Escola Tècnica Superior d'Enginyeria ETSE-UV

Grado en Ingeniería Multimedia

Universitat de València

Trabajo de Fin de Grado

Módulo de creación automática de finales personalizados

AUTOR:
PABLO ALBERT PUCHOL

TUTORÍA:
MIGUEL LOZANO IBÁÑEZ

JULIO, 2023

VNIVERSITAT DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria ETSE-UV

Trabajo Fin de Grado

Módulo de creación automática de finales personalizados

AUTOR: PABLO ALBERT PUCHOL

TUTOR: MIGUEL LOZANO IBÁÑEZ

TRIBUNAL

PRESIDENTE/A:

VOCAL 1:

VOCAL 2:

FECHA DE DEFENSA:

CALIFICACIÓN:

Declaración de autoría:

Yo, Pablo Albert Puchol, declaro la autoría del Trabajo Fin de Grado titulado "Módulo de creación automática de finales personalizados de monitorización de usuario aplicado a la creación del nivel final de un videojuego 2D" y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual. El material no original que figura en este trabajo ha sido atribuido a sus legítimos autores.

Valencia, 13 de septiembre de 2024

Fdo: Pablo Albert Puchol

Resumen:

Este proyecto trata de abordar la creación de niveles de los videojuegos en dos dimensiones basada en las acciones que realiza el jugador a lo largo de la partida. Para su desarrollo, se hará uso del software de Unity con el fin de generar la prueba de concepto, un escenario 2D donde se podrán realizar las pruebas y se demostrará el funcionamiento del objeto que monitorizará al usuario.
El objetivo último es generar un módulo que pueda calcular el camino que recorrerá nuestra historia, a partir de los elementos que sean considerados necesarios para la descripción de la actitud de nuestro usuario y esto condicione el nivel final que le tocaría a nuestro jugador.

Abstract:

This project tries to address the creation of two-dimensional video game levels based on the actions that the player performs throughout the game.
For its development, Unity software will be used in order to generate the proof of concept,which is a 2D scenario where the tests can be carried out and the operation of the object that will monitor the user will be demonstrated.
The ultimate goal is to generate a module that can calculate the path that our story will follow, based on the elements that are considered necessary for the description of the attitude of our user and this conditions the final level that our player would have.

Agradecimientos:

A mi tutor, por ofrecerme la apoyarme en el desarollo de este proyecto.
Al personal del departamento de informática y del IRTIC por colaborar en el apartado de estimaciones y poder aportar su conocimiento como expertos.
Finalmente, a mi familia y amigos, por haberme apoyado durante mi trayecto universitario.

Índice general

Índice de figuras

Índice de ecuaciones

Índice de cuadros

Índice de códigos

Capítulo 1

Introducción

1.1. Contexto

A lo largo de la historia, ha habido cierta confusión respecto a la narrativa en el ámbito de los videojuegos, a menudo relacionándola con términos como historia, trama o guión. Aunque estos términos son populares en diferentes relatos, son pocas las personas que realmente arrojan luz sobre el tema y evitan este desconocimiento.

A partir de las conclusiones presentadas en el artículo de David Oña [9], podemos definir la narrativa como una forma de expresión discursiva que puede manifestarse a través de diversos medios y que requiere una serie de eventos que ocurren en un espacio temporal.

Siguiendo esta idea, podemos afirmar que todos los elementos presentes en un videojuego transmiten una historia a través de su uso, incluso las imágenes por sí solas pueden presentar una secuencia coherente de eventos.

Según Víctor Navarro, profesor e investigador en el Tecnocampus, un videojuego se divide en dos sistemas: el sistema central y el sistema exterior. El sistema exterior se refiere al contexto del jugador y los dispositivos de interacción, mientras que el sistema central está compuesto por las mecánicas, los patrones de acción, las reglas, los obstáculos, los personajes, los objetivos y otros elementos propios del videojuego. Por lo tanto, gran parte del potencial narrativo se encuentra en el sistema central, y la estructuración y organización de estos elementos constituye la narrativa del videojuego.

En este proyecto, se pretende desarrollar un módulo sencillo capaz de capturar las acciones realizadas por el jugador durante el transcurso del juego, guardar esta información en una base de datos y utilizarla para modificar la narrativa de la última parte de la historia o el último nivel, de manera que esté vinculada a la jugabilidad del usuario.

Con este fin, se creará un videojuego en 2 dimensiones, donde el jugador avanzará de nivel interactuando con diferentes elementos del entorno, mientras se registra su actividad.

La importancia de este módulo radica en su capacidad para desarrollar la narrativa de la última parte de la historia o los últimos niveles del juego. Sin este módulo, nuestra historia sería bastante simplificada y el juego terminaría abruptamente. Sin embargo, al utilizar el módulo, podemos establecer un punto en la historia en el cual, en función de la lista de acciones realizadas por el jugador hasta ese momento, se generará automáticamente un nivel acorde con su actitud de juego.

Este entorno forma parte de lo que se conoce como "Prueba de concepto", que se realiza para verificar que el concepto o teoría, en este caso, nuestro módulo, puede generar niveles de forma adecuada. Esto se logra mediante la creación de recursos como enemigos, coleccionables o secretos, que permiten definir la experiencia del usuario.

1.2. Objetivos

El propósito general del proyecto es generar la estructura del nivel final mediante la monitorización de las acciones del usuario. Sin embargo, a partir de este propósito surgen otras necesidades adicionales que deben cumplirse para lograr el objetivo completo. Estos subobjetivos incluyen:

- Creación de los modelos 2D y animaciones
  - Buscar referencias para los modelos de los personajes, enemigos y los jefes que aparecerán por el nuestros escenarios.
  - Diseñar el aspecto de los personajes, enemigos y los jefes en base a las referencias encontradas y seleccionadas.
  - Realizar los modelos 2D para los personajes, enemigos y los jefes en base a una referencia concreta.
  - Modelar los recursos relacionados con los personajes, enemigos y los jefes en base al diseño previo.
  - Establecer el estilo visual de las animaciones.
  - Definir las diversas animaciones que tendrán cada uno de los objetos y los actores presentes en el videojuego.
- Creación de los efectos de sonido y vídeo
  - Recompilar paginas web, que publiquen varios recursos musicales sin derechos de autor.

- Recopilar los sonidos base que representen las diferentes acciones del juego o el sonido de fondo.
- Editar y manipular los sonidos base para que se ajusten a las acciones y eventos del juego.
- Sincronizar los efectos de sonido con las acciones del juego para que coincidan adecuadamente con la jugabilidad y la experiencia de los jugadores.
- Desarrollar un guión o esquema detallado que describa la secuencia de escenas y eventos en el vídeo.
- Crear diseños conceptuales y *storyboards* que representen visualmente cada escena del vídeo.
- Animar y editar las escenas que forman parte de los vídeos finales del juego.

- Diseño de interacciones
  - Identificar los datos relevantes del usuario, que mostraran el perfil del usuario y que se guardaran en nuestro sistema.
  - Definir los puntos de interacción, los cuales, al ser desarrollado para PC, estarán principalmente relacionados con eventos de teclado y ratón, y muchos de ellos estarán ligados o influenciados por dichos dispositivos.
  - Establecer la lógica de captura de las interacciones que realiza el jugador mientras esta jugando.
  - Implementar el módulo de toma de decisiones para el nivel final.

- Base de datos
  - Diseño del esquema de la base de datos, en función de las interacciones consideradas relevantes del usuario.
  - Creación de la estructura de la base de datos en un tipo de formato de guardado.
  - Implementar las funciones del motor de videojuegos para consultar y guardar información en la base de datos.

Capítulo 2

Estado del Arte

2.1. Introducción

A lo largo de este capítulo, exploraremos las diversas tipologías de creación de historias para videojuegos y otros elementos audiovisuales que se pueden relacionar de forma simbiótica con nuestra temática. Además, se revisarán las metodologías utilizadas para clasificar a los diferentes usuarios en el contexto de los videojuegos, así como las tecnologías más adecuadas para llevar a cabo el proyecto de manera eficiente, tanto en términos temporales como económicos. También se llevará a cabo una búsqueda exhaustiva de antecedentes de videojuegos que empleen métodos similares a los presentados en este proyecto, o que puedan mejorar dichas funcionalidades.

En esta sección, nos adentraremos en el marco teórico general que fundamenta las ideas de nuestro proyecto. Se expondrán los métodos sobre los cuales se basan nuestras propuestas, y se realizarán comparaciones entre las tecnologías de software recomendadas, en función de los beneficios que cada una aporte en términos de cantidad y calidad. Además, se analizarán los antecedentes existentes, que nos servirán como referentes y competidores de los cuales aprender o llegar a superar.

El objetivo de este capítulo es revisar las bases teóricas y tecnológicas necesarias para el desarrollo y comprensión de nuestro proyecto, así como identificar las oportunidades y desafíos que enfrentaremos en el camino. A través de un análisis exhaustivo del estado actual del arte en la industria de los videojuegos, buscamos construir sobre los cimientos existentes y aportar otro punto de vista en nuestra propuesta.

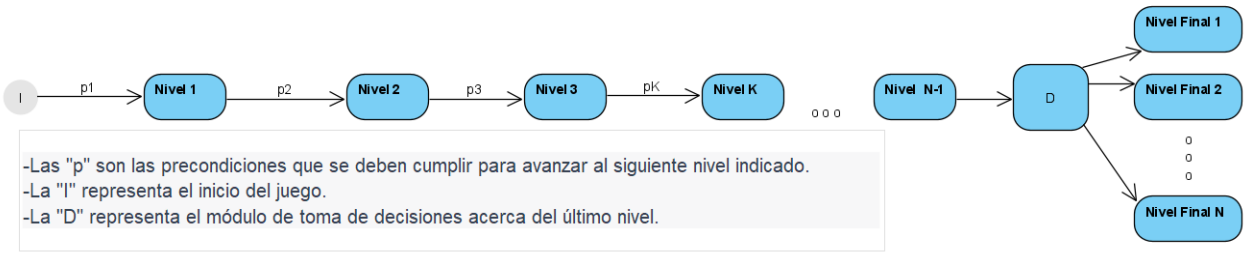

Figura 2.1: Esquema de funcionamiento del módulo

En esta figura se muestra el flujo de comportamiento de nuestro juego y cómo el uso del módulo afectaría la última condición para seleccionar el final que correspondería al jugador. Se puede observar que existe una condición para pasar de un nivel a otro, pero al final no se basará en una condición predefinida por el programador, sino en la toma de decisiones de nuestro módulo.

2.2. Storytelling

El *storytelling* o arte de contar historias, se define como una narrativa çautivadora"que transmite un mensaje o enseñanza. Esta técnica, utilizada en diversas formas de entretenimiento, busca generar un impacto emocional en el público, generándoles una conexión profunda con la obra.

Desde hace mucho tiempo, los profesionales del arte narrativo han reconocido la importancia de transmitir valores y lograr que la audiencia se identifique con la historia. La habilidad de contar una buena historia se basa en permitir que el público comprenda el mensaje, establecer una conexión emocional y mostrar el avance y superación de los personajes.

Para contar una historia efectiva, se considera esencial abordar dos aspectos principales: el plano racional, que apela a los procesos lógicos de las personas, y el plano instintivo, que se enfoca en lo emocional. Además, se deben tener en cuenta otros detalles, como se menciona en el artículo de *Planning Formalisms and Authoring in Interactive Storytelling* [5], como el uso de acciones que puedan generar fallos o errores en los personajes para añadir variedad y despertar la curiosidad del público.

En el contexto de los videojuegos, el storytelling se refiere a la forma en que se presenta y desarrolla la historia dentro del juego. A través de narrativas, personajes, diálogos, eventos y otros elementos, se busca crear una experiencia inmersiva y emocionante para los jugadores. También se busca proporcionar un sentido de logro al completar objetivos

o alcanzar desenlaces específicos, lo que cautiva y mantiene a las personas comprometidas con el juego.

El conocimiento del storytelling es altamente aplicable al módulo desarrollado en este proyecto, ya que hay una estrecha relación entre el control de las acciones de los personajes para generar un final cautivador y lo mencionado en el artículo de *Planning Formalisms and Authoring in Interactive Storytelling* [5] sobre el uso de la inteligencia artificial (IA) para generar y controlar los eventos narrativos de manera agradable para la audiencia.

El storytelling en los videojuegos puede adoptar diversas formas, desde escenas cinemáticas pregrabadas hasta diálogos interactivos dentro del juego. También puede variar en complejidad, desde historias lineales y simples hasta tramas ramificadas con múltiples finales alternativos. Para facilitar y optimizar la creación del guión del juego y permitir esta atractiva complejidad, se recomienda que la monitorización del jugador se centre en las principales características del personaje o en las interacciones clave a lo largo del escenario. Esto permitirá crear una historia dinámica basada en el carácter mostrado por el personaje que el jugador controla a lo largo de la 'historia' del videojuego.

2.3. Caracterización del usuario

En esta sección se revisan dos trabajos que abordan los puntos clave mencionados en el punto 2.2 y que podrían servir de apoyo para la implementación de la inteligencia artificial en la generación eficiente de los niveles finales de un videojuego, teniendo en cuenta el carácter de los usuarios.

Además, según la información en del articulo de la pagina web Playmotiv [10], se han propuesto numerosos métodos para crear una teoría general de clasificación de los usuarios, investigando sus objetivos, su relación con el juego y su interacción con otros jugadores. Sin embargo, muy pocos de estos métodos han tenido éxito. El problema radica en que los jugadores no se limitan a un solo tipo de juego debido a la variedad y complejidad de las opciones disponibles, lo que implica que su forma de jugar puede variar considerablemente según el contexto.

Entre todas las investigaciones que han intentado abordar este problema, destacan los dos métodos más extendidos en los que se basa el algoritmo de creación del presente proyecto. Estos métodos son:

2.3.1. La taxonomía de Bartle [1]

Fue creado en los años 2000 por el inglés Richard Bartle y se basa en la clasificación de los jugadores en función de dos ejes: sus preferencias en cuanto a la interacción con el juego o con otros jugadores, y su preferencia por la acción o la interacción.

Triunfadores: son aquellos jugadores que prefieren interactuar con el mundo del juego y centrarse en la acción. Buscan objetivos que superar y logros que alcanzar. En la vida real, suelen ser personas que buscan destacar, enfocadas en el éxito y que trabajan de forma independiente. Sin embargo, existen excepciones, por lo que se introdujo un tercer eje de motivación. Los Triunfadores pueden clasificarse en la subclase de "Oportunista", aquellos que solo buscan su propio beneficio, y la subclase de "Planificador", aquellos que se suman a los objetivos comunes de un grupo.

Exploradores: son jugadores que disfrutan interactuando con el mundo del juego, pero no les atrae tanto la acción como la interacción con el entorno. Les gusta explorar el mundo virtual de forma individual, buscando secretos, zonas inexploradas y áreas inalcanzables hasta ese momento. Su objetivo principal es mejorar y aprender constantemente. Dependiendo de cómo lo hagan, también pueden dividirse en la subclase de "Científico", aquellos que siguen todas las reglas del juego en sus pruebas, y la subclase de "Hacker", aquellos que intentan romper los límites de las mecánicas del juego y explorar más allá de las normas establecidas.

Socializadores: son jugadores que disfrutan de la interacción con otros miembros de la comunidad y no tanto del juego en sí. Son buenos trabajando en equipo y le dan más importancia a la interacción para crear relaciones y vínculos con sus compañeros. También pueden clasificarse en las subclases de "Networker", aquellos que intercambian conocimiento explícito para aumentar su red de información, y "Amigo", aquellos que intercambian información implícita para socializar con otros.

Asesinos: son jugadores que disfrutan de la interacción con otros miembros de la comunidad, pero buscan más la competencia que la colaboración. Dependiendo de su objetivo, pueden clasificarse en la subclase de "Político", aquellos que acumulan logros para destacar por encima de los demás, y la subclase de "Griefer", aquellos capaces de dañar a otros jugadores para obtener ventajas.

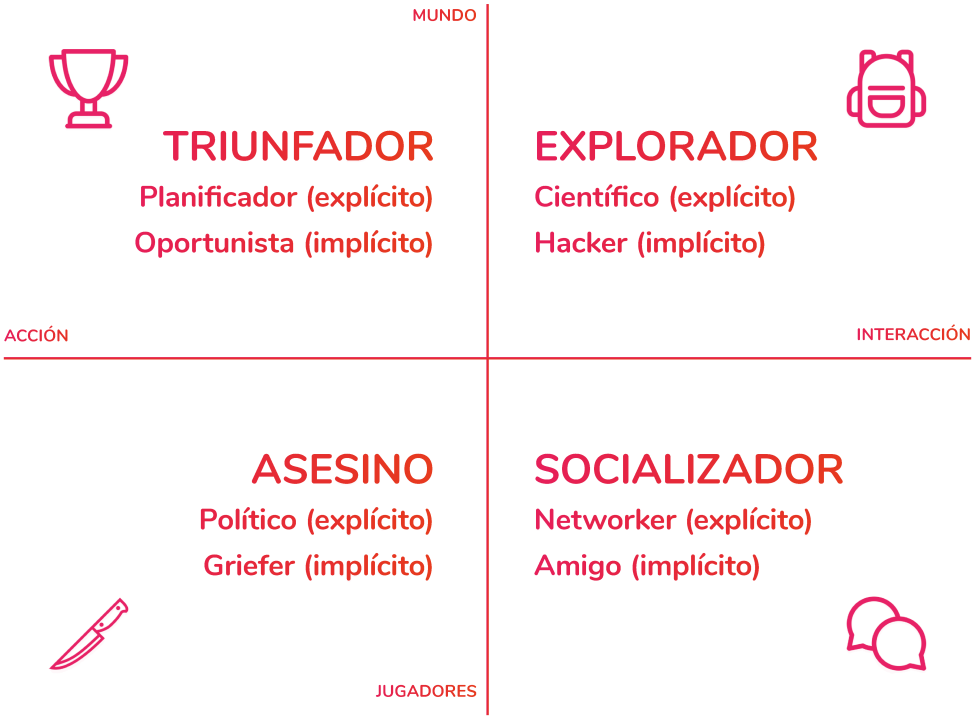

Figura 2.2: Gráfica de taxonomía de Bartle donde se representan los diferentes tipos de usuarios según su comportamiento

Un ejemplo concreto en el que se utilizó este método como base fue un proyecto en el cual se implementó una clasificación automática de los jugadores en los equipos de salas de escape, basada en el estilo de juego de los usuarios [12].

2.3.2. La clasificación de Andrzej Marczewski [7]

Fue establecida por Andrzej Marczewski como una clasificación más enfocada en los objetivos de cada perfil de jugador, en lugar de su relación con otros jugadores o el juego en sí. Se basa en una clasificación a partir de cuatro ejes, lo que da lugar a seis tipos de jugadores diferentes.

Los Socializadores están motivados por la relación y buscan interactuar con otros jugadores para crear conexiones sociales.

Los Espíritus Libres están motivados por la autonomía y disfrutan de la exploración y la creatividad en el juego.

Los Triunfadores están motivados por la maestría y la superación. Buscan aprender

cosas nuevas y mejorar mientras enfrentan desafíos y retos en el juego.

Los Filántropos están motivados por el propósito y el significado. Este grupo es altruista, se preocupa por la comunidad y busca hacer del juego una experiencia enriquecedora.

Los Jugadores están motivados por las recompensas y beneficios propios. Están dispuestos a hacer cualquier cosa para obtener las recompensas que el juego ofrece.

Los Disruptores están motivados por el cambio y, a menudo, buscan perturbar el sistema de juego, ya sea directamente o a través de la interacción con otros jugadores, con el objetivo de provocar un cambio positivo o negativo. Estos cambios pueden incluso desafiar las normas o reglas establecidas en los juegos.

A partir de la información ya mostrada de estas dos taxonomías, se pudo aclarar que la cantidad de parámetros necesaria para la clase de usuarios que se describe en la segunda está más orientada a los juegos en línea, por lo que al crear un videojuego offline donde importa la interacción del jugador con los diferentes elementos del sistema, se elegirá la primara investigación como base para seleccionar los atributos clave sobre los que dependerá la creación del ultimo escenario.

Aun teniendo tantas clases de jugadores, el proyecto se centrará en las cuatro tipos principales de usuario de la taxonomía de Bartle y para que el modulo cree cada uno de los finales relacionados con cada una de las ramas de esta teoría, el modulo se basará en estas variables que se presentan a continuación sobre nuestro escenario de prueba:

Variables que determinarán la selección del final asesino, en el cual el jugador busca ser el más fuerte matando enemigos:

- Número de enemigos eliminados por el jugador.
- Lista de los diferentes niveles por los cuales ha pasado el jugador, necesaria para el recuento de enemigos con los que el jugador ha evitado pelear.
- Cantidad de enemigos presentes en cada nivel.

Variables que determinarán la selección del final triunfador, en el cual el jugador busca obtener el premio al ser el más veloz:

- Tiempo transcurrido dentro de la mazmorra, excluyendo pausas y tiempo dedicado a seleccionar niveles.
- Promedio de tiempo requerido para completar el juego.

Variables que determinarán la selección del final de socializador, en el cual el jugador busca la interacción con los demás personajes del videojuego:

- Número de personajes no jugables (NPCs) con los que el jugador ha interactuado mediante diálogos.
- Lista de los diferentes niveles por los cuales ha pasado el jugador, necesaria para el recuento de NPCs con los que podría haber dialogado el jugador.
- Cantidad de NPCs presentes en cada nivel.

Variables que determinarán la selección del final de explorador, en el cual el jugador quiere buscar en todos los rincones de la mazmorra:

- Distancia total recorrida por el jugador dentro de la mazmorra, teniendo en cuenta que se considera incluso cuando el jugador repite el mismo recorrido.
- Dimensiones de cada nivel en términos de tamaño y extensión.

2.4. Tecnologías Audiovisuales

En esta sección, se aborda la justificación del uso de las tecnologías seleccionadas en el proyecto, a través de la muestra y comparación de diversas tecnologías utilizadas en proyectos similares, así como aquellas presentadas durante el desarrollo académico.

La comparación de los diferentes tipos de tecnologías se realizará mediante tablas comparativas que resaltarán los beneficios y ventajas que ofrecen en relación a los aspectos relevantes para lograr una mayor eficiencia temporal y distribución de recursos.

La metodología empleada para la selección de una tecnología sobre otras se basará principalmente en la cantidad de ventajas que ofrezca en comparación con el resto. En caso de que la comparación arroje resultados similares, se dará prioridad a la importancia de las ventajas dentro del conjunto comparado.

2.4.1. Modelado 2D

En este subapartado, se realiza una comparativa entre los programas de creación de sprites [1] en 2D que ofrecen una amplia gama de herramientas de dibujo, recursos visuales o tutoriales que explican cómo lograr diferentes estilos utilizando la aplicación, así como aquellos que permiten reducir el tiempo necesario para obtener resultados satisfactorios en una "Prueba de concepto".

Se han incluido algunos de los programas más reconocidos y apreciados por la comunidad de artistas digitales en esta área. A continuación, se presentan los software de modelado seleccionados tras el análisis y evaluación correspondiente:

	BRL-CAD	Photoshop	Photopea
Interfaz de usuario	**Regular**	**Bueno**	**Excelente**
Herramientas de dibujo	**Malo**	**Excelente**	**Excelente**
Capacidad de edición	**Malo**	**Bueno**	**Bueno**
Compatibilidad con formatos	**Bueno**	**Excelente**	**Excelente**
Innovación	**Malo**	**Excelente**	**Bueno**
Comunidad y soporte	**Regular**	**Bueno**	**Excelente**
Integración con otros sistemas	**Regular**	**Excelente**	**Bueno**
Accesibilidad	**Regular**	**Bueno**	**Excelente**
Precio	**Bueno**	**Regular**	**Excelente**

Cuadro 2.1: Comparación de los diferentes software de modelado de recursos.

La elección definitiva de Photopea[4] se basa en la simplicidad de su interfaz de usuario y sus herramientas de dibujo y edición, que, aunque comparten similitudes con las de Photoshop, no están tan sobrecargadas y permiten realizar las acciones necesarias para crear el escenario.

Además, Photopea nos brinda una alternativa sólida que nos permite acceder a servicios a largo plazo. Cuenta con un sólido soporte distribuido a nivel global, ofrece diversas formas de acceso para los usuarios y tiene un precio bastante asequible.

2.4.2. Animación 2D

En este subapartado, se realiza una comparación entre programas de animación en dos dimensiones que permiten crear las animaciones correspondientes a los diferentes finales a los que el jugador puede llegar. Estos programas cuentan con herramientas que facilitan la tarea y ofrecen la posibilidad de añadir efectos especiales simples que se integren con las tabletas gráficas.

[1] Conjunto de imágenes que representa un personaje u objeto (o una parte de ellos) de manera gráfica

Entre los programas más utilizados, destacan aquellos que tienen un nivel de aprendizaje sencillo y ofrecen una amplia variedad de efectos para la creación de las animaciones. A continuación, se presentan los siguientes programas seleccionados para su comparación:

	Adobe Animate CC	OpenToonz
Funciones de animación	**Bueno**	**Regular**
Interfaz de usuario	**Regular**	**Bueno**
Herramientas de dibujo	**Bueno**	**Excelente**
Funciones de edición	**Excelente**	**Bueno**
Capacidad de importación y exportación	**Excelente**	**Excelente**
Biblioteca de recursos	**Bueno**	**Excelente**
Soporte para audio	**Bueno**	**Bueno**
Soporte de hardware	**Excelente**	**Excelente**
Precio	**Regular**	**Excelente**

Cuadro 2.2: Comparación de los diferentes software de animación en 2D.

La elección final de OpenToonz[8] se debe a la simplicidad de sus herramientas de dibujo y edición. Aunque presentan cierta similitud con otras aplicaciones, no están sobrecargadas y permiten crear videos finales y animaciones de personajes, con la posibilidad de exportarlos en los formatos más comunes en la industria de los videojuegos.

Por otro lado, OpenToonz ofrece una interfaz que aunque parece al inicio poco intuitiva para principiantes, es fácil de aprender en poco tiempo. También cuenta con una sólida base de soporte a través de comunidades distribuidas globalmente y una amplia selección de bibliotecas. El software ofrece un buen soporte para dispositivos de audio y hardware, y todo ello a un precio bastante accesible.

2.4.3. Editor Sonido

En este subapartado, se comparan los programas de edición de sonido que permiten modificar la música y los sonidos que se utilizarán como banda sonora o efectos de sonido en momentos específicos dentro de la interfaz o el escenario del juego.

Se han seleccionado los siguientes programas, que son ampliamente utilizados y cuentan con una metodología de aprendizaje ágil, para llevar a cabo la comparación:

	Adobe audition	Audacity
Interfaz de usuario	**Bueno**	**Excelente**
Funciones de edición	**Excelente**	**Bueno**
Efectos de audio	**Excelente**	**Excelente**
Soporte de formato	**Excelente**	**Excelente**
Control de nivel	**Excelente**	**Bueno**
Soporte de plug-ins	**Bueno**	**Bueno**
Automatización	**Excelente**	**Regular**
Exportación de audio	**Bueno**	**Excelente**
Soporte de múltiples pistas	**Excelente**	**Excelente**
Precio	**Regular**	**Excelente**

Cuadro 2.3: Comparación de los diferentes software de edición de audio.

La elección final de Audacity[6] se debe a la simplicidad de su interfaz de usuario y a las herramientas de edición de sonido que ofrece. Aunque no cuenta con tantas opciones como Adobe, contiene funciones y herramientas de edición bien organizadas y fáciles de entender, lo que lo convierte en una excelente opción para usuarios nuevos. Con este software, es posible editar fácilmente los sonidos del juego, ya sea para adaptarlos a una banda sonora sencilla o para crear efectos de sonido que mejoren la experiencia inmersiva del usuario.

Audacity también ofrece un control de nivel similar al programa de Adobe, lo que facilita su manejo e intuitividad. También permite trabajar con múltiples pistas, lo cual es útil para la creación de una banda sonora compleja. Aunque no cuenta con tantas características automatizadas para la edición de sonido como otros programas, su precio reducido compensa esta limitación de manera significativa.

2.4.4. Motores gráficos 2D/3D

En este subapartado, se comparan los motores gráficos disponibles para la creación de videojuegos en 2D. Se busca evaluar su disponibilidad de recursos, tanto en sus páginas oficiales como en el contenido compartido por la comunidad de diseño gráfico. Además, se considera la variedad de herramientas que ofrecen, buscando aquellas que sean fáciles de utilizar y que faciliten los cálculos relacionados con las mecánicas del juego y el módulo.

Se han seleccionado los siguientes motores, los cuales son ampliamente utilizados y brindan soporte para la creación de juegos en dos dimensiones:

	Unity	Unreal Engine 5
Rendimiento	**Excelente**	**Buena**
Herramientas de desarrollo	**Excelente**	**Regular**
Soporte multiplataforma	**Excelente**	**Bueno**
Gráficos	**Bueno**	**Excelente**
Física	**Excelente**	**Bueno**
Audio	**Bueno**	**Bueno**
Comunidad y soporte	**Excelente**	**Bueno**
Integración	**Bueno**	**Excelente**
Actualizaciones regulares	**Excelente**	**Bueno**
Precio	**Excelente**	**Excelente**

Cuadro 2.4: Comparación de los diferentes motores gráficos del mercado.

La elección final de Unity[14] se debe principalmente a que es un motor gráfico orientado a la creación de videojuegos en dos dimensiones, lo que proporciona una amplia variedad de métodos y funciones de apoyo para la creación de contenidos dinámicos y realistas. Aunque no cuenta con tantas opciones como Unreal[3], Unity ofrece una amplia gama de funciones y herramientas de edición que permitirán que nuestro videojuego tenga un rendimiento óptimo.

Asimismo, Unity cuenta con un sólido respaldo por parte de su comunidad, se actualiza con frecuencia y ofrece una rentabilidad comparable a Unreal.

2.5. Antecedentes : juegos similares

En este apartado, se resalta la existencia previa de videojuegos que emplean mecánicas similares a las que se persiguen como objetivo final en el proyecto. Es importante llevar a cabo un seguimiento de los videojuegos más famosos que presentan similitudes con el resultado que deseamos obtener.

Además, se presentan otros motivos para llevar a cabo esta revisión de antecedentes, como son:

- Mostrar el funcionamiento de cada una de las mecánicas de selección de rutas a través de la interacción del jugador con el entorno virtual, que cambiará la narrativa del juego.

- Identificar mejoras en el módulo que no se encuentran en otros software del mercado, con el fin de lograr un mayor éxito comercial en caso de competir con un proyecto final.

- Aprender de los errores que se han presentado previamente en otros videojuegos, a fin

de ahorrar tiempo y dinero que de otro modo se habrían invertido innecesariamente en corregir el código.

2.5.1. Undertale [11]

Undertale es conocido por su sistema de elección y sus múltiples rutas de juego que afectan la historia y el resultado del juego. La selección de las rutas predefinidas del juego se basa en las decisiones y acciones del jugador a lo largo de su partida.

Las principales rutas que existen son:

Ruta Pacifista: La Ruta Pacifista es considerada la ruta "buena" del juego y para acceder a esta ruta, el jugador debe evitar matar a ningún enemigo y resolver conflictos a través del dialogo. Lo que se consigue al seguir esta ruta es que se realicen cambios significativos en la historia, donde ninguno de los enemigos muere a manos del protagonista y a medida que va avanzando el jugador deberá de ir completando de forma gradual los desafíos secundarios que permiten completar a su vez los desafíos pacifistas.

Ruta Neutral: La Ruta Neutral es la ruta predeterminada del juego y se activa si el jugador ha realizado alguna acción agresiva o no ha alcanzado los requisitos mínimos para la Ruta Pacifista. En esta ruta, el jugador tiene más libertad para decidir si quiere ser amigable o agresivo con los enemigos. Lo que se consigue al seguir esta ruta puede variar dependiendo de las elecciones del jugador, lo que llevará a diferentes finales.

Ruta Genocida: La Ruta Genocida es la ruta más desafiante y se activa cuando el jugador elige ser violento y derrotar a todos los enemigos en el juego. La variable que mide el juego para determinar que ruta debe de activar para el jugador, es la cantidad de muertes que ha realizado y su relación a una cantidad específica de enemigos en cada área. Lo que se consigue al seguir esta ruta, es que se realicen cambios significativos en la historia y los personajes, y que en un futuro el jugador deberá de enfrentará a desafíos más difíciles.

Es importante destacar que, aunque los finales o rutas alternativas no se desarrollen de manera dinámica, el cálculo utilizado para la selección de un camino u otro en la historia guarda similitudes con la parte mencionada en la taxonomía de Bartle 2.3, centrándose específicamente en la psicología del jugador asesino y del jugador socializador.

Figura 2.3: Interfaz de elecciones de *Undertale*.

2.5.2. Heavy Rain [16]

En el videojuego Heavy Rain, la selección de rutas de juego se basa en las decisiones y acciones del jugador a lo largo de la historia. El juego presenta múltiples personajes y una trama ramificada, lo que significa que las elecciones del jugador pueden tener un impacto significativo en el desarrollo de la historia y el destino de los personajes. Al tener una gran diversidad de finales, nos centraremos más en los métodos y variables de selección que proporciona el juego:

Elecciones y acciones: Durante el juego, el jugador asume el control de varios personajes, en diferentes situaciones donde debe de realizar una elección que pueden llegar a ser morales, éticas o tácticas. Cada una de las acciones de los protagonistas pueden tener consecuencias importantes en el desarrollo de la trama, ya que hay decisiones que pueden influir en la vida o muerte de los personajes, la relación entre ellos y el curso general de la historia.

Ramificaciones de la historia: Se presentan múltiples ramificaciones y finales posibles predefinidos en el juego. Esto significa que las acciones y decisiones del jugador pueden llevar a diferentes eventos, escenas y resultados. Algunas elecciones pueden abrir nuevas pistas o revelar información adicional, mientras que otras pueden llevar a eventos dramáticos o giros inesperados en la historia.

Resultados y consecuencias: Cada acción y elección del jugador tiene un impacto en el desarrollo de la trama y el destino de los personajes. Algunas decisiones pueden conducir a situaciones favorables o resolver misterios, mientras que otras pueden tener resultados trágicos o complicar aún más la situación. El juego tiene múltiples finales que se desbloquean en función de las elecciones realizadas durante la partida por el jugador.

Es importante tener en cuenta que Heavy Rain es un juego de narrativa interactiva que limita la captura de las interacciones dinámicas del jugador y se centra más en la parte psicológica del jugador. Por lo tanto, si tomamos la taxonomía de Bartle 2.3 como referencia para la lectura de los puntos vitales necesarios para determinar la personalidad del usuario, nos encontramos con que los puntos para calcular el final socializador y el final de explorador se ven fuertemente restringidos por las reglas del juego. Esto resulta en una exploración reducida del escenario para el jugador o en la obligación de que la interacción del jugador con los personajes sea fiel a la trama del juego.

Figura 2.4: Interfaz de elecciones de*Heavy Rain*.

2.5.3. Until Dawn [13]

En el videojuego Until Dawn, la selección de rutas de juego se basa en las decisiones y acciones del jugador a lo largo de la historia. El juego presenta una trama interactiva de horror en la que los jugadores asumen el papel de varios personajes y deben tomar decisiones que influirán en el desarrollo de la historia y el destino de los personajes desde varias historias secundarias que ocurren en paralelo.

Al tener una gran diversidad de finales, nos centraremos más en los métodos y variables de selección que proporciona el juego:

Decisiones morales: A lo largo del juego, los jugadores se enfrentarán a decisiones morales difíciles que pueden tener consecuencias significativas. Estas elecciones pueden implicar salvar o no a un personaje durante una persecución, confiar o desconfiar de ciertos personajes, o elegir entre opciones que pueden afectar la vida o muerte de los protagonistas. Estas decisiones determinarán el curso de la historia y el destino de los personajes, afectando a su vez a la narrativa del videojuego.

Exploración y descubrimiento: Los jugadores pueden explorar el entorno del juego, descubrir pistas y tomar decisiones basadas en la información que recopilen. Al encontrar pistas y secretos ocultos, se pueden desbloquear nuevas opciones de diálogo , acciones y vídeos ilustrativos sobre eventos que pueden afectar enormemente a la historia.

Acciones rápidas y eventos de tiempo rápido: Durante ciertos momentos clave del juego, los jugadores deben realizar acciones rápidas o tomar decisiones en tiempo real que evitan que el jugador reflexiones sobre sus decisiones y realice la primera elección que pase por su mente. Estas acciones pueden incluir esquivar peligros, escapar de amenazas o realizar acciones precisas en momentos de tensión.

Efecto mariposa: es el concepto principal que se utiliza en el videojuego, donde incluso las decisiones aparentemente insignificantes pueden tener repercusiones significativas en la historia. Algunas elecciones tempranas pueden desencadenar eventos posteriores que afectarán la trama y las relaciones entre los personajes. Esto significa que todo tipo de elecciones realizadas durante el juego son clave para abrir nuevas ramificaciones en la historia y llevar a diferentes resultados.

Es importante destacar que Until Dawn presenta múltiples finales y variaciones en la historia, lo que brinda a los jugadores la oportunidad de experimentar diferentes rutas y desenlaces que dependen de las elecciones y acciones realizadas a lo largo del juego. Sin embargo, una desventaja que se observa en el videojuego es que las ramificaciones mostradas durante el juego están preestablecidas y almacenadas por los desarrolladores en el software, lo que limita el número de finales que un jugador puede descubrir.

Figura 2.5: Interfaz de elecciones de *Until Dawn*.

Capítulo 3

Especificación del proyecto

3.1. Introducción

En esta sección, presentamos una descripción detallada de nuestro proyecto, delineando sus estimaciones, alcance y requisitos principales. Aquí, exploraremos la esencia misma de nuestra propuesta revelando la visión y los resultados esperados que buscamos lograr.

El proyecto que presentamos es el resultado del análisis de las tendencias o demandas de los últimos años y la búsqueda de satisfacer una necesidad latente dentro de la comunidad de los videojuegos, aportando una solución sencilla y efectiva.

A lo largo de esta especificación, destacaremos los elementos clave que conforman nuestro proyecto, incluyendo los aspectos técnicos, funcionales y operativos. Además, se detallarán los recursos necesarios, el cronograma de actividades y los entregables esperados.

En resumen, esta especificación del proyecto es la piedra angular sobre la cual se construirá y ejecutara todo el desarrollo del mismo. Constituirá una base para una planificación sólida, una comunicación efectiva y la consecución exitosa de los resultados deseados y a medida que avancemos en esta especificación, el lector tendrá mas claro el conocimiento sobre nuestro proyecto.

3.2. Requisitos

Los requisitos están estrechamente ligados con los objetivos que se desean alcanzar a través del proyecto y mediante su clarificación establecemos una guía sólida que orientará

todas las etapas del proyecto.

En esta sección, se proporcionará una visión general de los requisitos funcionales y no funcionales en los que se pueden clasificar dependiendo en los atributos que se centren.

3.2.1. Requisitos Funcionales

Los requisitos funcionales son los encargados de especificar las funciones y características especificas que el sistema ha ser capaz de proporcionar. En el caso que se presenta, son los siguientes:

- Reproducir el escenario en 2D y que sea funcional en 3ª persona.
- Controlar el personaje con teclado y ratón.
- Crear al principio, un nivel donde aprender a controlar al personaje.
- Mover al jugador de sala en sala hasta llegar al final.
- Volver a empezar a jugar desde el inicio cuando el personaje llega al final.
- Configurar el volumen, brillo y otros aspectos visuales del juego de forma dinámica.
- Mostrar las posiciones de las teclas que controlan las diferentes acciones del personaje en el panel de ayuda.
- Autoguardado automático al transferirse de un nivel a otro.
- Construir puntos de guardado temporal por el mapa.
- Posibilidad de salir del juego en cualquier momento.
- El jugador puede realizar un movimiento rápido, saltar, escalar, correr y atacar.
- Enlazar los efectos de partículas y sonido con sus respectivas acciones o fondos de escenario.
- Permitir interactuar con personajes y con objetos a lo largo del mapa.
- Crear un espía que calcule el final.
- Crear una base de datos que almacene la información del personaje.

3.2.2. Requisitos No Funcionales

Estos requisitos complementan a los requisitos anteriores, centrándose en los atributos de calidad y aspectos técnicos, tales como rendimiento, seguridad y usabilidad. En el caso que nos ocupa, dichos requisitos se detallan a continuación:

- El personaje contará con 3 vidas, aunque puede recuperarlas por el camino.
- Hay enemigos con diferentes vidas y métodos de ataque.
- Hay jefes con diferentes vidas y métodos de ataque.
- El jugador podrá no atacar a los enemigos para poder pasarse un nivel, a no ser que sea la sala del jefe.
- El juego tendrá una animación sencilla de transición entre pantallas.
- El jugador puede regresar en cualquier momento al menú principal, pero perderá los datos a menos que los haya guardado previamente.
- El juego debe contar con una opción de adaptación para personas con daltonismo.
- El juego debe tener un aspecto visual atractivo tanto para adolescentes como para personas de mayor edad.
- No almacenará información personal del usuario pero si del personaje.
- La banda sonora será inmersiva y misteriosa.
- Se hará uso de los programas *Photopea* y *DragonBonesPro* para la creación de más personajes o enemigos.
- Se hará uso del programa *Audacity* para modificar audios.
- El motor gráfico con el que se desarrollará el escenario será *Unity*.
- El lenguaje de programación que se utilizará en los scripts[1] será *C#*.
- El programa deberá de funcionar para *Windows 10*.
- Debe de solo poder jugar un usuario.
- El jugador guarda en su base de datos los trofeos o logros que vaya consiguiendo.
- El jugador tiene una opción para consultar su colección de trofeos o logros.

[1]Conjunto de instrucciones o código de programación que se utiliza para controlar el comportamiento y la funcionalidad de un programa o aplicación

3.3. Especificación de las funcionalidades

En esta parte se completa la descripción de cómo va a operar nuestro proyecto. A partir de los requisitos funcionales que hemos establecido previamente, se detallará un listado más extenso de funcionalidades.

	Funcionalidaes
1º	El personaje podrá moverse en todas las direcciones
2º	El personaje podrá atacar, trepar paredes y saltar de forma progresiva
3º	Hay enemigos que se pueden matar de diferentes formas
4º	El jugador puede usar su cuerpo o su arma para derrotar a los enemigos
5º	El jugador puede recoger monedas y elixires por el mapa que pueden ayudarle en la aventura
6º	No se pueden guardar los objetos que no sean monedas en el personaje
7º	Un diseño de niveles que sea lineal y permita al jugador progresar utilizando diversas estrategias.
8º	El jugador puede viajar de un nivel a otro
9º	El ultimo escenario se formará a partir de las acciones del jugador
10º	Hay jefes cada cierto número de niveles
11º	Los jefes tienen patrones y parámetros diferentes que se ajustan a la temática del nivel en el que se encuentra el jugador
12º	Hay un desafió diferente al llegar al final de la mazmorra
13º	El jugador puede avanzar de forma continua por el mapa o usar los puntos de teletransporte para llegar a otro nivel o volver al principio del juego
14º	La historia del juego es muy simple
15º	Los NPC pueden dar cualquier tipo de información al jugador
16º	El modo de juego es el que decida el jugador mediante sus acciones
17º	Cada una de las acciones de los enemigos, NPCs y jugador tendrán un efecto de sonido, que informe más fácilmente del entorno al usuario.
18º	El jugador podrá modificar el nivel de volumen de música ambiente del juego
19º	El jugador podrá modificar el nivel de brillo del juego
20º	El jugador podrá modificar el nivel de volumen de los efectos del juego
21º	El jugador podrá modificar el tamaño de pantalla del videojuego
22º	El jugador podrá modificar el tipo de visión
23º	El progreso lo puede guardar al pasar de un nivel a otro y temporalmente en los checkpoints a lo largo del mapa
24º	El usuario puede conseguir trofeos de los diferentes finales
25º	La plataforma principal de juego será el ordenador
26º	El espía tendrá en cuenta los asesinatos del jugador
27º	El espía tendrá en cuenta las conversaciones del jugador
28º	El espía tendrá en cuenta el desplazamiento del jugador
29º	El espía tendrá en cuenta el tiempo de juego del jugador

Cuadro 3.1: Especificaciones del sistema.

3.4. Gestión de alcance

Este apartado tiene como objetivo definir y delimitar claramente los límites y metas del proyecto, garantizando así que se cumplan las expectativas de los interesados y se entreguen los resultados especificados.

Realizar una gestión del alcance implica analizar la temática que abarca el trabajo y determinar las actividades y tareas necesarias para alcanzar los resultados deseados, así como los recursos y el tiempo estimado para su realización.

Es importante tener en cuenta estos aspectos para poder trazar un proyecto sólido que no se desvíe durante su desarrollo. Siguiendo las tareas descritas en este apartado en orden, evitaremos olvidar algún punto importante y evitaremos problemas de dependencia en caso de tener una planificación incorrecta.

La gestión del alcance de este proyecto se basará en una Estructura de Descomposición

de Tareas (EDT) que proporcionará una representación gráfica de la descomposición jerárquica del proyecto, dividiéndolo en elementos más pequeños llamados paquetes de trabajo. Cada paquete de trabajo corresponderá a una tarea o actividad específica que contribuye al logro del objetivo general del proyecto.

La posición de una tarea en la lista determinará su nivel de detalle y alcance. Las tareas de mayor jerarquía tendrán descripciones más generales y abarcarán un mayor contenido temático, mientras que las tareas de menor nivel estarán más detalladas y presentarán una estructura más simple.

Nombres de las tareas	Entreg.
| 1 | Preparación	**E1**
| 1.1 | Búsqueda bibliográfica y webgrafía	
| 1.2 | Redacción de motivos y objetivos	
| 1.3 | Redacción del Estado del Arte	
| 1.4 | Definir requisitos del proyecto	
| 2 | Análisis	**E2**
| 2.1 | Crear Çasos de Uso"	
| 2.2 | Crear "Diagrama de Estados"	
| 2.3 | Crear "Diagrama de Actividad"	
| 2.4 | Crear "Modelo de guardado de datos"	
| 2.5 | Crear "Diagrama de Secuencia"	
| 2.6 | Crear "Diagrama de Clases"	
| 2.7 | Estimaciones temporales y económicas	
| 3 | Diseño	**E3**
| 3.1 | Diseño del juego	
| 3.1.1 | Sinopsis de la historia	
| 3.1.2 | Establecer el género del juego	
| 3.1.3 | Diseño de personajes [principal + 6 NPCs]	
| 3.1.4 | Diseño de enemigos [4 terrestres + 2 aéreos]	
| 3.1.5 | Diseño de jefes [2 tipos]	
| 3.1.6 | Diseño de recursos de los escenarios	
| 3.1.7 | Diseño de las interfaces [inicio + 2 tipos de juego + opciones + muerte]	
| 3.2 | Diseño de las mecánicas del juego	
| 3.2.1 | Mecanismo para abrir puertas	
| 3.2.2 | Mecanismo para desplazarse entre niveles	
| 3.2.3 | Mecanismo para almacenar recursos	
| 3.2.4 | Mecanismos de guardado	

Cuadro 3.2: Estructura de descomposicón de tareas (EDT) - Primera parte

Nombres de las tareas	Entreg.
| 4 | Implementación	**E4**
| 4.1 | Adquisición de música	
| 4.2 | Adquisición de efectos de sonido	
| 4.3 | Adquisición de elementos de UI	
| 4.4 | Modelado	
| 4.4.1 | Personajes [principal + 6 NPCs]	
| 4.4.2 | Enemigos [4 terrestres + 2 aéreos]	
| 4.4.3 | Jefes [2 tipos]	
| 4.4.4 | Escenarios [2 tipos]	
| 4.5 | Animación de los vídeos	
| 4.6 | Programar e incorporar	
| 4.6.1 |Escenarios del juego [inicio + nivel base + mazmorra + final]	
| 4.6.2 |Interfaces de usuario (UI) [inicio + 2 tipos de juego + opciones + muerte]	
| 4.6.3 | Físicas	
| 4.6.4 | Mecánicas	
| 4.6.4.1 | Desplazamiento entre escenas y niveles	
| 4.6.4.2 | Recolección Monedas y Pociones	
| 4.6.4.3 | Guardado de partida	
| 4.6.4.4 | Guardado de localización	
| 4.6.5 | Sonidos [fondo + 4 efectos de personaje + 3 efectos para cada enemigo + 4 efectos para cada jefe]	
| 4.6.6 | Función de selección de final	
| 4.6.7 | Función de generación de finales [4 tipos]	
| 5 | Pruebas y resultados	**E5**
| 5.1 | Descripción de experimentos	
| 5.2 | Resultados obtenidos	
| 6 | Finalización	**E6**
| 6.1 | Redacción de la memoria	
| 6.2 | Calibración del programa	

Cuadro 3.3: Estructura de descomposicón de tareas (EDT) - Segunda parte

En los EDT anteriores, se puede observar la división de las tareas en 3 grandes grupos que se identifican como niveles generales de estructuración del trabajo. A partir de estos puntos generales, se pueden derivar los elementos específicos que conformarán nuestro proyecto.

El primer grupo corresponde al análisis preliminar, donde se establecen los objetivos a cumplir o alcanzar en nuestro proyecto, las bases técnicas sobre las que se fundamenta nuestro tema y las motivaciones que impulsan su realización. Este grupo se aborda en los capítulos 1 y 2 de esta memoria.

A continuación, el siguiente bloque se refiere a todas las acciones concretas que dan forma a nuestro proyecto, y se desarrolla en este mismo capítulo. Aquí se realiza la planificación temporal y de alcance, se definen los requisitos y se realizan las estimaciones necesarias para una gestión óptima del proyecto. También se abordan las fases previas a las pruebas, como el diseño e implementación. Todo este trabajo y sus tareas se describe en el capítulo 4.

En el tercer apartado, centrado en las pruebas y los resultados, se aborda el capítulo 5, donde se documentan los diferentes experimentos realizados para el desarrollo final de nuestro módulo, y se presentan los resultados obtenidos a partir de diversos planteamien-

tos.

Las entregas en este proyecto están relacionadas con los capítulos que deberían formar parte de una memoria de proyecto formal, y en ellos se incluirá la documentación pertinente para su corrección por parte del supervisor del proyecto.

E1	Preparación
Distribución temporal	Al finalizar la tarea 1
Descripción	En esta entrega se entregaría la preparación inicial del proyecto, que incluye la búsqueda bibliográfica y webgrafía relacionada, la redacción de los motivos y objetivos del proyecto, así como la redacción del Estado del Arte y la definición de los requisitos del proyecto.

Cuadro 3.4: Lista de primera entrega

E2	Análisis
Distribución temporal	Al finalizar la tarea 2
Descripción	En esta entrega se entregaría el análisis del proyecto, que incluye la creación de casos de uso, diagramas de estados, diagramas de actividad, modelo de guardado de datos, diagrama de secuencia y diagrama de clases. También se incluirían las estimaciones temporales y económicas del proyecto

Cuadro 3.5: Lista de segunda entrega

E3	Diseño
Distribución temporal	Al finalizar la tarea 3
Descripción	En esta entrega se entregaría el diseño del juego, que abarca aspectos como la sinopsis de la historia, el establecimiento del género del juego, el diseño de personajes (principal y NPCs), diseño de enemigos, diseño de jefes, diseño de recursos de los escenarios y diseño de las interfaces.

Cuadro 3.6: Lista de tercera entrega

E4	Implementación
Distribución temporal	Al finalizar la tarea 4
Descripción	En esta entrega se entregaría la implementación del juego, que incluye la adquisición de música, efectos de sonido y elementos de UI. También se realizaría el modelado de personajes, enemigos, jefes y escenarios. Además, se llevaría a cabo la animación de los vídeos y la programación e incorporación de elementos como escenarios, interfaces de usuario, físicas, mecánicas, sonidos, función de selección de final y función de generación de finales.

Cuadro 3.7: Lista de cuarta entrega

E5	Pruebas y resultados
Distribución temporal	Al finalizar la tarea 5
Descripción	En esta entrega se entregarían las pruebas y resultados del juego, que incluyen una descripción de los experimentos realizados y los resultados obtenidos en dichas pruebas.

Cuadro 3.8: Lista de quinta entrega

E6	Finalización
Distribución temporal	Al finalizar la tarea 6
Descripción	En esta entrega se entregaría la finalización del proyecto, que implica la redacción de la memoria del proyecto y la calibración del programa.

Cuadro 3.9: Lista de sexta entrega

3.5. Planificación temporal

Dentro de la gestión del tiempo en los proyectos, es fundamental comenzar con la definición de las actividades a realizar. Estas actividades se traducen en tareas e hitos que conforman el alcance del proyecto, tal como se ha establecido en la sección anterior mediante la Estructura de Descomposición de Tareas (EDT).

El proceso de planificación temporal implica realizar un estudio de las duraciones estimadas para cada actividad y programar la secuencia entre ellas. Una vez que se ha establecido el conjunto de tareas, el siguiente paso es estimar el tiempo necesario para completar cada una de ellas.

Utilizaremos la técnica de estimación por experto, en la cual consultaremos a un único experto con experiencia y conocimientos en este tipo de actividades. El experto nos proporcionará una estimación de la duración de cada tarea basada en su experiencia. Además, aplicaremos la técnica de estimación por tres valores, la cual consiste en obtener tres estimaciones para cada tarea: el tiempo más optimista (to), el más pesimista (tp) y el más probable (tm).

$$t_e = \frac{t_o + 4t_m + t_p}{6} \tag{3.1}$$

Al combinar estas técnicas, obtendremos una estimación más precisa y realista del tiempo necesario para completar cada tarea en el proyecto.

El experto designado para realizar las estimaciones de tiempo en horas del proyecto es:

- **Miguel Lozano**. Profesor que tutoriza este proyecto, y encargado de la docencia de las asignaturas de Simulación y Técnicas procedurales de animación, durante el grado. Profesor en el campus de la ETSE-UV (Escuela Técnica Superior de Ingeniería de la Universidad de Valencia) y pertenece al departamento de informática. Sus áreas de investigación se centran en el modelado y la simulación por ordenador, aplicadas a diversos ámbitos. Su amplio conocimiento y experiencia en estas áreas hacen de él un experto calificado para realizar las estimaciones necesarias en nuestro proyecto.

Gracias a la cooperación del experto mencionado anteriormente, se logró generar la siguiente tabla de estimaciones:

Nombres de las tareas	t. optimo	t. medio	t. peor	t. estimado
| 1 | Preparación	**44**	**68**	**86**	**67,00**
| 1.1 | Búsqueda bibliográfica y webgrafía	12	24	30	23,00
| 1.2 | Redacción de motivos y objetivos	4	8	10	7,67
| 1.3 | Redacción del Estado del Arte	18	24	30	24,00
| 1.4 | Definir requisitos del proyecto	10	12	16	12,33
| 2 | Análisis	**58**	**72**	**100**	**74,33**
| 2.1 | Crear Çasos de Uso"	10	12	16	12,33
| 2.2 | Crear "Diagrama de Estados"	6	8	12	8,33
| 2.3 | Crear "Diagrama de Actividad"	6	8	12	8,33
| 2.4 | Crear "Modelo de guardado de datos"	6	8	12	8,33
| 2.5 | Crear "Diagrama de Secuencia"	10	12	16	12,33
| 2.6 | Crear "Diagrama de Clases"	10	12	16	12,33
| 2.7 | Estimaciones temporales y económicas	10	12	16	12,33
| 3 | Diseño	**110**	**204**	**280**	**102,67**
| 3.1 | Diseño del juego	46	84	112	82,33
| 3.1.1 | Sinopsis de la historia	2	4	6	4,00
| 3.1.2 | Establecer el género del juego	2	4	6	4,00
| 3.1.3 | Diseño de personajes [principal + 6 NPCs]	12	24	30	23,00
| 3.1.4 | Diseño de enemigos [4 terrestres + 2 aéreos]	12	24	30	23,00
| 3.1.5 | Diseño de jefes [2 tipos]	4	8	12	8,00
| 3.1.6 | Diseño de recursos de los escenarios	4	8	12	8,00
| 3.1.7 | Diseño de las interfaces [inicio + 2 tipos de juego + opciones + muerte]	10	12	16	12,33
| 3.2 | Diseño de las mecánicas del juego	10	20	32	20,33
| 3.2.1 | Mecanismo para abrir puertas	2	4	6	4,00
| 3.2.2 | Mecanismo para desplazarse entre niveles	2	4	6	4,00
| 3.2.3 | Mecanismo para almacenar recursos	4	8	12	8,00
| 3.2.4 | Mecanismos de guardado	2	4	8	4,33
| 4 | Implementación	**84**	**168**	**226**	**163,67**
| 4.1 | Adquisición de música	4	8	12	8,00
| 4.2 | Adquisición de efectos de sonido	2	4	6	4,00
| 4.3 | Adquisición de elementos de UI	2	4	6	4,00
| 4.4 | Modelado	32	64	84	62,00
| 4.4.1 | Personajes [principal + 6 NPCs]	12	24	30	23,00
| 4.4.2 | Enemigos [4 terrestres + 2 aéreos]	12	24	30	23,00
| 4.4.3 | Jefes [2 tipos]	4	8	12	8,00
| 4.4.4 | Escenarios [2 tipos]	4	8	12	8,00
| 4.5 | Animación de los vídeos	2	4	6	4,00
| 4.6 | Programar e incorporar	42	84	112	81,67
| 4.6.1 |Escenarios del juego [inicio + nivel base + mazmorra + final]	12	24	30	23,00
| 4.6.2 |Interfaces de usuario (UI) [inicio + 2 tipos de juego + opciones + muerte]	4	8	12	8,00
| 4.6.3 | Físicas	4	8	12	8,00
| 4.6.4 | Mecánicas	10	20	28	19,67
| 4.6.4.1 | Desplazamiento entre escenas y niveles	2	4	6	4,00
| 4.6.4.2 | Recolección Monedas y Pociones	2	4	6	4,00
| 4.6.4.3 | Guardado de partida	4	8	10	7,67
| 4.6.4.4 | Guardado de localización	2	4	6	4,00
| 4.6.5 | Sonidos [fondo + 4 efectos de personaje + 3 efectos para cada enemigo + 4 efectos para cada jefe]	4	8	10	7,67
| 4.6.6 | Función de selección de final	4	8	10	7,67
| 4.6.7 | Función de generación de finales [4 tipos]	4	8	10	7,67
| 5 | Pruebas y resultados	**8**	**16**	**20**	**15,33**
| 5.1 | Descripción de experimentos	4	8	10	7,67
| 5.2 | Resultados obtenidos	4	8	10	7,67
| 6 | Finalización	**44**	**80**	**96**	**76,67**
| 6.1 | Redacción de la memoria	32	56	64	53,33
| 6.2 | Calibración del programa	12	24	32	23,33
TOTAL	**348**	**608**	**808**	**499,67**

Cuadro 3.10: Estimación de tres valores proporcionada por **Miguel Lozano**.

A partir de los datos extraídos de las tablas anteriores, encontramos que el total de horas necesarias para la realización del proyecto según nuestro experto sería de 499,67 horas.

El resultado proporcionado por estas estimaciones se presenta como una media en horas, sin embargo, para una mejor visualización en un diagrama de Gantt, se realizará la conversión a días (considerando un día como una jornada de ocho horas) y se mostrarán las medias de cada una de estas tareas en días.

Bloque de tareas	**Duración (horas)**	**Duración (días aprox.)**
Preparación	67,00	8,38
Análisis	74,33	9,29
Diseño	102,67	12,83
Implementación	163,67	20,46
Pruebas y resultados	15,33	1,92
Finalización	76,67	9,58
Total	**499,67**	**62,46**

Cuadro 3.11: Estimación temporal final por bloque de tareas.

A continuación, presentamos las estimaciones previamente realizadas para cada tarea mediante un diagrama de Gantt. En este diagrama, las duraciones de las tareas se expresan en días estimados:

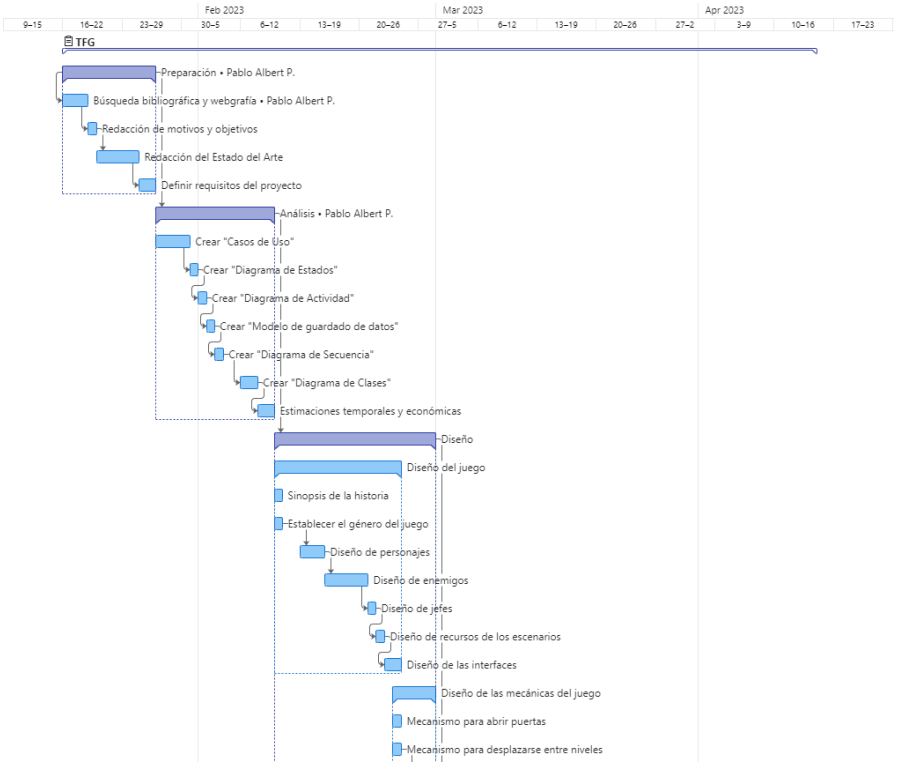

Figura 3.1: Diagrama de Gantt (primera parte).

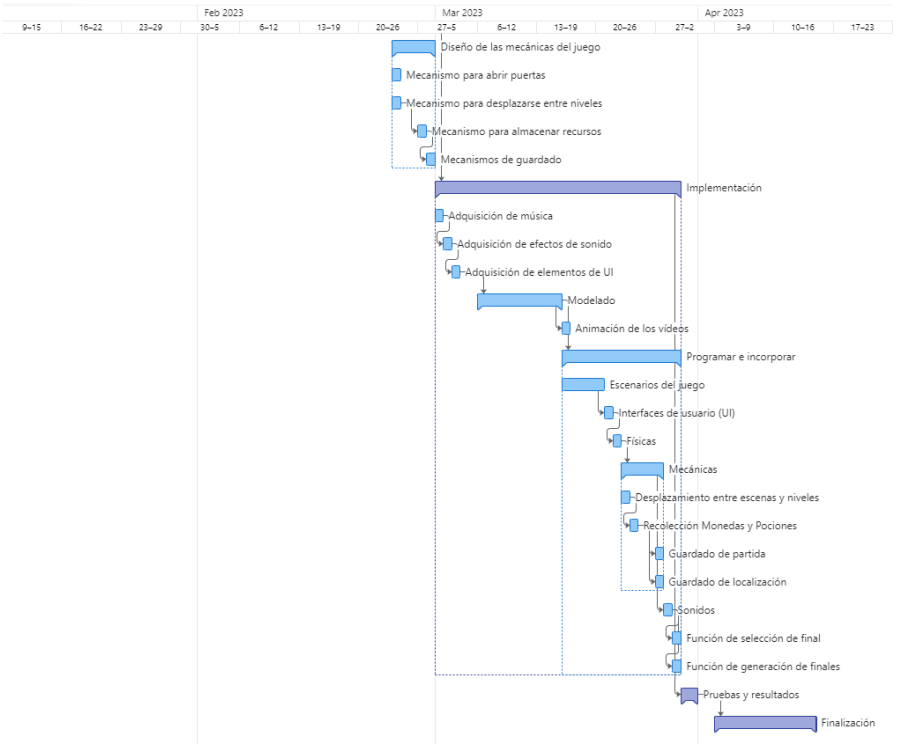

Figura 3.2: Diagrama de Gantt (segunda parte).

3.6. Estimación de costes

Esta sección está dedicada a la gestión de costos directos asociados al proyecto del Trabajo de Fin de Grado. Por lo tanto, se enfoca en la estimación de los valores de diferentes tipos de costos, como costos de hardware, software y personales (trabajo realizado por el propio equipo).

3.6.1. Costes de hardware

En lo relativo a los costes del hardware, en este proyecto se reduce al coste de los equipos informáticos usados para la realización de las actividades del personal.

Portátil

Nombre	ASUS TUF Gaming F15 FX507ZC4-HN002
Procesador	Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz 2.21 GHz
Disco Duro	1TB SSD
Memoria	16,0 GB
Gráfica	Intel Core i7-12700H
Pulgadas de pantalla	15.6"
Precio (PC Componentes)	1299 €

Cuadro 3.12: Información portátil utilizado.

Pantalla

Nombre	AOC 24B2XDAM
Procesador	Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz 2.21 GHz
Frecuencia de actualización	75Hz
Calidad imagen	FullHD
Pulgadas de pantalla	23.8"
Tiempo de respuesta	4 ms
Profundidad de color	8 bits
Precio (PC Componentes)	89,99 €

Cuadro 3.13: Información pantalla auxiliar utilizada.

Dispositivos de entrada

Ratón:

Nombre	Gaming Mouse iMice GW-X7
RGB	Si
Tecnología de conectividad	Inalámbrico
Ajustable DPI	Si
Precio (Amazon)	20,52 €

Cuadro 3.14: Información ratón utilizado.

Tableta Gráfica:

Nombre	Wacom Intuos Tableta Gráfica
Área activa	216 x 135 mm
Conexión Bluetooth	Si
Software creativos y educativos	Si
Sensibilidad	Ajustable
Precio (Amazon)	77,42 €

Cuadro 3.15: Información tableta gráfica utilizada.

Equipo de sonido

Teclado Midi:

Nombre	M-Audio Keystation Mini
Número de teclas	32
Número de pads	4
Precio (Amazon)	45,00 €

Cuadro 3.16: Información teclado midi utilizado.

Cascos:

Nombre	M-Audio Keystation Mini
Tipo	Cardioide
Tecnología de conectividad	Inalámbrico
Bateria	5,3 V
Precio (Amazon)	27,99 €

Cuadro 3.17: Información casco utilizado.

Y Lo que vamos a ver a continuación es el coste total del material hardware que se usara durante la vida del proyecto, sus años de vida útil y también su amortización, para poder tenerlos guardados para poder realizar nuestro presupuesto posteriormente con mayor facilidad.

Rol	Componente	Precio/Unidad	Vida Útil	% Uso	Amortización
Director	Portátil	1299€	10 años	100	779,4 €
	Ratón	20,52 €	100 horas	75	0 €
Diseñador artístico	Portátil	1299€	10 años	100	779,4 €
	Pantalla	89,99 €	10 años	100	89,99 €
	Tableta gráfica	77,42 €	4 años	65	58,065 €
	Ratón	20,52€	100 horas	35	0 €
Programador	Portátil	1299€	10 años	100	779,4 €
	Pantalla	89,99 €	5 años	100	89,99 €
	Ratón	20,52€	100 horas	75	0 €
Diseñador técnico	Portátil	1299€	10 años	100	779,4 €
	Pantalla	89,99 €	5 años	100	89,99 €
	Tableta gráfica	77,42 €	4 años	65	58,065 €
	Ratón	20,52 €	100 horas	35	0 €
Diseñador de sonido	Portátil	1299€	10 años	100	779,4 €
	Pantalla	89,99 €	5 años	100	89,99 €
	Teclado MIDI	45,00 €	6 años	50	45,00 €
	Cascos	27,99€	5 años	100	5,598 €
	Micrófono	38,99€	30 años	75	38,99€
	Ratón	20,52€	100 horas	50	0 €
Tester	Portátil	1299€	10 años	100	779,4 €
	Ratón	20,52€	100 horas	75	0 €
Total					5.243,25 €

Cuadro 3.18: Coste económico de los componentes hardware del equipo utilizado por los miembros.

Además, se considera una amortización de 6 € para los cables HDMI que se conectan a las pantallas, ya que son utilizados por primera vez para el proyecto.

3.6.2. Costes de software

En cuanto al software, se debería analizar el costo económico de utilizar los programas mencionados en el capítulo 2, así como otros utilizados para el desarrollo de la memoria, a fin de obtener el costo total de su uso en todo el proyecto. Sin embargo, como se ha especificado en ese capítulo, la descarga de los programas es gratuita o son proporcionados por la universidad, por lo que no habría gastos asociados a la instalación básica. Además, al recomendar una estructuración sencilla del escenario para la creación del módulo, no se requerirán licencias de nivel avanzado para un desarrollo complejo del videojuego.

Por lo tanto, la tabla de software reflejará que no se generarán gastos económicos relacionados con el software que afecten al proyecto.

Software	Precio/Licencia
Photopea	Código abierto (gratuito)
OpenToonz	Código abierto (gratuito)
Audacity	Código abierto (gratuito)
Unity	Código abierto (gratuito)
Visual Paradigm	Código proporcionado por la universidad (gratuito)
GanttProject	Código abierto (gratuito)
Overleaf	Plataforma de edición gratuita

Cuadro 3.19: Costes de licencia de los softwares utilizados.

3.6.3. Costes personal

En cuanto a los costes personales, nos referimos a los gastos asociados al personal que conforma el equipo encargado del desarrollo del proyecto. En esta sección, se detallarán los roles que asumirían los miembros del equipo y la duración de su participación a lo largo del proyecto.

Hay que tener en cuenta que la estimación de los salarios de las personas involucradas se basa en datos del mercado laboral de España durante el último año, utilizando información del Boletín Oficial del Estado (B.O.E.) [2], así como una búsqueda en portales de empleo como *LinkedIn*.

Rol	Salario Bruto (€/mes)	Duración (meses)	Coste Total
Director general	1.765,51	4	7.062 €
Diseñador artístico	1.283,52	2	2.567 €
Diseñador técnico	1.331,06	2	2.662 €
Programador	1.283,52	3	3.851 €
Jefe de sonidos	1.051,43	2	2.103 €
Tester	875,48	2	1.751 €
TOTAL	7.590,52	15,00	**19.995,58 €**

Cuadro 3.20: Costes del personal presente en el proyecto.

.

3.7. Viabilidad y Riesgos

Para asegurar el desarrollo exitoso del proyecto hasta su finalización, es importante tener en cuenta varios aspectos que pueden tener implicaciones a largo plazo o dar lugar a problemas una vez finalizado. Por lo tanto, en esta sección se explicarán los diversos aspectos de viabilidad del proyecto y los posibles riesgos que podrían surgir durante su período de ejecución.

3.7.1. Viabilidad económica

En esta sección, evaluaremos la viabilidad del presupuesto calculado, considerando si es realista solicitar el coste total al responsable de financiar el proyecto y si existe posibilidad de obtener financiación para su realización.

Coste del proyecto					
Recursos	Duración (meses)	Salario (€/mes)	Unidades	Contingencias	Coste (€)
Coste personal					
Director general	4	1.765,51	-	2 %	7203,28
Diseñador artístico	2	1.283,52	-	2 %	2618,38
Diseñador técnico	2	1.331,06	-	2 %	2715,36
Programador	3	1.283,52	-	2 %	3927,57
Jefe de sonidos	2	1.051,43	-	2 %	2144,92
Tester	2	875,48	-	2 %	1785,98
Subtotal					**20395,49**
Coste material					
Costes directos					
Recursos	Nº días uso	Precio_Amortizado x día	Unidades	% Uso	Coste Amortizado (€)
Software	88	0	7	100	0,00
Portátil	88	0,618571429	6	100	326,61
Ratón	88	1,6416	6	75	650,07
Pantalla	88	0,085704762	4	100	30,17
Tableta Gráfica	22	0,092166667	1	65	1,32
Teclado MIDI	22	0,035714286	1	50	0,39
Cascos	88	0,026657143	1	100	2,35
Micrófono	88	0,006188889	1	75	0,41
Costes indirectos					
Alquiler	4	300	-	2 %	1224,00
Luz	4	90	-	2 %	367,20
Agua	4	50	-	2 %	204,00
Internet + teléfono	4	90	-	2 %	367,20
Mantenimiento	4	45	-	2 %	183,60
Gastos Administrativos	4	250	-	2 %	1020,00
Subtotal					**4377,31**
TOTAL					**24772,80**

Cuadro 3.21: Tabla de estimación de coste total del proyecto.

Precio de venta	
	Valor (€)
Precio total del proyecto	24772,80
Beneficios (40 % sobre total)	9909,12
Precio de venta (sin IVA)	34681,93
21 % de IVA	7283,20
Precio total de venta	**41965,13**

Cuadro 3.22: Tabla de estimación de coste del producto para obtener beneficios.

3.7.2. Viabilidad legal

En este apartado, abordaremos los aspectos legales y regulaciones aplicables en el ámbito de los videojuegos para evaluar su viabilidad legal. Algunos de los aspectos legales que se suelen considerar son los siguientes:

Derechos de autor:

- Usar contenido original, que no se utilicen gráficos, música, personajes o diálogos totalmente copiados de otro juego sin la autenticación de estos. En el caso de nuestro videojuego, no anticipamos problemas en este aspecto, ya que la mayoría de los elementos son obtenidos de páginas de recursos sin ánimo de lucro. En caso de utilizar componentes de un videojuego conocido, nos aseguraremos previamente de obtener la autorización del autor, realizando modificaciones para evitar similitudes con el original.al.

Marcas comerciales:

- Evitar el uso no autorizado de marcas comerciales o nombres de empresas en cualquier parte del videojuego. En nuestro caso, no tendremos problemas, ya que crearemos escenarios y recursos sin ningún tipo de simbología o referencia a empresas específicas.

Derechos de imagen:

- Evitar el uso de personajes famosos, reconocibles o celebridades sin permiso previo. En nuestro caso, este problema también se resuelve al utilizar un paquete de recursos generales que no permite la caracterización de ninguna persona famosa de la actualidad.

Protección de datos:

- Cumplir con las leyes de protección de datos aplicables, como el Reglamento General de Protección de Datos (GDPR) en la Unión Europea, al manejar la información personal de los usuarios.En nuestro videojuego, nos comprometemos a no divulgar tu información de juego a terceros para que no puedan utilizar tus estadísticas sin tu consentimiento.

Contenido inapropiado:

- Consiste en que nos aseguremos de que el juego cumpla con las clasificaciones de edad y restricciones legales relacionadas con contenido violento, sexual, difamatorio u ofensivo, típico de cualquier videojuego.Dado que nuestro videojuego está dirigido a todas las edades, no incluiremos contenido violento con derramamiento de sangre ni utilizaremos lenguaje soez o vulgar.

Licencias de software:

- Tendremos que verificar si el juego utiliza software de terceros y asegurarnos de que el proyecto cumple con los términos de las licencias correspondientes, incluyendo el software del motor gráfico utilizado.

Contratos de colaboración:

- La norma se concentra más en la acción de establecer contratos claros y vinculantes cuando se trabaja con otros programadores u otros artistas profesionales para definir los términos de la colaboración y los derechos de propiedad intelectual. En nuestro caso, al ser un proyecto principalmente realizado por el alumno, esto no debería ser una preocupación. Sin embargo, si se requiere ayuda externa para solucionar un problema en el horario planificado, el responsable deberá firmar un contrato con el nuevo asistente, teniendo en cuenta las consideraciones mencionadas anteriormente.

Derechos de patentes:

- La norma busca proteger la innovación y las técnicas utilizadas en un trabajo, en caso de que la institución lo considere pertinente. Dado que nuestro videojuego busca introducir una nueva forma de creación de niveles basada en las acciones del jugador durante la partida, es recomendable aplicar esta protección de patentes.

Contratos de licencia de usuario final (CLUF):

- Se concentra en establecer un contrato que establezca los términos y condiciones de uso del juego por parte de los usuarios. Esto es común en cualquier videojuego, ya sea de renombre o de tamaño pequeño para dispositivos móviles. Por lo tanto, nuestro videojuego también debe implementar este contrato en algún momento del juego para evitar responsabilidades legales.

3.7.3. Análisis de riesgos

En este apartado, nos adentraremos en el análisis de los riesgos asociados a nuestro proyecto. El objetivo principal es evaluar y valorar detenidamente todas las posibles situaciones de riesgo que podrían afectar el desarrollo del juego, teniendo en cuenta tanto su probabilidad de ocurrencia como el impacto que podrían generar en el proyecto.

Es fundamental comprender y anticiparse a los posibles obstáculos que podrían surgir durante el proceso de desarrollo, ya que esto nos permitirá tomar medidas preventivas y establecer estrategias de mitigación adecuadas. Al identificar y analizar los riesgos, estaremos preparados para enfrentarlos de manera efectiva, minimizando cualquier impacto negativo en el proyecto.

En esta etapa, examinaremos diversos escenarios que son comúnmente considerados en el análisis de riesgos de proyectos, con el objetivo de abordarlos de manera proactiva y garantizar un desarrollo exitoso del videojuego.

Escenarios de riesgo	**Probabilidad de caso**	**Impacto de caso**	**Riesgo Inicial**
Obtener requisitos poco claros o ambiguos	Alta	Baja	Medio
Planificar con dificultad	Media	Baja	Media
Conflictos entre compañeros de equipo	Baja	Baja	Baja
Mala asignación de recursos	Baja	Medio	Baja
Bajas laborales	Medio	Medio	Medio
Realizar un número de pruebas insuficientes	Baja	Alta	Medio
Ausencia injustificada de personal técnico	Baja	Baja	Baja
Aparición de costes imprevistos	Alto	Baja	Medio
Depender de terceros (ya sea por su tecnología o por sus activos)	Medio	Bajo (si tenemos en cuenta el apartado legal de antes)	Baja
Mal feedback sobre el juego	Medio	Alta	Alta
Errores en los bocetos de diseño	Alta	Baja	Medio
Añadir nuevas características en cualquier punto del proyecto	Alta	Alta	Inaceptable
Problemas legales con otras empresas	Baja	Alta	Medio

Cuadro 3.23: Lista de posibles riesgos del proyecto

Es crucial realizar una planificación exhaustiva de los riesgos y establecer medidas de mitigación o contingencia para abordarlos. Sin embargo, en ciertos casos, es necesario abordar de inmediato los problemas inaceptables antes de avanzar en el proyecto. En el caso de nuestro proyecto, nos comprometemos firmemente a no agregar tareas adicionales más allá de las que se hayan planificado previamente.

En caso de que surja el riesgo de recibir comentarios negativos sobre el proyecto, tenemos un plan de contingencia establecido. Este plan involucraría la contratación de

terceros para crear diseños que sean más atractivos para la comunidad, y posponer los descansos de los miembros del equipo.

Mientras que para mitigar los riesgos descritos anteriormente de menor nivel, los trataríamos de la siguiente forma:

- Para reducir los requisitos poco claros o ambiguos, convendría que se hicieran varias entrevistas a los clientes y preguntar sobre los casos de uso que el reportero tiene, para evitar malentendidos.

- Para evitar planificar con dificultad, en nuestro caso podemos acudir a la ayuda de los profesores que ya tienen experiencia en este campo.

- Para evitar o reducir el ausentismo laboral, se podría implementar un mantenimiento regular de los recursos utilizados por nuestro equipo, así como establecer períodos obligatorios de descanso lejos de las pantallas para permitir el descanso visual y relajación.

- Para evitar mal feedback [2], se pueden realizar pruebas con usuarios en las fases de beta y que aunque el número pueda llegar a ser insuficiente, este pueda ayudar a realizar cambios antes.

- Para evitar costes imprevistos, se deberá de evaluar semanalmente los recursos usados y realizar una revisión del código.

- Si queremos evitar el uso de malos diseños, tendríamos que crear un grupo encargado de revisar que los bocetos presentan una calidad aceptable por el público.

- Para evitar posibles problemas legales, nos aseguraremos de utilizar contenido propio o recursos que sean públicamente reconocidos y legalmente utilizados por programadores o diseñadores en la industria. Esto garantizará que estemos utilizando materiales populares de forma legítima y evitaremos cualquier infracción de derechos de autor u otras cuestiones legales.

[2]En el contexto de un videojuego, se refiere a la información y comentarios que los jugadores proporcionan sobre su experiencia de juego.

Capítulo 4

Desarrollo

4.1. Introducción

En este capítulo, daremos inicio al análisis detallado de los requisitos del proyecto, centrándonos en comprender las funcionalidades que este debe ofrecer y cómo se integran para brindar una experiencia completa al usuario.

Posteriormente, nos enfocaremos en el diseño del proyecto, donde definiremos la arquitectura general del sistema, donde también se incluirá la estructura de la interfaz de usuario, asegurándonos de que sea todo intuitivo y atractivo para el usuario final.

Una vez finalizado el análisis y diseño, comenzara la implementación donde se llevará a cabo la programación, creación e integración de los recursos del proyecto.

4.2. Análisis

En esta sección nos adentraremos en la fase de análisis, la cual marca el punto de partida para el trabajo a realizar en nuestro proyecto. Durante esta etapa, se llevará a cabo un análisis exhaustivo de los requisitos del proyecto.

En primer lugar, se plantearán los casos de uso específicos para el videojuego, los cuales plasman los requisitos previamente explicados en la sección 3.2 de este capítulo. Estos casos de uso ayudarán a comprender las funcionalidades y capacidades del sistema. A través de ellos, se podrá identificar qué acciones y comportamientos debe ser capaz de realizar el videojuego con nuestro modulo.

Por último, se abordará la fase de análisis a través de los diagramas de actividad. Estos diagramas mostrarán los diferentes procesos que el usuario lleva a cabo dentro del videojuego para cumplir con cada uno de los casos de uso identificados. Estos procesos pueden incluir acciones como la interacción con personajes y la respuesta a las interacciones del usuario por parte del sistema o del espía.

4.2.1. Casos de uso

Para analizar nuestro proyecto, al igual que en asignaturas de cursos anteriores, comenzaremos por definir los casos de uso de manera gráfica y literal. Estos casos de uso describirán las funcionalidades principales que se alinean con los requisitos mencionados en el punto 3.2.

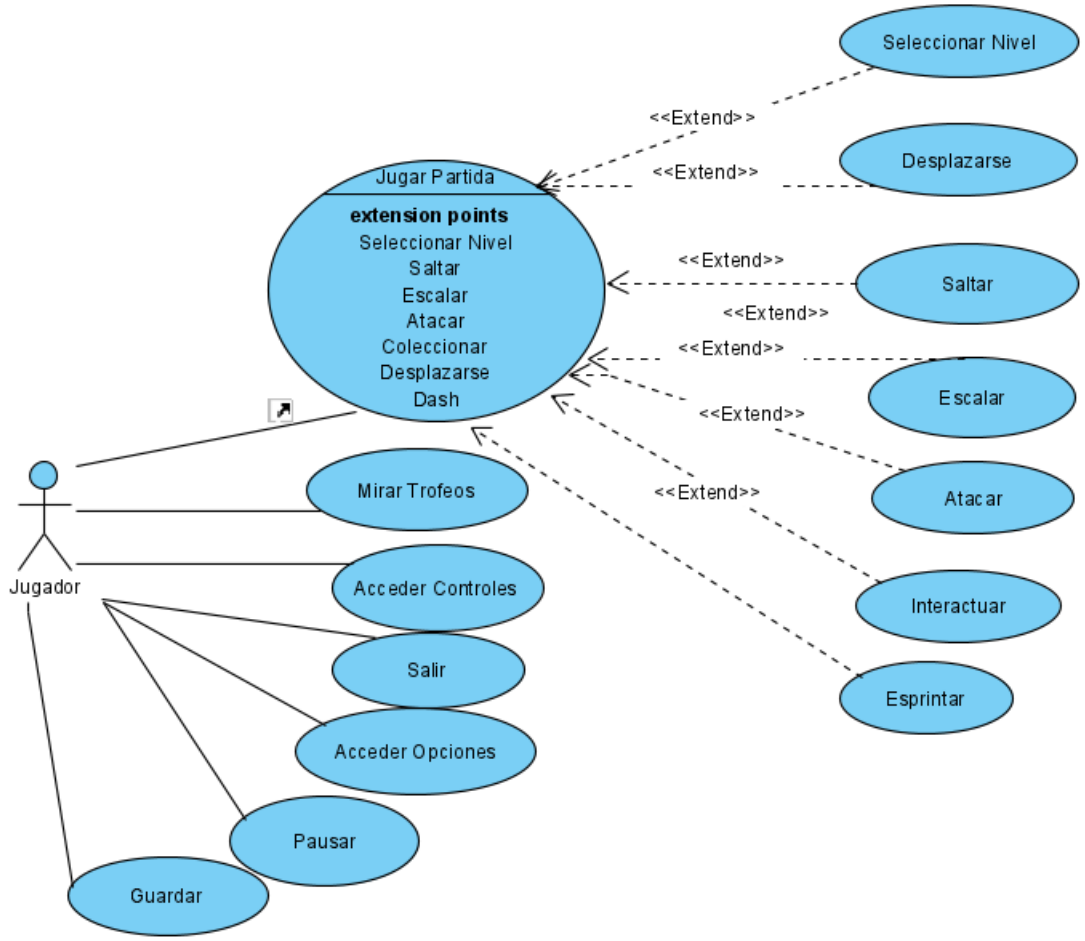

Figura 4.1: Diagrama de casos de uso.

ID Caso de Uso:	Jugar Partida	
Actor principal:	Jugador	
Actores secundarios:	———	
Breve descripción:	El usuario puede entrar a jugar al juego	
Precondiciones:	Necesita estar en el menú principal	
Flujo de los eventos: (teniendo como caso principal el de seleccionar nivel)	Acciones del actor	Respuesta del Sistema
	El jugador accede al juego	
		El sistema muestra el menú principal
	El jugador selecciona la opción de "jugar"	
		El sistema carga la escena del nivel de selección de nivel
	El jugador puede seleccionar uno de los niveles	
		El sistema carga la escena del nivel
Flujo alternativo (desplazarse):	Acciones del actor	Respuesta del Sistema
	Véase CU7 "Desplazarse"	
Flujo alternativo (saltar):	Acciones del actor	Respuesta del Sistema
	Véase CU8 "Saltar"	
Flujo alternativo (escalar):	Acciones del actor	Respuesta del Sistema
	Véase CU9 "Escalar"	
Flujo alternativo (atacar):	Acciones del actor	Respuesta del Sistema
	Véase CU10 "Atacar"	
Flujo alternativo (interactuar):	Acciones del actor	Respuesta del Sistema
	Véase CU11 "Interactuar"	
Flujo alternativo (Esprintar):	Acciones del actor	Respuesta del Sistema
	Véase CU12 "Esprintar"	
Postcondiciones:	———-	
Excepciones:	Cuando el jugador muere, no se puede usar los controles de juego	

Cuadro 4.1: Descripción del CU1 de "Jugar Partida".

ID Caso de Uso:	Mirar Trofeos	
Actor principal:	Jugador	
Actores secundarios:	———	
Breve descripción:	El usuario puede observar los trofeos que ha conseguido con los diferentes finales, tanto en al principio del juego como en el primer nivel.	
Precondiciones:	Para ver algún trofeo, se debe haber pasado el juego por lo menos una vez	
Flujo de los eventos:	Acciones del actor	Respuesta del Sistema
	El jugador accede al juego	
		El sistema muestra el menú principal
	El jugador selecciona la opción de "trofeos"	
		El sistema muestra los trofeos y la descripción de estos
Flujo alternativo (mira los trofeos en la sala de trofeos):	Acciones del actor	Respuesta del Sistema
		(Acción 2) El sistema muestra el menú principal
	El jugador selecciona la opción de "jugar"	
		El sistema carga la escena del nivel de selección de nivel
	El jugador entra dentro de la torre de trofeos	
		El sistema carga la escena de la torre de trofeos
		EL sistema carga los trofeos obtenidos por el jugador
Postcondiciones:	———-	
Excepciones:	El jugador no vera más de 2 trofeos del mismo logro, por lo que al realizar otra vez un final, este no se vera reflejado	

Cuadro 4.2: Descripción del CU2 de "Mirar Trofeos".

ID Caso de Uso:	Acceder Controles	
Actor principal:	Jugador	
Actores secundarios:	———	
Breve descripción:	El usuario puede ver los controles del juego	
Precondiciones:	Necesita pausar el juego o que este en el menú principal	
Flujo de los eventos:	Acciones del actor	Respuesta del Sistema
	El jugador accede al juego	
		El sistema muestra el menú principal
	El jugador selecciona la opción de “controles”	
		El sistema muestra los controles del personaje
Flujo alternativo (acceder a los controles mientras se juega):	Acciones del actor	Respuesta del Sistema
		(Acción 2) El sistema muestra el menú principal
	El jugador selecciona la opción de “jugar”	
		El sistema carga la escena del nivel de selección de nivel
	El jugador pulsa el botón de pausa	
		El sistema muestra el menú de juego
	El jugador selecciona la opción de “controles”	
		El sistema muestra los controles del personaje
Postcondiciones:	———————-	
Excepciones:	Cuando el jugador muere, no puede ver los controles	

Cuadro 4.3: Descripción del CU3 de “Acceder Controles“.

ID Caso de Uso:	Salir	
Actor principal:	Jugador	
Actores secundarios:	———	
Breve descripción:	El usuario puede dejar de jugar al videojuego	
Precondiciones:	Haber muerto, haber pausado el juego o estar en el menú principal	
Flujo de los eventos:	Acciones del actor	Respuesta del Sistema
	El jugador accede al juego	
		El sistema muestra el menú principal
	El jugador selecciona la opción de “salir”	
		El sistema pregunta si está seguro de la acción
	El jugador afirma su acción	
		El sistema se cierra
Flujo alternativo (salir durante el juego):	Acciones del actor	Respuesta del Sistema
		(Acción 2) El sistema muestra el menú principal
	El jugador selecciona la opción de “jugar”	
		El sistema carga la escena del nivel de selección de nivel
	El jugador pulsa el botón de pausa	
		El sistema muestra el menú de juego
	(Acción 3) El jugador selecciona la opción de “salir”	
Flujo alternativo (salir durante una muerte):	Acciones del actor	Respuesta del Sistema
		(Acción 2) El sistema muestra el menú principal
	El jugador selecciona la opción de “jugar”	
		El sistema carga la escena del nivel de selección de nivel
	El jugador entra dentro de un nivel	
		El sistema carga la escena del nivel
	El jugador se muere	
		El sistema muestra el menú de muerte
	(Acción 3) El jugador selecciona la opción de “salir”	
Flujo alternativo (negarse a salir):	Acciones del actor	Respuesta del Sistema
		El sistema pregunta si está seguro de la acción
	El jugador selecciona la opción de “jugar”	
		(Acción 4) El sistema carga la escena del nivel de selección de nivel
	El jugador se niega	
		El sistema vuelve al menú que mostrara anteriormente
Postcondiciones:	———————-	
Excepciones:	———————-	

Cuadro 4.4: Descripción del CU4 de “Salir“.

ID Caso de Uso:	Pausar	
Actor principal:	Jugador	
Actores secundarios:	———	
Breve descripción:	El usuario puede parar el tiempo de juego	
Precondiciones:	Necesita de estar jugando o de estar en el piso de seleccionar nivel	
Flujo de los eventos:	Acciones del actor	Respuesta del Sistema
	El jugador accede al juego	
		El sistema muestra el menú principal
	El jugador selecciona la opción de "jugar"	
		El sistema carga la escena del nivel de selección de nivel
	El jugador pulsa el botón de pausa	
		El sistema muestra el menú de juego
Postcondiciones:	El delta de tiempo se ha de quedar a tiempo hasta que se vuelta a retomar el juego	
Excepciones:	———	

Cuadro 4.5: Descripción del CU5 de "Pausar".

ID Caso de Uso:	Guardar	
Actor principal:	Jugador	
Actores secundarios:	———	
Breve descripción:	El usuario guarda sus datos al realizar ciertas acciones en el juego	
Precondiciones:	Necesita pausar el juego	
Flujo de los eventos:	Acciones del actor	Respuesta del Sistema
	El jugador selecciona un nivel	
		El sistema carga la escena del nivel
	El jugador se pasa un nivel	
		El sistema guarda todos los datos del jugador
Flujo alternativo (decide retroceder de nivel):	Acciones del actor	Respuesta del Sistema
		(Acción 2) El sistema carga la escena del nivel
	El jugador vuelve por la entrada	
		El sistema guarda todos los datos del jugador
Flujo alternativo (decide usar punto de control):	Acciones del actor	Respuesta del Sistema
		(Acción 2) El sistema carga la escena del nivel
	El jugador interactúa con un punto de control	
		El sistema hace una copia temporal de la vida y de la posición del guardado
Postcondiciones:	Datos guardados de forma segura	
Excepciones:	Cuando se termina el juego, se hace un reinicio de todos los datos de valoración, exceptuando los trofeos que ha conseguido	

Cuadro 4.6: Descripción del CU6 de "Guardar".

ID Caso de Uso:	Desplazar	
Actor principal:	Jugador	
Actores secundarios:	———	
Breve descripción:	El usuario puede desplazarse hacia los lados	
Precondiciones:	Necesita continuar el juego o que este en el piso de selección de nivel	
Flujo de los eventos:	Acciones del actor	Respuesta del Sistema
	El jugador selecciona un nivel	
		El sistema carga la escena del nivel
	El jugador pulsa las techas "w"," a" o las flechas.	
		El sistema mueve el personaje hacia la dirección de la tecla
Postcondiciones:	No debe de haber obstáculos en la zona de desplazamiento	
Excepciones:	El jugador no puede controlar el desplazamiento al morir o al pasar de un nivel a otro	

Cuadro 4.7: Descripción del CU7 de "Desplazar".

ID Caso de Uso:	Saltar	
Actor principal:	Jugador	
Actores secundarios:	———	
Breve descripción:	El usuario puede saltar por el mapa	
Precondiciones:	Necesita de tocar anteriormente el suelo	
Flujo de los eventos:	Acciones del actor	Respuesta del Sistema
	El jugador selecciona un nivel	
		El sistema carga la escena del nivel
	El jugador pulsa las techa espaciadora	
		El sistema mueve de forma dinámica el personaje hacia arriba
Postcondiciones:	———	
Excepciones:	El jugador puede saltar mientras esta escalando una pared	

Cuadro 4.8: Descripción del CU8 de "Saltar".

ID Caso de Uso:	Escalar	
Actor principal:	Jugador	
Actores secundarios:	———	
Breve descripción:	El usuario puede desplazarse por las paredes verticales	
Precondiciones:	Necesita estar en contacto con una pared escalable	
Flujo de los eventos:	Acciones del actor	Respuesta del Sistema
	El jugador selecciona un nivel	
		El sistema carga la escena del nivel
	El jugador mantiene la tecla "c"	
		El sistema mantiene el jugador pegado a la pared
Postcondiciones:	———-	
Excepciones:	———-	

Cuadro 4.9: Descripción del CU9 de "Escalar".

ID Caso de Uso:	Atacar	
Actor principal:	Jugador	
Actores secundarios:	Enemigo	
Breve descripción:	El usuario puede desplazarse hacia los lados	
Precondiciones:	No estar atacando previamente	
Flujo de los eventos:	Acciones del actor	Respuesta del Sistema
	El jugador selecciona un nivel	
		El sistema carga la escena del nivel
	El jugador pulsa las techas "z"	
		El sistema realiza el evento de ataque hacia donde este apuntando el personaje
Postcondiciones:	Si choca el ataque con un enemigo, el jugador le quita vida a este	
Excepciones:	El jugador no puede pulsar de forma seguida el botón de atacar.	

Cuadro 4.10: Descripción del CU10 de "Atacar".

ID Caso de Uso:	Interactuar	
Actor principal:	Jugador	
Actores secundarios:	———	
Breve descripción:	El usuario puede realizar acciones con ciertos recursos del mapa	
Precondiciones:	Necesita estar a cierta distancia del objeto interactuable	
Flujo de los eventos:	Acciones del actor	Respuesta del Sistema
	El jugador selecciona un nivel	
		El sistema carga la escena del nivel
	El jugador pulsa las techas "v" sobre un objeto interactuable	
		El sistema activa el evento relacionado con el objeto interactuable
Postcondiciones:	———-	
Excepciones:	No puede interactuar con objetos que ya ha interactuado anteriormente y en cinemáticas no puede interaccionar con nada	

Cuadro 4.11: Descripción del CU11 de "Interactuar".

ID Caso de Uso:	Esprintar	
Actor principal:	Jugador	
Actores secundarios:	————	
Breve descripción:	El usuario puede desplazarse en muy poco tiempo cierta cantidad de distancia	
Precondiciones:	Necesita desplazarse hacia un lado lateral	
Flujo de los eventos:	Acciones del actor	Respuesta del Sistema
	El jugador selecciona un nivel	
		El sistema carga la escena del nivel
	El jugador pulsa las techas "x"	
		El sistema activa el efecto de esprint y desplaza rápidamente al personaje
Postcondiciones:	————-	
Excepciones:	————-	

Cuadro 4.12: Descripción del CU12 de "Esprintar".

ID Caso de Uso:	Acceder Opciones	
Actor principal:	Jugador	
Actores secundarios:	———	
Breve descripción:	El usuario puede modificar diferentes elementos audiovisuales del juego de forma dinámica	
Precondiciones:	Necesita estar jugando, no estar en el menú principal	
Flujo de los eventos:	Acciones del actor	Respuesta del Sistema
		El sistema carga la escena del nivel
	El jugador pulsa el botón de pausa	
		El sistema muestra el menú de juego
	El jugador pulsa el botón de opciones	
		El sistema muestra el menú de opciones
	El jugador modifica el tamaño de pantalla	
(modificar el tamaño de pantalla)		El sistema cambia el tamaño de la ventana y guarda el cambio
Flujo alternativo	Acciones del actor	Respuesta del Sistema
		(Acción 5) El sistema muestra el menú de opciones
	El jugador modifica el slider de volumen general	
(modificar el volumen general):		El sistema cambia el volumen general y guarda el cambio
Flujo alternativo	Acciones del actor	Respuesta del Sistema
		(Acción 5) El sistema muestra el menú de opciones
	El jugador modifica el slider de volumen de los efectos	
(modificar el volumen de los efectos):		El sistema cambia el volumen de los efectos y guarda el cambio
Flujo alternativo	Acciones del actor	Respuesta del Sistema
		(Acción 5) El sistema muestra el menú de opciones
	El jugador modifica el slider del brillo	
(modificar el brillo de la pantalla):		El sistema cambia el brillo y guarda el cambio
Flujo alternativo	Acciones del actor	Respuesta del Sistema
		(Acción 5) El sistema muestra el menú de opciones
	El jugador modifica el tipo de daltonismo seleccionado en la lista	
(modificar el color del daltonismo):		El sistema cambia la paleta de colores según el modo seleccionado y guarda el cambio
Postcondiciones:	————-	
Excepciones:	Cuando el jugador está en el menú principal, no se muestra esta propiedad	

Cuadro 4.13: Descripción del CU13 de "Acceder Opciones".

4.2.2. Diagrama de estados

En esta subsección, nos adentraremos en los diagramas de estado en el contexto de nuestro proyecto. Los diagramas de estado son una herramienta crucial para modelar el comportamiento dinámico de los elementos interactivos dentro del juego. Proporcionan una representación visual de los diferentes estados en los que puede encontrarse un objeto o personaje, así como las transiciones entre dichos estados.

El objetivo principal de utilizar diagramas de estado en el desarrollo de videojuegos es comprender y definir claramente las reglas y mecánicas del juego, así como capturar la lógica y secuencia de eventos que ocurren durante el gameplay. Estos diagramas nos permiten visualizar de manera estructurada cómo los diferentes elementos del juego interactúan entre sí y cómo responden a las acciones del jugador.

Jugador

Durante la elaboración de este documento, se llegó a la conclusión de que las principales acciones del jugador son las siguientes:

- Reposo: El personaje permanece inmóvil en su posición, ejecutando una animación que simula estar descansando de pie.

- Caminar: El personaje se desplaza lateralmente por el suelo, realizando una animación de pasos en un bucle continuo.

- Atacar: El personaje realiza una animación de ataque hacia la dirección apuntada, y se desbloquea el área de ataque de la espada.

- Esprintar: El personaje se lanza y rueda en una animación, desplazándose a una velocidad mayor que al caminar y sin poder detenerse.

- Escalar: Cuando el personaje encuentra una superficie escalable, puede trepar por la pared y colgarse de ella.

- Saltar: Dependiendo de la duración de la acción, el salto se realiza con mayor o menor fuerza. Hay dos animaciones diferentes, una para cuando el personaje está ascendiendo y otra para cuando está descendiendo.

Con este conjunto de acciones del jugador, se obtiene el siguiente diagrama de estados:

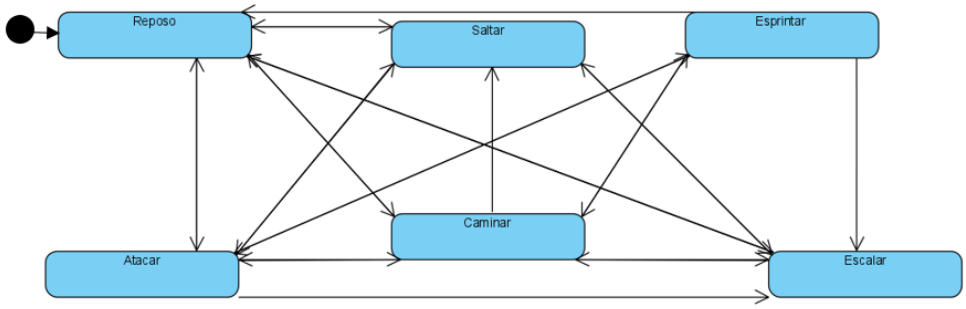

Figura 4.2: Diagrama de estado del Jugador.

En este caso, para que el usuario pueda controlar al personaje, deberá utilizar una serie de comandos:

- Reposo: El personaje se encuentra en reposo cuando no se están pulsando teclas de movimiento.
- Desplazamiento lateral: Al pulsar las teclas “a“, “d“, “←“ o “→“, el personaje se desplaza hacia los lados.
- Salto: Al pulsar la barra espaciadora, el jugador realiza un salto, y la altura del salto dependerá de cuánto tiempo se mantenga pulsada.
- Ataque: Al pulsar la tecla “z“, el personaje realiza un movimiento de ataque.
- Movimiento brusco: Al pulsar la tecla “x“ junto con las teclas de desplazamiento, el personaje realiza un movimiento brusco en la dirección indicada.
- Escalada: Al mantener pulsada la tecla “c“ junto con las teclas “w“, “s“ o las flechas verticales, el personaje puede escalar superficies escalables.

Los comandos permitirán al usuario interactuar con el personaje y controlar sus acciones en el juego, al mismo tiempo que nuestro módulo interceptará y recabará información para la generación del final del juego.

NPCs

Mientras nuestro personaje se desplaza por el mapa, puede encontrarse con personajes secundarios que brindarán ayuda o agregarán un toque humorístico a la historia. Para ello, estos personajes tendrán las siguientes acciones:

- Reposo: El personaje se mantiene inmóvil en su posición, ejecutando una animación que simula estar descansando de pie.

- Caminar: El personaje se desplaza lateralmente por el suelo, mostrando una animación de pasos en bucle.

- Dialogar: El personaje inicia una conversación que proporciona información al jugador cuando se interactúa con él.

A partir de este conjunto de acciones, se pueden representar con el diagrama de estados como se muestra a continuación.

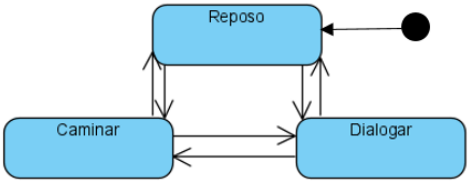

Figura 4.3: Diagrama de estado de los NPCs.

Enemigos

Cuando nuestro personaje se adentra en la mazmorra, se encontrará con diversos enemigos que intentarán hacerle daño de diferentes formas, lo que motivará al jugador a combatir y esquivar sus ataques. Estas acciones se representan de la siguiente manera:

- Reposo: El enemigo se mantiene inmóvil en su posición, ejecutando una animación como si estuviera descansando de pie.

- Caminar: El enemigo se desplaza lateralmente en el suelo, mostrando una animación de pasos en bucle.

- Perseguir: El enemigo comienza a desplazarse en dirección al jugador al detectarlo en un rango de distancia.

- Atacar: El enemigo realiza un ataque en función de la última dirección en la que detectó al jugador.

- Morir: El enemigo se desvanece en su posición y desaparece después de un tiempo.

Con esta lista de posibles acciones que pueden tomar los enemigos, obtenemos un diagrama de estados que se muestra a continuación.

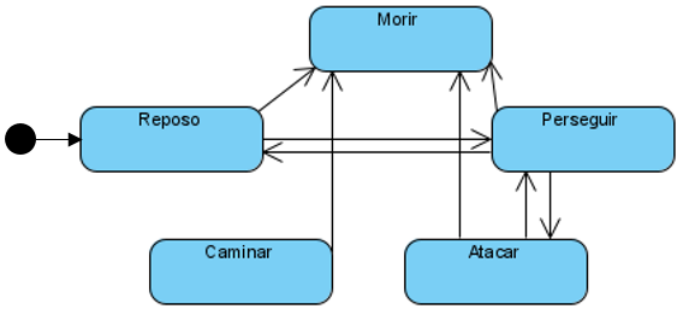

Figura 4.4: Diagrama de estado de los enemigos.

Jefe 1

Cuando nuestro personaje llega a la sala del primer jefe, se enfrentará al primer enemigo poderoso, que posee las siguientes acciones:

- Reposo: El jefe se mantiene inmóvil en su posición, ejecutando una animación como si estuviera descansando de pie.

- Perseguir: El jefe comienza a desplazarse en dirección al jugador.

- Embestir: El jefe inicia una carrera desenfrenada hasta el límite lateral donde detectó por última vez al jugador.

- Atacar con estoque: El jefe realiza un ataque lateral extendido hacia la posición del jugador.

- Atacar con guillotina: El jefe realiza un ataque descendente desde arriba hacia la zona lateral donde se encuentra el jugador.

- Aturdirse: El jefe se queda confuso durante un tiempo en su posición.

- Morir: El jefe se desvanece en su posición y desaparece después de un tiempo.

A partir de la lista de posibles dinámicas del jefe, obtenemos un diagrama de estados que se muestra a continuación.

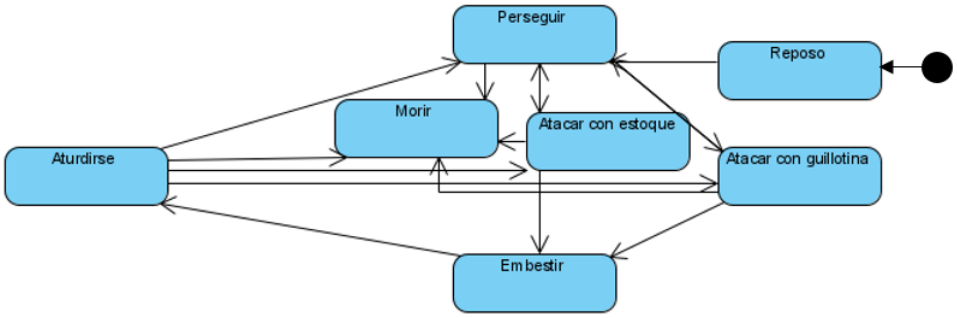

Figura 4.5: Diagrama de estado del primer jefe.

Jefe 2

Cuando nuestro personaje llega a la sala del segundo jefe, se enfrenta al último enemigo poderoso, que posee las siguientes acciones:

- Reposo: El jefe se mantiene inmóvil en su posición, ejecutando una animación como si estuviera descansando de pie.

- Perseguir: El jefe comienza a desplazarse en dirección al jugador.

- Atacar con disparo: El jefe dispara una serie de proyectiles en forma de bomba de racimo a diferentes velocidades y ángulos.

- Atacar con pinchos: El jefe realiza un ataque lateral de pinchos hacia el jugador desde el suelo.

- Atacar con maza: El jefe realiza un ataque con una maza hacia el jugador.

- Atacar con lluvia: El jefe activa la lluvia ácida en el escenario, lo que dificulta la jugabilidad.

- Morir: El jefe se desvanece en su posición y desaparece después de un tiempo.

El resultado de estas acciones se muestra a continuación en un diagrama de estados.

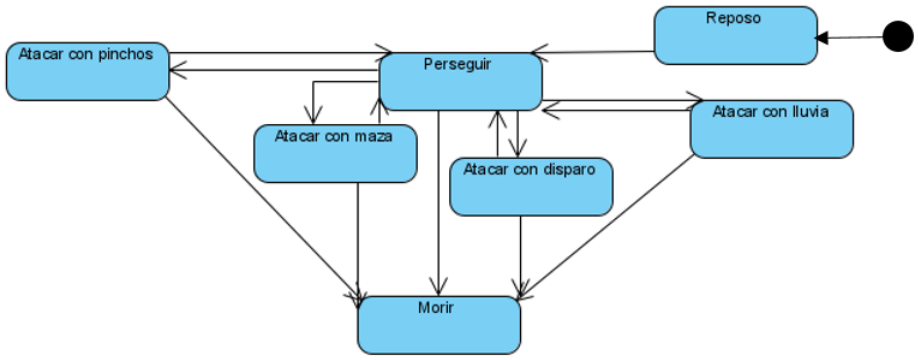

Figura 4.6: Diagrama de estado del segundo jefe.

4.2.3. Diagrama de actividad

En esta última subsección del apartado de Análisis, desde la perspectiva de la Ingeniería del Software, presentamos una simulación de la prueba de funcionamiento del sistema con el jugador. Para ello, utilizamos un diagrama de actividades que representa el flujo de la mayoría de las funcionalidades del proyecto, brindando una visión completa de su ejecución.

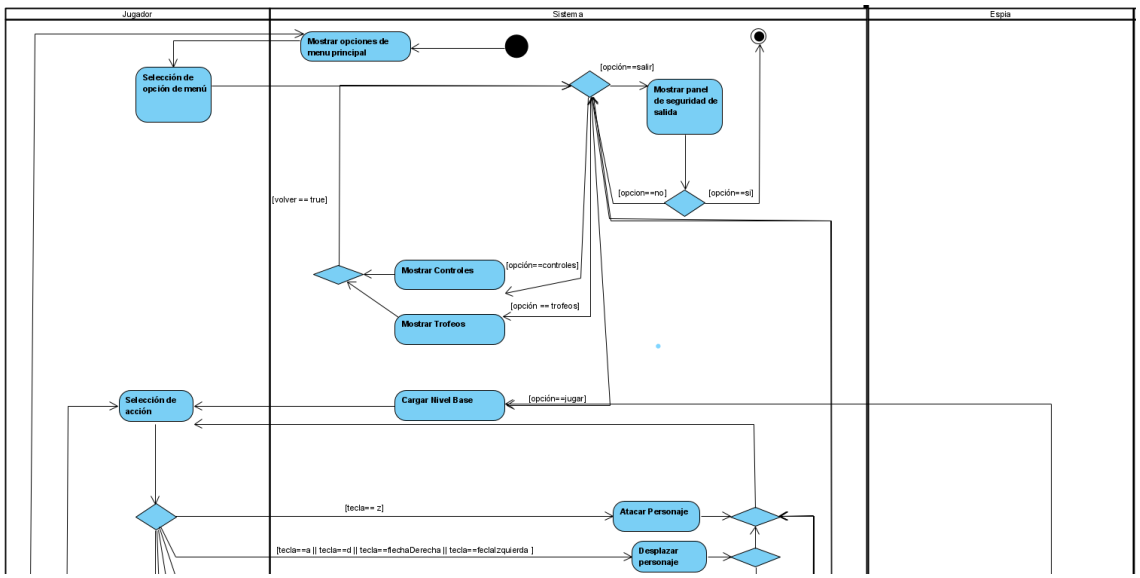

Figura 4.7: Diagrama de actividad del sistema - Primera Parte.

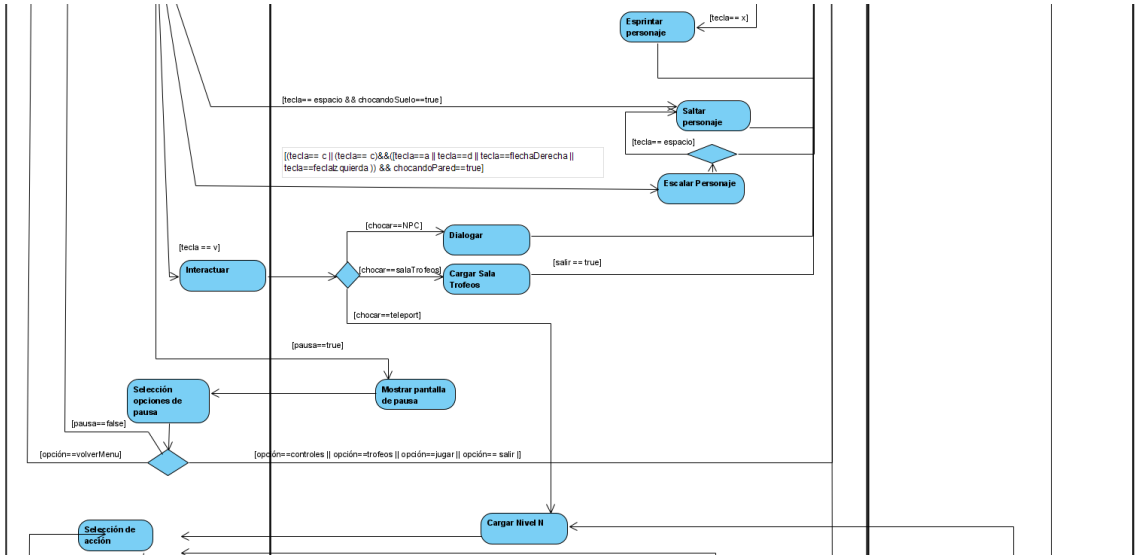

Figura 4.8: Diagrama de actividad del sistema - Segunda Parte.

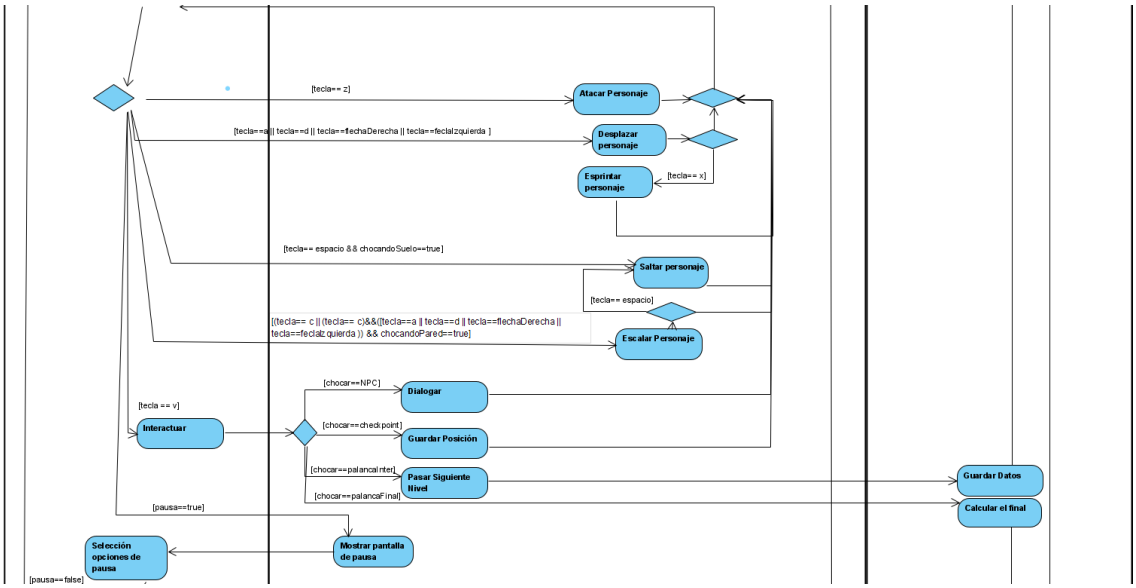

Figura 4.9: Diagrama de actividad del sistema - Tercera Parte.

4.3. Diseño

En este apartado, se presenta el diseño del trabajo, abarcando aspectos como la interfaz de usuario, el método de guardado del progreso y las diferentes clases que conforman la base de datos del programa en su totalidad.

Comenzaremos por explicar el diseño de la interfaz de usuario, ya que desempeña un papel crucial. Para obtener más información detallada y comprender a fondo los detalles y la razón detrás de dicho diseño, se podrá observar en el punto 4.4.

A continuación, exploraremos el método de guardado del proyecto y las consideraciones que se tuvieron en cuenta al diseñarlo. Además, examinaremos las diferentes clases que forman parte del programa y su interacción para lograr el funcionamiento deseado.

En resumen, este apartado ofrece una visión detallada y exhaustiva del análisis llevado a cabo en cuanto al diseño de la interfaz de usuario, el método de guardado del proyecto y las clases fundamentales del programa.

4.3.1. Diseño de interfaces de usuario

Figura 4.10: Diagrama de UI de Salida.

En el panel de interfaz de salida, se ha tenido en cuenta la posibilidad de que el jugador pueda salir accidentalmente de una sección o pantalla importante. Para evitar confusiones y facilitar la navegación, se ha diseñado la interfaz con preguntas y botones de opciones de gran tamaño y con una tipografía clara. De esta manera, se busca asegurar que el jugador pueda identificar fácilmente las opciones disponibles y evitar acciones no deseadas al interactuar con la interfaz.

Figura 4.11: Diagrama de UI de Carga.

Sobre el panel de carga, hemos considerado la variabilidad de características de los ordenadores en los que se podría utilizar el juego. Con el fin de mejorar la experiencia del usuario, se ha implementado un panel de carga que se muestra mientras el nivel se carga por completo. Esto permite que los jugadores con equipos menos potentes o con tiempos de carga más prolongados puedan tener una indicación visual de que el proceso de carga está en curso y evitar la confusión o impaciencia que podría generar una pantalla en blanco o estática. De esta manera, se busca brindar una experiencia más fluida y amigable para todos los usuarios, independientemente de las características de sus dispositivos.

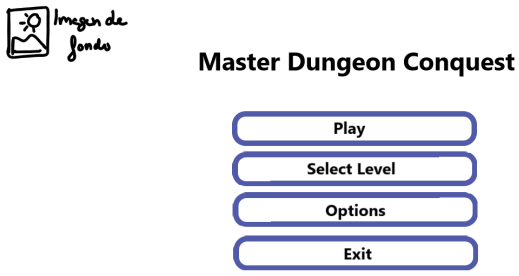

Figura 4.12: Diagrama de UI de Incio.

En el panel inicio del propio juego, hemos adoptado la estrategia de imitar la estructura inicial presente en otros juegos ya creados. En esta estructura, se suele incluir un fondo que refleja el estilo visual del juego, seguido de las opciones que estarán disponibles dentro del inicio. Esta elección se basa en la idea de proporcionar a los jugadores una introducción coherente y familiar, estableciendo desde el principio la estética y las funcionalidades que encontrarán a lo largo de su experiencia de juego. Al emular esta estructura comúnmente utilizada, buscamos facilitar la comprensión y la inmersión del jugador en el entorno del juego desde el momento en que inicia la aplicación.

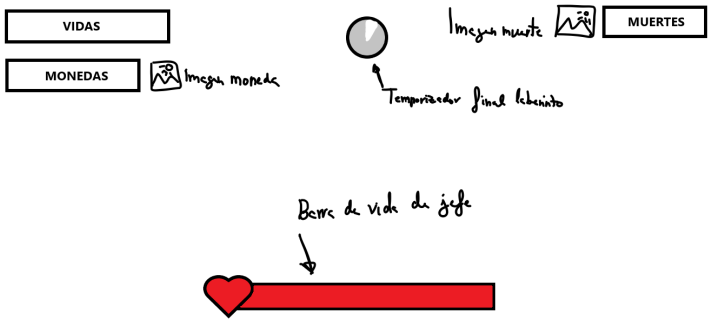

Figura 4.13: Diagrama de UI de Juego.

El panel de juego implementa un sistema de control para gestionar los distintos elementos de la interfaz durante el transcurso del juego. A través de eventos, activaremos los paneles correspondientes según las situaciones que se presenten. Por ejemplo, se mostrará la barra de vida cuando el jugador se enfrente a un jefe, o se mostrará un temporizador cuando se inicie una cuenta regresiva en un nivel con límite de tiempo.

Además, hemos decidido mantener algunos elementos estáticos y siempre visibles en la interfaz, como las vidas disponibles, el número de monedas recolectadas y el recuento de muertes. Sin embargo, nos aseguramos de equilibrar la cantidad de información mostrada para evitar una interfaz sobrecargada. Si la interfaz se vuelve demasiado compleja,

consideraremos omitir ciertos elementos, como el recuento de muertes, para mantener una experiencia visual más limpia y centrada en los aspectos principales del juego.

Figura 4.14: Diagrama de UI de Muerte.

Teniendo en cuenta las funciones que normalmente están presentes en las interfaces de muerte, hemos diseñado este panel con el fin de que el jugador pueda elegir entre continuar desde el último punto guardado, reiniciar el nivel desde el principio o salir rápidamente de la partida en caso de que así lo decida el jugador.

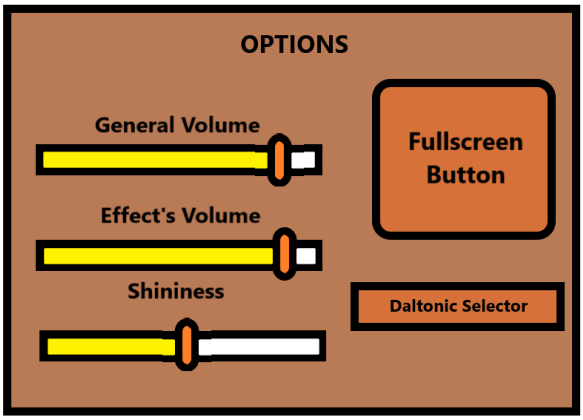

Figura 4.15: Diagrama de UI de Opciones.

En el panel de las opciones, hemos incluido posibilidades que permiten ajustar aspectos clave de la calidad audiovisual del videojuego. Estas opciones incluyen el control del volumen de sonido, permitiendo ajustar tanto el sonido ambiente como los efectos especiales. También se ha incorporado la posibilidad de ajustar el brillo tanto de la escena como de la interfaz. Además, se ha incluido una opción para modificar el tamaño de pantalla y un

modo especial para aquellos que padecen problemas de daltonismo, permitiendo ajustar la gama de colores según las principales ramas de esta enfermedad.

Figura 4.16: Diagrama de UI de Diálogo.

En la interfaz de dialogo, hemos optado por la opción de un panel donde se muestra una imagen de referencia junto con el diálogo. De esta manera, el jugador podrá identificar fácilmente qué personaje del juego está hablando.

4.3.2. Diseño de base de datos

Los principales métodos para guardar y recuperar datos en Unity son los siguientes:

- Serialización binaria: Permite conservar el estado de cualquier objeto o clase, y es útil para almacenar grandes cantidades de datos de manera masiva.

- Serialización JSON: Convierte objetos a formato JSON y permite el almacenamiento masivo de datos, al igual que la serialización binaria.

- PlayerPrefs: Almacena y accede a las preferencias de los jugadores entre sesiones de juego. A diferencia de los dos métodos anteriores, solo puede almacenar datos individuales de tipo “int“, “float“, “bool“ o “string“.

La implementación de cada uno de estos métodos sigue un procedimiento similar, pero con diferentes métodos. A continuación, se presentarán ejemplos de los dos métodos más utilizados durante nuestras clases de desarrollo de videojuegos:

Serialización JSON:

```
// Para guardar los datos del personaje
string playerJson = JsonUtility.ToJson(myPersonaje);
string path = Path.Combine(Application.persistentDataPath, FileName);
File.WriteAllText(path, playerJson);
// Para leer los datos del personaje
string path = Path.Combine(Application.persistentDataPath, FileName);
string playerJson = File.ReadAllText(path);
```

```
myPersonaje = JsonUtility.FromJson<clasePersonaje>(playerJson);
```

Código 4.1: Funciones de gurdado de Serialización JSON

PlayerPrefs:

```
PlayerPrefs.SetInt("vidas", numVidas); // Para crear una clave de guardado
int vidas = PlayerPrefs.GetInt("vidas"); // Accede al contenido guardado
bool existeClave = PlayerPrefs.HasKey("vidas"); // Comprobar si existe la
    clave
PlayerPrefs.DeleteKey("vidas"); // Borra la clave y se pierde el dato
    guardado
```

Código 4.2: Funciones de gurdado de PlayerPrefs

Entre estos métodos, el que nos resulta más fácil y cómodo de usar en proyectos relacionados con videojuegos es PlayerPrefs. Tomamos esta decisión al comienzo del proyecto para asegurarnos de que los datos se fueran guardando a medida que avanzábamos y así poder realizar un seguimiento en caso de que alguna dinámica del juego presentara errores con los datos almacenados.Posteriormente consideraríamos cambiar el método de guardado de PlayerPrefs a la serialización JSON.

En resumen, tenemos una base de datos flexible que nos permite modificar el método de guardado a lo largo del tiempo. A continuación, mostramos el diagrama de modelo de datos que utilizaríamos.

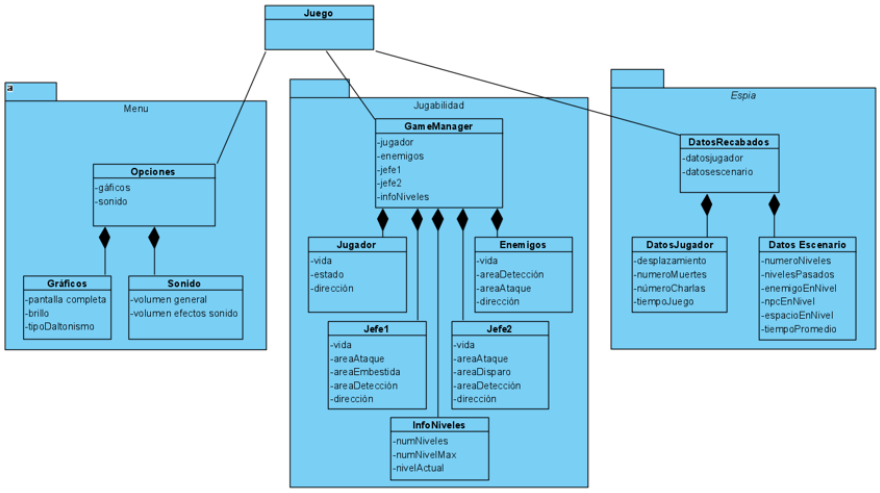

Figura 4.17: Diagrama del modelo de datos.

4.3.3. Diagrama de clases

Después de realizar el análisis de los datos que se utilizarían en nuestro proyecto y como se guardarían dentro de nuestro proyecto, identificamos las clases necesarias para su

funcionamiento y las dependencias entre ellas. Estas son las clases que hemos definido:

- GameManager: esta clase agrupa la mayoría de las clases y tiene información global del proyecto. Es responsable de garantizar la seguridad de los datos guardados y controlar el acceso a ellos por parte de otras clases con privilegios.

- Jugador: en esta clase se almacena la información relacionada con el personaje principal, como la dinámica del juego, la cantidad de vidas en un punto de guardado temporal y la cantidad de monedas recolectadas.

- Enemigos simples: esta clase almacena información sobre los diferentes tipos de enemigos y, en el caso de enemigos que siguen un circuito, también guarda los puntos de desplazamiento. Para los enemigos inmóviles, se almacena la distancia de detección y ataque en ciertos casos. Cuando un enemigo pierde todas sus vidas, se registra esta información para calcular el nivel final.

- Jefes: esta clase almacena información similar a los enemigos simples, pero está diseñada para los jefes finales del juego. Además de los atributos que poseen, también se definen los movimientos de ataque, las distancias y los eventos que los activan. Al perder todas sus vidas, el jefe permite pasar al siguiente nivel y se cierra la zona de combate.

- Cámara: esta clase almacena la posición de la cámara y los límites de cada nivel. Dado que la animación es en 2D, no se registran rotaciones.

- Interactuables: esta clase contiene información sobre los objetos con los que el jugador puede interactuar. Almacena el tipo de interacción, los eventos que se activan al interactuar y si ya se ha utilizado anteriormente.

- Opciones: esta clase es la base para clases más específicas que afectan a la calidad audiovisual del juego.

- Gráficos: esta clase almacena valores relacionados con el tamaño de la pantalla, los esquemas de colores para abordar el daltonismo y el brillo de la escena. Además de almacenar estos valores, permite cambios dinámicos en ellos.

- Sonido: esta clase almacena valores relacionados con el volumen general de la escena y los efectos de sonido especiales. Al igual que la clase *Gráficos*, también permite cambios dinámicos en estos valores.

- Espía: en esta clase que a partir de la información almacenada del personaje principal, a la hora de llegar al nivel final, determinará cual de los niveles se debería de generar para que encaje con la personalidad del usuario.

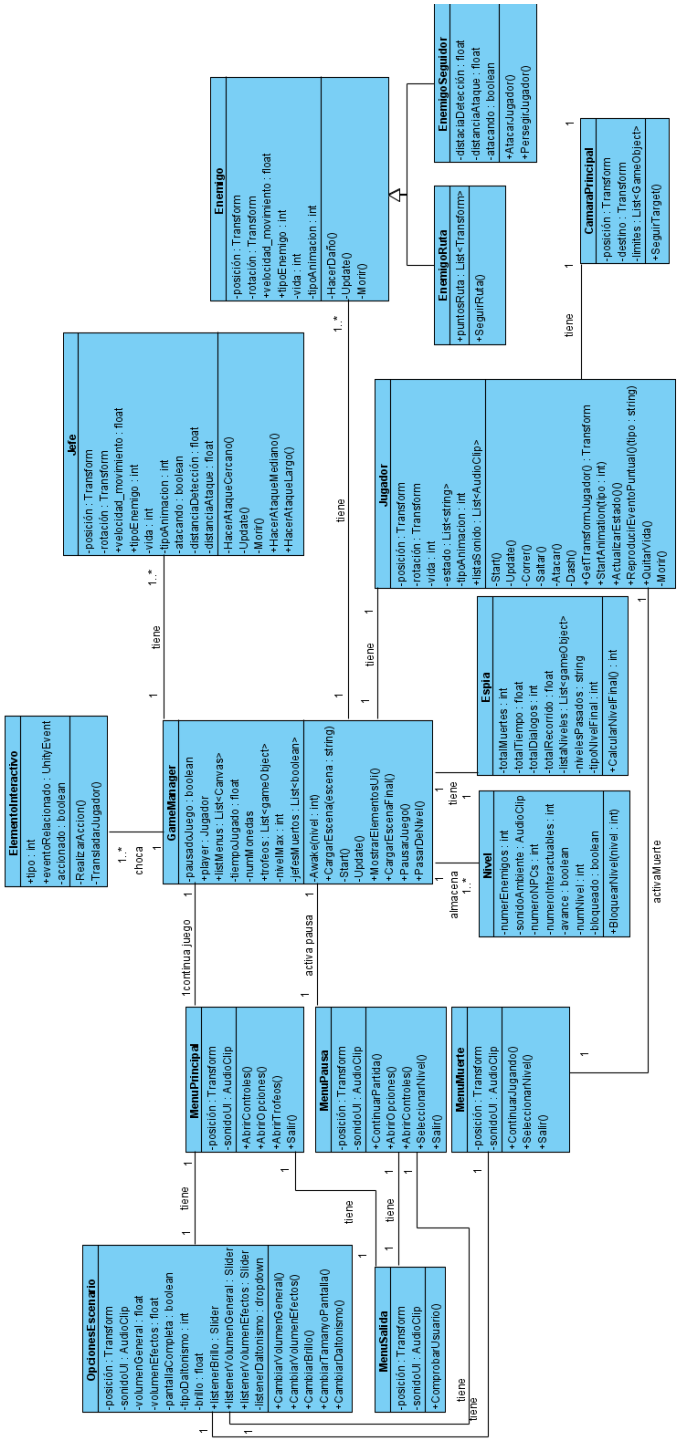

Figura 4.18: Diagrama de clases del sistema.

OpcionesEscenario

Descripción	
En esta clase implementamos el comportamiento del menú del cambio de variables audiovisuales del juego	
Atributos	
posición	Variable que guarda la posición que tendría en pantalla su panel
sonidoUI	Variable que guarda el sonido de usar la interfaz del usuario
volumenGeneral	Variable sobre el nivel del volumen del sonido ambiente
volumenEfectos	Variable sobre el nivel del volumen de los efectos
pantallaCompleta	Variable para comprobar si esta activa la pantalla completa
tipoDaltonismo	Variable que indica el tipo de daltonismo seleccionado
brillo	Variable sobre el nivel del brillo que se utiliza en el juego
listenerBrillo	Variable del elemento que recoge el valor del brillo
listenerVolumenGeneral	Variable del elemento que recoge el volumen general
listenerVolumenEfectos	Variable del elemento que recoge el volumen de los efectos
listenerDaltonismo	Variable del elemento que recoge el tipo de daltonismo
Operaciones	
CambiarVolumenGeneral	Función que capta, guarda y modifica el volumen general
CambiarVolumenEfectos	Función que capta, guarda y modifica el volumen de los efectos
CambiarBrillo	Función que capta, guarda y modifica el brillo
CambiarTamanyoPantalla	Función que cambia de forma alterna entre tamaños de pantalla
CambiarDaltonismo	Función que capta, guarda y modifica el tipo de daltonismo

Cuadro 4.14: Descripción de la clase “OpcionesEscenario“.

MenuSalida

Descripción	
En esta clase implementamos el comportamiento del menú que se asegura de la acción de salida del usuario	
Atributos	
posición	Variable que guarda la posición que tendría en pantalla su panel
sonidoUI	Variable que guarda el sonido de usar la interfaz del usuario
Operaciones	
ComprobarUsuario	Función que se asegura de que el usuario a realizado la acción que quería realizar

Cuadro 4.15: Descripción de la clase “MenuSalida“.

MenuPrincipal

Descripción	
En esta clase implementamos el comportamiento del menú del inicio del juego	
Atributos	
posición	Variable que guarda la posición que tendría en pantalla su panel
sonidoUI	Variable que guarda el sonido de usar la interfaz del usuario
Operaciones	
AbrirControles	Función que activa el panel donde se muestran los controles
AbrirOpciones	Función que activa el panel donde se muestran las opciones
AbrirTrofeos	Función que activa el panel donde se muestran los trofeos
Salir	Función que permite acabar con el juego, pero que manda al "MenuSalida" para comprobar tu acción

Cuadro 4.16: Descripción de la clase "MenuPrincipal".

MenuPausa

Descripción	
En esta clase implementamos el comportamiento del menú de pausa del juego	
Atributos	
posición	Variable que guarda la posición que tendría en pantalla su panel
sonidoUI	Variable que guarda el sonido de usar la interfaz del usuario
Operaciones	
ContinuarPartida	Función para desactivar el panel de pausa
AbrirOpciones	Función que activa el panel donde se muestran las opciones
AbrirControles	Función que activa el panel donde se muestran los controles del juego
SeleccionarNivel	Función que vuelve al nivel básico donde se puede elegir nivel
Salir	Función que permite acabar con el juego, pero que manda al "MenuSalida" para comprobar tu acción

Cuadro 4.17: Descripción de la clase "MenuPausa".

MenuMuerte

Descripción	
En esta clase implementamos el comportamiento del menú del inicio del juego	
Atributos	
posición	Variable que guarda la posición que tendría en pantalla su panel
sonidoUI	Variable que guarda el sonido de usar la interfaz del usuario
Operaciones	
ContinuarJugando	Función que restaura el juego hasta el último punto de guardado
SeleccionarNivel	Función que vuelve al nivel básico donde se puede elegir nivel
AbrirTrofeos	Función que activa el panel donde se muestran los trofeos
Salir	Función que permite acabar con el juego, pero que manda al "MenuSalida" para comprobar tu acción

Cuadro 4.18: Descripción de la clase "MenuMuerte".

ElementoInteractivo

Descripción	
En esta clase implementamos el comportamiento de los objetos con los que puede interactuar el usuario con los controles	
tipo	Variable que indica el tipo de interactuable que se trata
eventoRelacionado	Variable que almacena las acciones que tendrá que realizar al interaccionar con el objeto
accionado	Variable que muestra si se activó ya la interacción
Operaciones	
RealizarAccion	Función que ejecuta el evento relacionado con el tipo de objeto
TransladarJugador	Función que permite desplazar a un personaje de un lado a otro del interaccionable

Cuadro 4.19: Descripción de la clase "ElementoInteractivo".

GameManager

Descripción	
En esta clase implementamos el comportamiento del menú del inicio del juego	
Atributos	
pausadoJuego	Variable que indica si el tiempo se a parado
player	Variable que permite acceder a las variables del jugador
listMenus	Variable que guarda los menús que puede usar al momento
tiempoJugado	Variable que cuenta el tiempo jugado en una partida completa
numMonedas	Variable que almacena el número de monedas
trofeos	Variable que almacena los tipos de trofeos que hay en el juego
nivelMax	Variable que almacena el nivel máximo en una partida para mostrar después los niveles que puede elegir
jefesMuertos	Variable que indica las salas de jefe que se han completado y bloqueado
Operaciones	
Awake	Función que instancia con prioridad el manejador del juego
Start	Función que instancia los atributos
Update	Función que actualiza y hace uso de sus atributos a medida de que el jugador va interactuando con el mapa
CargarEscena	Función que permite teletransporte a un nivel seleccionado
MostrarElementosUi	Función que permite ver de forma dinámica los elementos UI que representan las estadísticas del jugador
PausarJuego	Función para parar el tiempo
PasarDeNivel	Función para pasar de un nivel a otro nivel vecino
CargarEscenaFinal	Función de activación del nivel final

Cuadro 4.20: Descripción de la clase "GameManager".

Nivel

Descripción	
En esta clase implementamos el comportamiento de los niveles y los datos que identifican a cada uno	
Atributos	
numerEnemigos	Variable que almacena el número de enemigos
sonidoAmbiente	Variable que indica el audio que usa de fondo
numeroNPCs	Variable que almacena el número de personajes pacifistas
numeroInteractuables	Variable que almacena el número de objetos interactuable
avance	Variable que indica si se avanza en un nivel o se retrocede
numNivel	Variable que identifica el nivel
bloqueado	Variable que indica si está bloqueado un nivel
Operaciones	
BloquearNivel	Función que bloquea el paso a un nivel de jefe

Cuadro 4.21: Descripción de la clase "Nivel".

Espía

Descripción	
En esta clase implementamos el comportamiento del calculador de la parte innovadora de la creación de niveles	
Atributos	
totalMuertes	Variable que indica el número de asesinatos del jugador
totalTiempo	Variable que indica el tiempo que tardó en llegar hasta el final
totalDialogos	Variable que indica el número de veces que ha dialogado en el juego
totalRecorrido	Variable que indica cuanta distancia ha recorrido el jugador
listaNiveles	Variable que indica los niveles del juego
tipoNivelFinal	Variable que almacena por los niveles que ha viajado el jugador
Operaciones	
CalcularNivelFinal	Función que a partir de los atributos de la clase, calcula cual nivel se debe desbloquear

Cuadro 4.22: Descripción de la clase "Espía".

Jefe

Descripción	
En esta clase implementamos el comportamiento de los enemigos más complejos del juego	
Atributos	
posición	Variable que indica la posición donde se encuentra el jefe
rotación	Variable que indica el sentido del jefe
velocidad_movimiento	Variable que indica la velocidad a la que se desplaza el jefe
tipoEnemigo	Variable que indica de cual jefe se trata
vida	Variable que indica cuanto le queda de vida al jefe
tipoAnimacion	Variable que indica cual animación se debe de ejecutar al momento
atacando	Variable que indica que ya está atacando el jefe y evitar que ataque mientras está atacando
distanciaDetección	Variable que indica a la distancia donde te persigue el jefe
distanciaAtaque	Variable que indica a la distancia donde te ataca el jefe
Operaciones	
HacerAtaqueCercano	Función que realiza las acciones para que el jefe ataque a corta distancia
HacerAtaqueMediano	Función que realiza las acciones para que el jefe ataque a mediana distancia
HacerAtaqueLargo	Función que realiza las acciones para que el jefe ataque a larga distancia
Update	Función que se encarga de comprobar si detecta al jugador y que decisiones ha de tomar
Morir	Función donde el jefe desaparece y deja caer una llave

Cuadro 4.23: Descripción de la clase “Jefe“.

Enemigo

Descripción	
En esta clase padre implementamos el comportamiento de los enemigos simples del juego	
Atributos	
posición	Variable que indica la posición donde se encuentra el enemigo
rotación	Variable que indica el sentido del enemigo
velocidad_movimiento	Variable que indica la velocidad a la que se desplaza el enemigo
tipoEnemigo	Variable que indica de cual enemigo se trata
vida	Variable que indica cuanto le queda de vida al enemigo
tipoAnimacion	Variable que indica cual animación se debe de ejecutar al momento
Operaciones	
HacerDaño	Función que permite al enemigo dañar al jugador para restarle vida
Update	Función que se encarga de comprobar si detecta al jugador y que decisiones ha de tomar
Morir	Función donde el enemigo desaparece y hace que sume el contador de asesinatos del jugador

Cuadro 4.24: Descripción de la clase “Enemigo“.

EnemigoRuta

Descripción	
En esta clase hija de “Enemigo” implementamos el comportamiento de los enemigos que siguen un patrón o camino	
Atributos	
puntosRuta	Variable que indica los puntos por donde se cruzara el enemigo por el mapa
Operaciones	
SeguirRuta	Función que traslada de forma uniforme al enemigo por los puntos

Cuadro 4.25: Descripción de la clase “EnemigoRuta“.

EnemigoSeguidor

Descripción	
En esta clase hija de "Enemigo" implementamos el comportamiento de los enemigos que solo persigue al jugador al detectarlo	
Atributos	
distaciaDetección	Variable que indica a la distancia donde te persigue el enemigo
distanciaAtaque	Variable que indica a la distancia donde te ataca el enemigo
atacando	Variable que indica que ya está atacando el enemigo y evitar que ataque mientras está atacando
Operaciones	
AtacarJugado	Función que permite al enemigo atacar en la dirección donde se encuentra el jugador
PersegirJugador	Función que traslada al enemigo hacia la posición donde se encuentra el jugador

Cuadro 4.26: Descripción de la clase "EnemigoSeguidor".

Jugador

Descripción	
En esta clase implementamos el comportamiento del jugador que controla nuestro usuario	
Atributos	
posición	Variable que indica la posición del personaje
rotación	Variable que indica la dirección a la que se dirige el personaje
vida	Variable que indica la vida del personaje
estado	Variable que indica el estado en el que se encuentra nuestro personaje
tipoAnimacion	Variable que indica la animación que debe realizar al momento
listaSonido	Variable que almacena los sonidos que puede realizar el personaje
Operaciones	
Start	Función que permite instanciar los atributos de la clase
Update	Función que se encarga de detectar los eventos de teclado que activan otras funciones y comprobar que está vivo nuestro personaje
Correr	Función que permite desplazarse hacia los laterales
Saltar	Función que permite subir de forma progresiva en vertical o diagonal
Atacar	Función que permite realizar daño a enemigos
Dash	Función que hace un desplazamiento rápido del jugador
GetTransformJugador	Función que permite a otras clases acceder a la posición del jugador
StartAnimation	Función que inicializa nuestras animaciones automatizadas por interacciones
ActualizarEstado	Función que permite cambiar el estado de nuestro personaje
ReproducirEventoPuntual	Función que permite reproducir una grupo de acciones al mismo tiempo relacionada con la escena
QuitarVida	Función que quita una vida del jugador
Morir	Función que muestra el panel de muerte de nuestro jugador

Cuadro 4.27: Descripción de la clase "Jugador".

CamaraPrincipal

Descripción	
En esta clase implementamos el comportamiento de la única cámara del juego	
Atributos	
posición	Variable que indica la posición dentro del juego
destino	Variable que indica cual es el objetivo a seguir
limites	Variable que almacena el límite de la cámara para los diferentes niveles
Operaciones	
SeguirTarger	Función que traslada la cámara hacia una posición donde el destino este centrado y sin realizar cambios bruscos

Cuadro 4.28: Descripción de la clase "CamaraPrincipal".

4.3.4. Diagramas de secuencia

Después de revisar las descripciones de las clases que podrían formar parte del videojuego, presentaremos ahora, para cada uno de los casos de uso mencionados anteriormente, un diagrama de secuencia que representa las principales acciones que el usuario puede realizar en nuestro proyecto.

Cabe destacar que todas estas acciones se centran en la plataforma de PC, por lo que al explicar las interacciones del usuario con el sistema, tendremos en cuenta este factor y cómo el sistema responderá en consecuencia.

A continuación, se muestra el diagrama de secuencia que ilustra las interacciones entre el usuario y el sistema en diferentes escenarios.

Para el caso de uso "Jugar Partida":

El siguiente diagrama describe los diferentes decisiones que el usuario puede tomar al comenzar o continuar la partida y los principales controles que tiene sobre el personaje.

- Al utilizar "moverJugador()", se requiere el evento de las teclas correspondientes a los movimientos.
- Al utilizar "atacar()", es importante la dirección en la que se encuentra el jugador y que se pulse la tecla adecuada.
- Al utilizar "escalar()" o "saltar()", depende de si hay colisión con paredes o suelo y que se utilice el evento de tecla adecuado.
- Al utilizar "dash[1]()", es necesario haber utilizado previamente "moverJugador()".

Las demás opciones se detallan claramente en el diagrama o en las referencias correspondientes.

[1]Término utilizado en el contexto de los videojuegos para referirse a un movimiento rápido o una habilidad que permite al jugador desplazarse rápidamente en una dirección específica.

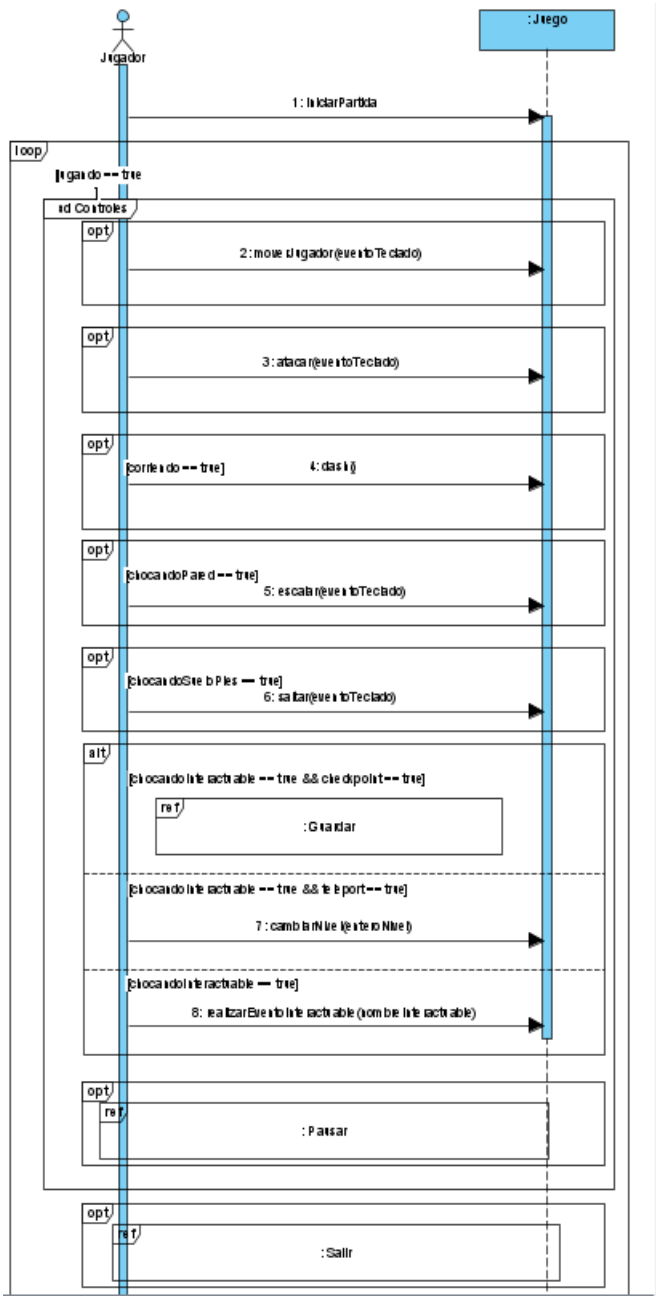

Figura 4.19: Diagrama de secuencia de "Jugar Partida".

Para el caso de uso "Mirar Trofeos":

Este diagrama describe la forma en la que el jugador puede acceder a la visualización de los trofeos desde cualquier punto del juego. El jugador tiene la posibilidad de ingresar a la sección de trofeos para ver su progreso y los logros desbloqueados. Una vez dentro de esta sección, el jugador puede explorar los diferentes trofeos y sus descripciones.

El diagrama de secuencia correspondiente ilustrará de manera más detallada las interacciones y pasos necesarios para realizar esta acción en el juego.

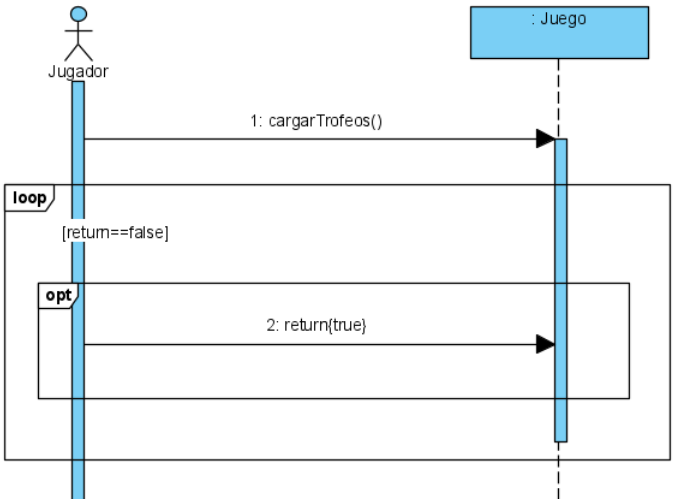

Figura 4.20: Diagrama de secuencia de "Mirar Trofeos".

Para el caso de uso "Salir":

Este diagrama describe el proceso de salida del juego y presenta una confirmación al jugador para asegurarse de que desea abandonar completamente la partida, para evitar pérdidas involuntarias de datos.

- Al utilizar la función "volverJuego()", el juego regresa a la pantalla desde la cual se activó la acción de salida. Esta opción permite al jugador reconsiderar su decisión y continuar jugando sin abandonar por completo el juego.

- Al utilizar la función "cerrarSesión()", el sistema se cierra por completo y se pierde cualquier progreso o datos que no hayan sido guardados previamente. Esta opción debe ser utilizada con precaución, ya que implica una desconexión total del juego y no permite la recuperación de información no guardada.

El diagrama de secuencia correspondiente proporcionará una representación más detallada de las interacciones y pasos necesarios para llevar a cabo estas acciones en el juego.

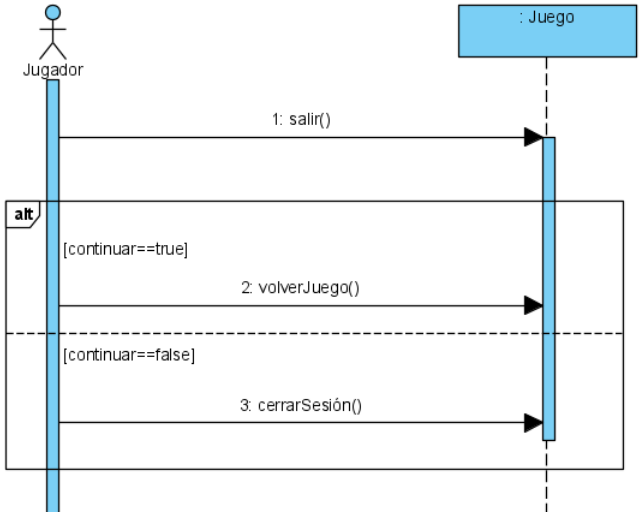

Figura 4.21: Diagrama de secuencia de "Salir".

Para el caso de uso "Pausar":

Este diagrama describe las opciones disponibles en el menú de pausa y cómo se relacionan con otros diagramas de secuencia.

- Al seleccionar la opción "Continuar", el jugador puede reanudar el juego desde el punto exacto en el que se pausó. Esta opción permite al jugador continuar la partida sin interrupciones y retomar el flujo del juego.

- Al seleccionar la opción "volverMenuPrincipal", el jugador vuelve de forma inmediata, al menú que se muestra al iniciar la aplicación.

- Al seleccionar la opción "Acceder controles", se muestra al jugador una pantalla con los controles y las instrucciones del juego. Esta opción brinda al jugador la posibilidad de recordar o aprender los comandos necesarios para jugar.

- Al seleccionar la opción "Acceder Opciones", se accede a un menú donde se pueden modificar las configuraciones del juego, como el volumen y la calidad gráfica. Esta opción permite al jugador personalizar la experiencia de juego de acuerdo a sus preferencias.

- Al seleccionar la opción "Salir", se muestra una confirmación al jugador para asegurarse de que desea abandonar la partida y salir del juego.

El diagrama de secuencia que se muestra a continuación destaca por el bucle que detiene el tiempo hasta que el usuario realice alguna acción o detenga la pausa.

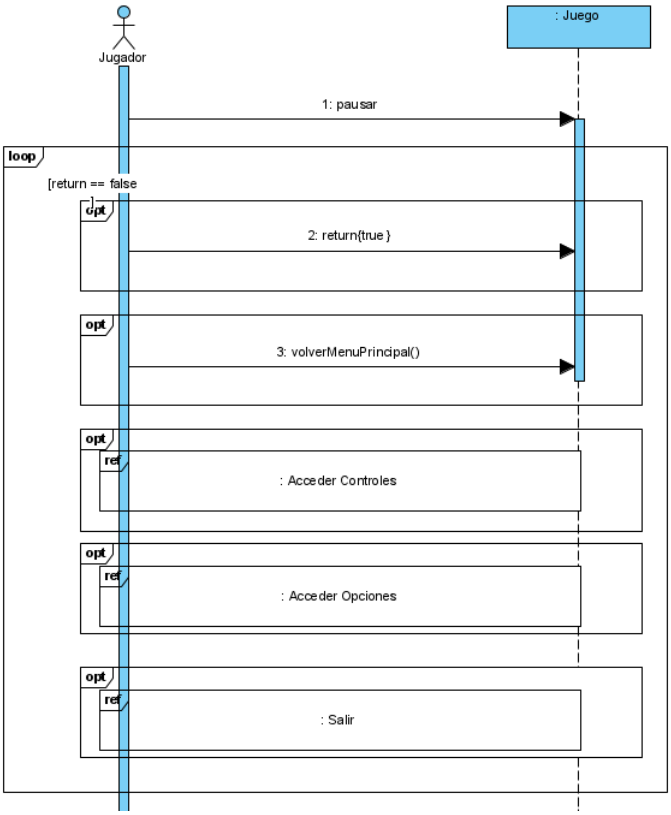

Figura 4.22: Diagrama de secuencia de "Pausar".

Para el caso de uso "Acceder Controles":

Este diagrama describe la forma en que el jugador puede acceder a la pantalla de controles desde cualquier punto del juego. Además, se muestra la opción de regresar al punto anterior al salir del bucle formado al acceder a los controles.

En el diagrama de secuencia se ilustra la interacción del usuario con el sistema para acceder a la pantalla de controles. Una vez dentro, el jugador puede revisar la información de los controles y luego salir de la pantalla para regresar al punto del juego donde se encontraba previamente.

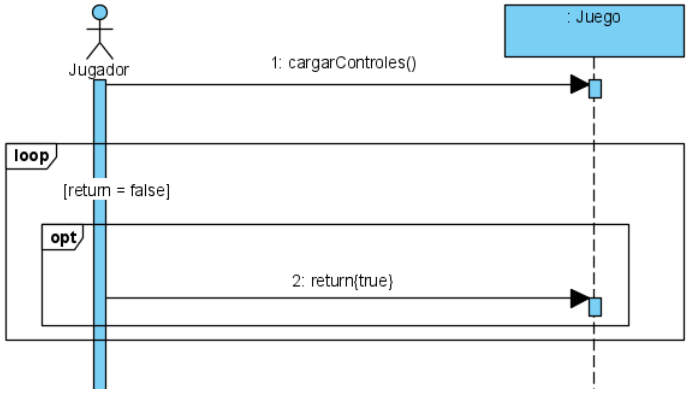

Figura 4.23: Diagrama de secuencia de "Acceder Controles".

Para el caso de uso "Acceder Opciones":

En este diagrama, se muestran las acciones necesarias para acceder y modificar los elementos audiovisuales del juego. A continuación se presentan las secuencias de acciones que se llevan a cabo en este caso de uso:

- Cargar Opciones: El jugador seleccionó la opción "Opciones" en el menú principal o durante el juego y el sistema muestra la interfaz de opciones.

- Modificar Volumen: El sistema muestra los controles deslizantes para ajustar el volumen general y los efectos de sonido y el jugador modifica los deslizadores para ajustar el volumen según sus preferencias, además de guarda los nuevos valores de volumen.

- Modificar Brillo: El sistema muestra el control deslizante para ajustar el brillo de la pantalla y el jugador modifica el deslizador para ajustar el brillo según sus preferencias, además de guarda los nuevos valores de brillo.

- Cambiar Opciones de Daltonismo: En este el sistema muestra con un desplegable las diferentes opciones de corrección de color para jugadores con daltonismo y después el jugador elige la opción que mejor se adapte a sus necesidades, para que esta sea guardada por el sistema posteriormente.

- Activar/Desactivar Pantalla Completa: El sistema muestra una casilla de verificación para activar o desactivar el modo de pantalla completa y según sea el estado de la marca, se guardará en el estado de la pantalla completa en el sistema.

Una vez que el jugador ha realizado los ajustes deseados, puede volver al juego o realizar otras modificaciones en las opciones.

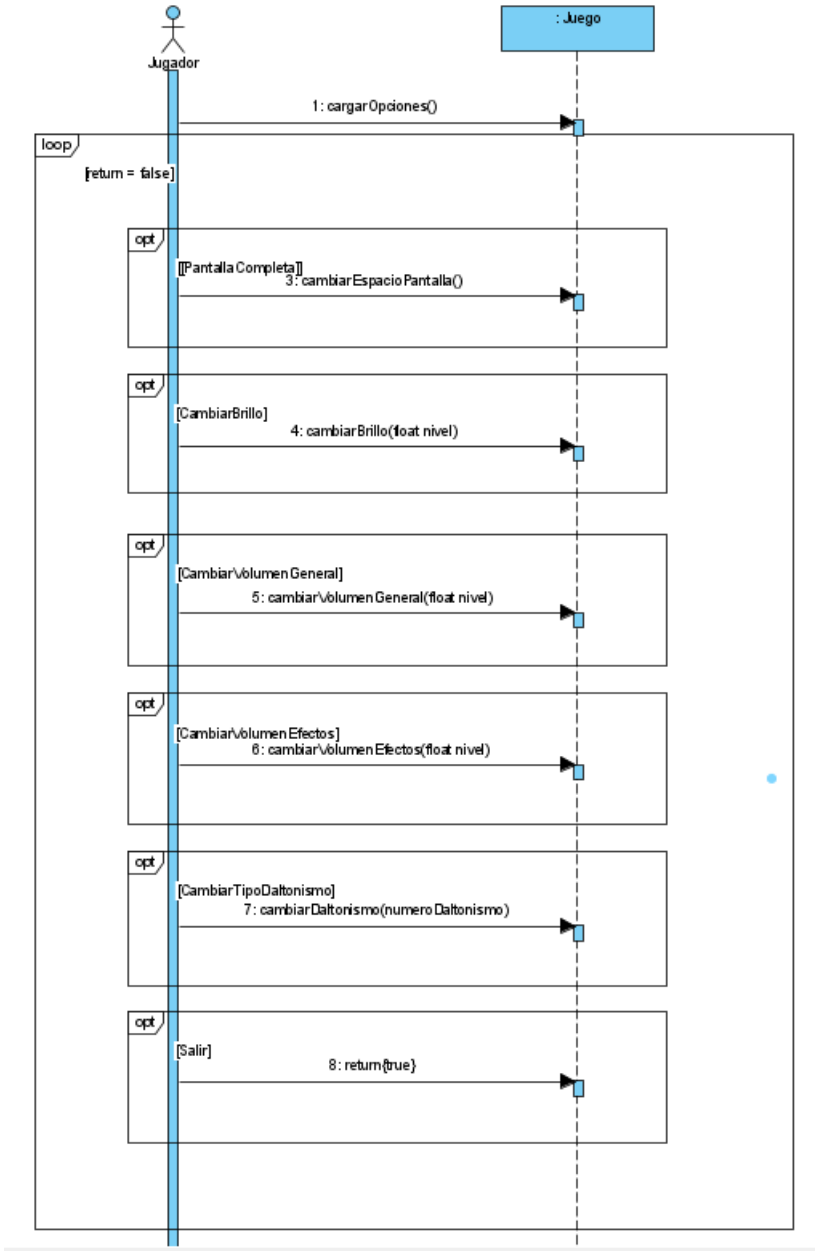

Figura 4.24: Diagrama de secuencia de "Acceder Opciones".

Para el caso de uso "Guardar":

Para el caso de uso 'Guardar':

Este diagrama describe las diferentes formas automáticas o interactivas de guardar

datos en nuestro juego. A continuación se presentan las secuencias de acciones que se llevan a cabo en este caso de uso:

- Guardar posición: El sistema guarda temporalmente la posición actual del jugador mientras se encuentra dentro de la mazmorra.

- Guardar datos de partida: El sistema guarda los datos del jugador en el sistema de almacenamiento, permitiendo retomar la partida desde el último punto guardado relacionado con la entrada o inicio de un nivel.

Una vez que el jugador ha realizado la acción de guardar, los datos quedan almacenados y pueden ser utilizados posteriormente para cargar la partida guardada. En el siguiente diagrama de secuencia, se ilustra de forma clara las propiedades del caso de uso que representa.

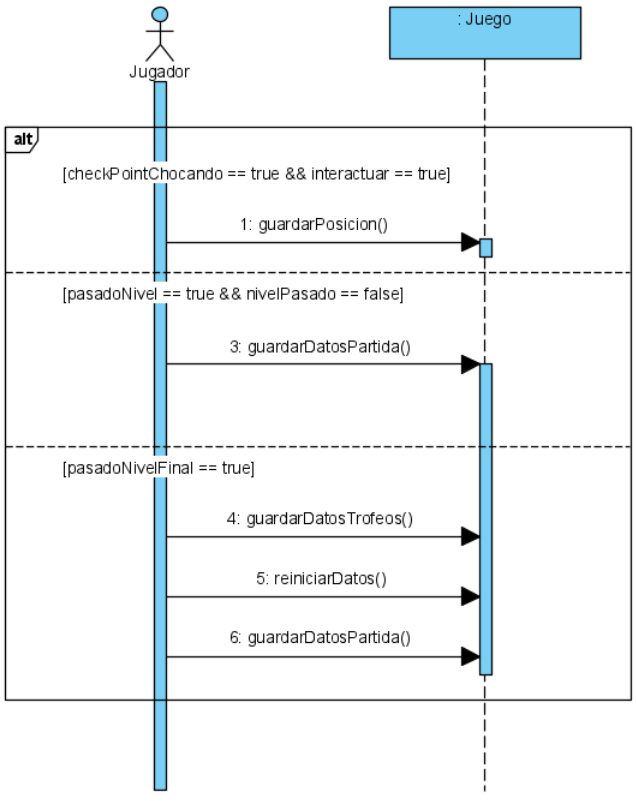

Figura 4.25: Diagrama de secuencia de“Guardar“.

4.4. Implementación

Una vez contempladas las fases del análisis y diseño de la herramienta a desarrollar, en esta sección se detallarán los principales componentes, técnicas y metodologías empleadas en la implementación de este módulo para la generación automática de un final adaptado a la actitud del jugador. Se analizará cómo se integra con el resto del juego, cómo se procesan los datos del jugador y cómo se generan los distintos desenlaces en función de las elecciones realizadas.

La implementación del proyecto abarcará aspectos clave como el diseño de niveles, la programación de la lógica del juego, la creación de activos visuales y sonoros, la integración de efectos especiales y mucho más.

4.4.1. Herramientas de desarrollo

En este subapartado, se brinda información detallada sobre las herramientas seleccionadas, incluyendo su propósito, funcionalidades y ventajas específicas que las hacen adecuadas para el desarrollo del videojuego en cuestión.

Aunque en el estado del arte se pueden haber realizado análisis, este subapartado complementa dichos análisis al proporcionar una descripción detallada de las herramientas específicas utilizadas en el desarrollo del videojuego. Esto brinda una visión más completa y precisa de la implementación del proyecto, permitiendo una comprensión más profunda de cómo se utilizaron las herramientas para lograr los objetivos establecidos.

Unity

Como se menciona en el capítulo del estado del arte (Capítulo 2), para el desarrollo de la "prueba de concepto" de nuestro módulo, se hizo uso de Unity y varias de sus herramientas con el fin de lograr un resultado altamente satisfactorio para el cliente.

Entre los recursos principales utilizados se encuentran los siguientes:

- *Materiales físicos 2D*: son componentes que permiten simular propiedades físicas como la fricción, la densidad, la elasticidad y la colisión en objetos 2D. Estos materiales se aplican a los objetos en el motor de física de Unity y determinan cómo interactúan entre sí y con el entorno. En nuestro proyecto nos sirve para simular la fricción de nuestro personaje con los diferentes terrenos que se utilizan por el mapa del videojuego.

- *Audiomixer*: es una herramienta que permite gestionar y mezclar el audio en un proyecto. Actúa como un controlador centralizado donde se pueden ajustar niveles de volumen, panoramas, efectos y configuraciones de grupos de sonido. El "Audiomixer" ofrece una interfaz intuitiva y visual que permite crear y organizar diferentes grupos de audio, como música, efectos de sonido y diálogos, y asignarles efectos y ajustes individuales.En nuestro proyecto, utilizamos esta herramienta para agrupar la música general y los efectos de sonido, lo que nos facilita la tarea de modificar de forma dinámica los volúmenes en el menú de opciones.

- *Tilesets* y *Rule Tiles*: son herramientas poderosas que se utilizan para la creación y diseño de niveles en juegos 2D. Los "Tilesets" son conjuntos de gráficos predefinidos que contienen diferentes tipos de tiles, como suelos, paredes, obstáculos y elementos decorativos. Estos tiles se colocan en un grid para construir los escenarios del juego de manera rápida y eficiente. Por otro lado, los "Rule Tiles" permiten definir reglas y comportamientos específicos para la colocación automática de los tiles, lo que agiliza el proceso de diseño y evita la repetición manual de colocar tiles similares en ciertas situaciones. En nuestro proyecto, utilizamos estas herramientas para desarrollar nuestros niveles y los diferentes tipos de terreno debido a la facilidad que brindan para generar escenarios en videojuegos en dos dimensiones.

- *VideoPlayer*: es una herramienta que permite reproducir vídeos dentro de los juegos. Con el "Videoplayer", es posible integrar contenido multimedia como cinemáticas, secuencias de introducción o vídeos promocionales en el desarrollo de un juego. Proporciona funciones para controlar la reproducción, como iniciar, pausar, detener y ajustar el volumen del vídeo. En nuestro proyecto, utilizamos esta herramienta para mostrar las escenas de los diferentes finales por los que pasará nuestro jugador.

Además de todo lo mencionado, en nuestro proyecto se hizo uso de Unity debido a la amplia gama de componentes que ofrece para mejorar la calidad del juego y enriquecer las dinámicas de los personajes. Algunos ejemplos de estos componentes son el "Rigidbody 2D" para simular las físicas de los objetos, los "Colliders 2D" para calcular las colisiones con otros elementos del juego, y el "Animator" para crear animaciones basadas en una secuencia de frames y ajustar la velocidad de reproducción. Estos componentes son fundamentales para lograr interacciones realistas y una jugabilidad fluida en el desarrollo del juego.

Visual Studio Code

Como entorno de desarrollo preferido para escribir y editar los scripts en nuestro proyecto de Unity, se decidió utilizar Visual Studio Code por varias razones. En primer lugar, el programa ofrece una amplia gama de características y herramientas que facilitan el proceso de desarrollo. Su interfaz intuitiva y personalizable proporciona una experiencia de codificación eficiente y cómoda.

Además, Visual Studio Code es altamente compatible con C#, el lenguaje de programación principal utilizado en Unity. Proporciona resaltado de sintaxis, sugerencias automáticas, autocompletado y depuración integrada para C#, lo que agiliza el proceso de escritura de código y reduce los errores.

Otra ventaja significativa de Visual Studio Code es su extensibilidad. La plataforma admite una amplia variedad de extensiones, lo que nos permite personalizar aún más nuestro flujo de trabajo de desarrollo. Podemos agregar extensiones específicas para Unity que ofrecen características adicionales, como la integración con el motor de Unity, la depuración remota y la gestión de paquetes. Además, Visual Studio Code proporciona información sobre la popularidad y el creador de cada extensión, lo que nos permite verificar las extensiones más demandadas en este sector y elegir aquellas que mejor se adapten a nuestras necesidades y requisitos de desarrollo. Esta funcionalidad nos ayuda a encontrar y utilizar las herramientas más relevantes y de alta calidad para mejorar nuestra experiencia de desarrollo en Unity.

Photopea

En nuestro proyecto, utilizamos Photopea para crear animaciones de algunos recursos del videojuego. Esta herramienta ofrece funciones similares a programas avanzados como Adobe Photoshop, lo que nos permitió combinar sprites individuales en secuencias de imágenes para obtener animaciones fluidas. Con Photopea, ajustamos fotogramas clave, aplicamos efectos y realizamos modificaciones detalladas en cada cuadro de la animación. La elección de Photopea resultó conveniente y accesible, ya que es una aplicación web gratuita que no requiere instalación. Su interfaz intuitiva y capacidad para trabajar con capas y ajustes nos permitieron crear animaciones de alta calidad que mejoraron la apariencia y dinámica de los recursos en el videojuego.

OpenToonz

Empleamos OpenToonz para generar los videos animados que se presentan al finalizar el último nivel del juego. Con OpenToonz, creamos secuencias de animación detalladas y fluidas que explican los diferentes finales del juego. El proceso incluyó importar elementos gráficos como personajes y fondos, y configurar fotogramas clave para establecer movimientos y acciones específicas. La capacidad de trabajar con capas y realizar ajustes precisos en cada elemento nos permitió superponer fotogramas y obtener el resultado deseado. OpenToonz proporcionó una plataforma sólida y accesible para crear animaciones que muestran el perfil mostrado por el usuario a lo largo de la mazmorra.

Audacity

Audacity lo utilizamos para editar los sonidos recopilados de OpenGameArt.org y adaptarlos a las animaciones creadas en OpenToonz. Con Audacity, pudimos realizar modificaciones como recortar, ajustar el volumen y aplicar efectos especiales a los archivos de sonido para que se ajustaran a las acciones y eventos del juego. También sincronizamos los sonidos con las animaciones correspondientes y trabajamos con múltiples pistas de audio para crear una experiencia auditiva inmersiva. Audacity demostró ser una herramienta poderosa y versátil que nos permitió adaptar los sonidos y mejorar la experiencia de juego.

Visual Paradigm

Visual Paradigm es una herramienta de modelado visual ampliamente utilizada en la ingeniería de software y en el proceso de análisis y diseño de sistemas. En nuestro proyecto, utilizamos esta herramienta para la creación de los diagramas necesarios en las etapas de análisis y diseño.

Con Visual Paradigm, pudimos representar de manera clara y concisa los diferentes aspectos del sistema, como los requisitos, las entidades, las relaciones y los flujos de información. Utilizamos diagramas de casos de uso para identificar las interacciones entre el sistema y los usuarios, diagramas de estado para identificar las fases por la que pasan las entidades del juego, diagrama de actividad para simular la prueba de funcionamiento del sistema con el jugador, diagramas de clases para modelar la estructura y las relaciones entre las clases, y diagramas de secuencia para representar las interacciones entre los diferentes componentes del sistema.

Esta herramienta, al mismo tiempo, nos permitió crear estos diagramas de manera intuitiva y visual, lo que facilitó la comunicación y el entendimiento entre los miembros del equipo de desarrollo y los interesados en el proyecto.

4.4.2. Lenguaje de programación

En este proyecto, para la realización del código se ha hecho uso de C#, uno de los lenguajes de programación más recomendados y ampliamente compatible con el motor de Unity. Unity también proporciona una API específica para C# que facilita la creación de juegos y aplicaciones interactivas.

Aunque existe otro lenguaje compatible con Unity, C# ofrece un alto nivel de interoperabilidad con el motor de Unity, lo que nos permitió acceder y manipular más fácilmente los componentes y funcionalidades del motor.

4.4.3. Componentes externos

En nuestro proyecto, hemos aprovechado la integración de componentes externos para enriquecer las tecnologías utilizadas. Estos componentes externos son herramientas, bibliotecas o servicios desarrollados por terceros que nos permiten agregar funcionalidades adicionales y mejorar la experiencia de nuestros usuarios. En esta sección, exploraremos los componentes externos que hemos integrado y cómo se complementan con las tecnologías que se han utilizado, brindando beneficios adicionales a nuestro proyecto.

Shader de Daltonismo

Un shader[2] que simula el efecto de los distintos tipos de daltonismo en un videojuego. Permite aplicar transformaciones de color a la salida de la cámara en relación del tipo de daltonismo seleccionado en el menú de opciones.

El shader utiliza matrices de cambio de color predefinidas para simular los distintos tipos de daltonismo. Estas matrices se aplican a los colores de la textura original mediante multiplicación. El tipo de daltonismo se especifica mediante la variable ´´type´´ que se pasa desde el script en C#.

Además, consta de un pase en el subshader, que define los programas de vértices y fragmentos . En el programa de vértices, se transforma la posición del vértice y se calculan las coordenadas de textura. En el programa de fragmentos, se lee el color de la textura de entrada, se realiza la multiplicación con la matriz de cambio de color correspondiente y se devuelve el resultado.

Además, se especifican las propiedades del shader, en este caso, la textura principal (_MainTex) que se utiliza como entrada.

```
// Colorblind Effect Unity Plugin 1.0
Shader "Hidden/Wilberforce/Colorblind"
{
  Properties
  {
    _MainTex("Texture", 2D) = "white" {}
  }
  SubShader
  {
    Cull Off ZWrite Off ZTest Always

    Pass
    {
```

[2]Es un pequeño programa con instrucciones para colorear cada píxel renderizado según diversas variables.

```
14        CGPROGRAM
          #pragma target 3.0
16
          sampler2D _MainTex;
18        float4 _MainTex_TexelSize;

20        #pragma vertex vert
          #pragma fragment frag
22
          #include "UnityCG.cginc"
24
          // says which matrix we should use in fragment shader
26        // this is passed from the Csharp script
          uniform int type;
28
          // color-shifting matrices
30        static float3x3 color_matrices[4] = {
            // normal vision - identity matrix
32          float3x3(
              1.0f,0.0f,0.0f,
34            0.0f,1.0f,0.0f,
              0.0f,0.0f,1.0f),
36          // Protanopia - blindness to long wavelengths
            float3x3(
38            0.567f,0.433f,0.0f,
              0.558f,0.442f,0.0f,
40            0.0f,0.242f,0.758f),
            // Deuteranopia - blindness to medium wavelengths
42          float3x3(
              0.625f,0.375f,0.0f,
44            0.7f,0.3f,0.0f,
              0.0f,0.3f,0.7f),
46          // Tritanopie - blindness to short wavelengths
            float3x3(
48            0.95f,0.05f,0.0f,
              0.0f,0.433f,0.567f,
50            0.0f,0.475f,0.525f)
              };
52
          struct v2f {
54          float4 pos : SV_POSITION;
            float2 uv : TEXCOORD0;
56        };

58        // vertex shader
          v2f vert(appdata_img v)
60        {
            v2f o;
62          o.pos = UnityObjectToClipPos(v.vertex);
            o.uv = MultiplyUV(UNITY_MATRIX_TEXTURE0, v.texcoord);
64          return o;
          }
66
          // fragment shader
```

```
68          half4 frag(v2f i) : SV_Target
            {
70            // read the color from input texture
              half4 color = tex2D(_MainTex, i.uv);
72            // matrix multiplication with color-shifting matrix - index
    specified by 'type' variable
              float3 x = mul(color.rgb, color_matrices[type]);
74            // cast it to proper type before returning
              return half4(x,1.0f);
76          }

78          ENDCG
        }
80
    }
82 }
```

Código 4.3: Codigo de Shader de Daltonismo

Shader de Efecto de Onda

El shader “Hidden/Ripple Effect“ implementa un efecto de ondulación en un objeto renderizado, en nuestro caso lo realizamos en nuestra cámara. El shader utiliza varias propiedades para controlar su apariencia, como una textura base, una textura de gradiente, un color de reflexión y varios parámetros ajustables.

En la función “wave“, se calcula la amplitud de la onda en función de la posición del píxel, el origen de la onda que se encuentra en la posición del jugador y el tiempo. Luego, en la función “allwave“, se combinan las amplitudes de varias ondas para crear un efecto general de ondulación.

En el píxel shader “frag“, se calcula el desplazamiento de textura basado en las ondas y se aplica a la coordenada de textura original. Luego, se obtiene el color del píxel en la textura base y se mezcla con el color de reflexión según la intensidad de la ondulación.

El shader se define en un subshader que desactiva la escritura en el búfer de profundidad y no aplica niebla. Utiliza el modelo de fragment shader “3.0“ y utiliza las funciones “vert_img“ y “frag“ para el vértice y píxel shaders respectivamente.

```
Shader "Hidden/Ripple Effect"
2 {
      Properties
4     {
          _MainTex("Base", 2D) = "white" {}
6         _GradTex("Gradient", 2D) = "white" {}
          _Reflection("Reflection Color", Color) = (0, 0, 0, 0)
8         _Params1("Parameters 1", Vector) = (1, 1, 0.8, 0)
```

```
        _Params2("Parameters 2", Vector) = (1, 1, 1, 0)
        _Drop1("Drop 1", Vector) = (0.49, 0.5, 0, 0)
        _Drop2("Drop 2", Vector) = (0.50, 0.5, 0, 0)
        _Drop3("Drop 3", Vector) = (0.51, 0.5, 0, 0)
    }

    CGINCLUDE

    #include "UnityCG.cginc"

    sampler2D _MainTex;
    float2 _MainTex_TexelSize;

    sampler2D _GradTex;

    half4 _Reflection;
    float4 _Params1;    // [ aspect, 1, scale, 0 ]
    float4 _Params2;    // [ 1, 1/aspect, refraction, reflection ]

    float3 _Drop1;
    float3 _Drop2;
    float3 _Drop3;

    float wave(float2 position, float2 origin, float time)
    {
        float d = length(position - origin);
        float t = time - d * _Params1.z;
        return (tex2D(_GradTex, float2(t, 0)).a - 0.5f) * 2;
    }

    float allwave(float2 position)
    {
        return
            wave(position, _Drop1.xy, _Drop1.z) +
            wave(position, _Drop2.xy, _Drop2.z) +
            wave(position, _Drop3.xy, _Drop3.z);
    }

    half4 frag(v2f_img i) : SV_Target
    {
        const float2 dx = float2(0.01f, 0);
        const float2 dy = float2(0, 0.01f);

        float2 p = i.uv * _Params1.xy;

        float w = allwave(p);
        float2 dw = float2(allwave(p + dx) - w, allwave(p + dy) - w);

        float2 duv = dw * _Params2.xy * 0.2f * _Params2.z;
        half4 c = tex2D(_MainTex, i.uv + duv);
        float fr = pow(length(dw) * 3 * _Params2.w, 3);

        return lerp(c, _Reflection, fr);
    }
```

```
64      ENDCG

66      SubShader
        {
68          Pass
            {
70              ZTest Always Cull Off ZWrite Off
                Fog { Mode off }
72              CGPROGRAM
                #pragma fragmentoption ARB_precision_hint_fastest
74              #pragma target 3.0
                #pragma vertex vert_img
76              #pragma fragment frag
                ENDCG
78          }
        }
80  }
```

Código 4.4: Codigo de Shader de Efecto Onda

Librería Unity: Cinemachine

Cinemachine es una librería de Unity que proporciona herramientas y funcionalidades avanzadas para la creación de cámaras virtuales en juegos y aplicaciones interactivas. Permite crear y controlar fácilmente la cinematografía de la escena, logrando efectos visuales cinematográficos y una experiencia de juego inmersiva.

Una de las características principales que se muestra en la página oficial de Unity [15], es su capacidad para generar cámaras virtuales de forma dinámica y en tiempo real. Esto significa que puedes crear configuraciones de cámara complejas y automatizar su comportamiento mediante el uso de scripts y reglas predefinidas. Cinemachine ofrece diferentes tipos de cámaras virtuales, como cámaras de seguimiento, cámaras de enfoque automático y cámaras de seguimiento de objetivos, entre otras.

Cinemachine también es compatible con el Editor de Unity, lo que facilita la configuración y ajuste de las cámaras virtuales directamente desde la interfaz gráfica. Esto te permite iterar [3] rápidamente y realizar cambios en tiempo real para obtener el resultado deseado.

Esta librería ha sido utilizada en nuestro proyecto con el propósito de mejorar el seguimiento de la cámara del personaje. Su implementación nos ha permitido evitar movimientos bruscos en la vista que podrían causar molestias o mareos en los usuarios. Además, ha contribuido a sumergir a los usuarios en la experiencia del personaje que están controlando, proporcionando una sensación más inmersiva y fluida.

[3] Realizar cierta acción varias veces.

4.4.4. Interfaz del seleccionador del último nivel

Este componente es una parte integral de nuestro producto final, diseñado para el correcto funcionamiento de nuestro módulo. En este componente se especifican las variables mencionadas al final de la sección 2.3, las cuales permitirán a cualquier objeto del escenario calcular el carácter del usuario y determinar qué final se debe elegir.

A continuación, se muestra una imagen que representa la interfaz de este panel, que corresponde a la interfaz de este componente dentro de un objeto en Unity:

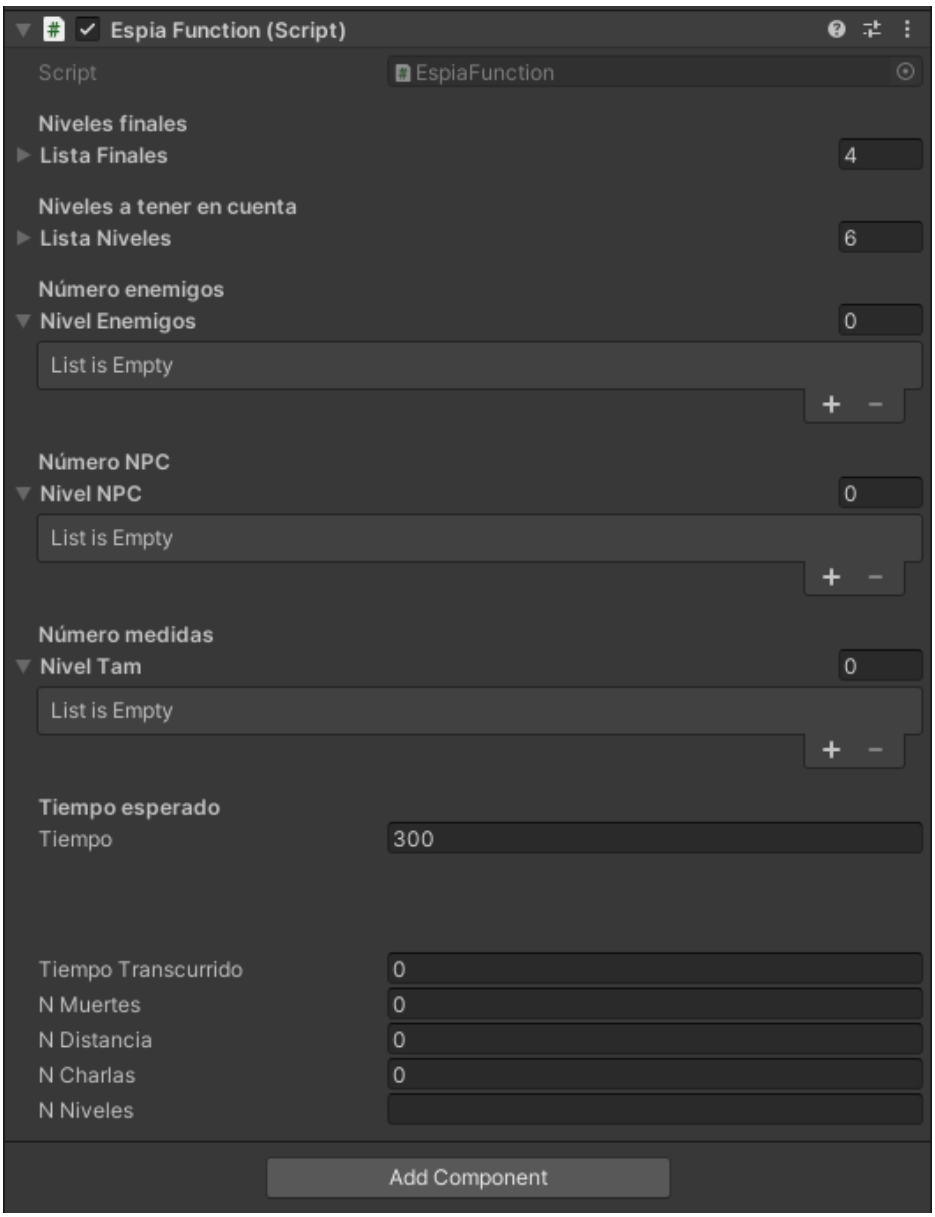

Figura 4.26: Interfaz del componente en Unity encargado de calcular y mostrar la personalidad exhibida por el usuario.

Dentro de la interfaz, la primera variable que se muestra es la lista que contiene los 4 posibles *Niveles Finales*, los cuales representan las diferentes actitudes mostradas por la interacción del jugador durante su tiempo de juego. Cada uno de estos finales tiene una variable asociada que se utilizará para determinar el nivel de la actitud predominante y ajustar así la dificultad correspondiente a cada uno de ellos.

Después, se muestra la *Lista Niveles*, que representa la estructura de nuestra mazmorra, donde el orden de los objetos es importante. A partir de los niveles detallados en esta lista, podemos llenar dinámicamente la información de otras listas, como *Nivel Enemigos*, que se genera a partir de los hijos del objeto padre "Enemigos"; *Nivel NPC*, que se genera a partir de los hijos del objeto padre "NPC"; y *Nivel Tam*, que se genera en función de los límites definidos en el componente "Render".

El parámetro que se muestra a continuación de las listas es un valor de tipo "float" llamado *Tiempo*, que representa el tiempo promedio que puede tomar a un jugador de nivel principiante llegar hasta el final del nivel, con un pequeño margen de error. Este margen permite la posibilidad de alcanzar el final Triunfador a un mayor rango de usuarios.

Los parámetros que se muestran al final de la interfaz sirven para comprobar el correcto funcionamiento del sistema y garantizar la ausencia de errores en la implementación del desarrollador.

El parámetro *Tiempo Transcurrido* actúa como un cronómetro que se activa mientras el jugador se encuentra dentro de la mazmorra, sin pausas. *N Muertes* indica el número de eliminaciones que el jugador ha realizado, mientras que *N Distancia* refleja la distancia recorrida por el jugador en la mazmorra. Por otro lado, *N Charlas* muestra la cantidad de conversaciones que el jugador ha mantenido a lo largo de la aventura.

Sin embargo, el parámetro más relevante es *N Niveles*, muestra una cadena de texto con los niveles que el jugador ha cruzado hasta el momento. Este parámetro resulta especialmente valioso, ya que nos permite equilibrar y analizar aspectos clave para comprobar el carácter y progreso del usuario en el juego.

4.4.5. Cálculos de los parámetros y de la selección del nivel final

En la interfaz hemos podido observar las variables presentes en el componente del código. En este apartado, exploraremos cómo se obtendrán los valores para estas variables y cómo se utilizarán para calcular el nivel final.

Para determinar automáticamente la cantidad de variables interactivas presentes en los niveles y almacenar sus valores en las listas correspondientes, hemos utilizado la siguiente función. Esta función nos permite obtener el número de cada variable clave a partir del nombre del objeto padre que las contiene.

```
GameObject GetChildWithName(GameObject obj, string name)
    {
        Transform trans = obj.transform;
        Transform childTrans = trans.Find(name);
        if (childTrans != null)
        {
            return childTrans.gameObject;
        } else
        {
            return null;
        }
    }
```

Código 4.5: Función de obtención de hijos

Una vez que tengamos acceso a toda la información recopilada sobre las interacciones del usuario en la mazmorra y los parámetros obtenidos mediante la monitorización del jugador, se activará un evento al completar el antepenúltimo nivel. Este evento desencadenará la función de cálculo del nivel final, la cual determinará qué final de la lista, especificada por el programador, se activará en la escena correspondiente. De esta manera, se adaptará el desenlace del juego según las acciones y elecciones del jugador durante su progreso en la mazmorra.

```
public void CalcularFinal()
    {
        float finalMuerte = 0;
        float finalLogro = tiempo/tiempoTranscurrido;
        float finalSocial= 0;
        float finalExplorador= nivelTam[0] + nivelTam[1] + nivelTam[2] +
    nivelTam[3] + nivelTam[4] + nivelTam[5];

        for(int i=0;i < nNiveles.Length;i++)
        {
            //Debug.Log("Estamos calculando el nivel " +
    int.Parse(""+nNiveles[i]));
            //Debug.Log("Estamos calculando el nivel " + nNiveles[i]);
            finalMuerte+=(nivelEnemigos[int.Parse(""+nNiveles[i])]);
            finalSocial+=(nivelNPC[int.Parse(""+nNiveles[i])]);
        }

        finalMuerte = nMuertes / finalMuerte;
        finalSocial = nCharlas / finalSocial;
        finalExplorador= nDistancia / finalExplorador;

        Debug.Log("finalLogro ="+ finalLogro);
        Debug.Log("finalMuerte ="+ finalMuerte);
        Debug.Log("finalSocial ="+ finalSocial);
        Debug.Log("finalExplorador =" + finalExplorador);

        if((finalMuerte > finalLogro)&&(finalMuerte >
    finalSocial)&&(finalMuerte > finalExplorador))
            listaFinales[1].SetActive(true);
```

```
        else if((finalLogro > finalSocial)&&(finalLogro > finalExplorador))
28          listaFinales[0].SetActive(true);
        else if ((finalExplorador > finalSocial))
30          listaFinales[3].SetActive(true);
        else
32          listaFinales[2].SetActive(true);

34  }
```

Código 4.6: Función de calculo de final

En esta función, se realiza una serie de cálculos y comparaciones para determinar el nivel final del juego que será activado. A continuación, se describen los pasos que se llevan a cabo:

- Se declaran y se inicializan variables que representan diferentes aspectos del juego, como el número de muertes, la interacción social y la exploración del jugador. También se ha calculado previamente la concordancia del jugador con el final de logro.

- Mediante un bucle que recorre los diferentes niveles por los que el jugador ha pasado, se calcula el total de enemigos y NPCs con los que el jugador hubiera podido interactuar en cada nivel. Estos valores se acumulan en las variables correspondientes.

- A continuación, se calcula la concordancia de cada uno de estos aspectos (muertes, interacción social y exploración) en relación con los demás finales posibles.

- Se realizan comparaciones entre las concordancias calculadas para determinar cuál de ellas es la mayor. Esto nos permite identificar el nivel que tiene una mayor compatibilidad con el carácter mostrado por el jugador.

- Finalmente, se activa el nivel final correspondiente al resultado de las comparaciones, utilizando la función *SetActive(true)* en la lista *Niveles Finales*.

A partir de estos pasos se llevará a cabo una evaluación detallada de diferentes aspectos del juego y del comportamiento del jugador, para seleccionar el nivel final más adecuado y activarlo en la escena del juego.

4.5. Desarrollo con Unity

El desarrollo de un videojuego en Unity implica mucho más que simplemente crear los assets [4] y diseñar los niveles. La programación del juego agrega funcionalidades que no están presentes de forma predeterminada en Unity, pero que son esenciales para lograr los objetivos del proyecto y permitir un análisis completo de la experiencia de juego. En esta sección del capítulo, se explorará todo el trabajo adicional realizado en el desarrollo del juego, aprovechando las capacidades y herramientas integradas en Unity para mejorar y enriquecer la experiencia de juego.

4.5.1. Interfaces de usuario

Menú Inicio

Es un canvas[5] que utiliza un modo de renderizado en espacio de cámara y se escala según el tamaño de la pantalla. En él, se encuentran los siguientes paneles principales:

- *Panel*, es el panel que presenta un fondo temático acorde al juego y una serie de botones con diferentes opciones. Estos botones incluyen las opciones de "Jugar", "Controles", "Logros" y "Salir".

- *Panel_ Trofeos*, es el panel donde se muestra de forma si están desbloqueados los trofeos y la descripción de como se podrían obtener.

- *Panel_ Aseg_ Salir*, es el panel utilizado para confirmar si el usuario desea salir del juego, presentando una pregunta y dos botones: uno de confirmación y otro de cancelación.

- *Panel_ Carga*, panel que se muestra al cargarse la siguiente escena, dentro se haya una animación de carga del personaje del "Herrero".

- *Panel_ Control*, el panel donde se muestra a partir de una imagen todos los controles de los que puede hacer uso el usuario.

[4] En el mundo del videojuego es todo elemento que es parte de un videojuego. Esos elementos pueden ser desde geometrías hasta sonidos o animaciones.

[5] Es un elemento de la nueva especificación HMTL5 que permite generar gráficos de forma dinámica a partir de formas 2D o mapa de bits a través de Javascript.

Todos estos elementos, excepto en el panel *Panel_ Aseg_ Salir*, tendrán un botón con la marca "X" que permitirá al usuario regresar al menú anterior. Además, todas estas elementos compartirán la misma posición en la interfaz.

El *Panel_ Aseg_ Salir* también tiene una diferencia importante, ya que depende de la información proporcionada por el script "GameManager" para mostrar los trofeos desbloqueados. Esto se logra mediante una comparación del número binario que representa los trofeos y permitir su desbloqueo en el panel.

Menú de Pausa

Es el mismo tipo de canvas mencionado en el punto 4.5.1, pero en este caso se desactiva el cronómetro del tiempo de juego del jugador y se muestran las propiedades de nuestro jugador. Permite reanudar el juego en el momento que el usuario decida, cargar la escena del inicio, cargar el escenario de selección de nivel o seleccionar una de las opciones que activará los siguientes paneles:

- *Panel_ Opciones*, panel que muestra los controles para modificar los diferentes parámetros audiovisuales del videojuego.

- *Panel_ Control*, es el panel donde se muestra a partir de una imagen todos los controles de los que puede hacer uso el usuario.

Todos estos elementos se mostrarán en la misma posición en la interfaz y contarán con un botón con la marca "X" que permitirá al usuario regresar al menú principal.

Además, para que los elementos UI del panel "Panel_Opciones" funcionen correctamente, dependerán de las siguientes funciones del script "GameManager":

- Al cambiar el daltonismo, el elemento "Dropdown_Daltonismo" activará la función "CambiarDaltonismo()", que modificará el tipo de daltonismo mostrado en la cámara utilizando el shader mencionado en la sección 4.4.3, y guardará el cambio en el sistema.

- Cuando se cambie el tamaño de la pantalla, el elemento "ButtonScreen" activará la función "SetScreenSize()", que cambiará el aspecto del botón, ajustará el tamaño de la pantalla al valor correspondiente y guardará el valor en el sistema.

- Para cambiar el volumen general, el elemento "Barra_Volumen" activará la función "SetBackgroundVolume()", que ajustará el volumen en el Audio Mixer y guardará el cambio en el sistema.

- Para cambiar el volumen de los efectos, el elemento “Barra_Volumen_Efectos“ activará la función “SetSFXVolume()“, que ajustará el volumen en el Audio Mixer y guardará el cambio en el sistema.

- Para cambiar el brillo general, el elemento “Barra_Brillo“ activará la función “SetBrightness()“, que modificará el brillo ajustando el valor de exposición utilizado en el post-procesado de brillo en “Brightness“.

Menú de Muerte

Es el mismo tipo de canvas mencionado en el punto 4.5.1, pero esta vez permite al jugador elegir su forma de morir mediante un grupo de botones. El botón “Button_Continuar“ permite al jugador reaparecer en la posición guardada o al inicio del nivel, mientras que el botón “Button_Select“ cambia a la escena del nivel base. Por otro lado, el botón “Button_MainMenu“ lleva al jugador de regreso a la pantalla de inicio.

4.5.2. Creación de niveles

Nivel Base

El objetivo de este nivel es familiarizarse con los controles del juego y probar algunas de las diversas formas de interactuar con el mundo.

En este nivel, encontrarás tres tipos de personajes: “Palomas“, “Guardias“ y “Habitantes“. Estos personajes te brindarán diferentes oportunidades de interacción y te permitirán explorar las mecánicas del juego de diferentes maneras. El jugador deberá aprovechar esta oportunidad para experimentar y aprender cómo interactuar con cada uno de ellos.

Nivel 0

El objetivo de este nivel es familiarizarse con el combate contra enemigos básicos, aprender a escalar paredes y comprender el funcionamiento de las plataformas rojas.

En este nivel, te enfrentarás a dos tipos de enemigos: “Arañas“ y “Murciélagos“. Estos enemigos te desafiarán y pondrán a prueba tus habilidades de combate. Aprende a esquivar sus ataques y utiliza tus habilidades de ataque para derrotarlos.

Además, encontrarás plataformas rojas que te proporcionarán un desafío adicional. Aprende a utilizarlas correctamente para avanzar por el nivel y superar obstáculos.

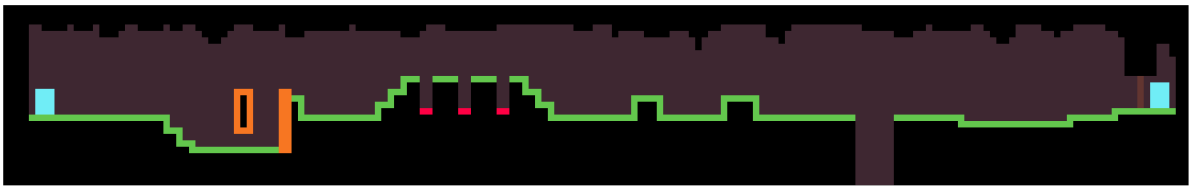

Figura 4.27: Diseño del nivel 0 de la mazmorra.

Nivel 1

El objetivo de este nivel es que el jugador pueda familiarizarse con las mecánicas del juego, especialmente en relación a los pasajes secretos, la escalada de paredes y el funcionamiento de las plataformas móviles.

Durante este nivel, el jugador encontrará diversos desafíos que le permitirán poner en práctica estas habilidades. Enfrentará a enemigos como “Arañas“, “Murciélagos“ y “Arqueros“, quienes pondrán a prueba sus habilidades de combate y evasión.

Las “Arañas“ estarán dispersas por el nivel, atacando al jugador estarán realizando sus rutas y solo atacaran si se les cruza algo por el camino.

Los “Murciélagos“, por otro lado, serán enemigos voladores que atacarán desde el aire. El jugador deberá aprender a anticipar sus movimientos y utilizar ataques adecuados para eliminarlos de manera eficiente.

Además de los enemigos mencionados, también habrá “Arqueros“ que atacarán al jugador a distancia. Será fundamental utilizar la cobertura y esquivar sus ataques para acercarse lo suficiente y derrotarlos.

A medida que el jugador explore el nivel, encontrará pasajes secretos que le permitirán acceder a áreas ocultas con recompensas adicionales. Será necesario prestar atención a los detalles del entorno y utilizar la habilidad de escalada para descubrir estos caminos ocultos.

Además, habrá plataformas móviles que se moverán por el mapa, que servirán para que se acostumbre el jugador a su uso.

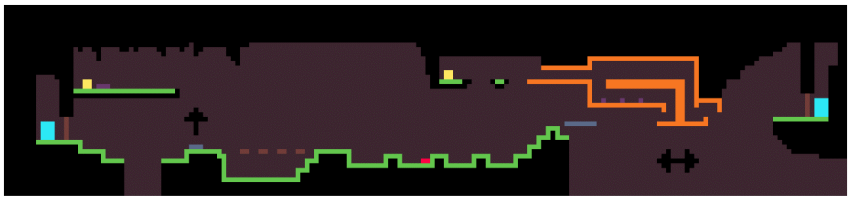

Figura 4.28: Diseño del nivel 1 de la mazmorra.

Nivel 2

El objetivo de este nivel es familiarizarse con el patrón de movimiento del primer jefe y mejorar tus habilidades de escalada.

En este nivel, el usuario se enfrentará al temible "Caballero Esqueleto", un enemigo formidable que requerirá de destreza y estrategia para ser derrotado. El jugador deberá de estudiar sus movimientos y patrones de ataque para encontrar sus puntos débiles y aprovecharlos en su beneficio.

Además de enfrentarse al "Caballero Esqueleto", también tendrá la oportunidad de fortalecer su habilidad de escalada. Se encontrara una sección del nivel en las que deberá escalar paredes para avanzar.

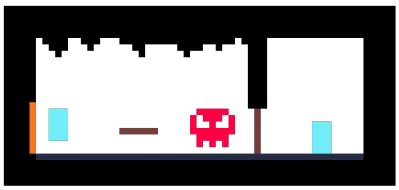

Figura 4.29: Diseño del primer nivel con jefe de la mazmorra.

Nivel 3

El objetivo de este nivel es que el jugador se familiarice con los nuevos enemigos que se presentarán. Durante esta etapa del juego, se introducirán enemigos como "Slimes", "Espíritus" y "Arqueros".

Los "Slimes" son criaturas gelatinosas que se desplaza siguiendo una ruta lentamente por el escenario. El jugador deberá aprender a esquivar sus ataques y encontrar estrategias efectivas para eliminarlos.

Los "Espíritus", por otro lado, son enemigos etéreos que pueden atravesar objetos sóli-

dos y realizar ataques a corta distancia. El jugador deberá ser ágil y utilizar movimientos evasivos para evitar sus ataques y encontrar oportunidades para contraatacar.

Además de los “Slimes“ y “Espíritus“, también se encontrarán los “Arqueros“, que serán del mismo tipo que en los niveles anteriores pero con un posicionamiento más complicado para el jugador.

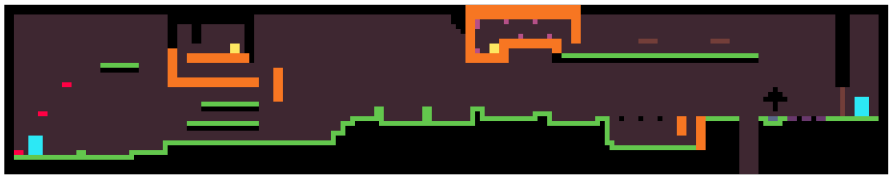

Figura 4.30: Diseño del nivel 3 de la mazmorra.

Nivel 4

El objetivo de este nivel es permitir al jugador mejorar sus habilidades de equilibrio, razonamiento y reflejos. A lo largo de esta etapa del juego, se encontrarán diversos desafíos diseñados para poner a prueba estas habilidades.

En este nivel, el jugador se enfrentará a una variedad de enemigos, incluyendo “Ogros“, “Slimes“, “Espíritus“ y “Arqueros“. Cada uno de estos enemigos presenta un conjunto único de habilidades y comportamientos que requerirán diferentes enfoques estratégicos para ser derrotados.

Los “Ogros“ son criaturas grandes y poderosas que atacarán al jugador con golpes fuertes y contundentes. El jugador deberá ser ágil y esquivar sus ataques mientras busca oportunidades para contraatacar.

Además de los Ogros, también se encontrarán “Slimes“, “Espíritus“ y “Arqueros“, de los cuales ya debería de tener experiencia el jugador para derrotarlos.

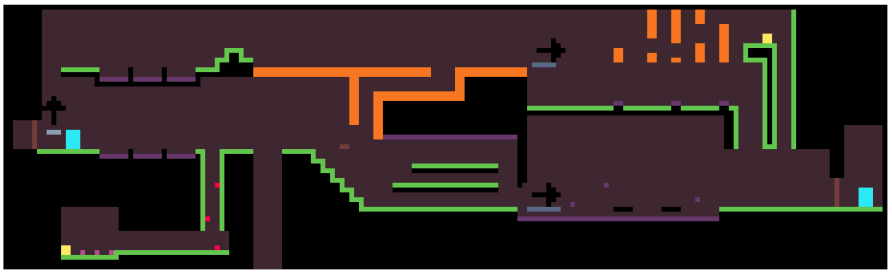

Figura 4.31: Diseño del nivel 4 de la mazmorra.

Nivel 5

El objetivo de este nivel, donde se encuentra el segundo jefe, es permitir al jugador acostumbrarse al movimiento y patrones de ataque del jefe, al mismo tiempo que se refuerza la habilidad de evitar golpes a distancia.

En este nivel, el jugador se enfrentará al temible “Ogro Slime“, un enemigo poderoso y resbaladizo. El “Ogro Slime“ tiene la capacidad de lanzar proyectiles a distancia para dañar al personaje. El objetivo del jugador será aprender a esquivar y evitar estos ataques a distancia, manteniendo una buena posición y moviéndose de manera estratégica.

Durante la fase de mitad de vida del jefe, el escenario puede generar una lluvia aleatoria de moco, representando un nuevo desafío para el jugador. Estas gotas de moco pueden caer de manera impredecible y dañar al personaje si son alcanzadas. El jugador deberá estar atento y ágil para moverse y evitar estas gotas de moco mientras continúa su enfrentamiento con el jefe.

Es importante que el jugador aprenda los patrones de movimiento y ataques del “Ogro Slime“, así como los momentos oportunos para contraatacar. La habilidad de esquivar y evitar golpes a distancia será fundamental para superar este desafío.

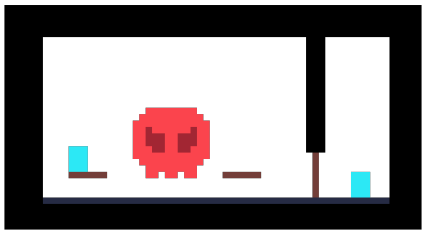

Figura 4.32: Diseño del segundo nivel con jefe de la mazmorra.

Nivel 6

El objetivo de este nivel, que se crea de forma dinámica basándose en la actitud del jugador, es llegar al final de la mazmorra y obtener la espada sagrada. Este nivel está diseñado para representar y reflejar el progreso y las decisiones del jugador hasta el momento en el juego.

Si el jugador alcanza el "Final de Logro", se enfrentará a un desafío adicional en forma de un laberinto aleatorio que obstaculizará su avance y brindará una experiencia más desafiante en nuestro videojuego. Por otro lado, si alcanza el "Final de Explorador", se encontrará con un laberinto constante donde los cofres estarán dispersos de forma aleatoria por todo el escenario, fomentando la exploración y la búsqueda de recompensas ocultas. Además, si logra llegar al "Final de Asesino", se enfrentará a un gran número de enemigos

y tendrá acceso a varias pociones de inmortalidad para enfrentarlos. Por último, si alcanza el "Final Social", se encontrará con una animada multitud con la que podrá interactuar y entablar diálogos enriquecedores.

Cada uno de estos finales representa una experiencia única y desafiante para el jugador, adaptada a su actitud y elecciones a lo largo del juego.

Figura 4.33: Diseño del nivel final de la mazmorra.

En la gran mayoría de los niveles, encontrarás comerciantes dispuestos a ayudarte y proporcionarte pistas útiles a cambio de un poco de tu tiempo como jugador. Estos comerciantes serán una valiosa fuente de información y podrán brindarte consejos valiosos para avanzar en el juego.

En cuanto al desplazamiento entre diferentes zonas, generalmente se requiere activar un mecanismo de apertura de puerta para pasar de una zona a otra. Sin embargo, en casos donde la puerta no exista o ya haya sido activada previamente, simplemente entrar en contacto con la apertura será suficiente para ser trasladado de un nivel a otro.

Además, se ha implementado un sistema para evitar confusiones entre hoyos y caídas. Se utilizan monedas estratégicamente ubicadas para indicar el camino adecuado o correcto durante los niveles. Estas monedas actúan como señales visuales para que los jugadores puedan tomar decisiones informadas y evitar trampas o rutas equivocadas.

4.5.3. Interacción con elementos de la escena

En cada juego, es crucial proporcionar al jugador diversas formas de interactuar con el entorno. En nuestro caso, optamos por implementar un objeto adicional con el componente “Collider2D“ activado para cada elemento interactivo. Estos objetos funcionan como desencadenadores de eventos. Cada recurso interactivo contiene un script llamado “Interactuable“, que indica el tipo de interacción y define cómo se debe activar el evento correspondiente.

Al utilizar los “Colliders2D“, podemos detectar cuando el jugador se acerca o entra

en contacto con un elemento interactivo específico. El script “Interactuable“ asociado al objeto se encarga de gestionar esta interacción y determinar qué evento debe activarse en función del tipo de interactuable. Esto nos brinda flexibilidad para personalizar la forma en que se desencadenan los eventos y permite al programador definir las interacciones de acuerdo con las necesidades del juego.

Entre los recursos interactivos disponibles en el juego se encuentran los “NPCs“, con los que el jugador puede entablar diálogos, las “Palancas“ que permiten avanzar en los niveles sin enfrentar jefes, los puntos de control (“Checkpoints“) que guardan temporalmente la información del jugador, el “SeleccionadorNiveles“ que permite al jugador teletransportarse a cualquier nivel de la mazmorra previamente alcanzado, y el “Finalizador“, que transporta al jugador a la escena que demuestra los resultados del final relacionados con la actitud mostrada por el jugador.

4.5.4. Enemigos

En esta sección, analizaremos a los desafiantes enemigos que se encuentran en cada rincón del juego. Estos antagonistas representan una amenaza para el jugador y pondrán a prueba sus habilidades y estrategias. Enfrentaremos una amplia variedad de enemigos con diferentes grados de dificultad, lo que contribuirá a hacer nuestra aventura más inmersiva y emocionante para los usuarios.

Los enemigos terrestres y aéreos en nuestro videojuego presentan diferentes características y formas de ataque. A continuación, se describe el funcionamiento de cada uno de ellos:

- Arañas: Estos enemigos tienen 3 vidas y siguen una ruta establecida que se establece a partir del script “Waypoints()“. Pueden ser eliminadas saltando encima de ellas o atacándolas con ataques de espada.

- Slimes: Los slimes también tienen 2 vidas y siguen una ruta establecida que se establece a partir del script “Waypoints()“. Al igual que las arañas, pueden ser eliminados saltando encima de ellos o atacándolos con ataques de espada.

- Murciélagos: Estos enemigos aéreos cuentan con 3 vidas y tienen la capacidad de detectar al jugador en un área determinada especificada en el script “BatControl“. Una vez que te han detectado, te perseguirán para atacarte de cerca. Para eliminarlos, puedes saltar encima de ellos o utilizar ataques con la espada.

- Espíritus: Al igual que los murciélagos, los espíritus tienen 3 vidas y también son capaces de detectar al jugador en un área específica según el script “SpiritControl“.

Sin embargo, se caracterizan por su velocidad de persecución más rápida que la de los murciélagos. Para derrotarlos, puedes saltar encima de ellos o utilizar espadazos.

- Arqueros: Estos enemigos, según el código del script “SkeletonControl“, pueden desplazarse al detectar al jugador y, a cierta distancia, comenzarán a lanzar flechas potentes. Tienen 3 vidas y representan una amenaza desde la distancia. Para vencer a los arqueros, deberás acercarte lo suficiente para atacarlos con ataques de espada.

- Ogros: Los ogros son enemigos terrestres resistentes con 6 vidas. Según el código del script “GolemControl“, solo pueden ser derrotados utilizando ataques con espada y no pueden ser eliminados saltando sobre ellos. Al igual que los demás enemigos, también tienen la capacidad de detectar al jugador en un área y perseguirlo para atacarlo de cerca.

- Caballero Esqueleto: Es el primer jefe en el juego y presenta una serie de habilidades y comportamientos durante el combate, los cuales están basados en el script "Jefe2Controler". Cuando el jugador aparece en su sala, se activa un "slider"que representa la vida del jefe.

  A corta distancia, el Caballero Esqueleto realizará de forma aleatoria dos ataques: una estocada a larga distancia o un balanceo de hacha hacia el lado donde detecta al jugador. A larga distancia, ejecutará una embestida que sólo se puede esquivar escalando o utilizando las plataformas del nivel que dejara al monstruo paralizado al chocar con una pared durante 2 segundos.

  Al derrotar al Caballero Esqueleto, recibirás una llave como recompensa, la cual te permitirá abrir la puerta que lleva al siguiente nivel. Enfrentarse a este jefe requiere dominar la habilidad de escalada, para poder esquivar sus ataques y poder atacar sus puntos débiles.

- Ogro Slime: Es el segundo jefe en el juego que presenta una serie de habilidades y comportamientos durante el combate basados en el script “Jefe3Controler“. Cuando el jugador accede a su sala, se activa un “slider“ que representa la vida del jefe.

  El Ogro Slime tiene tres ataques básicos: un golpe de puño a corta distancia, la aparición de un pincho de slime a mediana distancia y el disparo de bolas de slime a larga distancia. Después de disparar con el slime debe descansar para volver a realizar un nuevo ataque al jugador.

  Cuando se encuentra a mitad de vida, podrá utilizar un ataque especial llamado “Lluvia de slime“, que se activa a larga distancia mediante un puñetazo al suelo.

  Al derrotar al Ogro Slime, obtendrás una llave como recompensa, que permitirá abrir la puerta que lleva al siguiente nivel. Este jefe requiere de habilidad para esquivar, para poder evitar daño y poder acercarse a sus puntos débiles.

4.5.5. Mecánicas desarolladas más relevantes

En este apartado, exploraremos dos mecánicas fundamentales de nuestro videojuego en 2D:

- El desplazamiento entre niveles es una mecánica fundamental que brinda a los jugadores la oportunidad de sumergirse en distintos entornos y embarcarse en emocionantes aventuras a lo largo de la historia del juego. A través de diversas herramientas como palancas, muros levadizos, puertas enormes o incluso puntos de teletransporte, los jugadores podrán trasladarse de un nivel a otro, desbloqueando áreas previamente inexploradas. Esta dinámica no solo les permite enfrentarse a nuevos desafíos, sino también volver a visitar zonas anteriores en busca de pasadizos secretos y descubrir tesoros ocultos.

  Este mecanismo de desplazamiento entre niveles se basa en una serie de scripts clave que brindan una experiencia fluida y emocionante a los jugadores:

  - El script "PlayerController" desempeña un papel fundamental al permitir el cambio entre niveles dentro de la mazmorra. Al interactuar con una palanca, se activa la función "MovimientoFinalMapa(Vec2 direcciónDeSalida)", la cual realiza una transición visual en negro e internamente desplazará al jugador hacia el punto de inicio del nivel correspondiente. Además, este script ajusta los límites de la cámara para adaptarse al nuevo entorno, ofreciendo una experiencia visualmente coherente y sin interrupciones.

  - El script "Interactuable" se encarga de gestionar los puntos de teletransporte que cambian a la escena correspondiente al siguiente nivel. Al ejecutarse la función "CargarEscena(string escena)", se carga la nueva escena del nivel, y utilizando la función "SetIndiceNivelInicio(int inicio)" del script "GameManager", se indica el punto de inicio específico del nivel. Esto asegura que los jugadores comiencen exactamente donde corresponde, evitando confusiones y manteniendo la progresión adecuada.

- El método de guardado de partida permite a los jugadores guardar su progreso y retomar la aventura en el punto exacto donde lo dejaron. Ya sea mediante puntos de guardado automáticos, estaciones de guardado o un sistema manual de guardado, los jugadores podrán disfrutar de una experiencia de juego fluida y sin preocupaciones.

  Este mecanismo de guardado de progreso se basa en varios scripts clave:

  - El script "EspiaFunction" tiene la responsabilidad de realizar un guardado separado de las interacciones que determinarán la actitud final del jugador. Utiliza la función "GuardarDatosEspia()" para almacenar estos datos de manera independiente. Por otro lado, el script "GameManager" juega un papel fundamental en el sistema de guardado de la partida. A través de la función "GuardarPartidaPasada()", se guarda automáticamente toda la información relevante del jugador, incluyendo datos e inventario de recursos. Estas funciones se activan de forma automática cuando el jugador avanza de un nivel a otro.

-El script “GameManager“ también permite hacer copias temporales de la posición y las vidas del jugador utilizando puntos de guardado en el mapa. La función “GuardarPartida()“ guarda estos datos, lo que permite al jugador retomar la partida desde el último punto de guardado en el nivel. Sin embargo, es importante tener en cuenta que este progreso puede borrarse al salir del nivel en el que se encuentra el jugador.

Capítulo 5

Pruebas y resultados

5.1. Introducción

En este capítulo, se detallan los métodos utilizados para llevar a cabo las pruebas, la recopilación de datos y la interpretación de los resultados. Además, se analizan los aspectos técnicos y las limitaciones del módulo, así como las implicaciones y posibles mejoras para futuras iteraciones. El objetivo de este capítulo es validar la funcionalidad y el rendimiento del módulo de creación del nivel final realizado mediante el uso de las tecnologías mencionadas en el punto 2.4.

Figura 5.1: Capturas de los distintos finales del juego.

5.2. Prueba de rendimiento

La primera prueba se ha centrado en evaluar el desempeño y la eficiencia de nuestro modulo sobre un videojuego realizado sobre Unity. Se ha analizado cómo Unity se comporta en términos de cálculos, físicas y otros componentes al activar el módulo en comparación con cuando está desactivado dicho módulo. Se ha puesto especial atención en los apartados de sonido y gráficos para determinar cualquier impacto significativo en el rendimiento. El objetivo principal ha sido comparar el rendimiento de Unity con y sin el módulo adicional, y así identificar cualquier diferencia o mejora que pueda ofrecer en estas áreas del desarrollo del videojuego.

Figura 5.2: Gráfica del rendimiento gráfico.

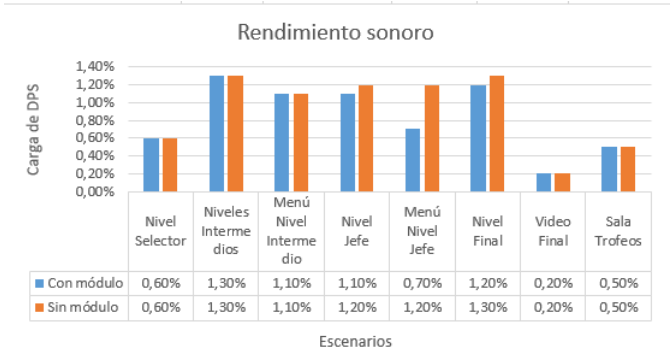

Figura 5.3: Gráfica del rendimiento del sonido.

Aunque en nuestras pruebas no se observe una diferencia significativa, sobre los escenarios de los niveles normales y finales se demuestra que el uso del módulo permite mantener o incluso mejorar la calidad gráfica y sonora de nuestro videojuego. Se observa una reducción en el tiempo de uso de la CPU, una mayor velocidad de cuadros por segundo (FPS) y una carga menor en el procesamiento de efectos especiales (DPS), debido a la diferencia entre tener todos los niveles cargados en el videojuego desde el inicio y cargar de forma dinámica el tipo de nivel final que deberá de estar presente en el juego

5.3. Prueba de funcionalidad con usuarios

La prueba de funcionalidad del módulo de selección de niveles dinámico se lleva a cabo mediante la participación de usuarios. El objetivo principal de esta prueba es verificar que el módulo genera niveles de forma adecuada y acorde a la actitud de juego de los usuarios. A través de esta prueba, se busca evaluar la capacidad del módulo para adaptarse y responder a las acciones y preferencias de los usuarios, brindando una experiencia de juego personalizada y satisfactoria.

Para la realización de estas pruebas, se contó con la colaboración de 6 usuarios con diferentes perfiles y niveles de experiencia en videojuegos:

- Persona 1: Estudiante universitario con experiencia en juegos de ordenador.
- Persona 2: Estudiante universitario sin experiencia en juegos de ordenador.
- Persona 3: Estudiante de instituto con experiencia en juegos de ordenador.
- Persona 4: Estudiante de instituto sin experiencia en juegos de ordenador.
- Persona 5: Adulto con experiencia en juegos de ordenador.
- Persona 6: Adulto sin experiencia en juegos de ordenador.

Estos usuarios fueron seleccionados con el objetivo de obtener una variedad de perspectivas y opiniones sobre el videojuego durante las pruebas.

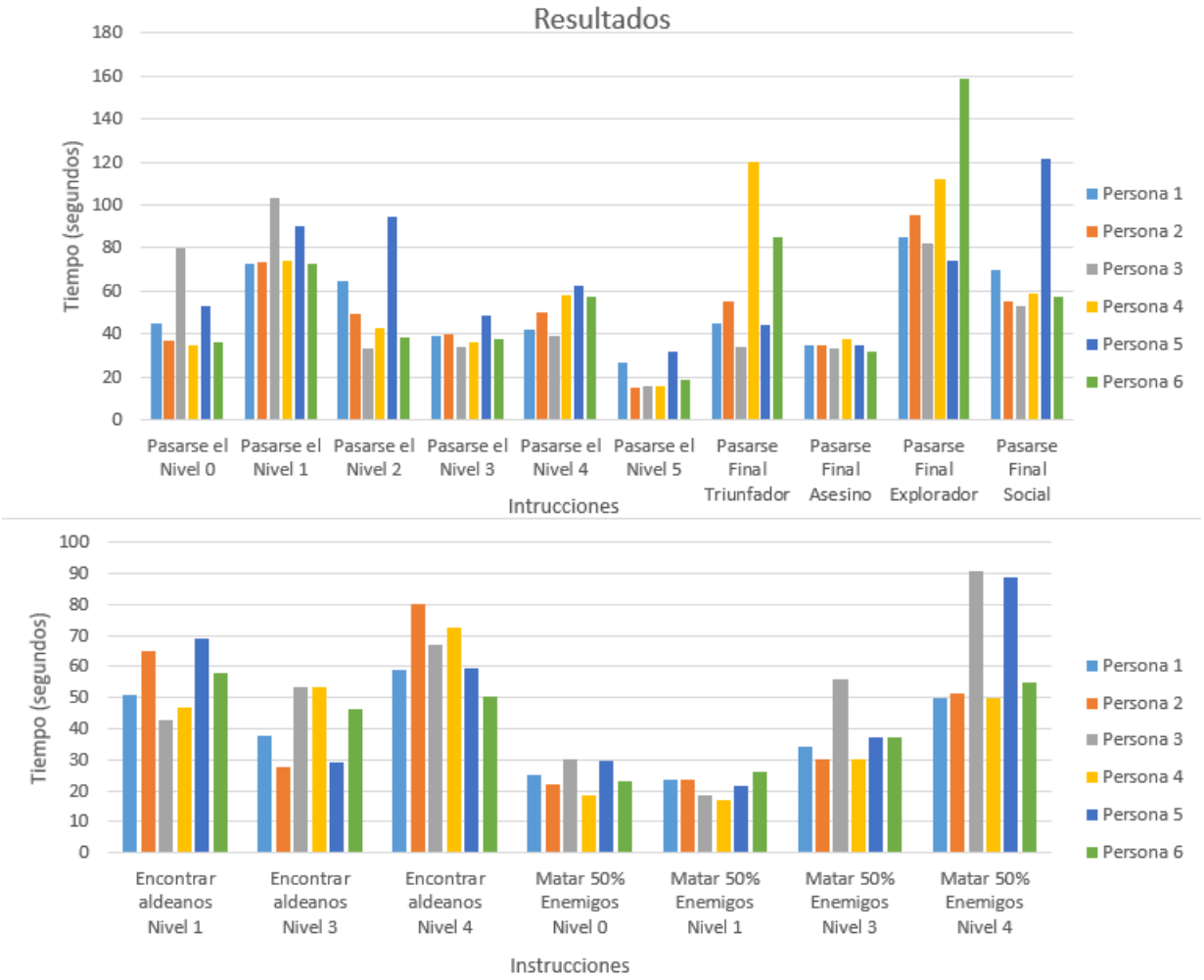

Figura 5.4: Gráfica de resultado de las pruebas de funcionalidad.

En el primer gráfico se muestra la velocidad con la que los jugadores intentaron completar los niveles, datos relevantes para tener en cuenta en la selección del final Triunfador. Se observa que, aunque en algunos niveles puede haber una diferencia significativa en los tiempos entre los jugadores con experiencia y los principiantes, esto se debe principalmente a que algunos jugadores se quedaron atascados en ciertos puntos del juego. Sin embargo, en general, la mayoría de los jugadores mostraron rutas y tiempos similares.

En el segundo gráfico se les pidió a los jugadores que encontraran y atacaran a cierto número de objetivos para desbloquear los diferentes niveles que no estaban disponibles por defecto, como el final Social y el final Asesino. En este gráfico se puede apreciar la diferencia en las actitudes de los diferentes usuarios, ya sea en el tiempo que les tomó eliminar a sus enemigos o en el tiempo que dedicaron a dialogar con los personajes no jugables (NPCs). Se observa una variedad en los tiempos y enfoques de juego de los jugadores.

En ambas gráficas los usuarios tuvieron una cantidad de 20 partidas sobres las cuales se sacaron mejores resultados sobre estas y para que se acostumbraran a la forma de juego. Aun así se puede notar en los gráficos que los adolescentes y jóvenes adultos tienden a ser más agresivos e inquietos en su estilo de juego, mientras que los adultos muestran una actitud más tranquila y prefieren interactuar con todos los elementos proporcionados por el videojuego.

5.4. Test SUS

La prueba final consiste en aplicar un Test de SUS para medir la eficacia del módulo y obtener datos cuantitativos y cualitativos sobre la experiencia proporcionada a los usuarios con el módulo, utilizando a los mismos usuarios que participaron en la prueba anterior. A los participantes se les realizarán 10 preguntas, la mitad de ellas serán afirmaciones positivas sobre el videojuego y la otra mitad serán negativas, las cuales deberán ser puntuadas en una escala del 1 al 5.

Las preguntas formuladas a los usuarios fueron las siguientes:

- Resulto fácil el videojuego.
- Aprendí rápidamente cómo jugar el videojuego.
- Las funciones y controles del videojuego son intuitivos.
- Disfruté explorando los diferentes niveles y escenarios del videojuego.
- El módulo de creación de niveles dinámicos mejoró mi experiencia de juego.
- Tuve dificultades para entender cómo jugar el videojuego.
- Los controles y funciones del videojuego resultaron confusos.
- La navegación por los niveles y escenarios del videojuego fue incómoda.
- El módulo de creación de niveles dinámicos no se adaptó bien a mi actitud de juego.
- Experimenté problemas técnicos o dificultades mientras jugaba el videojuego.

	Pregunta 1	Pregunta 2	Pregunta 3	Pregunta 4	Pregunta 5	Pregunta 6	Pregunta 7	Pregunta 8	Pregunta 9	Pregunta 10	Resultado Preguntas Positivas	Resultado Preguntas Negativas	Resultado Final
Persona 1	3	3	4	5	4	2	3	2	1	3	14	14	**70**
Persona 2	4	4	5	4	4	1	3	1	2	2	16	16	**80**
Persona 3	5	4	5	5	3	2	2	1	2	3	17	15	**80**
Persona 4	5	5	5	5	4	4	4	1	2	1	19	13	**80**
Persona 5	3	5	2	5	5	2	5	1	2	1	15	14	**72,5**
Persona 6	4	5	4	5	4	1	2	1	1	1	17	19	**90**
												MEDIA	**78,75**

Cuadro 5.1: Tabla de resultados del Test SUS.

A partir de los cálculos realizados en la tabla, podemos concluir que obtener una puntuación mayor a 60 indica que el videojuego en el que se encuentra nuestro módulo no requiere muchas correcciones y que nuestro proyecto ha proporcionado una gran satisfacción tanto a los principiantes como a los expertos en el género seleccionado para el videojuego.

Capítulo 6

Conclusiones

6.1. Conclusiones

La motivación principal de este proyecto ha sido lograr el desarrollo de un módulo dentro de un escenario en 2D creado en Unity, que nos permita monitorizar las interacciones realizadas por el usuario a lo largo de la mazmorra y determinar el final del juego de acuerdo a la actitud mostrada por el jugador. Nuestro motivación ha sido proporcionar una herramienta que permita a todos los tipos de jugadores disfrutar del juego y experimentar diferentes finales que dependen de ciertas acciones en los videojuegos.

En cuanto a los subobjetivos establecidos en la sección 1.2, hemos logrado implementar la gran mayoría de ellos de manera completa en nuestro proyecto, abarcando tanto las partes visibles para el usuario como la funcionalidad y la lógica interna del programa.

Durante el análisis realizado durante la creación del proyecto, hemos obtenido información relevante sobre los costes y tiempos asociados a proyectos en este sector, así como las limitaciones y ventajas que podrían surgir. También hemos realizado un estudio para determinar el precio de venta del producto, con el objetivo de obtener beneficios y atraer posibles inversores.

Respecto al diseño e implementación, hemos aplicado los conocimientos adquiridos a lo largo de nuestra carrera en gestión de proyectos, programación, animación, simulación y creación de videojuegos para desarrollar la prueba de concepto y el propio módulo. Este proyecto nos ha brindado la oportunidad de enfrentarnos a diversos desafíos y encontrar soluciones a problemas comunes en el desarrollo de videojuegos.

Durante el proceso, hemos comprendido la importancia de realizar copias de seguridad periódicas para evitar pérdidas de datos, así como guardar el proyecto en un disco externo como medida adicional de seguridad. Además, hemos aprendido la importancia de orga-

nizar adecuadamente los recursos, dado que los proyectos completos requieren una gran cantidad de elementos.

También hemos experimentado el valioso apoyo proporcionado por la comunidad de desarrolladores en este sector. En momentos de dificultad o al enfrentarnos a problemas específicos, hemos encontrado respuestas, consejos y soluciones a través de la interacción con otros desarrolladores y participando en comunidades en línea.

En base a los resultados obtenidos, hemos observado un gran interés generado por el juego entre diversos grupos de usuarios. Han mostrado aprecio por la selección del final basada en la actitud mostrada, así como por la adaptabilidad del juego, incluso para aquellos con poca experiencia en los videojuegos de ordenador, pero que disfrutan jugando en otras plataformas.

Esto nos indica que hemos logrado nuestro objetivo de brindar una experiencia atractiva y satisfactoria para una amplia gama de jugadores. La capacidad de adaptar el juego a diferentes actitudes ha sido bien recibida y ha generado un mayor nivel de disfrute y compromiso por parte de los usuarios.

En resumen, este proyecto ha cumplido con los objetivos planteados, brindando una herramienta en el ámbito de los videojuegos que permite una experiencia personalizada y atractiva para diferentes tipos de jugadores. Además, nos ha proporcionado un amplio conocimiento y experiencia en el desarrollo de proyectos de esta naturaleza, así como la oportunidad de aplicar y expandir nuestras habilidades en campos relacionados con nuestra carrera como son la informática y los videojuegos.

6.2. Trabajo Futuro

En cuanto al trabajo futuro, este proyecto podría ampliarse para incluir no solo los 4 tipos generales de actitudes de los jugadores, sino también especializarse en las 2 variantes derivadas de cada una de estas ramas, con el fin de ofrecer una experiencia más personalizada y especializada al usuario.

Para lograr esto, sería necesario diferenciar de manera más clara las interacciones durante la monitorización del jugador. Se podrían introducir nuevas variables que influyan en la generación de los niveles finales, reestructurar las funciones de guardado y ajustar la función de cálculo de selección del nivel final para abarcar la mayor cantidad de niveles posibles.

Otra mejora importante sería la implementación de una generación automática completa del último escenario en función del tipo de final al que llegó el usuario y teniendo en

cuenta el nivel de similitud que muestra con dicho final. Esto implicaría desarrollar una función que calcule el nivel de dificultad del final en base a la intensidad de la actitud del jugador.

Con estas mejoras, el módulo permitiría una generación de niveles más enfocada en los diferentes tipos de jugadores presentes en la industria de los videojuegos. Esto garantizaría una mayor satisfacción por parte de los jugadores al brindarles desafíos adaptados a su estilo de juego y preferencias. Además, esta mejora aumentaría el potencial de beneficios en la creación de este tipo de proyectos al captar la atención y fidelidad de una base de jugadores más amplia.

En resumen, la implementación de la generación automática del nivel en función de los finales y la personalidad del jugador constituye un importante camino a seguir para mejorar aún más la experiencia de juego y maximizar el impacto comercial de nuestro proyecto.

Bibliografía

[1] Guillermo Castilla. La taxonomía de bartle. https://creatividadenblanco.com/la-taxonomia-de-bartle/.

[2] Ministerio de Trabajo y Economía Social. Resolución de 27 de febrero de 2023, de la dirección general de trabajo, por la que se registra y publica el xx convenio colectivo nacional de empresas de ingeniería; oficinas de estudios técnicos; inspección, supervisión y control técnico y de calidad. https://www.boe.es/diario_boe/txt.php?id=BOE-A-2023-6346.

[3] Unreal Engine. Unreal engine 5. https://www.unrealengine.com/es-ES/unreal-engine-5.

[4] Andrea Flores. ¿cómo usar photopea? conoce la alternativa gratuita que podría superar a photoshop. https://www.crehana.com/blog/estilo-vida/aprende-como-usar-photopea/.

[5] Steven J. Mead Alicia Fornes Bisquerra Fred Charles, Miguel Lozano and Marc Cavazza. Planning formalisms and authoring in interactive storytelling. *Journal of visual communication and image representation*, pages 1–10, 2007.

[6] Otto Gonzalez. ¿qué es audacity? edita pistas como si fueras martin garrix. https://www.crehana.com/blog/transformacion-digital/que-es-audacity/.

[7] Gumsup. Conoce los 6 diferentes tipos de jugadores en gamificacion segun andrzej marczewski. https://gumsup.com/blog/conoce-los-6-diferentes\-tipos-de-jugadores-en-gamificacion-segun-andrzej-marczewski/.

[8] OpenToonz. Acerca de opentoonz. https://opentoonz.github.io/es/index.html.

[9] David Oña. Qué es la narrativa en los videojuegos y por qué no debes confundirla con historia, trama, guión, lore y otros términos. https://es.ign.com/mario-1/173328/feature/que-es-la-narrativa-en-los-videojuegos-y-por-que-no\-debes-confundirla-con-historia-trama-guion-lore.

[10] Playmotiv. Conoce los tipos de jugadores de gamificación y cómo cautivarlos. https://playmotiv.com/tipos-de-jugadores-gamificacion.

[11] César Rebolledo. Guía undertale, trucos y consejos. https://vandal.elespanol.com/guias/guia-undertale-trucos-y-consejos.

[12] Alejandro Villar Rubio. Creación automática de equipos de salas de escape basada en el estilo de juego de los jugadores. pages 1–129, 2021-2022.

[13] Santi Seguí. Until dawn, guía completa - consejos estratégicos. https://as.com/meristation/2015/08/26/guia_pagina/1440590161_148179.html.

[14] Unity. Plataforma de unity. https://unity.com/es/products/unity-engine.

[15] Unity. Powering cameras for films and games. https://unity.com/unity/features/editor/art-and-design/cinemachine.

[16] Luis López Zamorano. Cómo ver todos los finales de heavy rain. https://www.hobbyconsolas.com/noticias/como-ver-todos-finales-heavy-rain-272437.