Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

Universidad de Jaén

Escuela Politécnica Superior de Jaén

TRABAJO FIN DE GRADO

ECHOES OF THE ABYSS – DESARROLLO PROTOTIPO DE VIDEOJUEGO

Alumno

Francisco Javier Molina Arcos

Tutor

Juan Roberto Jimenez Perez

(Departamento de Informática)

Septiembre, 2023

Universidad de Jaén

Departamento de Informática

Don Juan Roberto Jimenez Perez, tutor del Trabajo Fin de Grado titulado: '**Echoes of the abyss – Desarrollo prototipo de videojuego**', que presenta Don Francisco Javier Molina Arcos, otorga el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2023

El alumno: El tutor:

Fco Javier Molina

Francisco Javier Molina Arcos Juan Roberto Jimenez Perez

Agradecimientos

En el tiempo que he realizado este proyecto he rechazado de forma constante a cualquier plan con una excusa idéntica, la realización del Trabajo de Fin de Grado, por lo que pido perdón y doy a las gracias a todo mi entorno, amigos y familiares, que además de lo mencionado me han aportado un gran impulso con su ayuda y consejos.

También quiero agradecer a todas las personas que dedicaron su tiempo a contestar mi estudio de mercado.

Finalmente, quiero mencionar a mi tutor, Juan Roberto Jiménez Pérez que me ha instruido en el camino de mi entrega dandome orientación en todo momento incluso habiendo actualizado su agenda en varias ocasiones para poder atenderme.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Especialidad (solo TFG)	Sin especialidad
Mención (solo TFG)	Sistemas Gráficos
Idioma	Español
Tipo	General
TFT en equipo	No
Autor/a	Francisco Javier Molina Arcos
Fecha de asignación	10/11/2021
Descripción corta	Este trabajo de fin de grado constituye una propuesta para el desarrollo de un prototipo de videojuego que se caracteriza por ser un juego de plataformas de orientación vertical, en el cual la mecánica principal se basa en el salto regulable, planteando como desafío llegar a la cima. Los objetivos del TFG incluyen la realización de un estudio de mercado orientado a indagar sobre las preferencias de posibles compradores en el ámbito de los videojuegos, la definición de las mecánicas de juego principales, la selección del motor de juego más apropiado y otras herramientas complementarias para el desarrollo, así como el análisis y diseño de la interfaz, la interacción con el usuario y el mundo del videojuego, y finalmente, la evaluación del prototipo resultante.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Contenido

Índice de ilustraciones

1 ESPECIFICACIÓN DEL TRABAJO

1.1 Introducción

El desarrollo de “Echoes of the Abyss” está motivado desde antes de comenzar el grado; mi plan original era realizar el Grado de Diseño y Desarrollo de videojuegos debido a mi gran pasión desde pequeño por este campo. Sin embargo, este se ofertaba en lugares distantes, de forma que opté por tomar el Grado de Ingeniería Informática ya que la asignatura “Desarrollo de Videojuegos” fue incluida de forma reciente.

Al profundizar en la asignatura, comprendí que la creación de un videojuego conlleva una serie de dificultades técnicas y creativas, dividiéndose en un equipo de desarrollo convencional en tareas asociadas a especialistas en cada área; esto me llevó a la convicción de que realizar un proyecto de tal envergadura de manera individual generaría una satisfacción personal significativa.

Antes de decidir el tipo y visión del videojuego, haré un estudio de mercado. Difundiré un cuestionario por distintas plataformas en línea para obtener suficientes respuestas significativas, ya que uno de los aspectos fundamentales del desarrollo es sacarle rentabilidad en el mercado, convirtiéndose así la opinión de los posibles compradores en un factor clave.

A lo largo de este informe, se presentarán gráficos y análisis relacionados con los resultados de este estudio de mercado, aportando datos concisos en el contenido que se trata.

1.2 Objetivos del trabajo

Respecto a los objetivos de mi Trabajo de Fin de Grado se encuentran los siguientes:

- Identificar el juego a desarrollar, destacando las características principales del mismo, así como el público objetivo.
- Definir las mecánicas principales el videojuego.

- Seleccionar el motor de juego más adecuado, así como otras herramientas complementarias que se requieran en el desarrollo o en interacción con el juego.
- Analizar y diseñar la interfaz e interacción con el usuario, así como identifica los dispositivos de interacción más adecuados.
- Analizar y desarrollar el mundo que se desarrolla.

Como base para cumplir estos objetivos me he basado en el estudio de mercado de los compradores potenciales como he comentado anteriormente; expondré de forma más detenida estos objetivos a lo largo del documento.

1.3 Antecedentes y estado del arte

El primer juego data de 1952, desarrollado como una investigación en la Universidad de Cambridge; desarrollaron el clásico tres en raya. En 1958, Bill Nighinbottham creo un juego básico con la dinámica del tenis que lo llamo Tennis for Two pero no lo registro por lo que Nolan Bushnell inauguro la compañía "Atari" en 1972 y lanzo el juego mencionado con el nombre PONG [1] (Ilustración 2).

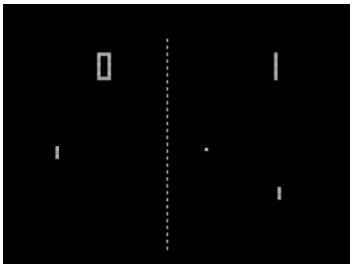


Ilustración 1: PONG

A partir de este momento empezaron a surgir nuevos juegos arcade, consolidándose como una gran alternativa de ocio en aquella época, hasta que se popularizaron las consolas. En 1972, surgiría la primera consola domestica denominada Magnavox Odyssey[2] (Ilustración 3, izquierda), esta se trataba de un

[1] En 1971, lanzo un juego llamado Computer Space

[2] El prototipo fue desarrollado por la empresa Sanders Associates, llamándolo "La caja marrón".

sistema multijugador de videojuegos que podía conectarse a la televisión. Cinco años más tarde, aparecería la famosa Atari 2600, siendo esta la primera consola que alcanzo una mayor audiencia, superando los 30 millones de ventas (Ilustración 3, derecha). [Retromaquinitas a]

Ilustración 2: Magnavox Odyssey y Atari 2600

Aunque siguieron saliendo consolas de primera generación como "Mattel Intellivision" [3]. La Atari no sería desbancada hasta poco después del crash del 83[4] cuando fue lanzada la Nintendo Entertainment System (Ilustración 4, izquierda) por parte de Nintendo en 1986. En 1988, Sega sacaría al mercado su Mega Drive Genesis pero no fue rival para la NES, además al año siguiente Nintendo realizaría la evolución definitiva de las consolas portátiles con el lanzamiento de la consola, "Game Boy" (Ilustración 4, derecha). [Tokio School]

Ilustración 3: NES y GameBoy

[3] Mattel Intellivision tenía mejores gráficos y era más potente que Atari 2600 pudiendo soportar juegos más sofisticados pero la falta de títulos de renombre ocasiono que sus ventas fueran inferiores ocasionando que Mattel abandonara los videojuegos en 1983. [Retromaquinitas b]

[4] Entre 1983 y 1985 ocurrió un declive de la industria del videojuego pasando los ingresos de 2.59 billones a 81 millones de euros, una pérdida de un 97%. [Nintendo Fandom]

En 1996, Nintendo volvería a revolucionar la industria del videojuego con el lanzamiento de la Nintendo 64 (Ilustración 5), esta fue llamada así porque disponía de un procesador de 64 bits, a diferencia de sus competidoras directas; Sega con su consola “Saturn” y Sony con su consola “PlayStation 1” (Ilustración 5). [Niixer 2020]

Ilustración 4: Nintendo 64, Saturn y PlayStation1

En la década de los 90, surgió la expansión online de los videojuegos ocasionando nuevas experiencias multijugador, contenido adicional y servicios en línea, además, en esta misma década, la compañía Sony lanzó al mercado una nueva consola denominada “PlayStation2”. Esta consola posee a día de hoy el Record Guinness por ser la consola más vendida de la historia y por poseer la mayor cantidad de títulos jugables.

Ilustración 5: PlayStation2

A partir de este punto podemos considerar que empezó la época moderna de los videojuegos. Gracias a los nuevos avances tecnológicos de la época, las consolas de nueva generación (PS3, XBOX360 y WII) dotaron a los videojuegos con gráficos de alta resolución, Inteligencia Artificial, historias y diseños más sofisticados.

Ilustración 6: PlayStation3, XBOX360 Y WII

Todo lo mencionado anteriormente en relación a los antecedentes guarda cierta conexión con los juegos independientes (Indies). Empresas como Sony y Nintendo siempre han ofrecido juegos Triple A, es decir, juegos con una gran cantidad de recursos, tanto humanos como monetarios, pero, el avance de la tecnología ha permitido que cualquier persona actualmente pueda adentrarse en esta aventura, aportando al sector del videojuego una gran amplitud de visiones e historias.

La industria del videojuego está en constante cambio, y en la actualidad, se realizan investigaciones en busca de nuevas formas de revolucionar los videojuegos. Uno de los campos más prometedores es el orientado a las interfaces EEG que utilizan la electroencefalografía para medir y registrar lo pulsos eléctricos que emiten el cerebro para poder controlar videojuegos con el pensamiento.

1.4 Descripción de la solución propuesta

"Echoes of the abyss" se encuentra situado en el mundo descrito en "La divina comedia" de Dante Alighieri [5], combinado con una nueva versión de la historia de Caperucita Roja.

En el videojuego que he desarrollado, asumes el papel de Caperucita Roja como protagonista principal, una joven de voluntad fuerte y curiosa que, al igual que en la versión original, se embarca en una aventura llena de misterios con el afán de encontrar a su abuela, la cual ha desaparecido en extrañas circunstancias.

La aventura comienza con Caperucita esperando en casa de su abuelita a que llegue al igual que todos los días, pero se produjo la puesta de sol y aún no había llegado; cuando ya no existía ápice de luz en el ambiente decidió sumergirse en el tenebroso bosque.

Guiada por una misteriosa luz, Caperucita se dirige hacia ella y descubre un enigmático portal del que brota una carta sin remitente con el siguiente mensaje "Tu abuela yace en el interior del portal". Al atravesar el portal, Virgilio[6] te comenta lo que él sabe acerca del paradero de tu abuela, avisándote de los peligros que atravesaras y deseándote suerte en tu camino.

Es un juego de plataformas que se distingue de títulos como Super Mario Bros al presentar una dinámica vertical en lugar de horizontal. En este juego la acción se desarrolla de abajo a arriba, otorgando al salto regulable un papel crucial como una de las mecánicas más destacadas y esenciales.

[5] Poema escrito en el Siglo XIV, considerada una obra ilustre de la literatura italiana.

[6] El guía que acompaña a Dante, en la Divina Comedia, a través del Infierno y el Purgatorio.

1.5 Estudio de mercado

Como previamente expresé, como parte fundamental de la estrategia de desarrollo del videojuego, he llevado a cabo una exhaustiva encuesta de mercado difundida en las plataformas de redes sociales, Twitter, Instagram y Tik Tok. Esta campaña de promoción me ha generado un buen apoyo, acumulando un total de 263 respuestas por parte de posibles jugadores hispanohablantes.

Estos datos me proporcionarán información valiosa para mejorar y adaptar el juego de acuerdo a las preferencias de la audiencia. Además, ha brindado datos esenciales tanto como para el modelo de negocio como para la potencial entrada del videojuego en el sector.

A continuación, incluiré todas las representaciones gráficas a las que hago referencias en la memoria obtenidas del estudio de mercado.

1.5.1 Edad de los participantes

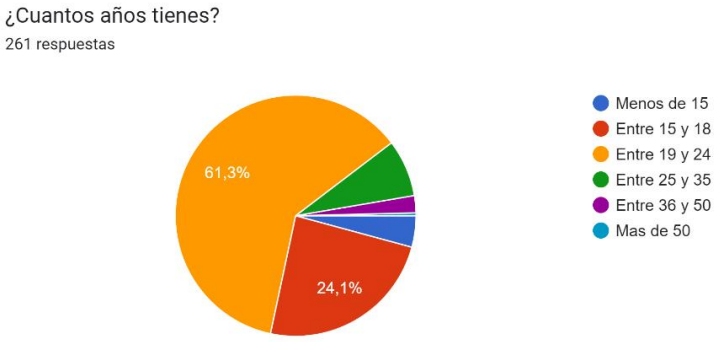


Ilustración 7: Estudio de mercado: Edad de los participantes

1.5.2 Tipos de música escuchados de forma habitual

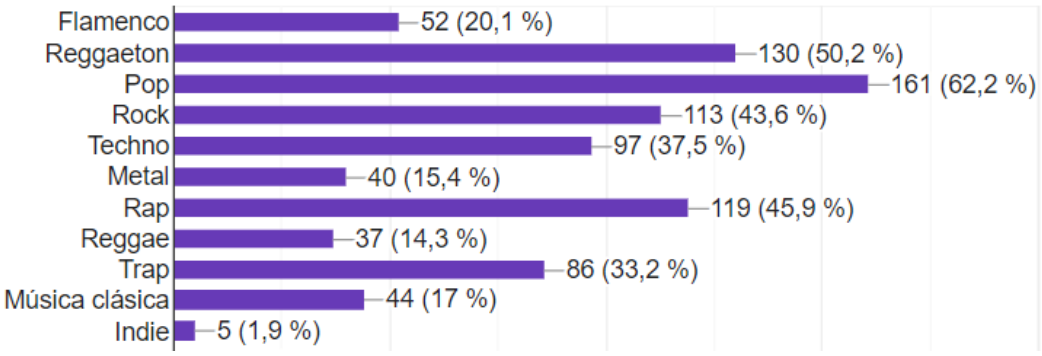


Ilustración 8: Estudio de mercado - Tipo de música escuchada

1.5.3 Tipos de dispositivos utilizados para jugar a videojuegos

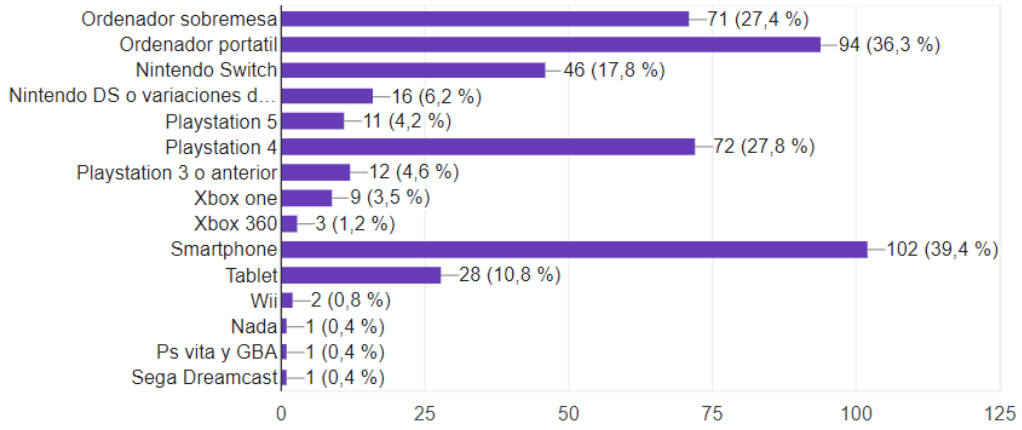


Ilustración 9: Estudio de mercado - Dispositivos utilizados para videojuegos

1.5.4 Controles preferidos

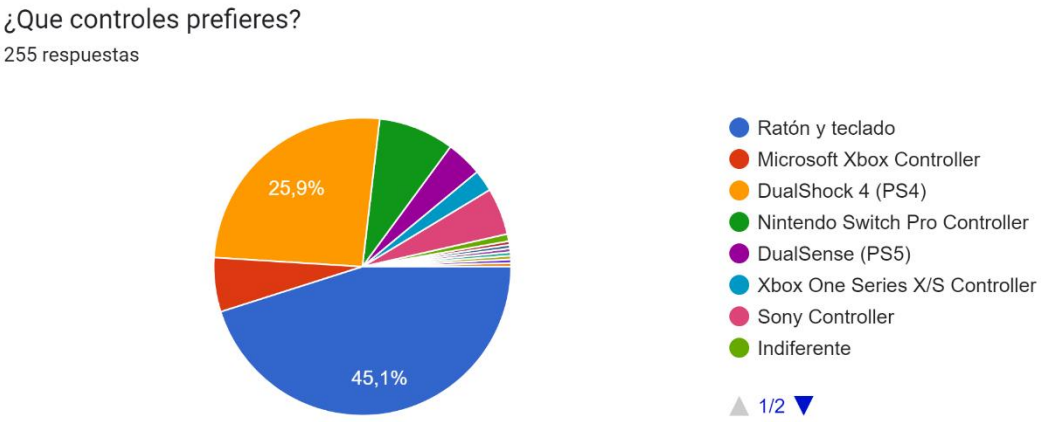


Ilustración 10: Estudio de mercado - Controles preferidos para jugar

1.5.5 Factores determinantes a la hora de comprar un juego

Ilustración 11: Estudio de mercado – Primer factor en la compra de un videojuego

1.5.6 Portal principal de información sobre videojuegos

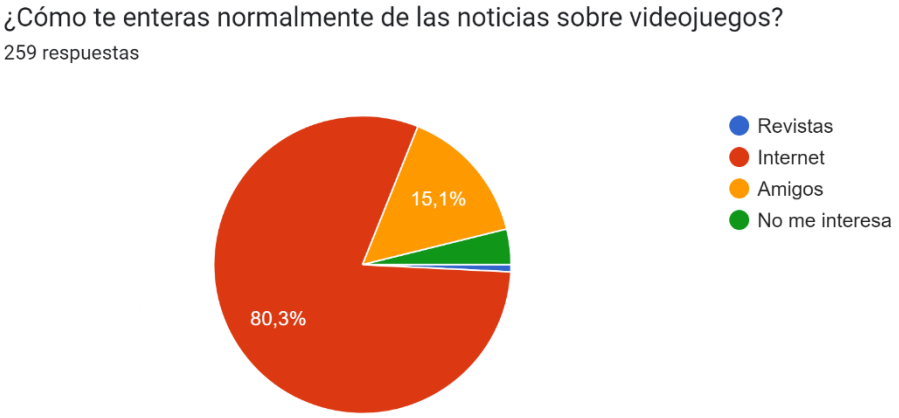


Ilustración 12: Estudio de mercado - Medio principal de información sobre videojuegos

1.5.7 Precio gastado por año en videojuegos

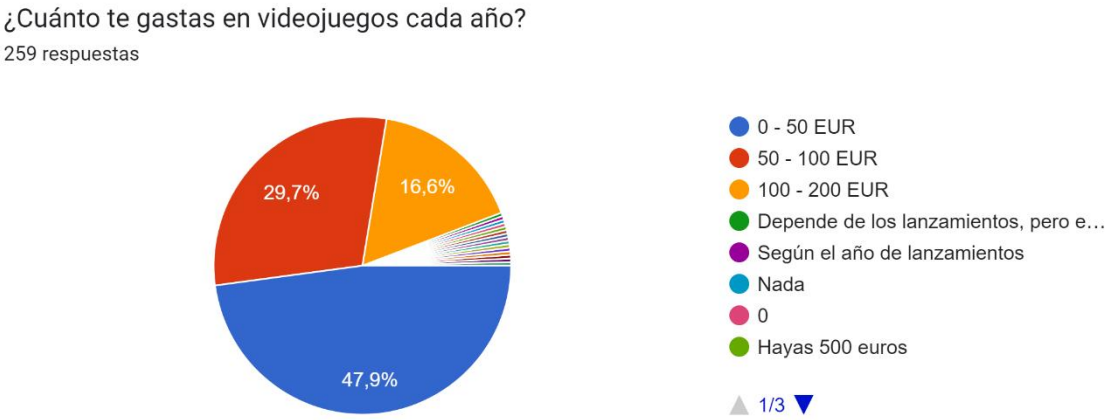


Ilustración 13: Estudio de mercado - Gastos por año en videojuegos

1.5.8 Los videojuegos como arte

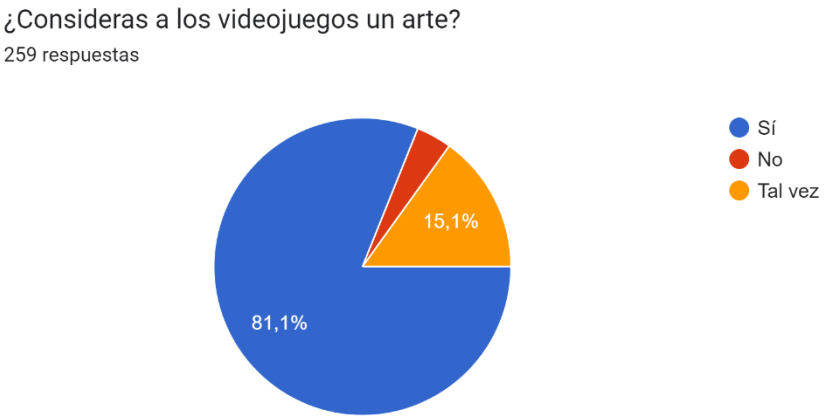


Ilustración 14: Estudio de mercado - ¿Los videojuegos son un arte?

1.5.9 ¿Los videojuegos son una pérdida de tiempo?

Ilustración 15: Estudio de mercado - ¿Los videojuegos son una pérdida de tiempo?

1.5.10 Tiempo medio utilizado al día para jugar a videojuegos

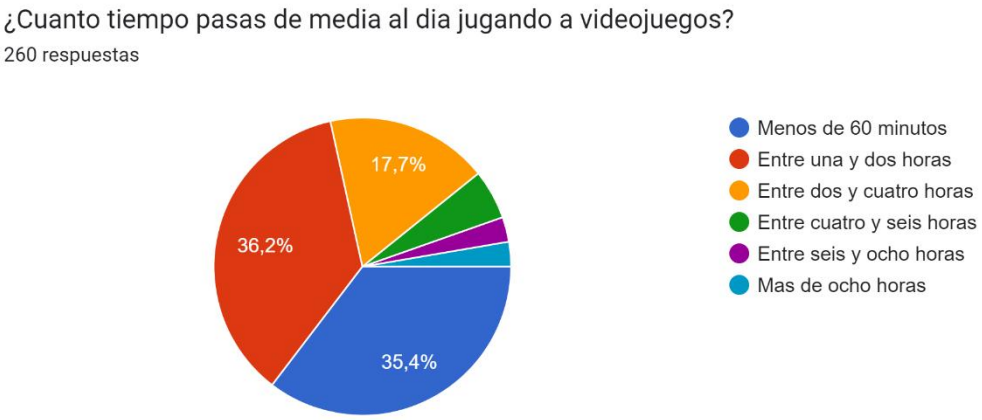


Ilustración 16: Estudio de mercado - Tiempo promedio diario de juego

1.5.11 Tiempo jugado respecto a la cuarentena por Coronavirus

Ilustración 17: Estudio de mercado - Tiempo jugado respecto a la cuarentena por Coronavirus

1.6 Requisitos iniciales

Para garantizar que el juego cumple con los estándares deseados y otorga una experiencia de juego envolvente y agradable se deberá asegurar que en el desarrollo se cumplan los siguientes requisitos iniciales:

- **Estabilidad y rendimiento**
  El desarrollo debe ser lo suficientemente meticuloso como para asegurar que no se produzcan errores ni caídas de fotogramas para así fomentar el uso del jugador sin interrupciones indeseadas.

- **Fidelidad a la divina comedia**
  Debería representarse de manera fidedigna los elementos del mundo relacionados con la temática y el ambiente de la "Divina comedia" de Dante Alighieri.

- **Movimiento del personaje principal**
  Caperucita Roja debe moverse de manera natural y fluida, sin problemas ni anomalías, para que el jugador pueda interactuar de forma orgánica y tranquila con el sistema.

- **Comprobación de niveles**
  Una vez realizado el diseño de los niveles se debe comprobar que todos se puede llegar a todas las plataformas de forma correcta.

- **Sistema de guardado**
  Se debería implementar un sistema de guardado que permita al jugador guardar su progreso en múltiples ranuras facilitando poder regresar a momentos específicos del juego.

- **Estética visual agradable**
  El juego debería presentar una estética que represente de forma coherente el entorno de la Divina Comedia combinado con algunos elementos de la cultura popular y la mitología griega.

- **Interfaz de usuario intuitiva**

  La interfaz de usuario (UI) debe diseñarse de manera intuitiva y clara para que los jugadore puedan navegar fácilmente y sin confusión a través de los menús y opciones.

- **Enemigos desafiantes**

  En el mundo se debería poder encontrar distinto tipos de enemigos que nos empujen de alguna forma provocando que paremos nuestra subida y descendamos en el mapa.

- **NPC o personajes no jugables**

  Caracteres cuya única interacción con el jugador sea la de realizar un dialogo haciendo así más claro el contexto de la historia general.

- **Internalización y localización**

  Se debe traducir y localizar el juego en varios idiomas y ubicaciones consiguiendo que pueda ser disfrutado por una mayor audiencia potenciando así los beneficios.

- **Banda sonora y efectos**

  Se debería añadir un sistema de audio que provoque una atmosfera que facilite la inmersión del jugador en el videojuego. El tipo de banda sonora que será incluida en el videojuego es música clásica ya que nos proporciona, bajo mi punto de vista, una atmosfera única con gran profundidad y variedad, permitiendo que el jugador se sumerja aún más en el videojuego, lo cual es un aspecto crucial. Además, podemos ver gracias al estudio que el 17 por ciento de los encuestados (Ilustración 8: Estudio de mercado - Tipo de música escuchada escucha este tipo de composiciones, provocando así que sea un atractivo más a añadir.

- **Multiplataforma**

  Debería ser desarrollado en múltiples plataformas de forma eficiente para así ampliar la audiencia, pero debido al tiempo plantee en el estudio la pregunta “¿Con que dispositivo juegas con frecuencia?”, gracias a ella, podemos ver

como domina en la representación gráfica (Ilustración 9: Estudio de mercado - Dispositivos utilizados para videojuegos la PlayStation 4, los ordenadores y los smartphones, entonces se intuye que estos son los dispositivos más rentables para los que desarrollar.

- **Controles**

  A pesar de la aparente asociación con la naturaleza multiplataforma, este requisito no está completamente ligado a ello. Aunque es cierto que algunas plataformas están restringidas a un tipo especifico de controlador, esto no se aplica uniformemente. Por ejemplo, en el caso de las computadoras, existe la flexibilidad de admitir varios tipos de controladores.

  Observando el gráfico (Ilustración 10: Estudio de mercado - Controles preferidos para jugar), podemos considerar que, en el contexto de las computadoras, también deberíamos tener en cuenta la posibilidad de realizar la compatibilidad en el juego con los controles de Xbox y PlayStation.

1.7 Restricciones

Uno de los principales problemas ha sido el tiempo disponible, ya que la creación de un videojuego es un trabajo arduo y minucioso donde se puede absorber muchas horas debido a detalles muy pequeños.

Si bien la complejidad no se limita únicamente a la programación en sí misma, también surge debido a la necesidad de generar una cantidad considerable de diseños. A pesar de que he confeccionado varios diseños por mi cuenta, la ejecución de un desarrollo de esta envergadura de manera de individual no es muy realista. Se puede pensar en pedir diseños por encargo a terceras personas o conseguirlos a través de distintas plataformas en la web, pero nos encontramos con el inconveniente de que los diseños más detallados tienen asociado un coste, teniendo que considerar y meditar, en este caso, sobre las restricciones económicas que tengamos.

1.8 Alcance

En esta sección, abordaré todos los elementos que no podré desarrollar debido a las restricciones comentadas.

Respecto a la multiplataforma, aunque los datos obtenidos en la representación gráfica (Ilustración 9: Estudio de mercado - Dispositivos utilizados para videojuegos), se puede ver como el porcentaje de público objetivo que utilizan los dispositivos móviles para jugar es muy elevado. Con el fin de lograr una adaptación exitosa, sería necesario rediseñar y reconfigurar la interfaz de usuario (UI) siendo más amigable en pantallas móviles, implementar controles compatibles con la pantalla táctil y ajustar el tamaño general del juego. No obstante, con las condiciones actuales, no es viable. Aunque para PlayStation 4 y Nintendo Switch el porcentaje también es elevado, pero pienso que no suficiente para el esfuerzo necesario a realizar.

En el caso de los controles, dentro del lenguaje de programación existe un código el cual permite detectar la pulsación de teclas. Sin embargo, esta implementación es poco configurable ya que no considera la variedad de controladores disponibles (solo el teclado y el ratón) y además no puede modificarse fácilmente desde fuera de los scripts al estar definido de forma directa en el código. Esta limitación lleva a los videojuegos comerciales a utilizar utilidades o recursos que son adquiridos de forma adicional. A continuación, proporcionaré una breve introducción sobre el recurso mencionado.

En un comienzo se debe crear una lista con todas las acciones posibles a realizar en el juego y en el código, en vez de comprobar si se ha pulsado una tecla en especifica, se utiliza la clase de Rewired para comprobar si se ha realizado una acción de la lista que hemos creado antes, finalizando con la asignación del pulsador del controlador deseado con la acción en especifica. No se realizará debido a su coste.

1.9 Tecnologías utilizadas

En esta sección, comentare las tecnologías utilizadas para la creación de mi videojuego.

En primer lugar, debo mencionar el motor de videojuegos, un entorno que brinda las herramientas esenciales para desarrollar cualquier videojuego. En este contexto, mi elección ha sido Unity[7]; esta selección se basa en múltiples razones, pero destaca como factor fundamental el hecho de estar familiarizado con el motor ya que fue el utilizado durante la realización de la asignatura “Desarrollo de videojuegos”, desarrollando en ella conocimiento que me han permitido manejarme por el entorno con cierta fluidez.

No obstante, descartando la experiencia previa que poseía, Unity tiene una interfaz sumamente intuitiva facilitando la navegación sin problemas entre los distintos menús y opciones. Además, presenta una gran adaptabilidad a distintas plataformas, alzándose como un punto importante para conseguir mayor audiencia, como comenté en puntos anteriores. Vale la pena destacar que su gratuidad proporciona la llegada de varios desarrolladores fomentando así una comunidad activa, donde resulta sencillo obtener respuestas a tus dudas.

En cuanto al aspecto gráfico, he empleado herramientas como Photoshop[8], GIMP[9] y Pixel Studio[10]. Gimp lo he usado más para modificar y ajustar recursos o imágenes que he obtenido de la web, debido a mi mayor experiencia con este programa de edición de imágenes. Por otro lado, he utilizado Photoshop para tareas como el rellenado de contenido libre de una imagen, ya que considero que ofrece un rendimiento superior en este aspecto. En última instancia, es importante mencionar a Pixel Studio que ha sido especialmente importante en mi videojuego de tipo pixel art. Dado que esta específicamente orientado hacia la creación de imágenes para este tipo de juego, otorga herramientas de fácil comprensión, facilitándome enormemente la tarea de edición de imágenes en algunos aspectos.

[7] https://unity.com/es

[8] https://www.adobe.com/es/products/photoshop.html

[9] http://www.gimp.org.es/

[10] https://store.steampowered.com/app/1204050/Pixel_Studio__pixel_art_editor/

Para organizar mi tiempo, he usado Toggl[11], una herramienta web donde puedes registrar el tiempo dedicado a diferentes categorías diferentes. A pesar de haber descubierto esta herramienta en una etapa relativamente avanzada de la realización del TFG, ha demostrado ser de un gran valor al mejorar mi gestión del tiempo, poniendo el foco en diferentes estados, permitiendo entregar un prototipo del proyecto en los plazos requeridos.

1.10 Estudio de alternativas y viabilidad

Cuando se trata del diseño de imágenes, considero que GIMP, Photoshop y Pixel Studio me brindan todo lo necesario en este sector. Sin embargo, en el campo de los motores gráficos se encuentra un competidor destacado para Unity, siendo este Unreal Engine.

Unreal Engine destaca principalmente sobre Unity en la capacidad de generar gráficos con un elevado detalle de realismo. Si tu objetivo es lograr un gran ambiente visual y llamativo donde destaquen elementos como las cinemáticas, optar por Unreal Engine sería la opción más acertada.

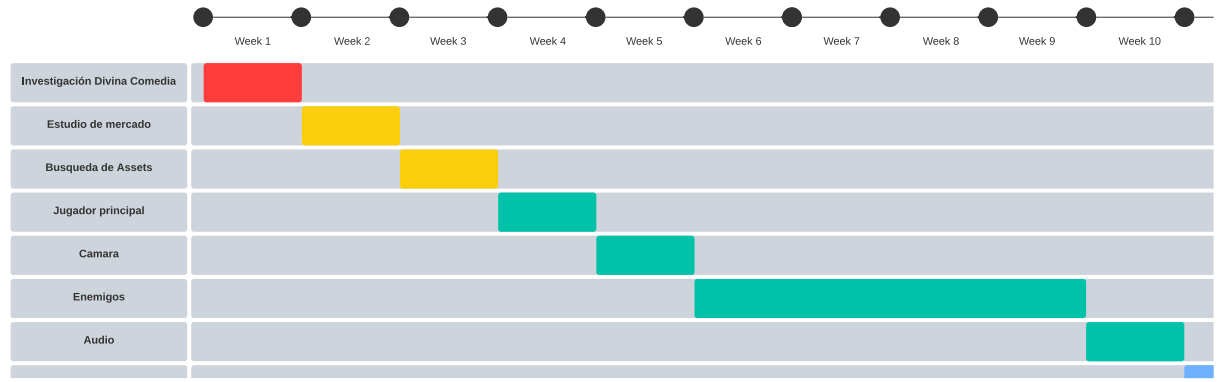
Además, disponemos del sistema Blueprint Visual Scripting. Este sistema utiliza una estructura de nodos para crear elementos del juego directamente desde la interfaz de Unreal siendo factible crear clases y objetos desde el propio motor, sin tener conocimientos de programación.

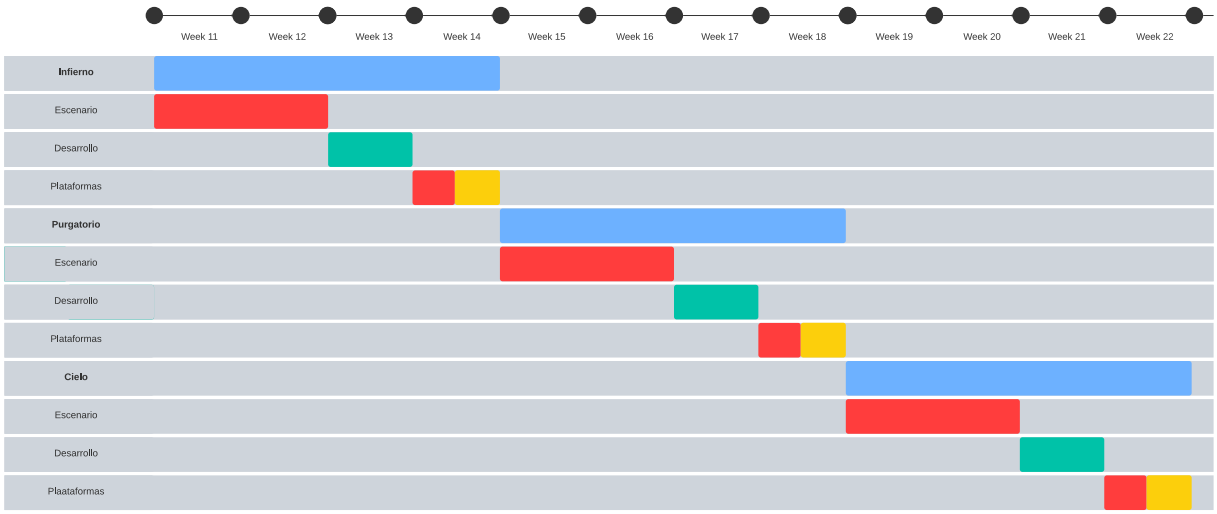
1.11 Planificación temporal

He llevado a cabo una planificación que abarca un periodo de 29 semanas. Esta planificación la muestro de manera detallada en el siguiente diagrama de Gantt, donde se exhiben las actividades programadas y su correspondiente periodo de ejecución. Esta planificación fue elaborada al comienzo del proyecto, por lo que es posible visualizar actividades que no han sido realizadas, como se ha especificado en la sección de alcance.

[11] https://toggl.com/

Ilustración 18: Leyenda del Diagrama de Gantt





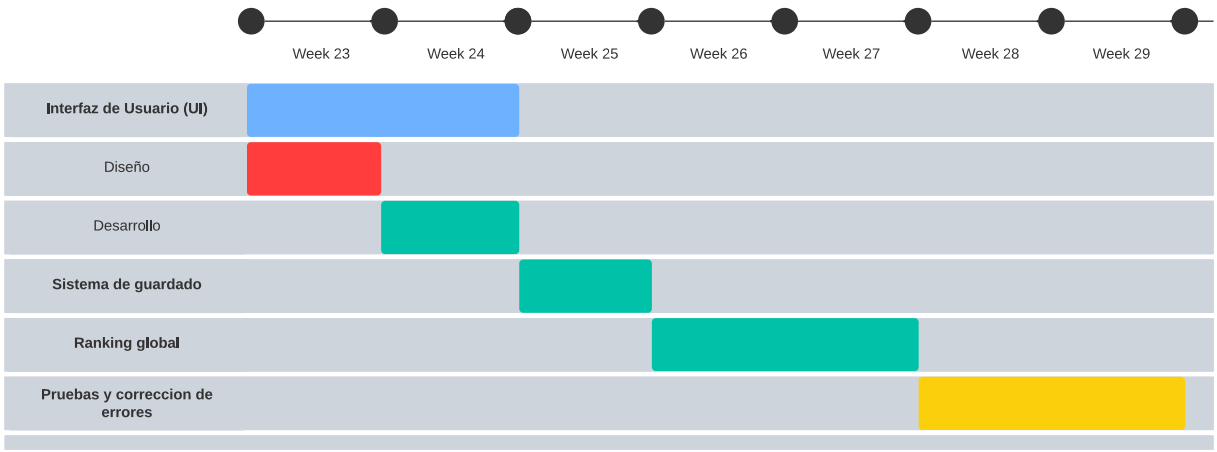


Ilustración 19: Diagrama de Gantt

Para visualizarse de forma óptima el diagrama de Gantt, he decidido adjuntarlo junto a la memoria del proyecto y también se puede acceder a él desde el enlace en la sección “Otras referencias”.

1.12 Metodología de desarrollo de software

Para poder llevar a cabo la realizacion de este proyecto de una forma dinámica, he seguido un desarrollo basado en la metodología Ágil o Scrum, en el cual, de forma semanal, me planteaba los objetivos a cumplir para la semana siguiente.

Debido a que en este caso no he tenido un cliente como tal, al que poder verificar y mostrar estos cambios, he contado con la ayuda de mi entorno a los cuales, de forma semanal, iba mostrándoles los cambios que he ido realizando. Ellos verificaban los cambios realizados y proponían una serie de cambios con el objetivo de mejorar el videojuego.

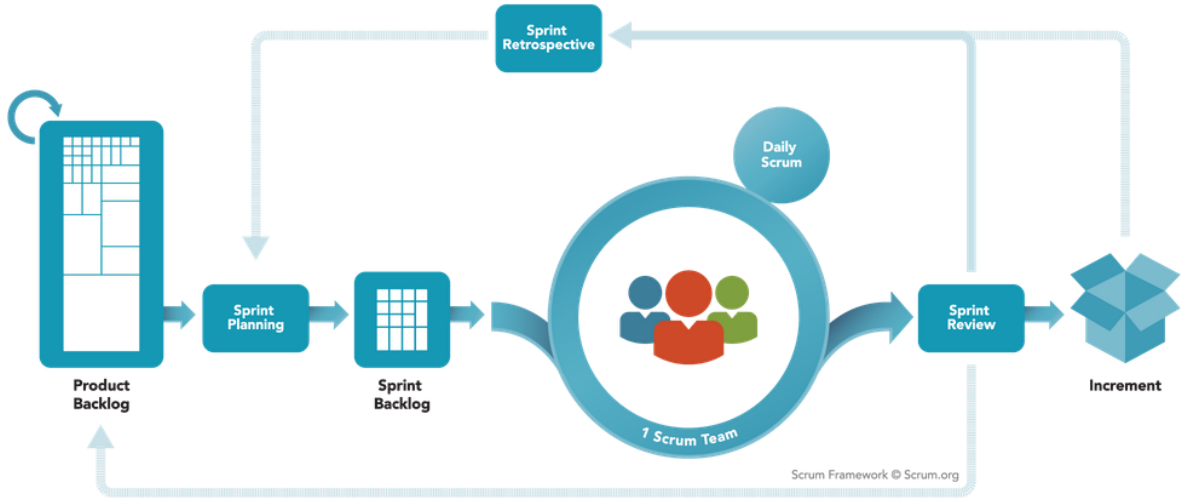


Ilustración 20: Desarrollo Ágil o Scrum

1.13 Presupuesto

El cálculo del presupuesto para el desarrollo de “Echoes of the Abyss” es complejo de obtener de forma precisa, ya que he intentado conseguir la mayoría de recursos de forma gratuita. Esto provoca que en el caso de lanzar el videojuego al público, será necesario contactar con los propietarios de los diseños que he utilizado, para obtener su permiso o llegar a un acuerdo económico.

Sin contar lo mencionado, los gastos para obtener el diseño del personaje principal han sido de veinte euros. En el caso de haber optado por desarrollar los controles, se agregarían cuarenta y cinco euros para el recurso Rewired. Además, se debe añadir el sueldo a percibir por parte del desarrollador. Si estimamos que el

tiempo completo de desarrollo para finalizar el proyecto son de mil horas, a nueve euros de media por hora, son nueve mil euros. También se debe tener en cuenta el presupuesto para promocionar el estudio de mercado que ha sido de treinta euros. Si sumamos todo lo mencionado obtenemos un resultado de 9095€.

Con el propósito de documentar de manera más precisa los recursos empleados y sus respectivos valores, presentare la siguiente tabla en la que categorizo los recursos utilizados, detallando su utilidad, indico su valor actual y proporcionando un enlace para su localización:

Utilidad	Valor (€)	Enlace
Decoración, Plataformas: Tile Set Bosque Herejes	0	https://goo.su/RiPhPyb
Decoración: Arbustos	0	https://goo.su/6Mjrf83
Decoración: Armas	0	https://goo.su/dVMfx2o
Decoración: Ballena	0	https://goo.su/B4DNmLL
Decoración: Candelabro	0	https://goo.su/2qX0K
Decoración: Demonio	0	https://goo.su/Q1o7pq8
Decoración: Entorno tétrico	0	https://goo.su/V6xFlVB
Decoración: Espada Minecraft	0	https://goo.su/ZiBXj
Decoración: Espejos	0	https://goo.su/0RUzO
Decoración: Fósil	0	https://goo.su/B00A8EL
Decoración: Gato negro	0	https://goo.su/8llihAV
Decoración: Manzana de Eva	0	https://goo.su/k1JiZD
Decoración: Objetos de riqueza	0	https://goo.su/Iljg
Decoración: Objetos de riqueza 2	0	https://goo.su/beRdx0z
Decoración: Objetos de riqueza 3	0	https://goo.su/PLUE
Decoración: Templo en ruinas Gula	0	https://goo.su/VCMk
Enemigo: Cangrejo	0	https://goo.su/shrI8Y
Enemigo: Flying Dragon	0	https://goo.su/UH9eNp
Enemigo: Libélula	0	https://goo.su/H4ms
Enemigo: Murciélago	0	https://goo.su/N3Lz8
Enemigo: Nutria	0	https://goo.su/HzxXC

Enemigo: Pollo	0	https://goo.su/WHozhji
Enemigo: Sprout	0	https://goo.su/syRv
NPC: Hombre mayor	0	https://goo.su/aUl0
NPC: personaje con cráneo y cuerpo separado	0	https://goo.su/4KJojF
Plataformas: Tile Set césped	0	https://goo.su/Oy5sZ
Plataformas: Tile Set Chocolate	0	https://goo.su/W4eS
Plataformas: Tile Set Mazmorras	0	https://goo.su/v18UwQv
Plataformas: Tile Set Volcánico	0	https://goo.su/jjHQcE
Protagonista: Caperucita	20	https://goo.su/rcEn
Rewired	44.91	https://goo.su/b7scU
Promoción del estudio de mercado en Twitter	10	https://goo.su/rfC4kBB
Promoción del estudio de mercado en Tik Tok	10	https://goo.su/qGl46Nz
Promoción del estudio de mercado en Instagram	10	https://goo.su/DS2W
Total	94.91	

1.14 Mecanismos de control de calidad

Cada vez que efectuaba ajustes significativos ene l videojuego, solía solicitar la opinión de personas de mi entorno, como si fueran potenciales compradores, para obtener una evaluación más precisa de lo realizado.

Esto se hace mucho en la industria del videojuego, aunque no siempre de una manera ética. En algunos casos, cuando se establece una fecha de lanzamiento específica para un videojuego, este se lanza para ese momento sin estar lo suficientemente probado. En consecuencia, utilizan a los jugadores que han comprado el videojuego como testers, y luego arreglan los errores encontrados con actualizaciones.

1.15 Otras referencias

Con el fin de facilitar una mayor compresión del proyecto por parte del lector, adjunto a continuación los enlaces a las páginas webs del estudio de mercado, a la hoja de cálculo donde se encuentran las traducciones de los textos del videojuego y a la carpeta de Google Drive donde se puede encontrar el diagrama de Gantt, el proyecto y el video de demostración:

- Estudio de mercado: https://forms.gle/16w9JYZwk5SwU6JU7
- Hoja de cálculo: https://goo.su/c8vz
- Carpeta Google Drive: https://goo.su/WYzQ6w

2 DISEÑO

2.1 Diseño Grafico

A pesar de mi intención de crear una narrativa atractiva, no he subestimado el valor del aspecto gráfico. Reconozco que este aspecto es esencial, ya que muchas personas priorizan este factor como aspecto principal a la hora de adquirir un juego. Como evidencia de lo comentado, se puede observar en la gráfica de mi estudio (Ilustración 11: Estudio de mercado – Primer factor en la compra de un videojuego), que un 19.1% de los participantes se fijan más en este aspecto.

Como he mencionado previamente, a pesar de que la estructura del mundo de mi obra está basada en la Divina Comedia, hasta el momento solo he podido realizar los nueve círculos del Infierno. En lo que respecta a la interfaz de usuario, mi intención

era que su aspecto variara de acuerdo al círculo en el que el jugador se encontrara, pero no ha sido posible.

Actualmente, la interfaz de usuario está asociada visualmente al círculo de la avaricia. Esta relación se puede deducir debido a la utilización de una textura que evoca al mármol y color dorado dominante. Además, en el Slider de volumen, situado en el menú de opciones, utilizo como botón de desplazamiento una moneda de oro con una animación.

Para activar la opción de pantalla completa, se dispone de un Toggle que en el caso de estar activo, mostrara un diseño de dos espadas de oro cruzadas, del famoso juego “Minecraft”.

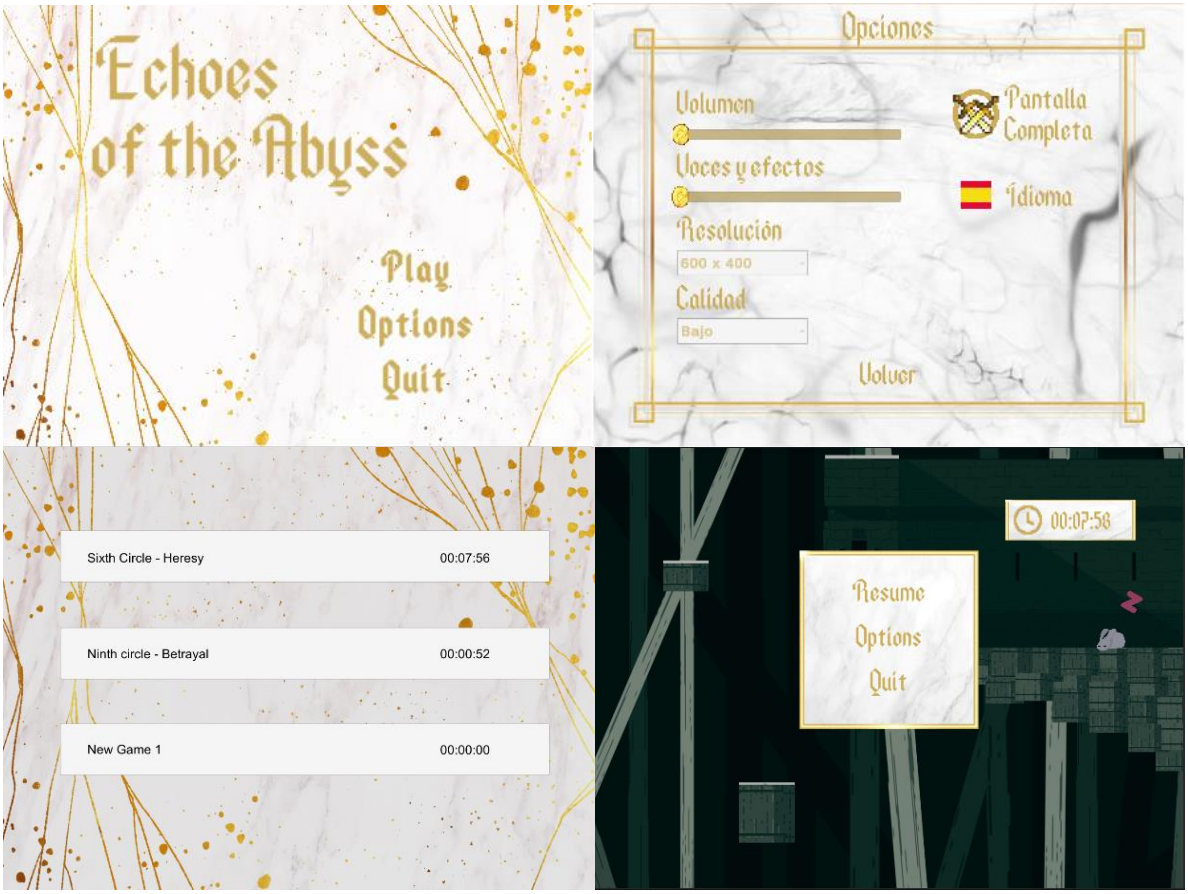


Ilustración 21: Interfaz de usuario Greed

Al igual que lo visto en la interfaz de usuario de la avaricia, he buscado seleccionar los colores más adecuados en cada contexto para transmitir la situación particular en la que nos encontramos. Como ejemplo, denotan los tonos rojos de la ira o los matices morados que evocan la lujuria

Cada uno de los círculos cuenta con referencias de alguna forma. Compartiré algunas de estas referencias, aunque he de decir que tengo un documento con todas las que quería añadir, pero por límites de tiempo no he podido añadirlas todas.

En el Limbo se albergan las almas que no han sido bautizadas y que han incurrido en grandes crímenes. El castigo en este círculo es el de estar en un estado en el que nunca podrán llegar al paraíso. Inspirado en esta idea, tenía el plan de concebir un mundo donde siempre tengan presente que nunca podrán estar ante la presencia de Cristo: una iglesia o catedral. Para ser exacto, mi diseño se basa en el Santuário São João Bosco, una iglesia de Brasilia.

En este mundo también se presentan a mentes ilustres, debido a ello, he incluido personajes referentes como Socrates.

Ilustración 22: Limbo e iglesia, Santuário São João Bosco

En el círculo de la lujuria, se pueden apreciar varios elementos cuya representación radica en ella, como son las cadenas, los espejos y las plataformas creadas tomando como base el chocolate. Mi objetivo fue tener como foco principal a los espejos, por lo que podemos presenciar que se puede interactuar con algunos de ellos narrando aspectos relacionados con su simbolismo. Los espejos presentados son:

- **Espejo griego de cobre**: Era utilizado por las brujas de Tesalia.
- **Espejo de Blancanieves**: Presente en el famoso relato.
- **Conjunto de espejos lujosos**: Utilizados en la antigua Venecia para mostrar el poder adquisitivo.
- **Espejo de obsidiana**: Para los aztecas, vinculado con el dios Tezcatlipoca, dios de la noche y las cosas materiales, protector de los esclavos.

- **Espejo roto**: Hay múltiples leyendas con ellos, considerados de buena o mal suerte dependiendo del lugar.

Ilustración 23: Circulo 2 - Lujuria

El tono predilecto de la gula es el naranja. El castigo de este círculo es una lluvia de granizo incesante de una sustancia viscosa como el petróleo, además, de ser atormentados por los ladridos de Cerbero.

Con el objetivo de implementar la lluvia mencionada, utilizo un sistema de partículas de Unity. Para lograr esto, he modificado la superposición de escena para ser visto en 2D, ya que, por defecto, trabaja sobre escenas 3D. Además, es necesario anular la velocidad de inicio y ajustar la gravedad para que estas sean generadas de arriba abajo. A continuación, he utilizado la función “shape” y “Scale” para especificar desde donde deben originarse las partículas. Aprovechando “Rate Over Time” en la sección de “Emission” podemos controlar la cantidad de gotas de lluvia generadas en un intervalo de tiempo. Por último, debemos cambiar el material para tener distinto color. En caso de querer ajustar el tamaño que ocupan todas las partículas podemos usar el parámetro “Start Life Time”.

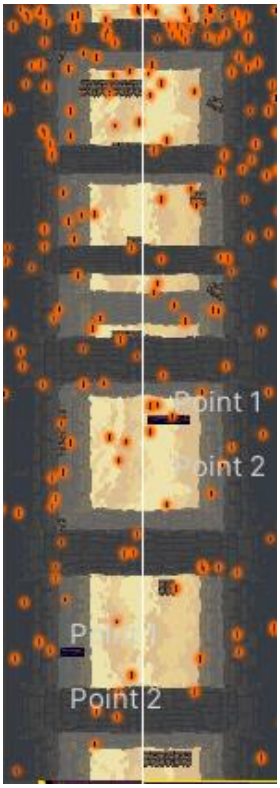


Ilustración 24: Circulo 3 - Gula

En el cuarto circulo, que es el circulo de la avaricia, se presenta una dualidad interesante. Aquí convergen individuos que acumulan riquezas en exceso junto a aquellos que las derrochan sin pensar. Este contraste se manifiesta visualmente a través de una división clara del mundo: al comienzo, se alza una montaña de abundantes riquezas, mientras por el otro lado, se extiende un terreno inhóspito y estéril.

En la montaña de riquezas, se despliegan varios elementos, incluyendo obras de arte muy conocidas. Un dragón en esta cumbre simboliza todas las historias en las que se relata como esta criatura legendaria defiende el tesoro. Además, dentro de este escenario, dos figuras humanas discuten. El individuo situado en la zona de riqueza cuestiona al otro por qué despilfarra, mientras que el segundo, ubicado en el lado más pobre, le pregunta al otro el motivo de su ahorro.

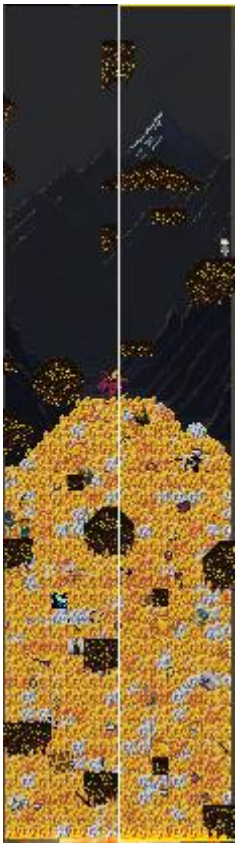


Ilustración 25: Circulo 4 - Avaricia

La ira se ve representada por tonos rojos. Simbolizada por volcanes debido a la naturaleza abrupta de ambas situaciones. Adicionalmente, la comparación entre la ira y un volcán puede verse en el proceso por el cual un volcán pasa de un estado de letargo a erupción, asociándolo con la liberación de las emociones acumuladas.

Ilustración 26: Circulo 5 - Ira

Mi objetivo en el sexto circulo fue dar la imagen de un mundo sobrio, con poca vida, lleno de tristeza debido a la condena sufrida por los herejes. Su castigo es permanecer en ataúdes ardientes.

Ilustración 27: Circulo 6 – Herejes

Dentro del séptimo circulo del infierno, se hallan aquellos que perpetraron violencia tanto hacia otros seres como hacia sí mismos. En esta última faceta es en la que me he basado principalmente. Es por esto que en este mundo se pueden divisar elementos como sogas y, al principio del mundo, las flores Lycoris Radiata se erigen como símbolo ubicado con la muerte y con las personas que han optado por el suicidio.

Además, la condena de aquellos que decidieron acabar con su vida radica en la pérdida de su esencia humana. Por este motivo, en el fondo de la escena, se presenta numerosos ojos que pestañean. Dado que se dice que son transmutado en árboles, he añadido en esta sección del mundo un enemigo que adopta la forma de un árbol con ojos que no puede hacer caso a su sentido.

Ilustración 28: Circulo 7 - Violencia

En el círculo de los fraudulentos, correspondiente con el octavo nivel, he querido dar una visión futurista donde destacan los logotipos de las principales criptomonedas. Esta representación refleja la realidad actual, donde el auge de las criptomonedas ha dado paso a una serie de negocios ilícito y fraudes. Estas

actividades delictivas se asocian con las criptomonedas debido a que no son rastreables.

El valor tan fluctuante que presentan estas monedas ha atraído a individuos que desean lucrarse de manera rápida. En respuesta a esta tendencia, han surgido varias empresas entorno a las criptomonedas cuyo objetivo es atraer a personas para que recluten a otras personas, el típico esquema de las estafas piramidales.

Debido a esto, he decidido incluir la figura del líder de una de las empresas mas famosas del tipo que he comentado. Además, la representación visual muestra como esta rodeado de heces, el cual es uno de los castigos para los fraudulentos.

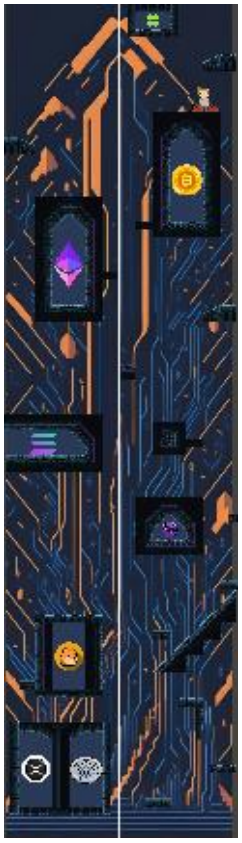


Ilustración 29: Circulo 8 - Fraudulentos

Para finalizar con los círculos, llegamos al noveno nivel conocido como el círculo de los traidores, el noveno nivel. Aquí, se encuentra Lucifer, quien con el poder del abatimiento de sus alas, congela el mundo. El ángel Lucifer, cuyo nombre significa “Portador de luz”, quería estar por encima de Dios, lo que desencadeno en una guerra

entre los ángeles. En este conflicto, su principal antagonista fue el ángel Gabriel. En un intento por prevalecer, Lucifer se convirtió en dragón, pero su intento fracaso y Gabriel, con un contundente golpe, lo mando al inframundo donde quedo eternamente clavado en un bloque. Esta condena junto con la ubicación de Lucifer, le dio nombre a este infierno.

Ilustración 30: Circulo 9 – Traidores

Mi enfoque principal se ha centrado en la conceptualización y creación de los círculos del Infierno, dado que los he desarrollado desde cero. En cuanto al diseño gráfico de otros elementos, como los personajes, he utilizado recursos previamente existentes. Sin embargo, en la mayoría de los casos, he tenido que realizar modificaciones y ajustes en dichos recursos para adaptarlos a las necesidades del juego o para garantizar que se integren de manera adecuada en la experiencia visual general. Este proceso de adaptación y personalización de los recursos ha sido fundamental para lograr una coherencia estética y una buena calidad visual del videojuego.

2.2 Diseño técnico

Tal como mencione previamente, he optado por utilizar el motor de juego Unity debido a una variedad de razones, siendo la más significativa mi familiaridad con este motor. Esto se debe a que fue el empleado durante mi participación en la asignatura “Desarrollo de videojuegos”

En lo que respecta a la programación, he elegido el lenguaje de programación C#, que es recomendado por Unity como principal opción y cuenta con una amplia gama de herramientas asociadas para facilitar el desarrollo. El entorno de programación utilizado en “Visual Studio 2019”, el cual Unity facilita su instalación como parte de su proceso de configuración.

En la ejecución de este proyecto, he empleado mi ordenador portátil personal de la marca “Asus Republic Of Gamers”, específicamente el modelo Strix G. Este dispositivo está equipado con un procesador “Intel i7-9750H”, una memoria RAM de 16 GB, una unidad de estado sólido (SSD) de 1TB y una tarjeta gráfica, “NVIDIA GeForce RTX 2060”.

3 DESARROLLO

Para hablar del contexto del desarrollo, exploraré todos los aspectos relacionados con la programación que constituyen el videojuego “Echoes of the Abyss”. Para brindar una explicación más precisa dividiré esta sección en subsecciones que reflejen los principales grupos de programación presentes en el videojuego. Adicionalmente, estas categorías serán desglosadas en los scripts implementados, dado que al fin y al cabo es donde se produce toda la acción.

3.1 Audio

3.1.1 AudioManager y LevelManager

En lo que al audio se refiere podemos encontrar estos dos Scripts principales, empezare hablando del AudioManager.

En el AudioManager disponemos de un patrón denominado Singleton, esto es un patrón de diseño que se encarga de asegurar que solamente existe una única instancia de nuestra clase, esto se realiza porque solamente queremos tener un controlador de audio en nuestro proyecto, además nos asegurarnos de no tener varios recursos innecesarios en la escena, obteniendo así un poco de optimización.

Para lograr esto, inicializamos una variable llamada “instance”, donde crearemos todo el proceso. Además, disponemos de otra variable denominada “_instance”, que cumple el papel de variable auxiliar para la instancia principal. Dentro de “instance”, realizamos una comprobación para saber si existe “_instance” previamente. En caso de que “_instance” no exista, intentaremos encontrar una instancia de la clase en la escena con “FindObjectOfType”. Si la instancia no se ve presente en la escena, la creamos manualmente; de lo contrario, agregamos dos variables de tipo “AudioSource” como hijos a la instancia. Estas variables nos servirán para controlar el audio de la música y de los efectos. Finalmente, devolvemos “_instance”.

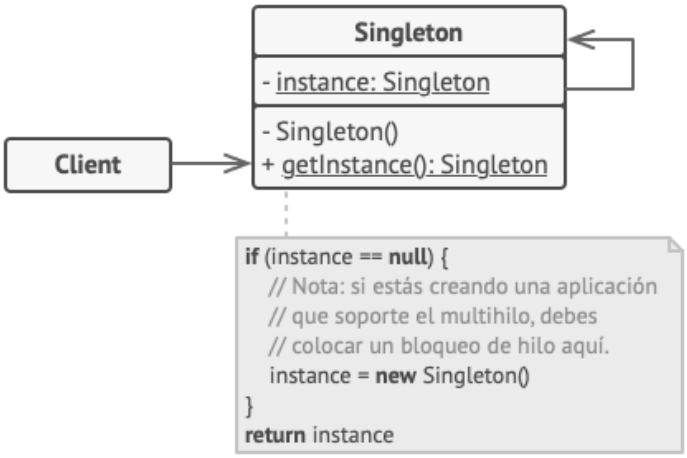


Ilustración 31: Esquema básico Singleton

Partiendo de lo comentado, si se desea invocar un método de “AudioManager”, será necesario haciendo referencia antes a “instance”, es decir, con la siguiente nomenclatura “AudioManager.instance.NombreMetoddoDeseado()”.

Tal como mencioné, es esencial contar con dos variables de tipo “AudioSource” para controlar el audio. En este sentido requerimos de dos métodos: uno para reproducir efectos y otro para la música de fondo. Como el sonido de los efectos solo necesita ser lanzado una única vez, simplemente utilizaremos el método “PlayOneShot” de la clase AudioSource con el clip que recibo como argumento.

Sin embargo, la música general se debe reproducir de manera continua en bucle. Para lograr esto, el método encargado de su reproducción (“PlayMusic()”) debe contemplar varias consideraciones. En primer lugar, comprobamos que la variable “clip” asociada al objeto “AudioSource” que representa a la música general es nula, en caso afirmativo, detenemos la reproducción con la función “Stop()”. En el caso contrario de que la variable no sea nula, debemos corroborar que es diferente al clip que recibimos como argumento, de ser así, asignaremos la nueva pista a la variable “clip”, activamos la variable “loop” para que la música se reproduzca en ciclo y lo ponemos en marcha mediante “Play()”;

En el “Update()”[12], comprobaremos que el parámetro “volume” de las variables de tipo “AudioSource” coincida con una variable homologa a ella que fue inicializada con su mismo valor. En el caso de tener distinto valor, significa que han sido utilizados los métodos de la clase creados con el objetivo de cambiar el volumen, por lo que tendremos que cambiar el volumen del AudioSource asociado.

Para finalizar con la parte de la música, comentar que la única función que presenta el “LevelManager” es la de llamar a la función “PlayMusic” de la clase “AudioManager” con un clip que se le haya proporcionado.

3.2 Cámara

3.2.1 CameraController1

Cuando se trata de las cámaras en los videojuegos en 2D, tenemos que mencionar la “Parallax Cámara”. Esta técnica ampliamente utilizada en la mayoría de juegos 2D con el objetivo de crear una ilusión de profundidad provocando que los objetos más distantes se muevan a una velocidad menor que los más cercanos al personaje principal. A pesar de que la implemente en un comienzo, he considerado que esta técnica no es útil en mi tipo de videojuego por múltiples razones.

En primer lugar, esta técnica es mas adecuada para un enfoque horizontal donde el jugador de desplace de forma lateral. En mi juego, donde el enfoque es vertical, podría no ser coherente este tipo de cámara

Además, el efecto visual que genera la técnica en cuestión, puede provocar que el jugador desvié la atención de su objetivo principal, ajustar de forma meticulosa la potencia y dirección del salto para conseguir llegar a la siguiente plataforma, originando una frustración adicional a la que genera la dificultad del videojuego.

[12] Función que es llamada de forma automática por el sistema cada frame del juego, es decir, si el juego está a 60 fps, la función será llamada 60 veces en un segundo.

Basándome en las razones introducidas anteriormente, opte por cambiar el tipo de técnica respecto a la cámara. En el nuevo enfoque, el mundo se divide en secciones cuyo tamaño es perfectamente divisible por el tamaño de la cámara. Esto genera que la cámara no sigue al jugador de forma automática, sino que se sitúa en una posición fija dependiendo donde este el personaje.

Este enfoque me permite añadir un poco más de dificultad al videojuego, ya que el jugador se encontrará en situaciones donde deberá hacer un salto de prevención antes de dirigirse a la siguiente a la siguiente plataforma. Si no lo hace perderá la referencia visual de donde aterrizar ocasionando una posible caída.

Para proporcionar una comprensión más clara de la nueva dinámica y sus consecuencias en la práctica, muestro las siguientes imágenes del videojuego:

Ilustración 32: Ejemplo enfoque actual de la cámara

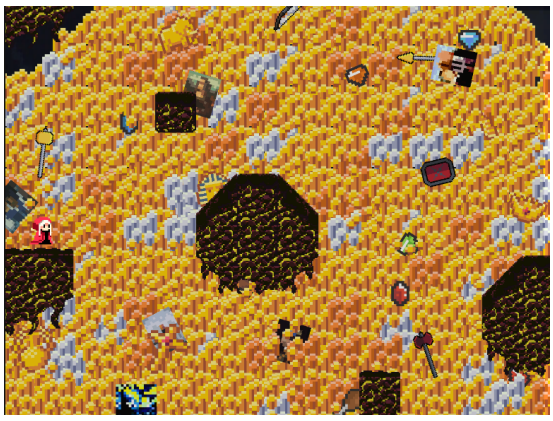


Ilustración 33: Cámara sigue al jugador

Al analizar la Ilustración 32, la imagen del lado izquierdo representa el estado anterior a la imagen de la izquierda, con la configuración actual de la cámara. Dicho esto, se puede observar que si no se realiza un salto preventivo puede ocasionar que el jugador, de manera accidental, choque contra la zona azul que he señalado en la imagen de la izquierda originando una posible caída. Si la cámara siguiera al jugador, la representación sería similar a lo que se contempla en la Ilustración 33, el jugador ya estaría advertido sobre la potencial amenazada reduciendo la emoción y el factor sorpresa del videojuego.

Para realiza esto, en el script “CameraController1”, inicio localizando un elemento de la escena con “FindGameObjectWithTag”; esta búsqueda se realiza en función de la etiqueta que recibo. Una vez localizado, dentro de la función “Update()”, calculo dos variables que hacen referencia al límite superior y al límite inferior de la pantalla en la que se encuentra el personaje.

Cuando el jugador, representado por el objeto que concuerda con la etiqueta, se pasa del límite superior, la cámara se desplaza hacia la sección de arriba. Si por el contrario excede el límite inferior de la pantalla, la cámara se desplaza hacia abajo. También se toma en consideración que cuando hacemos estos movimientos de cámara, no se pase de unos puntos que sirven como frontera a la superficie que queda fuera del mapa del videojuego.

3.3 Tiempo

3.3.1 TimeController

En este script, simplemente realizo un cálculo del tiempo que ha pasado desde el inicio del juego. Para lograrlo, empleo la función “Time.deltaTime” sumándola a una variable específica, inicializada a cero, para obtener el tiempo total transcurrido en segundos. Posteriormente divido esta variable para obtener las horas, minutos y segundos correspondientes. A continuación, transfiero la información a un GameObject de tipo Text, que está asociado al canvas de la escena en Unity, consiguiendo así que la información sea visible mientras el juego está en ejecución. Para lograr el formato deseado de “00:00:00”, utilizo la función “String.Format”; primero específico el formato deseado y, a continuación, incorporo las variables que deseo sustituir, en mi caso, las horas, minutos y segundo, en ese orden.

Es relevante mencionar que más adelante, cuando hable del sistema de guardado, volveré a hacer referencia a este script ya que influye en ese campo.

3.4 Comprobador de colisión

3.4.1 LayerChecker

Esta clase la utilizo de ayuda para varias partes específicas del videojuego en las que necesito saber si he detectado que el objeto que deseaba ha pasado por un sitio especifico o se encuentra dentro de un espacio concreto.

Con este objetivo, en el script, recibo por referencia desde el editor la elección de darle forma de circulo, línea o cubo a la zona que quiero verificar, y también especifico la capa del objeto con el que necesito comprobar si ha existido una colisión. En el “Update()”, empleo funciones específicas proporcionadas por Unity y pertenecientes a la clase “Physics2D”. Estas funciones me devuelven “true”, si el objeto de la capa especifica ha colisionado:

- Para formar una línea, utilizo “Physics2D.Raycast”, el cual traza un rayo. Este método toma como argumentos la posición de origen, la dirección del rayo, la distancia y la capa del objeto objetivo.
- Si quiero comprobar si existe colisión en una zona con forma de circulo, utilizo “Physics2D.OverlapCircle”. Esta función simula un círculo y requiere la posición del centro, el radio y la capa del objeto objetivo.
- Para la forma de cubo, empleo “Physics2D.OverlapBox”, que crea una caja de colisión. Los argumentos necesarios son la posición de partida, las dimensiones (ancho y alto, y la profundidad si estuviéramos en 3D) del cubo, el ángulo de rotación y la capa de objeto objetivo.

Establezco una variable llamada “isTouching” asignada a la detección de colisiones ya mencionada, convirtiéndose esta variable en un indicador de colisión. Una vez conseguido el indicador, puedo realizar acciones adicionales tomando como referencia el valor de la variable, “isTouching”.

3.5 Puntos guía

3.5.1 WayPointsManager, FollowWayPoints

De manera similar a “LayerChecker”, estas clases se emplean como recursos auxiliares y tienen la función de permitir que un objeto se desplace entre diferentes puntos fijos.

Para realizar este cometido, en primera instancia, en el editor de Unity, creamos GameObject dentro del objeto que deseamos que se desplace. A este GameObject le asignaremos el componente “WayPointsManager”. Posteriormente, creamos hijos dentro de este GameObject, y las posiciones de estos hijos determinaran los puntos entre los cuales nuestro objeto se moverá.

Dentro de “WayPointsManager”, se pueden identificar dos métodos, “GetNextPoint” y “GetRandomPoint”. En “GetNextPoint”, aprovechando un índice, recupero la posición del siguiente hijo y la devuelvo. Si el numero de hijos es menor que el valor del índice, el índice se reinicia a cero, iniciando así, de nuevo, el ciclo entre los hijos. Por otro lado, en GetRandomPoint, el índice se genera aleatoriamente entre cero y el numero de hijos disponibles, consiguiendo una posición aleatoria entre todas las que presentan los hijos.

La esencia central en “FollowWayPoints” radica en el hecho de que el objeto se desplazara hacia el waypoint actual siempre cuando la distancia y su posición actual sea superior a 0.1. Cuando el objeto está suficientemente cerca, actualizamos la posición actual con el método “GetNextPoint” ubicado en “WayPointsManager”, permitiéndole avanzar hacia el siguiente punto preestablecido. Aunque este ultimo script en un principio fue desarrollado para facilitar el desarrollo de objetos móviles, en la práctica hay ocasiones donde me ha resultado más cómodo llamar directamente a “WayPointsManager” en el script donde realizo la lógica del objeto.

3.6 Personaje Principal

3.6.1 CharacterController1, SwordController

Caperucita dispone de tres estados distintos: normal, blanco y oscuro. En la concepción original, se planifico que en el modo blanco tendría la capacidad de saltar mas alto, aunque su poder de ataque sería más reducido. En contraste, en el modo oscuro tendría una menor potencia de salto, pero sus ataques serán mas contundentes. Además, iban a existir zonas donde solo podías estar de un color o vendrían unos personajes a echarte de ese entorno. No obstante, debido a limitaciones temporales, estas funcionalidades no han sido implementada aun, aunque si he utilizado esta mecánica para una plataforma de la que hablare a posteriori.

Ilustración 34: Versiones de Caperucita

Dado que Caperucita es el primer personaje mencionado en la memoria, explicare en esta sección cómo funcionan las animaciones asociadas a ella.

Las animaciones 2D en Unity funcionan como una secuencia de imágenes. Por lo tanto, al iniciar con la animación, es esencial preparar estas imágenes previamente, teniendo en cuenta varios puntos si queremos un resultado óptimo:

- La imagen puede mostrar borrosidad debido a que Unity aplica compresión de texturas y un filtro a los pixeles por defecto. Para resolver este problema, nos dirigimos a las configuraciones del sprite correspondiente en Unity. En la opción “Filter Mode”, seleccionamos “No Filter”, y en la opción “Compression”, seleccionamos “None”.

- Cada sprite tiene un punto de origen desde el cual se realizan todas las transformaciones de la imagen, llamado pivote. Cuando creamos una animación, el pivote debe encontrarse en una posición que tenga sentido en relación con la imagen previa, asegurando de esta forma una adecuada transición entre ambas. Por lo general, los diseñadores que facilitan los recursos toman en cuenta esto para no complicar el proceso al desarrollador. Sin embargo, en los casos que no se tiene en cuenta, esta situación puede volverse una tarea extremadamente laboriosa para el desarrollador.

Cuando ya nos hemos asegurado de tener las imágenes bien preparadas, es momento de crear la animación. Mientras tenemos seleccionado el GameObject donde se encuentra el Sprite, abrimos la pestaña “Animation” y procedemos a crear una nueva animación. Después de elegir la ubicación donde deseamos guardar esta animación, se abre una línea de tiempo, dentro de la misma pestaña. Aquí tenemos la oportunidad de disponer las imágenes, provocando la sustitución del actual mostrado en el tiempo donde haya sido colocada la nueva imagen. En otras palabras, cuanto más cercanas coloquemos las imágenes en la línea de tiempo, más fluida será la animación.

Un mismo Sprite tiene la capacidad de albergar varias animaciones. Para visualizar todas las animaciones que contiene nos dirigimos a la ventana “Animator”. En la siguiente imagen, se muestran todas las animaciones asociadas a Caperucita.

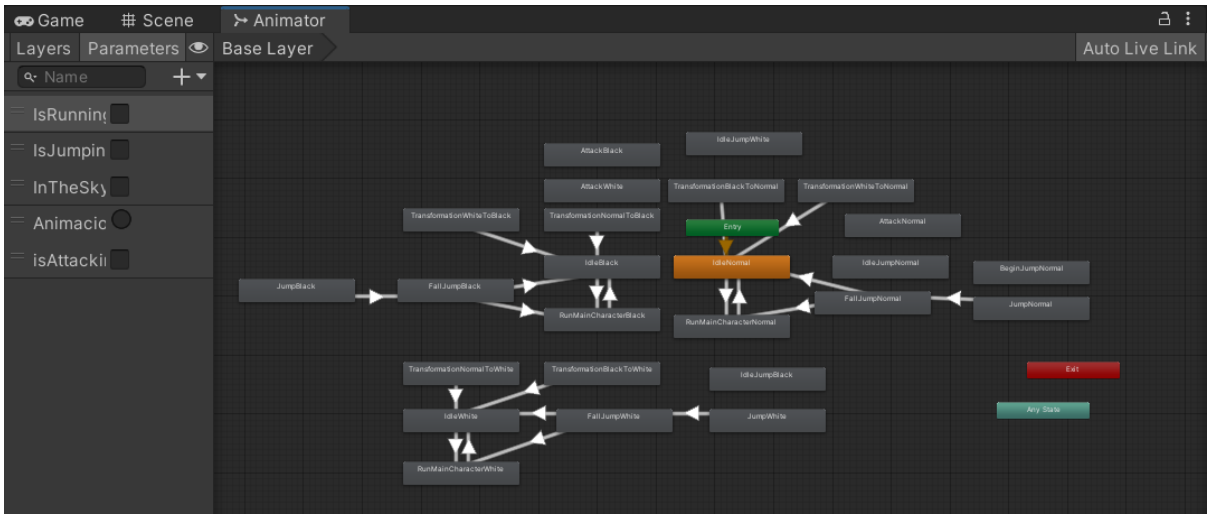


Ilustración 35: Animator de Caperucita

Si analizamos un poco la imagen anterior, se divisan nodos con diferentes colores:

- **Verde (Entry)**: Nodo de entrada, es con el que se comienza el grafo de animaciones.
- **Naranja (Idle)**: Es la animación principal, la que será ejecutada por defecto.
- **Azul (Any State)**: Desde este nodo podemos reproducir animaciones independientemente del nodo en el que nos encontremos.
- **Rojo (Exit)**: Se puede utilizar si se desea reproducir animaciones cada vez que salimos de un nodo.

Ahora, sumergiéndonos más en el aspecto de la programación, refiriéndome a la misma ilustración, hay unas flechas que especifican el sentido que tienen las animaciones. Aunque estas flechas pueden ir sin condiciones adicionales, es decir, cuando termina una animación se lanza la siguiente, también pueden llevar condiciones y en este aspecto, entra la columna “Parameters” que se puede ver a la izquierda. Estos parámetros, de tipo boolean, pueden ser invocados desde el código pudiendo cambiar su valor. Además, disponemos del comando “Animator.Play” para realizar cualquier animación.

Para realizar el salto, es necesario que el personaje esté tocando el suelo, y para controlar esto, utilizo “Physics2D.OverlapBox”, ya mencionado anteriormente. En los videojuegos comunes, el salto no se regula dependiendo de cuanto tiempo se deje mantenido el pulsador, ya que hay que tomar varios factores en cuenta. Sin entrar en demasiados detalles, la idea es que, mientras se pulsa el botón de salto, una variable ira aumentando su valor de forma progresiva. Esta variable se suma al vector hacia arriba del RigidBody de caperucita para efectuar el salto. Aunque, hay que establecer un límite en el aumento de la variable para evitar saltos infinitos. En contraposición, se implementa una fuerza de salto límite inferior para que todas las animaciones que forman parte de la dinámica del salto se ejecuten sin inconvenientes.

Para garantizar la ejecución adecuada de diversas animaciones, empleo corutinas, que son funciones invocables que me permiten parar la ejecución durante un tiempo determinado. Esto se logra con la sintaxis “yield return new WaitForSeconds(TiempoEnSegundos)”. Las corutinas podrán ser observadas, a lo largo del código del proyecto, siendo empleadas en múltiples propósitos.

Dentro del mismo script, existen cuatro métodos más a destacar, comenzare describiendo “handleControls()” y “handleFlip()”. En “handleControls()” simplemente se determina la dirección en la que el personaje se está moviendo en el eje horizontal mediante “Input.GetAxisRaw("Horizontal")”. Por otro lado, en “handleFlip()”, verificamos hacía que dirección se dirige el personaje y rotamos su eje X, en el caso de que haya cambiado la orientación.

Para abordar “handleAttack()”, he de decir que cree una interfaz con el objetivo de que todos los personajes con la posibilidad de recibir daño implementaran la función “TakeDamage”. En esencia, en “handleAttack()” se invoca al método “Attack()” de la clase “SwordController”. Esta clase trabaja de la siguiente manera: se recibe un colisionador como argumento, el cual representa la zona de impacto de nuestra espada. Si colisiona con otro colisionador, se activa la función proporcionada por Unity, “OnTriggerEnter2D”, recibiendo como argumento el colisionador con el que hemos colisionado. A partir de lo ya mencionado, que todos los personajes cuentan con la función “TakeDamage” para recibir daño, buscamos el componente asociado a este colisionador y llamamos a su “TakeDamage” con un valor de daño específico.

Acabo esta sección, comentando que, por el estilo de juego implementado, no tiene sentido que a caperucita se le resten puntos de salud (implementado inicialmente). En su lugar, en su función “TakeDamage”, recibe una fuerza distinta según el elemento que le hace daño, resultando en un empuje más fuerte o más suave, causando una potencial caída.

3.7 Enemigos

3.7.1 Elementos Comunes: DamageController, DamageFeedbackEffect

En mi videojuego, los enemigos constituyen un elemento que se desvían de los cánones tradicionales de los juegos, ya que no pueden infligir daño directo al personaje principal, sino que tienen la capacidad de empujarlo provocando que descienda en el mundo. Son un tipo de inteligencia artificial construida a través de unas reglas específicas, rigiéndose principalmente por una maquina de estados. Una maquina de estados, al igual que en lógica, es un modelo donde tenemos varias situaciones (estados) que realizan transiciones entre si y dependen de los casos anteriores. Este sistema me permite segmentar de forma clara las acciones del personaje permitiéndome tener más estructurado. Cada estado se vinculará a una función única que se activará cuando una variable en particular, de tipo enumerado, adopte el valor del estado correspondiente. En un comienzo, defino todos los posibles estados en el enumerado.

DamageController es un script presente en todos los enemigos ya que es el encargado de realizar el daño por contacto. En términos simples, cuando el enemigo choca con el personaje, se activa el evento “OnTriggerEnter2D” y se llama al “TakeDamage” de Caperucita.

A diferencia del anterior script, DamageFeedbackEffect, solo esta incorporado en los personajes que no poseen una animación propia de recibir daño. Primero, se crea un material de color blanco y se empleara la propiedad interna de los materiales denominada “Flash Amount”, la cual regula la visibilidad del material en pantalla. Una vez que el material se asigna a la opción “Material” del Sprite Renderer donde se ubica el Sprite del personaje a realizar el efecto, el script maneja el valor de flash amount del material alterando entre los valores máximo y mínimo en intervalos de tiempo, simulando el efecto de un parpadeo.

Si bien he incorporado esta sección en esta parte de la memoria porque para los enemigos se utilizan varios elementos ya mencionados, comentare como se utiliza cada elemento en los diferentes tipos de enemigos.

3.7.2 BatControllerFinal

El murciélago presenta como estados: “Inactive”, “Patrol” y “ChasePlayer”. Mientras no colisione con la cámara permanece en estado inactivo, este estado será igual en todos los enemigos. Una vez ocurre la colisión, cambia el estado a “Patrol”, y empieza a patrullar, moviéndose entre una serie de waypoints prefijados. Si el jugador entra dentro de un LayerChecker circular, que establezco como la visión del murciélago, lo ataca dirigiéndose hacia el. Cuando el jugador sale de su visión vuelve a patrullar.

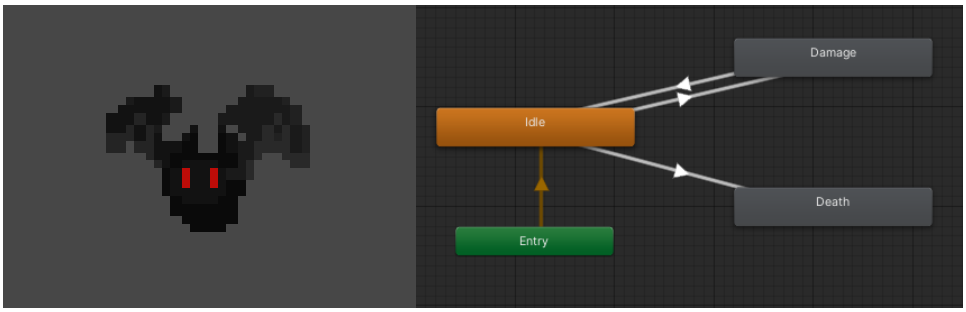


Ilustración 36: Murciélago y sus animaciones

3.7.3 CrabController

El cangrejo tiene como estados: “Inactive”, “Wait”, “Attack”, “Walk” y “Turn”. Si el jugador entra en alguno de los dos layer checker, en forma cubo, que tiene a izquierda y derecha el cangrejo, empieza a moverse en su dirección (“Walk”) y a partir de aquí tenemos dos situaciones. En la primera situación, si el cangrejo deja de sentir suelo debajo de él, significa que ya no puede seguir y da la vuelta, sería el estado “Turn”. En la otra situación, tenemos la posibilidad de que el jugador este tan cerca del cangrejo que se encuentre dentro de su rango de ataque, pasando al estado “Attack”. Una vez que ataca permanece quieto, en posición de alerta.

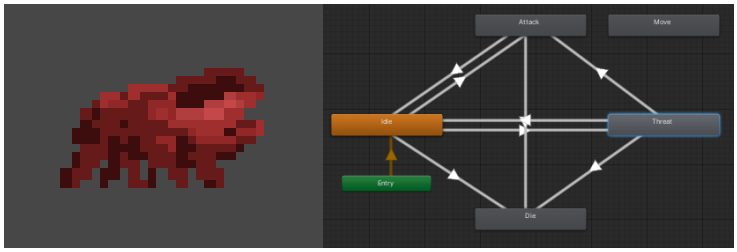


Ilustración 37: Cangrejo y sus animaciones

3.7.4 CuteChickenController

Los estados del pollo son: “Inactive”, “Idle”, “Walk” y “Turn”. Respecto a este enemigo, comentar que como se puede ver en el “Animator” no tiene animación de muerte, en los enemigos que pasa esto, cuando baja su vida a cero llamo a un prefab, una animación de fuego, con la función “Instantiate” pasándole como argumento la referencia del prefab, la posición que tendrá y el ángulo. Respecto a los estados, no tengo nada que añadir, ya que han sido comentados anteriormente.

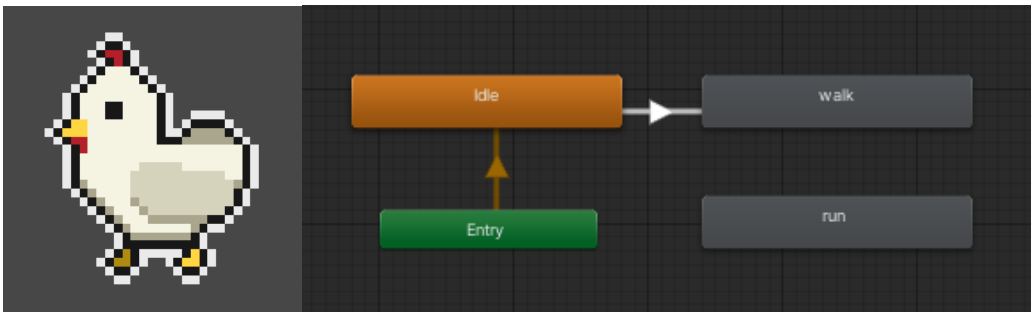


Ilustración 38: Pollo y sus animaciones

3.7.5 DragonFlyController

Las libélulas solo tienen dos estados: “Inactive” y “Patrol”. Dado el movimiento más impredecible de las libélulas, se han incluido una mayor cantidad de waypoints para su patrón de movimiento. Además, como la libélula cuenta con una animación de muerte como si yaciera en el suelo, cuando su vida baja a cero, desciende hasta el suelo y, tras un intervalo de tiempo, desaparece.

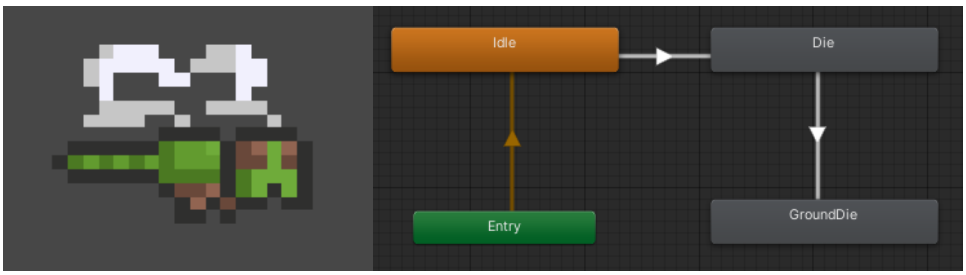


Ilustración 39: Libélula y sus animaciones

3.7.6 OtterController

Los estados de la nutria son: “Inactive”, “Silence”, “Walk” y “Turn”. Funciona de manera similar a “CrabController”, no obstante, en este caso, cuando la nutria pierde contacto con el suelo o colisiona con el jugador, realiza una animación con la cual indica que se guarde silencio y vuelve a dormir como al principio, el estado “Silence”.

Ilustración 40: Nutria y sus animaciones

3.7.7 FlyingDragonController

Este enemigo solo presenta dos estados: “Monitorear” y “Explosion”. El estado “Monitorear” se activa cuando Caperucita entra en su campo de visión, momento en el cual comienza a mirar dirección a ella. Si el personaje cruza un límite, el enemigo se abalanza hacia ella, y al chocar contra el suelo realiza una animación de explosión. En este momento, se invoca al método “Attack” de la clase “SwordController” provocando daño en el área de la explosión.

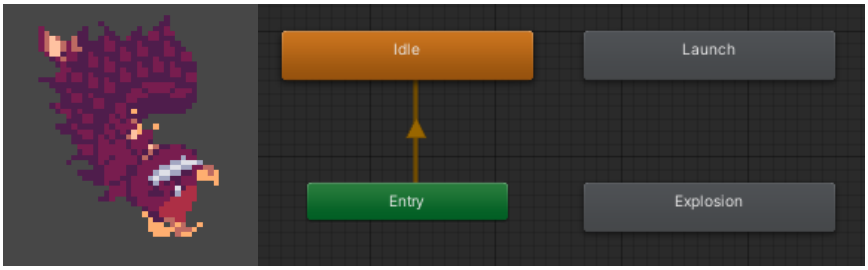


Ilustración 41: Flying Dragon y sus animaciones

3.7.8 SproutController

Los estados que presentan el enemigo Sprout son: “Inactive”, “Idle”, “Walk”, “Turn” y “ChasePlayer”. La diferencia con el cangrejo es que una vez que detecta al personaje principal no para de perseguirlo, independientemente de si ya le ha atacado.

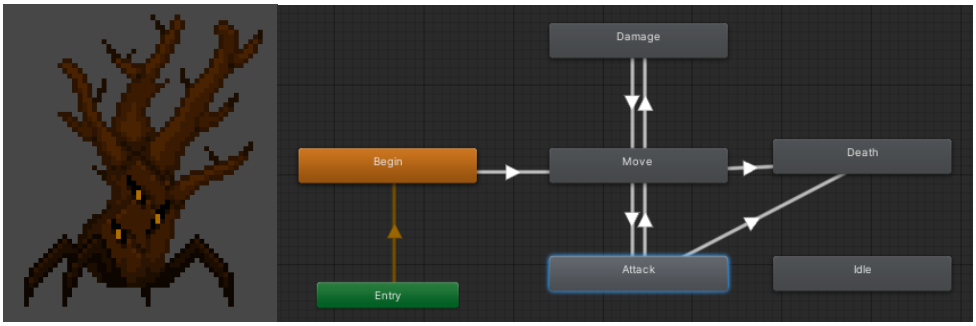


Ilustración 42: Sprout y sus animaciones

3.8 Decoraciones

3.8.1 ChandelierController, DragonKnightController, SkulController, CrownController

En cuanto elementos decorativos como la ballena, el gato negro y el fósil, no requieren script, ya que ejecutan animaciones de manera continua. Para los elementos Dragon Knight, Skull y Crown, una vez que se pasa cierto limite, se genera un numero aleatorio en cada frame y dependiendo de este valor se reproduce una animación especifica, controlando que termine la animación actual sin ser interrumpida por otra. En el caso del candelabro (Chandelier), mantiene una animación de forma constante, pero si lo atacas, las velas se apagan. Originalmente esta mecánica fue pensada para forma parte de un puzzle en el videojuego.

3.9 Sistema de guardado

3.9.1 GameData, SaveSystem

El sistema de guardado se erige como uno de los pilares mas fundamentales en mi videojuego, ya que, debido a su alta dificultad, sin él perderíamos todo el progreso una vez que el juego se cerrara. Cabe destacar que considere la posibilidad de implementar un modo de juego difícil en el cual se eliminara el sistema de guardado. Para atender a las necesidades de los speedrunners, quienes compiten por completar el juego en el menor tiempo posible, incorpores tres slots o ranuras de guardado. De esta forma, los jugadores pueden guardar en diferentes puntos del juego y regresar a ellos en otro momento, lo que les resulta útil para poder practicar secciones específicas del videojuego.

La clase “GameData” funciona como una definición de una instancia para una base de datos interna. Solo dispone de un constructor parametrizado donde guarda todos los datos que queremos que sean almacenados antes de cerrar el videojuego. En mi caso, los datos que estoy preservando son la posición del personaje principal y el tiempo transcurrido desde el comienzo de la partida.

Dentro del script “SaveSystem” se realizan las funciones que permitirán poder guardar y cargar. Los datos serán almacenados en archivos cuya ruta parte del directorio donde se ubican los datos persistentes del videojuego (“Application.persistentDataPath”). Se inicia creando un fichero en la ruta anterior utilizando “FileStream”. Luego, se genera una instancia de “GameData” con los datos que recibimos y, mediante un formateador binario, creado previamente, se agrega la instancia al fichero. Este método garantiza que los datos estén encriptados en binario, evitando que sean legibles si alguien intenta acceder al archivo. Para concluir, cerramos el fichero con la función “Close()” de “FileStream”, para forzar que la información haya sido transferida y guardada en el fichero de forma satisfactoria.

En “LoadPlayer” se puede ver como el proceso es similar, pero en sentido inverso. Aquí, recibimos una ruta del fichero a obtener los datos, descodificamos los datos para acceder a su contenido y devolvemos un objeto de tipo “GameData” con ellos. Esto se lleva a cabo si existe un archivo en la ruta especificada. Si no se encuentra ningún archivo, devuelve un valor “null”. Este último caso, significaría que se trata de una nueva partida, lo que desencadenaría en la activación del panel que he diseñado para cuando se comienza una nueva partida.

La llamada de los métodos de “SaveSystem” se deben realizar desde los scripts que implementan la lógica asociada a elementos que quieran ser guardados. Sin embargo, surge un inconveniente ya que “SavePlayer” requiere todos los elementos a ser guardados como argumento. Para resolver esto, nos apoyamos en “GameManager”.

“GameManager” es un script que actúa como administrador global de todos los elementos que forman parte del videojuego. Cuando se desea guardar la partida, “GameManager” recopila los elementos relevantes de lo que guardar datos e invoca a “SavePlayer”. De forma análoga, se hace igual para cargar la partida, pero proporcionando la ruta deseada y recibiendo los datos que previamente fueron guardados. Estos datos son entonces distribuidos hacia los elementos correspondientes por “GameManager”.

La comunicación del “GameManager” con el resto de componentes se realiza a través de eventos. Los eventos operan de una manera que, desde el comienzo de ejecución del juego, se vinculan con funciones específicas de otros scripts distintos a donde fueron creados. Cuando sea necesario, podemos activar estas funciones a través de la invocación del evento. Esto proporciona un nivel más elevado de organización en el codico y una disminución de los recursos del dispositivo, ya que no tenemos que tener una referencia directa a la clase que posee la función que queremos utilizar.

3.10 Localización

La internacionalización y localización son elementos importantes para obtener una mayor audiencia. La localización implica ajustar el videojuego para ser comprensible en varias regiones, en mi caso, estará más orientado a la traducción de textos y diálogos. Por otro lado, la internacionalización se basa en el hecho de adaptar el videojuego a diferentes regiones. Por ejemplo, algunos países prohíben la representación de sangre real en los videojuegos. En mi juego no he visto necesaria la internacionalización.

Para facilitar el cambio de idioma, me apoyo en un recurso obtenido de la Unity Asset Store. Para llevar a cabo la implementación empleare diversas clases provistas por este recurso: “LocalizationManager”, “LocalizationSync”, “LocalizedText”, “LocalizedDropdow”. Además, he creado una clase propia llamada “MultiLanguageScript” para completar esta funcionalidad.

Antes de nada, es imprescindible crear una hoja de cálculo en Google Sheet. Esta hoja actuará como el repositorio central que contendrá el texto en sus distintas versiones lingüísticas. Cada fila de esta hoja representara el mismo fragmento de texto traducido a diferentes idiomas. la primera columna desempeñara el papel de identificador para invocar al texto correspondiente, mientras que las columnas siguientes representaran cada uno de los idiomas. Así pues, en la primera fila de esta hoja, debemos incluir los nombres de las claves de identificación en la columna inicial, seguidos por los nombres de los distintos idiomas en las columnas siguientes. Es importante cambiar los permisos de la tabla a editable por cualquier persona que posea el enlace.

1	Key	English	Spanish
2	GameMenu.NewGame1	New Game 1	Nuevo Juego 1
3	InitialPanel.ScreenOne	Welcome, today I will reveal a story yc	Bienvenido, hoy te revela
4	InitialPanel.ScreenTwo	First of all, excuse my manners, could	Antes de nada, perdona
5	InitialPanel.ScreenThree	Nice to meet you, NombreDelJugador	Encantado de conocerte

Ilustración 43: Hoja de cálculo del texto traducido

Para establecer la conexión entre la tabla y Unity, se emplea el componente “LocalizationSync”. Se genera un GameObject en Unity al que se agrega dicho script. Se suministran como parámetros el nombre de la hoja en uso y el id de la tabla. Este último puede encontrarse en el enlace mismo, siendo una cadena de caracteres alfanuméricos ocupando un espacio después de “/d/”. Para dar por sincronizada la tabla, terminaremos pulsando en “Sync”.

Ahora, para modificar el texto deseado, agrego el script denominado “LocalizedText” o “Localized Dropdown”, depende del tipo de elemento al que desee realizar el cambio de idioma. En el atributo “Localization Key”, introduzco la clave del fragmento de texto correspondiente, localizado en la primera columna de la tabla. De esta manera, el sistema para traducir ya estaría en funcionamiento, pero tendrá un idioma fijo correspondiente con el parámetro “Language” ubicado en “Localization Manager”. El nombre de este idioma debe coincidir con el nombre de su columna.

Cabe señalar, que para realizar este proceso, el texto en cuestión no puede ser de tipo “TextMeshPro”, ya que puede provocar un error de compatibilidad. Debido a esto, me vi en la obligación de cambiarlos por “Text” en mi proyecto.

El script “MultiLanguageScript” tiene como función alternar entre los distintos idiomas registrados en la hoja de cálculo. Para ello, comprueba el idioma del sistema a través de “Application.systemLanguage”. En función de este idioma, se ajusta el parámetro “Language” al nombre correspondiente presente en la hoja de cálculo de Google Sheet. Actualmente, solo se presentan dos opciones de idiomas, “English” y “Spanish”. Por tanto, para ejecutar el cambio empleo un toggle con las banderas de los respectivos países. Estas banderas cambian dinámicamente según el idioma en uso en ese momento.

3.11 Interfaz de Usuario (UI)

3.11.1 PauseMenu, MainMenu

Las escenas de un videojuego son los espacios donde se desarrolla todo el diseño y estructura de un mundo. Puede variar desde la creación de niveles al desarrollo de menús y opciones.

Dentro de mi videojuego, se distinguen dos escenas. Una de ellas aloja el menú principal que emerge al iniciar el juego mientras que en la otra se desarrolla el videojuego. Dicho esto, se puede intuir que el botón de “Play” del menú principal y el botón de “Salir” del menú de pausa siguen la misma lógica, ya que en esencia desencadenan un cambio de escena. Para llevas a cabo la transición a una escena especifica, se recurre al uso de “Scenemanager.LoadScene(“NombreEscena”)”. Un aspecto a tener en cuento es asegurarnos que la escena se encuentre en la opción “Scenes in Build” dentro de las configuraciones de “BuildSetting”. “Salir” también guarda la partida.

En ambos menús, al hacer clic en el botón de “Opciones”, se despliega un panel que corresponde con el menú de configuración. La diferencia más marcada entre estos dos menús radica en los botones de acción: el botón de “Salir” del menú principal y el botón de “Resume” del menú de pausa.

Cuando se selecciona “Salir” del menú principal, la intención es cerrar el juego, para lo cual se emplea la función “Application.Quit()”. Respecto al menú de pausa, este menú surge al presionar la tecla “Escape”, deteniendo el tiempo de juego para que el jugador no se pueda mover. Cuando pulsamos “Resume”, el juego vuelve a su flujo temporal normal, simultáneamente desactivamos el panel del menú de pausa.

3.11.2 OptionMenu, CanvasResolutionController

Este script tendrá toda la lógica relacionada con las operaciones que se realizan en el menú de opciones. Para cambiar el volumen simplemente se llaman a los métodos del AudioManager ya comentados con el valor que me aporta un slider.

Para definir la calidad visual, el primer paso consiste en determinar cuantos niveles de calidad deseamos presentar en nuestro juego. Para ello accedemos a la siguiente ubicación: “Edit – Project Settings - Quality”. Unity maneja estos niveles de calidad de manera similar a un vector, permitiéndonos hacer referencia a cada uno mediante el uso de “QualitySettings.SetQualityLevel()” con un índice. Este índice será aportado de forma dinámica a través del DropDown encargado de cambiar la resolución.

Respecto el cambio de resolución es una funcionalidad mas compleja donde pueden salir varios inconvenientes. En un comienzo, desarrollé un script donde se pudiera cambiar entre todas las resoluciones que permitiera el dispositivo donde se ejecute el videojuego. Sin embargo, esto no es posible ya que dependiendo de la resolución tenía distintos anchos y altos de pantalla respecto a cómo fue diseñado el videojuego de forma original. Esto provocaba que la cámara mostrara partes de la escena que no deberían ser visibles de forma habitual.

Para solucionar esto nos debemos basar en el “aspect ratio” [13], garantizando el mismo valor en todas las resoluciones. En mi caso particular, el valor de esta relación de aspecto es de 1.33. De esta forma, la escena conserva su composición visual independientemente de la resolución utilizada.

Empleo “Screen.Resolution” para filtrarlas posteriormente para solo quedarme con las del aspect ratio mencionado. En paralelo, compruebo si la resolución de la pantalla actual (Screen.currentResolution) coincide con alguna resolución de las que

[13] Relación que existe entre el ancho y alto de una pantalla

estoy añadiendo en el DropDown, en ese caso, cambio el texto que se ve por defecto en el dropdown a la resolución actual.

Con todas las resoluciones posibles en el Dropdown, le asigno una función de forma dinámica, donde obtengo el texto que ha sido seleccionado. Dicho texto tiene un formato del tipo “width x height”, por lo que utilizo “Split(“x”)” para dividirlo en dos partes. A continuación, convierto el ancho y el alto en valores enteros (int.TryParse) eliminando los espacios en blanco con “Trim”. En este punto, simplemente utilizo “Screen.SetResolution” pasándole el nuevo ancho y alto de la resolución deseada.

Aunque ya podemos cambiar la resolución, existe un problema, y es que los elementos que forman parte del canvas no se ajustan de forma adecuada a las nuevas resoluciones. Para solucionar esto, Unity proporciona la herramienta “anchor” que son como una especie de flechas que nos permiten indicar como se sitúa cada elemento de la interfaz de usuario en el canvas, pero la utilización de anchor es muy tediosa provocando que no se consigan los resultados esperados.

Por el motivo indicado, considere crear un script para mover los elementos que producían problemas. Sin embargo, me di cuenta que el problema radicaba en que esos elementos mantenían las medidas de la resolución anterior. Como solución, en el nuevo script creado, ajuste un parámetro del canvas denominado “Canvas Scaler”. Este ajuste permitió escalar todos los elementos de manera uniforme hasta alcanzar el resultado deseado.

3.11.3 CheckData

En el menú de juego, donde se sitúan todas las ranuras de partidas guardadas, muestro un texto con el circulo del infierno y el tiempo transcurrido de la partida cuando el jugador la guarda. Para hacer esto, en “CheckData”, decodifico el fichero asociado a cada ranura con un procedimiento similar al visto en la función cargar. Si el fichero existe, con la posición del jugador determino el circulo donde se encuentra, y almaceno el tiempo. Utilizo “Localize” de “LocalizationManager” para obtener el nombre del circulo con la llave de localización correspondiente.

3.11.4 Dialogue, Dialogue1

Estos scripts gestionan los diálogos que surgen en el videojuego. Hay dos scripts de dialogo adicionales que se comentaran en secciones posteriores.

Para llevar a cabo este proceso, primero es importante destacar que las líneas estarán definidas en un vector. La función central de estos script es “ShowLine()”, la cual despliega gradualmente cada línea de dialogo en la pantalla, mostrando de carácter en carácter.

En primera instancia se verifica si el jugador está lo suficientemente cerca del NPC[14] para entablar una interacción. De ser así, al pulsar la tecla “h” o “Enter”, el dialogo se inicia pasando por “ShowLine”. Mientras el dialogo esta en curso, si no se ha terminado de mostrar toda la línea, al pulsar de nuevo se mostrará la línea completa; si se ha terminado de mostrar la línea, se actualiza el índice con el que recorro el vector y pasamos a mostrar la siguiente línea. Cuando ya no quedan más líneas por mostrar, se desactiva el panel de dialogo y reinicio el vector, poniendo el índice a cero.

“Dialogue1” se diferencia en que el dialogo no continua, sino que cada vez que se muestra una línea, debes volver a hablar con el personaje si quieres que te diga algo más.

3.11.5 StartGamePlayerName

Como comenté anteriormente, cuando se carga la partida, si el fichero no existe se activa el panel que he diseñado ya que es una nueva partida. El funcionamiento es similar al presentado con los diálogos, se diferencia en:

- Mientras se realiza el dialogo, he añadido una serie de imágenes. Para mostrarlas, son activadas tomando en cuenta el índice de la línea del dialogo con la que corresponden.

[14] Personaje no controlable

- En la segunda linea del dialogo, aparece un input para que el jugador ponga su nombre, debe tener entre cuatro y dieciséis caracteres. Creo un patrón con “Regex.Scape”, y empleo “Regex.Replace” para sustituir en la linea todos los elementos que coincidan con mi patrón por la cadena de texto del input.

3.11.6 RowUi, Score, ScoreData, ScoreManager, ScoreUi, PlayfabManager

Una vez completado el videojuego, se desvela una clasificación a nivel mundial con los diez jugadores que han logrado superarlo en menor tiempo entre todos los participantes. Además, en el mismo panel, se muestra la posición en el ranking y el tiempo necesitado por el jugador actual para finalizarlo. Para gestionar esta funcionalidad, utilizo Playfab, una plataforma lanzada por Microsoft con el objetivo de facilitar las funciones que requieran almacenamiento de información en la nube para los desarrolladores de videojuegos. En esta sección, detallare los pasos realizados para concretar el ranking, abarcando desde el manejo de Playfab hasta el diseño mismo.

Dentro de un proyecto, en la plataforma web de Playfab, destacan dos pestañas fundamentales para nuestro propósito: Players y LeaderBoard. La pestaña Players nos proporciona un listado de todos los jugadores que han interactuado con el videojuego. Por otro lado, en la pestaña LeaderBoard, se establece el ranking partiendo de un valor de todos los jugadores anteriores. Para vincular Playfab con Unity es necesario instalar e importar la extensión que facilita para Unity y realizar el proceso de registro en el editor. Con esto, ya podremos utilizar los servicios de Playfab desde el código.

Respecto a la programación, es relevante destacar que cada vez que se llama a alguna función de la API de Playfab, debemos proporcionar ciertos parámetros. Estos parámetros incluyen una petición especifica, una función que se ejecutara en el caso de que la petición sea válida y otra función que será invocada en el caso de que se haya producido algún error.

Cuando se pone en marcha el videojuego, nuestro primer objetivo es llevar a cabo el registro del jugador en Playfab. Para lograrlo, creamos una solicitud del tipo “LoginWithCustomIdRequest”. A esta petición suministramos un valor de “CustomId” extraído de “SystemInfo.deviceUniqueIdentifier”, el cual proporciona un identificador único del dispositivo con el que se ha lanzado el videojuego. Además, ajustamos el parámetro “CreateAccount” a ”true”, lo que posibilita la creación de una cuenta en Playfab en el caso de que no exista ningún jugador que posea el identificador especificado. Luego, se llama al método “LoginWithCustomId” de la API.

Tal como mencione al explicar el funcionamiento de los diálogos, comente que había dos scripts mas, uno de ellos es “DialogueFinalCaronte”. En el momento culminante del recorrido por el infierno, hay un obstáculo que impide pasar a Caperucita. Al impactar contra este, aparece Caronte, quien está vinculado con el mencionado script. En este punto se transmiten los datos al ranking global de Playfab. Para esta acción, se utiliza la función “UpdatePlayerStatistics” de la API. Se proporciona una petición de tipo “UpdatePlayerStatisticsRequest”, especificando el nombre de la característica que requiere actualización, en mi caso el nombre del LeaderBoard, y la puntuación a añadir del jugador. Con esta acción, se completa la transferencia de información a la clasificación de Playfab.

Empezare comentando como crear el ranking en la interfaz de usuario. “ScoreData” funciona como un almacén de datos de “Score” que tendrán un nombre y una puntuación. “RowUI” define todos los textos que serán añadidos en las filas de la tabla de puntuaciones. Lo más destacable de “ScoreManager” es un método mediante el cual se añaden puntuaciones a “ScoreData”.

En “ScoreUI”, hacemos una invocación a un método del controlador de PlayFab para obtener toda la información del Leaderboard, esta se denomina “GetLeaderBoard”. Durante este proceso, empleamos la función “GetLeaderboardAroundPlayer”, a la que le proporcionamos el nombre del Leaderboard como argumento en la solicitud. Cuando la petición no da problemas, almaceno los datos en tres vectores: posición, puntuación e identificador de los jugadores.

Con los datos ya obtenidos, se añaden a “ScoreData” tantas “Score” como jugadores se presenten en el ranking. A continuación, aprovechamos esta información para generar una nueva fila (Row) por cada “Score”, en la tabla de resultados mostrada en la interfaz de usuario.

4 EXPERIMENTACIÓN, PRUEBAS Y DISCUSIÓN

Para abordar las pruebas en este proyecto, podemos clasificarlas en dos categorías: pruebas relativas a la implementación y pruebas de las plataformas.

En el contexto de las pruebas relativas a la implantación, el proceso se involucra una metodología similar a la programación convencional, independientemente d ela creación de videojuegos. Se llevo a cabo una comprobación constante para verificar si el código que estaba siendo desarrollado funcionaba según lo previsto. Cuando surgía problemas y el funcionamiento no se ajusta a lo deseado, se requería un análisis minucioso del código para identificar el origen del error. Sin embargo, es importante señalar que el entorno de programación Unity con el lenguaje C#, no se tiene la capacidad para depurar línea a línea, como se hace comúnmente en algunos lenguajes de programación. Esto añadía un nivel de complejidad adicional a la tarea de encontrar y resolver problemas, ya que se debe recurrir a la inspección visual del código o la inserción de instrucciones “Debug.Log” para imprimir mensajes en la consola.

Cuando la ejecución del juego no era posible, lo que indicaba un fallo significativo como se ve en la imagen siguiente, se adoptaba una estrategia especifica. Se insertan mensajes en consola en varias partes del código, etiquetados como “error1”,”error2”, “error3” y así sucesivamente. Si, al revisar la consola, solo se observaban mensajes hasta “error2”, esto señalaba que el error se encontraba en algún punto anterior al mensaje de “error3”. En casos donde el juego se ejecutaba, pero no se alcanzaban los resultados deseados, se utilizaban mensajes de consola para visualizar información relevante y aumentar la comprensión del problema.

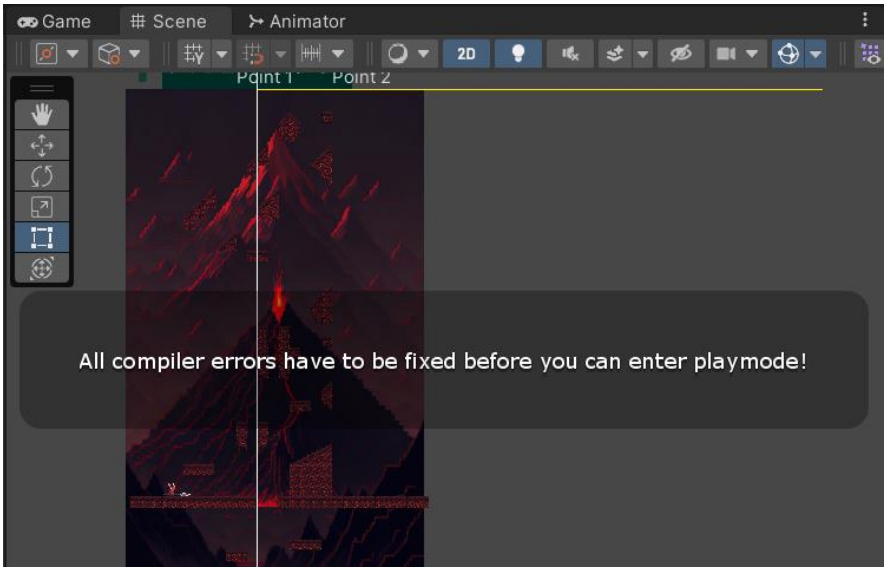


Ilustración 44: Error al ejecutar el entorno debido a fallo en codificación

Como ejemplo de uno de ellos desafíos que enfrente, me encontré en situaciones en las que los recursos gráficos utilizados presentaban imperfecciones, como pixeles muertos. Estas imperfecciones daban lugar a problemas como colisiones inesperadas con el suelo cuando se trataba de personajes que debían moverse. Para abordar esta cuestión, tuve que tomar medidas como corregir el diseño defectuoso o ajustar las propiedades de la gravedad del cuerpo del personaje para evitar que quedara atrapado en el suelo. Aunque estas modificaciones eran necesarias para el correcto funcionamiento del juego, resultan imperceptibles para el usuario final, dado que la diferencia entre el diseño y el suelo es mínima y no afecta a la experiencia del juego de manera evidente.

Con el objetivo de mejorar la experiencia del usuario y evitar un bloqueo por parte del jugador debido a un mal diseño de nivel, he realizado pruebas con cada uno de los pares de plataformas del juego. Estas pruebas han consistido en mover el personaje a la plataforma origen de forma manual y intentar saltar a la plataforma destino al menos una vez. Haciendo esto con todos los pares consecutivos de plataformas y ajustando aquellas plataformas que no eran conseguibles podemos obtener la certeza de que el juego es completable al cien por cien.

5 MODELO DE NEGOCIO

Uno de los objetivos fundamentales en la realización de videojuegos es la obtención de una rentabilidad ya que es un proceso costoso que necesita una gran cantidad de recursos humanos y monetarios.

En el mercado actual de los videojuegos se presentan varios modelos, destacando tres principales [FourWeekMBA]:

- **Pay to Play**: Se hace un único pago, cuando se compra el juego. Ejemplo: God of War, Assassin's Creed, Far Cry.
- **Free to Play**: El juego es gratis desde un inicio, pero obtiene remuneración mediante anuncios o con pago pequeños de elementos adicionales como pueden ser nuevos aspectos, mejoras … Ejemplo: Fortnite, Clash Royale.
- **Suscripción**: Se realiza un pago para poder acceder de forma temporal a todos los elementos del videojuego. Ejemplo: World of Warcraft.

La gran diferencia entre Pay to Play y Free to Play, radica en el hecho de que los juegos Free to Play al ser gratuitos desde un comienzo, logran reunir una audiencia más amplia obteniendo ingresos de una pequeña parte de esta. Sin embargo, los Pay to Play, aunque pueden presentar una audiencia más reducida, obtienen un pago de todos sus jugadores al inicio.

En el panorama actual del mercado, se puede ver, como utilizan un modelo económico dependiendo del tipo de videojuego. En aquellos donde el principal atractivo es el aspecto narrativo, prevalece el enfoque Pay to Play. Por otro lado, en los videojuegos centrados en partidas breves, se decantan por el modelo Free to Play, con la esperanza de captar la atención del jugador para realizar una transacción.

Los juegos de Suscripción quedan fuera de esta ecuación debido a que, en su mayoría, el único prominente es el videojuego World of Warcraft, el cual ganó notoriedad desde su lanzamiento en 2004, posicionándose como un gran interés para los jugadores. Existe un modelo relativo a la suscripción en el que grandes compañías ofrecen un servicio mediante el cual se puede acceder a una gran cantidad de videojuegos. Por ejemplo, Microsoft con su servicio Game Pass. En mi caso, esta

última opción no es viable, ya que la empresa es la que elige los videojuegos que serán añadidos.

En mi opinión, el aspecto fundamental que separa a un videojuego independiente y uno desarrollado por una gran compañía es la publicidad, el cual es sumamente costoso. Aunque hay una alternativa muy viable en forma de publicidad a través de creadores de contenido, esta opción abarca diversos niveles económicos, facilitando en cierta medida la obtención audiencia.

Además, en la gráfica (Ilustración 12: Estudio de mercado - Medio principal de información sobre videojuegos) se puede apreciar claramente como internet es el mayor portal para obtener información sobre videojuegos.

Respecto a la publicidad, también quisiera destacar que el estudio de mercado fue promocionado a través de Twitter, Instagram y TikTok, con un mismo presupuesto en cada plataforma. Notablemente, TikTok generó casi el doble de audiencia y participantes en comparación con las otras dos plataformas.

A partir de lo ya expuesto, y en conjunto a los datos que se muestran en la gráfica (Ilustración 13: Estudio de mercado - Gastos por año en videojuegos) que detalla el gasto anual de las personas en videojuegos, podemos deducir que un juego independiente, como el mío, que no presenta una gran audiencia y no tiene una calidad comparable a un Triple A [15], no puede tener un gran costo en el caso de ser Pay to Play. Por tanto, se requerirá establecer un precio asequible con el fin de atraer audiencia.

En el caso de optar por un modelo Free to Play, podemos añadir elementos, como una nueva banda sonora, para llamar la atención de los jugadores. Además, en el contexto de los juegos independientes, se alberga la esperanza de que los jugadores que verdaderamente disfrutan del videojuego consideren donar una pequeña cantidad al creado como forma de apoyo.

[15] Videojuegos AAA son los juegos realizados por grandes compañías que cuentan con una gran cantidad de recursos a su disposición.

6 CONCLUSIONES Y TRABAJOS FUTUROS

En conclusión, puedo decir que mi experiencia en la realizacion del Trabajo de Fin de Grado ha sido profundamente enriquecedora. A lo largo de los meses de elaboración he presentado momentos desafiantes. Situaciones en las que los resultados no coincidían con lo esperado llevándome semanas de esfuerzo intentar ver cuál era la raíz del problema. Estos periodos de frustración, me han servido para poder asimilarla de una mejor forma y afrontarla. La sensación de logro al superar los obstáculos se convirtió fue increíblemente gratificante.

Si bien es cierto que a lo largo de mi recorrido académico he experimentado retos similares en otros proyectos del grado, esta experiencia se siente diferente. Quizás por el hecho de que no me fue asignado, sino que tuve la responsabilidad de todas las decisiones a realizar desde cero.

En cuanto al desarrollo de videojuegos como tal, he aumentado enormemente los conocimientos aprendidos de la asignatura “Desarrollo de videojuegos”. He logrado no solo manipular con mayor fluidez el editor de Unity, sino también he podido profundizar mi aprendizaje respecto a varios aspectos de C#. Este crecimiento ha llegado a un punto en el que siento que puedo concebir cualquier concepto relacionado con los videojuegos si lo planteo de forma adecuada.

Si hablamos respecto al diseño, también he podido potenciar los conocimientos que tenia de Photoshop y Gimp, utilizando herramientas que nunca había utilizado como el rellenado de imágenes de Photoshop. Aunque, a veces, esto ha sido una tarea más tediosa que el desarrollo porque si buscamos recursos en la web, encontraremos más elementos para entornos 3D, además me dirigí hacia una categoría especifica 2D, como es Pixel Art. Aunque no me arrepiento de la decisión tomada, ya que creo que el aspecto visual es muy adecuado con el videojuego.

Esta experiencia también me ha servido para valorar mas a las personas que me rodean. En este viaje, he rechazado en numerosas ocasiones oportunidades para pasar tiempo con mis seres queridos por el desarrollo del proyecto. Aunque podía haber esperado ciertas quejas, solo he recibido un apoyo constante.

También he aprendido cosas de otros países que raramente se encuentran documentadas en fuentes convencionales. A través de las promociones he conseguido participantes hispanohablantes de todo el mundo para mi estudio. Esto ha brindado la oportunidad de que compartan conmigo la situación en su país con el sector. Por ejemplo, un chico de Venezuela me explico que su único recurso para jugar es la piratería, dado que los juegos de generaciones pasadas tienen precios exorbitantes en el país y los juegos de las nuevas generaciones simplemente no llegan. Además, debido a la inflación de la moneda, enfrentan restricciones significativas para realizar pagos en línea. Esto que he comentado no debería ser tolerado, ya que, en mi percepción, los videojuegos son obras de arte que cuentan una historia con la que puedes interactuar, a diferencia de una serie o una película. No obstante, esta opinión no es solo mía, como se puede ver en las gráficas (Ilustración 14: Estudio de mercado - ¿Los videojuegos son un arte?)(Ilustración 15: Estudio de mercado - ¿Los videojuegos son una pérdida de tiempo?), destaca esta opción de forma clara.

El diseño de niveles ha resultado ser un desafío, ya que he garantizado la creación de saltos que provocaran frustración a los jugadores que se intenten pasar mi videojuego. Esto implica que el tiempo de juego sea una incógnita, ya que dependerá en gran medida de la habilidad del jugador.

A pesar de que he disfrutado creando mi videojuego, la fuerte competencia dentro de la industria y el hecho de que mis únicos recursos sean mis propios esfuerzos indican, siguiendo la lógica, que no se puede pensar en obtener una gran rentabilidad. Aunque es cierto que, en la época del confinamiento debido a la pandemia, el sector experimento un gran impulso (Ilustración 17: Estudio de mercado - Tiempo jugado respecto a la cuarentena por Coronavirus), actualmente hemos regresado a índices normales, donde las grandes compañías abarcan casi por completo el mercado.

Aunque lo que he comentado podría sonar pesimista, me refiero más bien a que no puedo considerarlo un trabajo en el sentido tradicional, sino más bien como un hobby apasionante. En este punto del recorrido, en el que mi creación presenta un gran avance, estoy decidido a añadir todos elementos que ya tengo pensados para

ver culminada mi obra de forma adecuada. Además de numerosas referencias o escenas que sirvan de anexo entre los distintos mundos, tengo la tarea pendiente de crear el Purgatorio y el Cielo. Aunque como esta tarea es muy ambiciosa buscare a personas que quieran embarcarse en este proyecto conmigo, principalmente a un diseñador, ya que esa es la faceta en la que me siento menos hábil.

7 APÉNDICES

7.1 Guía original del Trabajo Fin de Título

(Cód.: 21/22-3462) Desarrollo de un prototipo de videojuego

Tutor del TFG: **JUAN ROBERTO JIMENEZ PEREZ**
Modalidad: Proyecto de Ingeniería | Tipo: TFG General
Número máximo de estudiantes: 2 (2 asignados)
Idioma: Castellano
Asignado al alumno con DNI:26513899M
Asignado al alumno con DNI:77380414L

Este trabajo de fin de grado es una propuesta tipo para el desarrollo del prototipo de un videojuego. El tipo de videojuego es de libre elección, se puede optar por videojuegos de aventuras, de estrategia, en primera o tercera persona, en tiempo real, por turnos, en 3D o en 2D, juegos en red, orientados a realidad virtual o aumentada, específicos para algún tipo de plataformas o usando diferentes dispositivos, etc.

Dada la complejidad que hoy en día tienen los videojuegos que suelen ser productos software complejos donde se deben sincronizar de manera perfecto diferentes subsistemas tanto hardware como software, para su desarrollo se partirá de un motor de juegos que se ajuste a las necesidades específicas del juego. Es labor del estudiante elegir el más adecuado.

Independientemente del videojuego que se desarrolle hay aspectos que tienen especial relevancia y que será imprescindible definir de manera adecuada: las mecánicas de juego principales y la experiencia de usuario que en el fondo determinarán la jugabilidad del mismo.

Las mecánicas principales identificarán su funcionamiento interno incluyendo, por ejemplo: la leyes físicas que dominan en el juego, la economía interna o como el jugador consigue los recursos necesarios para avanzar, los criterios para pasar de un nivel a otro si es un juego de progresión, las reglas de comportamiento, si se requieren, etc. Evidentemente, las mecánicas concretas son muy dependientes del juego en si.

Así mismo, la experiencia del usuario también viene determinada por la interfaz de usuario, los mecanismos y dispositivos de interacción, la cámaras utilizadas, los elementos visuales y acústicos, los mecanismos de navegación, etc. Todos estos aspectos deben definirse a lo largo del desarrollo y suelen requerir el uso de prototipos (no siempre software) para evaluar la idoneidad de los mismos. Para el modelado de escenas, personajes, animaciones, etc. se podrán utilizar tanto elementos preexistentes disponibles en la red como otros de propia elaboración.

En resumen, teniendo en cuenta que es una propuesta general para el desarrollo de distintos prototipos, es labor del estudiante destacar de manera adecuada los elementos que definen el videojuego específico que desarrolla.

Conocimientos Previos

Cursar o haber cursado la asignatura "Desarrollo de videojuegos" del Grado en Ingeniería Informática.

Objetivos del TFG

- Identificar el juego a desarrollar, destacando las características principales del mismo así como el público objetivo.
- Definir la mecánicas de juego principales.
- Seleccionar el motor de juego más adecuado así como otras herramientas complementarias que se requieran en el desarrollo o interacción con el juego específico.
- Analizar y diseñar la interfaz e interacción con el usuario, así como identificar los dispositivos de interacción más adecuados al tipo de videojuego que se propone.
- Analizar y diseñar el mundo sobre el que se desarrolla, los personajes, animaciones, efectos visuales y de sonido.
- Evaluar el prototipo.

Metodología a Desarrollar

A continuación se propone una metodología genérica que sirva de referencia a cada TFG concreto.

- Definición del propósito, clasificación y los aspectos principales de la mecánica de juego que se propone.
- Planificación. Estimación de recursos y costes.
- Análisis de motores de juegos y principales componentes a utilizar.
- Análisis y diseño de las mecánicas del juego. Es posible que surja la necesidad de desarrollar prototipos (incluso en papel) que simulen el comportamiento de estas mecánicas.
- Análisis y diseño de la interfaz y de la interacción, identificando los dispositivos más adecuados.
- Análisis y diseño del mundo virtual incluyendo escenas, personajes y otras entidades.
- Modelado o selección de personajes, animaciones, efectos visuales y sonido.
- Implementación y pruebas.
- Evaluación del prototipo.
- Elaboración de la memoria del trabajo.

Documentos y Formatos de Entrega

Deberá entregarse un CD/DVD con el siguiente contenido:

- Memoria del TFG. (PDF)
- Código fuente y ejecutables.
- Video demostrativo del prototipo de videojuego.
- Anexos para la presentación del TFG:
  - Informe favorable del tutor.
  - Autorización de publicación.

7.2 Guía de instalación

Para llevar a cabo la instalación del videojuego, proporcionaremos un archivo que incluye todos los elementos necesarios para su ejecución. Este archivo ha sido

generado por Unity garantizando que todos los recursos necesarios han sido añadidos.

Una vez que el archivo de instalación se haya descargado correctamente, el proceso de instalación en sí es sencillo y directo. Los usuarios simplemente deberán ejecutar el archivo ejecutable (“.exe”) denominado ‘TFG’.

7.3 Manual de usuario

Este manual proporciona una guía detallada de las funcionalidades y comandos esenciales del videojuego, con el objetivo de facilitar una experiencia de juego sin contratiempos para el jugador. Se presentarán a continuación los elementos fundamentales para una compresión exhaustiva del videojuego.

7.3.1 Controles del personaje principal

- **Movimiento horizontal**: Para desplazarse horizontalmente en el videojuego, se emplean las teclas “A” y “D” o las flechas izquierda y derecha del teclado.
- **Salto Controlado**: La acción de saltar se ejecuta mediante la tecla “Barra espaciadora”. Se debe tener en cuenta que la duración de la presión determinara la longitud del salto. Por tanto, es crucial calcular con precisión la fuerza aplicada para alcanzar la plataforma siguiente.
- **Transformación Multifacética**: El personaje principal posee la capacidad de transformarse en distintas versiones a través de las teclas “Q”, “E” y “R”. A lo largo del mundo encontraremos plataformas la cuales no podremos usar si son del mismo color que nuestro personaje.
- **Ataque**: La tecla “Ctrl” activa la función de ataque, permitiendo al jugador defenderse de los adversarios y desafíos presentes en el juego.

7.3.2 Aspectos relevantes

- **Interacción con Personajes No Jugables**: Los NPC con los que puedes entablar diálogos estarán señalados por un icono cuando el jugador se encuentre a una distancia adecuada para interactuar. La tecla “H” o “Enter” inicia y continua los diálogos con estos personajes.

- **Mecánica del salto**: El salto sigue una trayectoria parabólica y su distancia es directamente proporcional al tiempo que se mantiene pulsado la tecla comentada anteriormente.
- **Rebote en plataformas y paredes**: El personaje principal cuenta con la habilidad de rebotar contra plataformas y paredes, una capacidad crucial para superar desafíos específicos y alcanzar zonas inaccesibles de otro modo.
- **Efecto de empuje de los enemigos**: Los enemigos presentes en el juego no infligen daño directo al personaje principal. En cambio, aplican una fuerza de empuje variable dependiendo del enemigo, lo que puede ocasionar que el personaje descienda en el mundo. Esta mecánica se debe tener en cuenta a la hora de planificar la navegación por el mundo.

7.4 Enlaces de los recursos utilizados

Character Skinless Yarik by NovelY. (s. f.). itch.io. https://novely.itch.io/character-skinless-yarik

Apple by XXAShuraXX. (s. f.). itch.io. https://xxashuraxx.itch.io/apple

Treasure+ by SciGho. (s. f.). itch.io. https://ninjikin.itch.io/treasure

2D Pixel art Dungeon Collectables by Elthen's Pixel Art Shop. (s. f.). itch.io. https://elthen.itch.io/2d-pixel-art-dungeon-collectables

Light and Dark by DreamGryphon. (s. f.). itch.io. https://dreamgryphon.itch.io/light-and-dark

GOLDEN GOODS by TheScrawlHive. (s. f.). itch.io. https://thescrawlhive.itch.io/golden-goods

Environment Pack 01 by Hugues Laborde. (s. f.). itch.io. https://hugues-laborde.itch.io/environment-pack-01

Forest Pixel Platformer 2D Art TilesSet by SarturXz. (s. f.). itch.io. https://sarturxz.itch.io/forest-pixel-platformer-2d-art-tilesset

Moten Lava 32 x 32 Tile set for platformer Games by TheConceptofChris. (s. f.). itch.io. https://theconceptofchris.itch.io/moten-lava-32-x-32-tile-set

Free 3 colors Bush Pack by SaGy. (s. f.). itch.io. https://sagydemn.itch.io/free-3-colors-bush-pack

Pixel Art Sewer Background by Lil Cthulhu. (s. f.). itch.io. https://lil-cthulhu.itch.io/pixel-art-sewer-background

Grotto Escape II | 2D Textures & Materials | Unity Asset Store. (2017, 16 junio). Unity Asset Store. https://assetstore.unity.com/packages/2d/textures-materials/grotto-escape-ii-86689

Platformer Set | 2D Environments | Unity Asset Store. (2021, 15 junio). Unity Asset Store. https://assetstore.unity.com/packages/2d/environments/platformer-set-150023

Green Woods by Hello_Tazzina. (s. f.). itch.io. https://hello-tazzina.itch.io/green-woods

2D Top Down Chocolate Bar Tileset Pack by UchiMama. (s. f.). itch.io. https://uchimama.itch.io/2d-chocolate-tilesets

Fossil by NYKNCK. (s. f.). itch.io. https://nyknck.itch.io/fossil

Pixel Medieval Chandelier by NYKNCK. (s. f.). itch.io. https://nyknck.itch.io/pixel-medieval-chandelier

Old Man by Ulerinn. (s. f.). itch.io. https://ulerinn.itch.io/free-old-man

Wizard Sprite by Warren Clark. (s. f.). itch.io. https://lionheart963.itch.io/wizard

2D Flying Dragon Enemy by Trimurti Games. (s. f.). itch.io. https://trimurtigames.itch.io/2d-flying-enemy

Blue Whale by RAPIDPUNCHES. (s. f.). itch.io. https://rapidpunches.itch.io/blue-whale

Evil Black Cat by Mirquiso. (s. f.). itch.io. https://mirquiso.itch.io/evil-black-cat

Bat sprites by Rentro_Ghost. (s. f.). itch.io. https://rentro-ghost.itch.io/bat-sprites

2D Action Platformer Fantasy Character - Dragon Knight by Pixramen. (s. f.). itch.io. https://pixramen.itch.io/2d-action-platformer-fantasy-character-dragon-knight

Background Ruined Temple by Szadi Art. (s. f.). itch.io. https://szadiart.itch.io/background-ruined-temple

Free Pixelart Tileset Dungeon 64x64 - 4 versions by MiguelsGP. (s. f.). itch.io. https://miguelsgp.itch.io/free-tileset-dungeon

The Tiny Alchemist - Tileset 1 by Penusbmic. (s. f.). itch.io. https://penusbmic.itch.io/the-tiny-alchemist-tileset-1

2D Pixel Art - Distant Forest 64x64 | 2D Environments | Unity Asset Store. (s. f.). Unity Asset Store. https://assetstore.unity.com/packages/2d/environments/2d-pixel-art-distant-forest-64x64-194105

Diseño de fondo abstracto free vector. (2017, 17 febrero). Freepik. https://www.freepik.es/vector-gratis/diseno-fondo-abstracto_1054592.htm#query=textura%20morada&position=0&from_view=keyword&track=ais

Sunchoote, S. (2018, 11 octubre). *Espejo del monstruo del arte del pixel del vector*. Dreamstime. https://es.dreamstime.com/espejo-del-monstruo-arte-pixel-vector-image128412694

Non. (s. f.). *Pixel Art set Isolated Luxury mirror*. iStock. https://www.istockphoto.com/es/vector/pixel-art-set-espejo-de-lujo-aislado-gm1259921787-369097588

Vector Pixel Art Mirror Hang Vector de stock. (s. f.). Adobe Stock. https://stock.adobe.com/es/images/vector-pixel-art-mirror-hang/227015694

Pixeleart. (2023). Download the A Mirror in Pixel Art Style 22471667 royalty-free vector from Vecteezy for your project and expl. . . *Vecteezy*. https://www.vecteezy.com/vector-art/22471667-a-mirror-in-pixel-art-style

Contributors to Pixel Worlds Wiki. (s. f.). Mirror. *Pixel Worlds Wiki*. https://pixelworlds.fandom.com/wiki/Mirror

Vector Pixel Art Mirror Broken - Obrazy - Redro. (s. f.). Redro. https://redro.pl/obraz-vector-pixel-art-mirror-broken,176744919

Paige, S. (2019, 9 junio). *661. Mirror*. Pinterest. https://www.pinterest.com.au/pin/417427459210486615/?amp_client_id=CLIENT_ID%28_%29&mweb_unauth_id=%7B%7Bdefault.session%7D%7D&_url=https%3A%2F%2Fwww.pinterest.com.au%2Famp%2Fpin%2F417427459210486615%2F&open_share=t

Minecraft Swords by Miguel_PM_Romeu. (s. f.). itch.io. https://miguel-pm-romeu.itch.io/minecraft-s

Pile of envelope letters with wax seal and rope vintage craft paper in cartoon style Premium vector. (2022, 25 enero). Freepik. https://www.freepik.com/premium-vector/pile-envelope-letters-with-wax-seal-rope-vintage-craft-paper-cartoon-style_22692342.htm

OpenSea. (s. f.). *Classic Pixel Painting #48/99 Jesucristo Crucificado - Diego Velázquez - Classic Pixel Art*. OpenSea. https://opensea.io/assets/ethereum/0x495f947276749ce646f68ac8c248420045cb7b5e/62566048070024666516766559438217179879214118796311479919869006667840126713857

Chessman Columns by MirruTatep on DeviantArt. (2019, 1 julio). DeviantArt. https://www.deviantart.com/mirrutatep/art/Chessman-Columns-803883760

Aetherna. (2017, 1 febrero). *2D platform Ground Stone Tiles*. OpenGameArt.org. https://opengameart.org/content/2d-platform-ground-stone-tiles

Ilustración simple de cuerda blanca vectorial para desesperanza o suicidio en fondo negro premium Vector. (2022, 17 marzo). Freepik. https://www.freepik.es/vector-premium/ilustracion-simple-cuerda-blanca-vectorial-desesperanza-o-suicidio-fondo-negro_24659363.htm

Natrot. (s. f.). *Stone Black texture background. dark cement wall*. iStock. https://www.istockphoto.com/es/foto/fondo-de-textura-negra-de-piedra-pared-de-cemento-oscuro-gm1318101781-405336371

Lidiuchii. (2019, 6 septiembre). *Blackwork*. Pinterest. https://www.pinterest.es/pin/797137202778509060/

Pixel Coin Images – browse 12,488 stock photos, vectors, and video. (s. f.). Adobe Stock. https://stock.adobe.com/search?k=pixel+coin&asset_id=456377967

Pixel Stlye Cartoon Vista Frontal Cryptocurrency Matic o Polygon Purple Coin Premium Photo. (2022, 7 noviembre). Freepik. https://www.freepik.es/fotos-premium/pixel-stlye-cartoon-vista-frontal-cryptocurrency-matic-o-polygon-purple-coin_34016011.htm

Sol Logo | Pixel Art maker. (s. f.-b). https://pixelartmaker.com/art/71646dd9813dd57

Colección de personas en estilo Pixel Art Premium Vector. (2016, 28 septiembre). Freepik. https://www.freepik.es/vector-premium/coleccion-personas-estilo-pixel-art_948574.htm

Rewired | *Utilities Tools* | *Unity Asset Store*. (2014, 30 octubre). Unity Asset Store. https://assetstore.unity.com/packages/tools/utilities/rewired-21676

8 DEFINICIONES Y ABREVIATURAS

- **Testers**: Persona que realiza prueba de un desarrollo para colaborar su buen funcionamiento.
- **Toggle**: Interruptor con el que se pueden optar por dos opciones.
- **Prefab**: Es la abreviatura de prefabricado, cuando un elemento es desarrollado y diseñado se puede almacenar directamente en la carpeta de assets para ser utilizado cuando convenga.
- **Canvas**: Lugar donde se sitúan los elementos visibles de la interfaz de usuario.

9 BIBLIOGRAFÍA

[FourWeekMBA] “La Industria del Juego: Un análisis completo.” FourWeekMBA. Recuperado el 8 de agosto de 2023: https://fourweekmba.com/es/industria-del-juego/

[Niixer 2020] (2020, 19 noviembre). “Breve historia de los videojuegos.” Niixer. Recuperado el 15 de noviembre de 2022: https://niixer.com/index.php/2020/11/19/breve-historia-de-los-videojuegos/

[Nintendo Fandom] “Crisis del videojuego de 1983.” Nintendo Fandom. Recuperado el 16 de Noviembre de 2022 en: https://nintendo.fandom.com/es/wiki/Crisis_del_videojuego_de_1983

[Retromaquinitas a] “Catálogo de Consolas: Magnavox Odyssey.” Retromaquinitas. Recuperado el 15 de noviembre de 2022: https://retromaquinitas.com/catalogo/consolas/philips/odyssey/

[Retromaquinitas b] “Catálogo de Consolas: Mattel Intellivision.” Retromaquinitas. Recuperado el 15 de noviembre de 2022: https://retromaquinitas.com/catalogo/consolas/misc-70s-80s/intellivision/

[Tokio School] “La historia del videojuego: El ascenso imparable de una industria del entretenimiento.” Tokio School. Recuperado el 18 de noviembre de 2022: https://www.tokioschool.com/noticias/historia-videojuego-ascenso-imparable-industria-entretenimiento/