Principado de Asturias Consejería de Educación	IES N 1 Gijón	**PROYECTO** **DAM**	Cofinanciado por la Unión Europea

INSTITUTO DE EDUCACIÓN SECUNDARIA NÚMERO 1 DE GIJÓN

FAMILIA PROFESIONAL DE INFORMÁTICA Y COMUNICACIONES

Ciclo Formativo Desarrollo de Aplicaciones Multiplataforma

2º Curso

Proyecto de Desarrollo de Aplicaciones Multiplataforma

Modalidad Presencial

Tipo: Desarrollo de aplicación. Modalidad Videojuegos

RUN RUBY!

Videojuego 2D plataformas desarrollado en GODOT para PC

Documento-Memoria

Autor/a: Gabriel Álvarez de Pablo

Fecha: 1 de junio de 2024

ÍNDICE

1. Introducción 2
1.1 Presentación del alumno 2
1.2 Título del proyecto y tipo de proyecto elegido 2
2. Definición del proyecto 2
2.1 Descripción general 2
2.2 Definición de requisitos 3
3. Planificación del proyecto 7
3.1 Planificación de actividades y tareas 7
3.2 Estimación de costes 9
3.3 Previsión de riesgos del proyecto 9
3.3 Estimación de costes ..... **Error! Bookmark not defined.**
4. Análisis y Diseño del proyecto 10
4.1 Diseño y arquitectura 10
4.2 Modelo de datos 16
4.3 Modelo de procesos 17
5. Construcción del proyecto 19
5.1 Codificación 19
5.2 Pruebas 22
5.3 Manual de instalación 23
5.4 Manual de usuario 23
6. Evaluación final del proyecto 29
6.1 Evaluación del diseño, del proceso y del resultado 29
6.2 Conclusiones y lecciones aprendidas 30
6.3 Posibles ampliaciones futuras 30
7. Bibliografía/webgrafía 31

1. Introducción

1.1 Presentación del alumno

Este proyecto de desarrollo de aplicaciones multiplataforma ha sido realizado por Gabriel Álvarez de Pablo, estudiante del Ciclo Formativo de Grado Superior de Desarrollo de Aplicaciones Multiplataforma en el Instituto de Educación Secundaria Número 1 de Gijón. Con la tutorización colectiva de Pilar García, y la individual de Lourdes Cabeza.

1.2 Título del proyecto y tipo de proyecto elegido

El proyecto "RUN RUBY!" es un videojuego de plataformas competitivo que se desarrolla utilizando el motor de juegos Godot y el lenguaje de programación GDScript. El objetivo del juego es competir contra jugadores globales, utilizando recursos del mapa y mecánicas de movimiento para ser el más rápido en llegar a la meta.

2. Definición del proyecto

2.1 Descripción general

El cliente ha solicitado el desarrollo de un juego que incluye la creación de sprites y sonidos para una experiencia visual y auditiva atractiva. El juego debe tener mecánicas de movimiento básico, como moverse horizontalmente, saltar y atacar, así como características de calidad de vida que faciliten la jugabilidad, como búfer de salto y coyote time (explicadas más adelante). Ya que el juego se lanzará a nivel global, este deberá estar en inglés. Además, el juego debe incluir habilidades especiales como doble salto, dash (breve embestida) y caminar por el aire.

El nivel de introducción debe ser complejo, es decir, debe proponer un pequeño reto para el jugador, de esta manera aprenderá para que son las habilidades, ya que sin ellas no podría superar el obstáculo o lo haría de forma más lenta. El sistema de identificación del jugador debe ser sencillo y solo requerir el nombre del jugador, y la información del jugador se guardará en un JSON encriptado.

El juego también debe incluir varios menús, como el de login, principal, opciones y ranking, así como menús durante el juego para pausar o reiniciar el juego. Es fundamental realizar pruebas exhaustivas de las mecánicas de juego para garantizar su funcionamiento correcto y equilibrado, y evaluar el rendimiento del juego para optimizar su fluidez y evitar problemas de carga o lag.

Finalmente, es importante mantener el código documentado en todo momento para facilitar su mantenimiento y futuras actualizaciones, y crear un manual de usuario que explique las mecánicas, controles y características del juego para orientar a los jugadores y maximizar su disfrute.

2.2 Definición de requisitos

Requisitos funcionales

Creación de Assets

El cliente ha especificado la necesidad de diseñar sprites para el personaje, nivel, cristales y cartas de habilidades, así como la creación de sonidos como música de fondo y efectos de sonido para una experiencia más inmersiva. Esto permitirá a los jugadores interactuar con un entorno visualmente atractivo y auditivamente rico.

- **Personaje**: El personaje debe ser un samurái llamado Ruby. Este debe tener una espada. Además, debe tener animaciones básicas, como caminar, saltar, caer y atacar.
- **Nivel**: Los colores de los segmentos del nivel deben ir acorde con los cristales que los contienen, blanco en el caso de varios, combinando así los colores.
- **Cristales**: Los cristales deben tener una animación de rotación. El color de estos debe ir acorde con las habilidades que proporcionan.
- **Cartas de habilidades**: Las cartas deben contener un icono de su elemento y el color que le corresponde.
- **Música de fondo**: La música de fondo debe ser animada, que se reproduzca en bucle, y no tenga cambios drásticos. En el caso de la música del nivel, debe de causar tensión.
- **Efectos de sonido**: Los efectos de sonido deben ser adecuados, si se usa una carta de fuego, reproducir un sonido de fuego.

Mecánicas del juego

Movimiento Básico

- **Movimientos Horizontales**: El jugador debe poder moverse horizontalmente a través del nivel.
- **Saltos**: El jugador debe poder saltar sobre obstáculos y plataformas.
- **Ataques**: El jugador debe poder realizar ataques para romper los cristales.
- **Uso de Habilidades**: El jugador debe poder activar habilidades especiales para modificar su movimiento.

Calidad de Vida (QoL)

- **Deslizamiento en el Techo**: El jugador debe poder deslizarse en el techo para evitar movimiento estricto en el aire.
- **Coyote Time**: El jugador debe tener un breve margen extra invisible para que pueda saltar sin caerse, aunque visualmente el personaje estaría fuera de la plataforma, con el fin de facilitar la jugabilidad.
- **Búfer de Salto**: El jugador debe tener un búfer de salto, que le permita saltar al tocar el suelo, pese a que la tecla haya sido presionada antes de tocar el suelo, facilitando el movimiento vertical.
- **Doble ataque**: El jugador debe tener un breve período de tiempo después de realizar un ataque en el que pueda realizar otro ataque, haciendo así uno doble.

Habilidades

- **Doble Salto**: El jugador debe poder realizar un segundo salto en el aire, permitiendo una mayor flexibilidad en el movimiento vertical.
- **Dash**: El jugador debe poder realizar un rápido movimiento en línea recta para superar obstáculos.
- **Caminar por el Aire**: El jugador debe poder caminar en el aire durante un breve período de tiempo, pero solo horizontalmente.
- **Caída Rápida**: El jugador debe poder realizar una caída rápida, esta deberá canalizarse, es decir, deberá detenerse brevemente antes de realizar la caída rápida.

Estas mecánicas proporcionan profundidad al gameplay y permiten al jugador explorar y superar desafíos de manera creativa.

Diseño de Niveles

El nivel de introducción debe ser complejo y presentar las mecánicas básicas de forma gradual y desafiante. La progresión de dificultad, la distribución de obstáculos y la creatividad en la disposición de plataformas son aspectos clave a considerar. Esto permitirá a los jugadores aprender y mejorar sus habilidades de manera efectiva.

Datos de Juego

El sistema de identificación del jugador debe ser sencillo y solo requerir el nombre del jugador. Una vez realizada la identificación, este nombre será almacenado en un JSON. Para no permitir al jugador sobrescribir este archivo, se guardará la información del jugador de forma encriptada. Además, se debe implementar un sistema de ranking global para fomentar la competencia entre jugadores y permitir la comparación de puntajes.

Menús

El juego debe incluir los siguientes menús:

Menú de Login

- **Nombre**: El jugador debe poder ingresar su nombre.
- **Aceptar**: El jugador podrá confirmar su identificación.
- **Atrás**: El jugador puede regresar al menú principal.

Menú Principal

- **Login**: El jugador podrá identificarse.
- **Empezar**: El jugador puede comenzar el juego.
- **Ranking**: El jugador puede ver la tabla de clasificación.
- **Opciones**: El jugador puede acceder a las opciones del juego.
- **Salir**: El jugador puede salir del juego.

Menú de Opciones

- **Controles**: El jugador podrá ver la lista de controles del juego.
- **Volumen General**: El jugador puede ajustar el volumen general del juego.
- **Volumen de Música del Nivel**: El jugador puede ajustar el volumen de la música del nivel.
- **Volumen de Efectos de Sonido**: El jugador puede ajustar el volumen de los efectos de sonido.
- **Atrás**: El jugador puede regresar al menú principal.
- **Salir**: El jugador puede salir del juego.

Menú de Ranking

- **Tabla de Clasificación**: El jugador puede ver la tabla de clasificación global.
- **Cerrar tabla**: El jugador puede regresar al menú principal.

Menús Durante el Juego

- **Pausa**: El jugador puede pausar el juego.
    - **Continuar**: El jugador puede continuar el juego.
    - **Menú Principal**: El jugador puede regresar al menú principal.
    - **Salir**: El jugador puede salir del juego.
- **Victoria**: El jugador ha ganado el nivel.
    - **Reiniciar**: El jugador puede reiniciar el nivel.
    - **Ranking**: El jugador puede ver la tabla de clasificación.
    - **Menú Principal**: El jugador puede regresar al menú principal.

Pruebas

Es fundamental realizar pruebas exhaustivas de las mecánicas de juego para garantizar su funcionamiento correcto y equilibrado. Además, se debe evaluar el rendimiento del juego para optimizar su fluidez y evitar problemas de carga o lag que puedan afectar la experiencia del jugador.

Documentación

Es importante mantener el código documentado en todo momento para facilitar su mantenimiento y futuras actualizaciones. Además, se debe crear un manual de usuario que explique las mecánicas, controles y características del juego para orientar a los jugadores y maximizar su disfrute.

Requisitos no funcionales

Requisitos de software

- Utilizar la última versión del motor de juegos para el desarrollo del juego.
- Emplear la herramienta Aseprite para el diseño y animación de los personajes y escenarios del juego en estilo pixel art.
- Utilizar Audacity para la edición y creación de los efectos de sonido y la banda sonora del juego.
- Integrar el sistema de control de versiones Git y el repositorio GitHub para la gestión del código fuente y la colaboración del equipo de desarrollo.

Requisitos de hardware

- Contar con estaciones de trabajo con procesadores Intel Core i5 o AMD Ryzen 5 como mínimo para un desarrollo eficiente.
- Disponer de al menos 8GB de memoria RAM en cada estación de trabajo para manejar los recursos gráficos y de audio del juego.

- Utilizar tarjetas gráficas dedicadas NVIDIA GeForce GTX 1060 o AMD Radeon RX 580 para garantizar un rendimiento adecuado durante la ejecución del juego.
- Contar con discos SSD de al menos 500GB para agilizar los tiempos de carga de los niveles y recursos del juego.

Estos requisitos no funcionales adicionales asegurarán que el equipo de desarrollo cuente con las herramientas y el hardware necesarios para crear un juego 2D de alta calidad como "RUN RUBY!", cumpliendo con los estándares de la industria y facilitando un flujo de trabajo eficiente.

3. Planificación del proyecto

3.1 Planificación de actividades y tareas

La planificación de actividades y tareas se basa en un enfoque estructurado para el desarrollo del juego. A continuación se detallan las actividades y tareas clave que se deben abordar:

Investigación y planificación

- **Explicación**: Esta fase implica investigar las mejores prácticas en el desarrollo de juegos y planificar la estrategia general del proyecto.

Creación de assets

- **Sprites**: Diseñar los sprites necesarios para los personajes, niveles y elementos del juego.
- **Sonido**: Crear efectos de sonido y música de fondo para mejorar la experiencia auditiva del juego.

Mecánicas de movimientos

- **Básicas**: Implementar movimientos básicos como caminar, saltar y atacar.
- **Calidad de Vida**: Incluir características que mejoren la jugabilidad, como deslizamiento en el techo y búfer de salto.
- **Habilidades**: Desarrollar habilidades especiales como doble salto, dash y otras capacidades únicas.
- **Movimiento Complejo**: Implementar movimientos avanzados que añadan profundidad al gameplay.

Diseño de niveles

- **Nivel de Introducción Complejo**: Crear un nivel inicial desafiante que introduzca gradualmente las mecánicas básicas del juego.

Datos de juego

- **Identificación**: Establecer un sistema de identificación de jugadores simple y seguro.
- **Ranking Global**: Implementar un sistema de ranking global para fomentar la competencia entre jugadores.

Menús (principal, opciones, ranking, pausa, victoria)

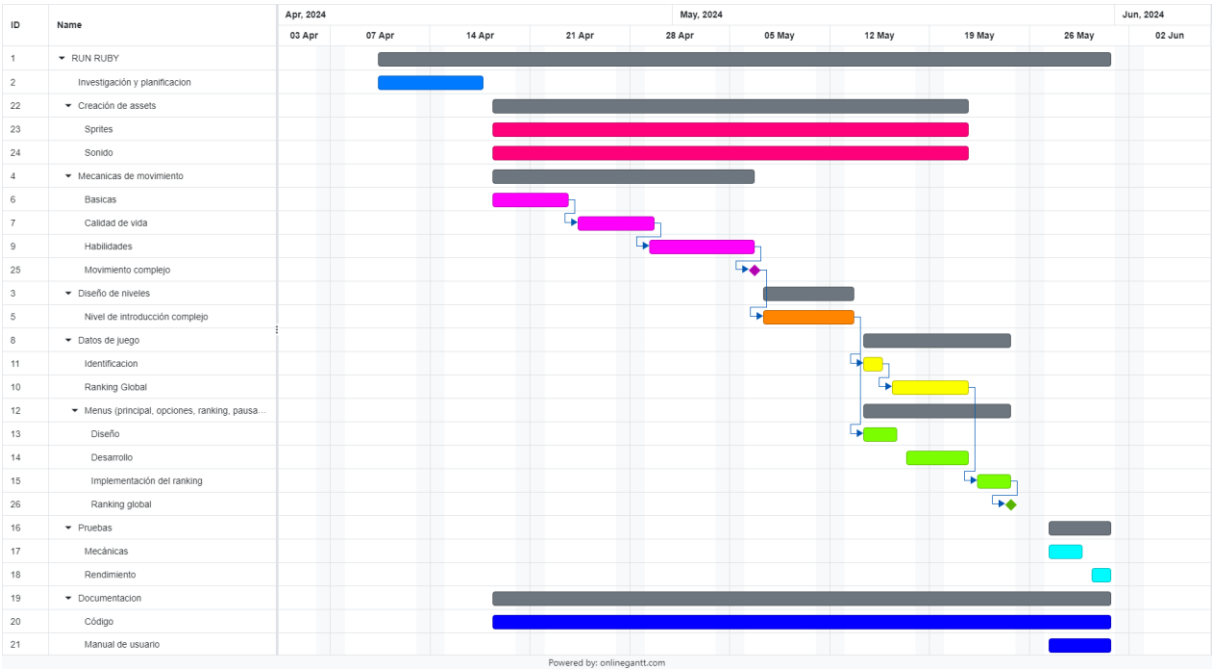
- **Diseño**: Crear la interfaz de usuario para los diferentes menús del juego.
- **Desarrollo**: Programar la funcionalidad de los menús, incluyendo opciones, ranking y pausa.
- **Implementación del Ranking**: Integrar el sistema de ranking en los menús del juego.
- **Ranking Global**: Permitir a los jugadores acceder y comparar sus puntajes en un ranking global.

Pruebas

- **Mecánicas**: Realizar pruebas exhaustivas para garantizar el correcto funcionamiento de las mecánicas del juego.
- **Rendimiento**: Evaluar el rendimiento del juego para optimizar la fluidez y evitar problemas técnicos.

Documentación

- **Código**: Mantener el código documentado para facilitar su mantenimiento y futuras actualizaciones.
- **Manual de Usuario**: Crear un manual detallado que explique las mecánicas, controles y características del juego para orientar a los jugadores y maximizar su disfrute.



3.2 Estimación de costes

Costos de desarrollo de assets

- Contratar artistas y diseñadores para crear los assets del juego (personajes, entornos, efectos, etc.)
- Estimar un costo de $20,000 para el desarrollo de assets

Gastos en licencias de software

- Utilizar motores de juego y herramientas de desarrollo con licencias comerciales
- Estimar un costo de $5,000 para licencias de software

Tiempo de desarrollo y recursos humanos

- Contratar un equipo de desarrollo compuesto por 1 director de proyecto, 2 programadores, 1 diseñador de niveles y 1 artista
- Estimar un costo de $50,000 para el equipo de desarrollo durante 2 meses de trabajo

3.3 Previsión de riesgos del proyecto

Riesgo de Pérdida de Datos

- **Riesgo**: La pérdida de datos del juego puede causar la destrucción de la base de datos y la información del jugador.
- **Mitigación**: Implementar un sistema de backup regular de los datos y utilizar encriptación para proteger la información del jugador.

Riesgo de Pérdida de la Codificación

- **Riesgo**: La pérdida de la codificación puede causar la imposibilidad de mantener y actualizar el juego.
- **Mitigación**: Documentar el código en todo momento y mantener una copia de seguridad de los archivos de código.

Problemas de Compatibilidad con Diferentes Plataformas

- **Riesgo**: El juego no se adapta correctamente a diferentes plataformas y dispositivos, lo que puede afectar su rendimiento y jugabilidad.
- **Mitigación**: Realizar pruebas en múltiples plataformas y dispositivos durante el desarrollo y asegurarse que el juego cumpla con los requisitos de cada plataforma.

Feedback Negativo de los Usuarios

- **Riesgo**: El juego recibe retroalimentación negativa de los usuarios, lo que puede afectar su reputación y atractivo.
- **Mitigación**: Realizar pruebas con usuarios beta para obtener retroalimentación temprana y estar preparado para implementar actualizaciones y mejoras basadas en el feedback

Esta planificación detallada, junto con la estimación de costos y la previsión de riesgos, proporcionará una base sólida para el desarrollo exitoso del proyecto de juego.

4. Análisis y Diseño del proyecto

4.1 Diseño y arquitectura

He determinado que el desarrollo de "RUN RUBY!" se llevará a cabo utilizando el motor de juegos Godot y el lenguaje de programación GDScript. Esta elección tecnológica permite crear un juego 2D optimizado, con un rendimiento excepcional y una curva de aprendizaje suave para el equipo de desarrollo.

Las principales ventajas de utilizar Godot y GDScript son:

- Rendimiento optimizado para juegos 2D, garantizando una experiencia de juego fluida.
- Sintaxis similar a Python en GDScript, facilitando el desarrollo para programadores familiarizados con lenguajes de alto nivel.
- Herramientas integradas en Godot, como el editor visual de escenas y el sistema de nodos, que aceleran el proceso de desarrollo y fomentan la colaboración.
- Una comunidad activa y en constante crecimiento, con una amplia disponibilidad de recursos, tutoriales y soporte.

Con estos requisitos definidos y la elección de la tecnología adecuada, se procederá a planificar y desarrollar "RUN RUBY!" para ofrecer una experiencia de juego 2D de alta calidad, competitiva y atractiva para el público objetivo.

En el desarrollo del juego "RUN RUBY!", se ha realizado un análisis detallado del diseño y la arquitectura del proyecto, centrándose en la implementación de un nivel único que evoluciona de ser guiado a caótico, es decir, para sobrepasar los primeros obstáculos, debe se deberá usar las habilidades, a partir de eso se utilizarán para hacer mejores tiempos, ofreciendo múltiples rutas para llegar al final, donde el jugador debe elegir la más rápida. Para lograr esto, se han utilizado las siguientes tecnologías y elementos:

GITHUB

En el desarrollo del juego "RUN RUBY!", se ha decidido utilizar GitHub como sistema de control de versiones para gestionar el código y los recursos del proyecto de manera eficiente. GitHub ofrece una serie de ventajas significativas que contribuyen al éxito del desarrollo del juego.

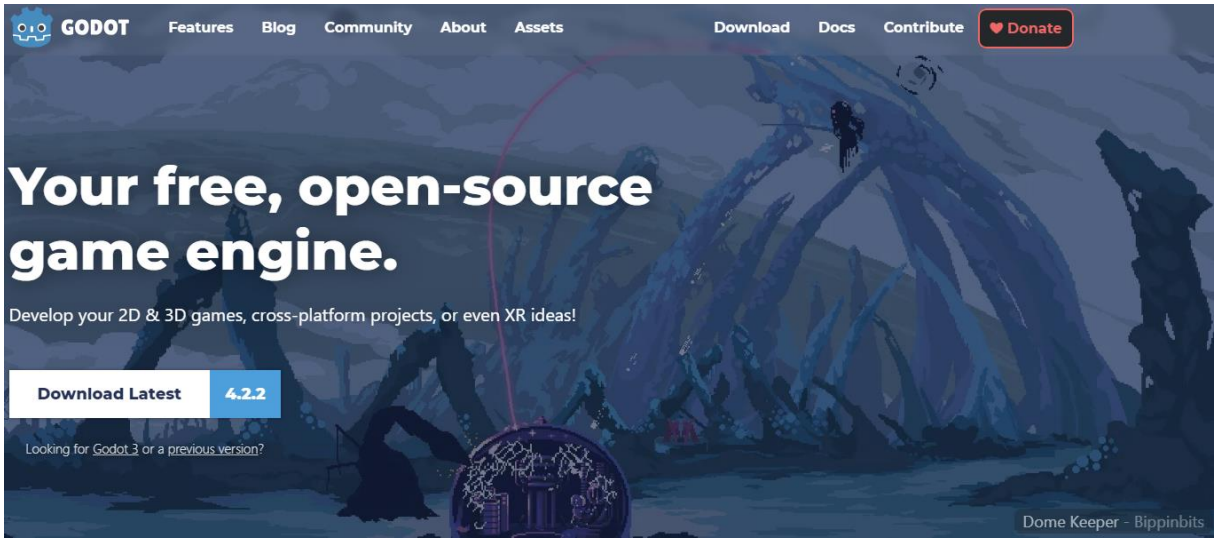
Estas son las ventajas de GitHub:

1. **Control de Versiones**: GitHub permite mantener un historial detallado de todos los cambios realizados en el código y los recursos del juego. Esto facilita la colaboración entre el equipo de desarrollo y proporciona una visión clara de la evolución del proyecto.
2. **Colaboración**: GitHub facilita la colaboración entre los miembros del equipo al permitirles trabajar en ramas separadas y fusionar los cambios de manera controlada. Esto mejora la eficiencia y la organización del trabajo en equipo.
3. **Seguridad**: GitHub ofrece medidas de seguridad avanzadas para proteger el código y los recursos del juego. Se pueden establecer permisos y restricciones para garantizar la integridad y confidencialidad de los archivos.
4. **Gestión de Problemas**: GitHub incluye herramientas para gestionar problemas, tareas y solicitudes de extracción. Esto permite un seguimiento detallado de los problemas encontrados durante el desarrollo y facilita su resolución de manera estructurada.
5. **Integración Continua**: GitHub se integra fácilmente con servicios de integración continua, lo que permite automatizar pruebas y despliegues del juego. Esto mejora la calidad del código y acelera el proceso de desarrollo.
6. **Documentación**: GitHub proporciona herramientas para documentar el código y los recursos del juego de manera clara y organizada. Esto facilita la comprensión del proyecto y su mantenimiento a lo largo del tiempo.

En resumen, la elección de GitHub como sistema de control de versiones para el desarrollo de "RUN RUBY!" ofrece una plataforma robusta y eficiente para gestionar el código, los recursos y la colaboración del equipo de desarrollo. Con GitHub, se garantiza un flujo de trabajo ordenado, seguro y colaborativo que contribuirá al éxito del proyecto.

GDScript y GODOT

Se ha optado por Godot como motor de juego debido a sus ventajas significativas sobre Unity. Godot es un motor de código abierto y gratuito, sin suscripciones anuales ni regalías, lo que lo convierte en una opción más accesible y sostenible a largo plazo para desarrolladores indie. Además, Godot presenta una curva de aprendizaje más suave, especialmente para aquellos familiarizados con lenguajes de programación como Python, gracias a la facilidad de uso de GDScript. Estas diferencias son:

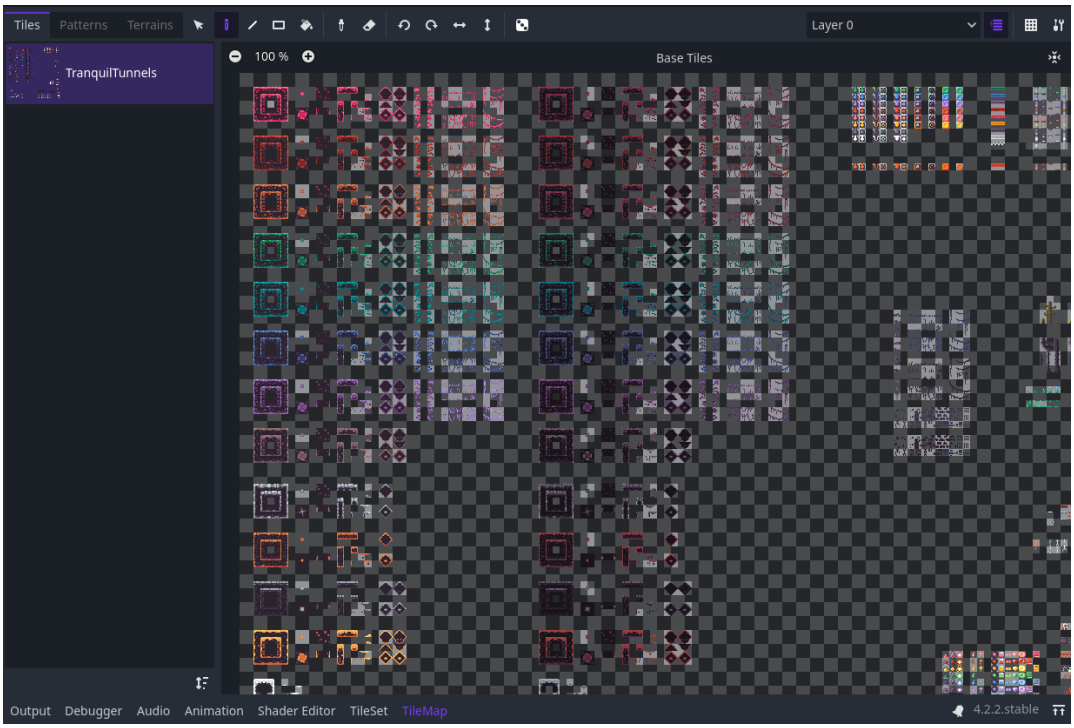


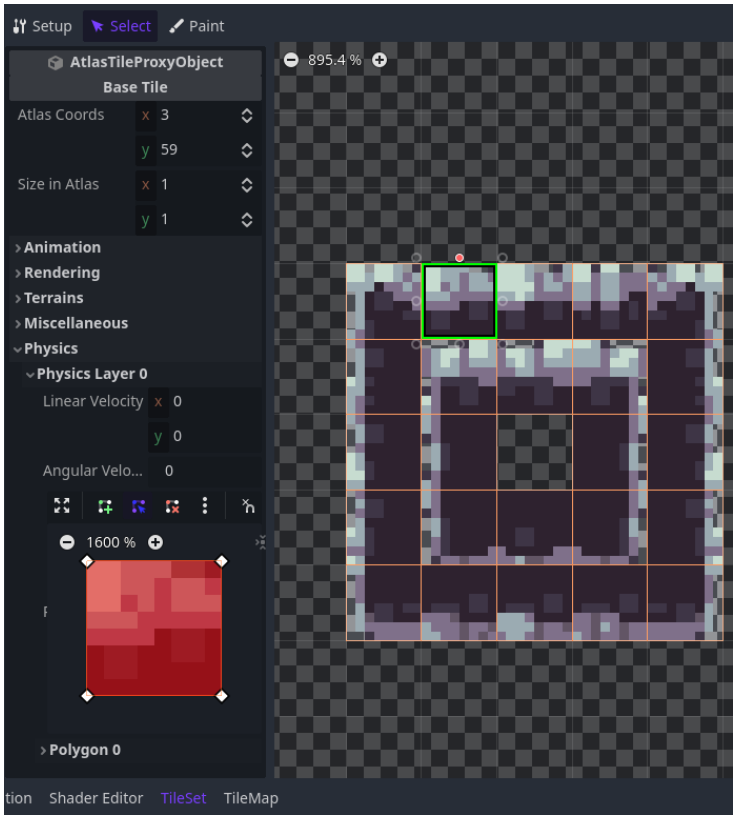
1. **Lenguaje de Programación**: Godot utiliza GDScript, un lenguaje de programación similar a Python, mientras que Unity utiliza C# o JavaScript. Esto puede requerir adaptaciones en el código y la estructura de los proyectos.
2. **Nodos y Escenas**: En Godot, los nodos y escenas se manejan de manera similar a Unity, pero con algunas diferencias en la forma en que se organizan y se utilizan. Por ejemplo, en Godot, los nodos pueden ser instanciados y gestionados de manera más flexible.
3. **Animaciones**: Godot utiliza un sistema de animaciones más simple y fácil de usar que Unity. En Godot, las animaciones se manejan utilizando un nodo AnimationTree y se pueden controlar utilizando condicionales y funciones.
4. **Física**: Godot tiene un sistema de física integrado que es similar a Unity, pero con algunas diferencias en la forma en que se configura y se utiliza. Por ejemplo, en Godot, se puede activar o desactivar el proceso físico en función de las necesidades del juego.
5. **Audio**: Godot tiene un sistema de audio integrado que es similar a Unity, pero con algunas diferencias en la forma en que se manejan los efectos de sonido y la música. Por ejemplo, en Godot, se pueden cargar y reproducir archivos de audio utilizando la función preload.

6. **Gestión de Paquetes**: Godot utiliza un sistema de gestión de paquetes que es similar a Unity, pero con algunas diferencias en la forma en que se manejan los paquetes y las dependencias. Por ejemplo, en Godot, se pueden instalar paquetes utilizando el gestor de paquetes de Godot.
7. **Integración con Bibliotecas**: Godot tiene una mayor cantidad de bibliotecas integradas y compatibles, como SilentWolf para leaderboards globales, lo que facilita la integración de funcionalidades adicionales en el juego.
8. **Curva de Aprendizaje**: Godot tiene una curva de aprendizaje más suave para desarrolladores que están familiarizados con lenguajes de programación como Python, gracias a la facilidad de uso de GDScript.
9. **Costo y Licencia**: Godot es gratuito y de código abierto, sin suscripciones anuales ni regalías, lo que lo convierte en una opción más accesible y sostenible a largo plazo para desarrolladores independientes.
10. **Comunidad y Soporte**: Godot tiene una comunidad activa y un soporte oficial que es similar a Unity, pero con algunas diferencias en la forma en que se manejan las preguntas y los problemas. De todas maneras, gracias al reciente problema con el nuevo sistema de licencia de Unity, Godot ha crecido exponencialmente, aumentando así su comunidad.

Creación de TileMap

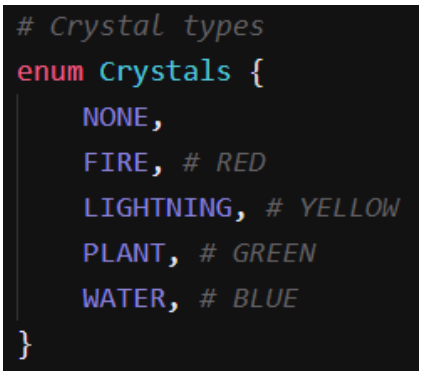
Para crear un TileMap en Godot, se utiliza la clase TileMap que proporciona una forma sencilla de crear y gestionar mapas de tiles. Los tiles pueden ser de diferentes tipos, como sprites, colisiones o incluso objetos móviles. La creación de un TileMap en Godot implica la creación de una capa de tiles y la configuración de las propiedades de cada tile, como su posición, tamaño y tipo.





Gestión de cristales de habilidades

Para gestionar los cristales de habilidades en el juego, se ha utilizado un '*enum*' para almacenar los cristales disponibles y sus respectivos efectos. Cuando el jugador obtiene un cristal se emite una señal. Esta señal es recibido por aquellos objetos que estén conectados a esta, de esta forma, el jugador obtiene el cristal que se ha obtenido sin tener que referenciar de forma directa los objetos entre sí, evitando problemas. Luego, se puede utilizar esta información para activar los efectos del cristal en el juego.



SilentWolf para almacenar Leaderboards globales

SilentWolf es una biblioteca de Godot que proporciona una forma sencilla de almacenar y gestionar leaderboards globales. SilentWolf utiliza un sistema de bases de datos en línea para almacenar los datos de los leaderboards, lo que permite a los jugadores acceder a ellos desde cualquier lugar. Además, SilentWolf ofrece una interfaz fácil de usar para gestionar los leaderboards, incluyendo la capacidad de agregar, eliminar y actualizar registros.

La elección de SilentWolf para almacenar leaderboards globales se basa en su facilidad de uso y su capacidad para manejar grandes cantidades de datos. Además, SilentWolf es una biblioteca oficial de Godot, lo que significa que se puede confiar en su compatibilidad y soporte.

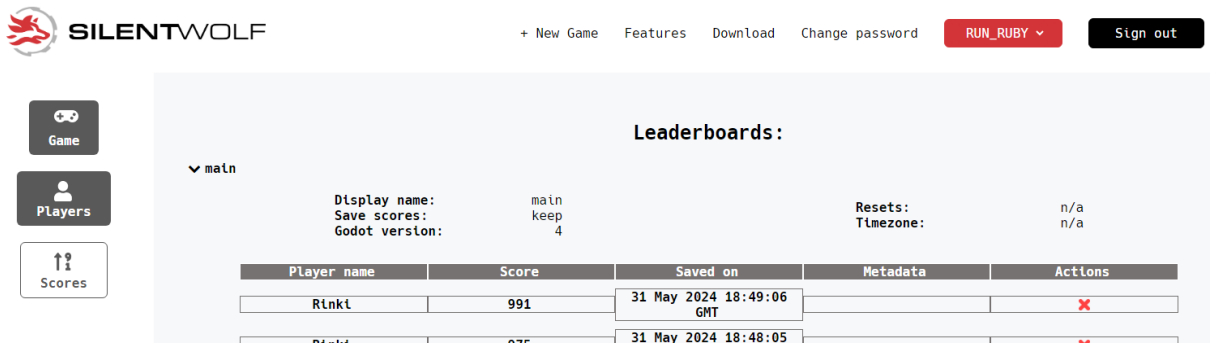
Conclusión

Este enfoque en el diseño y la arquitectura del juego "RUN RUBY!" ha permitido establecer una base sólida para el desarrollo de un nivel dinámico y desafiante, que se alinea con los requisitos y la visión del cliente, garantizando una experiencia de juego atractiva y envolvente para los jugadores.

4.2 Modelo de datos

Estructura de la Base de Datos de SilentWolf

Se ha optado por usar el plugin de SilentWolf para Godot, en este se utiliza un script para guardar de forma automática en la base de datos un nombre y una puntuación. La estructura de la base de datos de SilentWolf se basa en un sistema de bases de datos en línea que permite a los jugadores acceder a los leaderboards desde cualquier lugar. Estas son las características que contiene la base de datos por defecto:



Componentes de la Base de Datos

- **Leaderboards**: Los leaderboards son los registros de los mejores puntajes de los jugadores. Estos se almacenan en la base de datos y se actualizan en tiempo real.
- **Jugadores**: Los jugadores son los usuarios que participan en el juego. Cada jugador tiene un registro en la base de datos que incluye su nombre, puntaje.
- **Puntajes**: Los puntajes son los resultados de los jugadores en diferentes niveles o desafíos. Estos se almacenan en la base de datos y se utilizan para calcular los leaderboards.
- **Niveles**: Los niveles son los desafíos que los jugadores deben superar para obtener puntajes. Cada nivel tiene sus propias condiciones y desafíos.
- **Desafíos**: Los desafíos son los objetivos que los jugadores deben cumplir para obtener puntajes. Estos pueden incluir tareas como recoger cristales, derrotar enemigos o completar niveles.

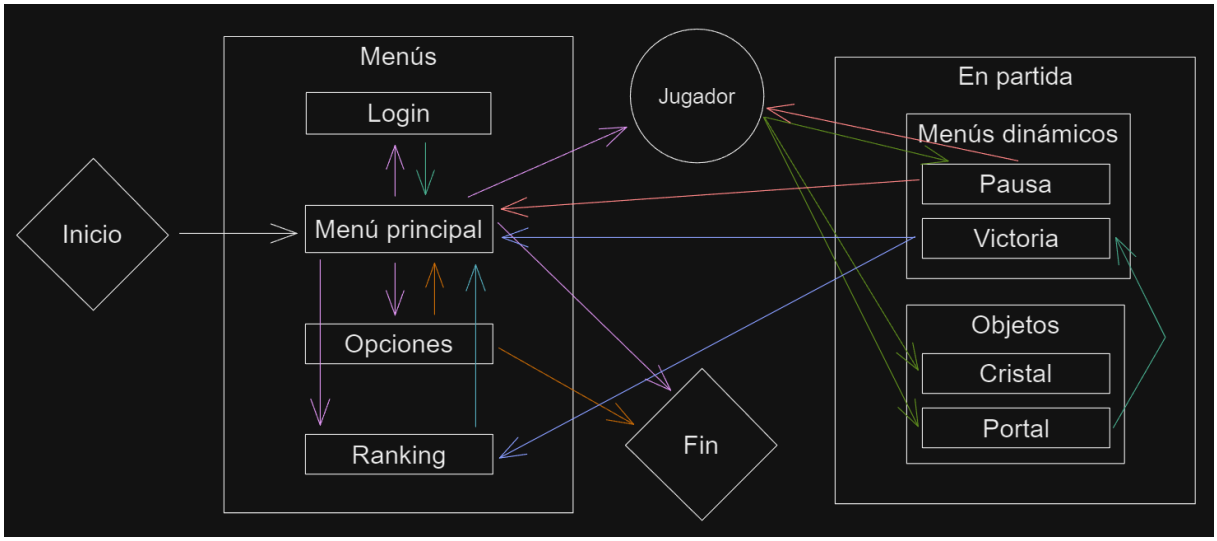
Funcionalidades de la Base de Datos

- **Almacenamiento de Leaderboards**: SilentWolf almacena los leaderboards en la base de datos y los actualiza en tiempo real.
- **Gestión de Jugadores**: SilentWolf gestiona los registros de los jugadores, incluyendo su nombre, puntaje y otros datos relevantes.
- **Gestión de Puntajes**: SilentWolf gestiona los puntajes de los jugadores, incluyendo la suma de los puntajes en diferentes niveles o desafíos.
- **Gestión de Niveles**: SilentWolf gestiona los niveles, incluyendo sus condiciones y desafíos.
- **Gestión de Desafíos**: SilentWolf gestiona los desafíos, incluyendo los objetivos que los jugadores deben cumplir para obtener puntajes.

JSON como persistencia de datos local

Como método para guardar datos localmente, en este caso, utilizado para almacenar el Nombre del jugador, y así no hacer que se tenga que identificar cada vez que entre en el juego, se ha optado por un archivo JSON. Mencionado posteriormente en la parte de codificación, en el archivo JSON se almacenan los datos de forma encriptada, ya que así se evita la manipulación de este. Se guarda bajo la carpeta del usuario, haciendo así que si otro usuario se conecta en el mismo equipo, no tenga el mismo archivo. Si el archivo no existe, se genera al poner el nombre por primera vez.

4.3 Modelo de procesos



El modelo de procesos describe el flujo de interacción del jugador a través de diferentes etapas y menús. A continuación se detalla el modelo de procesos con las diferentes secciones y opciones disponibles:

Inicio

El juego comienza con el menú principal.

Menús

- **Login**: El jugador puede acceder al menú de inicio de sesión para ingresar al juego.
- **Menú Principal**: Desde el menú principal, el jugador puede acceder a diferentes opciones como jugar, configurar opciones, ver el ranking y salir del juego.
- **Opciones**: Permite al jugador ajustar configuraciones como el volumen y ver los controles.
- **Ranking**: Muestra la tabla de clasificación con los mejores puntajes de los jugadores.

Jugador

El jugador interactúa con el juego, moviendo a Ruby a través del nivel.

En Partida

- **Menús Dinámicos**:
  - **Pausa**: Durante el juego, el jugador puede pausar la partida para salir, reiniciar o tomar un descanso.
  - **Victoria**: Al completar un nivel, se muestra la pantalla de victoria con opciones para reiniciar, ver el ranking o regresar al menú principal.

- **Objetos**:
  - **Cristales**: El jugador puede romper cristales que otorgan cartas con habilidades especiales.
  - **Portal**: Los portales permiten al jugador finalizar el nivel.

Fin

El juego concluye cuando el jugador decide salir del juego.

Este modelo de procesos describe de manera detallada las diferentes etapas y opciones disponibles para el jugador. Desde el inicio hasta la finalización de la partida, se establece un flujo claro de interacción que guía al jugador a través de los menús, las opciones y las acciones dentro del juego.

5. Construcción del proyecto

5.1 Codificación

En el desarrollo de "RUN RUBY!", se ha integrado la librería SilentWolf para GODOT, una herramienta que facilita el almacenamiento de Scores y Leaderboards de manera eficiente. Esta integración ha mejorado significativamente la experiencia de juego al proporcionar una funcionalidad de competencia y seguimiento de progreso para los jugadores de forma global. Con esta adición, "RUN RUBY!" ahora cuenta con un sistema robusto de almacenamiento de datos y puntuaciones, permitiendo a los jugadores competir entre sí y mantener un registro de sus logros en el juego de manera transparente y efectiva.

Manejo de Coyote Time y Jump Buffer

```
# Handle the coyote timer
if coyote_timer > 0:
    coyote_timer -= delta

# Add the gravity
if not is_on_floor() and not is_gliding:
    velocity.y += gravity * delta

# Reset the coyote timer when on floor
if is_on_floor():
    coyote_timer = COYOTE_TIME

# Handle the jump buffer timer
if jump_buffer_timer > 0:
    jump_buffer_timer -= delta

# Handle the jump
if ((Input.is_action_just_pressed("WASD_SPACEBAR") and coyote_timer > 0) or (is_on_floor() and jump_buffer_timer > 0)) and not is_gliding:
    velocity.y = JUMP_VELOCITY # Add the jump velocity
    coyote_timer = 0 # Reset the coyote timer
    jump_buffer_timer = 0 # Reset the jump buffer timer

# Get the input direction
direction = Input.get_axis("WASD_A", "WASD_D")
```

Este fragmento de código maneja el tiempo de coyote (coyote_timer) y el buffer de salto (jump_buffer_timer). El coyote time es un período de tiempo en el que el personaje puede saltar después de dejar el suelo, y el buffer de salto es el tiempo en el que el personaje puede saltar antes de aterrizar. Estos mecanismos permiten al jugador saltar y moverse de manera más fluida y controlada.

La lógica detrás de este código es compleja, ya que debe manejar varios estados y condiciones para determinar si el personaje puede saltar o no. El uso de variables y condicionales para controlar estos estados y condiciones es un ejemplo de cómo se pueden manejar complejidades en el desarrollo de juegos.

Animaciones y Estados de Animación

```
# Animations
animation_tree.active = true # Enable the animation tree
state_machine = animation_tree.get("parameters/playback") # Get the player's state machine
animation_tree.connect("animation_finished", self._on_animation_finished) # Connect the animation finished signal
```

```
# Handle the animation finished signal
func _on_animation_finished(anim_name):
    # Handle the attack animations
    if anim_name == "Attack1" and not state_machine.get_current_node() == "Attack2":
        # If the first attack is done, switch back to idle
        is_attacking = false
        hitbox_area.deactivate_hitbox()
    elif anim_name == "Attack2":
        # If the second attack is done, switch back to idle
        is_attacking = false
        hitbox_area.deactivate_hitbox()
```

Este trozo de código maneja las animaciones del personaje y los estados de animación. El '*animation_tree'* es un objeto que controla las animaciones del personaje, y el '*state_machine*' es un objeto que gestiona los estados de animación. El código conecta la señal '*animation_finished'* del '*animation_tree*' a una función que maneja el fin de las animaciones.

La lógica detrás de este código es interesante porque muestra cómo se pueden manejar las transiciones entre diferentes estados de animación. El uso de condicionales y funciones para manejar estos estados es un ejemplo de cómo se pueden implementar complejidades en el desarrollo de juegos.

Persistencia de datos local encriptada

```
func _save_data(path : String):
    # Open save file with encryption
    var file = FileAccess.open_encrypted_with_pass(path, FileAccess.WRITE, SECURITY_KEY)

    # Print error if file can't be opened
    if file == null:
        print(FileAccess.get_open_error())
        return

    # Data to be saved
    var data = {
        "player_data":{
            "name": player_data.name,
            "score": player_data.score,
        }
    }

    # Save data
    var json_string = JSON.stringify(data, "\t")
    file.store_string(json_string)
    file.close()
```

```
func _load_data(path : String):
    if FileAccess.file_exists(path): # File exists
        # Open save file with encryption
        var file = FileAccess.open_encrypted_with_pass(path, FileAccess.READ, SECURITY_KEY)

        # Print error if file can't be opened
        if file == null:
            print(FileAccess.get_open_error())
            return

        # Load data
        var content = file.get_as_text()
        file.close()

        # Parse data
        var data = JSON.parse_string(content)
        # Check if data is valid
        if data == null:
            printerr("Cannot parse %s as json_string: (%s)" % [path, content])
            return

        # Set player data
        player_data = PlayerData.new()
        player_data.name = data.player_data.name
        player_data.score = data.player_data.score

    else: # File doesn't exist
        printerr("Cannot open non-existing file at %s!" % path)
```

Este fragmento de código maneja la encriptación y desencriptación de datos utilizando la función '*FileAccess.open_encrypted_with_pass*'. Esta función abre un archivo en modo de lectura o escritura utilizando una clave de seguridad proporcionada. La clave de seguridad se utiliza para encriptar y desencriptar los datos almacenados en el archivo.

La función '*_save_data*' utiliza esta función para abrir el archivo en modo de escritura y encriptar los datos utilizando la clave de seguridad. Luego, los datos se almacenan en el archivo utilizando la función '*file.store_string*'.

Por otro lado, la función '*_load_data*' utiliza esta función para abrir el archivo en modo de lectura y desencriptar los datos utilizando la clave de seguridad. Luego, los datos se cargan en la variable '*content*' utilizando la función '*file.get_as_text*'. Finalmente, los datos se parsean utilizando la función '*JSON.parse_string*' y se almacenan en la variable '*player_data*'.

Este enfoque de encriptación y desencriptación proporciona una forma segura de almacenar y cargar datos en el juego, protegiendo la información del jugador de posibles intrusiones malintencionadas. Si no, el usuario podría editar el contenido del JSON libremente, incrementando su puntuación por ejemplo.

5.2 Pruebas

Pruebas de rendimiento

- Medir el rendimiento del juego en diferentes dispositivos y configuraciones de hardware
- Optimizar el uso de recursos como CPU, GPU y memoria para mantener un rendimiento fluido
- Identificar y corregir cuellos de botella que puedan afectar la experiencia de juego

Pruebas de jugabilidad

- Realizar sesiones de prueba con usuarios beta para evaluar la jugabilidad y la curva de aprendizaje
- Identificar problemas de diseño de niveles, mecánicas de movimiento y dificultad
- Recopilar feedback de los usuarios y realizar ajustes para mejorar la experiencia de juego

Pruebas de compatibilidad

- Probar el juego en una variedad de dispositivos y plataformas (PC, móvil, consolas)
- Hay que asegurar que el juego se ejecute correctamente en diferentes sistemas operativos y configuraciones
- Identificar y corregir problemas de compatibilidad que puedan afectar a los usuarios

Pruebas de funcionalidad

- Verificar que todas las funcionalidades del juego, como el sistema de ranking global, los menús y las opciones de configuración, funcionen correctamente
- Probar escenarios de uso comunes y casos extremos para garantizar la estabilidad del juego

Pruebas de seguridad

- Implementar medidas de seguridad para proteger la integridad de los datos de los usuarios
- Probar el juego contra posibles ataques o intentos de manipulación del sistema de ranking

Estas pruebas exhaustivas ayudarán a garantizar que el juego sea estable, funcional y ofrezca una experiencia de juego fluida y satisfactoria para los usuarios en todas las plataformas compatibles.

5.3 Manual de instalación

El proyecto se compila para la plataforma indicada, se genera un ejecutable que no requiere instalación.

Ejecutar el archivo "*RUN_RUBY.exe*".

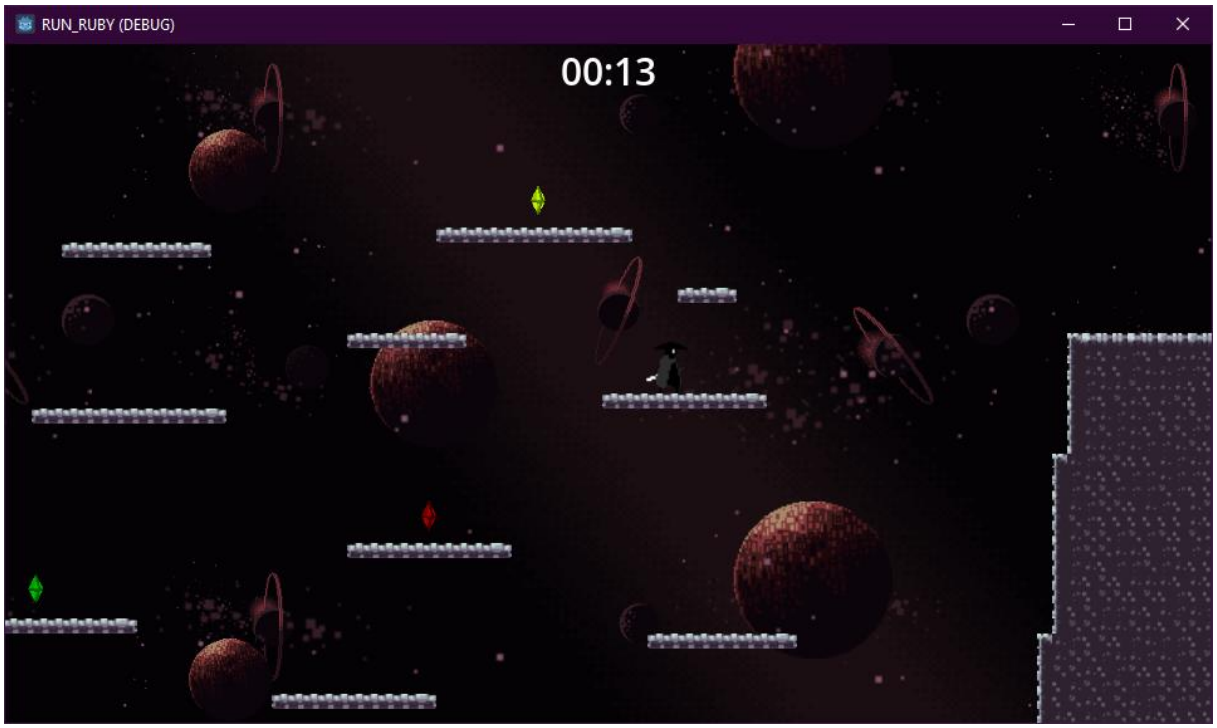
5.4 Manual de usuario

Manual de Usuario - "RUN RUBY!"

¡Bienvenido al juego "RUN RUBY!". A continuación, encontrarás toda la información necesaria para disfrutar al máximo de esta emocionante experiencia.

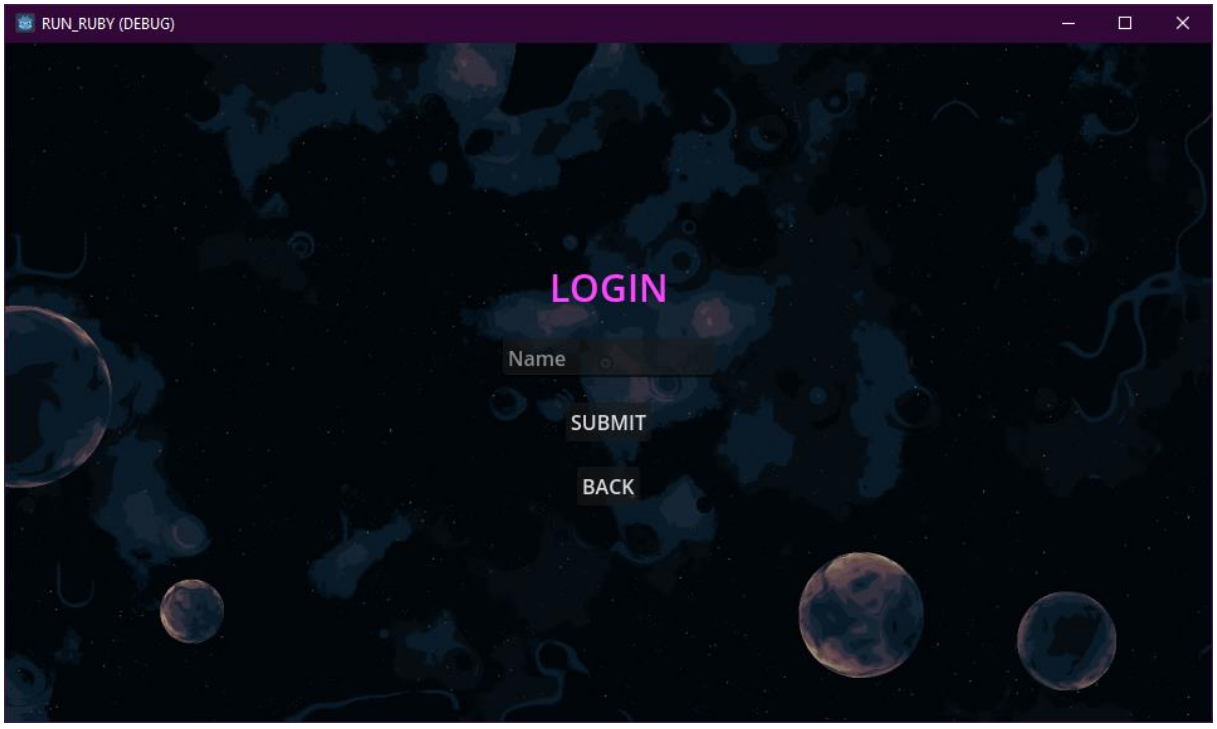
Objetivo del Juego:

Tu objetivo en "RUN RUBY!" es llegar a la meta lo más rápido posible, superando obstáculos y encontrando el camino más eficiente para ello, debiendo explorar todos los caminos posibles y posibilidades con los cristales. Utiliza tus habilidades y destrezas para completar cada nivel de forma creativa y eficiente.

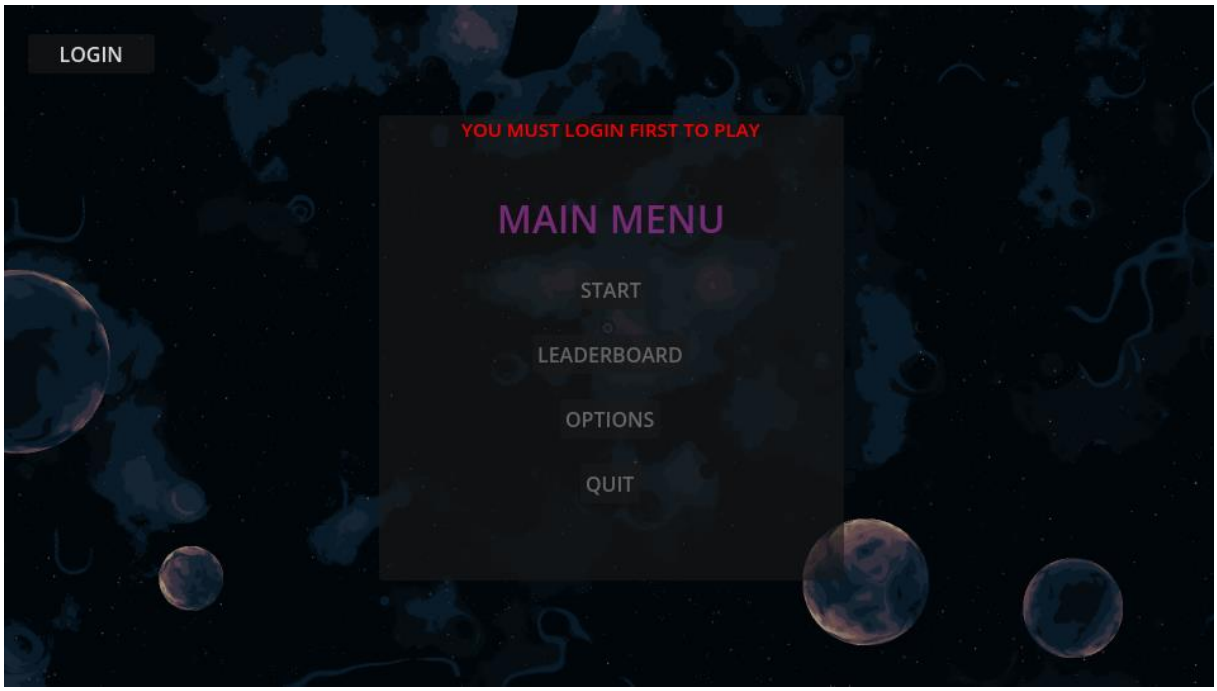


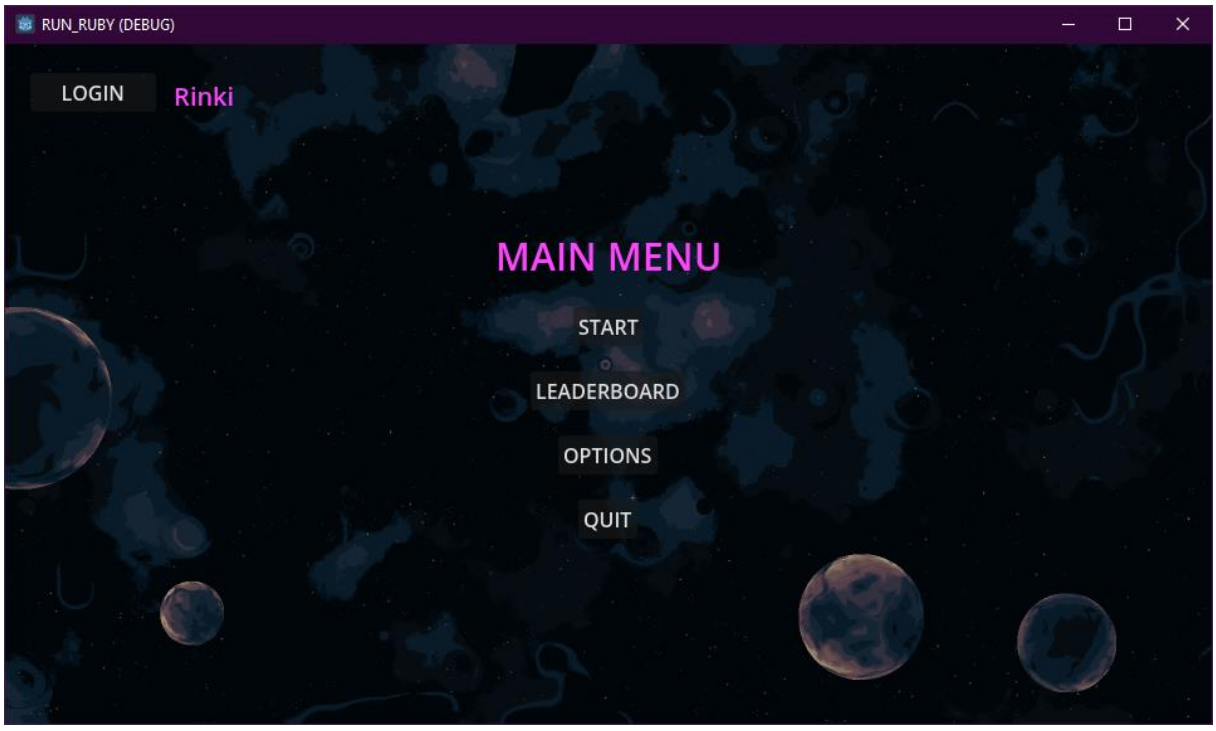
Menús:

- **Menú de Login**: Ingresa tu nombre para identificarte en el juego.

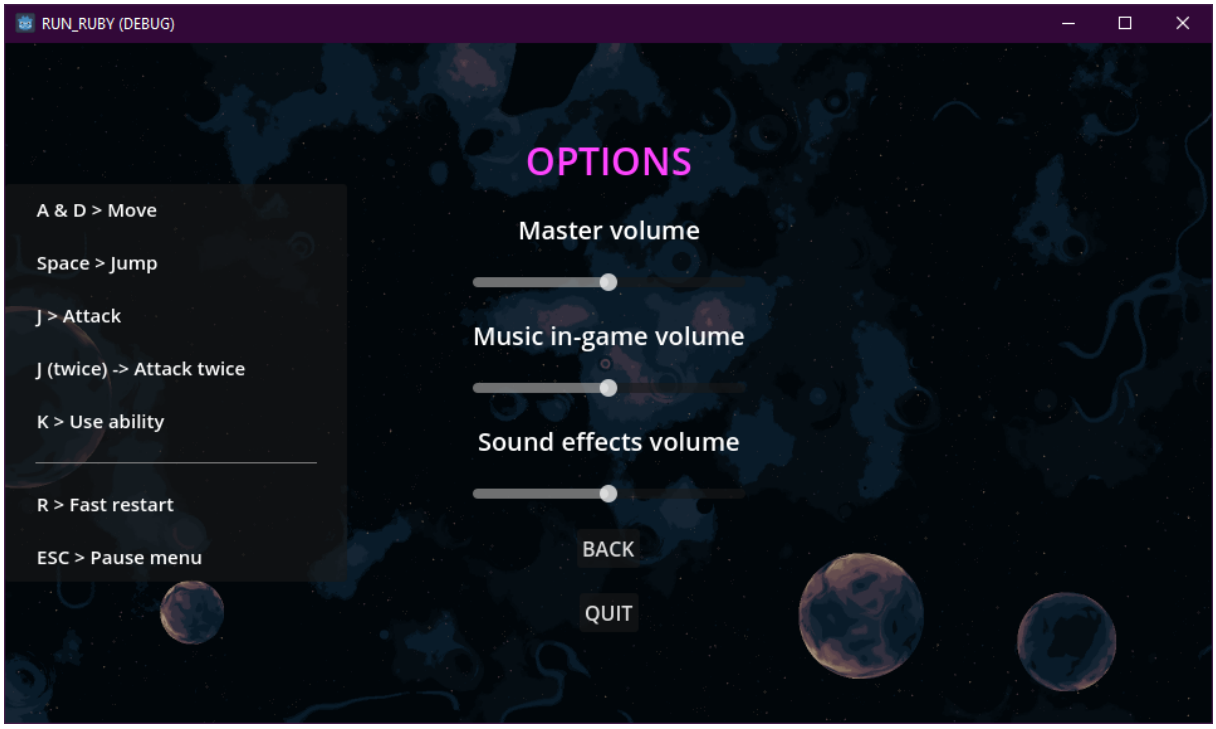


- **Menú Principal**: Accede al login, comienza el juego, accede a las opciones y consulta el ranking.

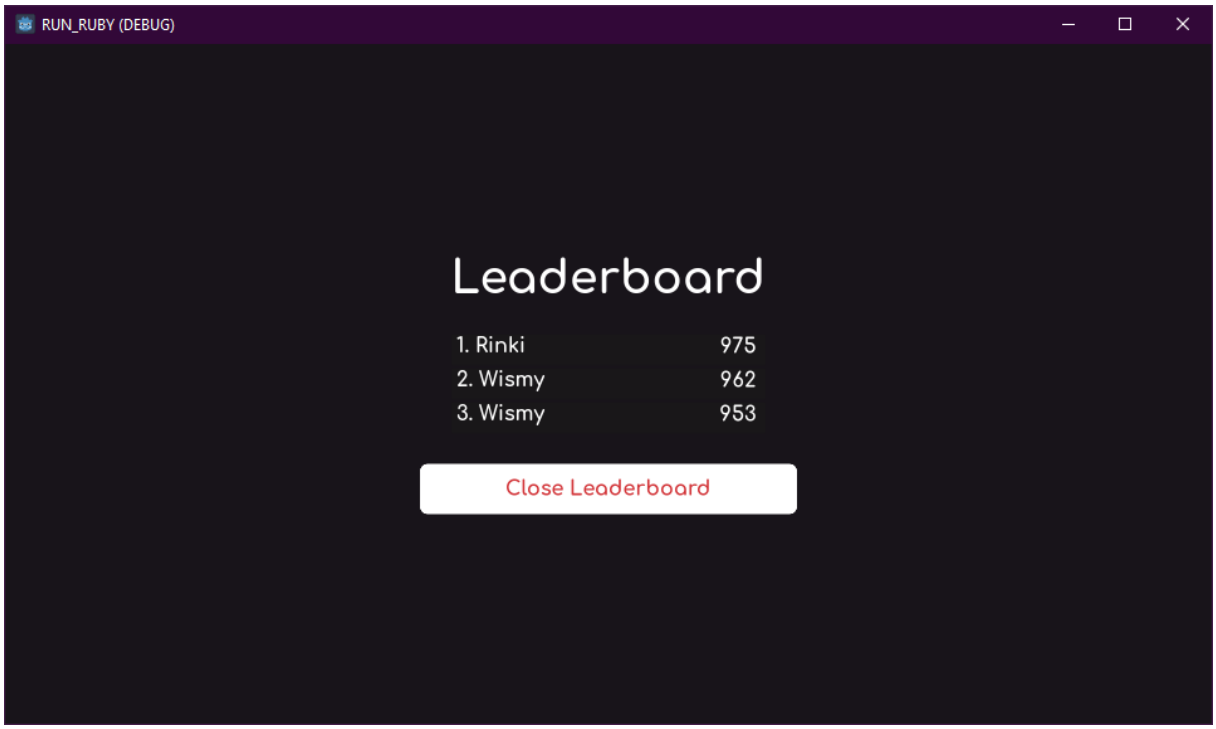




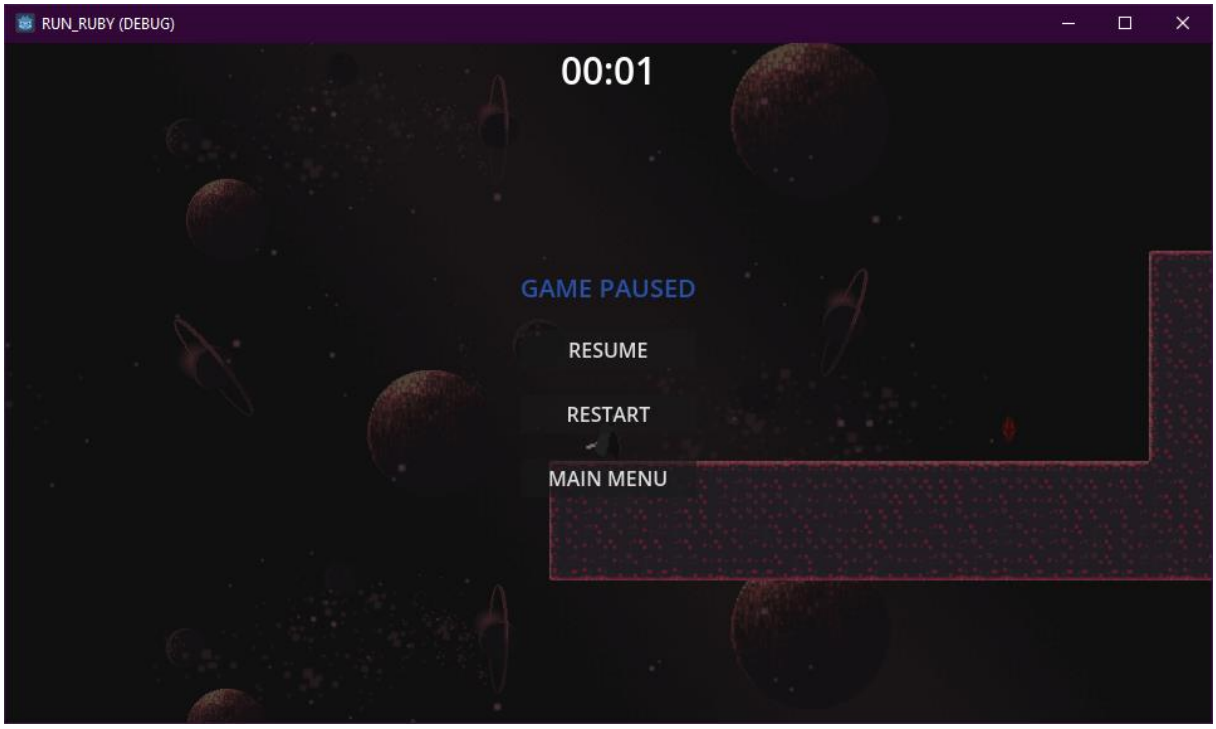
- **Menú de Opciones**: Ajusta los volúmenes del juego.



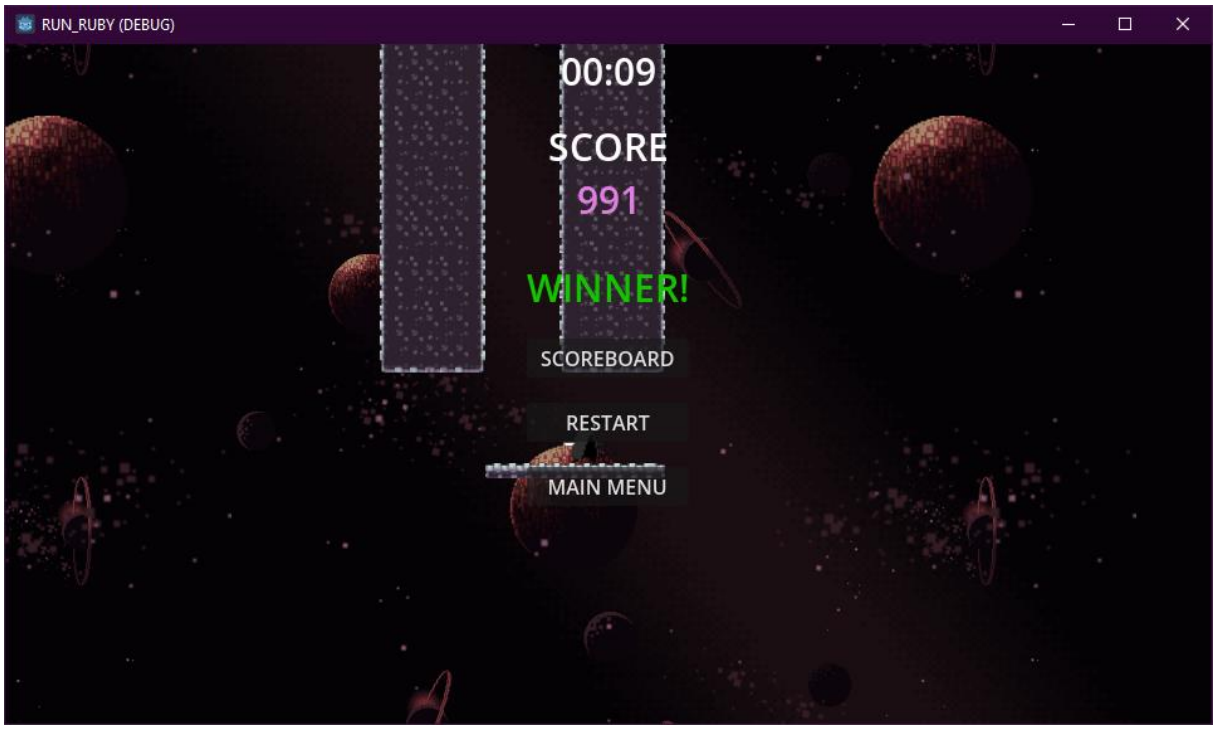
- **Menú de Ranking**: Consulta la tabla de clasificación global.



- **Menú de Pausa**: Pausa el juego cuando quieras.



- **Menú de Victoria**: Decide que hacer al terminar la partida y ver tu puntuación.



Controles:

- **Movimiento Horizontal**: Utiliza las teclas ***A*** (izquierda) y ***D*** (derecha) para moverte horizontalmente en el nivel.

- **Salto**: Presiona **Barra espaciadora** ***** para saltar y superar obstáculos.

- **Ataque**: Usa la tecla ***J*** para realizar ataques y romper cristales en tu camino, que te proporcionarán habilidades especiales.

- **Habilidades Especiales**: Obtenidas al romper los cristales, actívalas con la tecla ***K***.

- **Carta de fuego *(Doble salto)***: Te impulsa verticalmente igual que un salto, pero con más fuerza (1.5x).

- **Carta de rayo *(Dash)***: Te permite realizar un rápido movimiento horizontal en línea recta.

- **Carta de planta *(Caminar por el aire)***: Te permite caminar horizontalmente en el aire durante un breve tiempo (1s).

- **Carta de agua *(Caída rápida)***: Tras una breve canalización (0.1s), realizarás una caída rápida.

Recomendaciones:

- Explora cada nivel para descubrir secretos y desafíos adicionales.
- Practica tus habilidades para mejorar tu tiempo y puntaje en el juego.
- ¡Diviértete y disfruta de la experiencia única que ofrece "RUN RUBY!"!

¡Ahora estás listo para sumergirte en la acción y la diversión de " RUN RUBY!"! ¡Que la aventura comience!

6. Evaluación final del proyecto

6.1 Evaluación del diseño, del proceso y del resultado

El diseño y desarrollo de "RUN RUBY!" ha sido un proceso desafiante pero gratificante. Si bien el juego actual se limita a un solo nivel, el enfoque en la calidad y la modularidad del código ha permitido crear una base sólida para futuras expansiones. El uso de GODOT y GDScript ha demostrado ser una elección acertada, brindando flexibilidad y eficiencia en la implementación de las mecánicas del juego.

Además, la experiencia de trabajar con una nueva tecnología como GODOT ha sido una oportunidad valiosa para superar los desafíos y aprender a utilizar herramientas y lenguajes de programación diferentes. En particular, la transición desde Unity, con el que estuve familiarizado en clase, ha sido un desafío interesante. La necesidad de adaptarme a un nuevo entorno de desarrollo y a un lenguaje de programación como GDScript ha sido un proceso de aprendizaje significativo, que me ha permitido mejorar mis habilidades y ampliar mi perspectiva sobre las posibilidades de desarrollo de juegos.

En resumen, el desarrollo de "RUN RUBY!" ha sido un proceso satisfactorio que me ha permitido aplicar los conceptos aprendidos en clase y superar los desafíos de trabajar con una nueva tecnología.

6.2 Conclusiones y lecciones aprendidas

A través del desarrollo de "RUN RUBY!", se han adquirido valiosas lecciones sobre la importancia del diseño modular, las pruebas exhaustivas y la optimización del rendimiento. Además, se ha aprendido a trabajar con GODOT y GDScript, ampliando las habilidades en el campo del desarrollo de videojuegos.

6.3 Posibles ampliaciones futuras

Si bien el juego actual ofrece una experiencia sólida y desafiante, existen múltiples oportunidades para expandir y mejorar el proyecto en el futuro. Algunas posibles ampliaciones incluyen:

- Agregar más componentes interactivos en el mapa, como plataformas móviles, trampas y obstáculos.
- Introducir enemigos y desafíos adicionales para aumentar la dificultad y la emoción del juego.
- Desarrollar más niveles con diferentes diseños y desafíos, manteniendo la jugabilidad fluida y la competitividad.
- Implementar un sistema de guardado y carga de progreso para que los jugadores puedan retomar sus partidas.
- Añadir más habilidades y mejoras para el samurái, brindando más opciones estratégicas a los jugadores.

Gracias a la arquitectura modular y la ausencia de código "hardcoded", estas ampliaciones y mejoras pueden implementarse de manera eficiente, aprovechando la base sólida establecida durante el desarrollo inicial de "RUN RUBY!".

7. Bibliografía/webgrafía

- **Documentación oficial de Godot**: Godot Engine. (s.f.). Documentación de Godot. Recuperado de https://docs.godotengine.org/
- **Artículos y guías sobre el desarrollo de videojuegos en 2D**: Game Developer. (s.f.). Game Developer | Noticias y blogs del sector del juego. Recuperado de https://www.gamedeveloper.com/
- **Videotutoriales de YouTube**: YouTube. (s.f.). YouTube. Recuperado de https://www.youtube.com
- **Assets de Itch.io**: Itch.io. (s.f.). Itch.io. Recuperado de https://itch.io/game-assets/free
- **Base de datos de SilentWolf**: SilentWolf. (s.f.). SilentWolf. Recuperado de https://silentwolf.com
- **Diagrama de Gantt**: OnlineGantt. (s.f.). OnlineGantt. Recuperado de https://www.onlinegantt.com/#/gantt
- **Diagrama de casos de uso**: ElPatoDraw. (s.f.). ElPatoDraw. Recuperado de https://elpatodraw.com
- **Control de versiones**: GitHub. (s.f.). GitHub. Recuperado de https://github.com