se combinan para crear una experiencia emocional y coherente.

2.3.3. Juegos de Estrategia

Los juegos de estrategia son un tipo de juego que se centra en la toma de decisiones a largo plazo. Sirven principalmente para probar la capacidad del jugador para analizar situaciones, predecir lo que hará un oponente y poner en acción un plan para ganar. Puede ser un juego

en vivo o una serie por turnos y puede jugarse en tiempo real (RTS) o por turnos (TBS), tratando principalmente con unidades, territorios, tecnología y/o economías. El éxito no se basa en la velocidad con la que los jugadores ejecutan, sino en la inteligencia táctica y la flexibilidad.

Los aspectos que distinguen a un videojuego de estrategia son los siguientes:

- Toma de decisiones complejas: el jugador debe evaluar múltiples variables antes de actuar.
- Gestión de recursos: se requiere administrar unidades, energía, dinero o tiempo de forma eficiente.
- Planificación táctica: el éxito depende de anticipar movimientos enemigos y optimizar rutas o acciones.
- Escenarios dinámicos: el entorno o los oponentes pueden cambiar el curso del juego.
- Progresión basada en dominio estratégico: el avance se logra mediante la conquista, defensa o resolución de objetivos.

2.3.4. Juego de Plataformas

Los juegos de plataformas son otra categoría de videojuegos que enfatizan la habilidad, la precisión y el control sobre el movimiento del jugador, mientras tienen un objetivo final de atravesar entornos compuestos por plataformas, obstáculos y enemigos, típicamente a través de saltos y progresiones verticales y horizontales. En el centro de estos títulos está el control, la precisión y la manipulación de movimientos que los hace muy efectivos en un entorno virtual. Es este juego simple, combinado con una velocidad que requiere reflejos y coordinación, lo que distingue a este tipo de juego.

En su forma más clásica, los juegos de plataformas tienen niveles que se dividen en secciones o fases donde el jugador debe resolver problemas físicos, saltando sobre los huecos, evitando trampas o luchando contra enemigos para avanzar en la pantalla. El género ha evolucionado gradualmente, introduciendo cosas como un formato narrativo, exploración y mecánicas adicionales, como habilidades especiales o mecanismos para la progresión del personaje.

Los aspectos que definen a un videojuego de plataformas son los siguientes:

- Jugabilidad basada en el movimiento: el jugador debe controlar con precisión los saltos, caídas y desplazamientos del personaje.

- Diseño de niveles estructurado: los escenarios están organizados en plataformas, obstáculos y zonas de riesgo que ponen a prueba la destreza del jugador.
- Desafíos de habilidad y tiempo: se requiere sincronización y rapidez para superar los obstáculos.
- Progresión por niveles: el avance se logra al completar etapas o mundos con dificultad creciente.
- Estilo visual claro y distintivo: los gráficos suelen resaltar la profundidad y distancia entre plataformas, facilitando la orientación del jugador.

2.3.5. Juegos de Rol (RPG)

Los RPG, también conocidos como juegos de rol, son un género que enfatiza el desarrollo de personajes, la toma de decisiones y la inmersión dentro de un entorno narrativo complejo. El jugador asume uno o más arcos de personajes, establece sus habilidades y atributos para completar los roles, y decide jugar/interpretar a los personajes. Este tipo de videojuego ofrece a los jugadores libertad de exploración, personalización de personajes y se caracteriza por su narrativa ramificada.

Los aspectos que distinguen a un videojuego de rol son los siguientes:

- Desarrollo de personajes: el jugador mejora atributos, habilidades y equipo a lo largo del juego.
- Narrativa profunda y ramificada: las decisiones afectan la historia, los diálogos y los finales posibles.
- Exploración libre: los mundos suelen ser abiertos o semiabiertos, con múltiples caminos y misiones secundarias.
- Sistema de combate variado: puede ser por turnos, en tiempo real o táctico, con énfasis en estrategia y sinergia.
- Inmersión en el universo del juego: la ambientación, el lore y los personajes secundarios enriquecen la experiencia.

2.3.6. Juegos de Simulación

Los juegos de simulación se utilizan para construir juegos de computadora que simulan interacciones realistas o plausibles, simulaciones, aplicaciones o situaciones similares (pero no idénticas) a la realidad. Su función principal es proporcionar una simulación en vivo donde observas, gestionas y tomas decisiones en un entorno simulado. Puede incluir tanto

simuladores sociales, económicos o de vida como simuladores de vuelo, conducción o agricultura, y muchos más subgéneros. El juego trata sobre precisión, repetición, control y comprensión de los sistemas interrelacionados en nuestras cabezas a través del juego repetitivo, con repetición y comprensión sobre la interconexión de estos sistemas.

Los aspectos que distinguen a un videojuego de simulación son los siguientes:

- Modelado de sistemas reales o ficticios: se representan procesos físicos, económicos o sociales con cierto grado de fidelidad.
- Interacción basada en parámetros: el jugador modifica variables y observa sus efectos en el sistema.
- Progresión no lineal: no siempre hay un objetivo final; el juego puede ser abierto o basado en metas autoimpuestas.
- Énfasis en la observación y el análisis: se premia la comprensión del sistema más que la acción directa.
- Estética funcional: los gráficos y sonidos suelen priorizar la claridad sobre el espectáculo visual.

2.3.7. Juegos de Terror

Los juegos de terror son una forma de juego diseñada para hacer que el jugador se sienta nervioso, tenso e incómodo con el uso de atmósferas desagradables, peligros impredecibles y recursos narrativos oscuros. Los objetivos de este tipo de videojuego son crear un ambiente muy intenso para que el jugador deba enfrentarse a lo desconocido, lo sobrenatural o lo mental. Este tipo de videojuego podría intercalarse con aventura, acción, supervivencia, etc., pero en los juegos tradicionales siempre podemos mantener este enfoque central de debilidad y gestión del miedo.

Los aspectos que distinguen a un videojuego de terror son los siguientes:

- Ambiente opresivo: escenarios oscuros, sonidos inquietantes y diseño visual que genera tensión constante.
- Sensación de vulnerabilidad: el jugador suele tener recursos limitados o estar en desventaja frente a las amenazas.
- Narrativa perturbadora: historias que exploran lo desconocido, lo sobrenatural o lo psicológico.

- Mecánicas de supervivencia: gestión de recursos, sigilo o escape en lugar de combate directo.
- Reacciones emocionales intensas: sustos, ansiedad y tensión sostenida como parte de la experiencia.

2.4. Licencias Open-Source

Las licencias Open-Source permiten a los usuarios utilizar, ejecutar, modificar y redistribuir el software sin restricciones. Aunque esta licencia a menudo se confunde con el software de código abierto, el software libre se basa en el concepto de libertad y está destinado a los usuarios, mientras que el código abierto otorga a los desarrolladores más poder y pone más énfasis en la función práctica del software.

Existen varios tipos de licencias:

- Licencias de Software de Dominio Público
- Licencias de Permiso
- Licencias de Copyleft
- Licencias de Atribución
- Licencias de Patente

2.5. SCRUM vs Programación Extrema

Las metodologías agiles tienen como definición de aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno (Sotomayor, 2024). Para las empresas, su utilización ha permitido reducir los costes en sus operaciones y aumentar la productividad, llegando a generar mayores ganancias.

Tanto Scrum como Programación Extrema son metodologías que tienen un parecido en los siguientes aspectos:

- Se basan en entregas incrementales del software funcional, implementando cambios pequeños y continuos en lugar de grandes actualizaciones al final del proyecto.
- Su objetivo común es poner en marcha el proyecto lo antes posible, garantizando un flujo constante de mejoras y retroalimentación durante todo el proceso de desarrollo.

- Priorizan la interacción constante entre los miembros del equipo, promoviendo un entorno de trabajo colaborativo y orientado a la resolución de problemas.
- Se busca mantener un diseño de software simple y comprensible para facilita la comprensión del sistema, el mantenimiento del código y la detección temprana de errores.

Por otro lado, ambas metodologías gestionan el desarrollo del proyecto y sus aspectos de diferente manera, siendo explicadas en la siguiente tabla:

Tabla 3

Diferencia entre Metodologías de Software

Extreme Programming	**SCRUM**
Se centra en la programación, la codificación y la metodología basada en pruebas.	Se centra principalmente en la gestión.
Requiere sólo una o dos semanas de trabajo en equipo.	Los equipos trabajan en "sprints", que pueden durar desde semanas hasta meses.
Utiliza tarjetas de historias para realizar un seguimiento de las tareas.	Utiliza tableros de tareas para realizar un seguimiento de las tareas.
La programación extrema permite cambios en líneas de tiempo predeterminadas.	Scrum no permite cambios en su cronograma o principios.
El propósito del sprint es la creación de software libre de errores.	Los sprints tienen como objetivo dar como resultado un producto funcional.
Funciona estrictamente en orden de prioridad secuencial.	El cliente prioriza las funciones a crear y se espera que el equipo las ejecute en ese orden.

Nota: La tabla muestra los aspectos característicos de las metodologías SCRUM y Programación Extrema para comprender en que se diferencian. De: (Abdullahi, 2022)

En base a lo descrito y dado que el proyecto es del desarrollo de un juego, donde se requiere una organización flexible, un trabajo iterativo y una constante retroalimentación durante el proceso creativo y técnico, Ambos enfoques son métodos adecuados para este tipo de trabajo, ya que la estructura puede ajustarse y las etapas pueden abordarse mediante la

planificación por etapas. Para esta tarea, estas son cada una de las etapas que consisten en los diseños a cada nivel, los elementos del juego y la funcionalidad del juego, pero todos son iguales en un paso de tiempo manejable mientras se mejora a medida que avanza el proyecto. El enfoque Scrum adoptado en este proyecto sería el siguiente:

1. Definición del Product Backlog: Se recopila y prioriza todas las características y funcionalidades que formaron parte del videojuego, tales como el diseño de niveles, la programación del movimiento del personaje principal, la implementación de enemigos, la gestión de colisiones, la integración de sonidos y efectos visuales, entre otros elementos. Todo esto se organiza en una lista priorizada denominada Product Backlog, que servirá como guía general del desarrollo.
2. Sprints: El proceso de desarrollo se divide en iteraciones cortas con una duración aproximada de 2 a 4 semanas cada uno. En cada sprint se seleccionaron elementos específicos del Product Backlog para ser implementados y probados, con el objetivo de obtener al final de cada ciclo una versión funcional del videojuego que refleje los avances alcanzados.
3. Revisión del Sprint o Sprint Review: Al finalizar cada sprint, se realiza una demostración del progreso obtenido, mostrando las nuevas funcionalidades o mejoras implementadas. Esta revisión permitió evaluar el cumplimiento de los objetivos establecidos, obtener retroalimentación y ajustar las prioridades del Product Backlog si fuera necesario.
4. Retrospectiva del Sprint (Sprint Retrospective): Tras cada revisión, se realizo una reflexión sobre el desempeño como trabajo para el Sprint siguiendo cada revisión para mirar hacia atrás en el trabajo realizado, observando los éxitos, así como las oportunidades de mejora. A través de este análisis, se identifican acciones específicas para mejorar la organización, comunicación y desempeño para los próximos sprints del proceso de revisión, y mejorar el desarrollo para el próximo videojuego.

De esta manera, el desarrollo del juego se convierte en un ejercicio oportuno y eficiente, permitiendo ajustes constantes a los temas y diseño del juego en respuesta a posibles novedades, y la entrega constante de valor a los usuarios finales.

Por otra parte, la implementación de la metodología de Extreme Programming se llega a efectuar de la siguiente manera:

1. Planificación: En esta etapa se identifican las funcionalidades esenciales del videojuego y se dividen en pequeñas historias de usuario que describen lo que el jugador podrá realizar dentro del entorno. Estas historias se priorizan y se estiman en función del tiempo y esfuerzo necesarios para su implementación, estableciendo una planificación inicial flexible que se ajustará conforme avance el desarrollo.
2. Diseño: Se define una arquitectura de software simple y clara que facilite la comprensión del código y su mantenimiento. En el caso del videojuego, se establecerán las clases, escenas y nodos principales en Godot, asegurando que cada componente tenga una responsabilidad bien definida. La simplicidad del diseño permite realizar cambios o mejoras sin afectar otras partes del sistema.
3. Codificación: La programación se realiza siguiendo buenas prácticas, el refactoring constante y la integración continua del código. Cada funcionalidad nueva del videojuego, como el movimiento del jugador o la detección de colisiones, se desarrolla en pequeños incrementos, asegurando que el producto siempre sea funcional.
4. Pruebas (Testing): Cada módulo o característica del videojuego se somete a pruebas unitarias y funcionales para garantizar su correcto funcionamiento antes de integrarlo al resto del sistema. Esto incluye verificar la jugabilidad, las físicas del entorno, el comportamiento de los enemigos y la estabilidad general del juego en cada iteración.
5. Retroalimentación y mejora continua: Se recopilan observaciones del equipo y de los usuarios de prueba tras cada versión jugable. Con base en esta retroalimentación, se realizan ajustes al código, mecánicas o diseño visual, mejorando progresivamente la calidad del videojuego. La comunicación constante y la revisión continua permiten mantener un producto estable y en evolución.

Capítulo 3: Análisis de Herramientas y Metodologías seleccionadas para el Desarrollo del Videojuego

En la siguiente sección se tiene como propósito analizar las herramientas y metodologías seleccionadas para el desarrollo de la demo del videojuego de aventura y plataformas en 2D. A partir de un enfoque técnico y metodológico, se examinan los criterios que sustentan la elección del motor de videojuego Godot Engine y la metodología ágil Scrum, así como los aspectos relacionados con los géneros del videojuego, la elección del formato 2D y el uso de herramientas complementarias que fortalecen el flujo de trabajo. Este análisis busca justificar de manera técnica y pedagógica cada una de las decisiones adoptadas, evaluando sus ventajas, limitaciones y su coherencia con los objetivos planteados en el proyecto.

3.1. Criterios de Evaluación

Para la selección de las herramientas y la metodología de desarrollo, se definieron los siguientes criterios de evaluación:

- Aplicabilidad educativa: que las herramientas y procesos puedan ser comprendidos y replicados fácilmente por estudiantes o desarrolladores independientes.
- Accesibilidad técnica: que presenten una curva de aprendizaje adecuada y una instalación sencilla.
- Coste y licenciamiento: que sean de uso libre o de código abierto, sin costos asociados de licencia o regalías.
- Capacidad de prototipado: que permitan crear versiones jugables en corto tiempo.
- Soporte para 2D: que integren sistemas nativos de físicas, animación y renderizado en dos dimensiones.
- Integración con metodologías ágiles: que su flujo de trabajo permita la iteración continua y el control de versiones.
- Portabilidad y exportación: que faciliten la generación de ejecutables funcionales, especialmente en plataformas como Windows.
- Comunidad y documentación: que cuenten con soporte activo, tutoriales, foros y ejemplos prácticos.

Estos criterios orientaron el análisis comparativo y la selección final de las tecnologías aplicadas al desarrollo del videojuego.

3.2. Análisis del Motor de Videojuego: Godot Engine

3.2.1. *Características generales*

Godot Engine es un motor de videojuegos gratuito, multiplataforma y de código abierto, distribuido bajo la licencia MIT, lo que permite su uso sin restricciones ni pago de regalías. Fue diseñado con el objetivo de ofrecer un entorno unificado para el desarrollo de juegos 2D y 3D, priorizando la simplicidad, el modularidad y la eficiencia.

Su estructura basada en escenas y nodos facilita la organización jerárquica de los elementos del juego, permitiendo una programación más visual e intuitiva. Además, su principal lenguaje de scripting, GDScript, está inspirado en Python, lo que facilita el aprendizaje para los nuevos desarrolladores.

3.2.2. Fortalezas técnicas relevantes

- Diseño por escenas y nodos: favorece la reutilización y modularidad del contenido.
- Motor 2D nativo: optimizado para físicas, animaciones, colisiones y renderizado bidimensional.
- Lenguaje GDScript: sintaxis sencilla y orientada al prototipado rápido.
- Exportación multiplataforma: posibilidad de exportar a Windows, Linux, macOS, Android, iOS y HTML5 con un solo clic.
- Licencia MIT: completamente abierta y sin regalías.
- Amplia comunidad y documentación: soporte continuo por parte de usuarios y desarrolladores alrededor del mundo.
- Sistema de señales (signals): facilita la comunicación entre nodos de forma desacoplada.
- Herramientas visuales integradas: depurador, monitoreo de frames y perfiles de rendimiento.

3.2.3. Limitaciones y consideraciones

A pesar de sus ventajas, Godot presenta ciertas limitaciones que deben considerarse:

- Las diferencias entre las versiones principales (por ejemplo, 3.x y 4.x) pueden causar incompatibilidades en los scripts o escenas.
- Para proyectos de gran escala puede ser necesaria la integración con C# o módulos nativos.

- Su ecosistema de recursos comerciales es más limitado que el de motores de pago como Unity o Unreal Engine, aunque la comunidad compensa esta carencia mediante la creación de activos gratuitos.

3.2.4. Comparación general de motores

Tabla 4

Cuadro Comparativo entre motores Godot Engine, Unity e Unreal Engine

Criterio	**Godot**	**Unity**	**Unreal Engine**
Licencia	MIT, sin regalías.	Gratuita hasta cierto umbral de ingresos; posibles licencias comerciales.	Gratuita con regalías del 5% sobre ingresos comerciales.
Curva 2D	Muy amigable; motor 2D nativo.	Buena, aunque basada en entorno 3D.	Limitada; optimizado principalmente para 3D.
Lenguaje por defecto	GDScript (sencillo y similar a Python).	C# (lenguaje moderno y ampliamente usado).	C++ y Blueprints (más complejos para principiantes).
Comunidad y recursos libres	Activa y creciente; abundante contenido libre y colaborativo.	Muy amplia; numerosos tutoriales y recursos, muchos de pago.	Grande pero enfocada en entornos profesionales y AAA.
Exportación a Windows	Directa y sin configuración adicional.	Directa; requiere plantillas de exportación.	Directa; alto rendimiento en entornos PC.

Nota: La tabla muestra los aspectos característicos de cada motor de juego (Parente, 2024)

En base a lo descrito, la herramienta de Godot ofrece la combinación ideal entre accesibilidad, libertad de uso y herramientas 2D nativas. Su arquitectura modular y su orientación educativa lo convierten en una opción adecuada para el desarrollo de una demo de videojuego en 2D, permitiendo documentar de forma clara cada fase del proceso.

3.3. Metodología de Desarrollo: SCRUM

En base a lo descrito en la anterior sección sobre las metodologías Scrum y Extreme Programming, la metodología Scrum se seleccionó debido a su adaptabilidad, su enfoque iterativo e incremental, y su orientación a la entrega constante de valor. Con esta metodología se pudo planificar y revisar avances en ciclos cortos, identificados como sprints, garantizando que las funciones clave sean priorizadas y ajustadas con base en la retroalimentación constante.

La siguiente tabla explica cuáles fueron las funciones establecidas en el proyecto en base a los Roles de la metodología:

Tabla 5

Responsabilidades de los Roles de Scrum

Rol	**Responsabilidades**
Product Owner	Define la visión del juego, prioriza características y valida entregas.
Scrum Master	Facilita la comunicación, remueve impedimentos y coordina el equipo.
Desarrollador	Implementa mecánicas, diseña niveles, crea assets y realiza pruebas.

3.3.1. Definición de los Artefactos y Eventos para el Proyecto

La metodología Scrum se estructura mediante artefactos y eventos que permiten mantener un flujo de trabajo ordenado, controlado y adaptable a las necesidades del proyecto. En el caso del desarrollo de la demo del videojuego 2D de aventura y plataformas, estos elementos se implementan de manera simplificada, priorizando la funcionalidad en cada nivel y la retroalimentación constante de mejoras y cambios.

Los artefactos representan los componentes tangibles del proceso y permiten gestionar las tareas y resultados obtenidos en cada ciclo de trabajo:

- Product Backlog: Lista priorizada que reúne todas las características y funcionalidades que se planean implementar en el videojuego, tales como el movimiento del personaje, la física de colisiones, el sistema de recolección de objetos, la interfaz del usuario y los distintos niveles.

- Sprint Backlog: Conjunto de tareas seleccionadas desde el Product Backlog que se abordarán durante un sprint específico. Cada sprint tiene objetivos concretos y alcanzables, lo que permite avanzar de forma incremental hacia una versión completa de la demo del videojuego.
- Incremento: Resultado funcional obtenido al finalizar un sprint. En este proyecto, cada incremento corresponde a una versión jugable del videojuego, que incluye las mecánicas implementadas durante el ciclo. Este incremento permite realizar pruebas tempranas, detectar errores y ajustar los elementos del juego antes de continuar con el siguiente sprint.

Las ceremonias de Scrum son los eventos o actividades que estructuran la comunicación, la evaluación y la mejora continua dentro del proceso de desarrollo:

- Sprint Planning: Reunión de planificación en la que se definen los objetivos específicos del sprint, las tareas a realizar y los criterios de éxito, esta fase consiste en determinar qué funcionalidades del videojuego se desarrollarán durante cada iteración.
- Daily Stand-up: Revisión diaria o autoevaluación del progreso. Este evento se adapta a una reflexión breve sobre los avances, los obstáculos encontrados y las acciones a realizar para mantener el ritmo de trabajo.
- Sprint Review: Presentación del incremento al finalizar cada sprint. En esta etapa se prueba la versión jugable, se analiza su funcionamiento y se registran observaciones para posteriores mejoras. Este paso es esencial para verificar que las mecánicas, físicas y elementos visuales cumplan con los objetivos planteados.
- Sprint Retrospective: Evaluación del proceso al final de cada sprint. Permite identificar las fortalezas y debilidades del ciclo anterior y planificar estrategias para optimizar los siguientes. Este enfoque garantiza una mejora continua tanto del producto como del método de trabajo.

De esta manera, la aplicación de la metodología Scrum en el desarrollo de la demo del videojuego permitió mantener un proceso estructurado, flexible y orientado a resultados medibles. Gracias a su naturaleza iterativa, cada sprint produjo avances tangibles que facilitaron la validación temprana de las mecánicas implementadas, fortaleciendo tanto la eficiencia del trabajo como la calidad del producto final.

3.4. Análisis de los géneros del videojuego

El diseño de un videojuego se encuentra en gran medida influenciado por el género al que pertenece, ya que el mismo define las mecánicas de juego, la estructura narrativa, los objetivos del jugador y la experiencia interactiva que se busca ofrecer. En este proyecto, se ha optado por combinar dos géneros clásicos y complementarios: aventura y plataformas, cuya integración proporciona un equilibrio entre exploración, desafío y progresión.

La combinación de ambos géneros ofrece una estructura equilibrada que integra mecánicas de exploración, recolección y desafío físico, lo que enriquece la experiencia del jugador sin aumentar la complejidad técnica del desarrollo. Esta fusión permite que cada nivel del juego cumpla dos funciones:

En el aspecto narrativo, el componente de aventura proporciona una motivación argumental y un propósito recreativo: el jugador no solo debe superar niveles, sino que avanzar dentro de una progresión temática, vinculada a la recolección de objetos, el desbloqueo de zonas o la resolución de desafíos que representan etapas dentro del mundo del juego. Esto genera un sentido de continuidad y coherencia entre los niveles, reforzando la inmersión y la conexión con el entorno.

Por otra parte, el componente de plataformas introduce mecánicas de precisión y control, propias de los desafíos de desplazamiento horizontal, saltos calculados y sincronización con el entorno. Estas dinámicas convierten la narrativa en un elemento interactivo, ya que los objetivos narrativos o de progresión se alcanzan mediante la ejecución de acciones físicas en el entorno virtual.

A nivel técnico, la integración de ambos géneros permitió aprovechar al máximo las capacidades nativas del motor de Godot Engine, especialmente con sus sistemas de física, animación y colisiones, además del manejo modular de escenas y niveles mediante el módulo de TileMaps. El motor ofrece las herramientas necesarias para combinar elementos interactivos propios de la aventura (como triggers, señales o condiciones de avance) con

mecánicas físicas propias de plataformas siendo el caso de saltos, detección de colisiones y movimiento horizontal fluido.

3.5. Justificación del uso de 2D en el desarrollo de la demo

La elección de un entorno de desarrollo en 2D para la creación de la demo responde principalmente a factores técnicos, de diseño y de optimización de recursos. En el contexto de proyectos independientes o académicos, el desarrollo 2D permite concentrar los esfuerzos en la lógica del juego y en las mecánicas principales sin depender de una infraestructura visual o técnica compleja. Esta decisión favorece la eficiencia del proceso y la claridad en la documentación del desarrollo.

Desde un punto de vista técnico, “la jugabilidad en 2D reduce significativamente los costos de desarrollo. Los activos 2D simples son más rápidos de producir, lo que reduce el tiempo de desarrollo y permite a los equipos lograr más con menos personal” (Business Money, 2024). Estas características reducen considerablemente la carga de procesamiento y simplifican el flujo de trabajo del desarrollador, lo que permite obtener resultados funcionales en menor tiempo y con menor uso de recursos computacionales. Además, el rendimiento en entornos 2D es superior en equipos de gama media o baja, garantizando una mayor compatibilidad con diferentes dispositivos.

En cuanto al diseño y ámbito artístico, optar por un estilo 2D ofrece una mayor versatilidad estética. Permite trabajar con diferentes estilos gráficos, como el pixel art, los vectores o las ilustraciones digitales, utilizando herramientas accesibles y de código abierto, reduciendo la dependencia de modelado tridimensional o renderizado avanzado, sin sacrificar calidad ni expresividad. Asimismo, el arte 2D puede mantener un estilo atemporal que se adapta bien a otra clase de proyectos.

En la siguiente tabla, se muestra una comparativa más explícita en cuanto al desarrollo de videojuegos en 2D y 3D:

Tabla 6

Comparación Técnica entre Desarrollo 2D y 3D

Criterio	Desarrollo 2D	Desarrollo 3D
Complejidad técnica	Las físicas, animaciones y colisiones son más simples de implementar.	Requiere control de cámaras, modelado, iluminación y cálculos espaciales.
Rendimiento	Alto rendimiento incluso en equipos con bajos recursos; tiempos de carga reducidos.	Mayor demanda de procesamiento y memoria; depende del hardware y la optimización.
Producción de recursos	Requiere gráficos planos (sprites, tilesets), fáciles de crear y modificar.	Necesita modelos 3D, texturizado y rigging, aumentando el tiempo de producción.
Diseño de niveles	Basado en cuadrículas o capas.	Requiere un diseño espacial complejo con dimensiones y profundidad.
Depuración y pruebas	Los errores visuales o de físicas son más fáciles de identificar.	El manejo de cámaras y transformaciones puede generar errores difíciles de detectar.
Tiempo de desarrollo	Permite ciclos rápidos de prueba y mejora.	Implica más etapas de modelado, iluminación y renderizado.
Estilo visual	Enfocado en arte 2D y pixel art.	Estilo realista o cinematográfico.
Compatibilidad con dispositivos	Ejecutable en una amplia gama de equipos.	Limitada a dispositivos con GPU y CPU potentes para un rendimiento fluido.

Nota: La tabla muestra la diferencia de los estilos 2D y 3D en el desarrollo de videojuegos

Capítulo 4: Desarrollo del Videojuego

4.1. Listado General de Requerimientos del Videojuego “The Scar of the Tomb”

El listado de requerimientos constituyó como un esencial dentro del proceso de desarrollo, ya que permitió definir con claridad las características, funcionalidades y limitaciones planteadas del producto a construir. En este proyecto, los requerimientos se establecieron con el propósito de orientar el desarrollo de la demo del videojuego 2D de aventura y plataformas bajo el nombre “The Scar of the Tomb”, diseñada en el motor Godot Engine, aplicando la metodología Scrum.

De esta manera se garantizó que el producto cumpliera con los estándares técnicos y de jugabilidad esperados, además de ofrecer una experiencia fluida y coherente con el alcance planteado en las etapas de planificación.

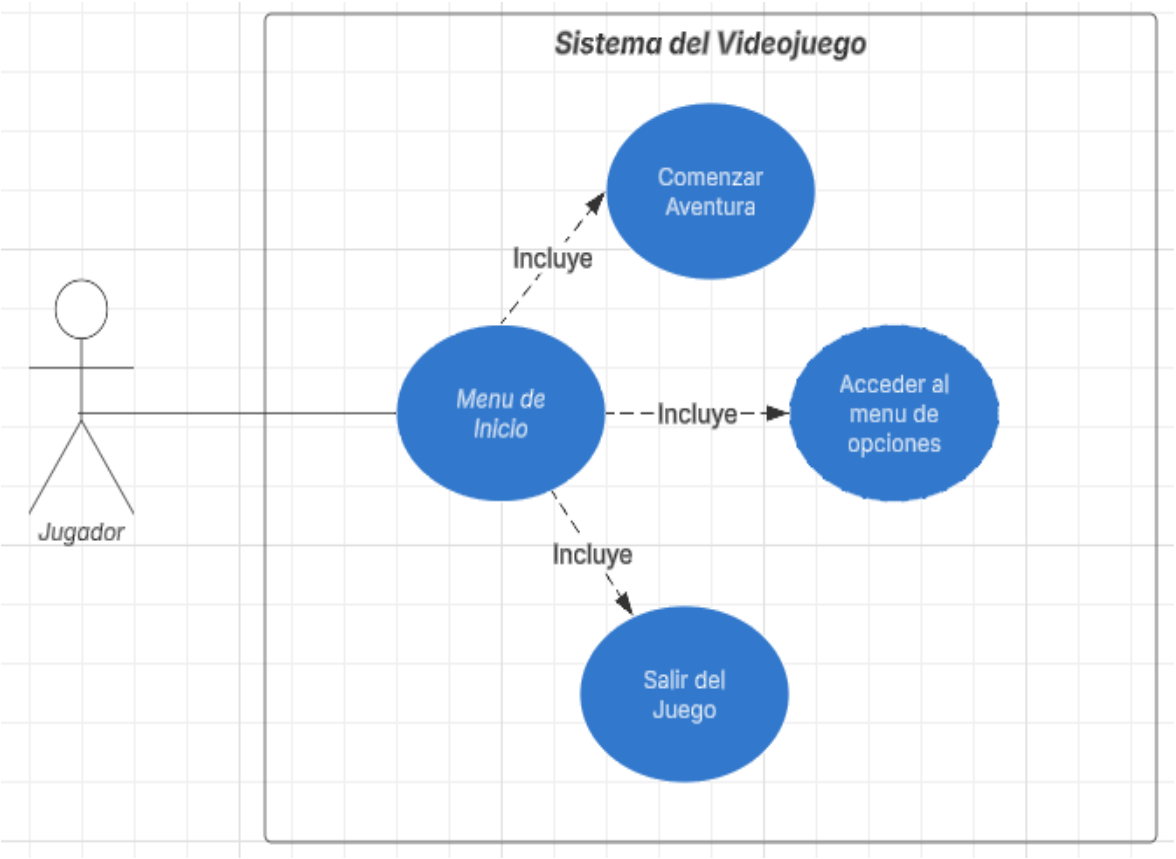
4.1.1. Diseño Conceptual de la Demo

En esta etapa se definieron las interacciones fundamentales entre el jugador y el sistema, se establecieron los elementos que dieron forma al funcionamiento del juego y su flujo general, con el fin de representar de manera visual las funciones principales antes de la implementación técnica.

Para ello, se elaboró varios diagramas de casos de uso, en el que se identificaron los actores involucrados y las acciones que cada uno podía realizar dentro del entorno del videojuego. Estos diagramas representaron la relación entre el jugador y las funciones esenciales, como el movimiento, la recolección de objetos, la interacción con los niveles y el manejo de menús.

Figura 1

Diagrama de Caso de Uso de Iniciar Juego



Caso de Uso 1: Iniciar Juego

Actor Principal: Jugador

Descripción: Permite que el jugador acceda al menú principal e iniciar la partida, cargando el primer nivel del juego.

Precondiciones: El juego debe estar instalado y ejecutándose correctamente.

Postcondiciones: Se carga el nivel inicial del juego y comienza la sesión de juego.

Secuencia Normal:

1. El jugador abre el ejecutable.
2. El sistema muestra el menú inicial.
3. El jugador selecciona “Iniciar Aventura”.
4. El sistema ejecuta la carga del primer nivel.

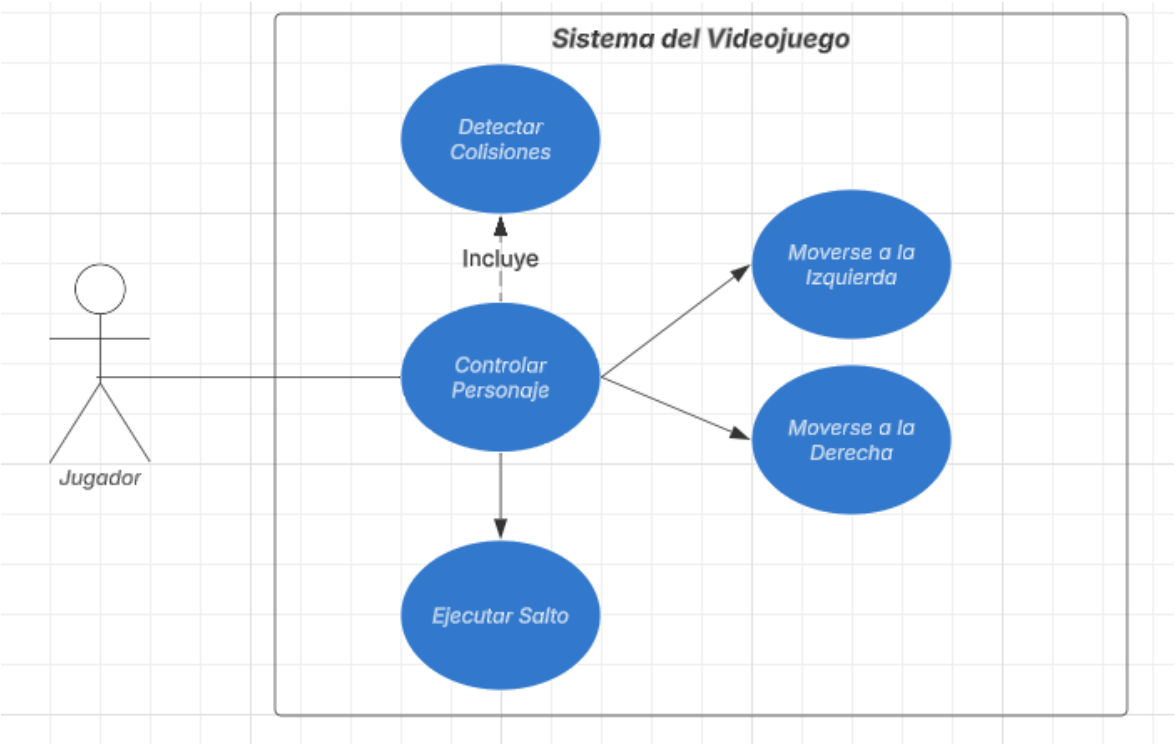
Flujo Alterno:

A1: El jugador selecciona “Salir”: el sistema cierra la aplicación.

A2: El jugador ingresa a “Opciones”: se activa Caso de Uso 3.

Figura 2

Diagrama de Caso de Uso de Controlar Personaje



Caso de Uso 2: Controlar Personaje

Actor Principal: Jugador

Descripción: Permite que el jugador controlar los movimientos básicos del personaje, incluyendo desplazamiento y salto.

Precondiciones: El nivel debe estar cargado.

Postcondiciones: El personaje responde correctamente a los comandos del jugador.

Secuencia Normal:

1. El jugador pulsa teclas de movimiento.
2. El sistema desplaza al personaje.
3. El jugador presiona el botón de salto.

4. El sistema ejecuta la mecánica de salto.
5. El sistema detecta colisiones durante la acción.

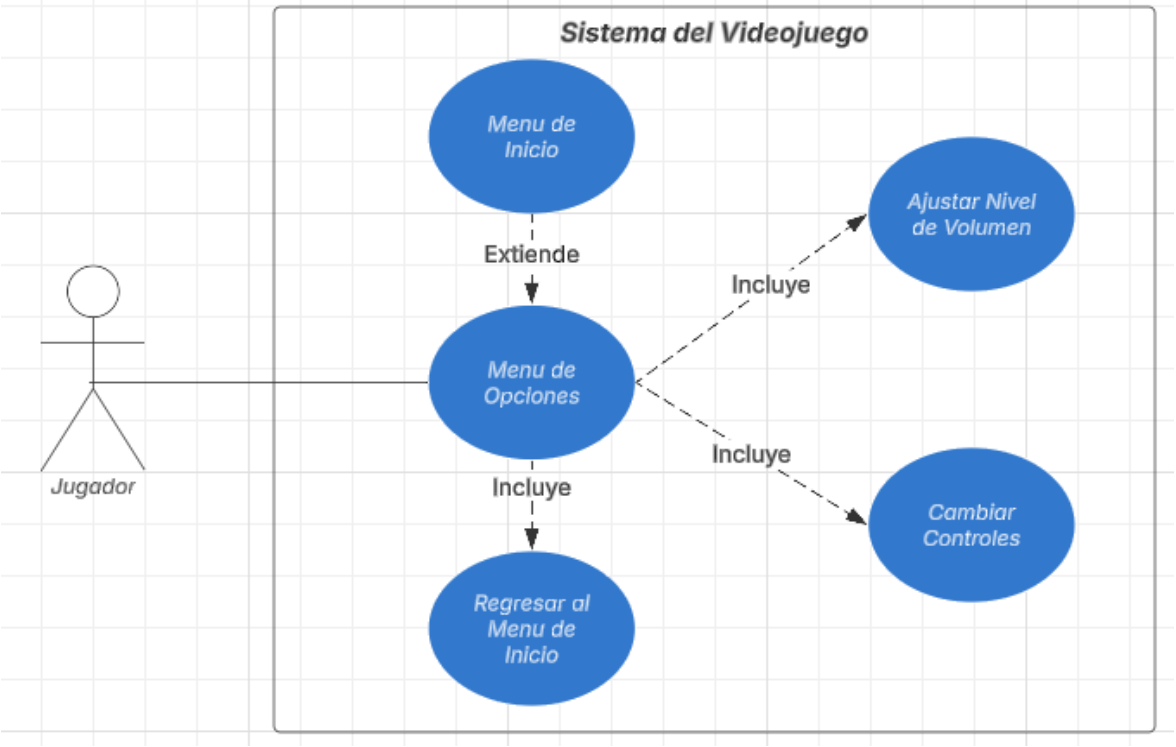
Flujo Alterno:

A1: Si el jugador intenta saltar sin poder hacerlo, el sistema no ejecuta la acción.

A2: Si colisiona con una pared o enemigo, se procesa la reacción correspondiente.

Figura 3

Diagrama de Caso de Uso de Administrar Menú de Opciones



Caso de Uso 3: Administrar Menú de Opciones

Actor Principal: Jugador

Descripción: Permite que el jugador acceda y modifique parámetros dentro del menú principal.

Precondiciones: El menú debe estar abierto.

Postcondiciones: Los cambios se guardan y se aplican durante la sesión.

Secuencia Normal:

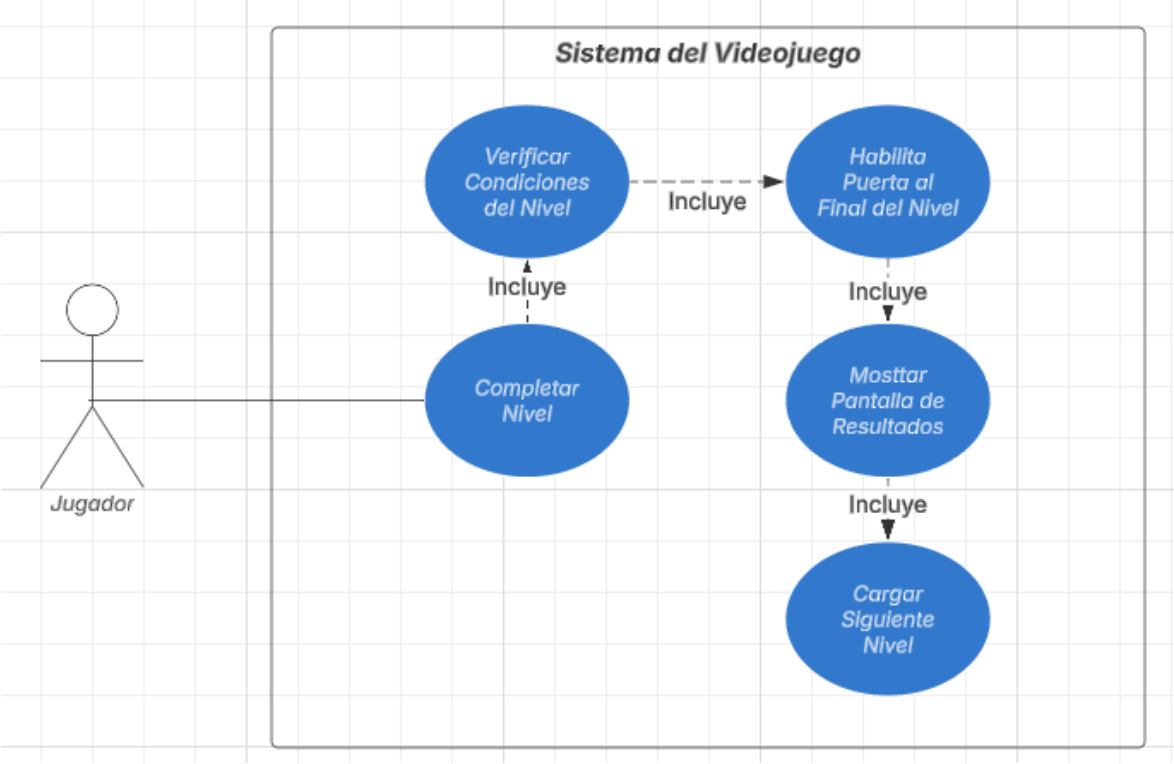
1. El jugador accede al menú principal.
2. Selecciona la opción "Opciones".
3. Ajusta parámetros disponibles.
4. Confirma cambios y regresa al menú anterior.

Flujo Alterno:

A1: El jugador cierra el juego.

Figura 4

Diagrama de Caso de Uso de Completar Nivel



Caso de Uso 4: Completar Nivel

Actor Principal: Jugador

Descripción: Permite que el jugador avance al siguiente nivel luego de cumplir con las condiciones establecidas.

Precondiciones: El jugador debe haber recolectado todos los objetos necesarios.

Postcondiciones: Se muestra la pantalla de resultados y luego se carga el siguiente nivel.

Secuencia Normal:

1. El jugador recolecta todos los objetos.
2. El sistema verifica que se cumpla el objetivo.
3. El sistema habilita la puerta al final del nivel.
4. El jugador cruza la puerta.
5. El sistema muestra la pantalla de resultados.
6. El sistema carga el siguiente nivel.

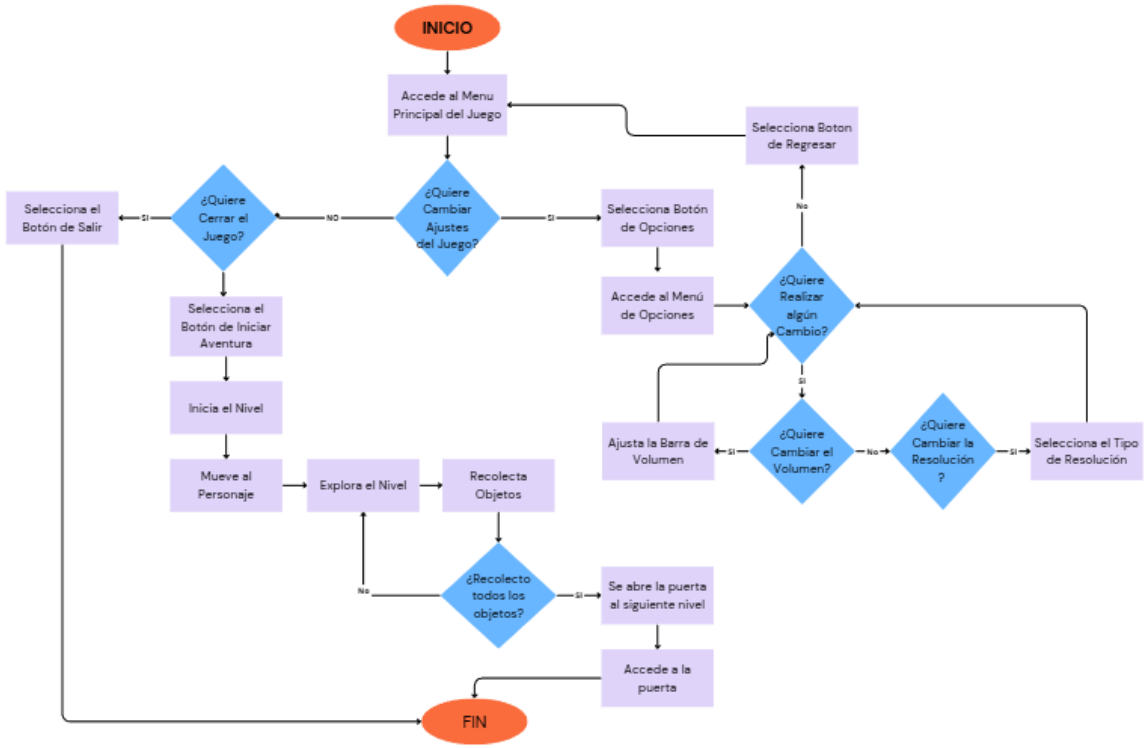
Flujo Alterno:

A1: El jugador no recolecta todos los objetos: el nivel no se completa.

A2: El tiempo llega a cero: se muestra pantalla de derrota y opción de reiniciar nivel.

Figura 5

Diagrama de Flujo del Videojuego “The Scars of the Tomb”



4.1.2. Requerimientos Generales del Sistema del Videojuego

1. Requerimiento de Jugabilidad
   - El jugador debe poder controlar al personaje mediante desplazamiento horizontal.
   - El personaje debe poder ejecutar saltos para superar obstáculos dentro del nivel.

- El sistema debe detectar colisiones entre el personaje, el entorno y los objetos interactivos.
- El jugador debe poder recolectar objetos distribuidos en cada nivel.
- El sistema debe permitir completar un nivel cuando se cumplan las condiciones establecidas.
- El juego debe permitir reiniciar el nivel desde el menú de pausa.
- El sistema debe mostrar una pantalla de resultados al completar un nivel.

2. Requerimiento de Interfaz y Navegación
    - El videojuego debe presentar una pantalla inicial con las opciones “Jugar”, “Opciones” y “Salir”.
    - El sistema debe incluir un menú de opciones accesible desde la pantalla principal o desde el menú de pausa.
    - El jugador debe poder ajustar parámetros básicos del juego como volumen general.
    - El sistema debe presentar un menú de pausa con opciones de “Continuar”, “Reiniciar” y “Salir al Menú Principal”.
    - La interfaz debe mostrar información relevante: tiempo restante, objetos recolectados y progreso del nivel.
3. Requerimientos de Progreso y Niveles
    - El sistema debe permitir avanzar al siguiente nivel al cumplir los objetivos del nivel actual.
    - Los niveles deben incluir plataformas, obstáculos y objetos recolectables para cumplir con la mecánica del juego.
    - El progreso debe mantenerse durante la sesión activa del jugador.
4. Requerimientos Tecnicos
    - El videojuego debe ejecutarse en sistemas Windows 10 o superior.
    - El rendimiento debe mantenerse a 60 FPS en equipos de gama media.
    - El tiempo de carga de un nivel no debe superar los 3 segundos.
    - Todos los recursos gráficos utilizados deben poseer licencias abiertas o de libre uso.
    - La estructura del proyecto debe basarse en escenas y nodos para facilitar su mantenimiento.

5. Requerimientos de Diseño 2D
    - El videojuego debe desarrollarse completamente en entorno 2D.
    - El estilo visual debe mantenerse consistente utilizando sprites y tilesets definidos para el proyecto.
    - La cámara debe seguir al jugador sin vibraciones ni errores de interpolación.

4.1.3. Requerimientos Funcionales

Los requerimientos funcionales describen las acciones que el sistema debía realizar y las interacciones entre el personaje controlado por el jugador y el entorno del videojuego. En la demo, estos requerimientos se enfocaron en las mecánicas básicas, la interacción con objetos y la navegación a través de los niveles. La siguiente lista corresponde a los requerimientos identificados:

1. El jugador puede desplazarse horizontalmente hacia la izquierda y la derecha.
2. El jugador puede realizar saltos de intensidad definida al presionar una tecla.
3. El sistema detecta colisiones entre el jugador, el terreno y los objetos interactivos.
4. Al recolectar todos los objetos de un nivel, el sistema permitió el avance al siguiente nivel.
5. La pantalla de inicio cuenta con los botones de “Jugar”, “Opciones” y “Salir”.
6. El juego incluyó un menú de pausa con las opciones “Continuar”, “Reiniciar” y “Salir al menú principal”.
7. Cada nivel muestra un temporizador de 300 segundos que limita la duración de la partida.
8. Al finalizar un nivel, se presentó una pantalla con los resultados obtenidos.
9. El sistema guarda temporalmente la puntuación o progreso de la partida durante la sesión activa.

4.1.4. Requerimientos No Funcionales

Los requerimientos no funcionales establecieron las condiciones de calidad, rendimiento, compatibilidad y mantenibilidad que el videojuego debía cumplir, asegurando una experiencia estable y eficiente:

1. El videojuego se ejecuta en sistemas operativos Windows 10 o superior.
2. El rendimiento se mantiene estable a 60 FPS en equipos de gama media.
3. El tiempo de carga por nivel no supera los 3 segundos.
4. El diseño de la interfaz fue intuitivo y accesible.

5. El código se documentó mediante comentarios y nomenclatura estandarizada.
6. El proyecto siguió una estructura modular basada en escenas y nodos de Godot.
7. Los recursos gráficos (sprites, tilesets, fondos) son de libre uso o bajo licencias abiertas.
8. El sistema permite la ampliación futura de niveles y funcionalidades.
9. La exportación del proyecto se realizó con las plantillas oficiales de Godot, garantizando compatibilidad.

4.1.5. Funcionalidades Principales del Videojuego

A partir de los requerimientos definidos, en la siguiente tabla se muestran las principales funcionalidades que se establecieron y conformaron la base del videojuego. Estas funcionalidades representaron las mecánicas esenciales y los componentes que se implementaron durante los sprints de desarrollo.

Tabla 7

Funcionalidades Identificadas del Videojuego “The Scars of the Tomb”

Funcionalidad	**Descripción**	**Objetivo**
Movimiento del personaje	Desplazarse lateralmente con animaciones asociadas.	Facilitar el control fluido del jugador.
Sistema de salto	Mecánica de salto y detección de colisiones con plataformas.	Superar obstáculos y alcanzar zonas elevadas.
Colisiones y físicas	Detectar interacción entre jugador, entorno y objetos.	Asegurar realismo y coherencia en las mecánicas.
Recolección de objetos	Permitir sumar puntos o coleccionables.	Incentivar la exploración y el progreso.
HUD de interfaz	Mostrar información: tiempo, puntuación y número de ítems recolectados.	Mantener informado al jugador durante la partida.

Funcionalidad	Descripción	Objetivo
Menú principal y pausa	Pantallas de navegación e interacción.	Brindar control y facilidad de uso.
Sistema de niveles	Panel con niveles disponibles.	Aumentar la dificultad de forma gradual.
Pantalla de resultados	Presentar el desempeño del jugador al finalizar cada nivel.	Ofrecer retroalimentación inmediata.
Exportación de la demo	Compilación y generación de ejecutable para Windows.	Permitir su distribución y prueba final.

4.2. Aplicación de Metodología SCRUM

Durante la ejecución del proyecto se aplicó la metodología Scrum para la organización del desarrollo por medio de ciclos iterativos, priorizar funcionalidades esenciales y obtener incrementos jugables al final de cada sprint, lo cual facilitó la validación temprana de mecánicas y la documentación progresiva en el manual técnico.

4.2.1. Roles y Responsabilidades

La asignación de los siguientes roles fue realizada y adaptada en base a los principios de Scrum para que funcione dentro de un equipo unipersonal:

1. Product Owner y Development Team: Se asumen ambas funciones, encargándose de validar los avances, establecer prioridades y asegurar que los resultados cumplieran con los objetivos planteados.
2. Scrum Master: Por medio de una autogestión, se planificaron los sprints, se gestionó el tiempo para completar cada sprint, se coordinaron las tareas y se resolvió los impedimentos que surgieron durante el desarrollo.

Por medio de esta adaptación se conservaron los principios de organización, transparencia y mejora continua que caracterizan a Scrum y permitieron trabajar de manera fluida.

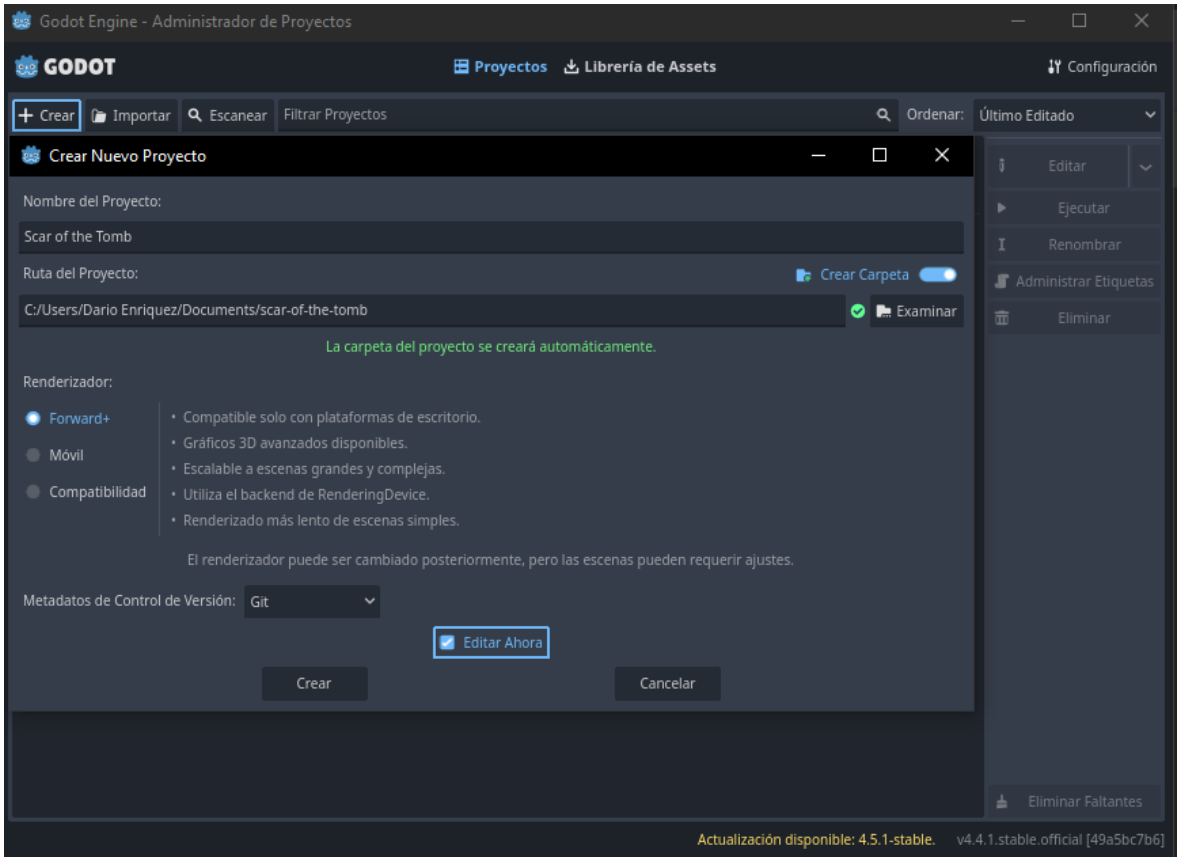
4.2.2. Configuración Previa del Entorno de Desarrollo

Antes del inicio de los sprints, se efectuó una fase de preparación en la que se configuró el entorno de trabajo con el motor Godot Engine y se organizó la estructura del proyecto, estableciendo carpetas para recursos, scripts, escenas y niveles.

La versión con la que se trabajó durante el desarrollo del videojuego fue la versión estable 4.41 lanzada el 26 de marzo de 2025.

Figura 6

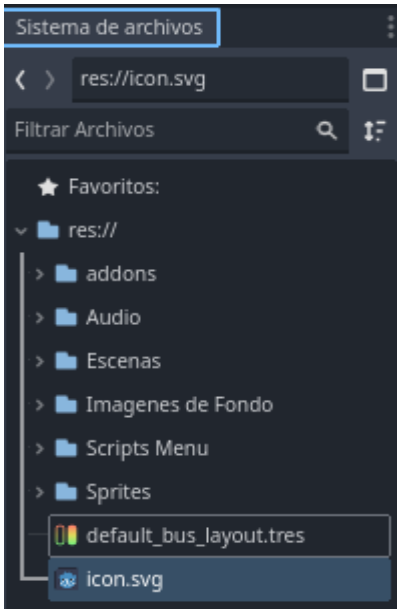
Creación del Nuevo Proyecto en el Administrador de Proyectos en Godot



A continuación, se muestra la organización de las carpetas manejada para el almacenamiento de los Scripts, Audios, Niveles, Assets, etc., utilizados, de manera que se mantuviera un control y orden durante el desarrollo

Figura 7

Organización de las Carpetas del Proyecto



4.2.3. Product Backlog

La elaboración del Product Backlog permitió tener de manera más clara una lista priorizada de las características principales que debía incluir la demo del videojuego.

Cada ítem que se muestra en la siguiente tabla se evaluó en función de su relevancia para la jugabilidad, el impacto visual y su relación con los objetivos generales del proyecto.

Tabla 8

Product Backlog del Videojuego “The Scars of the Tomb”

Ítem del Product Backlog	Descripción	Prioridad
Movimiento y salto del jugador	Desarrollo del sistema básico de control para movimiento lateral y salto.	Alta
Sistema de colisiones y detección de plataformas	Implementación de colisiones precisas para interacción con el entorno, enemigos u obstáculos.	Alta
Diseño y progresión de niveles	Creación de tres niveles con dificultad progresiva y desafíos combinados entre plataformas y recolección.	Alta

Ítem del Product Backlog	Descripción	Prioridad
Sistema de recolección de objetos	Mecánica principal del juego: recolección de ítems para completar niveles y avanzar en la demo.	Alta
Interfaz HUD	Visualización en pantalla de información relevante: objetos recolectados y tiempo restante.	Media
Pantalla de inicio y menú de pausa	Inclusión de un menú funcional con opciones básicas de inicio, pausa y salida.	Media
Pantalla de nivel completado o fin de juego	Escena de retroalimentación al jugador tras cumplir los objetivos del nivel o finalizar la demo.	Media
Efectos de sonido y música de fondo	Implementación de audio básico para ambientar la experiencia del jugador.	Baja
Exportación final del juego a Windows	Generación del ejecutable funcional de la demo en formato .exe.	Alta
Documentación técnica y manual de desarrollo	Elaboración del manual explicativo que describe el proceso de creación del videojuego en Godot.	Alta

4.2.4. Planificación de los Sprints

El trabajo fue dividido en cinco sprints, con una duración aproximada de dos semanas cada uno.

En cada sprint se desarrollaron incrementos funcionales del videojuego, que fueron revisados y ajustados al finalizar cada iteración.

- Sprint 1: Menú principal, sistema de pausa y ajustes visuales.
- Sprint 2: Diseño de primer nivel con movimientos del jugador, con sistema de recolección de objetos y contador en HUD.
- Sprint 3: Diseño del Segundo nivel con detección de colisiones, animaciones básicas y cámara que sigue al jugador.

- Sprint 4: Diseño del tercer nivel con pantalla de finalización y temporizador activo.
- Sprint 5: Documentación final del manual y pruebas de exportación.

4.2.5. Daily Scrum

En estas sesiones breves de autoevaluación se revisaron el progreso y los obstáculos encontrados durante cada sprint mediante la realización de las siguientes preguntas:

¿Qué avances logré el día de ayer?

¿Qué haré hoy para completar el Sprint?

¿Hay alguna dificultad que esté frenando el avance?

De esta manera se mantuvo una retroalimentación constante y control sobre el avance del proyecto.

4.3. Diseño Conceptual del Videojuego

El diseño conceptual del videojuego establece la visión creativa planteada para el proyecto, orientándose en los siguientes aspectos:

4.3.1. Visión General

El presente trabajo propone el desarrollo de una demo de un videojuego de aventura de plataforma 2D con un enfoque a la utilización de un motor de videojuego de código abierto. Su caracterización frente a otros títulos similares se halla en la representación histórica de leyendas y mitos de diversas culturas del mundo por medio del diseño de los objetos y enemigos.

4.3.2. Género y Publico Objetivo

El videojuego pertenece al género de plataformas 2D y de aventura y se encuentra dirigido a jugadores atraídos por mecánicas dinámicas que los desafíen.

4.3.3. Ambientación y Estilo Visual

El juego se encuentra ambientado en la exploración de unas ruinas de un mundo de fantasía. Los escenarios muestran una predominancia por los colores claros, siendo el caso del color marfil de las edificaciones que se encuentran como escenario de fondo. En el caso del arte, se utilizó un estilo pixel art, proporcionando un estilo retro.

Dentro de los distintos niveles, el jugador encontrara objetos denominados reliquias que simbolizan mitos y leyendas de distintas culturas del mundo.

4.3.4. Mecánica y Jugabilidad

El objetivo principal del juego es recolectar reliquias repartidos por el mapa para desbloquear la salida, evitando a los enemigos y llegar a la puerta desbloqueada antes de que se acabe el tiempo.

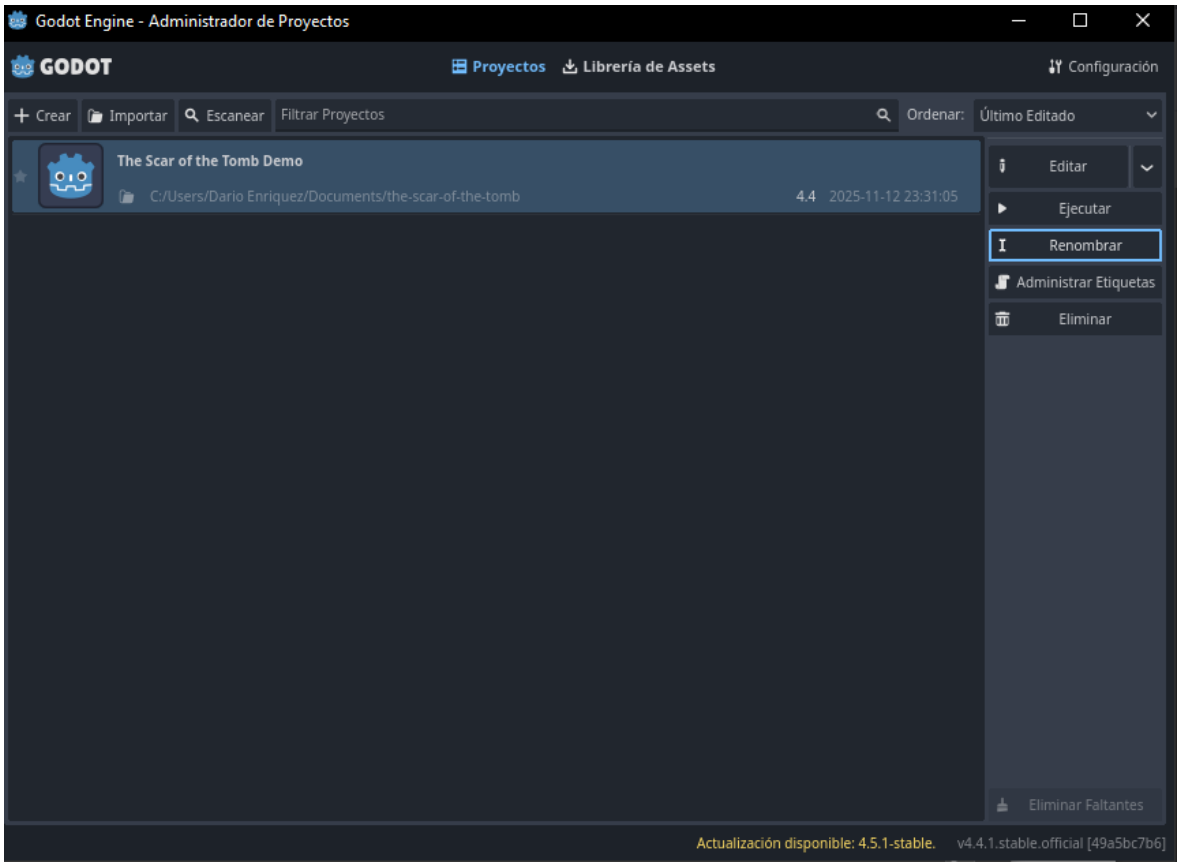
- El jugador comienza al inicio del mapa y debe recolectar un número determinado de reliquias para desbloquear la salida y llegar a la misma antes de que se acabe el tiempo.
- Cada nivel tendrá enemigos representativos de una cultura y si tocan al jugador, saldrá el mensaje de “Game Over” y se le dará la opción de reiniciar el nivel o salir al menú principal.

4.4. Implementación del Prototipo en Godot Engine

Esta sección muestra la implementación de la demo del videojuego utilizando el motor establecido, Godot Engine. Se definen los componentes básicos del proyecto.

Figura 8

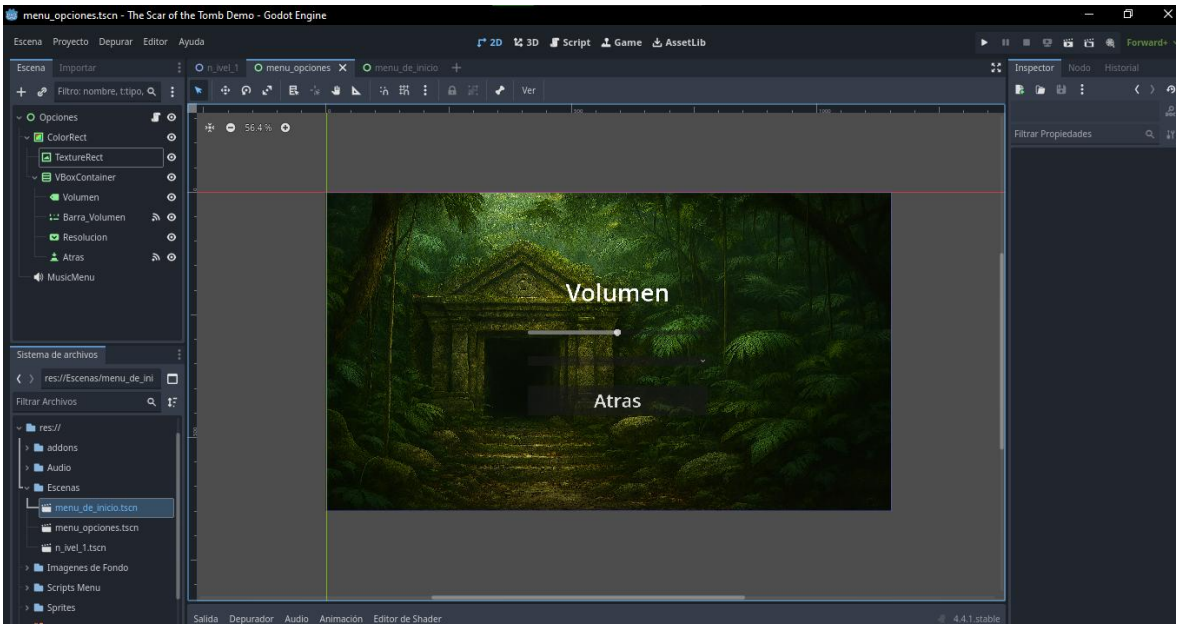
Ventana de Proyectos de Godot



Se observa la interfaz que maneja Godot al iniciar el programa, donde se puede realizar las acciones de editar proyectos, eliminarlos, renombrarlos, importarlos y crear nuevos, de igual manera existe una pestaña en la parte superior para acceder a librerías de assets creados por la comunidad para utilizarlos en los proyectos.

Figura 9

Ventana de Inicio del Proyecto

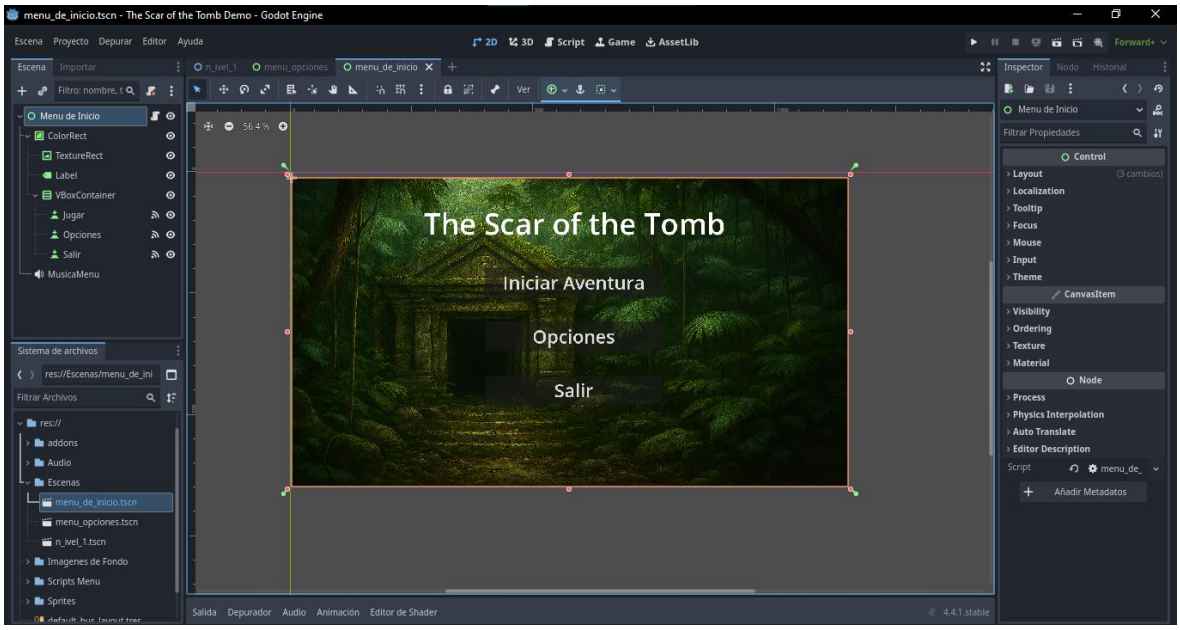


En esta ventana se puede observar las distintas acciones a realizar y herramientas a utilizar con un proyecta ya creado en Godot, siendo la de crear un nodo 2D, nodo 3D, un Script para las funcionalidades de los nodos.

4.4.1. Elementos del Menú Principal

Figura 10

Menú de Inicio del Juego



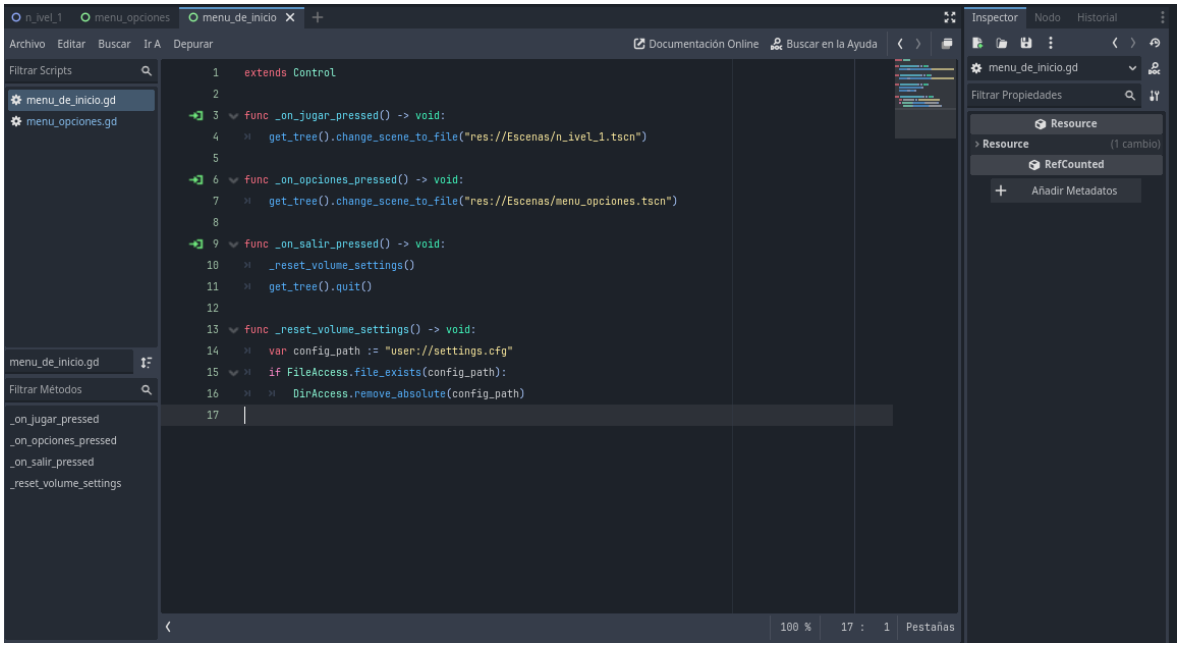
El menú de Inicio del Juego se compone de los siguientes nodos:

- ColorRect: Establece un Rectángulo de un color como fondo y se lo utiliza para definir las dimensiones a utilizar para el menú.
- TextureRect: Establece una imagen como fondo que se adapta al tamaño nodo ColorRect.
- Label: Muestra un texto Plano que se utilizó para colocar el nombre del Juego.
- VBoxContainer: Establece un contenedor el cual organiza los elementos colocados en su interior de manera automática según su disposición.
- Button: Un botón tematizado que puede contener texto e iconos. Se los utilizo para configurar funciones en el script
- AudioStreamPlayer: Nodo utilizado para la reproducción de sonidos. En este caso funciona como la música de fondo del menú.

A continuación, se muestra el Script vinculado al menú de inicio que cuenta con las funciones establecidas para los botones:

Figura 11

Script del Menú de Inicio



En el siguiente Script de Godot utilizando el lenguaje de GDScript, se establecieron las siguientes funciones:

_on_jugar_pressed(): Al hacer clic izquierdo en el botón, va a buscar un árbol que contenga al nodo y realizara un cambio de escena en base a la ruta del archivo que se le entregue, siendo el caso del nivel 1 de la demo.

_on_opciones_pressed(): Al hacer clic izquierdo en el botón, va a buscar un árbol que contenga al nodo y realizara un cambio de escena en base a la ruta del archivo que se le entregue, siendo el caso del menú de opciones generado para realizar cambios en el volumen y resoluciones.

_on_salir_pressed(): Al hacer clic izquierdo en el botón, va a activar la función de _reset_volume_settings() y buscar el árbol que contenga al nodo para cerrar la aplicación de su iteración actual.

_reset_volume_settings(): Establece la variable config_path que almacenara la dirección de un archivo de configuración, luego activa un condicional, donde en caso de que el archivo exista, se lo elimine, caso contrario, no habría acción alguna.

De igual manera, se realizó las mismas acciones para el menú de configuraciones y su script respectivo, con ciertos cambios.

Figura 12

Menú de Opciones del Juego

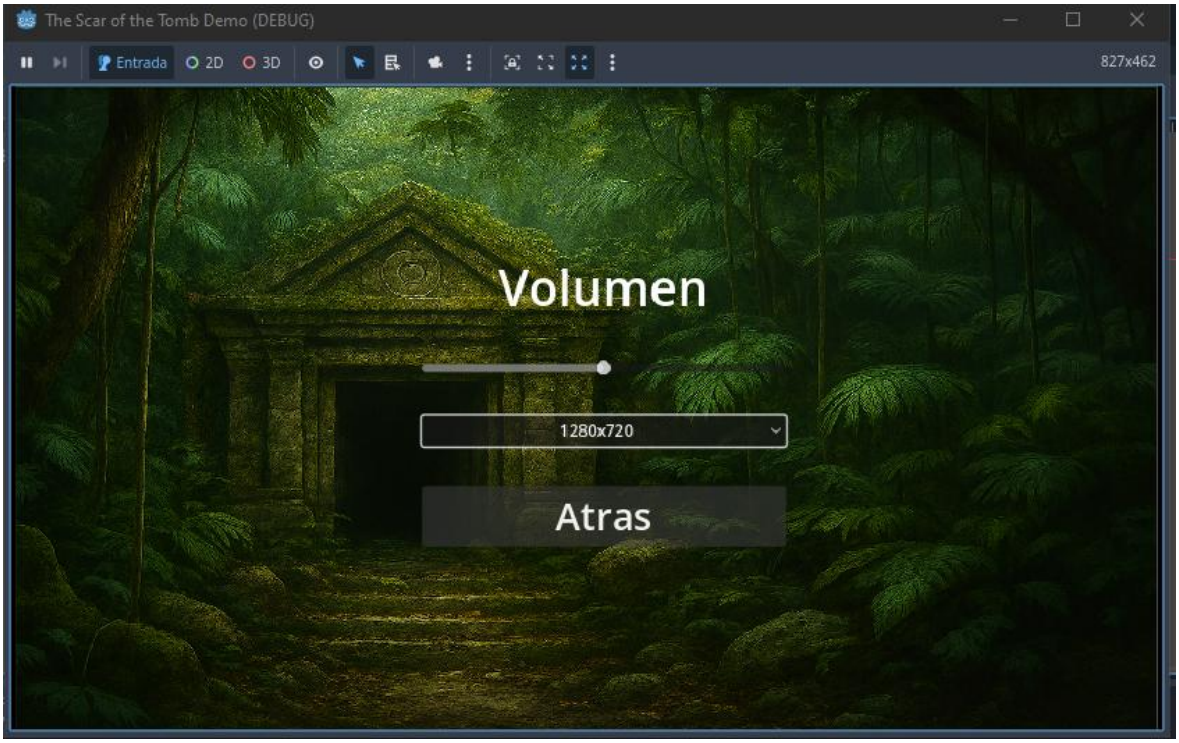


Figura 13

Vista Previa del Script del Menú de Opciones

```gdscript
extends Control

const CONFIG_PATH := "user://settings.cfg"

@onready var volume_slider: Slider = $ColorRect/VBoxContainer/Barra_Volumen
@onready var resolution_button: OptionButton = $ColorRect/VBoxContainer/Resolucion
@onready var menu_music: AudioStreamPlayer

# Lista de resoluciones y pantalla completa al final
var resolutions := [
    Vector2i(1280, 720),
    Vector2i(1600, 900),
    Vector2i(1920, 1080),
    Vector2i(2560, 1440)
]

func _ready() -> void:
    # --- Configurar volumen ---
    volume_slider.min_value = 0.0
    volume_slider.max_value = 1.0
    volume_slider.step = 0.01

    if not volume_slider.value_changed.is_connected(_on_barra_volumen_value_changed):
        volume_slider.value_changed.connect(_on_barra_volumen_value_changed)

    # --- Configurar resoluciones ---
```

El menú de opciones se conforma de los mismos elementos que el menú de inicio, con la diferencia de que cuenta con los siguientes elementos:

HSlider: Una barra deslizante que ajusta su valor de manera horizontal, siendo la izquierda lo mínimo y la derecha lo máximo, se lo utilizo para ajustar el nivel del volumen del bus de audio maestro del proyecto.

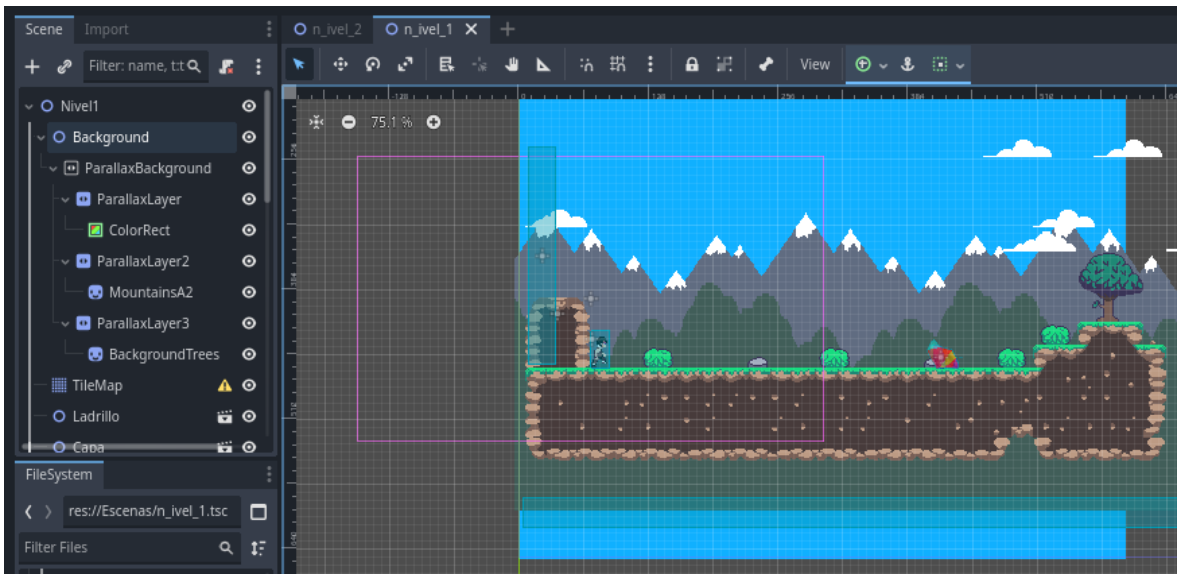
OptionButton: Un botón que despliega un menú de opciones. Se lo utilizo para colocar ciertos parámetros de resolución de pantalla para el gusto del jugador.

En la parte del Script, se configuraron las funciones respectivas para que los cambios de valores de la barra deslizante afecten al bus de audio y esos valores se guarden en un archivo de configuración, de igual manera con el botón de resolución, configurando el tamaño de la ventana de visualización del juego y guardando esos mismos valores en el archivo de configuración.

4.4.2. Elementos de los Niveles

Figura 14

Ventana del Primer Nivel



Cada nivel se encuentra conformado por los siguientes nodos que se encuentran organizados jerárquicamente y permitieron definir tanto la ambientación visual como la estructura jugable del nivel:

ParallaxLayer: Los elementos con los nombres Sky, Mountains, BackgroundsTree y WaterA8Frames funcionaron como las bases para el fondo del nivel, con el objetivo de dar

una ambientación más realista al momento de iniciar el nivel, cada nivel cuenta con varios ParallaxLayer que representan un elemento del fondo del nivel.

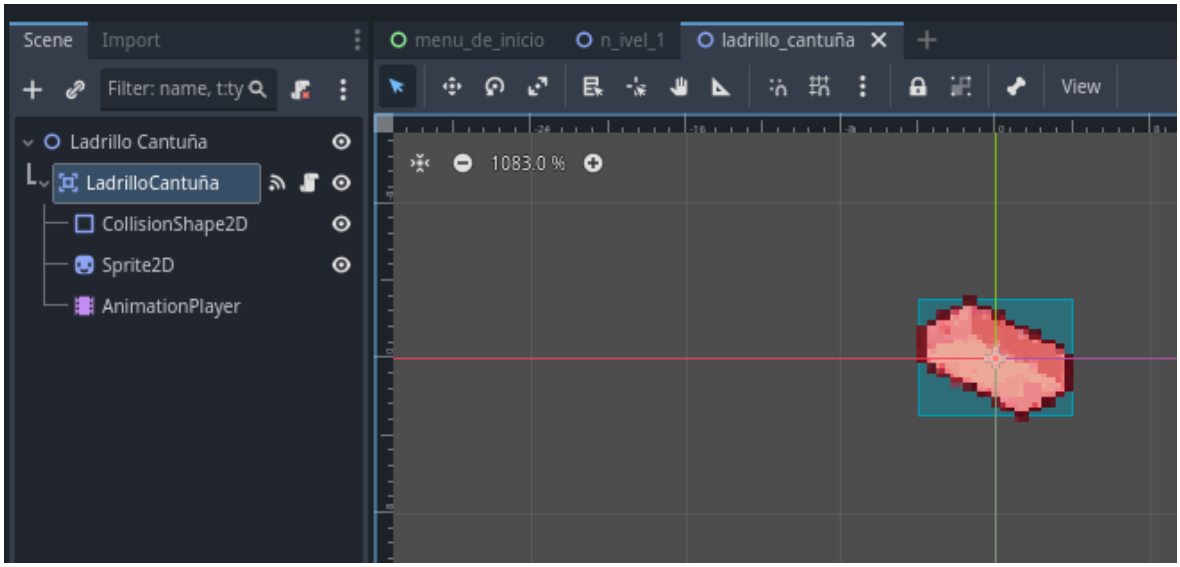
TileMap: A través del TileMap se construyeron plataformas, suelos y estructuras que conformaron el espacio jugable del nivel.

Se definieron las colisiones del suelo y paredes directamente desde el módulo de Tileset, el cual permite definir las dimensiones a utilizar de los sprites junto a su área de colisión. Su uso permitió construir el nivel de manera modular y eficiente, facilitando ajustes posteriores sin necesidad de modificar grandes secciones del escenario.

A continuación, se muestran los nodos heredados al nivel, que cuentan con sus propias funciones y scripts para la interacción del jugador con el nivel:

Figura 15

Ventana del Nodo “Ladrillo_Cantuña”

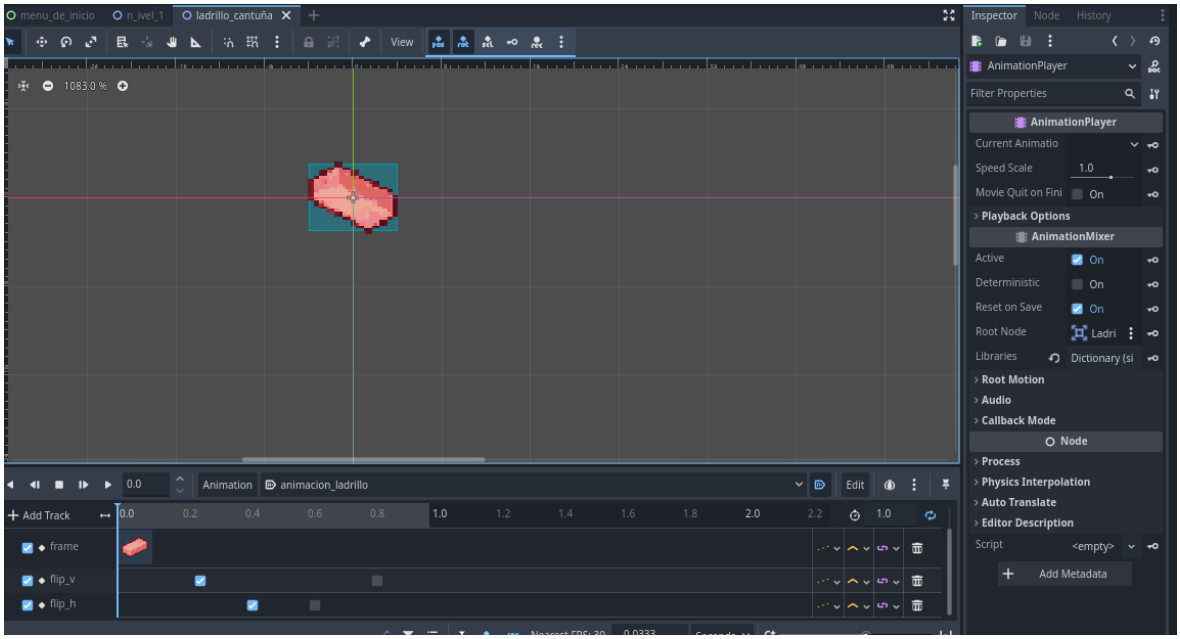


El nodo función de objeto recolectable del nivel, cuenta con los siguientes modulos:

Area2D: LadrilloCantuña funciona como un contenedor para almacenar las funciones de CollisionShape2D, Sprite2D y AnimationPlayer

CollisionShape2D: El nodo permite definir un área para la detección de colisiones del objeto.

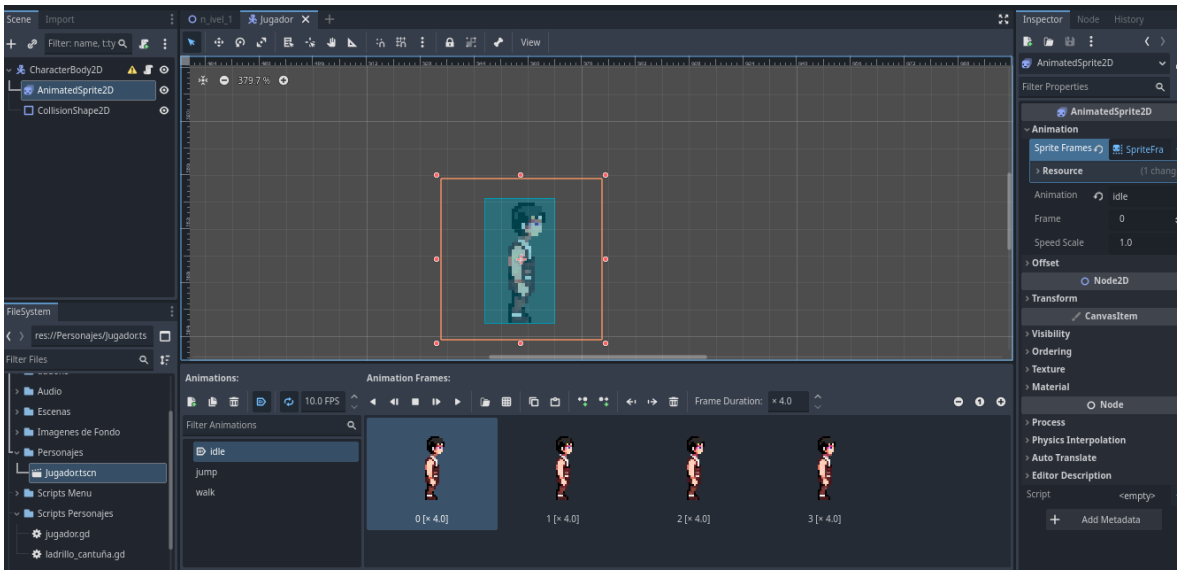
Figura 16
Elementos del Nodo AnimationPlayer



El nodo permitió definir los movimientos que realiza el objeto en un periodo de tiempo definido para crear una vista dinámica del objeto, siendo el caso de que el objeto realiza la misma acción de girar en una dirección en un bucle.

Figura 17

Ventana del Nodo Jugador



El nodo Jugador cuenta con los siguientes módulos para su funcionalidad:

AnimatedSprite2D: El módulo permite generar el movimiento del personaje en base a los diferentes sprites que conformen dicho movimiento. En el caso del proyecto, los

movimientos del personaje fueron los de Idle, cuando no realiza ningún movimiento, “jump”, cuando realiza la acción de saltar, y “walk”, cuando el jugador se mueve de izquierda a derecha en el mapa, la combinación de estas acciones permite darle más realismo al personaje en los niveles.

ColissionShape2D: Define el área de colisión del personaje ante objetos que cuenten con física.

Figura 18

Ventana de Animaciones del Jugador

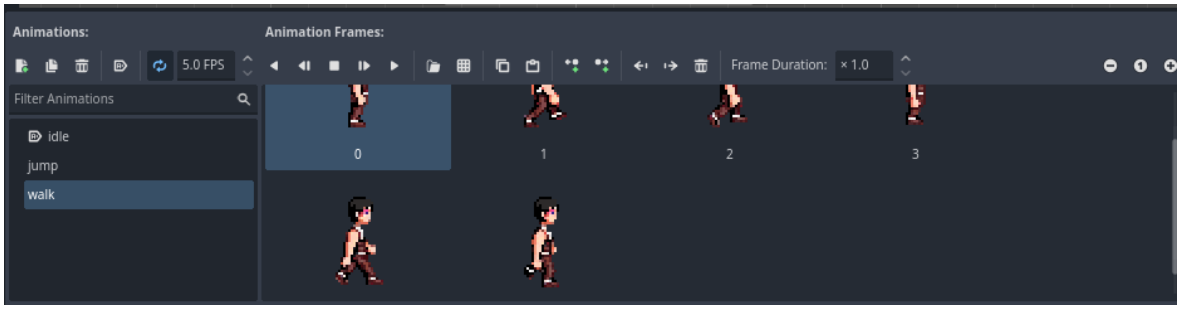


Figura 19

Vista Previa del Script del Jugador

```gdscript
extends CharacterBody2D

const SPEED = 120
const JUMP_FORCE = -250
const GRAVITY = 600

func _physics_process(delta):
    # Aplicar gravedad
    velocity.y += GRAVITY * delta

    # Movimiento horizontal
    var direction = 0
    if Input.is_action_pressed("ui_right"):
        direction = 1
    elif Input.is_action_pressed("ui_left"):
        direction = -1

    velocity.x = direction * SPEED

    # Saltar
    if is_on_floor() and Input.is_action_just_pressed("ui_accept"):
        velocity.y = JUMP_FORCE

    # --- ANIMACIONES ---
    var anim = $AnimatedSprite2D
```

El Script del Jugador cuenta con distintas variables y funciones para su funcionalidad:

SPEED, JUMP_FORCE, GRAVITY: constantes que definen la velocidad de movimiento, fuerza de salto y fuerza que atrae al jugador al suelo, las constantes SPEED y GRAVITY

fueron utilizadas para crear las físicas con las que el jugador se mueve en los niveles en la función _physics_process.

Dentro de la misma función se establecieron los controles de teclado con los cuales el jugador logra controlar al personaje, “ui_right” representa la flecha derecha del teclado, “ui_left” representa la flecha izquierda y “ui_accept” representa la barra espaciadora, estos elementos se encuentran conectados a la función “Input.is_action_pressed” para que el motor detecte cuando se presiona dichas teclas e interprete la acción definida.

Figura 20

Pantalla de Visualización de cada Nivel

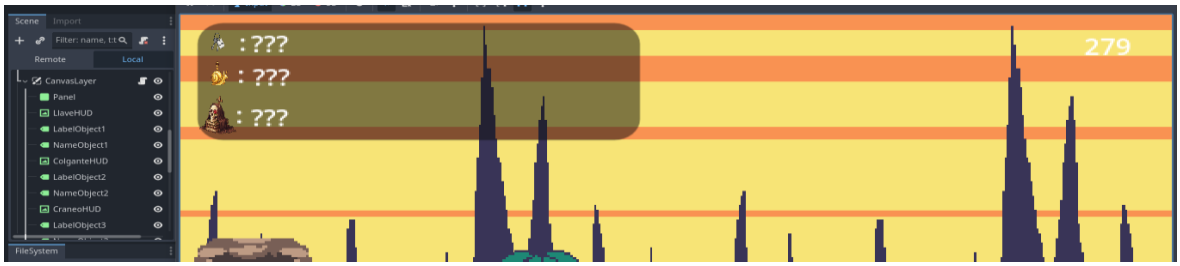


Figura 21

Pantalla de Nivel Completado

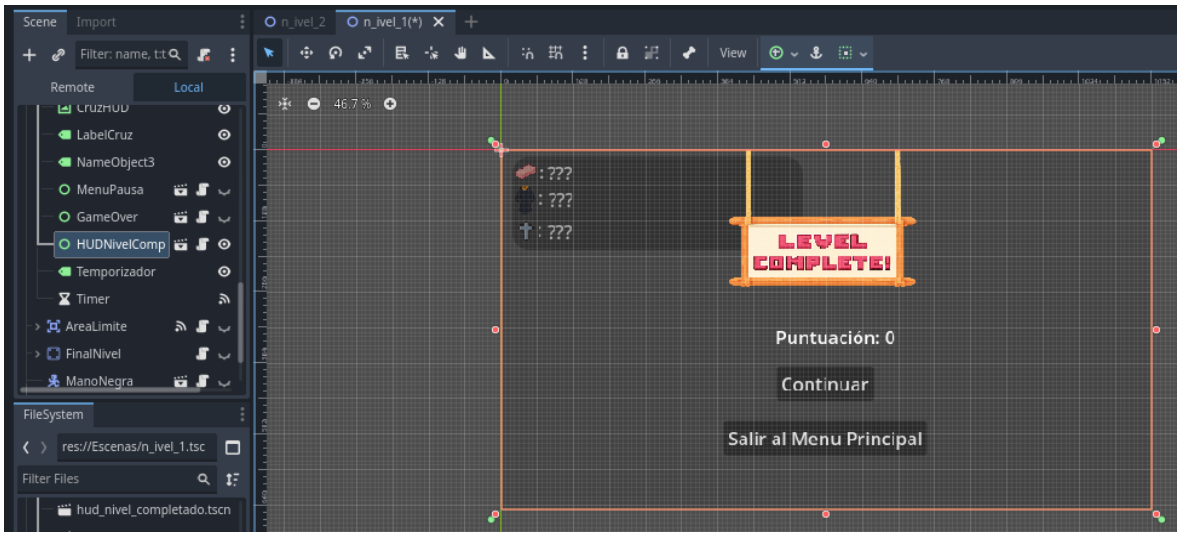
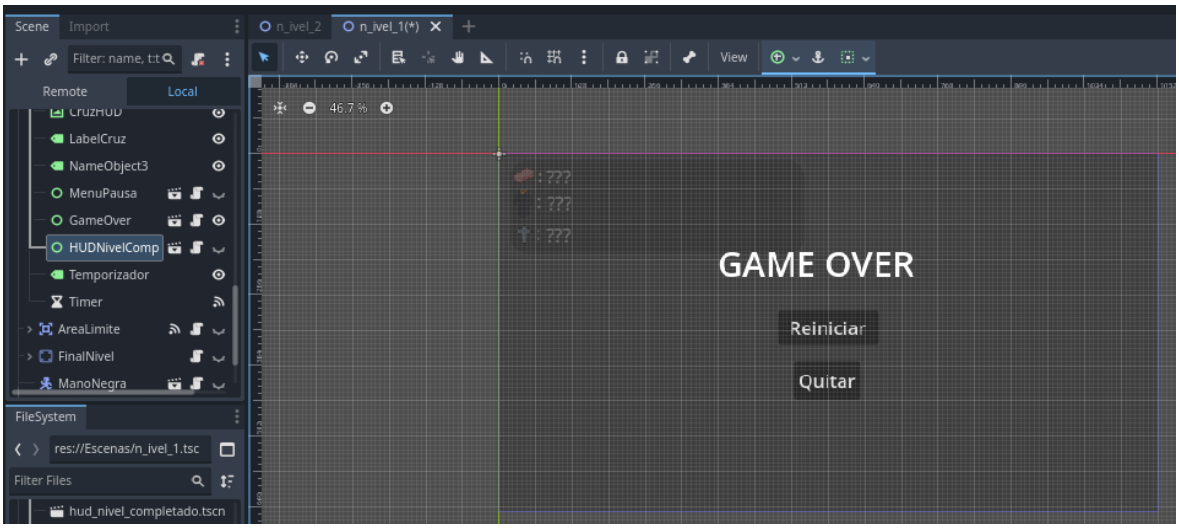


Figura 22

Pantalla de Nivel Fallido



La pantalla de visualización de cada nivel se encuentra compuesta por diferentes nodos de tipo control que representan los elementos que el jugador visualiza al iniciar cada nivel, siendo los nombres que adquieren los objetos recolectables, el temporizador, la pantalla de pausa y de juego terminado, tanto en caso de haber completado el nivel, como de haber fallado por diversos factores.

Cada pantalla cuenta con su respectivo script con funcionalidades para los botones y cambios dinámicos al momento de recolectar los objetos.

Figura 23

Vista Previa del Script de la Pantalla de Visualización

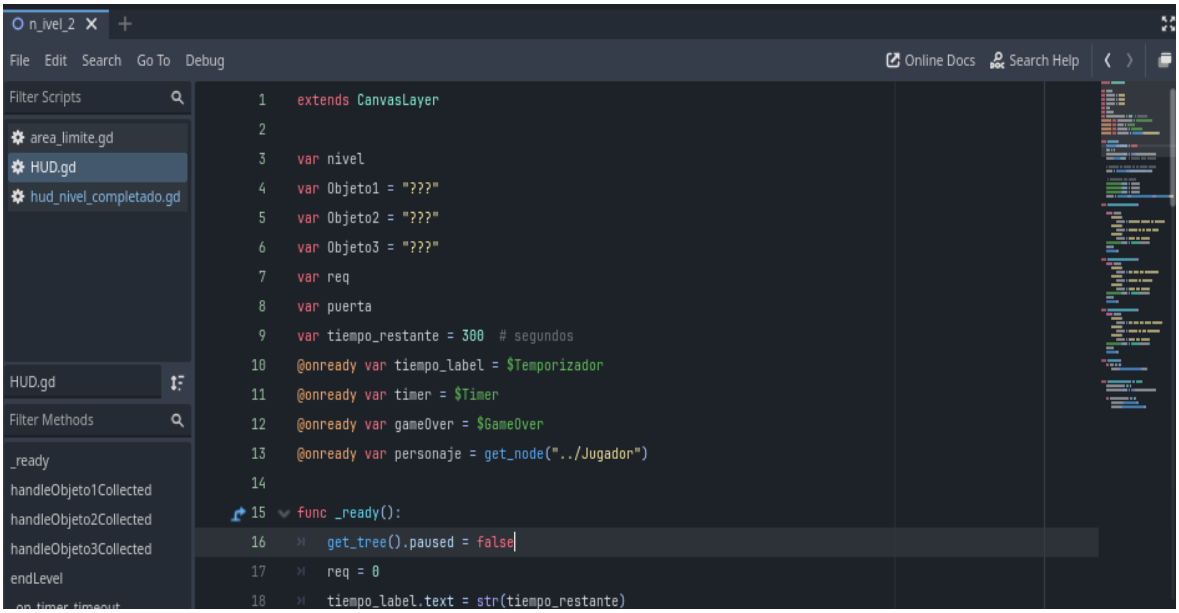


Figura 24

Vista Previa del Script de la Pantalla de Pausa

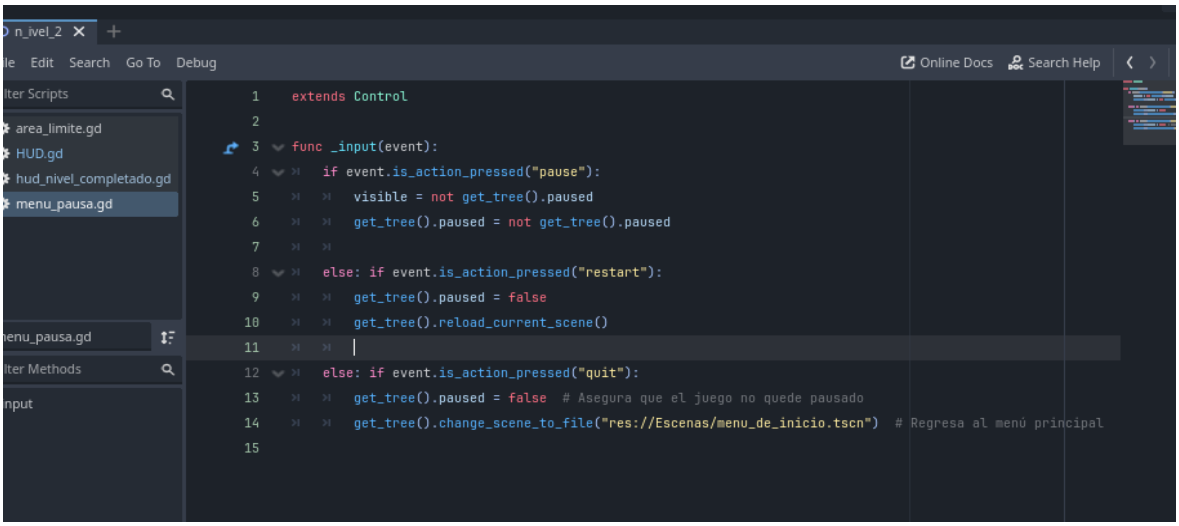
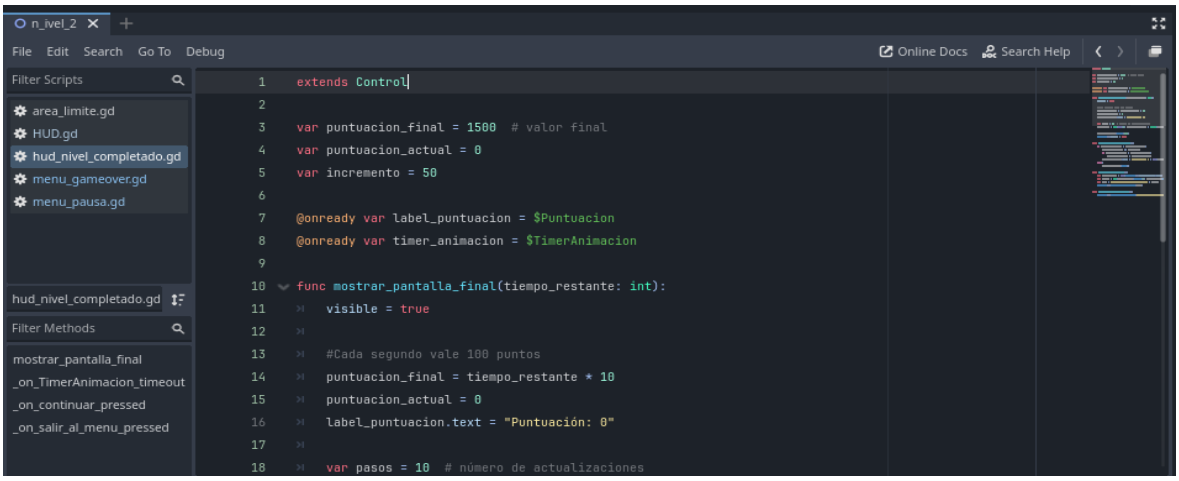


Figura 25

Vista Previa del Script de Nivel Fallido

Figura 26

Vista Previa de Script de Nivel Completado



4.5. Pruebas del Videojuego

Después de que se implementaron los componentes de la demo, se llevaron a cabo una serie de pruebas para verificar que el videojuego cumpliera con los requisitos funcionales y no funcionales previamente definidos. Estas pruebas se realizaron durante la fase posterior al desarrollo.

4.5.1. Pruebas Unitarias

Las pruebas unitarias tenían como objetivo verificar el funcionamiento individual de las mecánicas básicas del videojuego, evaluando cada componente de manera aislada antes de su integración con el resto del sistema. Estas pruebas se centraron principalmente en las acciones del jugador, la detección de colisiones, la recolección de objetos y el control del tiempo dentro de los niveles.

Tabla 9

Pruebas Unitarias Realizadas

ID	Componente evaluado	Descripción	Resultado esperado	Resultado obtenido
PU-01	Movimiento horizontal	Presionar las teclas izquierda y derecha	El personaje se desplaza de forma fluida en ambas direcciones	Correcto

ID	Componente evaluado	Descripción	Resultado esperado	Resultado obtenido
PU-02	Sistema de salto	Presionar la tecla de salto	El personaje realiza el salto con altura adecuada	El personaje si salta
PU-03	Colisiones	Colisión con plataformas y suelo	El personaje no atraviesa el terreno	El personaje se mantiene en el terreno
PU-04	Recolección de objetos	Colisión con objeto coleccionable	El objeto desaparece y aumenta el contador	En la pantalla se muestra el nombre del objeto
PU-05	Temporizador	Inicio de un nivel	El temporizador inicia en 300 segundos	El temporizador comienza al iniciar el nivel
PU-06	Menú de pausa	Activar pausa durante la partida	El juego se detiene y muestra el menú de pausa	Correcto

Estos resultados demuestran que las mecánicas evaluadas funcionaron correctamente de forma individual, cumpliendo con los requerimientos funcionales establecidos.

4.5.2. Pruebas de Integración

Las pruebas de integración fueron realizadas con el objetivo de comprobar el correcto funcionamiento de los distintos sistemas del videojuego en conjunto, evaluando la interacción entre las mecánicas implementadas. Estas pruebas fueron realizadas con los escenarios completos, verificando su funcionalidad desde el inicio del nivel hasta el final.

Tabla 10

Pruebas de Integración

ID	Escenario evaluado	Descripción	Resultado esperado	Resultado obtenido
PI-01	Inicio de nivel	Iniciar partida desde el menú principal	El nivel carga correctamente y el jugador puede moverse	Correcto
PI-02	Recolección + progreso	Recolectar todos los objetos del nivel	Se habilita el acceso al siguiente nivel	Correcto
PI-03	Temporizador + derrota	Agotar el tiempo del nivel	La partida finaliza y muestra pantalla correspondiente	Correcto
PI-04	Pausa + reanudación	Pausar y continuar la partida	El juego se reanuda sin errores	Correcto
PI-05	Reinicio de nivel	Reiniciar desde el menú de pausa	El nivel se reinicia correctamente	Correcto

Las pruebas demostraron que el sistema del videojuego interactúa correctamente sin generar errores de sincronización ni comportamientos inesperados durante la ejecución.

4.5.3. Pruebas de Usabilidad

Dado que el videojuego está orientado a una experiencia interactiva, se realizaron pruebas de usabilidad con el fin de evaluar la facilidad de uso, claridad de la interfaz y comprensión de las mecánicas por parte del jugador.

Estas pruebas se efectuaron mediante sesiones de juego controladas, observando el comportamiento del usuario y su interacción con los menús y controles.

Tabla 11

Pruebas de Usabilidad

ID	Aspecto evaluado	Descripción de la prueba	Resultado esperado	Resultado obtenido
PU-01	Controles	Uso de teclas de movimiento y salto	El jugador comprende los controles sin instrucciones externas	Aceptable
PU-02	Interfaz	Visualización del HUD	La información es clara y legible	Correcto
PU-03	Menús	Navegación en menú principal y pausa	Navegación intuitiva	Correcto
PU-04	Objetivo del nivel	Comprensión de la meta	El jugador identifica que debe recolectar objetos	Aceptable

Capítulo 5: Manual de Elaboración del Videojuego

5.1. Objetivo de la Guía

El objetivo principal de este manual fue proporcionar una documentación clara y organizada del proceso de creación de la demo del videojuego 2D “The Scar of the Tomb”, desarrollada en Godot Engine. Esta guía fue concebida como un recurso de apoyo para estudiantes y desarrolladores principiantes interesados en comprender el flujo de trabajo seguido durante la creación de un videojuego de aventura y plataformas en 2D.

Además, el manual permitió la consolidación de la experiencia adquirida a lo largo del curso del proyecto, convirtiéndolo en una guía muy práctica que guía a los estudiantes sobre cómo estructurar una escena, utilizar nodos, implementar las mecánicas más básicas, todos los elementos visuales y de audio del entorno de Godot, etc.

5.2. Proceso de Documentación del Videojuego

Esta documentación se llevó a cabo en paralelo con el desarrollo del juego, recopilando datos según fuera necesario, además de reunir piezas de información en cada etapa de implementación. Para este propósito, se tomaron capturas directas del entorno de desarrollo de Godot Engine y las documentaciones descriptivas también incluyeron explicaciones de los nodos y componentes utilizados en cada escena para su función y propósito.

La documentación se dividió en varios niveles; al principio, cubriendo aspectos básicos del proyecto (menú principal, configuración inicial del juego); avanzando lentamente hacia sistemas más complejos (control de personajes, diseño de niveles, gestión de la interfaz de usuario). Así, el contenido pudo presentarse con un enfoque estructurado propio, promoviendo esto para consulta y reutilización.

5.3. Estructura y Contenido del Manual

La organización del manual se dispuso con las escenas y piezas esenciales que formaron la demo del videojuego "La Cicatriz de la Tumba". En cada parte del manual, se da una descripción del entorno de creación dentro de Godot Engine y luego los nodos utilizados y lo que hacen en el juego en su conjunto.

Esto permitió la construcción paso a paso a través de un flujo coherente de cada juego, desde el menú principal hasta las escenas del juego, niveles y los sistemas de interfaz de usuario que se implementaron. Y así ayudó a explicar visualmente a los lectores y ayudarlos a comprender la jerarquía de nodos y cómo funcionaba la lógica en cada componente.

5.3.1. Menú de Inicio de Videojuego

La Figura 10 es la escena del menú de inicio del videojuego. Esta escena estaba destinada a preparar al jugador para la aventura y acceder a las opciones de juego o salir de la aplicación. El menú de inicio contiene los siguientes nodos principales:

- ColorRect: Utilizado para establecer un fondo base y definir el área visible del menú.
- TextureRect: Permitió mostrar una imagen de fondo adaptada al tamaño del contenedor.

- Label: Se empleó para presentar el título del videojuego en pantalla.
- VBoxContainer: Organizó de manera automática los elementos del menú, facilitando su alineación vertical.
- Button: Se utilizaron botones interactivos para las opciones Iniciar Aventura, Opciones y Salir.
- AudioStreamPlayer: Se integró para reproducir la música de fondo del menú principal.

La correcta organización de estos nodos permitió una navegación intuitiva y una presentación visual coherente con la temática del videojuego.

5.3.2. Escena del Nivel del Videojuego

La Figura 14 muestra la jerarquía de nodos del Nivel 1 del videojuego La Cicatriz de la Tumba, desarrollado en Godot Engine. Queríamos hacer el primer nivel jugable de la demo para que los jugadores se familiarizaran con la idea básica de movimiento, exploración y recolección de objetos. Los niveles futuros tenían estructura y función idénticas y no eran más que el patrón y la disposición de los conjuntos de mosaicos aplicados en el diseño del terreno. La escena del nivel se estructuró de manera tanto jerárquica como modular para proporcionar una interacción más cohesiva entre las características visuales, físicas y de juego del sistema.

Incluidos en los nodos centrales de la escena están los nodos principales:

- ParallaxBackground: Se utilizó para implementar el fondo del escenario, permitiendo un efecto de desplazamiento diferencial que aportó profundidad visual al nivel.
- ParallaxLayer: Se emplearon varias capas de parallax para separar visualmente los distintos planos del fondo, como el cielo, montañas y árboles, logrando una sensación de distancia entre los elementos.
- ColorRect: Se usó como base de color para el fondo, asegurando una correcta visualización del escenario y una transición visual uniforme.
- TileMap: Constituyó el elemento principal para la construcción del terreno y las plataformas del nivel. A través de este nodo se definieron las superficies transitables, obstáculos y límites del escenario, facilitando la reutilización de patrones en los diferentes niveles del juego.

- CharacterBody2D: Representó al personaje controlado por el jugador. Este nodo gestionó el movimiento, los saltos y la interacción con el entorno mediante colisiones.
- CollisionShape2D: Se utilizó para definir las áreas de colisión tanto del personaje como del entorno, garantizando una interacción coherente entre los elementos del nivel.

Que esta escena estuviera separada del fondo, el terreno jugable y el personaje hizo que tanto el desarrollo del juego como las futuras alteraciones fueran mucho más fáciles. Este patrón se utilizó en otros niveles de la demo del juego también, ayudando a mantener el mismo comportamiento que el juego y así la creación de nuevos escenarios también sería más fácil. La escena del nivel en general no solo sirvió para un propósito estético, sino que también delineó qué áreas del juego eran jugables y qué mecanismos de movimiento y flujo de interacción se estaban llevando a cabo entre el jugador y el entorno, formando esencialmente una base sólida para las siguientes etapas del desarrollo del juego.

5.3.3. Configuración y Uso de Tilemap en el nivel

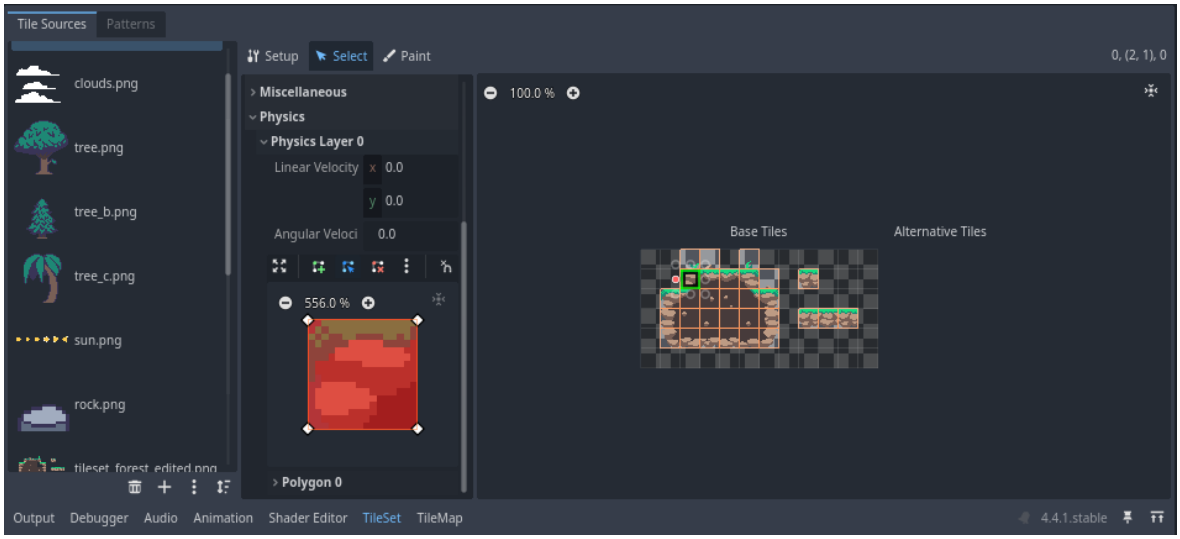
El TileMap fue uno de los componentes fundamentales en la construcción de los niveles del videojuego, ya que permitió modularizar y organizar sus diseños de suelo y plataformas de manera efectiva. Durante la demo de La Cicatriz de la Tumba, este nodo ha definido superficies transitables, ubicaciones delimitadas del escenario y obstáculos a lo largo de los niveles.

Como se muestra en la Figura 14, el nodo TileMap pertenecía a la jerarquía principal del nivel y funcionaba igual para todos los niveles del juego. A pesar de la forma en que los mosaicos se reorganizaron en varios niveles, el diseño general del mapa de TileMap era estático para hacer que el entorno se comportara de la misma manera.

El TileMap se configuró utilizando un TileSet creado al momento de crear el nodo, que describe una imagen en formato PNG y contiene los varios bloques de terreno, posiciones y elementos decorativos del caso. Para cada mosaico consideramos su forma de colisión y definimos esto basado en los niveles de colisión, permitiendo que el personaje interactúe correctamente con el entorno sin definir niveles de colisión uno por uno.

Figura 27

Ventana de Configuración de Tileset



Entre las principales funciones del TileMap dentro del videojuego se destacan las siguientes:

- Construcción rápida y flexible de terrenos y plataformas.
- Definición de áreas transitables y zonas no accesibles para el jugador.
- Control central de colisiones ambientales.
- Reutilización de patrones de diseño entre diferentes niveles del juego.

La utilización del TileMap hizo que la iteración durante la fase de desarrollo fuera mucho más fácil, permitiendo rehacer el diseño de niveles sin alterar la lógica del juego o los scripts de los personajes. Además, el sistema de colisiones que emplea Godot es la razón por la cual las mecánicas del juego permanecen muy estables y uniformes; está presente durante todo el juego.

Los niveles posteriores reutilizaron el TileMap usando la misma estructura y solo cambiando su disposición y complejidad a medida que se cambiaban las escenas, pero con una complejidad de diseño variable para aumentar gradualmente la dificultad sin cambiar el juego.

5.3.4. Estructura y Comportamiento del personaje jugador

Como se observa en la Figura 17, el personaje principal del videojuego se realizó como una escena independiente, permitiendo la repetición del personaje del juego a lo largo del mismo. Esta escena se basó en un nodo CharacterBody2D, apropiado para gestionar el movimiento, la física y las colisiones en juegos de plataformas 2D.

La jerarquía del jugador estuvo compuesta por los siguientes nodos principales:

- CharacterBody2D (Jugador): Nodo raíz encargado de gestionar el movimiento, la gravedad y la detección de colisiones.
- AnimatedSprite2D: Responsable de mostrar las animaciones del personaje, como reposo, desplazamiento y salto.
- CollisionShape2D: Define el área física del personaje para la detección de colisiones con el entorno y otros objetos.

Esta organización facilitó la separación de responsabilidades y el control preciso del comportamiento del personaje durante la partida.

La lógica que impulsaba las acciones del usuario se controlaba mediante un script relacionado con el nodo principal, dicho script permitió:

- El desplazamiento horizontal del personaje hacia la izquierda y la derecha.
- La ejecución de saltos, considerando la gravedad y el contacto con el suelo.
- La activación automática de animaciones según el estado del personaje (reposo, movimiento o salto).
- La detección de daño al entrar en contacto con enemigos.
- La interacción con el entorno mediante colisiones.
- Funcionalidad de recolección de objetos en base al ID del objeto con el que entra en contacto.

5.3.5. Estructura y Comportamiento del enemigo

Los enemigos se utilizaron como escenas independientes para reutilización y mantenimiento, con los siguientes nodos como componentes de su estructura:

- CharacterBody2D (Enemigo): Nodo raíz que determina el movimiento y las colisiones.
- AnimatedSprite2D: Maneja las animaciones de los enemigos.
- CollisionShape2D: Define el área de colisión física.
- RayCast2D (DetectarPiso): Permite verificar el contacto con el suelo.
- RayCast2D (DetectarPared): Detecta obstáculos para cambiar la dirección de movimiento.
- Area2D: Encargada de detectar la colisión con el jugador para aplicar daño.

El enemigo fue programado para realizar un movimiento automático de patrullaje horizontal dentro del nivel. Su comportamiento incluyó:

- Desplazamiento continuo en una dirección predeterminada.
- Cambio de dirección al detectar una pared o al perder contacto con el suelo.
- Interacción con el jugador, aplicando daño al entrar en contacto.

Este comportamiento se logró mediante el uso del nodo RayCast2D, que proporcionaba un medio de simular inteligencia simple sin necesidad de algoritmos de alto nivel y se centraba en el juego y el rendimiento.

5.3.6. Estructura y Comportamiento de los objetos recolectables

Los objetos recolectables, como llaves u otros ítems, fueron diseñados como escenas independientes para facilitar su instanciación en los niveles. Su estructura incluyó:

- Area2D: Nodo principal que detecta la interacción con el jugador.
- Sprite2D: Representa visualmente el objeto dentro del nivel.
- CollisionShape2D: Define el área de detección para la recolección.

Cuando el jugador entró en contacto con el área del objeto recolectable:

- El sistema detectó la colisión mediante el nodo Area2D.
- El objeto fue contabilizado como recolectado.
- El elemento fue eliminado de la escena para evitar múltiples recolecciones.

Este mecanismo permitió implementar de forma sencilla la condición necesaria para completar los niveles, basada en la recolección total de objetos.

5.3.7. HUD e Interfaz de Usuario del Videojuego

La interfaz de usuario del videojuego se diseñó con el objetivo de proporcionar información clara y accesible al jugador durante la partida, así como permitir la interacción con los distintos estados del juego, tales como pausa, finalización de nivel y fin de partida. La utilización de nodos de tipo Control, los cuales funcionan para la creación de interfaces escalables y adaptables, y el nodo tipo CanvasLayer, que se utilizó para la visualización de los elementos a observar en el desarrollo de los niveles garantizo que el jugador tuviera consistentemente acceso a información relevante mientras construía niveles.

5.3.7.1. HUD de Juego

El HUD del videojuego fue implementado mediante un nodo de tipo CanvasLayer, lo que permitió que los elementos de interfaz se mantuvieran siempre visibles en pantalla, independientemente del movimiento de la cámara o del desplazamiento del escenario. Esta decisión garantizó una visualización constante de la información relevante para el jugador durante el desarrollo de cada nivel.

Dentro del CanvasLayer algunos componentes de interfaz, tales como etiquetas y contenedores, se organizaron de manera que mostraran información clave del estado del juego, entre ellos:

- Temporizador del nivel.
- Cantidad de objetos recolectados.

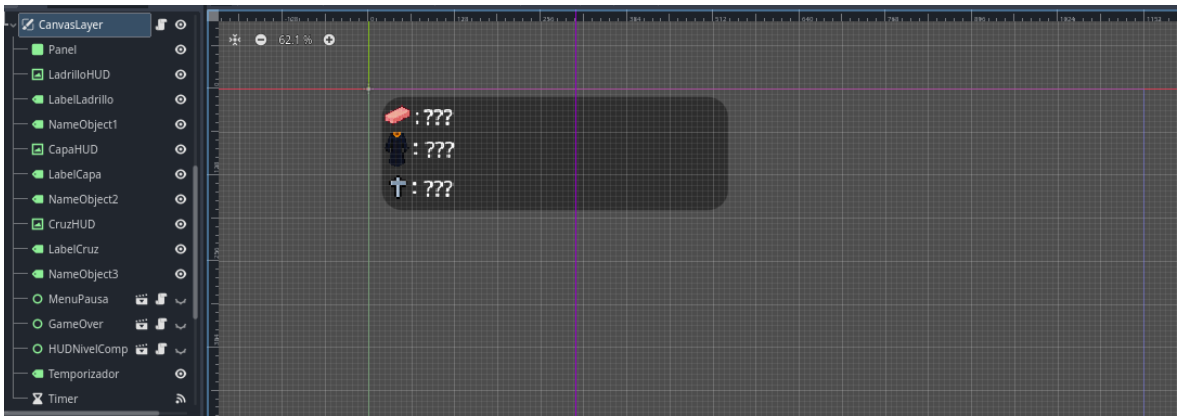
La lógica de funcionamiento del HUD fue centralizada en el script HUD.gd, que era para gestionar dinámicamente los parámetros indicados en la pantalla. Este script permitió desacoplar la interfaz gráfica de la lógica principal del juego, facilitando su mantenimiento y reutilización en distintos niveles.

A través del uso de variables y funciones definidas en el script, el HUD reaccionó a eventos del juego como la recolección de objetos, la disminución del tiempo disponible y la finalización del nivel, asegurando que la información presentada al jugador fuera coherente con el estado actual de la partida.

El uso de CanvasLayer en conjunto con un script dedicado permitió una estructura clara y ordenada de la interfaz, además de contribuir a una mejor experiencia de usuario al mantener los elementos visuales siempre accesibles y correctamente sincronizados con la jugabilidad.

Figura 28

Organización del CanvasLayer



5.3.7.2. Escena de Pausa

El menú de pausa fue implementado mediante una escena independiente Esta interfaz se mostró cuando el jugador detuvo la partida al presionar una tecla, permitiendo suspender temporalmente las acciones del jugador sin perder el progreso actual del nivel.

El menú de pausa incluyó las siguientes opciones:

- Continuar: reanuda la partida desde el punto en que fue pausada.
- Reiniciar nivel: reinicia el nivel actual desde su estado inicial.
- Salir al menú principal: retorna al menú principal del videojuego.

Estas opciones fueron controladas mediante teclas asignadas en la sección de “Mapa de Entrada” en las configuraciones del proyecto, siendo las teclas “P”, “R” y “Q” respectivamente para las acciones de reanudar, reiniciar y salir del nivel.

Figura 29

Ventana de Interfaz de Menú de Pausa

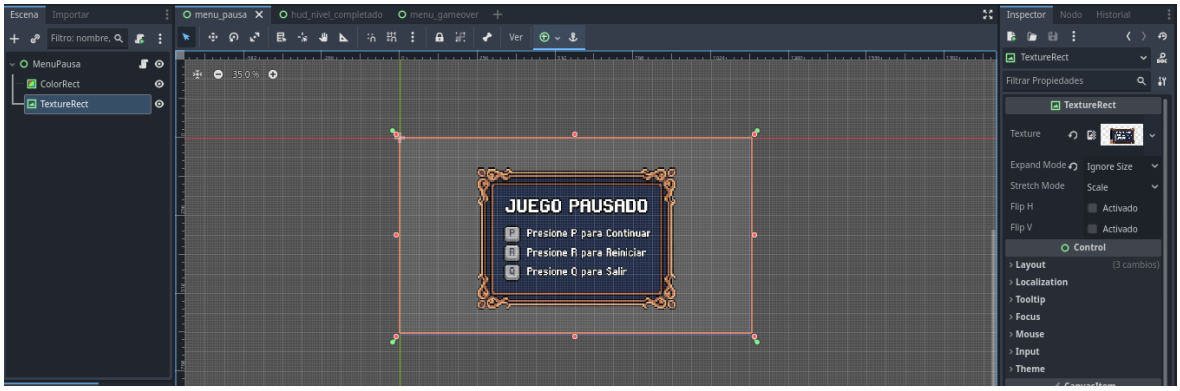
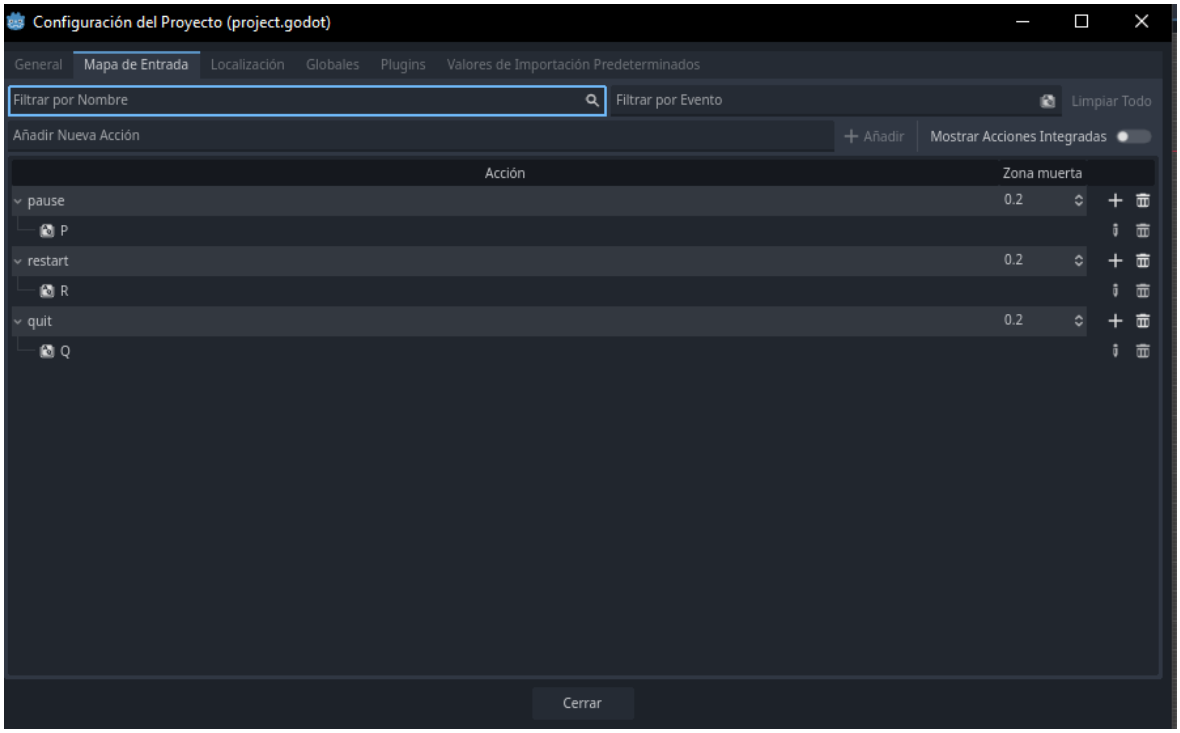


Figura 30

Configuración del Mapa de Entrada



5.3.7.3. Escena de Game Over

La escena de “Game Over” fue utilizada para representar el estado de fin de partida cuando el jugador no logró completar el nivel bajo las condiciones establecidas.

Esta interfaz incluyó:

- Un mensaje de Game Over claramente visible.
- Opciones para reiniciar el nivel o volver al menú principal.

La implementación de esta pantalla permitió cerrar el ciclo de juego de manera clara, brindando al jugador la posibilidad de intentar nuevamente o finalizar la sesión.

5.3.7.4. Escena de Juego Completado

Este nodo representó la interfaz que se mostró al finalizar exitosamente un nivel. Esta pantalla permitió informar al jugador que los objetivos habían sido cumplidos y presentar un resumen del desempeño obtenido.

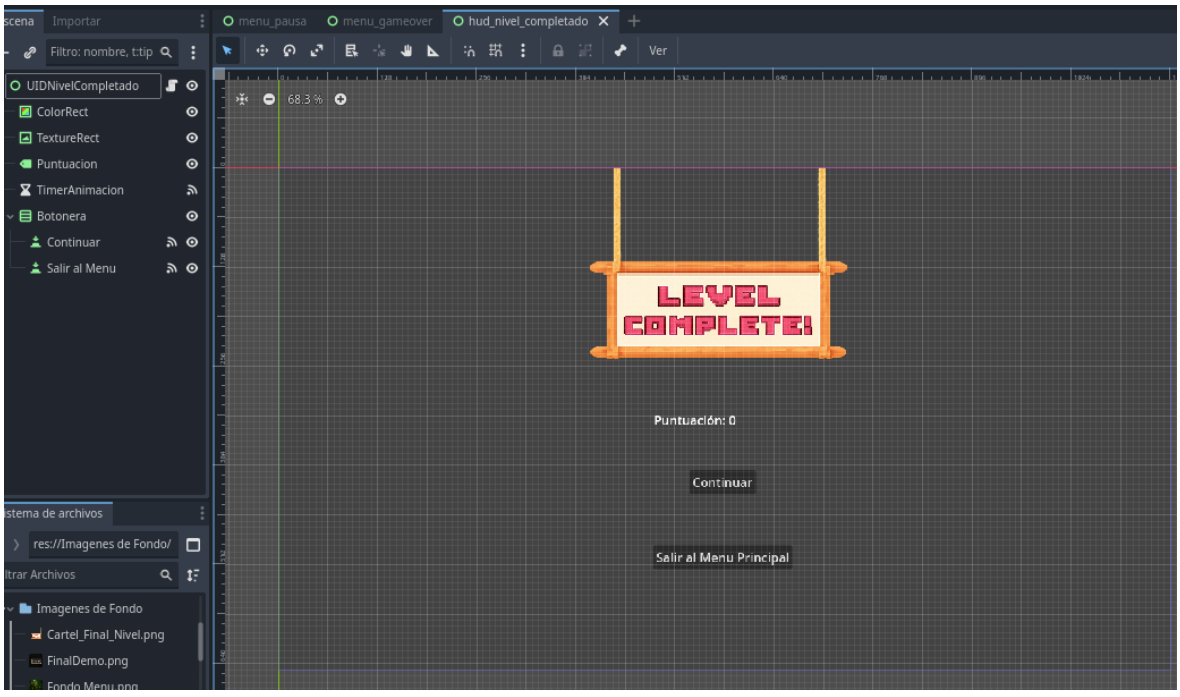
Entre los elementos visuales incluidos se encontraron:

- Mensaje informativo indicando la finalización del nivel.
- Datos de progreso, siendo el puntaje obtenido del tiempo restante.
- Botones de navegación, que permitieron avanzar al siguiente nivel o regresar al menú principal.

El script respectivo se encargó de recibir los datos del nivel completado y actualizar dinámicamente los valores mostrados en pantalla.

Figura 31

Ventana de Escena de Nivel Completado



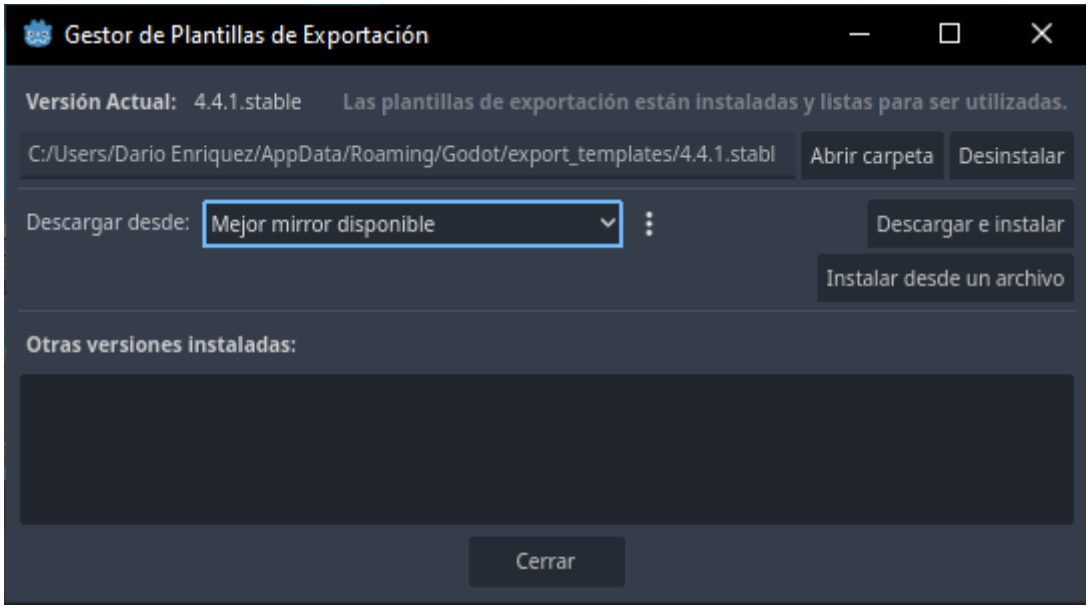
5.3.8. Proceso de Exportación

Con la finalización de la implementación e integración de las mecánicas principales relatadas, se inició el proceso de exportación, primero obteniendo las plantillas de exportación necesarias para generación del archivo ejecutable. Estas plantillas pueden

obtenerse de la página oficial de Godot y del mismo editor, descargando siempre la versión más actual al momento de realizar la búsqueda.

Figura 32

Ventana de Gestor de Plantillas de Exportación



De esa manera en la sección de exportación, ya se encuentra gestionada con los ajustes preestablecidos para generación del archivo ejecutable, el cual comprime todos los archivos que componen al juego y lo encriptan en base a la plantilla obtenida como método de protección ante la replicación de archivos.

En esta sección se configuraron los siguientes parámetros principales:

- Nombre del ejecutable del videojuego.
- El tipo de Formato de Encriptación.
- Inclusión automática de los recursos del proyecto.

Figura 33

Ventana de Exportación

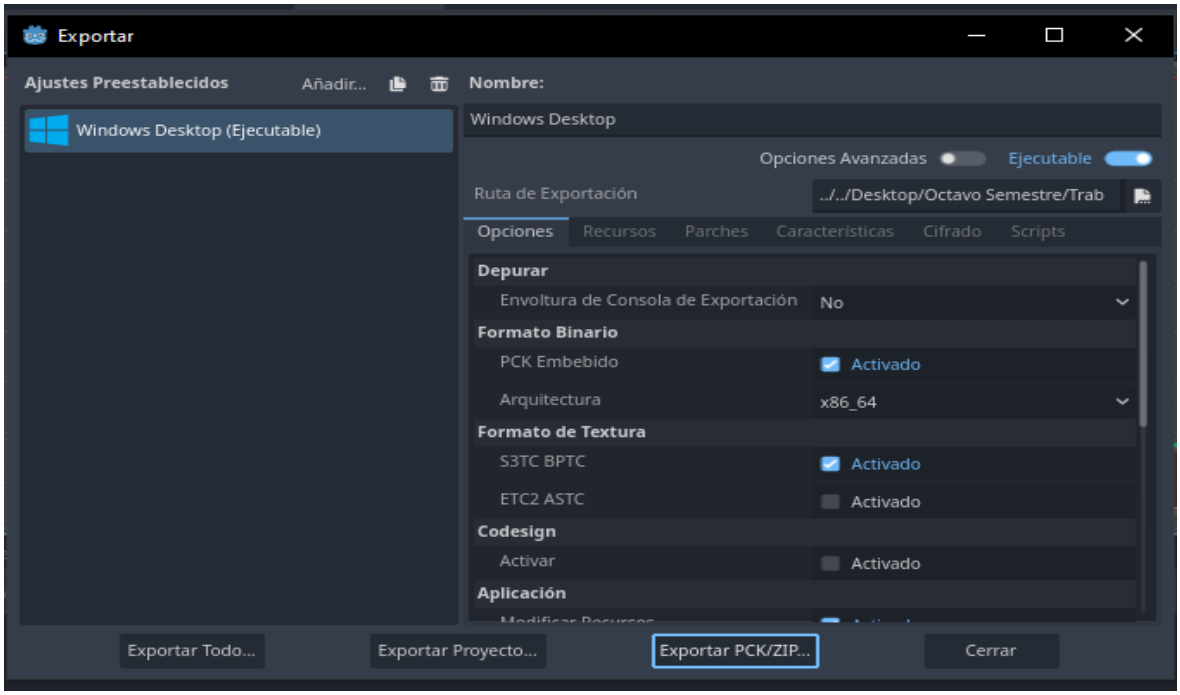
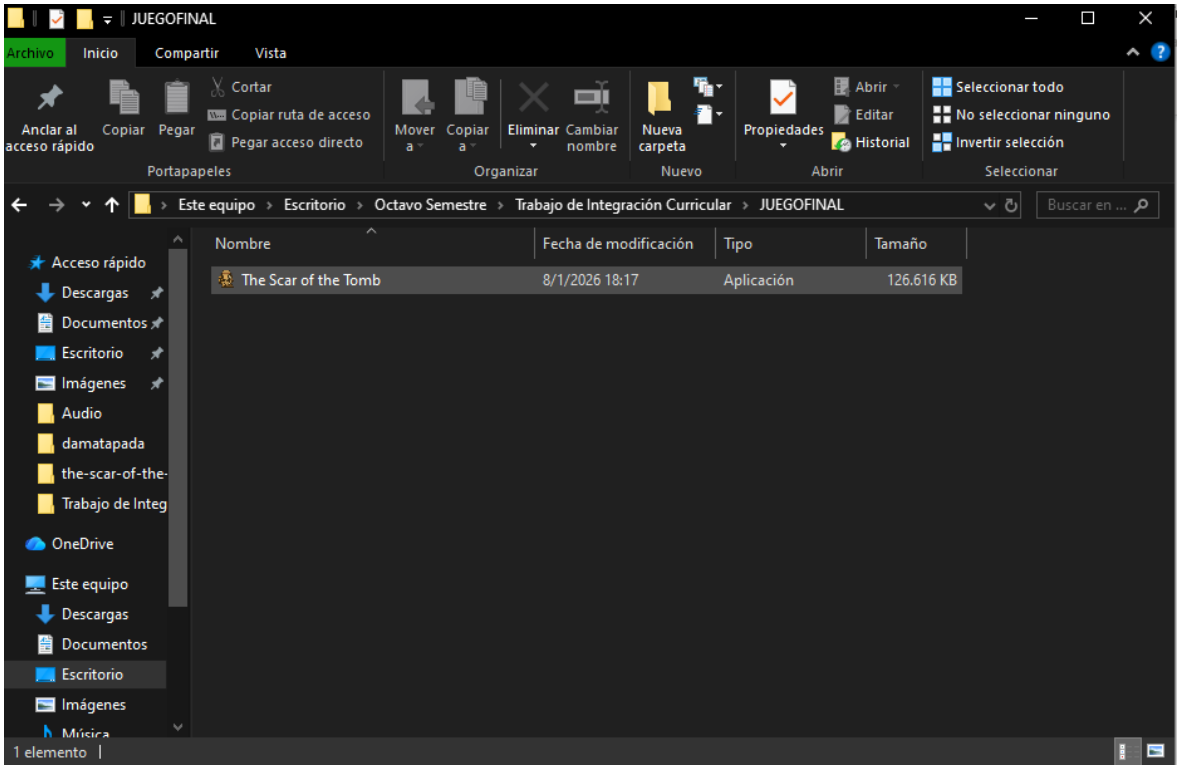


Figura 34

Archivo Ejecutable



5.4. Recomendación para Nuevos Desarrolladores

A partir de la experiencia obtenida durante el desarrollo de la demo del videojuego 2D de aventura y plataformas, se presentan a continuación una serie de recomendaciones dirigidas a nuevos desarrolladores que deseen iniciarse en la creación de videojuegos utilizando el motor Godot Engine.

En primer lugar, se recomienda comenzar con proyectos de alcance reducido. Definir mecánicas básicas y objetivos claros permite mantener el control del desarrollo, evitar la sobrecarga de funcionalidades y facilitar la finalización del proyecto. Un prototipo sencillo resulta más efectivo que un proyecto ambicioso que no llegue a completarse.

Es importante estructurar correctamente el proyecto desde el inicio, organizando las escenas, scripts y recursos gráficos en carpetas bien definidas. La utilización del sistema de escenas y nodos de Godot favorece la modularidad, la reutilización de componentes y la escalabilidad del videojuego.

Se aconseja aprovechar las herramientas nativas del motor, como el uso de señales (signals), TileMaps y sistemas de animación, ya que estas facilitan el desarrollo y reducen la necesidad de implementar soluciones complejas desde cero. Asimismo, el uso de GDScript permite una rápida implementación de mecánicas gracias a su sintaxis sencilla y legible.

Otra recomendación clave es realizar pruebas constantes durante el desarrollo. Probar cada funcionalidad al momento de implementarla ayuda a detectar errores de forma temprana y evita problemas acumulativos en etapas avanzadas del proyecto. En el caso de los videojuegos, es fundamental evaluar tanto el funcionamiento técnico como la experiencia de juego.

Además, se sugiere documentar el proceso de desarrollo de manera continua. Mantener registros claros de las decisiones tomadas, la estructura del proyecto y el funcionamiento de los sistemas implementados facilita el mantenimiento del código y sirve como referencia para futuros proyectos.

Finalmente, se recomienda utilizar recursos gráficos y sonoros con licencias abiertas o claramente definidas, así como apoyarse en la comunidad de Godot, que ofrece documentación oficial, tutoriales y foros de discusión que resultan de gran ayuda para resolver dudas y mejorar las habilidades de desarrollo.

Capítulo 6: Conclusiones y Recomendaciones

6.1. Conclusiones

Durante el desarrollo de esta demo, la implementación de la lógica de enemigos dentro del videojuego permitió comprobar la importancia de diseñar comportamientos autónomos y coherentes con el entorno. A través de la depuración de scripts y el ajuste de direcciones de movimiento, se logró que los personajes reaccionen de manera adecuada a colisiones y límites del escenario, lo cual demuestra que el motor Godot ofrece las herramientas necesarias para construir mecánicas de juego robustas y escalables.

El diseño de tres niveles con variaciones artísticas en Tilemaps, objetos y enemigos evidenció la capacidad de estructurar un sistema dinámico de progresión. Aunque las diferencias de nivel giraban en torno a la estética y presentación del juego, la inclusión del sistema de cambio automático de escenas ilustró que el proyecto puede escalarse con gran éxito al tiempo que se mantiene una arquitectura limpia y reutilizable que conduce a un mayor crecimiento de todo el videojuego.

La integración de las diferentes pantallas del videojuego, cada una jugada en diferentes puntos de la fase de demostración, proporcionó la experiencia de juego para el jugador de una manera satisfactoria. Lo hacen no solo añadiendo contenido al juego, sino también permitiendo la manipulación y enlace de los diversos sistemas en los que se basa el motor Godot para permitir que la demo funcione como se requiere, y el resultado se presente como un producto coherente o replicable.

6.2. Recomendaciones

Como retroalimentación de este trabajo, se llega a recomendar mantener y reforzar la modularidad del código implementado en el videojuego, asegurándonos de que cada script y nodo haga algo y esté disponible para ser reutilizado con el tiempo. Además de facilitar la depuración y el mantenimiento, de esta manera el juego puede expandirse a niveles más grandes sin tener que cambiar su diseño base. Esta práctica no solo facilita la depuración y el mantenimiento, sino que también permite escalar el videojuego hacia niveles más complejos sin necesidad de modificar la estructura principal. Con ello se asegura que el proyecto conserve su estabilidad y se convierta en una base sólida para nuevas versiones o adaptaciones.

También es aconsejable ampliar la documentación del proceso de desarrollo, incluyendo explicaciones claras sobre la lógica de los personajes, el sistema de temporizador y la integración de las pantallas. Una documentación detallada no solo respalda la defensa académica del trabajo, sino que también cumple con el objetivo de que la demo sea replicable en entornos de software libre. De esta manera, otros estudiantes o desarrolladores podrán comprender con facilidad el flujo de trabajo y replicar las funcionalidades implementadas.

Y finalmente se sugiere explorar la incorporación de elementos adicionales de retroalimentación audiovisual, como efectos sonoros o animaciones complementarias en la pantalla final, los cambios de nivel y en los enemigos. Aunque el sistema actual cumple con los objetivos planteados, estas mejoras podrían incrementar la inmersión del jugador y aportar un valor agregado al producto final. Aunque este sistema cumple con los objetivos establecidos, tales cambios podrían sumergir mejor al jugador y elevar el juego final. Además, este tipo de recursos también fomentan la motivación del usuario y ayudan a que la demo se sienta más completa y atractiva.

Bibliografía

Abdullahi, A. (08 de Julio de 2022). *CIO Insight*. Obtenido de Extreme Programming vs Scrum development: https://www.cioinsight.com/application-development/extreme-programming-vs-scrum/

Baig, E. C. (09 de Noviembre de 2022). *AARP*. Obtenido de El videojuego Pong celebra 50 años de su lanzamiento: https://www.aarp.org/espanol/hogar-familia/tecnologia/info-2022/videojuego-atari-pong-nolan-bushnell.html

Facultat d'Informàtica de Barcelona. (05 de Marzo de 2008). *Upc.edu*. Obtenido de Historia de los videojuegos: https://www.fib.upc.edu/retro-informatica/historia/videojocs.html

GameCloud. (25 de Enero de 2021). *GameCloud*. Obtenido de Scrum methodology in video games testing: https://gamecloud-ltd.com/scrum-methodology-in-video-games/

Grupo ARGOsoft - Consultores en Tecnología y Marketing Digital en El Salvador. (04 de Enero de 2024). *Qué es un Motor de Videojuego*. Obtenido de Qué es un Motor de Videojuego: https://www.argosoftgroup.com/2024/01/04/impulsando-la-magia-del-entretenimiento-que-es-un-motor-de-videojuego-y-los-principales-exponentes-en-el-mercado/

IFEMA MADRID. (17 de Junio de 2025). *Ifema.es*. Obtenido de ¿Cuántos tipos de videojuegos hay?: https://www.ifema.es/noticias/videojuegos/tipos-videojuegos

Keith, C. (26 de Agosto de 2010). *Game Developer*. Obtenido de Agile game development with Scrum: Teams: https://www.gamedeveloper.com/production/agile-game-development-with-scrum-teams

Linietsky, J., Manzur, A., & Comunidad Godot. (2014). *Godot Engine documentation*. Obtenido de Godot Engine documentation: https://docs.godotengine.org/es/4.x/about/introduction.html

Matt, P. (22 de Mayo de 2015). *Time.com*. Obtenido de This Is What Pac-Man‘s Creator Thinks 35 Years Later: https://time.com/3892662/pac-mans-35-years/

McDonald, K. (14 de Junio de 2017). *Agile Alliance | Promoting a more effective, humane, and sustainable way of working*. Obtenido de Extreme Programming: https://agilealliance.org/glossary/xp/

Microsoft. (04 de Octubre de 2023). *Microsoft.com*. Obtenido de ¿Qué es Scrum?: https://learn.microsoft.com/es-es/devops/plan/what-is-scrum

Nadill, L. (03 de Junio de 2021). *Press Over*. Obtenido de Historia de los juegos de plataformas: Mundo 1-2: https://pressover.news/articulos/historia-de-los-juegos-de-plataformas-mundo-1-2/

Nadill, L. (12 de Mayo de 2021). *Press Over*. Obtenido de Historia de los juegos de plataformas: Mundo 1-1: https://pressover.news/articulos/historia-de-los-juegos-de-plataformas-mundo-1-1/

Nazanin, S. (30 de Octubre de 2024). *Pixune*. Obtenido de 2D game art pipeline: https://pixune.com/blog/a-complete-guide-to-2d-game-art-pipeline/

Pascual, A. (01 de Enero de 2023). *3Djuegos.com*. Obtenido de Por qué se le llama "Aventura" a los juegos de aventura: https://www.3djuegos.com/juegos/colossal-cave-3d-adventure/noticias/que-se-le-llama-aventura-a-juegos-aventura

Redacción APD. (09 de Abril de 2024). *APD España*. Obtenido de ¿Qué es el método Scrum y cómo aplicarlo en tu empresa?: https://www.apd.es/metodologia-scrum-que-es/

Roy, A. (29 de Junio de 2024). *Gaming.net*. Obtenido de ¿Qué es un juego de plataformas?: https://www.gaming.net/es/what-is-a-platformer-game/

Sotomayor, S. G. (09 de Noviembre de 2024). *IEBS Business School*. Obtenido de Metodologías ágiles: ¿Qué son y cuáles son más utilizadas?: https://www.iebschool.com/hub/que-son-metodologias-agiles-agile-scrum/

Tomàs, O. (20 de Noviembre de 2024). *Licencia Open Source (Código Abierto): Tipos y Ventajas*. Obtenido de Licencia Open Source (Código Abierto): Tipos y Ventajas: https://www.grupocastilla.es/licencia-open-source/

UNIR - Universidad Internacional de La Rioja. (11 de Marzo de 2023). *Los motores de videojuegos: componentes y principales plataformas*. Obtenido de Los motores de videojuegos: componentes y principales plataformas: https://www.unir.net/revista/diseno/motores-videojuegos/

Anexos

7.1. Sprint Backlog

Sprint 1 Backlog

Tarea	Descripción	Prioridad	Responsable
Implementar menú principal	Crear pantalla inicial con opciones de juego y ajustes	Alta	Equipo
Desarrollar sistema de pausa	Programar la lógica para pausar y reanudar el juego	Alta	Equipo
Realizar ajustes visuales	Mejorar la presentación gráfica de la interfaz y HUD	Media	Equipo

Sprint 2 Backlog

Tarea	Descripción	Prioridad	Responsable
Diseñar primer nivel	Crear tilemap artístico y estructura básica del nivel	Alta	Equipo
Programar movimiento del jugador	Implementar correr y saltar con detección de colisiones	Alta	Equipo
Sistema de recolección de objetos	Integrar lógica para recoger ítems y mostrarlos en HUD	Alta	Equipo
Integrar contador en HUD	Mostrar tiempo restante en pantalla	Alta	Equipo

Sprint 3 Backlog

Tarea	Descripción	Prioridad	Responsable
Diseñar segundo nivel	Crear tilemap con variaciones de objetos y enemigos	Alta	Equipo
Implementar detección de colisiones	Programar respuesta a colisiones	Alta	Equipo
Integrar animaciones básicas	Añadir animaciones de movimiento y acciones	Alta	Equipo
Configurar cámara dinámica	Hacer que la cámara siga al jugador	Alta	Equipo

Sprint 4 Backlog

Tarea	Descripción	Prioridad	Responsable
Diseñar tercer nivel	Crear tilemap diferenciado con nuevos objetos	Alta	Equipo
Implementar pantalla de finalización	Mostrar resultados al terminar el nivel	Alta	Equipo
Activar temporizador	Integrar tiempo como elemento central de jugabilidad	Alta	Equipo

Sprint 5 Backlog

Tarea	Descripción	Prioridad	Responsable
Elaborar manual de documentación	Redactar proceso de diseño e implementación	Alta	Equipo
Pruebas de exportación	Exportar demo a sistemas Windows	Alta	Equipo

7.2. Daily Scrum

Sprint 1 - Daily Scrum

Día 1:

Logrado: Creación inicial del menú principal con estructura básica.

Por hacer: Añadir opciones funcionales al menú.

Obstáculos: Ninguno.

Día 2:

Logrado: Avances en el menú principal, se añadieron botones de inicio y opciones.

Por hacer: Implementar sistema de pausa.

Obstáculos: Ajustes en la lógica de entrada para pausar correctamente.

Día 3:

Logrado: Programado el sistema de pausa básico (pausar y reanudar).

Por hacer: Mejorar la interfaz visual del HUD.

Obstáculos: Falta de recursos gráficos adecuados para los ajustes visuales.

Día 4:

Logrado: Se realizaron ajustes visuales en la interfaz y HUD.

Por hacer: Integrar mejoras gráficas en el menú principal.

Obstáculos: Limitaciones de tiempo para pulir detalles visuales.

Día 5:

Logrado: Menú principal funcional, sistema de pausa implementado y ajustes visuales básicos completados.

Por hacer: Revisión general de tareas completadas en el sprint.

Obstáculos: Ninguno.

Sprint 2 - Daily Scrum

Día 1:

Logrado: Creación del tilemap artístico del primer nivel.

Por hacer: Implementar movimiento del jugador.

Obstáculos: Ninguno.

Día 2:

Logrado: Avances en la programación del movimiento (correr y saltar).

Por hacer: Integrar sistema de recolección de objetos.

Obstáculos: Ajustes en colisiones al saltar.

Día 3:

Logrado: Movimiento del jugador funcional con detección básica de colisiones.

Por hacer: Mostrar objetos recolectados en HUD.

Obstáculos: Falta de íconos gráficos para los objetos.

Día 4:

Logrado: Sistema de recolección implementado y objetos visibles en HUD.

Por hacer: Integrar contador de tiempo en HUD.

Obstáculos: Configuración inicial del temporizador.

Día 5:

Logrado: Primer nivel funcional con movimiento, recolección y contador en HUD.

Por hacer: Revisión general del sprint.

Obstáculos: Ninguno.

Sprint 3 - Daily Scrum

Día 1:

Logrado: Creación del tilemap del segundo nivel.

Por hacer: Implementar detección de colisiones avanzada.

Obstáculos: Ninguno.

Día 2:

Logrado: Avances en la lógica de colisiones con enemigos.

Por hacer: Añadir animaciones básicas al jugador.

Obstáculos: Ajustes en sprites de animación.

Día 3:

Logrado: Animaciones básicas integradas (correr, saltar).

Por hacer: Configurar cámara dinámica.

Obstáculos: Problemas iniciales con el seguimiento de cámara.

Día 4:

Logrado: Cámara configurada para seguir al jugador.

Por hacer: Ajustar límites de cámara en el nivel.

Obstáculos: Ajustes en tilemap para evitar desplazamientos fuera de área.

Día 5:

Logrado: Segundo nivel funcional con colisiones, animaciones y cámara dinámica.

Por hacer: Revisión general del sprint.

Obstáculos: Ninguno.

Sprint 4 - Daily Scrum

Día 1:

Logrado: Creación del tilemap del tercer nivel.

Por hacer: Implementar pantalla de finalización.

Obstáculos: Ninguno.

Día 2:

Logrado: Avances en la pantalla de finalización.

Por hacer: Integrar temporizador activo.

Obstáculos: Configuración inicial del temporizador.

Día 3:

Logrado: Temporizador activo y funcional en HUD.

Por hacer: Vincular temporizador con puntuación final.

Obstáculos: Ajustes en cálculo de puntuación.

Día 4:

Logrado: Pantalla de finalización muestra resultados con puntuación.

Por hacer: Mejorar animación de puntuación.

Obstáculos: Limitaciones gráficas en animación progresiva.

Día 5:

Logrado: Tercer nivel completo con temporizador y pantalla de finalización.

Por hacer: Revisión general del sprint.

Obstáculos: Ninguno.

Sprint 5 - Daily Scrum

Día 1:

Logrado: Inicio de la redacción del manual de documentación.

Por hacer: Detallar procesos de diseño e implementación.

Obstáculos: Ninguno.

Día 2:

Logrado: Avances en la documentación del sistema de personajes y niveles.

Por hacer: Documentar pantallas y HUD.

Obstáculos: Tiempo limitado para redactar con detalle.

Día 3:

Logrado: Documentación del HUD y pantallas completadas.

Por hacer: Pruebas de exportación a Windows.

Obstáculos: Configuración inicial de exportación.

Día 4:

Logrado: Pruebas de exportación realizadas con éxito.

Por hacer: Ajustar detalles finales del manual.

Obstáculos: Ninguno.

Día 5:

Logrado: Manual finalizado y exportación funcional en Windows.

Por hacer: Revisión general del sprint y cierre del proyecto.

Obstáculos: Ninguno.

7.3. Retrospectiva de Sprints

Revisión del Sprint 1

Resultados:

- Se implementó el menú principal con opciones básicas de inicio y ajustes.
- Se desarrolló el sistema de pausa funcional.
- Se realizaron ajustes visuales iniciales en la interfaz y HUD.

Inconcluso:

- Algunos detalles gráficos del menú quedaron sin pulir y se planificaron para futuros sprints.

Dificultades:

- La falta de recursos gráficos adecuados retrasó parte de los ajustes visuales.

Retrospectiva del Sprint 1

Aspectos Positivos:

- Se logró establecer la base visual y funcional del videojuego.
- El sistema de pausa se integró sin problemas técnicos.

Áreas de Mejora:

- Anticipar la necesidad de recursos gráficos para evitar retrasos en tareas dependientes.
- Acciones para el próximo sprint:
- Planificar con antelación los elementos gráficos necesarios para cada iteración.

Acciones para el Próximo Sprint

- Obtener los Tilemaps necesarios para el desarrollo de los niveles.

Revisión del Sprint 2

Resultados:

- Se diseñó el primer nivel utilizando TileMaps.
- Se implementó el movimiento del jugador con correr y saltar.

- Se integró el sistema de recolección de objetos.
- Se añadió el contador de tiempo en el HUD.

Inconcluso:

- Ajustes menores en la interfaz del HUD se dejaron para revisión posterior.

Dificultades:

- La falta de íconos gráficos adecuados para los objetos retrasó la integración visual.

Retrospectiva del Sprint 2

Aspectos Positivos:

- Todas las funcionalidades principales del nivel se completaron dentro del tiempo estimado.
- El sistema de recolección y el temporizador funcionaron correctamente en conjunto.

Áreas de Mejora:

- Definir criterios de calidad más claros para evaluar la jugabilidad desde etapas tempranas.

Acciones para el Próximo Sprint:

- Revisar la experiencia del jugador y ajustar detalles de jugabilidad antes de avanzar al siguiente nivel.

Revisión del Sprint 3

Resultados:

- Se diseñó el segundo nivel con variaciones en objetos y enemigos.
- Se implementó la detección de colisiones de manera más robusta.
- Se añadieron animaciones básicas al jugador.
- Se configuró la cámara dinámica que sigue al personaje.

Inconcluso:

- Algunos ajustes en la cámara y límites del nivel quedaron para revisión posterior.

Dificultades:

- La integración de animaciones requirió más tiempo por la necesidad de ajustar sprites.

Retrospectiva del Sprint 3

Aspectos Positivos:

- Se logró un avance significativo en la jugabilidad con animaciones y cámara dinámica.
- La detección de colisiones mejoró la interacción del jugador con el entorno.

Áreas de Mejora:

- Planificar con mayor detalle la integración de recursos gráficos para evitar retrasos.

Acciones para el Próximo Sprint:

- Consolidar las mecánicas ya implementadas y preparar la lógica de finalización de nivel.

Revisión del Sprint 4

Resultados:

- Se diseñó el tercer nivel con tilemap diferenciado.
- Se implementó la pantalla de finalización del nivel.
- Se activó el temporizador como elemento central de jugabilidad.
- Se vinculó el temporizador con la puntuación final.

Inconcluso:

- La animación progresiva de la puntuación quedó con detalles por mejorar.

Dificultades:

- Ajustes en la lógica del temporizador y su relación con la puntuación tomaron más tiempo de lo previsto.

Retrospectiva del Sprint 4

Aspectos Positivos:

- Se logró integrar la pantalla de finalización y el sistema de puntuación.

- El temporizador funcionó correctamente como mecánica central del nivel.

Áreas de Mejora:

- Optimizar la lógica de cálculo y animación de la puntuación para mejorar la experiencia del jugador.

Acciones para el Próximo Sprint:

- Documentar las funcionalidades implementadas y realizar pruebas de exportación.

Revisión del Sprint 5

Resultados:

- Se elaboró el manual de documentación del proceso de desarrollo.
- Se realizaron pruebas de exportación del videojuego en sistemas Windows.
- Se validó la funcionalidad completa de la demo.

Inconcluso:

- Ajustes menores en la presentación del manual se dejaron para revisión final.

Dificultades:

- La exportación inicial requirió ajustes en la configuración del proyecto.

Retrospectiva del Sprint 5

Aspectos Positivos:

- Se completó la documentación y se logró exportar la demo de manera funcional.
- El proyecto cumplió con los objetivos planteados en la planificación inicial.

Áreas de Mejora:

- Mejorar la organización del manual para facilitar aún más la replicación del proyecto.

7.4. Algoritmos de la Demo

Area Limite

```
extends Area2D

@onready var gameOver = get_node("../CanvasLayer/GameOver")

func _on_body_entered(body: Node2D) -> void:
	if body.get_name()=="Jugador":
		body.queue_free()
		gameOver.mostrar_game_over()
```

Enemigos

```
extends CharacterBody2D

var gravedad = 10

var rapidez = 35

var moviendo_izquierda = true

func _ready():
	$DamaCaminar.play("dama_caminar")

func _process(_delta):
	move_character()
	turn()

func move_character():
	velocity.y += gravedad
	if (moviendo_izquierda):
		velocity.x = rapidez
		move_and_slide()
	else:
```

```
            velocity.x = -rapidez
            move_and_slide()
func _on_area_2d_body_entered(body):
    if body.get_name() == "Jugador":
            body.recibeDaño()
            pass
func turn():
    if not $DetectarPiso.is_colliding():
            moviendo_izquierda = !moviendo_izquierda
            scale.x = -scale.x
     # Si hay un muro delante
    elif $DetectarPared.is_colliding():
            moviendo_izquierda = !moviendo_izquierda
            scale.x = -scale.x
```

Final Nivel

```
extends StaticBody2D
@export var contador_requerido := 3
var puede_pasar := false
func _ready():
    # Al inicio la puerta debe bloquear al jugador
    $Muro.disabled = false   # Muro activado
    $AreaPuerta.monitoring = false  # Sensor desactivado
func actualizar_estado(contador_actual):
    if contador_actual >= contador_requerido:
```

```
            puede_pasar = true
            # Desactivar muro para permitir pasar (deferred)
            $Muro.call_deferred("set_disabled", true)
            # Activar sensor del Area2D para detectar al jugador (deferred)
            $AreaPuerta.set_deferred("monitoring", true)
        else:
            puede_pasar = false
            # Mantener la puerta cerrada
            $AreaPuerta.set_deferred("monitoring", false)
func _on_area_puerta_body_entered(body):
    if puede_pasar and body.name == "Jugador":
        var panelNivelComp = get_node("../CanvasLayer/HUDNivelCompletado")
        var canvasLayer = get_node("../CanvasLayer")
        var tiempo = int(canvasLayer.tiempo_restante)
        panelNivelComp.mostrar_pantalla_final(tiempo)
```

HUD

```
extends CanvasLayer
var nivel
var Objeto1 = "???"
var Objeto2 = "???"
var Objeto3 = "???"
var req
var puerta
var tiempo_restante = 300  # segundos
```

```
@onready var tiempo_label = $Temporizador
@onready var timer = $Timer
@onready var gameOver = $GameOver
@onready var personaje = get_node("../Jugador")
func _ready():
	get_tree().paused = false
	req = 0
	tiempo_label.text = str(tiempo_restante)
	timer.start(1.0)  # dispara cada segundo
	# Detectar el nombre de la escena actual
	nivel = get_tree().current_scene.name
	 # Actualizar los labels
	$NameObject1.text = Objeto1
	$NameObject2.text = Objeto2
	$NameObject3.text = Objeto3
	puerta = get_tree().get_current_scene().find_child("FinalNivel",true,false)
func handleObjeto1Collected():
	match nivel:
		"Nivel1":
			Objeto1 = "Ladrillo Perdido de Cantuña"
		"Nivel2":
			Objeto1 = "Llaves de la Casa 1028"
		"Nivel3":
			Objeto1 = "Veleta del Gallo de la Catedral"
```

```
        $NameObject1.text = String(Objeto1)
        req+=1
        endLevel()
func handleObjeto2Collected():
        match nivel:
                "Nivel1":
                        Objeto2 = "La Capa del Estudiante"
                "Nivel2":
                        Objeto2 = "Collar de Posorja"
                "Nivel3":
                        Objeto2 = "El cofre del Tesoro del Pirata Lewis"
        $NameObject2.text = String(Objeto2)
        req+=1
        endLevel()
func handleObjeto3Collected():
        match nivel:
                "Nivel1":
                        Objeto3 = "La Cruz del Padre Almeida"
                "Nivel2":
                        Objeto3 = "Craneo de San Francisco"
                "Nivel3":
                        Objeto3 = "Cristo tallado por Caspicara"
        $NameObject3.text = String(Objeto3)
        req+=1
```

```
        endLevel()
func endLevel():
        if puerta != null:
                puerta.actualizar_estado(req)
func _on_timer_timeout() -> void:
        tiempo_restante -= 1
        tiempo_label.text = str(tiempo_restante)
        if tiempo_restante <= 0:
                personaje.queue_free()
                gameOver.mostrar_game_over()
```

HUD Final Demo

```
extends Control
func _ready():
        get_tree().paused = false
func _on_regresar_pressed():
        get_tree().change_scene_to_file("res://Escenas/menu_de_inicio.tscn")
```

HUD Nivel Completado

```
extends Control
var puntuacion_final = 1500  # valor final
var puntuacion_actual = 0
var incremento = 50
@onready var label_puntuacion = $Puntuacion
@onready var timer_animacion = $TimerAnimacion
func mostrar_pantalla_final(tiempo_restante: int):
```

```
        visible = true
        #Cada segundo vale 100 puntos
        puntuacion_final = tiempo_restante * 10
        puntuacion_actual = 0
        label_puntuacion.text = "Puntuación: 0"
        var pasos = 10  # número de actualizaciones
        incremento = int(max(1, puntuacion_final / float(pasos)))
        timer_animacion.start(0.1)
        get_tree().paused = true
func _on_TimerAnimacion_timeout():
        if puntuacion_actual < puntuacion_final:
                puntuacion_actual += incremento
                if puntuacion_actual > puntuacion_final:
                        puntuacion_actual = puntuacion_final
                label_puntuacion.text = "Puntuación: " + str(puntuacion_actual)
        else:
                timer_animacion.stop()

func _on_continuar_pressed():
        var nombre_nivel = str(get_tree().current_scene.name)  # "NivelX"
        var numero = int(nombre_nivel.substr(5, nombre_nivel.length() - 5)) + 1
        var ruta= ""
        if numero == 4:
                ruta = "res://Escenas/hud_final_demo.tscn"   # aquí pones la ruta que quieras
```

```
        else:
                ruta = "res://Escenas/n_ivel_%d.tscn" % numero
        get_tree().change_scene_to_file(ruta)
func _on_salir_al_menu_pressed():
        get_tree().change_scene_to_file("res://Escenas/menu_de_inicio.tscn")
```

Jugador

```
extends CharacterBody2D


@onready var gameOver = get_node("../CanvasLayer/GameOver")
const SPEED = 100
const JUMP_FORCE = -300
const GRAVITY = 600
func _physics_process(delta):
        # Aplicar gravedad
        velocity.y += GRAVITY * delta
        # Movimiento horizontal
        var direction = 0
        if Input.is_action_pressed("ui_right"):
                direction = 1
        elif Input.is_action_pressed("ui_left"):
                direction = -1
        velocity.x = direction * SPEED
        # Saltar
        if is_on_floor() and Input.is_action_just_pressed("ui_accept"):
```

```
            velocity.y = JUMP_FORCE
        # --- ANIMACIONES ---
        var anim = $AnimatedSprite2D
        # Flip horizontal según dirección
        if direction != 0:
            anim.flip_h = direction < 0
        # Seleccionar animación
        if not is_on_floor():
            anim.play("jump")
        elif direction == 0:
            anim.play("idle")
        else:
            anim.play("walk")
        move_and_slide()
func add_Objeto1():
        var canvasLayer = get_tree().get_root().find_child("CanvasLayer",true,false)
        canvasLayer.handleObjeto1Collected()
func add_Objeto2():
        var canvasLayer = get_tree().get_root().find_child("CanvasLayer",true,false)
        canvasLayer.handleObjeto2Collected()
func add_Objeto3():
        var canvasLayer = get_tree().get_root().find_child("CanvasLayer",true,false)
        canvasLayer.handleObjeto3Collected()
func recibeDaño():
```

```
        gameOver.mostrar_game_over()
```

Menu de Inicio

```
extends Control

func _on_jugar_pressed() -> void:
        get_tree().change_scene_to_file("res://Escenas/n_ivel_1.tscn")

func _on_opciones_pressed() -> void:
        get_tree().change_scene_to_file("res://Escenas/menu_opciones.tscn")

func _on_salir_pressed() -> void:
        _reset_volume_settings()
        get_tree().quit()

func _reset_volume_settings() -> void:
        var config_path := "user://settings.cfg"
        if FileAccess.file_exists(config_path):
                DirAccess.remove_absolute(config_path)
```

Menu Game Over

```
extends Control

func _ready():
        visible = false

func mostrar_game_over():
        visible = true
        get_tree().paused = true

func _on_btn_reiniciar_pressed() -> void:
        get_tree().paused = false
        get_tree().reload_current_scene()
```

```
func _on_btn_quitar_pressed() -> void:
	get_tree().paused = false
	get_tree().change_scene_to_file("res://Escenas/menu_de_inicio.tscn")
```

Menu Opciones

```
extends Control
@onready var volume_slider: Slider = $ColorRect/VBoxContainer/Barra_Volumen
@onready var resolution_button: OptionButton = $ColorRect/VBoxContainer/Resolucion
func _ready() -> void:
	volume_slider.min_value = 0.0
	volume_slider.max_value = 1.0
	volume_slider.step = 0.01
	resolution_button.clear()
	for r in Settings.resolutions:
		resolution_button.add_item("%dx%d" % [r.x, r.y])
	resolution_button.add_item("Pantalla completa")
	resolution_button.item_selected.connect(_on_resolution_selected)
	volume_slider.value = Settings.volume
	_apply_volume(Settings.volume)
	resolution_button.select(Settings.resolution_index)
	_apply_resolution(Settings.resolution_index)
func _on_barra_volumen_value_changed(value: float) -> void:
	Settings.volume = clamp(value, 0.0001, 1.0)
	_apply_volume(Settings.volume)
func _apply_volume(value: float) -> void:
```

```
	var bus_idx: int = AudioServer.get_bus_index("Master")
	if bus_idx >= 0:
		AudioServer.set_bus_volume_db(bus_idx, linear_to_db(value))
func _on_resolution_selected(index: int) -> void:
	Settings.resolution_index = index
	_apply_resolution(index)
func _apply_resolution(index: int) -> void:
	if index == Settings.resolutions.size():
	DisplayServer.window_set_mode(DisplayServer.WINDOW_MODE_FULLSCREEN)
	else:
	DisplayServer.window_set_mode(DisplayServer.WINDOW_MODE_WINDOWED)
		DisplayServer.window_set_size(Settings.resolutions[index])
func _on_atras_pressed() -> void:
	get_tree().change_scene_to_file("res://Escenas/menu_de_inicio.tscn")
```

Menu Pausa

```
extends Control
func _input(event):
	if event.is_action_pressed("pause"):
		visible = not get_tree().paused
		get_tree().paused = not get_tree().paused
	else: if event.is_action_pressed("restart"):
		get_tree().paused = false
		get_tree().reload_current_scene()
```

```
        else: if event.is_action_pressed("quit"):
                get_tree().paused = false  # Asegura que el juego no quede pausado
                get_tree().change_scene_to_file("res://Escenas/menu_de_inicio.tscn")
# Regresa al menú principal
```

Objetos

```
extends Area2D
func _on_body_entered(body):
        if body.get_name()=="Jugador":
                body.add_Objeto1()
                queue_free()
```