Project ECO

Autor: Jose A. Artero Fernández
Tutor: Gus Marcos Ballester.
Profesor: Joan Arnedo Moreno

Grado en Ingeniería Informática
Área de Ingeniería del software

Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Project JAAF (provisional)*
Nombre del autor:	*Jose Antonio Artero Fernández*
Nombre del colaborador/a docente :	Gus Marcos Ballester
Nombre del PRA:	Joan Arnedo Moreno
Fecha de entrega (mm/aaaa):	*06/2023*
Titulación o programa:	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*TFG Videojuegos*
Idioma del trabajo:	*Español*
Palabras clave	*Aventura, Unreal Engine, Acción*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo*

El Trabajo de Final de Grado se basa en realizare un videojuego utilizando y desarrollando las aptitudes adquiridas durante el grado de ingeniería informática en la Universitat Oberta de Catalunya. El desarrollo se sustenta en el motor gráfico Unreal Engine 5, motor de carácter general muy extendido dentro de los videojuegos, y la utilización de assets sólo para el apartado gráfico desarrollándose para el TFG todo el diseño y la programación en C++.

En el transcurso del trabajo se busca obtener un producto entregable y jugable dentro del segmento del PC, basado en el género de las aventuras de acción dónde se implementan las mecánicas y avances propias del género. Dicho genero trata de explicar una historia a medida que el jugador avanza en busca de objetivos que le permitan finalizar la obra, para ello, dentro de esta historia se introducen distintos tipos de fases que ayuden a crear un entorno dinámico que atraiga y cautive al participante.

Se pretende obtener un producto de calidad acorde con los estándares actuales de la industria del sector del videojuego, aunque con una duración mucho menor pero suficiente para transmitir una experiencia agradable.

Abstract (in English, 250 words or less):

The F.D.P is based on making a video game using and developing the skills acquired during the computer engineering degree at the Universitat Oberta de Catalunya. The development is based on the Unreal Engine 5 graphic engine, a general engine that is very widespread within video games, and the use of assets only for the graphic section, developing all the design and programming in C++ for the TFG.

In the course of the work, the aim is to obtain a deliverable and playable product within the PC segment, based on the genre of action adventures where the mechanics and advances of the genre are implemented. The game tries to explain a story as the player advances in search of objectives that allow him to finish the work, within this story different types of phases are introduced that help create a dynamic environment that attracts and captivates the participant.

It is intended to obtain a quality product in accordance with current industry standards in the video game sector, although with a much shorter duration but sufficient to convey a pleasant experience.

Resumen

El Trabajo de Final de Grado se basa en realizare un videojuego utilizando y desarrollando las aptitudes adquiridas durante el grado de ingeniería informática en la Universitat Oberta de Catalunya. El desarrollo se sustenta en el motor gráfico Unreal Engine 5, motor de carácter general muy extendido dentro de los videojuegos, y la utilización de assets sólo para el apartado gráfico desarrollándose para el TFG todo el diseño y la programación en C++.

En el transcurso del trabajo se busca obtener un producto entregable y jugable dentro del segmento del PC, basado en el género de las aventuras de acción dónde se implementan las mecánicas y avances propias del género. Dicho genero trata de explicar una historia a medida que el jugador avanza en busca de objetivos que le permitan finalizar la obra, para ello, dentro de esta historia se introducen distintos tipos de fases que ayuden a crear un entorno dinámico que atraiga y cautive al participante.

Se pretende obtener un producto de calidad acorde con los estándares actuales de la industria del sector del videojuego, aunque con una duración mucho menor pero suficiente para transmitir una experiencia agradable.

Abstract

The F.D.P is based on making a video game using and developing the skills acquired during the computer engineering degree at the Universitat Oberta de Catalunya. The development is based on the Unreal Engine 5 graphic engine, a general engine that is very widespread within video games, and the use of assets only for the graphic section, developing all the design and programming in C++ for the TFG.

In the course of the work, the aim is to obtain a deliverable and playable product within the PC segment, based on the genre of action adventures where the mechanics and advances of the genre are implemented. The game tries to explain a story as the player advances in search of objectives that allow him to finish the work, within this story different types of phases are introduced that help create a dynamic environment that attracts and captivates the participant.

It is intended to obtain a quality product in accordance with current industry standards in the video game sector, although with a much shorter duration but sufficient to convey a pleasant experience.

Palabras clave

Unreal Engine, Aventura, Acción

Índice

Figuras y tablas

Índice de figuras

Índice de tablas

1. Introducción

1.1. Introducción

Dentro del entretenimiento y el ocio, el consumidor, busca evadirse y disfrutar del momento por medio de distintos métodos que le permitan abstraerse del mundo actual y, además, vivir experiencias que de otro modo no serían posibles. Dentro de este ámbito nos podemos encontrar obras audiovisuales, entre otras, que introducen al consumidor como espectador de unos hechos que narran una historia. Dentro de las obras audiovisuales, disponemos de la rama de los videojuegos en donde el consumidor pasa de ser un mero espectador a dirigir la experiencia y ser parte de la obra.

La evolución que vive año tras año nos lleva a preguntar dónde estará el límite en el desarrollo de estos productos y que nos ofrecerá el futuro con el avance de la tecnología. Ya queda muy atrás los primeros juegos como el Galaxy Game de 1971 o el Pong de 1972, ahora la industria ofrece grandes producciones que nada tienen que envidiar a las películas de Hollywood como es el caso de Horizon Forbidden West de Guerrilla Games que muestran el potencial de esta rama del entretenimiento.

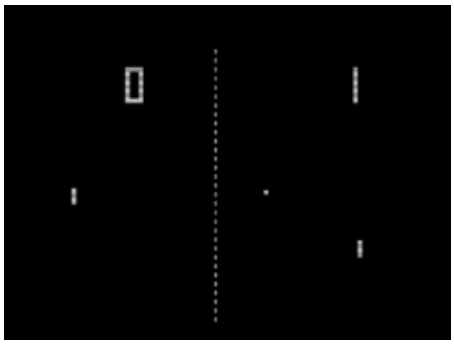


Figura 1: Pong 1072, Atari

Figura 2: Horizon Forbidden West 2022, Sony-Guerrilla Games

Esta gran evolución unida a que el sector de los videojuegos necesita abarcar muchas ramas de desarrollo, como programación, audio, diseño, y que llega a prácticamente cualquier dispositivo con un mínimo de potencia como teléfonos, tablets, wearables, consolas o RV lo que hace a este un excelente sector de crecimiento personal a cualquier persona que está involucrada en cualquier fase de su desarrollo y quiera introducirse en otras áreas.

1.2. Definición del trabajo

El proyecto pretende finalizar con videojuego totalmente funcional y que contenga todas las características que se esperan del género de aventura y acción. Se trata de una primera incursión de en este mundo y se inicia sin conocimientos previos en este campo, de modo que durante el desarrollo del proyecto se pretende adquirir las competencias necesarias que permitan iniciarse en el desarrollo de videojuegos de forma profesional.

Con la premisa de falta de conocimiento anterior sobre el desarrollo de videojuegos, se define una parte importante del trabajo a realizar, siendo esta el estudio y aprendizaje del motor gráfico escogido. Para realizar el proyecto se decide utilizar Unreal Engine, siendo este un referente cuando se quiere realizar proyectos con índole más fotorrealista, característica que se quiere incluir en el diseño de la obra propuesta por medio de la incorporación de la herramienta de Quixel Megascans. Esta herramienta consiste en un repositorio de materiales, texturas y otros elementos de alta calidad preparados para ser utilizados directamente con Unreal y su tecnología Nanite.

Unreal ofrece una gran calidad, pero su utilización exige un mayor aprendizaje y es más complejo que otras alternativas como Unity. Este motor establece dos sistemas de trabajo, mediante C++ y otra llamada Blueprints (programación gráfica), aunque la base del proyecto será en C++ también se incorpora este segundo sistema para algunas tareas menores.

El alcance se establece alrededor de las materias adquiridas en el grado de modo que se pongan en práctica y se profundicen en ellas, por lo tanto, queda fuera el aspecto gráfico del juego. En este caso se utilizarán assets de terceros que proporcionen el aspecto gráfico y sonoro al juego, pero la programación como movimientos, ataques, estados, o cualquier interacción se desarrollará en el transcurso del proyecto, en C++ mayoritariamente.

La aventura tiene un desarrollo sencillo, está pensada para ser jugada en dos niveles diferentes, el primer nivel es de preparación y búsqueda, con una aproximación al combate, y el segundo establece un ritmo de combate más alto para llegar a un jefe final.

El juego inicia en una zona costera, con bosque y playa, donde el jugador deberá adquirir las armas y un medallón para poder lanzar ataques mágicos. En esta zona se establecerá un NPC que orientará por medio de conversaciones al jugador de donde puede encontrar el equipo. El segundo nivel inicia, tras algún combate de aprendizaje, entrando en una cueva situada en una montaña llena de enemigos para llegar al centro de esta y derrotar al enemigo final.

Al ser una aventura se incorpora una historia que explique la razón del juego, en este caso el jugador es un explorador espacial que tras un accidente termina en un planeta desconocido en donde debe ayudar a la gente del lugar. Los aldeanos son atacados por unas criaturas que se ocultan dentro de la montaña, en unas cuevas, y nuestro personaje deberá destruirlos y regresar a casa.

Las mecánicas son las comunes del sector, el personaje puede caminar, correr, saltar y esquivar como movimientos y puede atacar con una espada, con dos tipos de daño, y un ataque mágico. Dispone de nivel de vida y nivel de ataque mágico, estos dos se rellenarán más o menos rápido según el nivel actual del personaje de modo que se recuperará antes y podrá lanzar más ataques mágicos a mayor nivel. El nivel subirá a medida que se eliminen enemigos o se encuentren ítems que ayuden a tal fin.

A los enemigos finales se le implementará un sistema de IA avanzada para que actúen contra el personaje principal, deberán percibir estímulos y actuar acorde a ellos para intentar destruir al personaje intentando predecir o avanzarse a sus movimientos.

Debido a que actualmente existen dos formas de acceder a las obras publicadas, una en físico y otra vía internet, se procederá a implementar los dos sistemas. Un sistema clásico con instalación y un sistema de Cloud Gaming para el acceso por medio de un navegador que permita jugar sin necesidad de ningún otro requisito más allá de un pc y una conexión a internet.

1.3. Objetivos generales

En un estado inicial se establece un conjunto de objetivos que definen el trabajo, de modo que se muestra la idea general propuesta y que se pretende realizar, siendo estos por orden:

- Adquirir conocimientos en el ámbito de los videojuegos por medio de la utilización del motor Unreal Engine de Epic Games, aprendizaje del software.
- Ampliar la base de conocimiento del lenguaje C++ adquirido en la UOC con el necesario para desarrollar en el motor gráfico.
- Aprendizaje de las distintas fases que son necesarias para la elaboración de un videojuego.
- Organización y planificación del proyecto durante todas las fases de este y alcanzar los hitos propuestos.
- Realización de un juego de aventura y acción tipo Kena: Bridge of Spirits.
- Establecer un sistema de IA para los enemigos en el juego.
- Establecer dos formas de acceso al juego, formato físico y un sistema de streaming.

No es tarea de este trabajo de Fin de Grado realizar un proyecto de gran envergadura ni que se incorpore características no esenciales que alarguen el desarrollo. Se debe entender la complejidad de iniciar un proyecto sin conocimientos previos y acotar el alcance de este, de lo contrario cabe la posibilidad de no cumplir los tiempos estipulados y hacer el proyecto inviable.

1.3.1. Objetivos principales

Objetivos de la aplicación/producto/servicio:

- Desarrollo de un videojuego de aventura y acción.
- Desarrollo de la IA.
- Establecer los servicios en la nube para ofrecer la opción de jugar online.

Objetivos para el cliente/usuario:

- Ayuda para aprender a utilizar la interfaz del juego.
- Interfaz y HUD amigable de cara al usuario.
- Acceso al juego sencilla e instalación de forma desatendida.
- Acceso al sistema online.

Objetivos personales del autor del TF:

- Aprendizaje del funcionamiento del motor gráfico y de plugin, en caso de ser necesarios.
- Ampliar conocimientos sobre el lenguaje C++

- Conocer y entender las fases y procesos de creación de videojuegos.
- Adquirir experiencia en la gestión de proyectos.
- Utilización de herramientas de versiones
- Utilización de herramientas de testeo

1.3.2. Objetivos secundarios

Objetivos adicionales que enriquecen el TF.

- Profundizar más en la programación en C++, de modo que adquiera un conocimiento más asentado y profesional.
- Diseño y animación de personajes y objetos.

1.4. Metodología y proceso de trabajo

Para realizar el proyecto, se opta a realizar un producto nuevo puesto que crear una versión obligaría a negociaciones con el propietario de la obra original, con lo cual retrasaría u obligaría a destinar una parte del presupuesto a pagar licencias. De igual modo todos los assets y material externo al proyecto cuentan con licencia libres para usos no comerciales.

La planificación inicial se enfoca en gestionar lo mejor posible el tiempo y recursos de que se dispone, al ser una única persona la que realiza el proyecto es preferible centrarse en cada punto hasta finalizarlo para pasar al siguiente y en consecuencia se decide utilizar una metodología en cascada, que al ser un sistema más estricto y monolítico provoca que los esfuerzos no estén dispersos en varias tareas de modo que se evitan confusiones o errores. El problema de este método reside en no tener una primera versión hasta muy avanzado el desarrollo.

En este caso y debido al carácter finito de la práctica, no se incorpora un mantenimiento del software entregado y por lo tanto las fases serian cuatro, pero al ser desarrollado por una única persona se considera dividir el bloque de implementación en dos partes para controlarlo de forma más eficiente.

Las fases son:

- Análisis, se trata de analizar y valorar el proyecto, establecer los recursos necesarios en hardware, software, personal, económico y viabilidad, entre otros, para poder llevar su realización hasta obtener el entregable.

- Diseño, en esta fase se estudiará y creará de forma detalla el concepto del juego y la infraestructura necesaria para llevarlo a la siguiente fase. Todo el diseño se centrará en realizar un producto mínimo viable (MVP) con la intención de obtener una entrega que cumpla con los requisitos indicados.

- Implementación de personajes, como se centra en conseguir un producto mínimo viable se inicia el proceso en las partes del control del jugador principal, sus características y la interacción del entorno por medio de estados e ítems, seguidamente se establece el HUD del juego para obtener la información en pantalla del estado de la partida. El siguiente paso se decide el desarrollo del menú principal y el menú de pausa y fin de partida básicos, sin desarrollar subapartados dentro de estos como guías o guardados de partidas.

  Se continúa creando el resto de los personajes y enemigos del juego de modo que toda la codificación principal quede en un estado muy avanzado.

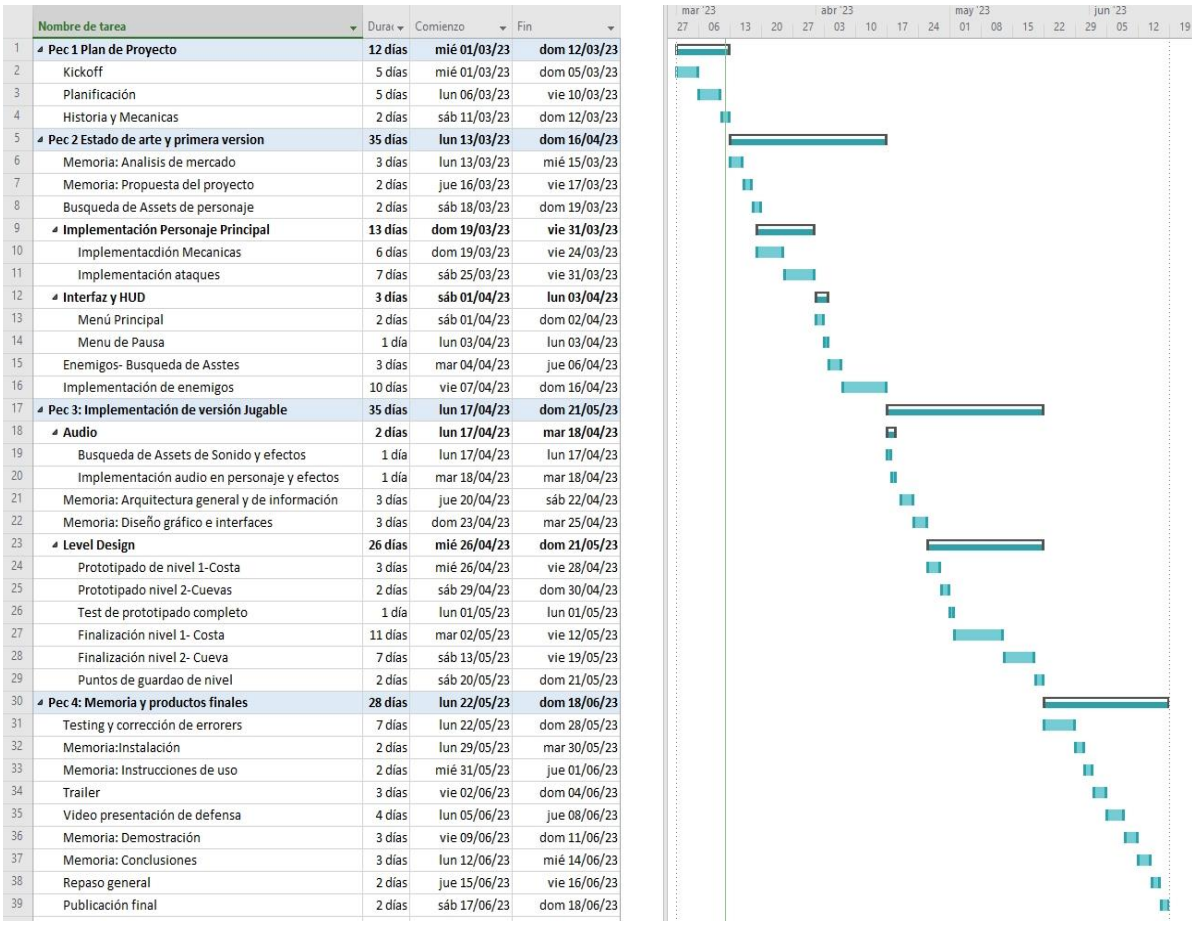
- Implementación de niveles, en este punto se prevé iniciar el prototipado de niveles y la construcción final de estos una vez aprobados los prototipos, en este punto toda la parte gráfica será implementada con assets y materiales de terceros. Al finalizar esta fase se dispondrá de un proyecto funcional y totalmente operativo para pasar a la siguiente fase.

- Testeado y correcciones, esta fase será la dedicada a testear el funcionamiento correcto del juego y refinar los procesos que no estén del todo lo bien finalizados que se debiera si sobrase tiempo.

1.5. Planificación

La planificación se basa por un lado en el calendario de las distintas PECs y por otro lado por la secuencia prevista para abordar la implementación del juego. Las fechas claves de la planificación se establece en las indicadas para la entrega de cada PEC, de modo que se desarrolle las tareas y su entrega dentro de las fechas permitidas.

La primera imagen muestra un diagrama de Gantt que permite hacer un seguimiento de los tiempos y fechas limites para cada hito dentro del proyecto.



	Nombre de tarea	Dura	Comienzo	Fin
1	**Pec 1 Plan de Proyecto**	**12 días**	**mié 01/03/23**	**dom 12/03/23**
2	Kickoff	5 días	mié 01/03/23	dom 05/03/23
3	Planificación	5 días	lun 06/03/23	vie 10/03/23
4	Historia y Mecanicas	2 días	sáb 11/03/23	dom 12/03/23
5	**Pec 2 Estado de arte y primera version**	**35 días**	**lun 13/03/23**	**dom 16/04/23**
6	Memoria: Analisis de mercado	3 días	lun 13/03/23	mié 15/03/23
7	Memoria: Propuesta del proyecto	2 días	jue 16/03/23	vie 17/03/23
8	Busqueda de Assets de personaje	2 días	sáb 18/03/23	dom 19/03/23
9	**Implementación Personaje Principal**	**13 días**	**dom 19/03/23**	**vie 31/03/23**
10	Implementacdión Mecanicas	6 días	dom 19/03/23	vie 24/03/23
11	Implementación ataques	7 días	sáb 25/03/23	vie 31/03/23
12	**Interfaz y HUD**	**3 días**	**sáb 01/04/23**	**lun 03/04/23**
13	Menú Principal	2 días	sáb 01/04/23	dom 02/04/23
14	Menu de Pausa	1 día	lun 03/04/23	lun 03/04/23
15	Enemigos- Busqueda de Asstes	3 días	mar 04/04/23	jue 06/04/23
16	Implementación de enemigos	10 días	vie 07/04/23	dom 16/04/23
17	**Pec 3: Implementación de versión Jugable**	**35 días**	**lun 17/04/23**	**dom 21/05/23**
18	**Audio**	**2 días**	**lun 17/04/23**	**mar 18/04/23**
19	Busqueda de Assets de Sonido y efectos	1 día	lun 17/04/23	lun 17/04/23
20	Implementación audio en personaje y efectos	1 día	mar 18/04/23	mar 18/04/23
21	Memoria: Arquitectura general y de información	3 días	jue 20/04/23	sáb 22/04/23
22	Memoria: Diseño gráfico e interfaces	3 días	dom 23/04/23	mar 25/04/23
23	**Level Design**	**26 días**	**mié 26/04/23**	**dom 21/05/23**
24	Prototipado de nivel 1-Costa	3 días	mié 26/04/23	vie 28/04/23
25	Prototipado nivel 2-Cuevas	2 días	sáb 29/04/23	dom 30/04/23
26	Test de prototipado completo	1 día	lun 01/05/23	lun 01/05/23
27	Finalización nivel 1- Costa	11 días	mar 02/05/23	vie 12/05/23
28	Finalización nivel 2- Cueva	7 días	sáb 13/05/23	vie 19/05/23
29	Puntos de guardao de nivel	2 días	sáb 20/05/23	dom 21/05/23
30	**Pec 4: Memoria y productos finales**	**28 días**	**lun 22/05/23**	**dom 18/06/23**
31	Testing y corrección de errorers	7 días	lun 22/05/23	dom 28/05/23
32	Memoria:Instalación	2 días	lun 29/05/23	mar 30/05/23
33	Memoria: Instrucciones de uso	2 días	mié 31/05/23	jue 01/06/23
34	Trailer	3 días	vie 02/06/23	dom 04/06/23
35	Video presentación de defensa	4 días	lun 05/06/23	jue 08/06/23
36	Memoria: Demostración	3 días	vie 09/06/23	dom 11/06/23
37	Memoria: Conclusiones	3 días	lun 12/06/23	mié 14/06/23
38	Repaso general	2 días	jue 15/06/23	vie 16/06/23
39	Publicación final	2 días	sáb 17/06/23	dom 18/06/23

Figura 3:: Diagrama de Gantt

Como se indico en la introducción de este punto, las fechas claves son la s especificadas por cada PEC, esos intervalos de fechas quedan reflejados junto a todos los hitos en la figura 3 anterior. En la siguiente tabla se exponen de forma más concreta e inequívoca las fechas claves según cronología de las PECs.

	Fechas claves del proyecto
PEC 1: Plan de proyecto	13 de marzo de 2023
PEC 2: Estado de arte y primera versión	16 de abril de 2023
PEC 3: Implementación de versión jugable	21 de marzo de 2023
PEC 4: Memoria y productos finales	18 de junio de 2023

Tabla 1: Fechas clave del proyecto

1.6. Presupuesto

En el desarrollo del proyecto intervienen diferentes aspectos que deben valorarse, como principales se consideran el coste de trabajo (hardware y software a utilizar) y el equipo humano.

1.6.1. Estimación de coste de trabajo

Hardware: Para realizar la estimación del hardware necesario para el proyecto se accede a la página oficial de Epic Games[1], en ella aparecen los requisitos mínimos, los recomendados y los idóneos. Se busca una relación precio-desarrollo para obtener un flujo de trabajo estable y ágil, por lo tanto, se busca un equipo que se encuentra entre el recomendado y el idóneo, puesto que puede ser que algunos de los avances que incorpora Unreal en iluminación necesiten más potencia.

Concepto	Valor
Pc PcCom Gold Elite	1.297,07€
Samsung Odyssey G3 27”	256,37€
Samsung LS24AM506NUXEN 24”	214,44€
Logitech Desktop MK120 Combo	29,99€
Logitech Z533 Multimedia Speaker	139€
Total	**1.936,87**

Tabla 2: Estimación coste hardware

El software será la parte más económica, puesto que Unreal es de uso gratuito y tan solo se paga regalías a partir de unas determinadas ventas, 5% a partir de 1 millón de dólares. Por lo tanto, el software será el propio de la máquina, uno de diseño gráfico y de edición de video para complementar otras áreas como marketing o algunos diseños.

Concepto	PVP	Cantidad	Valor total
Windows 11 pro Edition	199,99€	1	199,99 €
Adobe Creative Cloud	73,49€/mes	4	293,96 €
Unreal Engine 5.1	0	4	0
Total			**493,95**

Tabla 3: Estimación coste del software

1.6.2. Estimación del coste del equipo humano

Para poder hacer la estimación del equipo humano se debe conocer el salario de un desarrollador de videojuegos en España [2]. Este salario será el utilizado para calcular los cuatro meses de duración del desarrollo del proyecto.

Concepto	Valor
Salario bruto anual	32.100 €
Duración del proyecto	4 meses
Coste por desarrollador	10.700 €
Coste del equipo humano	**10.700€**

Tabla 4: Estimación del coste del equipo humano

1.6.3. Estimación final

La siguiente tabla muestra la estimación final del coste de realizar el proyecto, a los conceptos se le debe sumar un importe de un 15% de contingencia para posibles imprevistos.

Concepto	Valor
Estimación trabajo hardware	1.936,87€
Estimación trabajo software	493,95€
Estimación e. humano	10.700€
Subtotal	13.130,82€
Contingencia 15%	1.969,62€
Total	**15.100,44€**

Tabla 5: Estimación final del proyecto

En la estimación no se incluyen assets o material de terceros ya que se intentará utilizar material libre de derechos o con reconocimiento. Todos los precios son extraídos de las páginas de cada producto a fecha de realización de esta documentación, por lo que puede sufrir variaciones posteriores.

1.7. Estructura del resto del documento

El documento se compone de un total de 7 capítulos, incluyendo el actual capitulo.

1. Introducción.

Capítulo de justificación del trabajo de fin de grado, contextualización y planificación del desarrollo.

2.Estado de arte

En este capítulo se realiza un análisis del mercado de los videojuegos, se muestran antecedentes y se analiza los distintos motores del mercado.

3.Definición del juego

Se incluye la propuesta del proyecto a realizar, incorpora una conceptualización de los elementos del juego y su funcionamiento.

4. Diseño

Introducción más técnica al proyecto, se incluye descripción de los componentes, requisitos y assets de terceros utilizados.

5.Implementación

Procesos seguidos y métodos empleados para el desarrollo del juego. Concretando archivos y su relación dentro del juego.

6.Guia del usuario

Guía rápida sobre el funcionamiento y utilización del juego.

7.Conclusiones y Líneas de futuro.

Impresiones finales y propuestas de actualizaciones futuras.

2. Estado del arte

2.1. Antecedentes

En los inicios del género de acción y aventura [3], estos se ligaron a los lanzamientos de los famosos juegos de rol como Dungeons and Dragons, empezando a desarrollarse en los años 70 los primeros RPG del mercado en ordenadores, pero no fue hasta 1980 que apareció Adventure de Atari que algunos clasifican como el primer juego de acción y aventura.

Estos primeros juegos se realizaron en universidades y por lo tanto no se comercializaban, se compartían dentro de los campus, siendo algunos de los pioneros Dungeon, pedit5 o dnd. El primer juego en establecer la primera persona fue concretamente Moria, en 1975, siendo este un juego de RPG.[4]

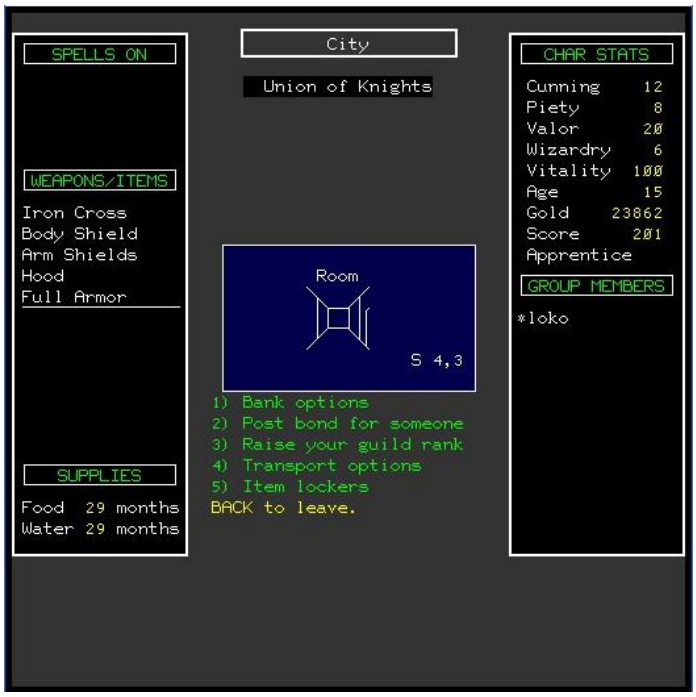


Figura 4 Moria 1975 fuente: https://exploradorrpg.wordpress.com/juegos/moria/

Actualmente un juego de aventura y acción se caracteriza por contar una historia y donde aparecen varios personajes como soporte a esta, permite evolucionar al personaje en varias habilidades físicas o cognitivas utilizando para ello la exploración del nivel, las resoluciones de rompecabezas, la investigación o la lucha.

Algunos de los juegos más famosos que se enmarcan en este género son Uncharted, The Secret of Monkey Island o el Kena: Bridge of Spirits.

Figura 5 Captura de Kena:Bridge of Spirits

Otro de los juegos relevantes dentro del panorama RPG es el The Legend of Zelda, juego perteneciente a la marca Nintendo y que tras su lanzamiento en 1986 cuenta con versiones para cada consola de la marca.

Figura 6 Cartucho de NES de “The Legend of Zelda”

Figura 7 Juego “The Legend of Zelda” de 1986

2.2. Estado del mercado

Según el informe Global Entretainment and Media Outlook 2022-26 de PWC[5] se espera que en 2026 la industria alcance 321.000 millones de dolares

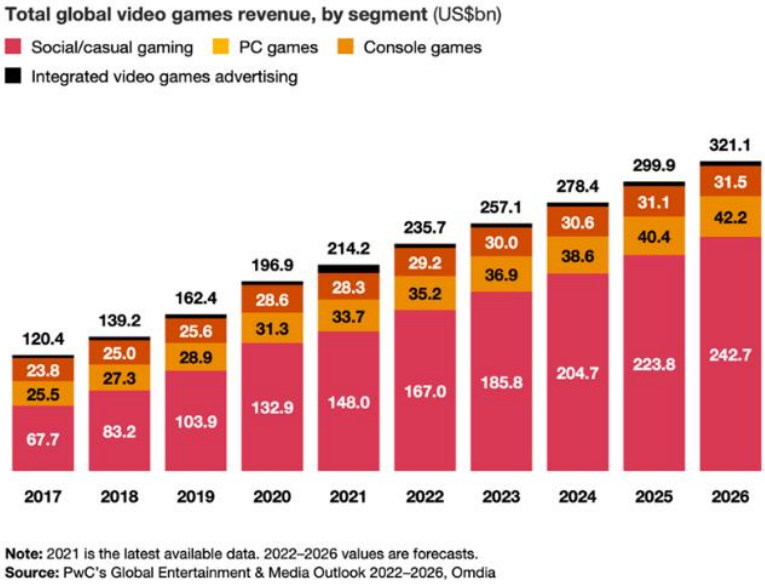


Figura 8 Previsión crecimiento del mercado fuente: El Foro Económico Mundial (weforum.org)

Este crecimiento es debido principalmente por la fase de confinamiento por causa del COVID-19, por lo que al encontrarse la gente confinada buscó una salida lúdica. De modo que entre 2019 y 2021 el mercado se expandió un 26%.[6]
Dentro de esta expansión encontramos que los juegos de acción gozan de una gran aceptación dentro del mercado, las ventas de estos juegos alcanzan niveles muy altos como es el caso de “The Legend of Zelda: Breath of the Wild” con cerca de 26 millones [7]o la gran aceptación de las adaptaciones a series como “The Last of Us”.

2.3. Motores de desarrollo

En la actualidad se utilizan motores gráficos para el desarrollo de videojuegos, entre otras cosas, que proporcionan una gran variedad de herramientas para la confección de todo el proyecto. Estos motores gráficos incluyen sistemas de animación, sistemas de renderizados, sistemas de partículas y físicas y cualquier otro factor que pueda ser necesario a la hora de ser utilizados.

Dentro de los estudios grandes de creación de videojuegos se utilizan motores propietarios como RE Engine de Capcom o Luminous Engine de Square-Enix, pero son inaccesibles al público en general de modo que disponemos de alternativas que están pensadas para el público en general. Estos motores

generalistas se basan en cobrar cuando el videojuego alcanza una cifra de ventas, cada motor tiene sus varemos para calcular las cantidades, de modo que cualquier estudio o persona puede hacer uso de ellos sin miedo a tener que hacer un desembolso inicial.

Por lo tanto, las dos alternativas más grandes o conocidas, que no únicas, son Unity y Unreal, las cuales se analizan a continuación.

2.3.1. Unity

Este motor fue creado por David Helgason, Nicholas Francis y Joachim Ante en 2004 y fue lanzado en 2005 en MAC para que el público en general tuviera acceso a la creación de videojuegos.

Este motor resulta ser más sencillo de utilizar que otros e incluye todos los apartados que se utilizan en el desarrollo de un juego, su lenguaje de programación es C#. La comunidad que está apoyando el proyecto es enorme con lo que es posible encontrar respuesta ante cualquier duda o problema de forma rápida.

Este motor resulta muy polivalente y permite la realización de juegos en todas las plataformas, siendo una gran herramienta que utilizan muchos de los estudios Indies.

2.3.2. Unreal

Unreal Engine está desarrollado por Epic Games, creadores de Fornite, y fue inicialmente creado para un juego tipo shooter llamado Unreal en 1998. Actualmente es uno de los motores más utilizados dentro de la industria, especialmente orientado al desarrollo 3D en videojuegos, arquitectura y films.

Este motor resulta más complejo a la hora de realizar cualquier desarrollo y su lenguaje de programación es C++, actualmente ofrece una serie de tecnología que lo desmarcan, por ahora, de sus competidores directo y que incluye el sistema Nanite y Lumen.

Debido a la orientación del motor al mundo del 3D, tanto su instalación como el trabajo diario con los proyectos necesitan la utilización de una gran cantidad de espacio en disco local como en servidores.

3. Definición del juego

3.1.1. Descripción del juego

La obra se trata de un videojuego de acción y aventura, se enmarca dentro de los juegos singleplayer donde un solo jugador es el encargado de desarrollar la acción de principio a fin. El jugador controla un personaje que debe superar dos niveles principales separados en áreas para poder finalizar la aventura de forma que su personaje evolucione mejorando y volviéndose más fuerte.

3.1.2. Objetivos propuestos al jugador

Se trata de que el jugador desarrolle una historia en unos escenarios por medio de misiones que harán que el personaje evolucione y consiga avanzar. Para conseguir los objetivos y para poder avanzar en la experiencia se proponen adquirir una serie de equipamiento, para lo cual el personaje deberá moverse por escenario de forma secuencial. Aunque cada escenario otorga movimiento libre a cualquier área de este, sin completar el objetivo anterior no es posible acceder al siguiente.

3.1.3. Plataforma destino

El diseño de la obra se centra en el ámbito de los ordenadores personales, debido a la gran capacidad que tienen estos a de adaptarse a las exigencias que se necesitan en cada momento de ejecución. En este punto cabe recordar que también se prevé la ejecución en la nube para ser jugado sin necesidad de instalación por lo que se deberá adaptar para su incorporación en servidores.

3.2. Conceptualización

3.2.1. Historia

El protagonista es una exploradora de mundos que tras sufrir un accidente con su nave despierta en un poblado desértico. Tras explorar un poco encuentra un reducido grupo de guerreros que intentan hacer frente a unas hordas de criaturas que aparecen desde las montañas para atacar y destruir a los que allí viven.

Los aldeanos se desplazaron a islas cercanas para sobrevivir y los guerreros intentan frenar los ataques de las criaturas, pero sus armas apenas causan daño a esos seres y los únicos objetos que pueden derrotarlos están maldecidos para que nadie de la isla pueda hacerse con ellos.

Nuestra protagonista es la única que puede hacerse con estos objetos puesto que ella no pertenece a la isla y por lo tanto debería poder hacerse con las armas necesarias para derrotarlos de una vez. De modo que siguiendo las indicaciones del grupo empieza un viaje por la isla para encontrar los objetos malditos y finalmente limpiar la isla de cualquier rastro de esos malvados engendros del mal.

3.2.2. Ambientación

La aventura transcurre en dos escenarios principales y consecutivos. Tras conseguir los objetivos del primer nivel se puede pasar al segundo.
Para realizar el arte de los niveles se recurre por un lado a los elementos gratuitos de la Epic Store y por otro lado de la plataforma de Quixel Megascans, donde se extraen los elementos que posteriormente se utilizan para dar la ambientación al juego. Los diseños que encuentran de forma independiente por lo que todo el diseño y montaje de la escena se realiza manualmente.
En cuanto la base de los niveles, el terreno y su textura se realizan íntegramente a mano, con las herramientas incluidas en el motor e incluyendo los materiales y texturas ofrecidas por Unreal.

os niveles son:
Nivel Isla: Es el primer nivel de inicio del juego, la aventura empieza en una isla compuesta por cuatro zonas donde realizar acciones.

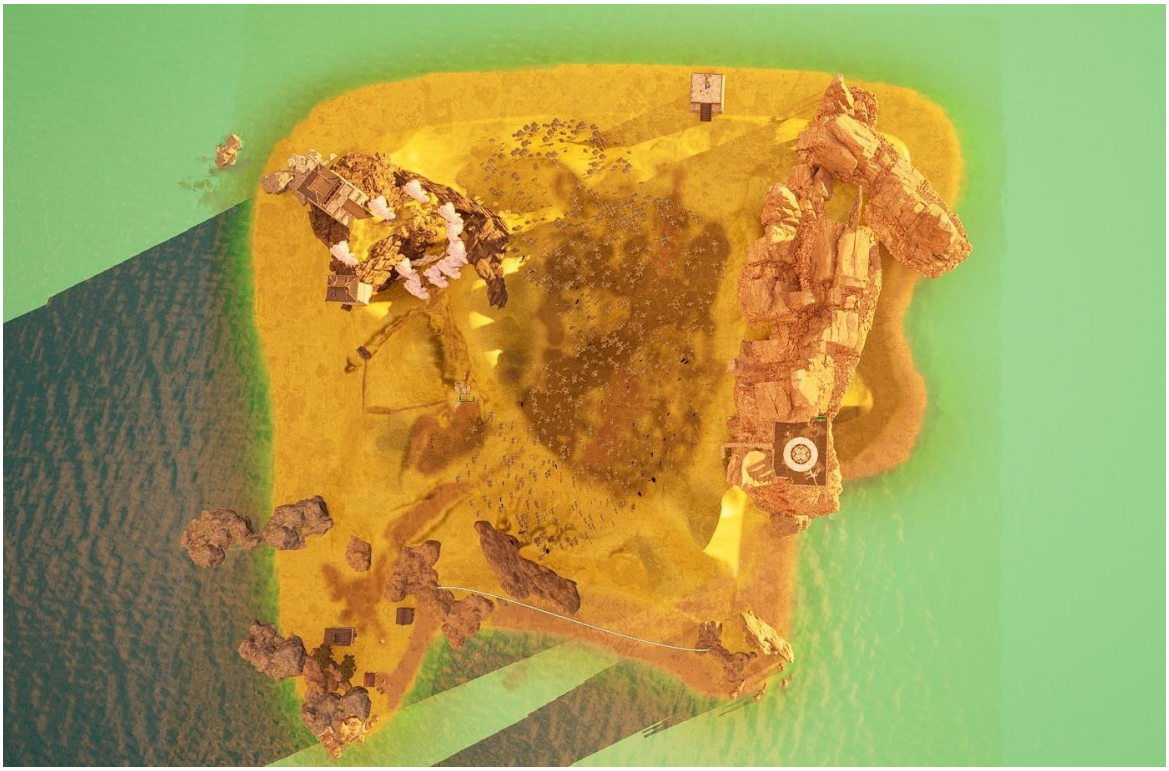


Figura 9 Nivel principal, isla.

Pueblo: zonal donde empieza el juego y es el punto de neurálgico de la aventura, en el se encuentran los NPC que guían al jugador.

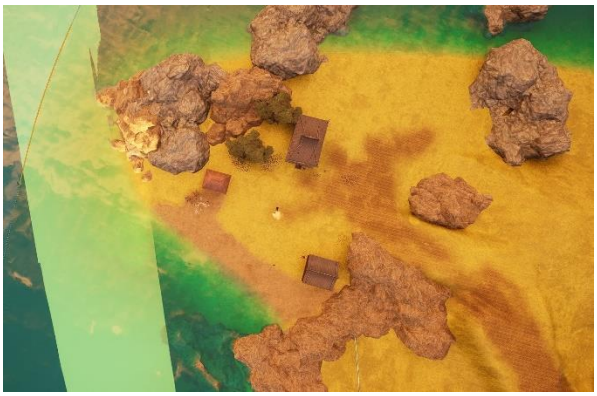


Figura 10 Poblado

Bosque del norte: zona compuesta por un gran bosque donde se encuentra uno de los objetivos de la aventura en un altar.

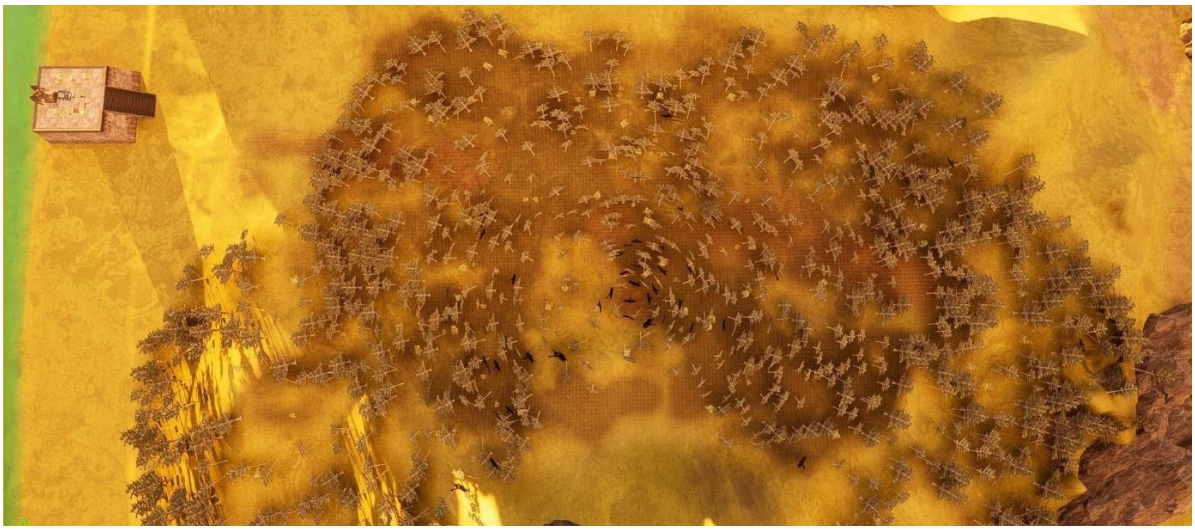


Figura 11 Bosque del norte

Templo en la montaña: se trata de una bonita zona con un templo protegido por las rocas de la montaña cuyo acceso está bloqueado.

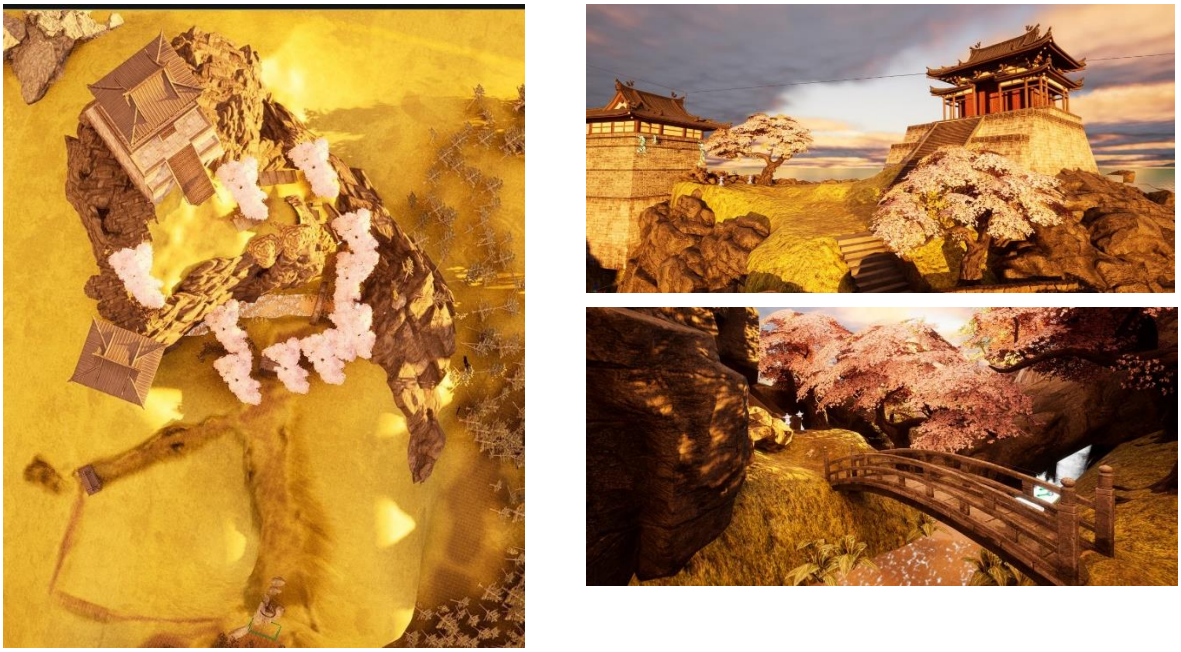


Figura 12 Templo en la montaña

Ruinas antiguas: lugar de nexo entre los dos mundos, esta zona proporciona un portal de paso entre la isla y el mundo maldito.

Figura 13 Ruinas antiguas

Cueva: Segundo nivel del juego es un nivel lineal que empuja al jugador a ir pasando secciones hasta llegar al final, este nivel contiene la mayoría de los enemigos del jugo. Al contrario que la isla, es un lugar oscuro compuesto por ruinas de un antiguo templo y cuevas milenarias.

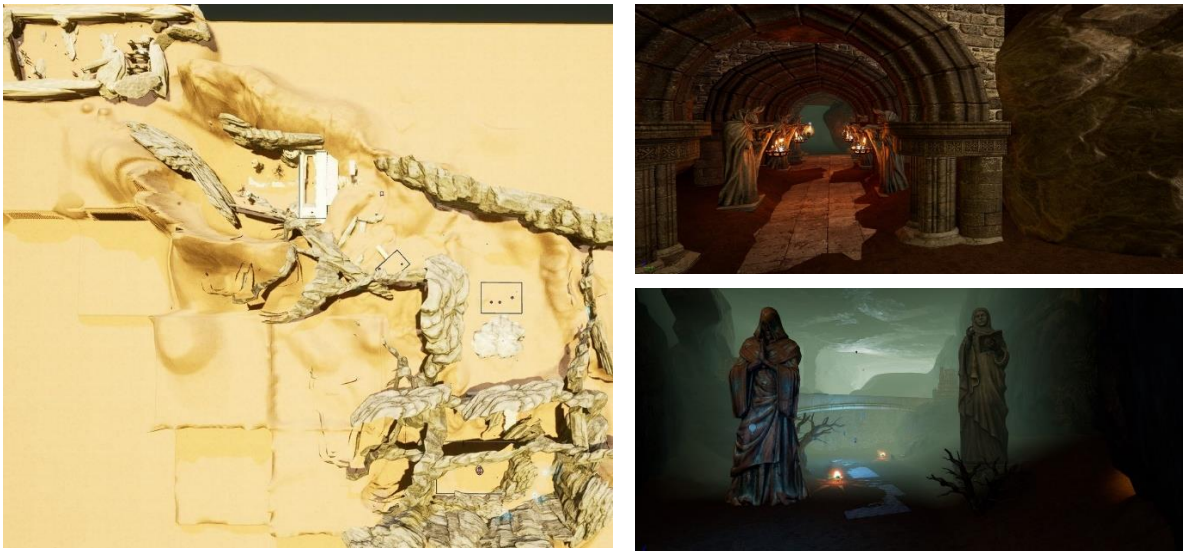


Figura 14 Nivel de la cueva, lugar maldito.

3.3. Personajes

El personaje principal del juego, los NPC y los enemigos básicos son assets y animaciones extraídas de la página Mixamo, esta plataforma permite adjuntar modelos precargados con multitud de animaciones para obtener finalmente un modelo animado personalizado. El enemigo final pertenece a la página de Epic Games, dentro de los complementos gratuitos que disponen para los desarrolladores.

3.3.1. Principal

El personaje principal de la obra es Den, una mujer fuerte y decidida, es una persona que no duda en ayudar a los demás. Entre sus habilidades se encuentra una gran pericia en utilizar la espada. Al personaje se le añade las armas, los efectos de partículas y sonidos para poder ser utilizado en el juego.

Figura 15 Den

3.3.2. Personajes no jugables

Como juego de acción y aventura, el personaje principal interacciona con otros individuos, estos se encargan de guiar al jugador al objetivo correspondiente en cada momento por medio de conversaciones. Disponemos de varios personajes no jugables (NPC) que realizan acciones y que en mayor o menor medida interaccionan con el jugador. Estos personajes cuentan la historia por lo que se le incorpora un sistema de dialogo que permite avanzar en el juego, este sistema se diseña dentro del motor para explicar la historia o misión o responder con una frase aleatoria.

Eve

Arissa

Morak

Brutus

Erika

Figura 16 Personajes no jugables

3.3.3. Enemigos

El juego dispone de tres personajes que ejecutan el rol de enemigos, dos de ellos se consideran de nivel inferior puesto que su daño es menor, y un tercero que es el enemigo final que causa más daño. Los personajes tienen movimientos básicos, por lo que se implementa el sistema de daño, vida y puntuación para interactuar con ellos. Del mismo modo que el personaje principal, las animaciones son complementadas con la inclusión de sonidos y efectos para dotar de mayor realismo la animación

- **Gólem**: se trata de un enemigo que se mueve relativamente rápido y ejecuta un combo de ataque melee, por lo tanto, el jugador puede recibir tres golpes consecutivos. A este enemigo no le afecta el ataque de magia a distancia.

Figura 17 Enemigo melee

- **Criatura**: este enemigo realiza un ataque de melee a una mano cuando el personaje está cerca, pero en el caso de estar a una distancia considerable puede realizar un ataque mágico, en este caso el personaje se mueve de forma más lenta y errática.

Figura 18 Enemigo melee y distancia

- **Jefe final**: la vida de este enemigo es muy superior a la del resto del juego, debido a su tamaño puede desplazarse de forma rápida y puede ejecutar dos ataques. El primero es un golpe que causa daño a todo lo que se encuentre a su alrededor y el segundo es una batería de proyectiles que lanza a distancia.

Figura 19 Jefe final

3.3.4. Mecánicas

Dentro del juego es posible ejecutar diferentes movimientos que aumentaran según avance en la aventura. En un inicio el personaje principal pude ejecutar los movimientos básicos de caminar, correr, saltar y rodar. La espada es añadida de forma que puede verse en la mano o colocada en la espalda del personaje, debido a que es un elemento externo de este se debe dotar de interacción con los enemigos para causar daño.

Caminar

Correr

Rodar

Saltar

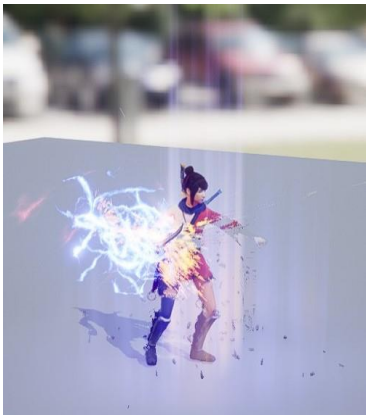
Figura 20 Movimientos básicos del protagonista

Una vez alcance cada objetivo planteado, el jugador, adquirirá un nuevo movimiento que estará vinculado al equipamiento que se utiliza.
En el momento de obtener la espada, el jugador obtendrá dos tipos de movimientos diferentes, un ataque normal con la espada que causa un daño puntual y un ataque mágico de espada que causa un daño alrededor del jugador pero que consume nivel de magia.

Cuando se alcance el objetivo del collar, el personaje obtendré un nuevo movimiento de magia. Este movimiento consta de del lanzamiento de un rayo a distancia que infringe daño, pero es impreciso y causa menor daño que la espada.

Espada

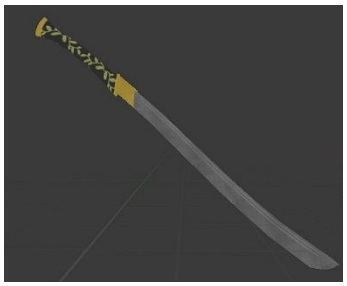
Espada mágica



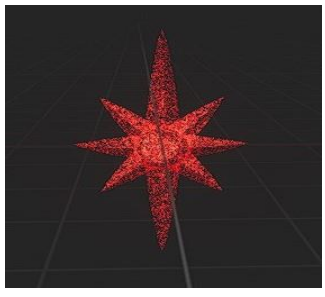
Rayo mágico

Figura 21 Ataques adquiridos en el transcurso del juego

El juego cuenta con tres ítems u objetos, dos de ellos son los dos objetivos principales de la aventura siendo estos la espada y el colgante mágico, el tercero se trata de una flor que proporciona al jugador la regeneración parcial de vida.



Espada



Colgante Mágico

Flor de vida

Figura 22 Ítems del juego

3.3.5. Vida, magia y puntos

Dentro del juego se estable un control de para la vida del jugador, enemigos e incluso de elementos del escenario, una vez agotada la vida de cualquier actor este morirá. Todo el diseño se realiza dentro del motor gráfico, utilizando elementos de Artstation, con licencia gratuita.

El jugador principal cuanta con una barra de color verde que permite conocer el nivel actual de vida de forma rápida e inequívoca, cada daño recibido restará una porción de la barra hasta que esta se agote y por lo tanto el jugador pierda la partida. El nivel en que disminuirá la barra dependerá del daño y no se estable en un valor único, por lo que cada enemigo puede infringir diferente cantidad daño según el ataque que realiza.
La vida cuenta con dos sistemas de regeneración, uno es al recoger una flor de vida que proporciona una recuperación instantánea de una porción de vida y la otra forma de regeneración es automática. Cuando el jugador no tiene la vida completamente regenerada, el sistema, empieza a aumentar el nivel de salud de forma automática con un tiempo establecido según el nivel de puntuación, por lo tanto, con un nivel uno será mucho más lento que en un nivel cuatro.

El nivel de magia, maná, se muestra con otra barra de color azul unida a la de vida, el funcionamiento es parecido al de la vida, si esta barra contiene suficiente nivel de magia el jugador podrá utilizar dicho ataque. La cantidad de ataques se establece en cuatro ataques por barra completa, por lo que hasta que el jugador no disponga como mínimo de un cuarto de dicha barra no podrá lanzar un ataque. La regeneración funciona de forma automática, al igual que la de vida, por lo que se autor regenera y aumentará más rápido en un nivel superior.

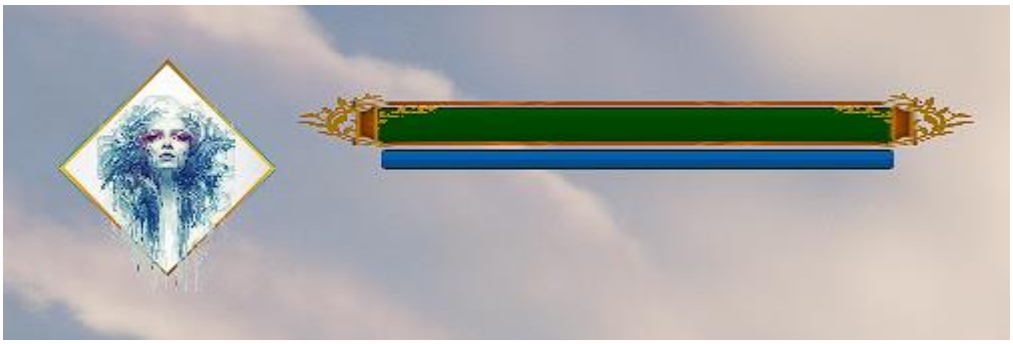


Figura 23 Barra de vida y de maná

Por otro lado, encontramos el nivel del jugador, este nivel aumenta a medida que se alcanza objetivos o se acaba con enemigos, en cada caso el valor de la puntuación será independiente para cada logro. Se dispone de cinco niveles, con el aumento de dichos niveles se consigue una regeneración de la salud y del nivel de maná, magia.

Figura 24 Barra de puntos y nivel

3.3.6. Interfaz e interacción

Se establece una interfaz para que el usuario sepa en todo momento el estado de su personaje, en pantalla se muestran el nivel de vida y maná por medio de barras en el lado izquierdo, en el lado derecho encontramos la sección de nivel donde se observa el nivel actual, los puntos conseguidos y los necesarios para pasar al siguiente nivel. En la parte inferior un recuadro muestra el arma equipada en cada momento y las disponibles para usar.

Figura 25 Interfaz de usuario.

Se añaden dos pantallas de diseño propio, una para la opción de pausa en el juego y otra en el momento de la muerte. Estas aparecen y permiten seleccionar mediante el ratón la opción deseada.

Ventana de pausa Ventana de fin de juego

Figura 26 Sistema de ventanas

Figura 27 Pantalla Principal del juego

El control del juego se dispone con dos periféricos, el teclado para el control de dirección y menús y e ratón que permite observar el entorno, establecer la dirección del movimiento y ejecutar el ataque. Las teclas específicas de cada ejecución aparecen en el menú principal del juego y se detallan en el manual de usuario con detalle.

3.4. Estrategia de marketing

La obra está concebida como parte de un aprendizaje y profundización en el ámbito de los videojuegos dentro de los estudios universitarios de ingeniería informática, por lo que no se establece ningún tipo de comercialización.

Sin embargo, se puede establecer estrategias para dar a conocer el trabajo fuera del ámbito académico de modo que más personas tengan conocimiento y por lo tanto acceso a este. De modo que es posible establecer líneas de divulgación como foros y grupos de debate, establecer un video pitch para introducirlo en plataformas audiovisuales o incluso generar pequeñas partidas en directo, en twitch por ejemplo.

4. Diseño

4.1. Arquitectura general de la aplicación

Para la implementación del todo el sistema y tal como se expresa en anteriores aparatados, se utiliza el motor Unreal Engine 5 de Epic. Este motor representa la tecnología más evolucionada actualmente en diseño de juegos 3D siendo la referencia dentro de los motores generales. Sería posible realizar el juego en el motor Unity, pero se quiere realizar un aprendizaje de las herramientas revolucionarias como Lumen o Nanite que permiten un sistema final más eficiente y optimizado.

Este motor se complementa con el IDE de desarrollo Rider de JetBrains para la programación en lenguaje C++ puesto que es muy ágil y con una disposición similar a los IDEs utilizado durante todo el grado, aunque internamente se debe contar con Visual Studio de Microsoft para su funcionamiento.

Para el servicio en la nube se implementa un sistema sobre Amazon Web Service denominado GameLift, este sistema de servicio en la nube está apoyado por alojamiento en el servicio S3 del mismo proveedor para las compilaciones.

4.2. Entorno de desarrollo

4.2.1 Requisitos Técnicos del entorno

Los requisitos técnicos los podemos encontrar en la página del motor utilizado y del IDE.

Unreal Engine 5
Requisitos recomendados

Sistema Operativo	Windows 10 64-bits versión .1909, revisión 1350 o superior, o versión 2004 y 20H2 versión .789 o superior
Procesador	Intel Quad-Core AMD 2.5GHz
Memoria RAM	8 GB RAM
Tarjeta Gráfica	Compatible con DirectX 11 o 12
Versión RHI	• DirectX11 • DirectX12 • Vulkan: AMD (21.11.3+) • NVIDIA (496.76+)

Lumen

Lumen Global Iluminación y Reflexiones	Software de trazado de rayos: • Tarjetas de vídeo con DirectX 11 compatibles con Shader Model 5 Trazado de rayos de hardware: • Windows 10 compatible con DirectX 12 • Las tarjetas de vídeo deben ser NVIDIA RTX-2000 series y superiores, o AMD RX-6000 series y superiores

Nanite

Geometría virtualizada Nanite	▪ Se admiten todas las versiones más recientes de Windows 10 (más reciente que la versión 1909.1350) y Windows 11 compatible con DirectX 12 Agility SDK. ▪ Windows 10 versión 1909: el número de revisión debe exceder o ser igual a .1350. ▪ Windows 10 versión 1909: el número de revisión debe exceder o ser igual a .1350. ▪ DirectX 12 (con atomics Shader Model 6.6) o Vulkan (VK_KHR_shader_atomic_int64) ▪ Controladores de gráficos más recientes

IDE

Visual Studio 2019

Sistemas operativos admitidos	Visual Studio 2019 se instalará y ejecutará en los siguientes sistemas operativos (se recomienda 64 bits; ARM no se admite): • Windows 11 versión 21H2 o posteriores: Home, Pro, Pro Education, Pro for Workstations, Enterprise y Education • Windows 10 versión 1703 o posteriores: Home, Professional, Education y Enterprise (LTSC y S no se admiten) • Windows Server 2019: Standard y Datacenter • Windows Server 2016: Standard y Datacenter • Windows 8.1 (con la actualización 2919355): Core, Professional y Enterprise • Windows Server 2012 R2 (con la actualización 2919355): Essentials, Standard y Datacenter • Windows 7 SP1 (con las actualizaciones más recientes de Windows): Home Premium, Professional, Enterprise y Ultimate

Visual Studio 2019

Hardware	• Procesador de 1,8 GHz o superior. Se recomiendan cuatro núcleos o superior. • 2 GB de RAM; 8 GB de RAM recomendado (mínimo de 2,5 GB si se ejecuta en una máquina virtual) • Espacio en disco duro: mínimo de 800 MB hasta 210 GB de espacio disponible, en función de las características instaladas; las instalaciones típicas requieren entre 20 y 50 GB de espacio libre. • Velocidad del disco duro: para mejorar el rendimiento, instale Windows y Visual Studio en una unidad de estado sólido (SSD). • Tarjeta de vídeo que admita una resolución de pantalla mínima de 720p (1280 x 720); Visual Studio funcionará mejor con una resolución de WXGA (1366 x 768) o superior.

IDE
JetBrains Rider

Sistemas operativos	Rider se instalará y ejecutará en los siguientes sistemas operativos (solo distribuciones de 64 bits): ▪ Windows 10 y 11 ▪ Servidor Windows 2019 y 2022 ▪ macOS 10.15 ▪ Linux ▪ Debian 9+ ▪ CentOS 7+ ▪ Ubuntu 16.04+ ▪ Fedora 30+ ▪ Otros con GLIBC 2.27+
FrameWork	• .NET Framework 4.7.2 o superior en Windows
Hardware mínimo requerido	CPU: 2 GHz Memoria: 4 GB RAM Disco: 2,5 GB

4.2.2 Herramientas utilizadas

Para realizar todo el trabajo se utilizan diferentes herramientas, aparte de las mencionadas anteriormente.

- **GitHub**: Repositorio con todo el proyecto, tanto código como gráfico. Se adquiere una cuenta ampliada para disponer de espacio suficiente para el proyecto.

- **SourceTree**: Herramienta de sincronización entre el proyecto local y el repositorio de GitHub.

- **Gimp**: Editor gráfico gratuito para la realización de ediciones gráficas para el montaje de interfaces y documentación.

- **OBS**: Grabación de pantalla y audio para la realización de videos.

4.3. Recursos utilizados en ProjectEco

Como se indica durante el trabajo, en la obra se utiliza recursos externos para la confección del arte. Ante la elección de realizar un juego en tres dimensiones, el diseño necesita una gran dedicación por lo que es imposible realizarlo en el tiempo previsto para la finalización y entrega del trabajo de fin de grado. Por lo tanto, se acude a distintos servicios para obtener los modelos y archivos **básicos** necesarios, dejando para el trabajo su complementación audiovisual.

Se obtienen modelos con características básicas, en ningún caso esos modelos o archivos tienen desarrollo o representan un modelo listo para utilizar. Se pretende establecer como hito dentro del trabajo el realizar todo el proceso desde que se obtiene el objeto hasta que finalmente se incorpora al juego, esto es dotándolo de cualquier desarrollo con lenguaje C++ completo o incorporación de mecánicas.

4.3.1. Assets utilizados

Se presenta un listado con los assets utilizados dentro del juego, sean individuales o packs de objetos.
Personajes del juego

Principal y NPC

- **Kachujin**: Modelo del personaje principal, se descarga con varios movimientos básicos. Obtenido en: https://www.mixamo.com/#/?page=1&query=kachujin&type=Character
- **Arissa**: Personaje NPC en la aldea, se descarga con varios movimientos. Ontenido en: https://www.mixamo.com/#/?page=1&query=Arissa&type=Character
- **Brute**: Personaje NPC en la aldea, se descarga con varios movimientos. Obtenido en https://www.mixamo.com/#/?page=1&query=brute&type=Character
- **Erika**: Personaje NPC en la aldea, se descarga con varios movimientos. Obtenido de https://www.mixamo.com/#/?page=1&query=Erika&type=Character
- **Eve**: Personaje NPC en la aldea, se descarga con varios movimientos. Obtenido en https://www.mixamo.com/#/?page=1&query=Eve&type=Character
- **Morak**: Personaje NPC principal en la aldea, se descarga con varios movimientos. Obtenido en https://www.mixamo.com/#/?page=1&query=morak&type=Character

Enemigos

- **Skeletonzombie**: Enemigo a distancia, se descarga con varios movimientos. Ontenido en https://www.mixamo.com/#/?page=1&query=skeletonZombie&type=Character
- **Buff**: Enemigo de melé, el modelo cuenta con las animaciones incorporadas por lo que no es necesarios obtenerlas como el caso de mixamo. Obtenido de un pack de la Epic store gratuito

perteneciente al juego Paragon en https://www.unrealengine.com/marketplace/en-US/product/paragon-minions

- **ASC Teuthisan:** Enemigo final, se obtiene de un proyecto gratuito de la Epic Store, pero sin animaciones por lo que se deben obtener utilizando mixamo y añadírselas al modelo estático. Obtenido en https://www.unrealengine.com/marketplace/en-US/product/asc-teuthisan

Packs de modelos de ambiente

Todos los packs son obtenidos de la Epic Store de forma gratuita, conforman los modelos de rocas, objetos, construcciones y vegetación.

- City of brase props
- Enviorement Pack 2
- Infinity Blade Assets
- Infinity Blade Effects
- Inifnity Blade Grass Lands
- Medieval Dungeon
- NE_US_Plants_Packs
- PN Open World Foliage
- CoulCave
- Water Materials

Imágenes y algún modelo.

- **Artstation**: https://www.artstation.com/
- **Cgtrader**: https://www.cgtrader.com/free-3d-models

Sonidos

Para el sonido, la fuente recurrida son las webs de audio gratuitas, siendo estas las siguientes.

- **Freesound**: https://freesound.org/
- **Zapsplat**: https://www.zapsplat.com/
- **Pixabay**: https://pixabay.com/es/sound-effects/
- **BBC**: https://sound-effects.bbcrewind.co.uk/
- **SoundCreate**: https://sfx.productioncrate.com/
- **Videvo**: https://www.videvo.net/es/

4.3.2. Arquitectura del sistema

El sistema del juego se basa en conseguir unos objetivos definidos en cada nivel. El primero de ellos se busca que el personaje adquiera las habilidades para poder luchar contra los enemigos y que conozca el lore del juego, por lo tanto, debería dialogar con los NPCs del juego. Este aspecto, como suele ser habitual en los juegos, es un aspecto opcional y lo que aporta es información de las causas que provocan la historia, pero se dispone un NPC principal que dará instrucciones al jugador en cada etapa. Dicho NPC, Morak, indica en cada momento cada una de las tres misiones a realizar en el primer nivel y sin la última conversación no es posible pasar al siguiente nivel.

El segundo nivel se debe eliminar a los enemigos hasta el jefe final, este es el principal logro por conseguir en este nivel y sin el cual no se da por finalizado el juego.

El siguiente esquema resume la lógica de todo el juego, desde el inicio hasta la salida de este.

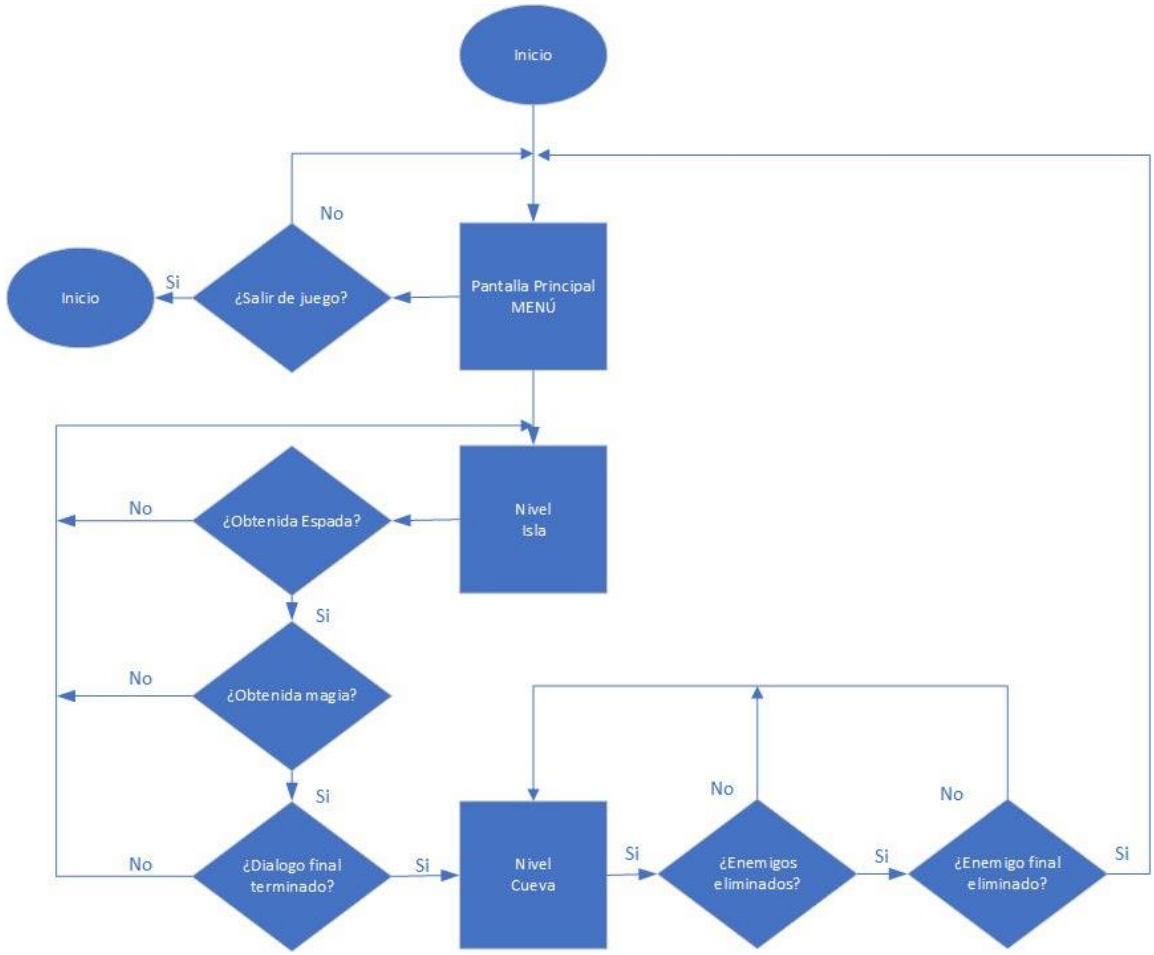


Figura 28 Lógica del juego

4.3.3. Componentes del juego

Dentro de Unreal se utilizan distintos tipos de objetos que controlan o pueden introducirse en el nivel, cada uno de ellos tiene una funcione especifica y pueden unirse a otros objetos para confeccionar elementos más complejos como animaciones.

Cada objeto pertenece a la clase Actor, esta clase es la principal de motor en cuanto a objetos instánciales dentro de un nivel y puede tratarse de cualquier elemento como cámaras, malla estática o la ubicación de un actor.

Todos los niveles incorporan los mismos elementos básicos, sin estos elementos no se obtendría un desarrollo correcto puesto que algunos de ellos manejan información sobre como transcurre la experiencia. En este proyecto se desarrolla cada uno de estos elementos en gran parte por código en lenguaje C++ y luego se visualiza dentro del motor como un elemento gráfico y fácilmente configurable.

A continuación, se muestra una breve descripción de los elementos esenciales utilizados en el juego.

Gameinstance: Controla el juego en nivel alto, sus variables no se pierden entre niveles por lo que es muy importante en juegos de más de un nivel que guardan estadísticas o estados.

GameMode: Este objeto define los elementos en la partida actual, cada nivel llama a este objeto que controla por ejemplo que personaje juega.

Actor: Es la clase principal de cualquier objeto que pueda situarse en un nivel, la clase principal se denomina AActor.

Pawn: Es la representación física de los actores controlados por los jugadores o la IA, NPCs por ejemplo, y controla cualquier interacción con el mundo.

PlayerController: Se encarga de llevar un control todos los jugadores que se encuentran en la partida, en nuestro caso al ser de un solo jugador encontramos que es el jugador 0.

Widget: Se utilizan generalmente para la interfaz y cualquier otro sistema visual que interactúe con el usuario como ventanas, diálogos, el propio HUD del juego es uno de ellos o incluso el menú principal del juego.

4.3.4. Player

En el juego tenemos el personaje principal que es el player, este objeto perteneciente a la clase Actor y se compone por varios elementos. Gracias a este elemento se puede desarrollar el juego puesto que es el que controla gran parte de las mecánicas por ser un juego de un solo jugador.

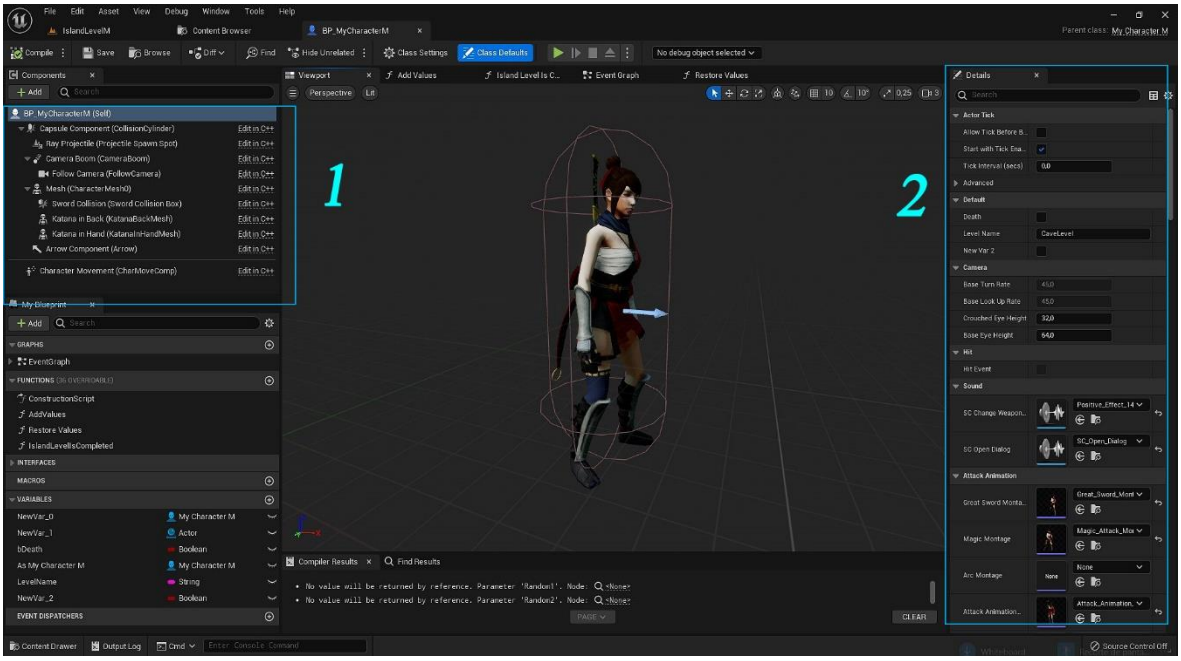


Figura 29 Player con los elementos dentro de Unreal

Prácticamente todo el objeto está desarrollado en lenguaje C++ ocupando el desarrollo principal unas 800 líneas de código más otras funciones y declaraciones, es el objeto más complejo con diferencia del juego.

A parte del código implementado, al abrir el Blueprint nos encontramos con muchos elementos a configurar, los más elementales son los siguientes.

Elementos en sección 1

Los elementos que forman el aspecto físico y de colisiones del player, encontramos:

- **Capsule**: Es el área donde el personaje recibe un estímulo al ser atravesado por otra malla, por ejemplo, recibir daño o establecer un dialogo.
- **Camera Boom**: Es la forma en que visualizaremos al jugador, la perspectiva.
- **Follow Camera**: Es la cámara por donde se visualiza la escena.
- **Mesh**: Contine el modelo 3D.
- **Sword Collision**: Es una capsula para el objeto espada añadido al modelo en su esqueleto.
- **Katana in Back**: Es la espada en la espalda, se visualiza solo cuando no se utiliza la espada.
- **Katana in Hand**: Es la espada en la mano y se visualiza solo cuando se utiliza.

Elementos en sección 2

La mayoría de los elementos en esta ubicación son variables definidas en el código y se utilizan para el desarrollo del juego, entre otras tenemos:

- **Animation**: Es la animación que se utiliza para este objeto, se utiliza un tipo llamado aminstance, y contiene la lógica de movimiento.
- **Attack Animation**: Contiene la secuencia de animaciones cuando se realiza un ataque, el elemento es de tipo AnimMontage.
- **Stats**:Tenemos los estados, stats, que contienen los niveles de vida, magia, daño causado, puntos del jugador, etc.
- **My Character M**: Contiene los widgets utilizados por el jugador como el de recibir daño o valores de configuración.

La cantidad de variables por cada elemento del objeto nos muy grandes por lo que se describen las básicas puesto que no es objeto del trabajo exponer detalladamente cada elemento del personaje.

4.3.5. Mapas del juego

En el juego disponemos de tres niveles, el primero de ellos se utiliza para mostrar el menú principal, posteriormente disponemos de los niveles propios del juego.

El Menú principal se compone por tan solo un widget y el resto del mundo está vacío.

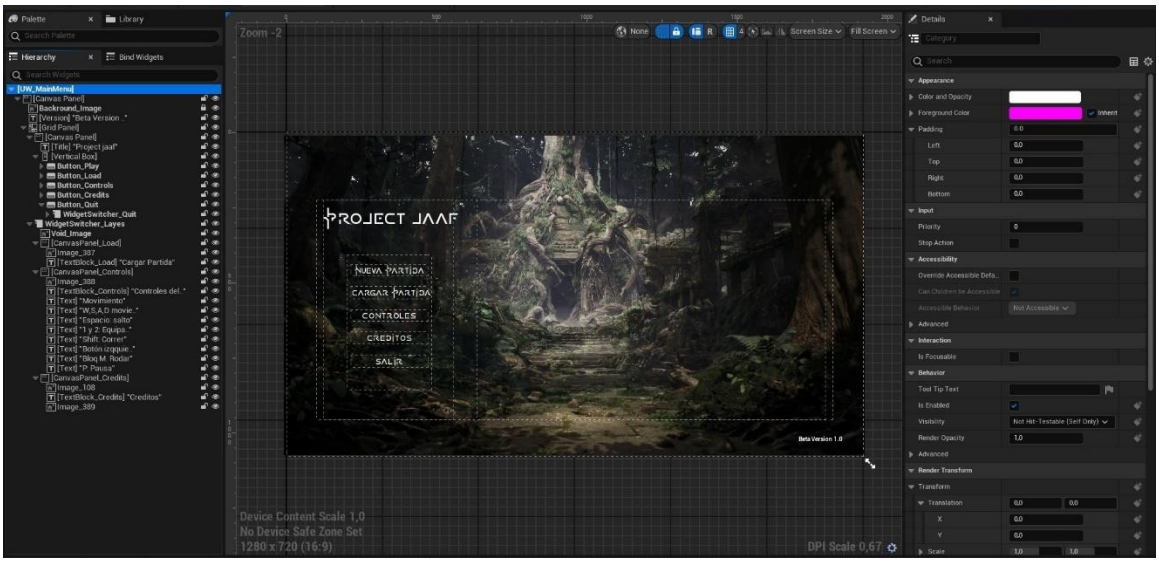


Figura 30 Diseño menú principal

En cuanto a su programación, se utiliza Blueprints para realizar el desarrollo. A continuación, se muestra una de las implementaciones que permite visualizar las distintas pantallas del menú.

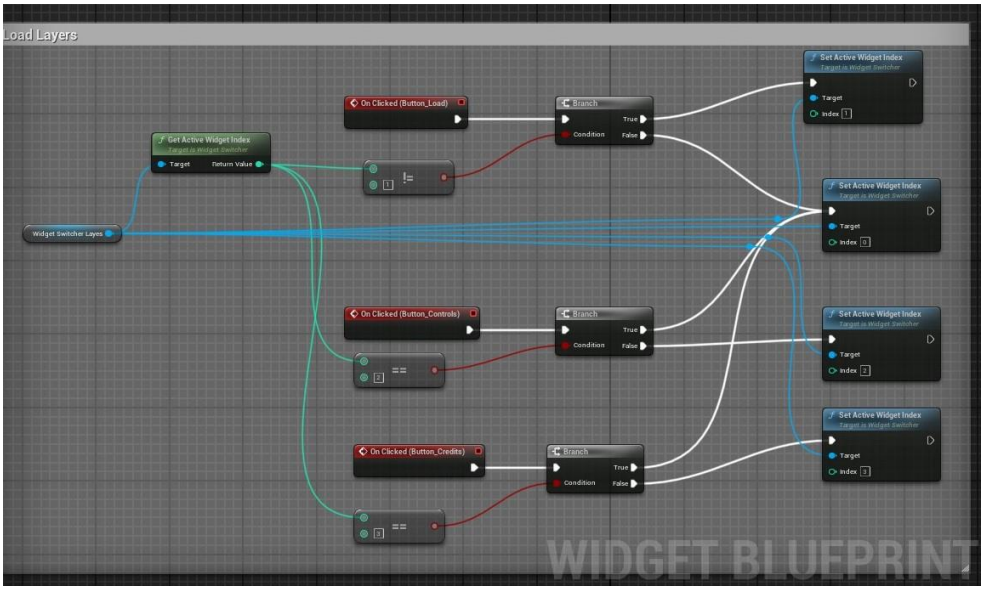


Figura 31 Función para mostrar las pantallas del menú

El nivel de la isla se compone por el terreno elaborado con las herramientas de esculpir que tiene Unreal, sobre el cual se utiliza una textura creada especialmente que contiene otras texturas. Con este método se consigue que al pintar con distintos materiales la transición de uno a otro se hace de forma suave y natural.

La vegetación se crea con un sistema procedural donde se establecen los elementos que intervienen en cada momento y Unreal crea una distribución utilizando esos elementos, de modo que podemos crear sensación de algo más natural y no repetitivo.

El resto de las estructuras son assets situadas en cada emplazamiento según la función que necesitemos.

Para este nivel obtenemos una serie de implementaciones sencillas que controla el paso de nivel y las variables puesto que deben ser guardadas dentro del GameInstance o se perderán. También controla algunas interacciones elementales del nivel como cajas de dialogo que aparecen en determinados puntos.

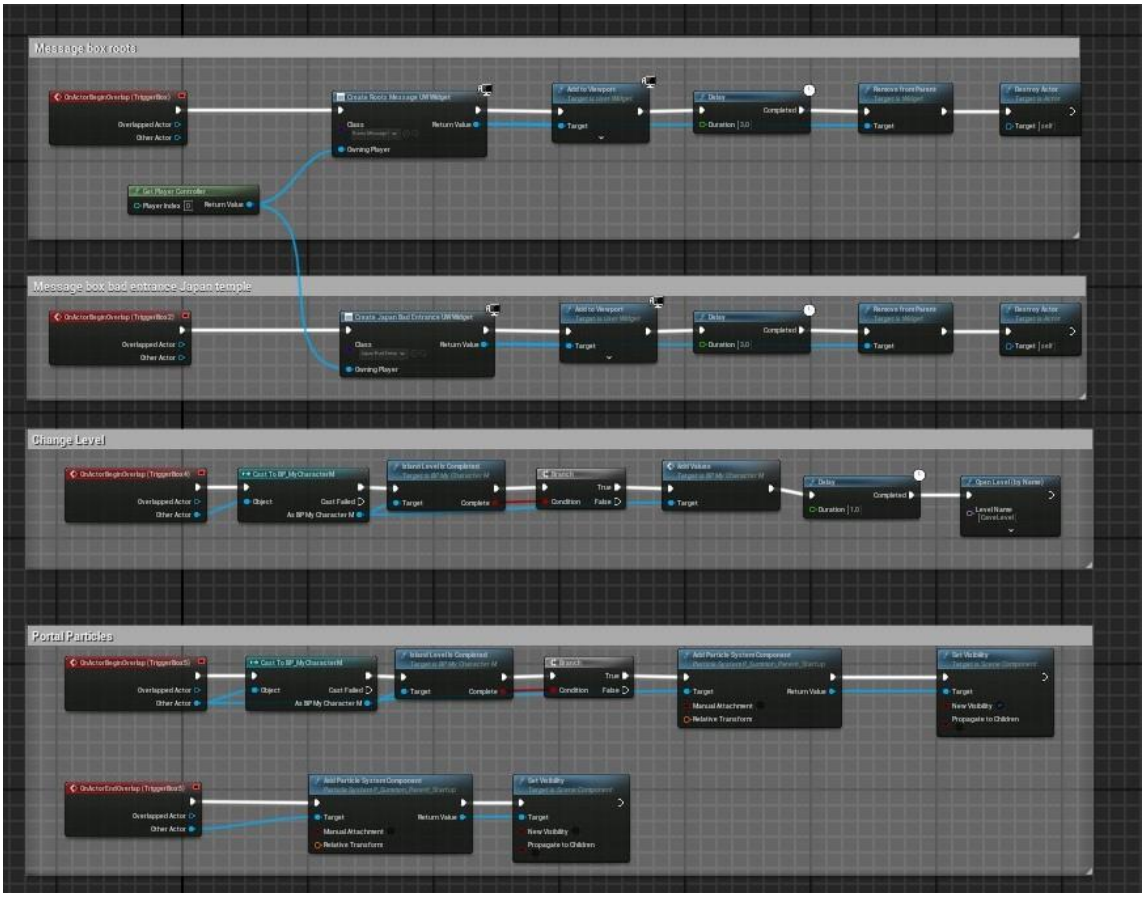


Figura 32 Parte de la programación visual del nivel de la Isla

El ultimo nivel es el de la cueva, este se basa en el mismo sistema que la isla en cuanto a su diseño e implementación. Se utiliza las herramientas de esculpido mara los desniveles y posteriormente se añaden assets. En este caso se utiliza gran cantidad de roca, aunque realmente solo se incluyen dos o tres modelos diferentes que al unirlos entre ellos da la sensación de ser todo roca.

Se incluyen cantidades más elevadas de luces para iluminar la cueva y se elimina el efecto de adaptación a la luz que incluye Unreal para simular la entrada o salida de espacios con cantidad de luz diferente.

4.3.6. Enemigos

El sistema de enemigos se basa en un objeto llamado Enemy, este objeto contiene las características básicas y comunes de los enemigos. Está clase se utiliza como base para el resto de los enemigos, de modo que reutiliza el código.

En esta clase padre, se desarrolla la parte básica que deberá tener el enemigo, de este modo encontramos las funciones de muerte o de recibir daño, puntuación y animaciones.

Para el control de vida se utiliza otro componente llamado HealthComponent, este componente es llamado por medio de las funciones propias de la clase Enemy y modifican la vida y el daño.

Podemos observar en la clase de definición de enemy las variables de puntuación, la declaración del componente de vida “Health” y los métodos OnDead y OnDamage entre otros.

```
#pragma once

#include ...



UCLASS()
3 derived blueprint classes
class PROJECTTFG_U_V1_API AEnemy : public ACharacter
{
    GENERATED_BODY()

public:
    // Sets default values for this character's properties
    AEnemy();
protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

protected:
    UPROPERTY(EditDefaultsOnly, Category=Components)
    class UHealthComponent*  Health {nullptr};  Changed in 14 blueprints
    void DisableMovement();
    void EnableMovement();
    void UpdateScore(float points);

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

    // Called to bind functionality to input
    virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;

private:
    UFUNCTION()
    void OnDead(AActor *DamagedActor, float Damage,const class UDamageType* DamageType, class AController* InstigateBy, AActor* DamageCauser);

    UFUNCTION()
    void OnDamage(float ActualLfe);

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Attack_Animation, meta=(AllowPrivateAccess="true"))
    UAnimMontage* DamageAnimation {nullptr};  Changed in 3 blueprints

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Attack_Animation, meta=(AllowPrivateAccess="true"))
    float BaseApplyScoreMagic {10.f};  Unchanged in assets

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Attack_Animation, meta=(AllowPrivateAccess="true"))
    float BaseApplyScoreMelee {10.f};  Unchanged in assets

    FTimerHandle TimerMovement;

public:
    int32 PlayMyAnimation(UAnimMontage* MontageToPlay, float InPlayRate, FName SlotName, float InTimeToStartMontageAt, bool Delay);

};
```

Figura 33 Clase Enemy.h

4.3.7. IA Enemigos

Los enemigos se basan en un sistema de IA del propio Unreal, este sistema está configurado para que el enemigo pueda ver y oír, de modo que el jugador deberá realizar alguna acción de ese tipo para ser detectado.

Para implementar esta funcionalidad se recurre a la programación de la clase EnemyAIController. Esta clase contiene los radios de acción de los sentidos, así como los tipos de sentidos activos.

Para las acciones que se realizan se desarrolla otras clases que controlan varias variables, siendo estas:

- **DistanceService**: calcula la distancia a la que se encuentra un objeto designado.
- **BTTAttackCreature** y **AttackMelee**: Realizan el ataque según si está en rango de ataque.
- **BTT_StopTimer**: establece un tiempo para regresar a un estado de reposo.

Podemos ver las clases incluidas para el sistema de IA.

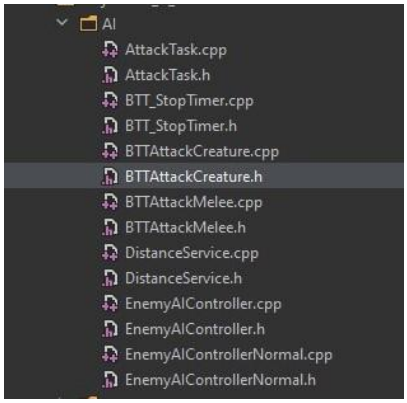


Figura 34. Listado de clases para IA

Esta implementación se complementa con un Behavior Tree y un Blackboard. Dentro del Behavior tree disponemos de un árbol donde ir implementando la IA, el sistema según pueda o no realizar acciones se decantará hacia una rama u otra. Dentro del Blackboard situaremos las variables y objetos que necesitamos para interactuar.

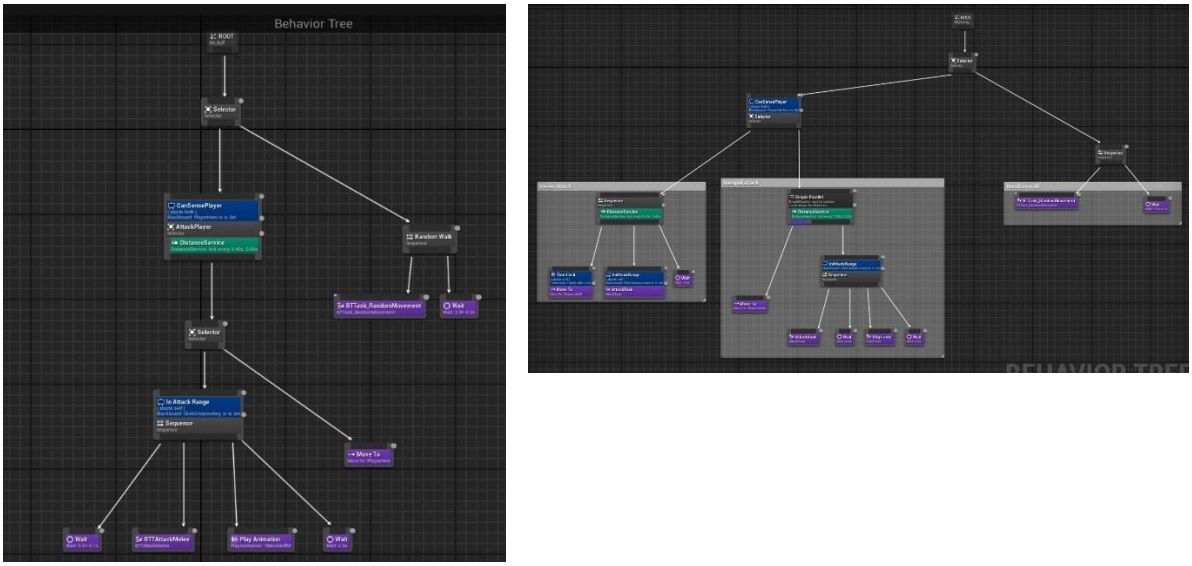


Figura 35. Behavior Tree

Por lo tanto, al recibir un estímulo, la IA, comprueba la distancia hasta el jugador para ver si es posible ejecutar un ataque, en caso de ser falso se acercará al jugador y volverá a comprobar la distancia hasta que pueda realizar un ataque. El ataque vendrá determinado por la distancia, si se encuentra a una distancia de melé el ataque será uno mientras que si es una distancia mayor el ataque será otro.

Pasado un tiempo si no recibe estimulo sensitivo el sistema regresa a su estado normal de “vigilancia”.

4.3.8. Animación

Para realizar las animaciones en Unreal se utilizan varios elementos para finalmente obtener un objeto animado. Primero disponemos de un Blueprint que es el objeto en sí, donde internamente se van incorporando lo que necesitamos, por ejemplo, el caso del player visto más arriba es la imagen de un Blueprint.

Para animar un objeto se debe crear un Animinstance que controla todo lo referente a la animación y donde podemos incluir lógica para el comportamiento del movimiento. Para establecer las velocidades y el movimiento raíz utilizamos una clase llamada BlendSpace, donde se establece las variables y los valores.

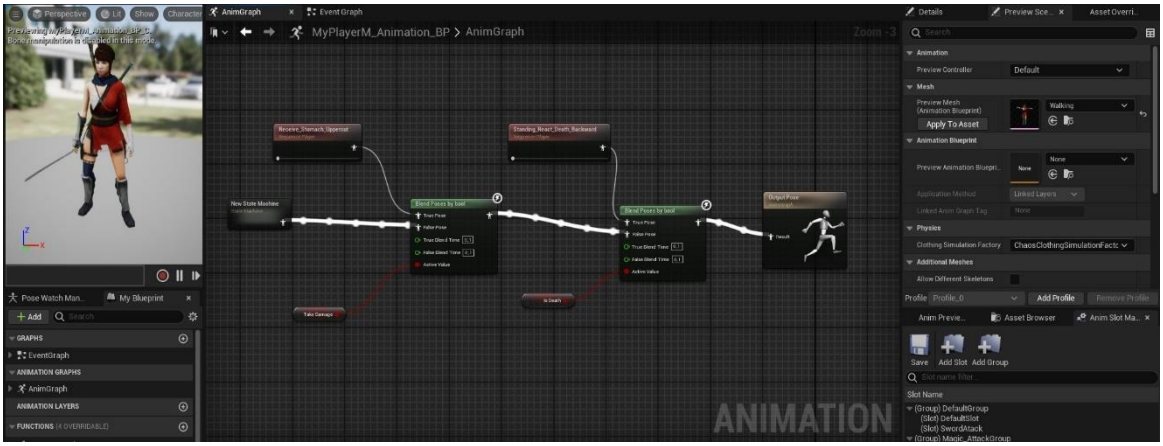


Figura 36. AnimInstance

Dentro del Animinstance se crean statemachines que enlazan variables y animaciones para obtener la acción que se necesita en cada momento. Para la característica de andar o correr, se evalúa la variable velocidad y según el valor la animación se irá extrapolando hacía la animación de caminar o correr.

Para el control de estados de cada modo se utilizan variables booleanas, así si en un momento está caminando y realiza un ataque, al cambiar el estado de la variable la animación resultante será la del ataque.

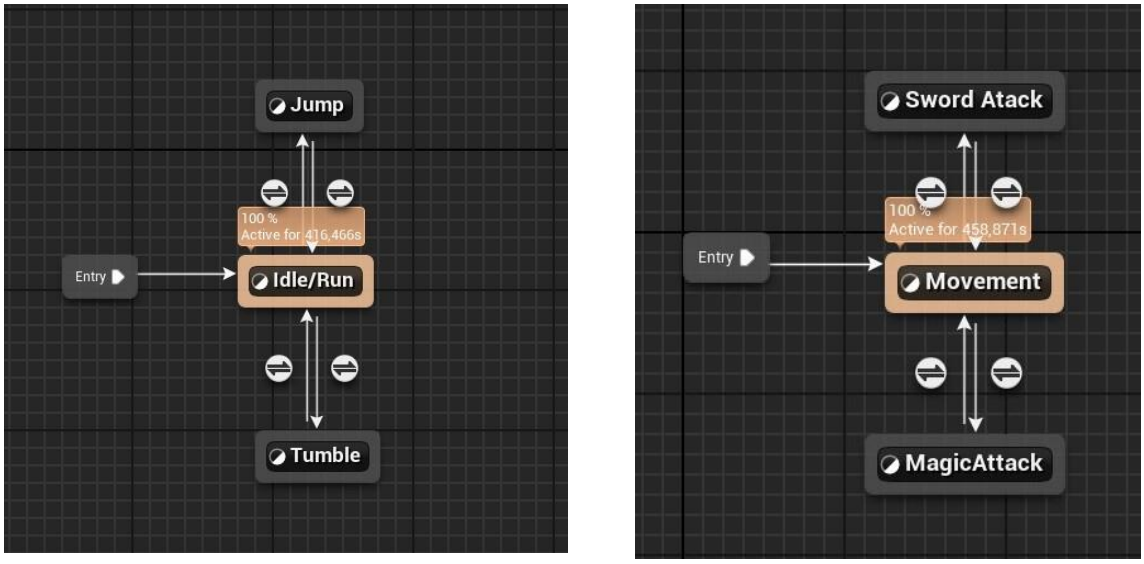


Figura 37. Statemachines

Para las animaciones de ataques, se utiliza un sistema llamado AnimMontage, en este caso se implementa para el jugador y para los enemigos. Este sistema consta de varias líneas de tiempo donde podemos incorporar todo tipo de efectos o llamadas a funciones.

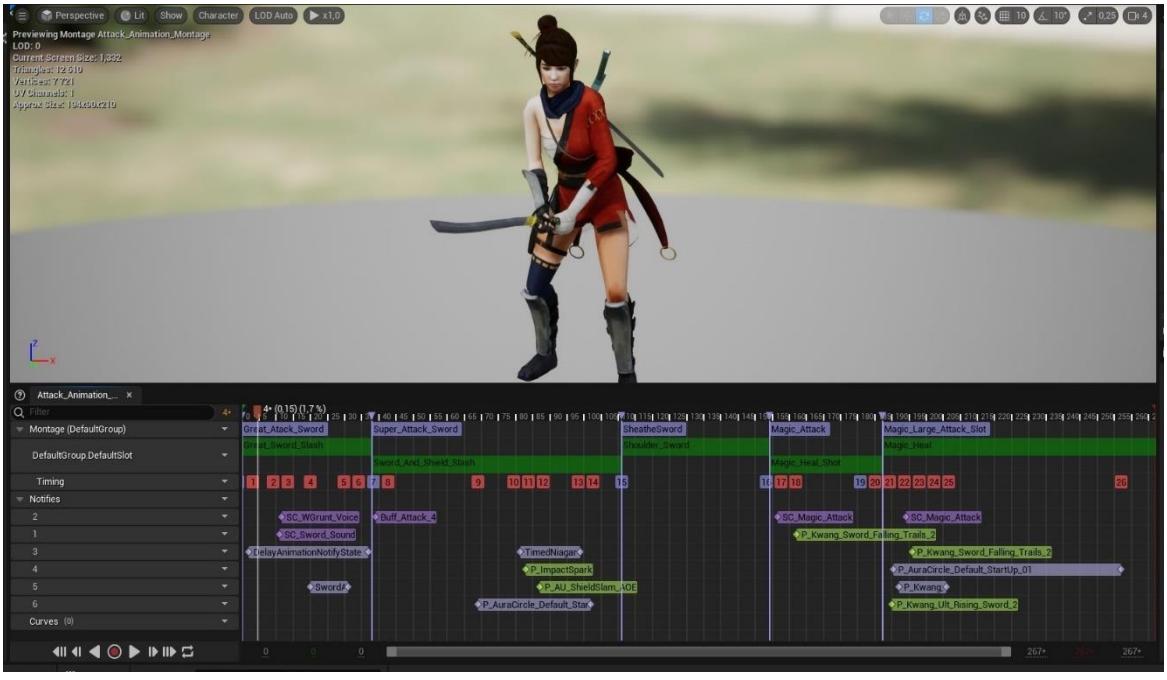


Figura 38. AnimMontages

4.3.9. Sistema de audio

Para el sistema de audio se utilizan se utilizan varias formas de implementación. Por un lado, se utilizan archivos sin ningún tipo de modificación para sistemas de ventanas o botones y por otro lado para animaciones. Para este último tipo se utiliza un objeto llamado SoundCue, este sistema permite realizar acciones sobre el fichero de audio.

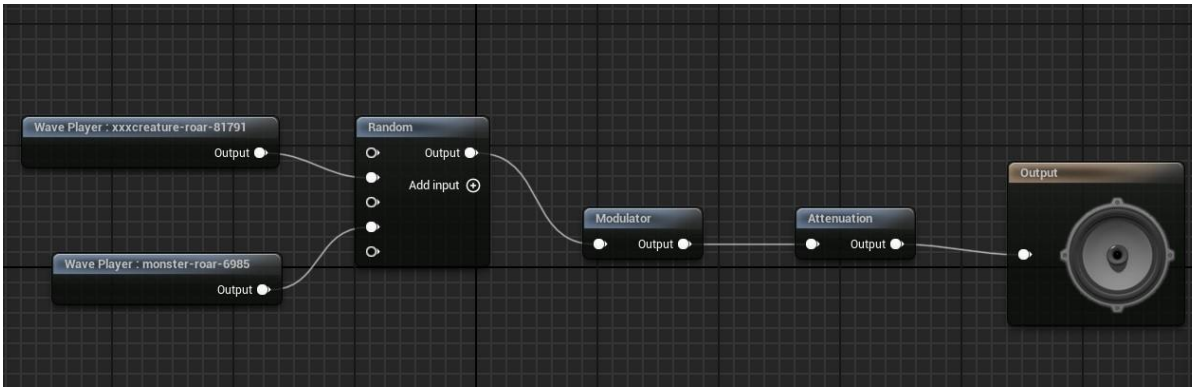


Figura 39. Sistema de audio SoundCue

Con este sistema es posible la incorporación de efectos como variar velocidades o efectos de distancia, de modo que con pocos audios se puede crear la ilusión de que el sonido siempre es diferente.

5. Implementación

En la realización del juego se opta por un desarrollo base de programación y sin recurrir a la utilización de assets que contengan implementado su propio código. De ese modo se adquiere el control completo del desarrollo promoviendo una utilización más optima de los recursos del proyecto y el sistema de modo que no se incorpora nada que no tenga una funcionalidad para el juego.

Para el desarrollo del juego se parte de los dos niveles mínimos indicados para que transcurra la aventura, a estos se incorporan objetos provistos de lógica que reaccionan a las interacciones del personaje y como no puede ser otro modo, los enemigos.

La implementación se realiza en gran parte en lenguaje C++, pero dentro del motor gráfico Unreal se le asigna un fichero adjunto para un fácil manejo de cada objeto. Estos ficheros son una forma visual de trabajar y evita la necesidad de modificar el código cada vez que se desea modificar un valor de modo que no es necesario intercalar continuamente entre Unreal y el IDE.

5.1. Implementación de Personajes

En la obra podemos encontrar distintos personajes que nos permiten interactuar con ellos de forma más o menos inmersiva. Encontramos desde unos simples personajes no jugables (NPG) que nos proporcionan información sobre el lore del juego y como guía para el avance de las misiones. Por otro lado, encontramos a los enemigos que tienen como único acometido en la obra que el personaje no concluya la aventura y pierda la partida.

A continuación, se indican las clases relacionadas con personajes más utilizadas dentro del juego, se expone entre paréntesis el nombre dentro de Unreal que está asociado a cada clase.

- **MyCharactherM(BP_MyCharacterM):** Tipo BluePrintClass para el personaje principal, Den, se diseña en base a la clase Actor de Unreal, esta clase proporciona cualidades necesarias para poder interactuar con el resto del mundo, realmente esta clase es la base para muchos de los objetos que se insertan en la escena. Dentro de la clase encontramos todo lo relacionado con el movimiento, vida, puntuación y niveles del personaje. Esta clase es la más extensa, con casi 800 líneas de código en un único fichero de C++ y muchas otras funciones en BluePrint en Unreal.

- **MyPlayerMAnimInstance(MyPlayerM_Animation_BP):** AnimInstance que contine la lógica de animación como el movimiento o salto, de modo que se relaciona con la clase del personaje al que está asignado y genera los movimientos que le pide este. Dentro de esta clase situamos,

dentro del modo visual, el esqueleto del personaje, sus maquinas de estado que indican la animación a realizar y los esqueletos para las físicas.

- **Attack_Animation_Montage**: De tipo AnimMontage, este fichero de Unreal incluye las animaciones de los diferentes ataques del personaje. Es posible incluir varias animaciones diferentes e incluirles elementos multimedia, así como disparadores, notify, para realizar acciones por código.

- **Enemy**: En esta clase padre de C++, se implementa la base para confeccionar a los enemigos de modo que al utilizar la herencia se parte de una estructura básica y común. Cuando se hereda de una de estas clases tan solo se debe completar los métodos propios de cada enemigo. Esta Clase incluye otra a modo de componente, se trata de la clase HealthComponent que se describe a continuación.

- **HealthComponent**: Clase en C++ que otorga la cualidad de vida al objeto al que es asignado. En este caso provee de puntos de vida, controla la vida o cuando se aplica daño. Este componente, se diseña de forma que se abstraiga del objeto al que se adjunta resultando un acoplamiento bajo. Con esta implementación es posible adjuntarlo a cualquier elemento actor y hacer que muera o se destruya, como puede ser personas, animales, piedras, barriles, etc.

- **EnemyAIController(BPAIEnemy):** Contiene Las especificaciones de la IA para los enemigos, se establece que estímulos tienen y como funcionan. En este caso incorpora la percepción por vista y por sonido.

- **BehaviorTree y Blackboard:** Clases en Unreal que organizan todas las lógicas de la IA, en ellas se aglutinan las clases realizadas en C++ para controlar a los enemigos, comprobación de datos o disparadores de acciones.

- **Personajes No Jugables**: Los NPC, al igual que cualquier actor del juego, parten de un fichero tipo BlueprintClass con los datos y acciones que pueden realizar.

5.2. Diseño Niveles

Dentro del juego encontramos tres niveles, dos de ellos están dispuestos para realizar el juego mientras que otro de ellos contiene el menú principal del juego únicamente. El desarrollo del juego

transcurre en una isla y en unas cuevas, por lo tanto, son dos tipos de terrenos totalmente diferentes y que implican un diseño único en cada caso.

La realización de cada nivel empieza con establecer un tamaño de cuadricula, esto nos dice lo grande que es nuestro nivel.

Con las herramientas de modelado se dan las formas de montañas o desniveles para posteriormente añadir elementos decorativos de distintos tipos.

- **Nivel Isla**: Se utiliza un sistema de creación de agua para realizar el proceso inicial sobre a rejilla de crear la isla. Se opta por crear cuatro áreas bien diferenciadas para el juego a modo de mini mundo abierto, siendo estas zonas la aldea, el bosque, la montaña y las ruinas. Se incluyen rocas como zonas de montaña y se crea los objetos decorativos en cada zona. La parte de vegetación y árboles se reserva para el final del diseño puesto que consumen muchos recursos.
  La inclusión de vegetación se realiza con un sistema procedural que permite generar la ambientación de modo más natural. Este sistema consiste en ir creando pinceles con distintos tipos de elementos, configurar cada elemento para establecer su aparición, valores de terreno o distancia de dibujado entre otros. En este nivel, la vegetación, se implementó en muchas ocasiones debido a la caída de frames que provocaba una elevada concentración de este en el terreno. Para la base del suelo se crea un material que contiene varias texturas internas, con este método es posible mezclarlas al pintar zonas y que se mezclen de forma correcta sin fallos visuales.

- **Nivel Cueva:** Partimos, al igual que en la isla, confeccionando las partes altas y bajas, pero en este caso se opta por cerrar todo el nivel con rocas para dar la sensación de estar en el interior de la montaña. Se busca la entrada de luz exterior para realizar efectos de iluminación y por lo tanto se intenta situar estas en zonas seleccionadas. Una vez preparada la base, se añade los elementos decorativos con ambientación de ruinas antiguas.

  Este nivel genera la obligación de insertar puntos de luz en distintas zonas, esto es debido a que al ser un lugar cerrado la iluminación exterior entra en poca cantidad y no permite ver en su interior.

En los niveles se busca romper la línea visual entre zonas, ya sea por introducir elementos que no permitan la visión directa o por la forma del terreno. Este diseño busca dos objetivos principales, uno sería no mostrar elementos que sean parte de otras misiones y que puedan dar pistas para la resolución. Por otro lado, se busca poder optimizar las zonas permitiendo una distancia de dibujado menor, este punto se desarrolla en el punto 5.6.

5.3. Volúmenes

Los niveles incorporan algunos objetos llamados “volumes”, estos objetos añaden funcionalidades al escenario y existen gran multitud, en este caso, los más utilizados son:

- **BlockingVolume**: Elemento que no permite el paso del personaje, similar a poner una pared invisible.
- **PainCousingVolume**: Volumen que al atravesarlo causa daño al personaje.
- **PostProcessVolume**: Permite configurar el aspecto visual de una zona o de todo el nivel.
- **NavMeshBoundsVolume**: Volumen que permite moverse a los personajes controlados po una IA.

5.3.1. Generación de Eventos

Un elemento muy importante dentro de los niveles son los llamados Triggers, estos elementos permiten que se inicien métodos al interactuar con ellos. En los niveles disponemos triggers para lanzar mensajes, para iniciar el método que crea la lucha en el altar o incluso para cambiar de nivel se utiliza un trigger. En el nivel de la isla se utilizan un total de 12 triggers y en la cueva 11.

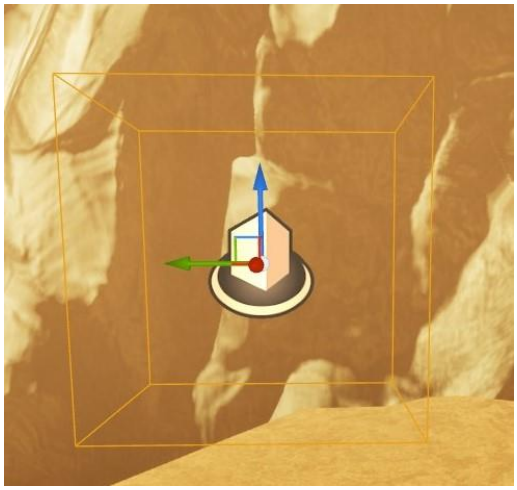


Figura 40. Trigger de colisión para lanzar eventos

5.4. Mecánicas

Las mecánicas permiten interactuar con el entorno, sin ellas sería difícil poder abordar la elaboración de un juego. Para Project Eco se disponen de unas mecánicas básicas de un juego de aventuras para poder afrontar la obra y terminarla.

5.4.1. Diálogos

Disponemos de NPCs para poder obtener información, este método de dialogo se implementa por medio de las mallas de colisión, componentes incluidos en los objetos que detectan cuando algo los toca o atraviesan. Por medio de estas mallas, podemos detectar cuando nos encontramos ante un personaje y nos permite empezar un método que muestre la caja de dialogo. Utilizando una variable de tipo string, se va extrayendo el texto almacenado y se presenta en pantalla hasta terminar la conversación.

Para que la interacción se más realista, se aplica variables de control para que los textos importantes no se repitan una vez expuestos, pero esto ocasionaría que el personaje quedase inactivo durante el resto del juego por lo que se incluyen una serie de frases que se reproducirán de forma aleatoria para obtener respuestas, en cualquier caso.

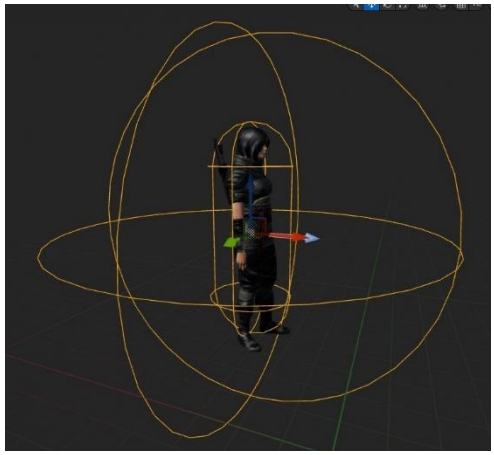


Figura 41. Malla de colisión (interior) y malla Trigger de dialogo (exterior)

5.4.2. Luchas

Las luchas se basan en el mismo sistema que los diálogos, el personaje detecta si algo atraviesa o golpea su malla, este hecho lanza toda una serie de métodos para controlar el daño recibido, controlar si la vida se termina ocasionando la muerte, los puntos de recompensa con la posible subida de nivel. Por otro lado, también se debe actualizar el HUD para que refleje lo datos actualizados en pantalla, todo esto se realiza en su mayoría por el código contenido en la clase del personaje principal, MyCharactherM, y el componente de vida HealthComponent.

Cuando un objeto golpea a otro, se llama a una función que transmite del atacante al receptor los datos de su ataque, el receptor debe actualizar sus datos y retorna información con la puntuación.

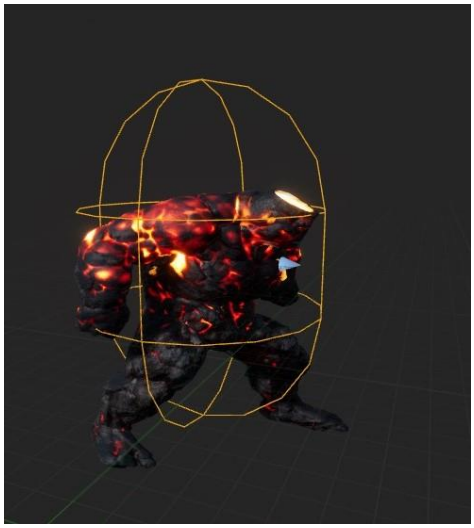


Figura 42. Malla de colisión de enemigo

5.4.3. Guardado

Durante el avance de la aventura puede ser que el jugador pierda la vida o que se vea obligado a detener la partida con la posible pérdida de datos y la obligación de volver a empezar. Para estos casos se incluyen puntos de guardado en los niveles que permiten que, en caso de ser necesario, guardar la partida con tan solo situarse cerca de uno de estos puntos.

Cada punto está identificado con un tótem de piedra con flores y velas, cuando el jugador se sitúa cerca de este punto se escuchará un sonido y aparecerán unas partículas en la estatua, de este modo es posible estar seguro de que la partida está guardada.

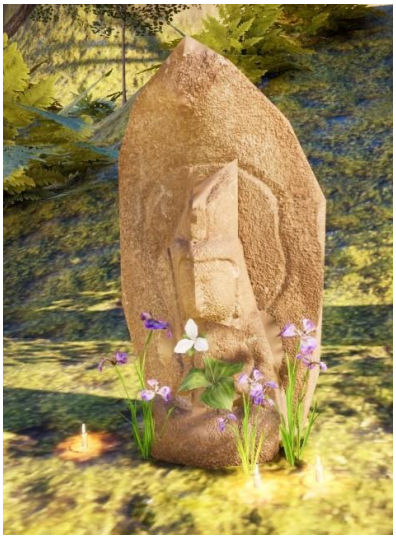


Figura 43. Figura para el guardado de partida

El modo de implementar esta función es con la utilización de un trigger que activa el método de inicio de guardado. El método reúne los datos a guardar en disco y los introduce en una estructura de tipo “SaveGame”, esta estructura no es más que un BluePrint con todos los datos, para posteriormente crear un archivo en disco llamado “slot” con el nombre que se le indica.

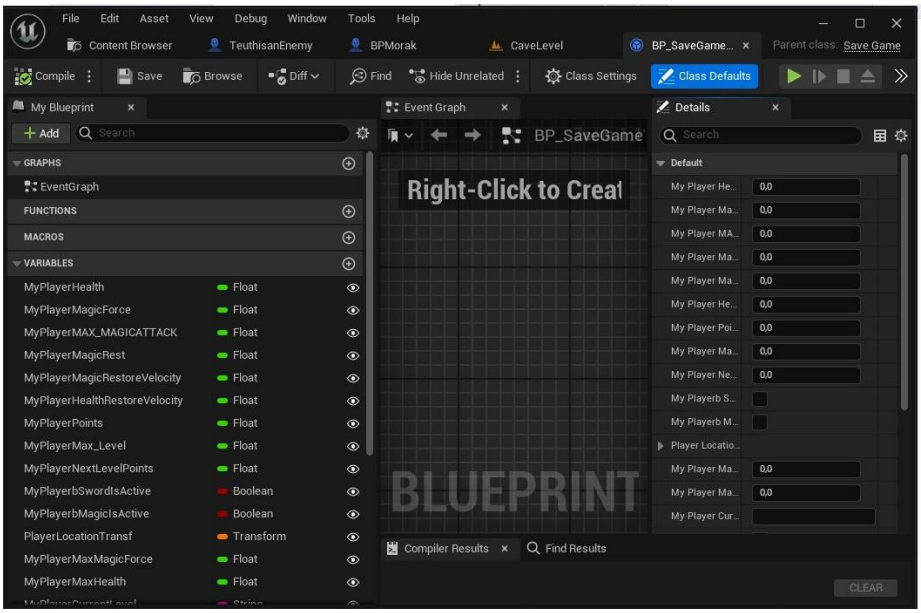


Figura 44. Fichero SaveGame del juego con los datos que contiene

Para cargar la partida, se realiza una carga nueva de nivel en donde, antes de iniciar la partida, se abre el slot y se extraen los datos hacía el fichero persistente gameinstance. Una vez se dispone de los datos dentro del sistema, se actualiza los valores del jugador y se carga todo el juego partiendo de los valores actuales.

5.5. Optimización

Una de las tareas más importantes a la hora de desarrollar un juego es la experiencia del jugador, para ello se debe conseguir un desarrollo agradable de la aventura sin que aparezcan ralentizaciones, por falta de frames por segundo, popping o cualquier otro elemento que distraiga al jugador. Existen muchos métodos de optimización que persiguen estos objetivos, una tasa alta de frames y que no exista carga lenta de objetos en pantalla cuando existen una cantidad elevada de estos en pantalla.

Debido al tiempo disponible para llevar a termino el TFG, se opta por unas optimizaciones que consuman un tiempo relativamente corto de implementación y deja fuera de este proyecto los de mayor impacto. Los métodos utilizados en este caso son los siguientes:

- **Distancia de dibujado**: Este método trata de aligerar los elementos en pantalla, para ello se indican desde el jugador las distancias máximas hasta donde el objeto estará presente en el nivel, a partir de esa distancia Unreal no dibujará el objeto. Este método provoca dos inconvenientes, el primero de ellos es que si la distancia es corta tendremos popping puesto que el objeto aparecerá instantáneamente delante del jugador, la otra es referente al diseño

de niveles ya que pueden aparecer zonas desiertas en la lejanía. Cuando se realiza el diseño de nivel, se debe prestar atención del alcance de la vista del jugador, la complejidad de este simple hecho puede hacer variar mucho el nivel, sobre todo para los niveles en cotas altas que otorgan vistas amplias del terreno.

Dentro de la isla resulta complicado aplicar el método por algunas vistas con un gran campo de visión, pero se incorporan dentro de prácticamente cualquier objeto pequeño y sobre todo a la vegetación. Dentro del nivel de la cueva es mucho más sencillo por tratarse de un sistema muy lineal tipo pasillo, se disponen de zonas que no permiten ver la siguiente sección de juego ni la dejada atrás por lo que se imponen valores más restrictivos en distancia de dibujado.

- **Merge Actors (fusionado):** Unreal gestiona de forma más optima el dibujo de un objeto que muchos que estén juntos. Para poder realizar esta tarea se utiliza una herramienta para convertir a un grupo de objetos en un solo, Merged Actors. Por lo tanto, se destina un tiempo para repasar todos los elementos de los niveles y fusionar gran número de ellos como consecuencia se obtiene que cualquier grupo de elementos dentro de la isla o de la cueva está fusionado. Este método es utilizado en gran medida en el nivel de la cueva, puesto que gran parte del escenario está confeccionado por rocas sueltas, de modo que se fusionan las paredes en grupos, el techo en secciones y los elementos arquitectónicos.

- **Nanite**: Una de las nuevas herramientas que se incluyen en la versión 5 de Unreal es el sistema de Nanite, este sistema permite utilizar muchos objetos sin que el juego se resienta. Dentro de Project Eco, se utiliza esta nueva tecnología para el sistema de rocas y elementos arquitectónicos de tamaño medio o grande.

6. Guía del Usuario

6.1. Requisitos de hardware

Project Eco está diseñado para funcionar en un PC estándar con una tarjeta grafica dedicada. Debido a la carga gráfica de algunos objetos en el juego, se desaconseja utilizar un PC que no disponga de una tarjeta NVIDIA o AMD dedicada.

Requisitos mínimos:

- Sistema Operativo: Windows 10 o superior
- CPU: AMD o Intel gama media
- Memoria RAM: 8GB
- DirectX: versión 11 o 12 (aconsejada)
- Espacio en disco: 3GB
- Teclado y ratón o gamepad

6.2. Instalación y ejecución

El acceso al juego se encuentra de forma totalmente libre en el repositorio de HitLab. Tras la descarga del repositorio, para iniciar el Project Eco tan solo se debe acceder a la carpeta y ejecutar ProjectTFG_U_v1.exe.

6.3. Funcionamiento

6.3.1. Pantalla Principal

- **Nueva Partida**

  Tras arrancar el juego, aparece la pantalla de inicio. Si es la primera vez que se juega o se quiere iniciar una partida desde el inicio, se debe seleccionar nuevo juego.

- **Seguir Partida**

  Una vez en marcha el juego, seleccionar la segunda opción del menú, continuar partida, con lo que el juego inicia en la última localización guardada.

- **Opciones**

  Disponemos de un menú donde configurar las opciones gráficas del juego.

- **Controles**

  Muestra los controles del juego, tanto para teclado como para gamepad

- **Créditos**

  Una breve referencia del proyecto

- **Salir**

  Salida del juego y cierre de la aplicación

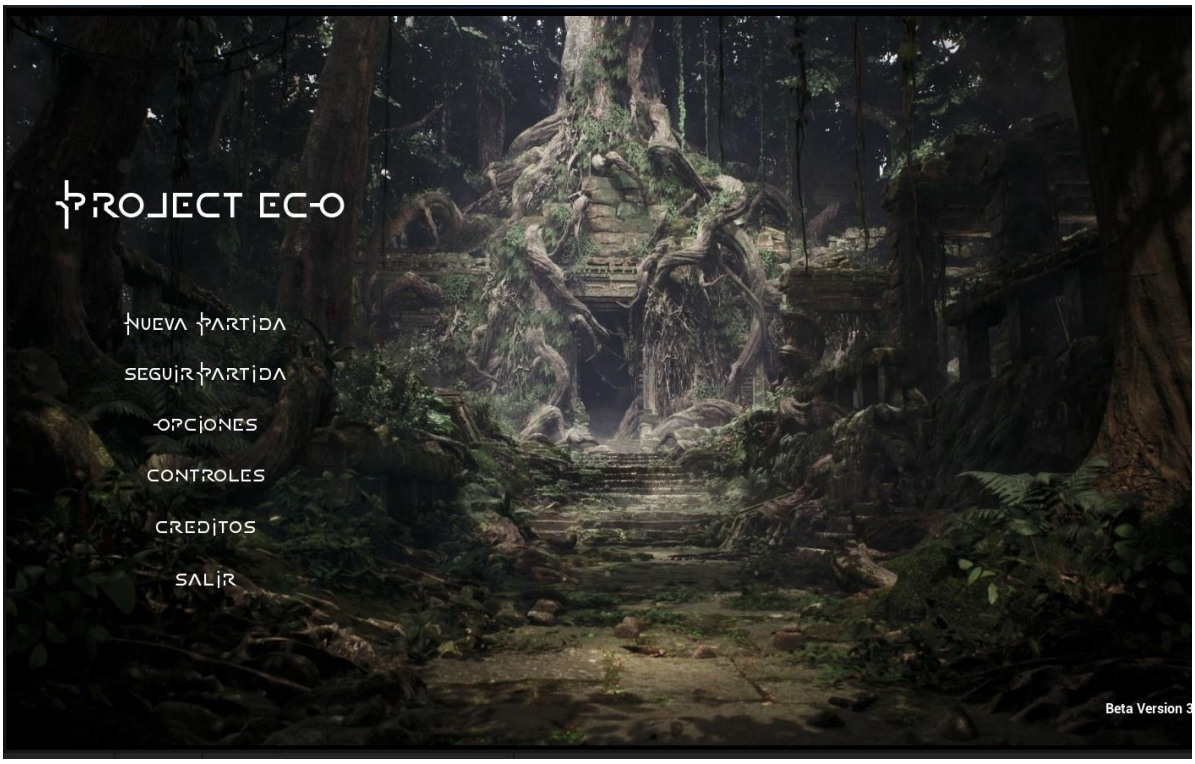


Figura 45. Menú principal del juego

6.3.2. Controles de juego

El juego está diseñado para la utilización de un teclado y ratón o bien, con un teclado y un gamepad.

- **Teclado**
  - W,A,S,D: Desplazamiento
    - Espacio: Salto
    - Tecla 1 y 2: Equipar arma
    - Shift: Correr
    - Bloq May: Rodar
- **Ratón**
    - Botón izquierdo: Ataque
    - Botón izquierdo + ctrl: Ataque cargado
    - Movimiento x y: Vista

- **GamePad**
    - Axis izquierdo: Movimiento
    - Axis derecho: Vista
    - Cruceta izquierda: Equipar espada
    - Cruceta derecha: Equipar magia
    - L2: Ataque cargado espada
    - R2: Ataque
    - Botón derecho: Rodar
    - Botón inferior: Saltar

- **Botones comunes en ambas configuraciones**
    - Tecla P: Pausa
    - Tecla M: Mapa
    - Tecla F: Dialogo

6.3.3. Juego

El juego empieza tras despertar la protagonista en un lugar desconocido, tras inspeccionar la zona el jefe del grupo dará indicaciones a Den para que realice las misiones necesarias por medio de conversaciones.

- **Armas**

  A medida que se avanza, se obtienen dos armas distintas para realizar ataques. Por un lado, tenemos una espada que puede realizar un ataque cargado con magia, cada ataque cargado

disminuye la barra de magia, y por otro lado disponemos de un ataque mágico. El ataque mágico causa menos daño que un ataque cuerpo a cuerpo y es difícil dañar al enemigo debido a que tiene un bajo rango de exactitud y requiere un tiempo para su ejecución con lo que el enemigo puede variar su posición.

- **Regenerar Vida**
  En caso de recibir daño, es posible regenerar parte de la barra recogiendo flores sanadoras.

Figura 46. Flor regeneradora de salud

- **Guardar partida**
  En cada punto del camino con un pequeño altar con flores y velas, es posible guardar la partida para regresar a ese punto si se pierde la vida o se desea continuar el juego posteriormente. Tan solo es necesario situarse cerca de la figura hasta que se escuche un sonido y aparezca una luz.

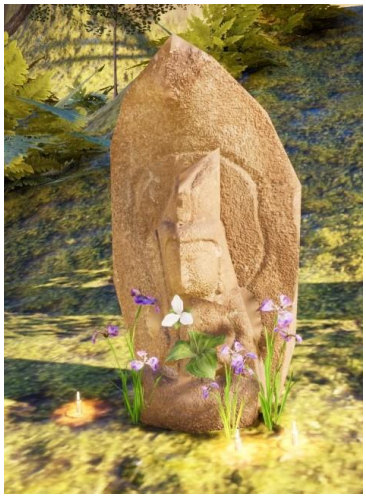


Figura 47. Altar de guardado de partida

- **Pausar juego o finalizar partida.**
  En caso de estar en medio de una partida y se desea parar el juego sin salir de él, se deberá presionar la tecla P para que aparezca el menú de pausa y el juego quede en espera. Si se desea terminar la partida mientras se juega, presionar la tecla P y seleccionar “Salir del juego” para que nos sitúe en el menú principal de Project Eco.

Figura 48. Menú de pausa

- **Mapa del juego**
  Al presionar la tecla M, aparece un mapa de la isla con sus zonas. Solo se conoce la isla por lo que no existe ningún otro mapa.

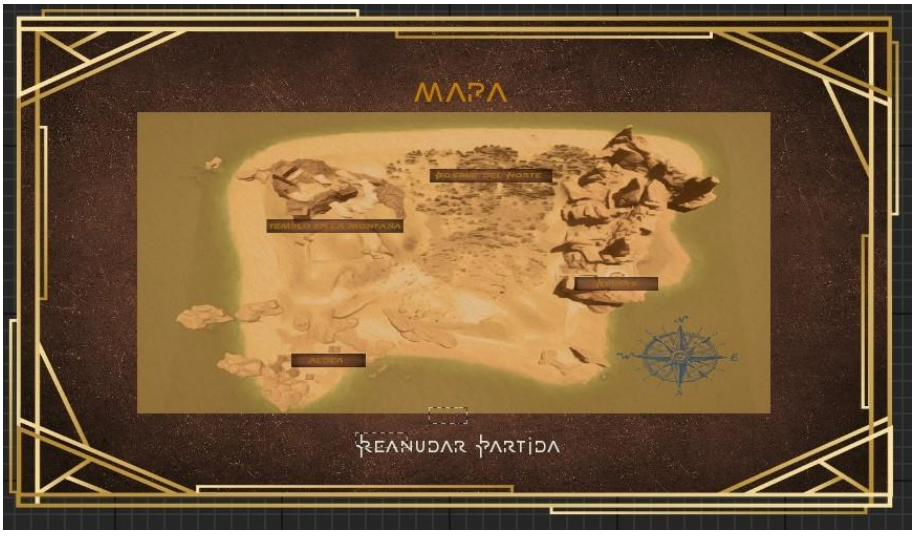


Figura 49. Mapa del juego

6.3.4. Interfaz de usuario

El juego dispone de un HUD visual que arroja información del estado del personaje en todo el transcurso de la partida.

Encontramos tres bloques diferenciados:

- Barras de estados
  - Barra verde: Vida restante
  - Barra azul: Magia restante
- Puntuación:
  - Barra: Muestra el avance hasta el siguiente nivel.
  - Numero: Nivel actual
  - Numeración: Puntos actuales | puntos siguiente nivel
- Armas:
  - Se muestran las armas a medida que se obtienen
  - Espada: Al apretar la tecla 1 o cruceta izquierda, se equipa esta arma
  - Magia: al presionar 2 o cruceta derecha, se equipa esta arma

Figura 50. Interfaz de usuario

7. Conclusiones y líneas de futuro

7.1. Conclusiones

El desarrollo de un juego como Project Eco ha sido un gran reto para mí, sin ningún tipo de experiencia más allá de la enseñada por la UOC el implementar un proyecto de este tamaño ha sido muy enriquecedor.

Tras pasar meses centrado en aprender y desarrollar cada área del juego, te das cuenta de lo importante que es cada persona dentro de un grupo de trabajo donde deben convivir distintas disciplinas. Esto te lleva a pensar en todos esos estudios indies que intentan lanzar sus juegos en distintas plataformas y que no suelen causar ningún impacto en los usuarios a pesar de su gran trabajo. Esos pequeños grupos de personas que con mucho esfuerzo y horas sacan productos con mucha calidad y que ahora, ves la enorme labor que hacen y los retos que deben superar.

En cuanto a la realización del proyecto, personalmente tenía muchas ganas saber cómo funciona la construcción de un juego. El alcance, como es lógico por el tiempo, es muy corto pero muy dinámico ya que permite tocar varias disciplinas y no estar centrado en una única tarea. Aprender un motor de tan altas especificaciones es un autentico reto y tras pasar estos meses trabajando con él, encuentro que tan solo conozco una pequeña parte de todo lo que puede hacer. Una vez te adaptas al motor y empiezas a aprender como funciona, se convierte en una herramienta muy ágil, que unido a los BluePoints para programar la convierten en el claro referente actual.

Aunque debo incluir que me deja una sensación un poco agridulce, a pesar de la gran potencia que ofrece Unreal 5 y su manejo cuando te adaptas, encuentro que es un sistema muy inestable, con continuos cierres del editor, fallos en texturas o funcionalidades, incluso en cierres del editor sin posibilidad de abrir el motor otra vez. Estos errores han provocado mucho retraso en el proyecto ya que obligan a buscar una solución en los grupos de internet y que en ocasiones lleva días. Realmente me parece exagerada la falta de robustez para controlar e indicar los errores de ejecución sin crasheos del motor.

En cuanto al resultado del proyecto debo decir que estoy contento con el producto, aunque queden cosas en el tintero por implementar. El poder realizar gran parte de del desarrollo sobre lenguaje C++ me reconforta y me ayuda a entender como funciona un videojuego desde un nivel más bajo.

Toda la planificación y los conceptos adquiridos durante el grado me han sido de gran ayuda, sin ellos no creo que llegase a cumplir los plazos de entrega, aunque algunos hitos se retrasaron, el conjunto de la planificación siguió los plazos establecidos.

En conclusión, el desarrollo de Project Eco ha sido muy gratificante, consiguiendo un producto cercando a las expectativas. Un proyecto donde poner en práctica parte de lo aprendido en el grado en cuanto a gestión de proyectos y programación. Aunque termino el proyecto con una sensación contradictoria en cuanto a Unreal Engine debido a su robustez.

7.2. Líneas de futuro

Como indico en mi conclusión, faltan cosas por añadir al juego por falta de tiempo y que me gustaría introducir en un futuro próximo.

En cuanto a nivel gráfico, quisiera incorporar otros conceptos de optimización como el level estreaming o World partition que permitan añadir elementos a los niveles. Con esta medida sería posible obtener unos escenarios de mayor calidad, parecidos a los realizados en primera instancia, pero que se tuvieron que degradar para una ejecución del juego más agradable.

En cuanto a desarrollo, me gustaría incorporar más mecánicas y retos en el juego, como plataformas móviles, escalada y puzles. A estas mecánicas se debería unir la optimización y mejora de las animaciones base de los personajes para que sean más fluidas.

Bibliografía

[1] **Epic Games**, *Unreal Engine Hardware and Software Specifications* [online], consultado el 08 de marzo de 2023. Disponible en: https://docs.unrealengine.com/5.0/en-US/hardware-and-software-specifications-for-unreal-engine/

[2] **Jobted,** *Sueldo del Programador de videojuegos en España* [online], consultado el 10 de marzo de 2023. Disponible en: https://www.jobted.es/salario/programador-videojuegos

[3] **Wikipedia**, *Videojuegos de ventura* [online], consulatdo el 25 de marzo de 2023. Disponible en: https://es.wikipedia.org/wiki/Videojuego_de_aventura

[4] **DestinoRPG,** *Historia de los videojuegos:Los orígenes del género RPG* [online], consultado el 25 de marzo de 2023, Disponible en: https://www.destinorpg.es/2015/08/historia-de-los-videojuegos-los.html?m=1

[5] **PWC**, *Perspectives from the Global Entertainment & Media Outlook 2022–2026* [online], consultado el 27 de marzo de 2023. Disponible en: https://www.pwc.com/gx/en/industries/tmt/media/outlook/outlook-perspectives.html

[6]**World Economic Forum,** *Los videojuegos están en auge y se espera que la industria siga creciendo. Este gráfico tiene todo lo que necesitas saber* [online], consultado el 27 de marzo de 2023. Disponible en: https://es.weforum.org/agenda/2022/09/el-juego-esta-en-auge-y-se-espera-que-siga-creciendo-este-grafico-le-dice-todo-lo-que-necesita-saber/

[7]**Atomix**, *Los 5 Zeldas Mejor venidios de la historia* [online], consultado el 28 de marzo de 2023. Disponible en: https://atomix.vg/los-5-the-legend-of-zelda-mas-vendidos-de-la-historia-2/

Anexos

Anexo A: Glosario

Asset: Un asset dentro de los videojuegos es cualquier objeto que puede ser utilizado dentro del proyecto, puede ser audio, imagen, modelo o cualquier archivo soportado por el motor gráfico.

C++: Mejora del lenguaje de programación C, es un lenguaje multiparadigma del tipo imperativo y orientado a objetos.

NPC: Non Playable Character o Personaje No Jugable (PNJ), es un personaje que interviene en el juego de forma activa o pasiva pero que no puede ser controlado por el jugador.

IA: La Inteligencia Artificial son sistemas que permiten a las maquinas imitar el comportamiento de una inteligencia viva.

Cloud Gaming: Es una forma de jugar en la nube por medio de hardware remoto de una compañía.

Streaming: Tecnologia que permite transmitir contenidos multimedia y visualizarlos directamente desde la red.

HUD: Head-Up Display es la información que se muestra en todo momento en la pantalla mientras se juega a un juego, barra de vida o puntuación, por ejemplo.

Blender: Es un programa de modelado y animación 3D gratuito.

Motor gráfico: Es un entorno con librerías de programación que permiten la creación, diseño y representación de un videojuego.

Motor propietario: Se trata de los motores gráficos creados y utilizados por una empresa y que no se encuentran disponibles en el mercado.

Nanite: Es un sistema de virtualización de geometría que pretende convertir grandes cantidades de triángulos en objetos, estos triángulos son más fáciles de tratar por el motor.

Lumen: Es un sistema de iluminación global dinámica que no llega a necesitar Raytracing y por lo tanto es más liviano para los cálculos.

Sinlge Player: Se trata de que solo juega una sola persona al juego.

Video Pitch: Es la variante de “Elevator Pitch”, se trata de vender tu producto en un tiempo muy corto, en este caso captar el interés por tu producto con un video de segundos.

Lore: Historia de una obra.

Blueprint: Sistema de programación visual de Unreal Engine, donde se dispone de unos nodos que hacen las funciones y unos conectores para el flujo.

Popping: Es un efecto visual no deseado, pasa cuando objetos en el juego aparecen de forma espontanea en vez de ser dibujados a medida que nos acercamos a él.

Anexo B: Entregables del proyecto

Código fuente del proyecto, repositorio en GitHub:

- https://github.com/JAArtero/ProjectTFG_U_v1

Juego compilado y empaquetado, GitLab

- https://gitlab.com/jaaf_UOC/betastfg