C.F.G.S. Desarrollo de Aplicaciones Multiplataforma

AGIS

Autor: Raúl Muñoz García

Tutor: Francisco Javier Ortega Nogueras

Índice

Introducción

Expectativas

La idea inicial con la que se comenzó el proyecto era un juego 2D altamente inspirado en los mundialmente conocidos videojuegos *Hollow Knight, Neon White* y *Celeste*. De tal manera que el juego combinase las mecánicas centrales del *Hollow Knight*, estas siendo el género, el estilo de la narrativa y facetas como el inventario, los enemigos y sus mecánicas, los escenarios y el diseño de niveles. En cuanto a *Celeste*, por preferencia personal se quería que el juego tuviese un gameplay rápido, con movimientos que respondan correctamente a las intenciones del jugador y que sea agradable pero complejo al mismo tiempo. Mientras que del *Neon White,* heredaba la mecánica de cartas sobre la que el juego está basado.

Figura 1: Hollow Knight, Nintendo Official Website

Figura 2: Celeste, Nintendo Official Website

Figura 3: Neon white, Nintendo Official Website

Antecedentes

Antes de comenzar con el desarrollo de este proyecto ya contaba con el desarrollo de dos aplicaciones de índole similar.

Uno de ellos fue un juego desarrollado en solitario, que empezó con la misma premisa que este proyecto, un juego que combinase *Hollow Knight* y *Celeste*, pero eventualmente se acabo decantando más hacia el estilo y el flujo de *Celeste*. Este juego fue desarrollado en C# en su totalidad y uno de los fuertes de este proyecto fue la fluidez de los controles y lo bien que reaccionaban a las acciones del usuario.

El segundo proyecto estaba destinado a ser una entrega en la reciente *Málaga Jam*, que consistía en un juego que recogía varios minijuegos dentro de él. Algo bastante diferente al estilo de su precedente.

Objetivos

El objetivo de este proyecto es desarrollar un juego 2D que cuente con mecánicas varias, sencillas y entretenidas, una base lógica, que permita ser escalado en un futuro. Un proyecto que cuente con guardado de datos tanto en local como con API, garantizando la persistencia de la información del jugador.

Además, se busca aplicar el mayor número de buenas prácticas posibles mediante el uso de una estructura modularizada y una clara jerarquía de clases permitiendo la incorporación de nuevas funcionalidades según la necesidad de forma progresiva.

Descripción

Funcionalidades

El resultado obtenido ha sido un videojuego 2D del género *metroidvania*, con una ambientación postapocalíptica y futurista. El título se estructura en varios niveles, incluyendo un tutorial inicial, y presenta una variedad de biomas diferenciados que enriquecen la exploración.

En cuanto a las mecánicas implementadas, el juego ofrece al usuario un sistema de inventario interactivo, tiendas para la compra de objetos, puntos de guardado distribuidos por el mapa, sistema de combate contra enemigos, así como habilidades o interacciones condicionadas por los objetos que el jugador posea.

La ambientación visual se ha construido mediante el uso de los assets de la serie de pago *Dark*, del artista Penusbmic, disponibles a través de su Patreon y en la plataforma itch.io. Estos recursos han permitido diseñar una interfaz coherente con el entorno del juego, logrando una estética consistente y envolvente. A nivel sonoro, se han integrado efectos, música y sonidos ambientales adaptados al contexto de cada escena, con el objetivo de potenciar la inmersión del jugador.

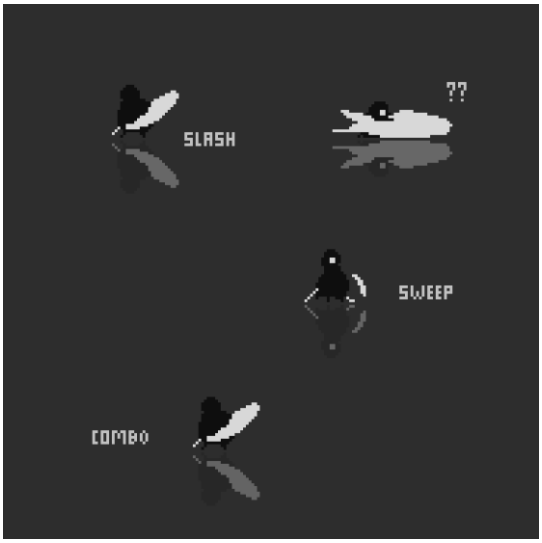

Figura 4: Ejemplo estético de los personajes, Assasin by Penusbmic

Respecto al sistema de guardado, se almacena información relevante como el inventario, la salud y la moneda de forma local en el dispositivo del usuario. Además, se registra el tiempo total de juego de cada partida en una base de datos *MongoDB*, a través de una API desarrollada específicamente para el proyecto.

Dicha API ha sido implementada en C# utilizando el framework *ASP.NET*, y se encarga de gestionar los datos persistentes del juego. Como se trata de una experiencia para un solo jugador, el sistema permite mantener hasta tres archivos de guardados activos por usuario, gestionados de manera independiente.

Organización

Este juego sigue el sistema de organización de Godot por nodos, lo que significa que en cada escena hay un nodo raíz del cual heredan todos los nodos que figuran en esta. Estos nodos pueden bien ser nodos como tal, o escenas, en cuyo caso el nodo raíz de la escena pasa a ser un nodo hijo de la escena en la que ha sido instanciada. Lo que es lo mismo, si tenemos una escena con un nodo raíz “A” con un nodo hijo “B” y otra escena con un nodo raíz “C”, si instanciamos la escena que contiene a los nodos *A* y *B*, “A” pasa a ser nodo hijo del nodo raíz “C” en la nueva escena, lo que implica que “B” es un nodo “nieto” de “C”.

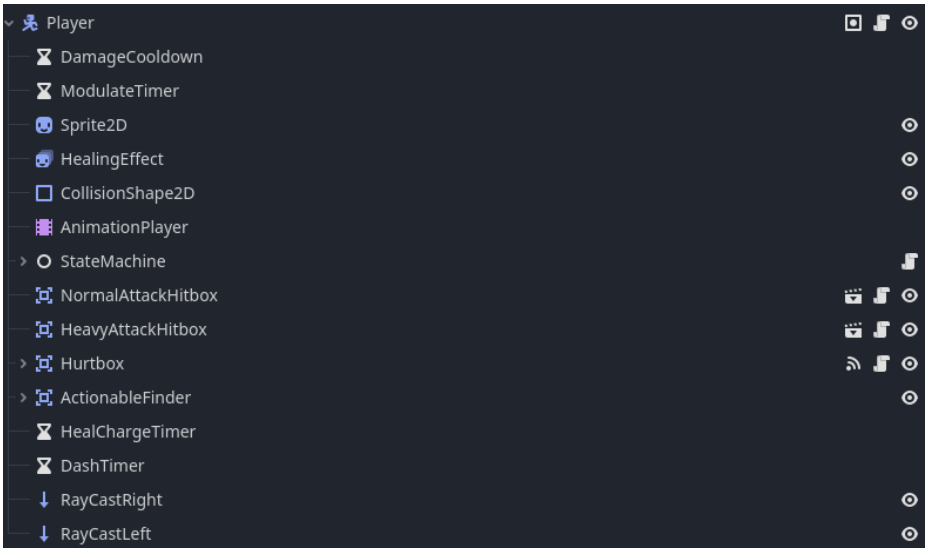

Figura 5: Árbol de nodos de la escena del jugador, Imagen propia

Para los archivos del juego, los archivos están separados por carpetas en función de a que categoría corresponden. Las categorías son: *Addons, Assets, Scripts, Scenes, Dialogues* y *Themes*. Internamente cada una de las carpetas cuenta con subcarpetas para una mayor modularización de los archivos. Todos aquellos archivos que puedan ser considerados como clases irreducibles están a nivel de la carpeta que les correspondan, un ejemplo de esto sería el script base de los enemigos que se encuentra a nivel de la carpeta *Enemigos* dentro de la carpeta *Scripts*.

Instalación y Preparación

- Godot 4.4.1 estable o superior.
- Aseprite o cualquier otro programa de creación de assets.
- ASP.Net 8 o superior.
- Visual Studio o Visual Studio Code.
- GitHub o Git.
- Audacity o similar.
- Postman o Insomnia para las pruebas de la API

Una vez se cumplan los requisitos, lo primero sería hacer fork o clonar el repositorio en GitHub que contiene el proyecto para así no perder el registro del control de versiones anteriores que figura en el historial de commits. Una vez se tenga el proyecto ya en nuestro sistema solo habría que abrirlo con Godot, actualizarlo a una versión mas reciente en caso de que la versión de Godot no sea 4.4.X o en caso de que el motor lo solicite.

Para la API, al igual que con Godot, tendremos que descargarnos o clonarnos su repositorio de GitHub y abrirlo con Visual Studio o Visual Studio Code. Como se ha mencionado anteriormente, es recomendable usar programas como Insomnia o Postman para realizar las pruebas, pero si se prefiere, se puede usar Curl o similar. Lo anterior es en el caso de que se quiera reutilizar la API existente. Si esa no es la intención, lo mas recomendable seria crear otra siguiendo la guía original de Microsoft.

Diseño funcional

Descripción

En este apartado se presentan los elementos clave del funcionamiento interno del juego. Se incluyen diagramas simples que muestran cómo se relacionan las entidades principales, que puede hacer el jugador y cómo fluye la lógica del juego.

Diagramas

En este apartado se van a tratar varios diagramas que representen varios aspectos del proyecto, desde las posibles interacciones del jugador con el entorno, el flujo de las pantallas en el juego, hasta las relaciones entre los elementos del proyecto.

Como caso de uso se va a exponer todas las posibles interacciones que tiene el jugador.

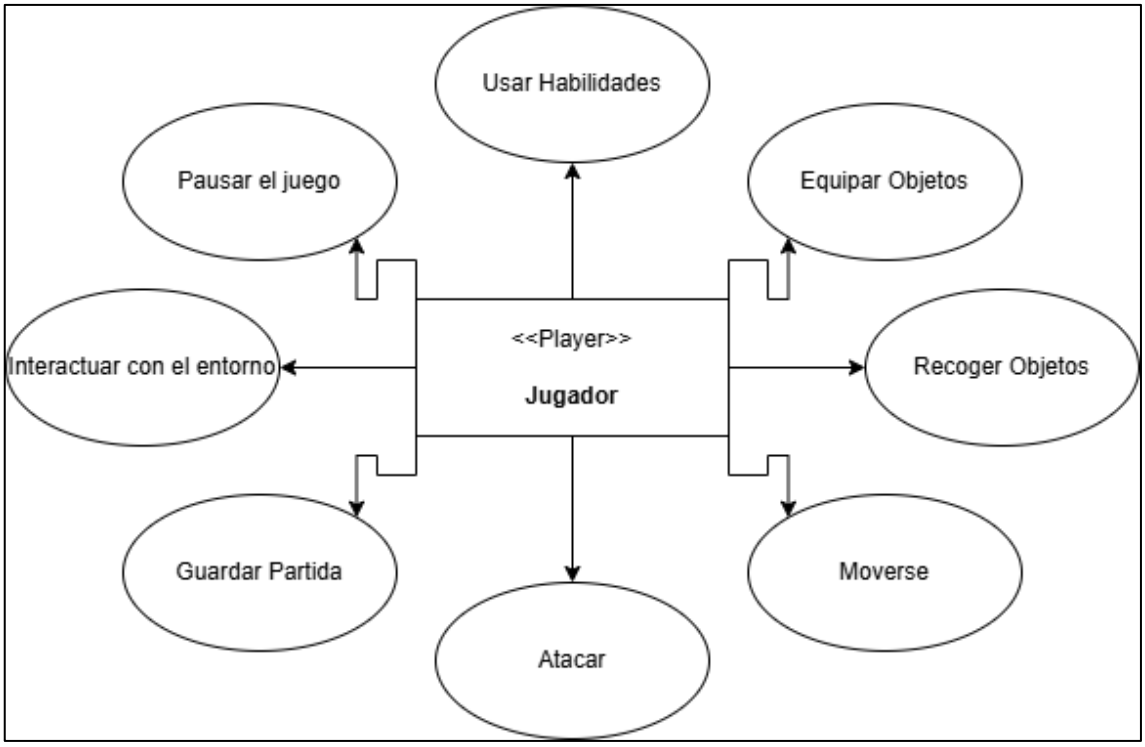

Figura 6: Casos de uso de las interacciones del jugador, Imagen propia

Para el diagrama de flujo, se presenta el flujo entre pantallas del proyecto, todas las pantallas a las que el usuario tiene acceso mientras juegan. Sin contar todas aquellas que se encuentren dentro de otra, como es el caso del menú de pausa.

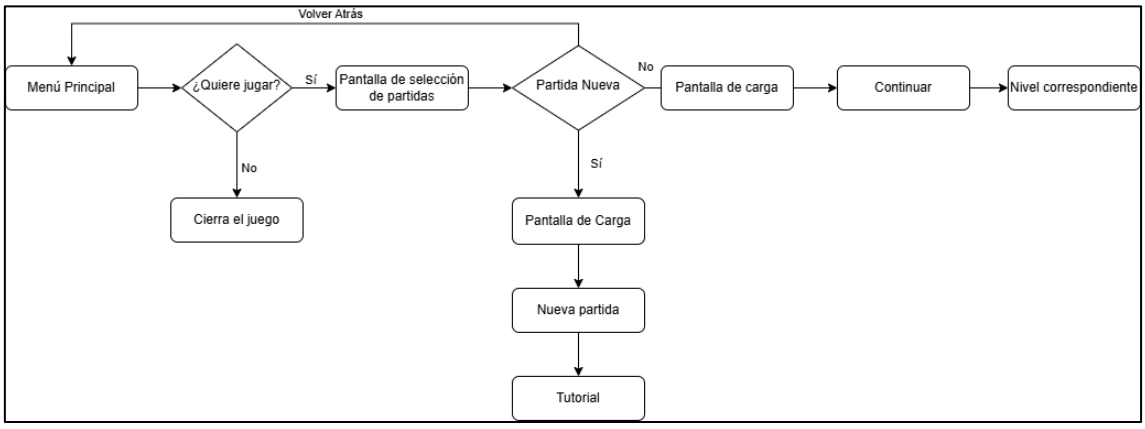

Figura 7: Flujo de pantallas, Imagen Propia

Como diagrama entidad relación, se ha decidido usar el sistema de inventario del jugador.

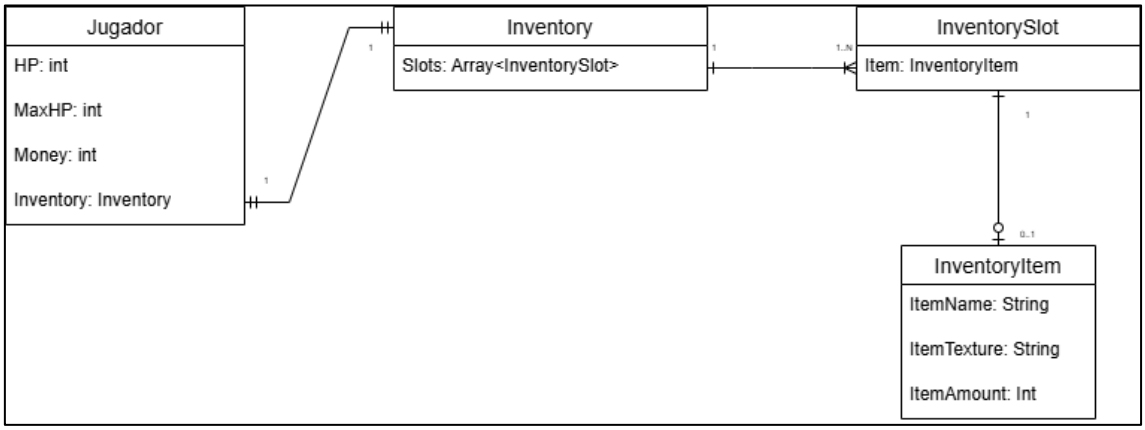

Figura 8: Diagrama entidad relación del inventario del usuario, Imagen Propia

Desarrollo

Secuencia de desarrollo

El desarrollo de este proyecto comenzó con el desarrollo de los personajes, tanto aliados como enemigos.

En el caso del jugador, la máquina de estados gestiona acciones como caminar, saltar, atacar, recibir daño y usar habilidades, entre otras. Cada estado representa un comportamiento concreto y contiene su propia lógica de entrada, ejecución y salida, evitando conflictos entre acciones simultáneas y simplificando la implementación de nuevas mecánicas.

De forma similar, los enemigos también cuentan con una máquina de estados que controla su patrullaje, detección del jugador, ataque y reacción al daño. Gracias al uso del plugin *Beehave*, se integra este sistema con árboles de comportamiento que permiten mayor complejidad y variabilidad en la IA, pero conservando el control estructurado de los estados individuales.

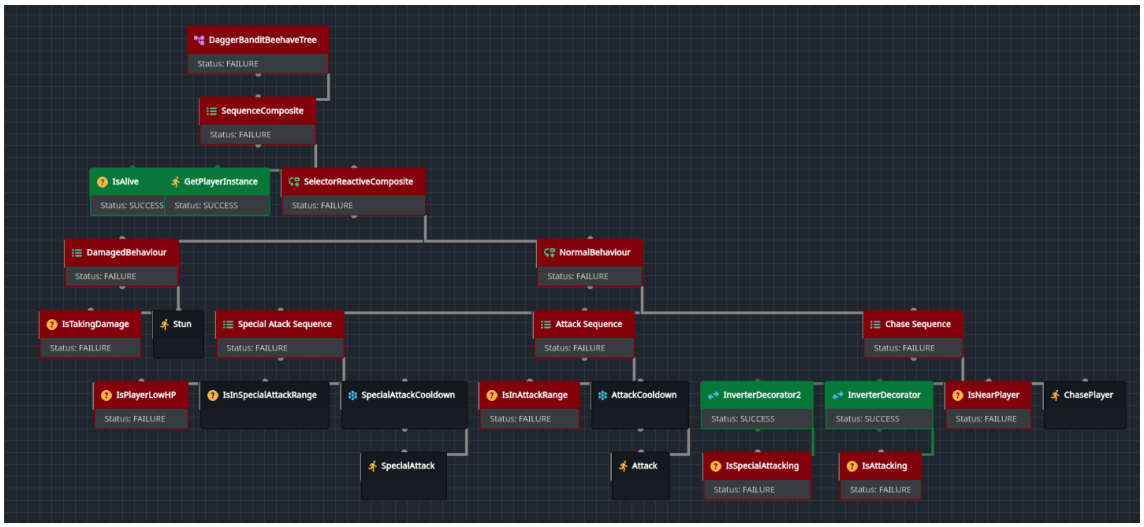

Figura 9: Máquina de estados con uso de Beehave en tiempo de ejecución, Imagen propia

Una vez los personajes habían sido terminados, se pasó al desarrollo de las escenas, los mapas, en concreto el tutorial. Una escena diseñada para introducir al usuario a las mecánicas y para que se familiarice con los controles y el aspecto de los objetos.

Dichas escenas constan de un nodo raíz del que partirán el resto de los nodos que compongan la escena. En general cada escena se compone de varias partes:

- El fondo.
- El tilemap que compondrá las plataformas y los detalles de la escena.
- Los enemigos.
- El jugador, la cámara y el inventario.
- Y los objetos adicionales, sean funcionales como las trampas o los cofres o aquellos que sean meramente decorativos, como arbustos o lianas.

Una vez desarrollados los mapas, se procedió a desarrollar el sistema de diálogos entre personajes haciendo uso del plugin *Dialogic*. Este plugin permite la creación y una amplia personalización de los diálogos del juego, su aspecto, la interacción que tiene con el juego e incluso modificar aspectos de este en función de las elecciones del usuario.

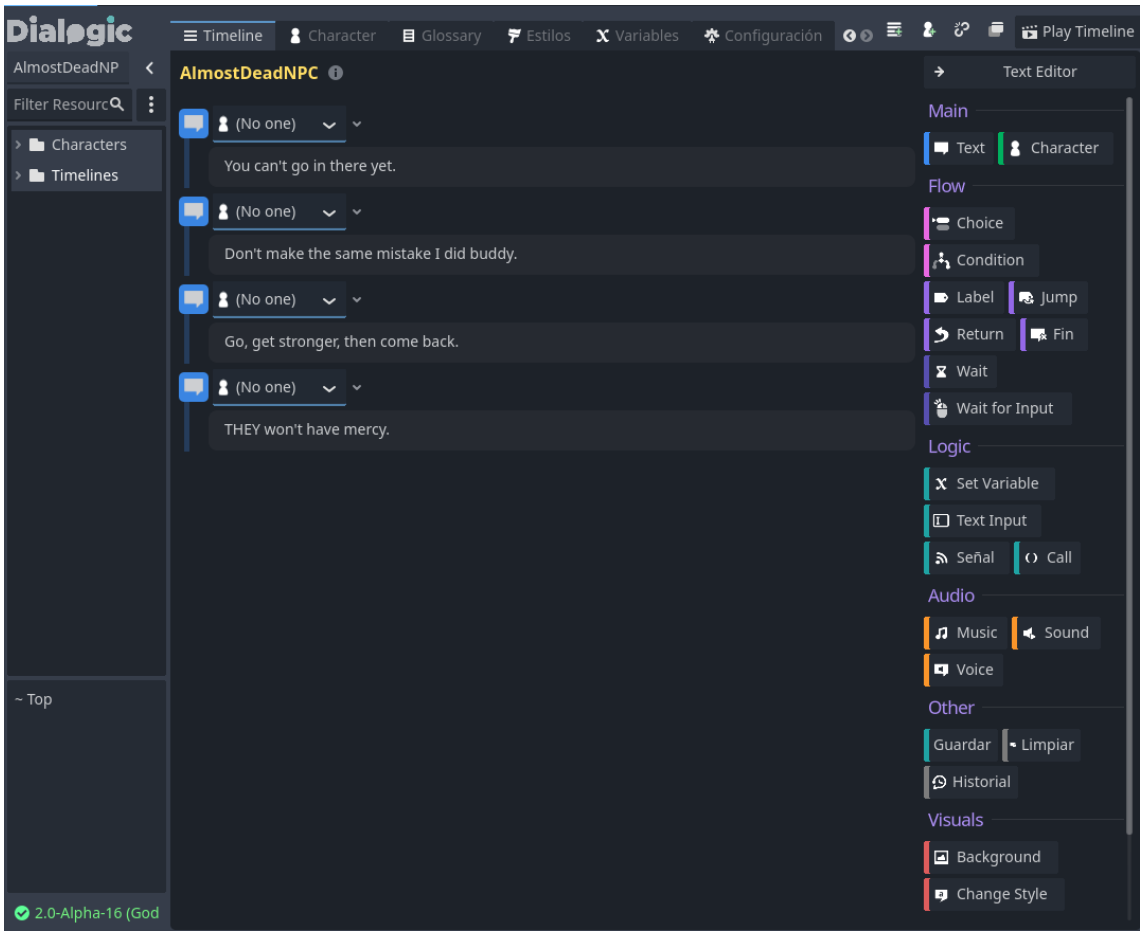

Figura 10: Panel de desarrollo de diálogos con Dialogic, Imagen propia

La persistencia de datos ha sido de los últimos aspectos que se han desarrollado puesto que muchos datos tenían que estar predefinidos para la buena y directa implementación y así evitar tener que realizar modificaciones constantemente debido a la implementación de nuevos campos, variables, características o cualquier aspecto que fuese necesario su guardado. Como ya ha sido mencionado previamente, la persistencia consiste en guardados de datos local y en la nube simultáneamente, por lo que la prioridad la tuvo el guardado local y eventualmente se añadió la API para guardar información de menor volumen.

El último punto de desarrollo del proyecto ha sido el sonido, puesto a que se requería tener todo ya definido, mecánicas, diálogos, objetos, enemigos para que se pudiesen elegir sonidos acordes a la temática del juego y no hubiera mucha

discordancia entre ellos. Este proceso puede llegar a ser tedioso si no se tiene clara la temática o no se tienen varios sonidos implementados en los que basarse para la elección de sonidos futuros.

He aquí una tabla con datos obtenidos de la sección de commits del proyecto. Cabe destacar, que muchos aspectos han sido desarrollados en paralelo según necesidad o en función de la implementación de modificaciones o funciones nuevas.

	Fecha de Inicio	*Fecha de Finalización*	*Tiempo Estimado*
Personajes	23/03/2025	09/06/2025	~20h
UI	01/04/2025	09/06/2025	~30h
Escenas / Niveles	23/03/2025	05/06/2025	~45h
Mecánicas	10/04/2025	03/06/2025	~55h
Persistencia	03/06/2025	05/06/2025	~10h
Sonido	06/06/2025	09/06/2025	~7.5h

Tabla 1: Tabla de tiempo de desarrollo, Tabla propia

Problemas y decisiones

Durante el proceso de desarrollo muchos problemas han sido resultados de malas decisiones, fallos inesperados, incompatibilidades o complicaciones excesivas de los sistemas. Algunos de ellos que merecen una especial mención son los siguientes:

- Problemas con la resolución.
- Problemas con la reproducción de audio.
- Problemas con el reconocimiento de mando en las interfaces.
- Problemas con el inventario del jugador.

La resolución de los problemas anteriores conllevó modificaciones significativas en el proyecto y su estructura. En lo relacionado a la resolución de los problemas, quedan aquí reflejadas las decisiones tomadas.

El problema con la resolución es posiblemente el factor que más ha marcado al proyecto. El proyecto cuenta con una resolución aproximada de 480x270 píxeles, sin embargo, está sobrescrita a 1920x1080 píxeles mediante la configuración de proyecto de Godot. Esta configuración causó una mala visibilidad de los sprites del juego, jittering en algunas ocasiones, distorsión en la fuente de los diálogos y en cualquier aspecto visual que no fuese de un tamaño preciso. Este problema apareció bastante tarde en el proceso de desarrollo del juego, por lo que la solución más efectiva que no requiriese modificar la totalidad del proyecto fue escalar de nuevo los assets para que la distorsión fuese la mínima posible.

Mientras que la reproducción de audio fue un problema causado debido al diseño del sistema de reproducción. Este sistema estaba configurado inicialmente como un pool de reproductores de audio que eran ocupados y liberados según la demanda. Esto suponía que, si el pool tenía un tamaño definido de 3 reproductores activos, solo era posible reproducir tres sonidos al mismo tiempo, lo que resultaba en que algunos sonidos eran parados antes de que pudiesen terminar debido a la necesidad de otro sonido de ser reproducido. La solución más efectiva fue modificar la capacidad total del pool y crear un reproductor adicional para sonidos que tienen que ser reproducidos durante una duración extendida como la música o el sonido ambiente.

Un clave requisito para un juego como lo es Agis es que pueda ser jugado con mando. Esto implica que todo pueda ser hecho con mando, navegación entre pantallas y juego. En un motor como lo es Godot, la integración de mando para las interfaces puede ser tediosa o excesivamente simple, dependiendo de el enfoque que se le dé. Como ya se ha mencionado, algunos de los problemas provienen de malas decisiones y este es uno de ellos.

Inicialmente, se desarrollaron todas las interfaces para ser navegadas con ratón y para dar un toque más personalizado, se estableció un cursor personalizado, que

consistía en una escena personalizada con su sprite, forma de colisión y área de detección. Teóricamente, esta estructura ofrecía una solución ideal para simular la interacción del ratón en las interfaces, sin embargo, en la práctica presentó múltiples problemas de funcionamiento. Un problema en la lógica que se encargaba de alternar entre el método de entrada en función si se estaba usando el teclado o el ratón resultó en un retraso en el desarrollo del desplazamiento del cursor durante el uso del mando. Esta funcionalidad en sí ha sido uno de los causantes de la mayoría de los fallos y de retrocesos del proyecto, puesto que sin importar lo que ocurriese, el desplazamiento del ratón mientras se usaba el mando resultaba en el puntero bloqueado en la esquina superior derecha de la pantalla.

Eventualmente, se abandonó esta idea y se cambió el enfoque a algo mucho menos llamativo, pero funcional. Se modificó el cursor original de Windows usando la funcionalidad interna de Godot, invalidando así el uso de una escena específica para la creación del cursor y se añadió la capacidad de registrar el foco en los componentes que conforman la interfaz para permitir la navegación.

De manera similar a la navegación entre pantallas, el inventario del jugador estaba diseñado primordialmente para su uso con teclado y ratón. No solo su uso con mando no estaba implementado inicialmente, sino que su uso con ratón era imposible. Problemas con la capacidad de los nodos para poder registrar la entrada por ratón retrasaron la implementación del mando durante un tiempo considerable. Este fallo se debía a que los nodos de control como son aquellos nodos que componen las interfaces se encontraban en la misma capa que la capa a la que pertenecen los nodos que conforman toda la estructura del mapa, esto generaba que la señal no era interceptada por los nodos de control sin la ayuda de un nodo Canvas Layer. Este nodo genera una nueva capa en la que los nodos de control se pueden hospedar y que la entrada por teclado, ratón y mando no sea interceptada por cualquier otro nodo.

Control de versiones

El proyecto cuenta con un extenso control de versiones mediante el uso de Git con la plataforma GitHub. En ella se encuentra el repositorio del proyecto con sus más de cien commits que reflejan perfectamente el proceso de desarrollo del proyecto durante el periodo de tiempo definido. Dicha plataforma permite al desarrollador volver a un punto de guardado específico si se necesita.

Pruebas

Este proyecto no cuenta con pruebas unitarias como tal, sino que ha sido probado por varios de mis compañeros y familiares y amigos. Estas personas han participado activamente como usuarios de prueba, aportando observaciones valiosas sobre aspectos como la visibilidad, el flujo de juego, los controles y la estética general.

Distribución

Proceso de exportación

Godot como herramienta permite una fácil exportación para diferentes formatos o sistemas operativos como Windows, Linux, MacOS, iOS y Android. El proceso para realizar la exportación es relativamente sencillo. Dentro de la configuración del proyecto nos encontramos la opción de exportar el proyecto. He aquí la descripción del proceso.

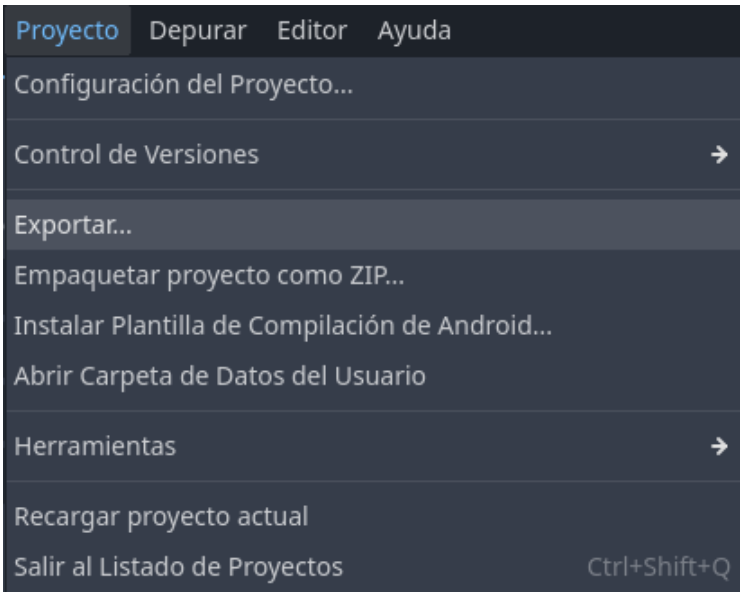

Figura 11: Panel de configuración del proyecto en Godot Engine, Imagen Propia

Dentro del menú de exportación, añadiremos la opción de exportación para Windows. En este menú seleccionaremos la opción de ejecutable para que se nos genere un archivo .exe para una fácil distribución. Cabe destacar que antes de poder exportar el proyecto como un archivo, habría que descargar las plantillas de exportación para Windows que ofrece Godot.

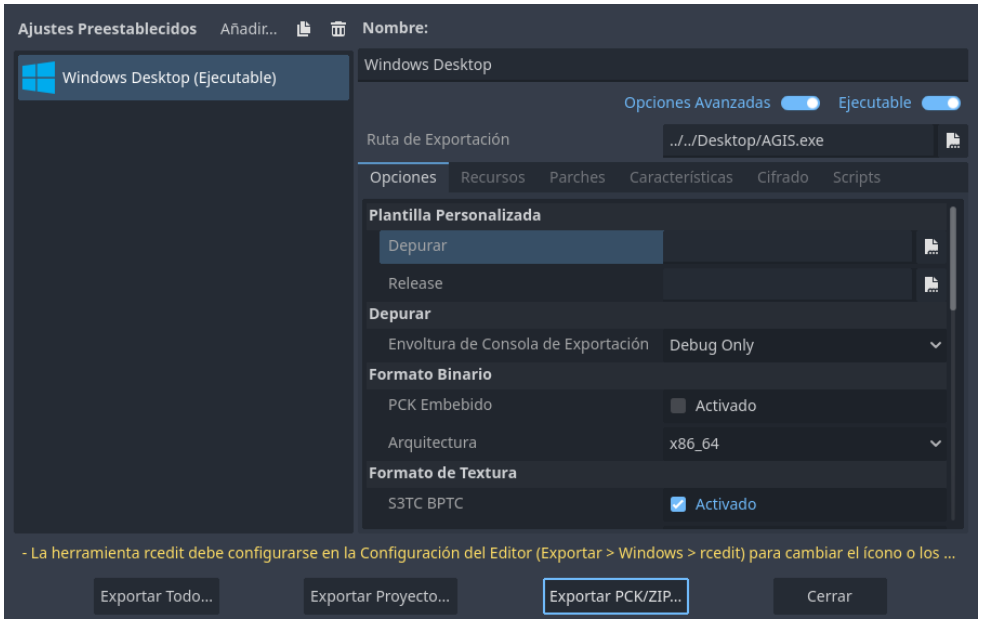

Figura 12: Panel de exportación, Imagen propia

Antes de generar el archivo, la opción “Envoltura de Consola de Exportación” se encarga de determinar si el proyecto será exportado con una consola de depuración o sin ella.

Para modificar aspectos como el icono de la exportación, hay que configurar la herramienta *rcedit*, como puede verse en la imagen superior. Una vez configurada, generaremos nuestro archivo .ico que será el próximo icono de la aplicación una vez exportada.

Para configurar esto, hemos de ir a la página oficial de GitHub de *rcedit* y descargar el ejecutable. Una vez descargado, en la pestaña de configuración del editor de Godot, iremos al apartado de exportación y añadiremos la ruta hasta el ejecutable.

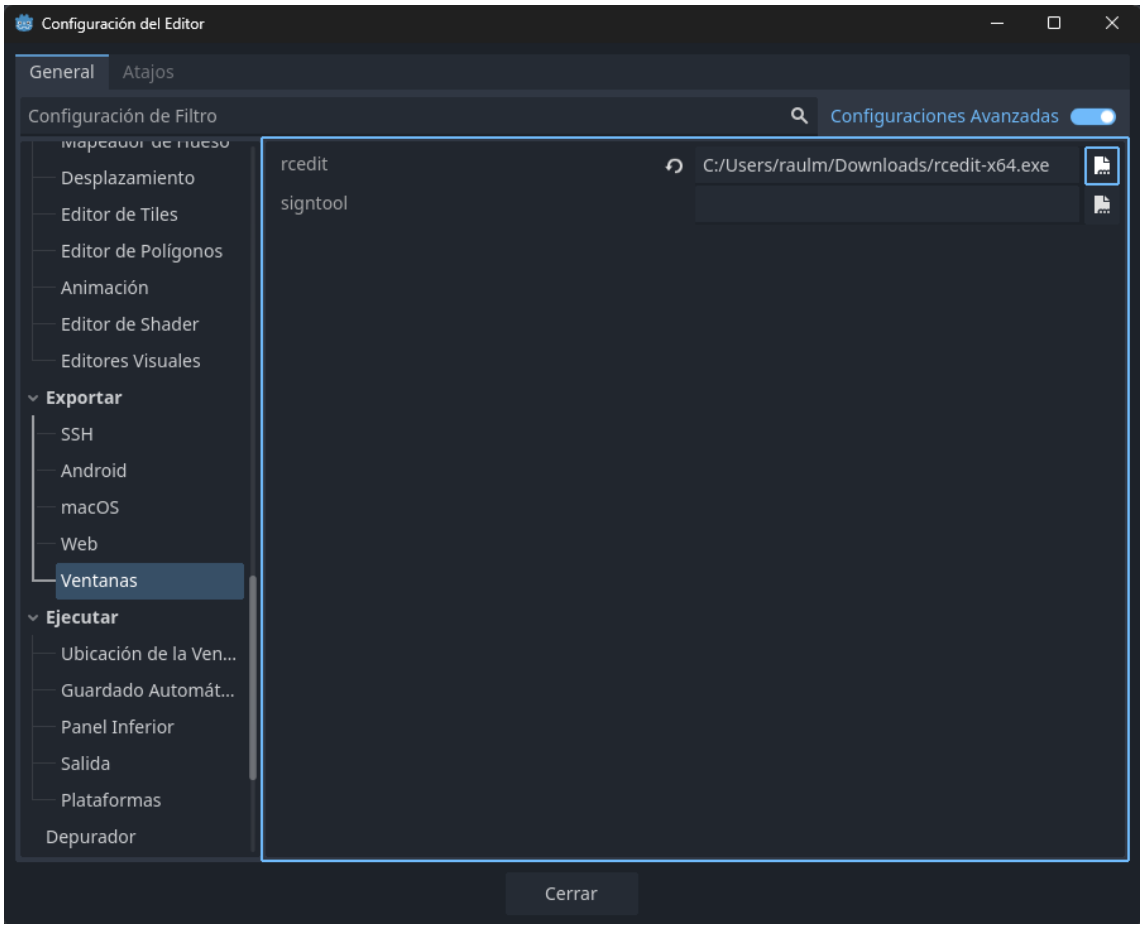

Figura 13: Panel de configuración del editor, Imagen propia.

Después de la acción anterior, volvemos al panel de exportación y comprobamos como la advertencia ha desaparecido.

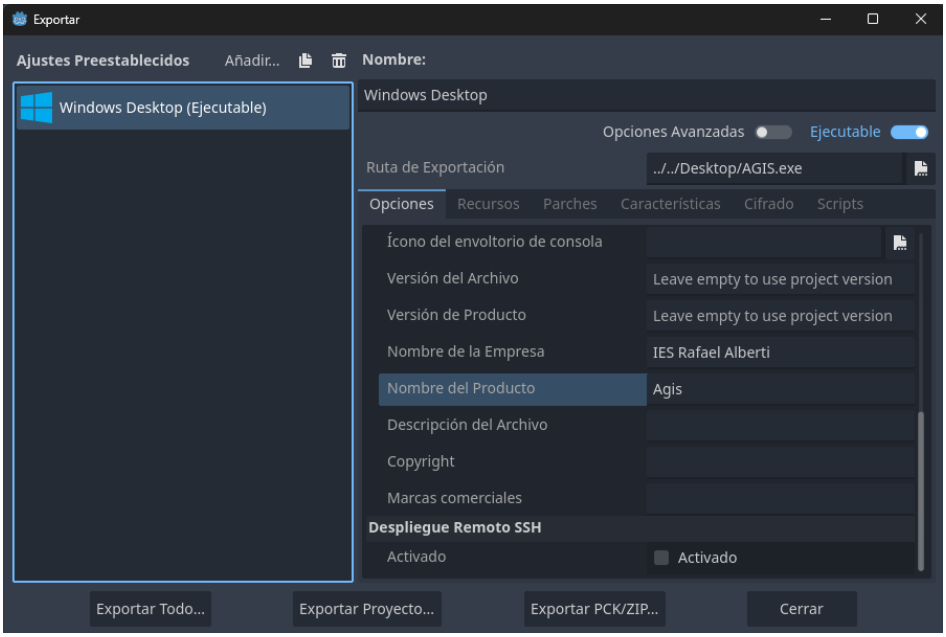

Figura 14: Panel de exportación, Imagen propia

Una vez estemos listos para exportar, después de haber firmado el ejecutable y añadido los datos necesarios, podemos proceder a la exportación.

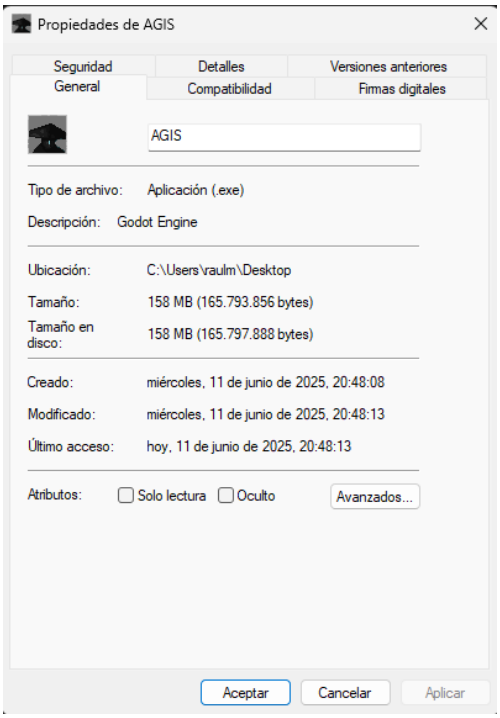

Figura 15: Propiedades del archivo exportado, Imagen propia

Distribución

Actualmente el proyecto se encuentra disponible en itch.io. El enlace a la página figura en el apartado *Enlaces y Referencias*.

Manual

Instalación

El proceso de instalación es extremadamente sencillo, en la mayoría de los casos, solo consiste en ejecutar el ejecutable.

GitHub

En caso de obtener el proyecto desde GitHub, solo habría que encontrar el archivo ejecutable dentro de la carpeta del proyecto.

Itch.io

En caso de haber descargado el proyecto de itch.io, el ejecutable es el único archivo que será descargado de la página.

Uso de la aplicación

Puesto que este proyecto es un juego, cuenta con una escena especializada que se encarga de enseñar al usuario las bases del juego mediante varias etapas en las

que aprenderá los controles básicos, como interactuar con los objetos del juego y como debe reaccionar a ciertas mecánicas que se encontrarán a lo largo del juego.

Conclusiones

Comparación

El resultado final del proyecto ha divagado bastante de la idea original, por lo que aprovecho para hacer una reflexión personal sombre mi proyecto, en el sentido de que empezó siendo un proyecto ambicioso en el que las mecánicas resultaban atractivas al jugador, eran fluidas y no interrumpían el flujo del juego ni resultaban molestas o contraproducentes. Esto, en la entrega sigue siendo uno de los puntos fuertes del juego, pero no han sido tan brillantes como las expectativas iniciales. Sí, el proyecto cuenta con varias mecánicas que hacen que el jugador pueda moverse a sus anchas sin importar los obstáculos, existe un sistema de combate sencillo y cómodo y un sistema de personalización de inventario muy simple pero robusto. Esto es algo que se pretendía tener a niveles superiores a los que se ha acabado obteniendo y aunque sea el producto mínimo viable, yo como su desarrollador, no me encuentro satisfecho con el proyecto. Ha llegado a un punto en el que el juego deja de obtener inspiración o mecánicas de Neon White el cual empezó siendo una clara inspiración para las mecánicas del juego acabó siendo olvidado puesto a la dificultad que presentaban las mecánicas para encajar con el estilo y debido a la alta dificultad de implementación. Pero, sobre todo, no estoy satisfecho con como se ve el proyecto, la resolución no es la adecuada y produce distorsiones en objetos concretos del juego.

Pero no todo lo que concluyo de este proyecto es algo negativo, durante el desarrollo de este proyecto he usado estrategias de desarrollo que no había usado anteriormente, como bien son las máquinas de estado, también he aprendido a dar uso a nodos de Godot que no había usado anteriormente como el *Animation Player*, que de primeras es un nodo muy sencillo que permite la reproducción de

animaciones haciendo uso de un *sprite*, o los *Audio Streams* para la reproducción de audio según necesidad o el *NinePatchRect* para que no se distorsionen los s*prites* cuando cambia su tamaño en la pantalla.

Mejoras a futuro

Como mejoras a futuro, propongo las siguientes:

- Cambios en la resolución del proyecto.
- Mejorar el sistema de combate, añadiendo más diversidad y nuevas armas.
- Mejorar los sonidos.
- Mejorar la IA de los enemigos. Actualmente creo que son muy sencillos y ciertos aspectos en su programación agradecerían un cambio.
- Diversificar más los biomas, añadiendo enemigos mucho más diferenciados. Así como una construcción de escenarios separados y más extensos.
- Expandir la trama para que sea más atractiva al jugador.
- Modificar la UI para hacerla más atractiva y menos monocromática.

Como un pequeño detalle adicional y por fanatismo personal hacia los juegos de peleas, me gustaría aprovechar todos los assets de personajes para desarrollar un modo multijugador que enfrente a dos personas similar a juegos como *Super Smash Bros* o *Rivals of Aether*.

Figura 16: Super Smash Bros Ultimate, Nintendo Official Website

Figura 17: Rivals Of Aether, Rivals of Aether Official Website

Una vez dicho lo anterior me gustaría concluir el apartado de conclusión diciendo que, pese a la situación, el plazo de desarrollo y a los requisitos no estoy completamente satisfecho con el resultado del proyecto. Pienso que cumple todos los requisitos y que es precisamente eso lo que hace que no esté contento, soy completamente consciente de que el fin de este proyecto no es desarrollar juegos AAA, pero debido a lo exigente y a los altos estándares que tengo cuando desarrollo proyectos, no considero este proyecto como uno de los mejores que haya hecho en temas de resultado, pero definitivamente, sí es uno de los más completos y entretenidos que he desarrollado en estos dos años.

Índice de tablas e imágenes

Índice de imágenes

Índice de tablas

Bibliografía y Referencias

Bibliografía

***Godot Engine Documentation*.** Documentación oficial del motor Godot.
Disponible en: https://docs.godotengine.org/en/stable/

***Microsoft Learn – ASP.NET Core*.** Documentación oficial del framework .NET utilizado en la API del proyecto.
Disponible en: https://learn.microsoft.com/es-es/aspnet/core

***MongoDB Manual*.** Manual oficial de uso y administración de MongoDB.
Disponible en: https://www.mongodb.com/docs/manual

***Dialogic Plugin (Godot)*.** Herramienta de código abierto para gestionar diálogos ramificados.
Disponible en: https://github.com/coppolaemilio/dialogic

Beehave Plugin (Godot). Plugin para la implementación de árboles de comportamiento en NPCs.
Disponible en: https://github.com/bitbrain/beehave

Referencias y créditos

Assets visuales

Disponible en: https://penusbmic.itch.io

Licencia: Uso permitido con atribución en proyectos no comerciales.

Fuente: Patreon y itch.io del autor.

Suno AI

Herramienta de generación de música asistida por inteligencia artificial.

Uso personal y no comercial autorizado por los términos del servicio.

https://suno.ai

GameBurp

Banco de sonidos con licencia comercial.

Los efectos utilizados en este proyecto están cubiertos por dicha licencia.

Concretamente aquellos pertenecientes a *2000 game sounds*.

https://gameburp.com

Freesound.org

Algunos sonidos están cubiertos por licencias **Creative Commons (CC)**:

- CC0 (uso libre sin atribución)
- CC-BY (requiere atribución)
- Todos los sonidos se han utilizado respetando su licencia original.

https://freesound.org

GameDev Market

Algunos de los assets utilizados en este proyecto están cubiertos por la Licencia

Comercial estándar (A) para compras realizadas después del 15 de enero de 2019. Esta licencia permite:

- El uso de los assets en proyectos personales y comerciales.
- La creación de productos derivados.
- La distribución y comercialización del producto final (*el juego*) sin límite de copias ni plataformas.

https://www.gamedevmarket.net/terms-conditions/

Extracto legal: Licence (A) - Effective 15 January 2019

Repositorio Del Proyecto

https://github.com/rmungar/TFG.git

Repositorio De La API

https://github.com/rmungar/TfgAPI.git