Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de un videojuego de lucha en Unity

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Enric Bonet Cortés

Tutor: Javier Lluch Crespo

2018-19

Agradecimientos

A mi familia y a mi pareja, por todo el apoyo y consulta ofrecida durante la realización del proyecto.

A mis amigos, Dani, José y Marcos, por todos los consejos y su colaboración en las pruebas.

Y a mi tutor, Javier Lluch, por todo la información y conocimientos que me ha aportado.

Resumen

Actualmente, la industria de los videojuegos, se ha convertido en una explotación cuya continua expansión es innegable. Como fruto de ello, han llegado al mercado diferentes herramientas que facilitan el desarrollo de estos. A lo largo del tiempo, ha ido evolucionando no solo el concepto de videojuego en sí, sino también el de videoconsola, hasta el punto de encontrar en el mercado, una gran variedad de hardware que compite con una serie de especificaciones realmente avanzadas a los origines de estas, las máquinas recreativas. Así, este proyecto consiste en el desarrollo de un videojuego ambientado en el entorno de las máquinas recreativas, la llamada temática "arcade", que comprende un estilo de lucha clásico y que se ha desarrollado con la herramienta de desarrollo de videojuegos Unity.

Palabras clave: videojuego, arcade, lucha, Unity.

Resum

Actualment, l'industria dels videojocs, s'ha convertit en una explotació de la qual es innegable parlar de las seua continua expansió. Com fruït d'això, han aplegat al mercat diferents ferramentes que faciliten el desenvolupament d'aquests. Al llarg del temps, ha anat evolucionant no sols el concepte de videojoc en sí, si no també el de videoconsola, fins al punt de trobar al mercat, una gran varietat de hardware que competeix amb una sèrie d'especificacions realment avançades als orígens d'aquestes, les màquines recreatives. Així, aquest projecte consisteix en el desenvolupament d'un videojoc ambientat en el entorn de les màquines recreatives, l'anomenada temàtica "arcade", que compren un estil de lluita clàssic i que s'ha desenvolupat amb la ferramenta de desenvolupament de videojocs Unity.

Palabras clave: videojoc, arcade, lluita, Untiy.

Abstract

Nowadays, industry of video games, has become a one whose continuous expansion is undeniable. As a result of this, have arrived at market, many different tools that allow us develop video games. Over time, have evolved the concepts of video game and, most important, the concept of video game console to this point, when you can see in market a lot of hardware that compete with specifications that are too much far of his origins, arcade machines. This project consists in a development of a video game set in the environment of arcade machines, that have a classic fighting style, and has been developed with Unity, a tool to develop video game.

Keywords : video game, arcade, fighting, Unity.

Índice

Índice de imágenes

1. Introducción

Unity es uno de los motores gráficos más potentes que existen en el mercado. Como tal, ofrece una cantidad de herramientas sencillas para el desarrollo de videojuegos por parte de desarrolladores principiantes.

En este documento se expondrá como se ha llevado a cabo la realización del proyecto "Victory Royale", un videojuego de género "fighting", inspirado en diferentes juegos de temática "arcade". Para ello, se han utilizado diferentes recursos que ofrece Unity como motor gráfico (GameObjects, Scenes, etc.), a la vez de funcionalidades que nos permite también este motor mediante el Scripting, como elementos de manejo de escena o las Coroutines. A estos recursos, se le añaden en el apartado gráfico y del diseño de personajes, otros que pueden encontrarse en Internet, en diferentes plataformas como la propia tienda de recursos de Unity: Asset Store.

Por último, y como se expondrá más adelante, la idea de "Victory Royale" se forma bajo un concurso organizado por parte de la página web www.minijuegos.com. Este concurso, consiste en un premio para el que sea elegido como el mejor videojuego por un jurado, ofreciendo, además, la posibilidad de publicar en dicha web los mejores videojuegos.

1.1 Motivación

Este proyecto, como ya se ha mencionado, se construye bajo una propuesta de participación en un concurso de juegos, organizado por una famosa página web de videojuegos. Para ello, se podría realizar cualquier tipo de videojuego. No obstante, se ha escogido este género debido a la cantidad de títulos famosos del género "fighting" que se pueden encontrar hoy en día. Algunos de estos, han servido como inspiración para diferentes aspectos de "Victory Royale". Además, y en la experiencia personal del desarrollador, muchos de ellos han marcado su experiencia como jugador.

A parte de esto, también han servido como motivación para este proyecto, los conocimientos obtenidos a través de algunas asignaturas cursadas, como *Introducción a los Sistemas Gráficos e Interactivos*, *Introducción a la Programación de Videojuegos* y *Entornos de Desarrollo de Videojuegos*.

Por último, el título "Victory Royale", ha sido escogido como guiño a un famoso videojuego actualmente, llamado "Fortnite", de género shooter. Este título, proviene del texto que aparece en el juego al ganar una partida.

1.2 Estructura

Esta memoria, queda estructura en 9 capítulos, o apartados, principales, contando el actual, además de Bibliografía, con referencias relevantes que han ayudado a su redacción, y un Anexo en el que se puede encontrar un Game Design Document (GDD) de "Victory Royale"

En el siguiente apartado, se detallarán los objetivos que ha sido planteados previamente a la realización del juego sobre Unity.

En el apartado del **Estado del Arte**, se hará una breve explicación del estado actual de la industria de los videojuegos, repasando su historia y enfatizando en los diferentes motores gráficos que actualmente se encuentran en el mercado, y, a su vez, pasando a hacer un pequeño análisis sobre algunos videojuegos del género "fighting", que han servido de referencia para "Victory Royale".

Tras esto, se expondrán los diferentes bloques de planificación en los que se ha divido el desarrollo del videojuego, empezando por el "storyboard" que ayudó a su vez a definir los objetivos que se habrán expuesto anteriormente.

Tanto los menús, como los personajes del videojuego, se han considerado como elementos importantes en el videojuego. Por ello, en el apartado **Análisis y Diseño**, se dedicará tiempo a estos elementos, así como a la interfaz de usuario, la cual se planteará como un objetivo importante. En este capítulo, se explicarán las diferentes propuestas que existieron para estos elementos, así como su resolución final.

No obstante, estos elementos pasarán por un proceso de implementación posterior dentro del motor Unity, para incorporarlas al videojuego. En el capítulo 6, **Implementación**, se tratará de explicar de forma sencilla, como han sido utilizados los diferentes recursos y herramientas que ofrece Unity, para desarrollar "Victory Royale".

Después del apartado de la **Implementación**, se inicia la parte final de este documento con el apartado de **Pruebas**, donde se expondrán las diferentes pruebas que se han realizado posteriormente a la implementación del videojuego, diferenciando entre pruebas alpha y pruebas beta, y donde se explicarán los diversos problemas que se han encontrado.

En el capítulo 8, **Resultados**, se explicará cómo se han resuelto los problemas encontrados en el apartado de anterior, volviendo a utilizar para ello, términos técnicos de Unity usados en el apartado de **Implementación**.

Para finalizar con los apartados principales de la memoria, en el apartado 9, se dará conclusión al proyecto, exponiendo el resultado de los objetivos planteados en el capítulo 2, y dando a entender algunas propuestas que quedan para un trabajo futuro sobre "Victory Royale", como la inclusión de nuevos personajes y nuevos enemigos.

2. Objetivos

Este proyecto, como ya se ha presentado en apartados anteriores, consiste en la creación de un videojuego de lucha de estilo clásico con un sistema de juego “arcade”, inspirado en las máquinas recreativas antiguas. Como objetivo principal del proyecto, sería el desarrollo de este videojuego, de nombre “Victory Royale”.

2.1 Objetivos del videojuego

Entrando en los objetivos que el videojuego, al ser un juego su objetivo principal debe de ser el de entretener al usuario. De esta forma, se ha ido condicionando el desarrollo del videojuego con diferentes ideas y objetivos que permitan a los jugadores divertirse.

2.1.1 Variedad

Primero y principal, para que un videojuego sea divertido, este debe de contar con varias alternativas de jugabilidad y de dificultad, que impidan al juego volverse algo monótono y que finalmente haga perder el interés en él.

Y no es de una forma explícita como se representa la variedad en la dificultad en “Victory Royal”, como podría ser un menú que te permite elegir diferentes modalidades como “Fácil”, “Avanzado” o “Muy Difícil”, sino de forma implícita con la elección de personajes. El juego consta de un trío de personajes a elegir por el jugador. Los tres personajes tienen diferentes acciones o, mejor dicho, golpes que pueden realizar. Este set de golpes es lo que realmente diferencian a cada uno de los personajes, más allá de su apariencia, y no es que solo se diferencien por el golpe en sí, es decir, su animación, si no que la diferencia de estos reside en su tiempo y daño de impacto. Con la suma de estos parámetros a la vida de cada uno de los personajes, la variedad en cuanto a la dificultad que se planteaba al principio de este párrafo, queda solventada permitiendo que cada personaje a elegir represente un estilo de juego y dificultad diferente.

En cuanto a la jugabilidad, inicialmente “Victory Royal” se planteaba con la idea de un único formato de solo un jugador, pero finalmente se ha introducido un segundo modo, que permitiese a dos jugadores jugar simultáneamente en el mismo ordenador. Estas dos modalidades de combates se encuentran en el videojuego bajo el nombre de “Solo”, en el caso de un solo jugador, y “Versus”, en el caso de dos jugadores. La inclusión de este modo multijugador introduce en el juego un factor de diversión y a la vez, competitivo, al no tener que enfrentarse solamente a una inteligencia artificial, como en el caso de “Solo”, sino a un amigo u a otra persona.

Otro factor que se ha tenido en cuenta para dotarlo de variedad, en este caso visual, es la inclusión de varios escenarios donde puedan transcurrir las peleas. Esto permite introducir variedad estética, sin incidir en la jugabilidad.

2.1.2 Interfaz gráfica

Otro de los objetivos que se ha planteado para el desarrollo del videojuego, es la implementación de una interfaz de usuario sencilla y funcional, que permitiese al usuario interactuar con diferentes aspectos del juego, como navegar por el flujo de las escenas o ajustar parámetros como el volumen del juego, de una forma sencilla y lo menos tediosa y complicada posible. Por ello, por ejemplo, tanto el color de los paneles como el de los botones de la interfaz, se han cuidado de tener una relación lógica que al interactuar el usuario le permita reconocer que ha llegado al sitio que quería alcanzar. Todos estos aspectos de la interfaz de usuario, se comentarán más en detalle en puntos posteriores en los que se hablará sobre su diseño.

2.1.3 Concurso www.minijuegos.com

Finalmente, como último objetivo de la creación de este videojuego, es el concurso de la página web www.minijuegos.com sobre el cual se conformado la idea de este proyecto. Este videojuego se presentará a dicho concurso, en el cual los mejores serán publicados en dicha página. Para ello, el videojuego será finalmente presentado en formato para WebGL, algo que se construye utilizando el módulo de Unity orientado precisamente a esta tecnología.

3. Estado del Arte

No es hasta los años 50 en los que, tras el nacimiento de las primeras computadoras de la mano de la Segunda Guerra Mundial, nacen los primeros videojuegos. El nacimiento de estos primeros videojuegos fue de carácter experimental, pues no fue hasta la década de los 70 en la que este tipo de juegos empezó a comercializarse junto con las máquinas recreativas.

De entre todos los tipos de videojuegos que existían entonces con las máquinas recreativas, los juegos de deportes incluían un amplio catálogo de sub-tipos de videojuegos. Uno de ellos, el de los videojuegos de lucha, se caracterizaba por disponer en escena dos personajes, bien controlados los dos por dos jugadores, o bien solo uno de ellos controlado, que se disponen a luchar entre ellos utilizando su movilidad y sus movimientos para derrotar al rival y ganar el combate. Estos juegos se enmarcan en una ambientación que hoy en día se llama "arcade" a pesar de la evolución de los componentes gráficos. En este apartado no solo se hablará sobre algunos de estos títulos clásicos de videojuegos que sirven como inspiración para "Victory Royale", sino que, además, se hará un repaso por las diferentes herramientas o motores de videojuegos que se pueden encontrar para el desarrollo de videojuegos.

3.1 Motores de videojuegos

Los motores de videojuegos, o motores gráficos, de hoy en día, permiten en gran medida desarrollar por completo un videojuego. Es por ello que han salido al mercado diferentes propuestas de motores, que animan cada vez más a que la gente cree su propio videojuego. Algunos de los más importantes son: Cryengine, UbiArt Framework, Source, GameMaker o Stencly; pero sin duda, por encima de estos, se encuentran dos motores que despuntan en el mercado, siendo los más utilizados hoy por hoy. A continuación, se entrará más en detalle a hablar sobre estos dos potentes motores, Unity y UnrealEngine.

3.1.1 Unity

Unity es considerado uno de los gigantes de los motores gráficos para el desarrollo de videojuegos. Fue desarrollada por Unity Technologies y es considerado un motor de videojuegos multiplataforma por implementar herramientas que permiten desarrollar juegos para todo tipo de dispositivos móviles y videoconsolas.

Empezó en 2005 como un software de pago solo disponible para usuarios de Mac OS X, implementando una versión gratuita y limitada. Más tarde, pasaría a estar disponible también para Microsoft Windows y Linux, y a partir de la versión Unity 5,

las limitaciones que aparecían en la versión gratuita desaparecieron, dejando como diferencias con la versión de pago solo algunas opciones de almacenamiento en nube.

Actualmente, es un potente motor de desarrollo de videojuegos para personas en aprendizaje por su facilidad, rapidez y versatilidad a la hora de crear escenas, y el uso de lenguajes conocidos como C# o JavaScript.

En el campo profesional, Unity también incorpora potentes herramientas para desarrollar juegos utilizando nuevas tecnologías como la realidad virtual (VR) y la realidad aumentada (AR).

Imagen 1: Rick and Morty Virtual Rick-ality hecho con Unity 5.

3.1.2 Unreal Engine

UnrealEngine es el otro de los que son considerados los dos gigantes del mercado en cuanto a motores gráficos. Fue desarrollado por la compañía Epic Games y fue lanzado oficialmente en 1998. Inicialmente se concebía para la realización de videojuegos de estilo “Shooter”, caracterizados por, generalmente, ser juegos cuya finalidad es la de disparar a un objetivo. Finalmente, este motor ha logrado utilizarse en gran variedad de géneros de los videojuegos. Al igual que Unity, este motor es multiplataforma.

Su última versión, UnrealEngine 4, ofrece una gran potencia como motor gráfico utilizando como lenguaje de programación principal C++ y, además, al igual que en el caso de muchos motores de videojuegos, de manera gratuita. Epic Games decidió ofrecer su software para la creación de videojuegos de forma gratuita, reclamando a cambio en el caso de comercializarse alguna creación con UnrealEngine, el 5% de las ganancias.

Este motor de videojuegos, es el origen de grandes títulos conocidos hoy en dia, aunque cuenta con un amplio repertorio de grandes videojuegos que han sido creados a partir de él a lo largo de su historia. Algunos de los videojuegos que han sido realizados con UnrealEngine recientemente son, el videojuego “Fortnite”, de la propia empresa Epic Games, o “Kingdom Hearts III”.

Imagen 2: Kingdom Hearts III hecho con UnrealEngine 4.

3.2 Género "Fighting"

"Victory Royale" es un videojuego del género conocido generalmente como "fighting" o, en castellano, de peleas, un género englobado dentro de los videojuegos de deportes y de estilo "arcade". Este género comparte una serie de características en todos sus juegos. Algunas de estas características, son su movimiento 2D o la posición de las barras de vida de los personajes siempre dispuestas en la zona superior. A continuación, se va a hablar de los grandes títulos que ha habido hasta la actualidad de este género.

3.2.1 Street Fighter

"Street Fighter" posiblemente sea el más famoso de esta lista de videojuegos de peleas. La empresa Capcom fue la encargada de sacar este título al mercado en el año 1987 en forma de videojuego de máquinas recreativas y, hoy en día, la franquicia de "Street Figther" sigue sacando nuevas ediciones del clásico de los juegos "fighting", conservando la temática "arcade" para plataformas actuales como la Nintendo Switch o la PlayStation 4.

La saga "Street Fighter" se ha caracterizado siempre por tener un movimiento de los personajes sencillo, no obstante, en el campo de los golpes y ataques que pueden realizar los personajes, cuentan con una larga lista de enrevesadas combinaciones de botones (los denominados combos), que permiten a cada personaje realizar movimientos especiales únicos de cada uno.

Imagen 3: Street Fighter original y Street Fighter V.

3.2.2 Mortal Kombat

Otra de las sagas que han marcado todo un género como el de los videojuegos de peleas, es la saga de videojuegos "Mortal Kombat". Midway Games fue la empresa que inició esta saga, sacando su primer videojuego en 1992, tomando como referencia la saga "Street Fighter" y al igual que con el título anterior, empezando como un videojuego de máquinas recreativas. No fue hasta su cuarta entrega, cuando los juegos fueron transportados a las consolas domésticas de la mano de la distribuidora estadounidense Acclaim Entertainment, pasando posteriormente a ser parte de la compañía Warner Bros a partir 2009 hasta la actualidad.

Hoy en día, el último título de la saga, Mortal Kombat 11, sigue siendo fiel a sus orígenes y conservando características propias como personajes originales y movimientos especiales realizados a partir de combos. Otra de las características más significativas de los juegos "Mortal Kombat", y que más llama la atención a sus compradores, son los llamados "fatalities". Este término, que nace precisamente de esta saga, se refiere a la oportunidad de, cuando se ha derrotado al otro personaje, realizar una última combinación de botones que permitan iniciar una secuencia animada que termine matando al personaje enemigo de una forma cruel, a modo de burla por haber ganado.

Imagen 4: Mortal Kombat original y Mortal Kombat 11.

3.2.3 Dragon Ball FighterZ

A parte de ser el último título de esta lista de videojuegos que han inspirado el proyecto de “Victory Royale”, “Dragon Ball FighterZ” es uno de los últimos juegos del género “fighting” que se han lanzado al mercado. Desarrollado por Arc System Works y distribuido por Bandai Namco Entertainment en 2018, “Dragon Ball FighterZ” está ambientado en a la de la obra de Akira Toriyama, el “manga” japonés “Dragon Ball”.

Este videojuego, comparte grandes características con otros juegos de peleas como los mencionados anteriormente incluyendo, además, la posibilidad de manejar tres personajes diferentes en la misma pelea e ir cambiando el personaje principal entre estos. A parte de las características que comparte con sus similares, “Dragon Ball FighterZ” se jacta de sus gráficos, simulando un entorno de máquinas recreativas y preservando los actuales avances en calidad y resolución.

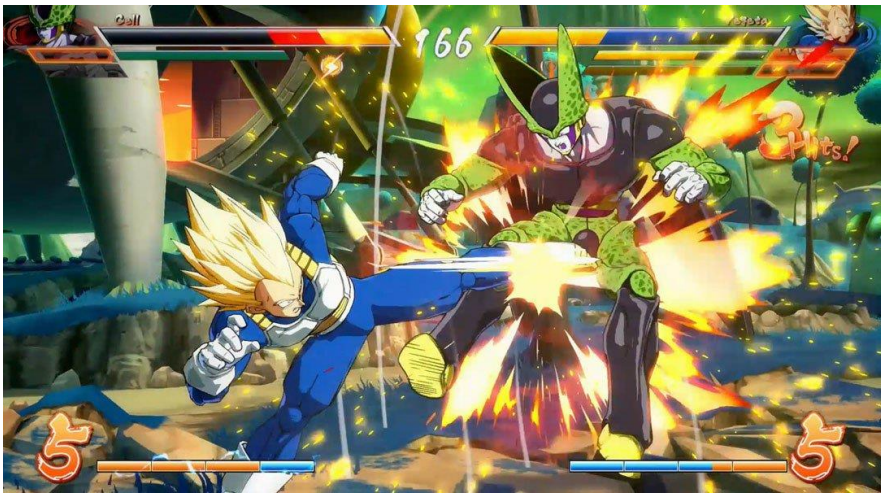

Imagen 5: Dragon Ball FighterZ

4. Planificación

Todo proyecto, requiere de una buena planificación para llevar acabo los diferentes objetivos que se proponen para este. A continuación, se expondrán las diferentes fases por las que se ha pasado “Victory Royale” en todo su proceso de desarrollo.

4.1 Storyboard

Una de las cosas más importantes para realizar un videojuego, es tener claro, de alguna forma, lo que se quiere realizar. Para ello, muchas veces tener una idea en la mente no basta para poder implementar todo lo que se desea que el videojuego haga.

Por esta misma razón, lo primero que por lo que se empieza a hacer en este proyecto, es un “storyboard”. El “storyboard” permite que se vean y se entiendan de una forma clara, las ideas que se han planteado desde un principio. De esta forma, el “storyboard” sirve de base para empezar a realizar el videojuego y como ayudar en todo el proceso de implementación posterior.

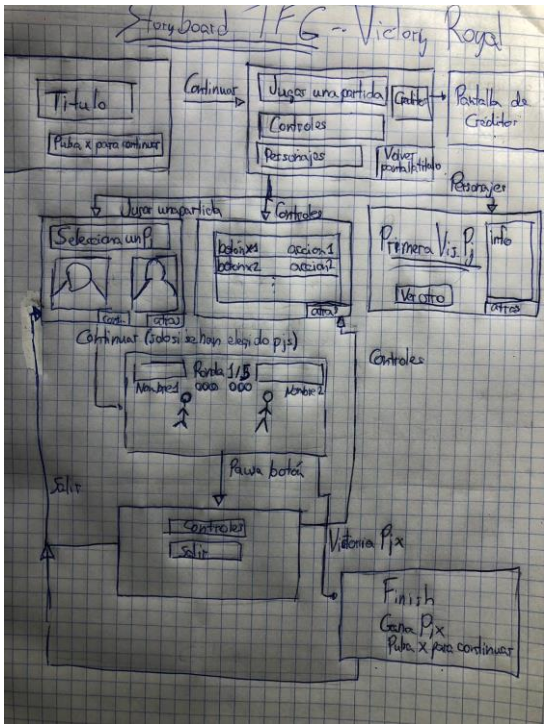

Imagen 6: Storyboard de VictoryRoyal

Como se puede observar en el “storyboard”, muchas funcionalidades importantes del juego final, se encuentran plasmadas desde de un principio en él, algunas tan

importantes como lo son el flujo de escenas del videojuego, el cual se mantiene cunado el jugador selecciona el modo “Solo”. También detalles como que el botón de seleccionar un personaje no te permita cambiar de escena si no se ha seleccionado previamente un personaje, se encontraban dentro del “storyboard”. Sin embargo, otras cosas como el diseño final de las escenas, fueron cambiadas desde su idea inicial, y también se introdujeron cosas nuevas que no aparecen en el “storyboard”, como los dos modos de juego o la pantalla de selección de escenario.

4.2 Implementación de la lógica del videojuego

Tras dejar plasmadas las ideas iniciales sobre las que conformar “Victory Royal”, se dio paso a desarrollar el videojuego sobre Unity3D.

Siguiendo el “storyboard”, se fueron implementando en orden las diferentes escenas y funcionalidades. Lo primero que se conformó, fueron el menú principal y la pantalla de selección de personajes. Desde un principio, se le prestó más atención a la segunda, pues una de las funcionalidades que se querían implementar en esta, era el menú de selección rotatorio con un panel de información sobre cada uno de los personajes del juego. El menú principal, mientras tanto, se planteaba como algo más sencillo, pues el menú iba a ser puramente una interfaz de usuario con botones y un fondo giratorio con algunos elementos dispuestos con él.

Después de implementar estas dos escenas, se pasó a desarrollar la escena del combate en solitario, junto a los personajes y el enemigo.

Justo tras acabar la implementación de la escena de juego se pasó a concluir el proyecto, no obstante, se decidió incluir dos nuevas funcionalidades al juego que no se encontraban en el “storyboard”. La primera de ellas fue el menú de selección de mapa o escenarios, el cual se planteó desde un inicio como una cuadrícula de botones que fuese cambiando el fondo de la escena, a modo de pre visualización de lo que sería el escenario del combate. La segunda, fue implementar el modo multijugador, el cual, con unas pequeñas modificaciones en la escena del menú principal, permitiera atravesar al jugador un flujo de escenas similar al del modo en solitario, para alcanzar una escena de juego (también parecida al del modo de un solo jugador), y que combatiese contra otro jugador en el mismo ordenador. Además, para esta implementación, también se realizaron severas modificaciones en los personajes.

Por último, se finalizó esta fase incluyendo elementos de sonido, como música de fondo y efectos sonoros en los golpes, en las diferentes escenas, así como el control del sonido por interfaz de usuario.

Esta fase no se ha comentado con detalle debido a que se le dedicará una mayor explicación en el apartado de “Implementación”. En este apartado se ha comentado brevemente para explicar las diferentes fases en las que se planificó este proyecto.

4.3 Inclusión del arte en el videojuego

Finalmente, la última fase del proyecto consistió en, una vez teniendo el videojuego y sus funcionalidades implementadas, modificar todos los detalles visuales del videojuego, que sobretodo, correspondían a la interfaz de usuario.

Los mapas que se utilizaron desde que se incluyó el menú de selección de mapas, fueron cambiados por otro con resolución HD, pues los primeros que se habían colocado pertenecían al enlace "http://www.custommapmakers.org/skyboxes.php", y presentaban una serie de errores en los márgenes de la Skybox. Los nuevos escenarios fueron descargados de la Asset Store de Unity y pertenecen al paquete "Skybox Series Free".

Lo último que se realizó en el proyecto, fue modificar la interfaz gráfica, cambiando los botones y los paneles del juego, así como la imagen inicial del título que se utilizaba a modo de ejemplo. De esta forma, se buscaba dotar al juego de una estética propia, como se puede observar en las siguientes imágenes.

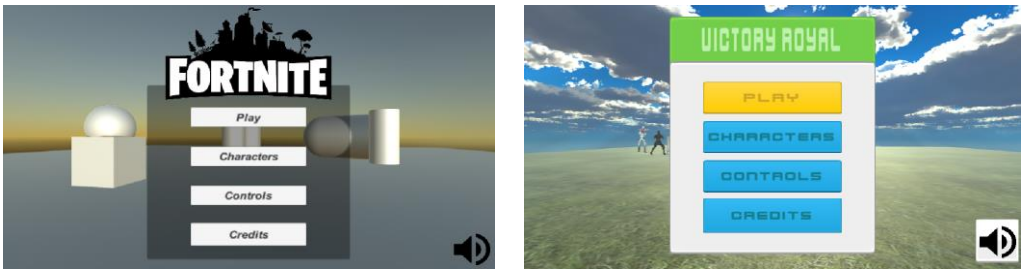

Imagen 7: Antes y después de modificar la interfaz gráfica.

5. Análisis y Diseño

Antes de pasar a implementar cualquier funcionalidad de “Victory Royale”, las más importantes se sometieron a un estudio previo con tal de calcular los recursos y las necesidades que esta llevaban consigo. Algunas características, como el menú principal, el cual ya se ha adelantado que se había planteado desde el principio como algo sencillo, no fueron sometidos a esta fase de análisis y diseño. Así pues, se pasará a continuación, a explicar los planteamientos previos que se hicieron sobre algunas características importantes del videojuego, sin entrar en detalles de implementación.

5.1 Análisis y diseño del menú de selección de personajes

El menú de selección de personajes de “Victory Royale”, es algo que fue pensado desde el primer momento en el que se planteó realizar un videojuego del estilo “fighting”. Desde este instante, se planteó realizar un menú rotatorio, debido a que su estética es realmente atractiva para el jugador, además de ser sencillo e intuitivo de utilizar por el usuario.

Gracias al segundo video disponible en la bibliografía, se pudo conformar una primera escena, que a falta de la inclusión de los personajes y de elementos artísticos como el cielo.

Con la parte lógica implementada, este menú permite capturar la pulsación de teclas que típicamente son asociadas al movimiento de los objetos, como la A o la D, o las flechas horizontales, para alternar entre cada uno de los tres elementos que se encuentran dispuesto de pantalla. De esta forma, en un futuro, el jugador o usuario, puede elegir con este método, el personaje con el que desea jugar.

5.2 Análisis y diseño del menú de selección de escenarios

Con tal de dotar al videojuego de mayor variedad estética, se planteó más tarde la inclusión de diferentes escenarios en los que pudiesen transcurrir los combates. Para ello, se debía de diseñar un nuevo menú que permitiese escoger al usuario entre un conjunto de mapas.

En primera instancia, y al ser una idea, por así decirlo, tardía, se planteó reutilizar el menú de selección de personajes con paneles que contenían las diferentes imágenes de los escenarios para finalizar con ello de forma rápida. No obstante, se optó por realizar un sistema más clásico de selección, que permitiese visualizar los escenarios de una forma más directa, pues con el sistema rotatorio implementado anteriormente, solo uno de los elementos queda centrado en pantalla, mientras que los otros pasan a un segundo plano.

La nueva idea consistía en utilizar los mismos paneles que se planteaban en la idea inicial, colocándolos como botones de Unity frente a la cámara en forma de matriz, y que su pulsación, permitiese visualizar el escenario antes de pasar al combate.

Los 6 mapas elegibles son los siguientes:

BlueFrezee: un escenario que contiene un fondo montañoso englobado en un ambiente azulado.

MegaSun: un escenario con un fondo nublado y un gran sol de atardecer que se encuentra suspendido en el aire.

DarkCity: un escenario decorado con una gran ciudad de fondo.

HighLands: un escenario que cuenta con un desierto en el amanecer como fondo.

UnearthlyRed: un escenario en el cual el combate transcurre en otro planeta diferente y alejado de la Tierra.

Stratosphere: como su nombre indica, un escenario que cuenta con la tierra vista desde la estratosfera como fondo.

5.3 Análisis y diseño de personajes

Sin duda, los personajes son los más importante en este videojuego, al igual que los son en muchos, y fue por ello por lo que se empezó a pensar en ellos desde tan temprano. Antes incluso que idear como realizar el menú de selección, se pensaba en la estética de los personajes y las habilidades. En un principio, estos iban a ser creados mediante la herramienta Blender de diseño 3D, no obstante, a falta de conocimientos, se optó por emplear modelados ya disponibles. Esto no debía significar dejar de lado que la estética fuese lo más parecida a lo que se pensaba de los personajes, y es que desde un inicio también se pensaba en un número mínimo de dos personajes elegibles, un personaje masculino y otro femenino. Otra cosa con la que los personajes contaban desde un inicio, era con un conjunto de habilidades y golpes que les diferenciaban entre ellos.

Los personajes del juego se caracterizan cada uno de ellos, aparte de en su diseño y movimientos, en su afinidad por cada una de las estadísticas base que hay: velocidad, fuerza y vida, entendiendo por velocidad y fuerza como conceptos de rapidez de atacar y daño de los golpes, respectivamente. Introducidas las particularidades de los personajes, se pasará a hablar de cada uno de ellos a modo de carta de presentación.

Como personaje masculino se eligió a Ethan, el modelado básico, por así decirlo, de Unity, que se encuentra en Standard Assets. Este personaje supuso el punto de partida y de experimentación para los siguiente dos, con los que, finalmente, conformaría la plantilla de personajes elegibles. A partir de él, se creó el script correspondiente al movimiento y control de los personajes, el cual se detallará más adelante. La estética que Ethan sugiere que es rápido, por lo que Ethan cuenta con un set de movimientos

veloces, superando en velocidad al resto de personajes. Esta alta velocidad, se contrarresta con la poca fuerza y vida que este personaje tiene, aunque esto no debe suponer un motivo para subestimar a Ethan, pues si no se combate debidamente contra él, puede bloquear cualquier movimiento de su adversario gracias a su velocidad de ataque.

El personaje femenino que se eligió fue Atenea, o mejor dicho Maria J J Ong, el cual es el nombre real de este modelado en la página de www.mixamo.com, de la cual procede. Gracias a esta página, se pudo completar el set de movimientos y golpes de tanto Atenea como de Eve, de la cual se va a hablar a continuación. El nombre de Atenea fue cambiado debido a que su nombre original era largo para el de un personaje de este videojuego. Sus características van condicionadas a su estética, al igual que con Ethan, pues debido a que lleva una armadura, su mayor característica es la vida. También posee una fuerza y una velocidad equilibradas, siendo la primera, superior a la de Ethan.

Por último, Eve fue el último de los personajes. Eve fue un personaje que se incluyó en "Victory Royale" para dotar al juego de mayor variedad de personajes. El modelado, al igual que Atenea y sus habilidades, pertenece a la página web www.mixamo.com. Si Ethan destacaba en velocidad y Atenea en vida, Eve lo hace en fuerza, pues el estilo de lucha principal de Eve son las patadas. Eso sí, de alguna forma su fuerza queda equilibrada con su escasa velocidad y su vida, inferior a la de Atenea.

Imagen 8: Ethan, Atenea y Eve.

Para terminar con los personajes, esta Derrick, el "zombie". Este personaje al contrario de los anteriores no es un personaje con el que se pueda jugar, sino que es el enemigo del modo "Solo" del videojuego. Siempre se pensó en este enemigo como en un "zombie" que persiguiese al jugador por el mapa causándole daño con sus golpes. El tercer video de la bibliografía, muestra cómo se puede realizar un enemigo similar a Derrick, aunque más adelante, se entrará en detalles sobre su implementación en "Victory Royale". Derrick posee una enorme cantidad de vida por ser un "zombie", además de que sus golpes pueden incluso tumbar a Ethan con solo dos de ellos gracias

a su fuerza. Su lentitud, es el factor que puede permitir al jugador llevarse la victoria frente a él.

Estos cuatro personajes, tienen en común que fueron creados con una animación especial que se lleva a cabo cuando ganan cualquier ronda del combate.

5.4 Análisis y diseño de la interfaz de usuario.

Como ya se ha adelantado en apartados anteriores, hacer una interfaz de usuario sencilla y cómoda para el usuario, era uno de los objetivos que se tenía en cuenta para este proyecto. Finalmente, se consiguió que la interfaz llevase al usuario a través de un flujo de escenas lógico y orientativo, desde el punto de vista del usuario.

A continuación, se muestra un diagrama del flujo de escenas hacia delante:

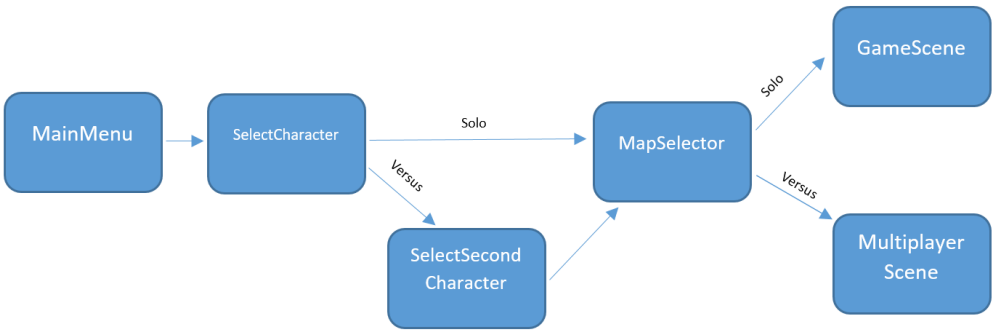

Imagen 9: Diagrama del flujo de escenas.

La interfaz a su vez, hace al usuario servirse de los botones “Esc” y “Enter” para avanzar entre escenas hacia atrás y hacia delante, respectivamente.

Otra de las cosas que se tuvo en cuenta al realizar la interfaz, es que no solo siguiese una lógica al avanzar entre escenas, sino que, además, estéticamente también siguiese cierta lógica. Algo que no se había mencionado antes, era el arte utilizado en los botones y los panales. Los “sprites”, o imágenes, que componen la interfaz gráfica, son de la página web https://opengameart.org. Pasando a explicar lo mencionado sobre la lógica estética de los elementos de la interfaz, se ha buscado siempre que los paneles siguiesen el mismo diseño, más o menos, de estilo minimalista: paneles claramente visibles y distinguibles entre ellos, que tuviesen un primer panel blanco sobre el que se vería la información principal del panel, y un segundo panel de otro color con un texto haciendo de título del panel. Este segundo panel, además de tener un color llamativo para poder distinguirlo con claridad, es del mismo color que el botón que lo ha abierto,

es decir, por ejemplo, el botón “Pause” de cualquier escena de juego que es de color amarillo, abre un panel “Pause” cuyo color en el segundo panel es de color amarillo. De esta forma, favorece a dar al usuario la sensación de haber llegado a su objetivo. Esta referencia de diseño, se sigue con todos los demás paneles dentro del juego.

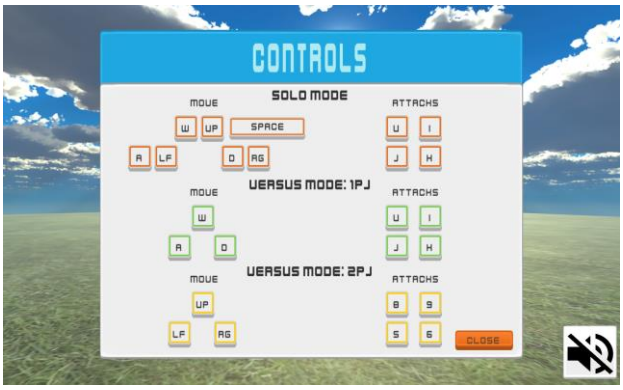

Imagen 10: Panel Controls abierto desde Controls del menú principal.

6. Implementación

Ahora sí, se va a entrar en detalle sobre la implementación del videojuego y sus elementos. En apartados anteriores, se ha hablado sobre qué tipo de planificación se ha llevado con respecto al desarrollo de este proyecto, definiendo 3 grandes bloques o fases. Este apartado va a englobar tanto la segunda fase, implementación lógica del videojuego, la cual se ha introducido en el apartado de **Planificación**.

6.1 Menú principal

Como ya se ha comentado con anterioridad, el menú principal fue lo primero que se creó en "Victory Royale". Para ello, fue necesaria la ayuda de la explicació del primer video que se encuentra en la bibliografía.

Primero se creó el fondo con elementos de Unity básicos, al cual se le añade un script ***Rotate_MainMenu*** para que de vueltas continuamente sobre el eje Y, utilizando en el método **Update()** del script, el método **Rotate()** sobre la componente **transform**.

Para realizar la interfaz de usuario, se utiliza un objeto de tipo **Canvas**, de Unity UI, al cual le añado un **Panel** translucido de color negro, superpuesto al fondo, de modo que se vea centrado en todo momento. Sobre este panel, se coloca una **Image** para poner una foto de prueba que posteriormente será sustituida.

Una vez dispuesta la base de la interfaz, se pasan a colocar los diferentes botones, **Button**, ajustándolos y colocandoles el texto idóneo para cada uno. Hecho esto, el menú carece de funcionalidad por lo que paso a realizar nuevas escenas de Unity (vacías por el momento) para enlazarlas a los diferentes botones, por medio de métodos que serán llamados en los eventos **onClick()** que lanzarán los botones al ser pulsados. Añadiendo en **Build Settings** de Unity las escenas, es como se permite que se lancen estas. Una vez hecho esto, en la escena del menú, se crea un objeto vacío (**Create Empty**) que se encargara de contener el script llamado ***MainMenuManager***. Dentro de él, se ha creado un método llamado **LoaderScene(String)**, que toma el parámetro String que se le pasa, y realiza **SceneManagement.LoadScene(String)** para cargar la escena cuyo nombre es el String que se le ha pasado. De esta forma, se asigna este método en onClick(), a cada botón que abre una nueva escena, pasándole como parámetro el nombre de la escena que a la que se quiere llamar con cada botón.

Los botones de los cuales consta el panel principal de la escena ***MainMenu*** son los siguientes:

-**Play**: este botón, abre un segundo panel llamado **Modo**, el cual se encuentra inicialmente deshabilitado, utilizando para ello el método **SetActive(False)**, de la clase **GameObject** de Unity. Para abrir Modo, el botón llama en **onClick()** a un nuevo método, realizando la llamada a SetActive(True), para el panel Modo, y

SetActive(False) para el panel principal. El panel Modo poseen dos botones, **Solo** y **Versus**, los cuales utilizan LoaderScene(String) para abrir la escena ***SelectedCharacter***, guardando a la vez en **PlayerPrefs** (un documento de tipo clave-valor, compartido por todas las escenas), en la clave **Modo** el nombre que llevan como texto dichos botones. De esta forma, en escenas posteriores se puede consultar el modo de juego que ha solicitado el jugador. Un botón **Back**, en el panel Modo permite realizar el proceso invertido al botón Play para volver al panel principal.

-Characters: Abre una escena llamada ***Characters***, parecida a ***SelectedChracters***, la cual aparece sin botón de seleccionar personaje.

-Controls: Abre un nuevo panel con información sobre los diferentes controles en ambos modos de juego. Este panel carece de lógica más allá de un botón que permite volver a mostrar solo el panel principal.

-Credits: Similar a Controls, mostrando el nombre del creador.

Por último, a parte de estos botones principales, la escena *MainMenu* cuenta con un botón situado en la esquina derecha inferior, que permite silenciar el sonido del videojuego.

6.2 Menú de selección de personajes

Tras tener el menú principal, se pasó realizar el menú de selección de personajes, qué, como ya se comentó en el apartado de planificación, será de tipo rotatorio. Apoyándonos en el segundo video de la bibliografía, se monta una primera escena, llamada *SelectedCharacter*, para el menú de selección. La escena al principio, como se comentó en su análisis, constaba solamente de tres objetos Capsule dispuestas en triangulo encima de un plano, Plane, con la cámara enfocando a la que se situaba en el centro.

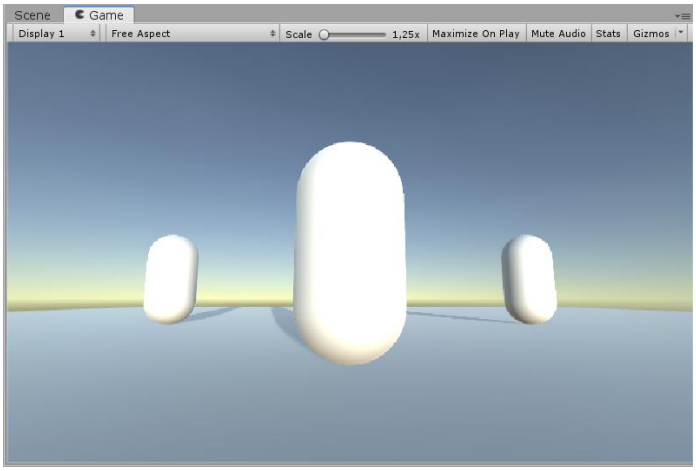

Imagen 11: Primer vistazo del menú de selección.

Después de organizar los elementos de esta escena, se empezó a programar el script, SelectionManager, que irá asociado al objeto Selección, un objeto vacío que contiene a las otras capsulas. Su función será la de girar, girando a su vez, las tres capsulas, reconociendo las flechas de dirección horizontal o el clásico A/D, lo que se reconoce en Unity con Input.GetAxis("Horizontal"). Cuando se reconocen las pulsaciones de estas teclas, el ángulo del objeto Selección se modifica en 90 grados. Hay que destacar de este script, el uso de Mathf.LerpAngle(float, float, float), una función que te permite transitar de un ángulo inicial a otro final (los primeros dos parámetros), en un determinado tiempo definido como su tercer parámetro de tipo float. Con este método, se modifica el Update() para que según la tecla que se pulse, el ángulo de Selección se modifique en un sentido o en otro, siendo la variación de este en 180 grados, en el caso de que se esté en una de las cápsulas de los extremos y se pulse en la dirección contraria, pasando a transitar a la otra cápsula extremo.

Finalmente, se colocan los elementos básicos de la interfaz de usuario para este menú. Esta interfaz gráfica, se compone de un Button que permite volver atrás en las escenas, un texto, Text, que varía según el personaje que estemos visualizando para seleccionar, y el Button que permite al usuario pasar a la pantalla de juego, haciendo uso de un nuevo script *UISeleccion*, que contiene un método similar al de LoaderScene(String) que, seleccionando un personaje, guarda su nombre usando SetString(String, String) en PlayerPrefs, bajo la clave "character". Este botón solo se puede pulsar cuando el personaje se sitúa en primer plano, es decir, en términos de lógica, el método Update() del script anterior deshabilita el Button de selección, cuando él ángulo en y del objeto Selección no es 0, -90 o 90, dentro de un rango de ±5 grados.

También esta escena cuenta con 3 texto que muestran las estadísticas de los personajes en función del que se esté visualizando en primer plano. Las estadísticas que se muestra son la vida, Life, en concepto de número entero, la fuerza, Strength, como la media de los 4 parámetros que van a condicionar la fuerza de los golpes de cada personaje y la velocidad, Speed, que sigue la siguiente fórmula:

$$Velocidad = 100 - \left(\sum_{i=1}^{4} Tgolpe_i\right) * 10$$

Esta misma escena, se ha reutilizado para crear la escena *Characters*, accesible desde *MainMenu*, la cual se diferencia por tener y también la escena *SelectSecondCharacter*, la cual se puede encontrar tras elegir un personaje en *SelectCharacter* en la y haber elegido el modo Versus. Esta escena permite seleccionar un personaje al segundo jugador y se diferencia de *SelectCharacter* en que en vez de guardar el nombre del personaje en PlayerPrefs con la clave "character", utiliza una clave nueva llamada "secondCharacter", para diferenciar los personajes de cada jugador al instanciarlos.

Ambas escenas, *SelectCharacter* y *SelectSecondCharacter*. introducen en PlayerPrefs 6 nuevas claves de tipo String con el valor "No", llamadas "WinPlayer1", "WinPlayer2", "PlayerVictory", "WinEnemy1", "WinEnemy2" y "EnemyVictory", que permiten en las escenas de combate, contabilizar las victorias del contrincante, utilizando las del "Enemy", tanto para la inteligencia artificial del modo Solo, como al personaje del segundo jugador en el modo Versus.

6.3 Personajes

Como ya se ha adelantado anteriormente, para realizar la implementación de los personajes se han utilizado modelados ya creados que se han descargado de diferentes sitios web, en especial de Unity Asset Store y la página web de Mixamo, de los cuales también se han sacado la mayoría de las animaciones utilizadas.

A continuación, se presentarán los componentes y sus principales características de los diferentes personajes elegibles por el jugador y también del enemigo del modo Solo.

6.3.1 Personajes principales: Ethan, Atenea y Eve

Ethan, fue el primero de los personajes que se implementaron. Básicamente, se ha utilizado como modelo de referencia para crear los otros dos, debido a que su modelado fue el primero de los tres en encontrarse.

Entrando ya en materia, lo primero que se hizo para implementarlo, fue importar el modelo de Ethan y sus animaciones básicas de andar y saltar al proyecto.

Tras importarlo, se añade el modelado a una nueva escena vacía, en la cual se le añade un componente **CharacterController** de Unity, el cual se debe ajustar adecuadamente a las medidas del personaje y nos permite obtener información del propio como saber si está tocado el suelo.

Lo siguiente es, tal vez, lo más complicado de la implementación del personaje: añadir el script que permite controlar el movimiento del personaje. Este script lo llamaremos, también, ***CharController***, y lo primero que se le añade, es el código necesario para controlar la dirección y el movimiento en tierra, lo cual el jugador podrá realizar por medio de A/D (izquierda/derecha) o con las flechas para moverse sobre del eje x, utilizando nuevamente Input.GetAxis("Horizontal") y, además, un parámetro de tipo float **MoveSpeed**, en la cual permite modificar la velocidad de movimiento que tendrá el personaje. La multiplicación de estos dos valores, se le asignara a la componente **movement.x** que contiene el atributo **transform** del objeto GameObject.

Al ser un juego en el que el movimiento se realiza solamente en el plano XY, el movimiento en Z no debería considerarse, así que, para impedirlo, se ha utilizado el método **LookRotation(Vector3)** de la clase **Quaternion** empleando el propio atributo "transform.movement", haciendo así que el personaje cambie rápidamente de dirección en X cuando se desee.

Después de controlar el giro del personaje, se le añade la posibilidad de saltar a este. Para ello, se utilizará la barra de espacio o la tecla "W" para hacer que salte, lo cual se puede obtener en Unity como **Input.GetButtonDown("Jump")** para el "Space" o **Input.GetKeyDown(KeyCode.W)** para la segunda alternativa. El salto se realiza

asignando al personaje una velocidad asociada a una variable de tipo float en su componente **transform.movement.y**. Tras haber saltado, esta velocidad se va modificando con respecto a la gravedad, que se encuentra en una segunda variable, de modo que el salto quede más realista, utilizando también para ello, **Time.deltaTime** que proporciona Unity para despreciar las diferencias entre las velocidades que pueden proporcionar las diferentes máquinas en las que se ejecute el videojuego.

Una vez el personaje ya tiene movimiento, lo siguiente que se incluye son las animaciones. Para ello se crea un nuevo **AnimatorController**, y en la ventana de **Animator** se le añaden las tres animaciones necesarias que va a tener de momento Ethan, el salto, correr y estar de pie. Estas animaciones transitaran de unas a otras con la ayuda de dos parámetros que he definido, **Speed** (Float) y **Jumping** (Bool), los cuales son modificados en el script de movimiento del personaje, por ejemplo, cuando el jugador pulsa espacio, donde Jumping pasa a ser True, pasando del estado que estuviese, al de salto.

Los golpes del personaje se añaden utilizando un recurso que no se había utilizado anteriormente, la **Coroutines** de Unity. Estos métodos funcionan a modo de hilos **Thread**, permitiendo ejecutar regiones de código separadas del código principal y empleando métodos de tipo **IEnumerator()**, en los cuales, usando **yield return new WaitForSeconds(float)**, se ejecutará el resto del código pasados unos segundos, dados por el parámetro float que se le pase. Para iniciar una Coroutine, se utiliza **StartCoroutine(IEnumerator())**.

Dicho esto, para los golpes se han realizado modificaciones en el Animator, creando así 4 nuevos estados con 4 parámetros propios de cada estado, los cuales hacen referencia a los 4 golpes que tendrá cada personaje. En el caso de Ethan, las animaciones que realiza cuando ejecuta el jugador un golpe se han descargado del paquete "FigthingMotinosVolume1" de la Asset Store de Unity.

Para controlar el resto del movimiento del personaje mientras se da un golpe, se utiliza una variable de tipo bool, llamada "**golpe"**, para impedir que el personaje realice un golpe o se mueva mientras está realizando un golpe ya.

El funcionamiento de las Coroutines de cada golpe tiene una estructura similar en todas, que queda plasmado en el siguiente seudocódigo:

Si se ha pulsado Tecla_GolpeX[1]:

· *Parámetro_GolpeX[2] = true, se ejecuta la animación*

· *golpe = true*

· *Esperar Tiempo_de_espera_GolpeX[3]*

[1] Tecla_GolpeX: tecla que acciona el golpe del personaje.
[2] Parámetro_GolpeX: parámetro del Animator que permite accionar la animación del golpe escogido.

· Si se encuentra dentro de rango y sigue vivo el enemigo: se le resta Daño_GolpeX[4] a su vida

· Parámetro_GolpeX = false, termina la animación, vuelve al estado Idle (estar parado)

· Esperar 0.05 segundos

· golpe = false

Restar la vida al enemigo se realiza utilizando un **SendMessage("HurtLife", Daño_GolpeX)**, lo cual ejecuta el método **HurtLife(int)** del enemigo, que le resta a la vida el número entero que se le haya pasado como parámetro. Junto a este método, se ha añadido otra animación al personaje que se lanza dentro de HurtLife(int), de forma inhibe cualquier otra animación, incluso las de los golpes, para hacer que el personaje reaccione al golpe que el enemigo le ha realizado. Para ello, se ha añadido un nuevo estado al Animator, el cual se llama Reached, con su parámetro específico y se ha utilizado una variable en el script, de tipo bool, llamada "**reached**" para que al igual que "golpe", bloquee

Por último, se le han añadido dos animaciones más al personaje, siendo una la que se ejecuta cuando se gana una ronda, es decir, la vida del personaje enemigo es 0, y la otra cuando el propio personaje pierde la ronda. Para ello se ha procedido de la misma forma que con los golpes, creando dos nuevos estados en el Animator, con sus propios parámetros dentro de este para controlar las transiciones, y el bool que bloquea el movimiento del personaje mientras se realiza alguna de estas animaciones.

Así queda finalmente el Animator de Ethan:

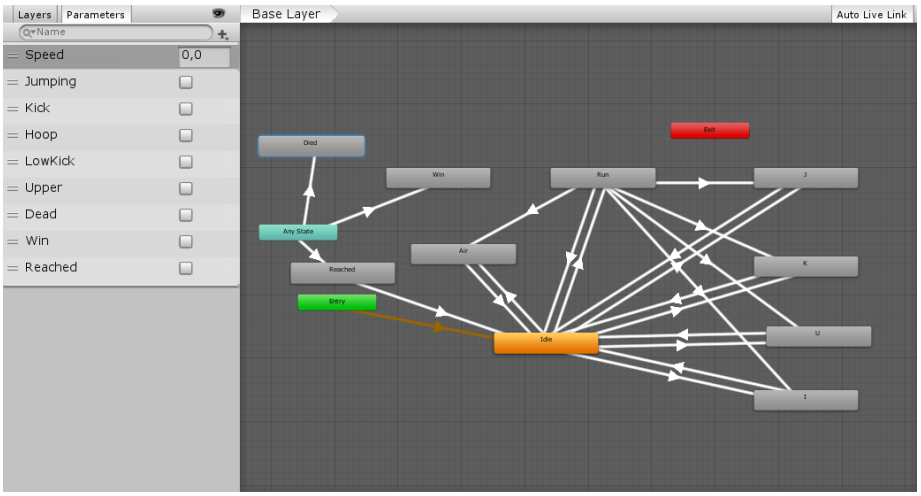

Imagen 12: Animator final de Ethan.

[3] Timepo_de_espera_GolpeX: variable de tipo float que indica el tiempo que dura en realizar la animación el personaje.
[4] Daño_GolpeX: variable de tipo int que corresponde a vida que se le resta al personaje contrario cuando es alcanzado por el golpe.

Para añadir a Atenea y Eve se ha realizado el mismo proceso que con Ethan, reutilizando el mismo script para su movimiento y cambiando los parámetros de los golpes mediante la interfaz de Unity, la cual posibilita modificar los valores de las variables que se han declarado como públicas desde la misma. Al igual que su composición como personajes elegibles, los Animator de las dos también son duplicados del de Ethan, cambiando en cada una las animaciones tanto de los golpes, como de las diferentes acciones de estas.

Finalmente, para habilitar los personajes para instanciarlos cuando sean escogidos, se ha creado como objetos **Prefabs**, un tipo de objetos que Unity permite instanciar mediante script con el método **Instantiate(String)**, donde el String que se le pasa como parámetro corresponde al nombre que tenga el Prefab a instanciar en el proyecto.

6.3.2 Personaje enemigo: Derrick

Derrick es el único de los 4 personajes que se han incorporado a este videojuego, que sigue una implementación diferente a los demás. La gran diferencia que tiene Derrick con el resto de personajes, es que es un personaje enemigo, el cual no puede ser controlado por el jugador y solo se puede jugar contra él, en el modo "Solo".

Para empezar, se importa el modelado a una escena en la cual, se le añaden componentes de Unity como el CharacterController, que ya habíamos colocado anteriormente en los otros personajes y, además, un componente **RigidBody**, que permite someter a cualquier GameObject de Unity a los cálculos del motor físico y un **NavMeshAgent**, un componente indispensable para crea inteligencia artificial el cual nos proporciona Unity. A este NavMeshAgente, contiene una variable "**destination**", a la cual le podemos pasar la posición del personaje controlado por el jugador, para que de esa forma le persiga.

Por último, se le añaden a Derrick dos componentes más, su propio Animator, el cual contendrá una máquina de estados con animaciones propias de Derrick, y un script ***EnemigoController***, el cual se encargará de controlar los movimientos y las acciones de Derrick.

El Animator de Derrick es más sencillo que los anteriores, pues básicamente cuenta con 4 estados fundamentales, las cuales corresponden a: la animación del zombie corriendo, la de atacar, y dos animaciones extras del zombie cayendo derrotado. Además, posteriormente se han añadido también, una animación Idle, para el zombie estando parado en una misma posición, y una animación para cuando Derrick gana una ronda, igual que se hizo con los otros personajes.

En cuanto al script *EnemigoController* que Derrick lleva, como se ha mencionado, en todo momento en su Update(), el "NavMeshAgent.destination", es la posición del jugador. Esta posición se obtiene gracias a que los personajes del jugador se instancian desde su Prefab dinámicamente, es decir, desde un script, al empezar una partida, añadiendo un **Tag** a estos, llamado "**Player**". Con esto, Unity permite buscar mediante

la clase GameObject, con el método **FindObjectWithTag("Player")**, el objeto que haya en la escena con el Tag "Player". De esta forma, podemos recuperar una referencia al personaje del jugador para obtener información como la posición o si aún sigue vivo.

Así, Derrick se dirigirá hacia el jugador en todo momento, y cuando se encuentre a una distancia considerable, la cual se puede medir utilizando **Vector3.Distance(Vector3,Vector3)**, que mide la distancia entre dos puntos de la escena, se detendrá y atacará al personaje del jugador. De esta forma, se realizará una llamada SendMessage("HurtLife", int), en caso de que haya conseguido impactar su golpe con el otro personaje, al igual que hacían el resto. En cuanto al resto de animaciones, se ejecutarán vía script cuando Derrick se quede sin vida, o el jugador sea abatido por el mismo.

Para luego poder instanciar a Derrick en la ***GameScene***, escena que se utilizan en el combate del modo "Solo", se hace un Prefab de este, al igual que con el resto de personajes.

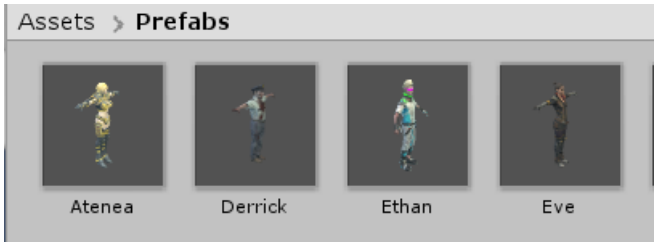

Imagen 13: Los cuatro personajes ya creados como Prefabs.

6.4 *GameScene* y *MultiplayerScene*

GameScene y *MultiplayerScene*, son dos escenas de Unity muy parecidas en las cuales transcurren respectivamente los combates del modo "Solo" y "Versus".

Estas escenas están formadas ambas por un Plane de Unity a modo de suelo, junto con dos **Cubes** estirados y colocado en los laterales del Plane, a modo de pared invisible, para impedir a los personajes salirse del mapa. También tiene un elemento Canvas, que como ya se ha mencionado anteriormente, contiene a los elementos de la interfaz gráfica.

6.4.1 Interfaz de usuario

Generalmente, la interfaz gráfica contiene un botón con título "**Pause**", que permite detener la partida cuando se pulse, utilizando **Time.scaleTime = 0**, y mostrar de igual forma que en el menú principal, un Panel con diferentes botones, permitiendo así la navegación a cualquiera de los menús anteriores a esta escena (*SelectedCharacter, MainMenu, etc.*). Además, contiene dos **ScrollBars**, junto con dos Text sobre estas y 2

Image debajo de cada ScrollBar. Estos elementos, hacen referencia a la información de los dos personajes que se encuentren combatiendo.

Las ScrollBars, son la vida de cada personajes, las cuales se ven modificadas por el método Update() del script ***UIGameScene*** utilizando las variables **Life** y **MaxLife** de cada personaje, a modo de fracción, es decir:

$$ScrollBar1.size = \frac{Char.Life}{Char.MaxLife}$$

Para obtener estos dos atributos de cada personaje, se obtiene una referencia a cada uno utilizando el método mencionado anteriormente GameObject.FindObjectWithTag(String), pasándole como String "Player", para obtener una referencia al personaje controlado por el jugador, en el modo "Solo" o el controlado por el primer jugador, en el modo "Versus", y "Enemy", para obtener una referencia a Derrick, en el modo "Solo", o una referencia al personaje del segundo jugador, en el modo "Versus"

El Text de cada una de ellas, indica el nombre cada personaje. En caso de que el modo "Versus" se elijan los dos personajes iguales, el del segundo jugador, llevara "Second" delante del nombre, para así diferenciar uno de otros.

En cuanto a las 2 Image que hay debajo de cada ScrollBar, hacen referencia al número de rondas que se hayan ganado por parte de cada jugador. Estas, inicialmente, se encuentran ocultas, de modo que utilizan las claves de PlayerPrefs "WinPlayer1", "WinPLayer2", "WinEnemy1" y "WinEnemy2", que se han mencionado en el apartado 6.2, para hacerse visibles. En el momento en el que "VictoryEnemy" o "VictoryPlayer" se vuelva True, las escenas se detienen, dando la victoria a uno de los dos personajes.

6.4.2 GameController y MultiplayerController

El ***GameController***, es tanto un objeto, como un script incluido en el mismo GameObject, que comparten ambas escenas, pasándose a llamar en la *MultiplayerScene*, ***MultiplayerController***, para diferenciarlos.

Hablando de los scripts solo, pues los GameObject són Empty, es decir, objetos vacíos, estos scripts se encargan tanto de instanciar los personajes utilizando los Prefabs ya hechos, como de modificar el mapa teniendo en cuenta el que haya sido elegido en la escena ***MapSelector***.

Ambos scripts, utilizando 3 variables con **[SerializedField]**, para guardar las referencias a los Prefabs de los personajes jugables. En el caso del *GameController*, tiene una más para Derrick, y el *MultiplayerController*, tiene 3 más para los personajes con los que puede jugar el segundo jugador, pues los Prefabs que se utilizan el modo "Versus", son diferentes a los usados en el modo "Solo", debido a unas diferencias en los scripts que controlan a los personajes.

Además de estos 4 o 6 campos con [SerializedField], también contienen ambos 6 campos más, que hacen referencia a cada uno de los posibles escenarios que pueden ser elegidos por el usuario.

Con estas referencias ya asignadas en cada script, *GameController* y *MultiplayerController*, hacen uso de PlayerPrefs para saber que deben instanciar en cada partida. Para el caso del personaje del jugador, en el modo "Solo", o el personaje del primer jugador, en el modo "Versus", se consulta la claver "carácter". En el caso del *MultiplayerController*, además, se consulta la clave "secondCharacter", para saber qué personaje ha sido escogido por el segundo jugador en la escena *SelectSecondCharacter*. Por último, en ambos scripts se utiliza la clave "**map**", instanciar por medio de **RenderingSetting.skybox**, el escenario que haya sido seleccionado en la escena *MapSelector*.

6.4.3 Cámaras

Por último, que aclarar el funcionamiento de la cámara, el cual es distinto en cada escena.

En la *GameScene*, la cámara es bastante sencilla, pues se dedica a perseguir al jugador llevando un ligero retardo que suaviza su movimiento. Esto se consigue cogiendo la posición el GameObject con Tag "Player" y pasándosela al "transform.position" de la cámara, utilizando el método **Vector3.SmoothDamp(Vector3,Vector3,Vector3, float)**, pasándole en los dos primeros Vector3, la posición en la que se está y a la que se quiere pasar, una velocidad como Vector3 en el tercer parámetro, siendo esta la velocidad de transición entre una posición y otra, y un último parámetro de tipo float, haciendo referencia al nivel de suavidad de la transición.

Finalmente, en la escena *MultiplayerScene*, se ha aplicado esto mismo, pero variando la posición objetivo, pues en la *GameScene* solo se tenía en cuenta al jugador. Básicamente, la posición objetivo de la cámara responde a los siguientes cálculos:

$$distancia = Vector3.Distance(player.position, enemy.position)$$

$$posiciónObjetivo = personaje.position + distancia/2$$

En la fórmula anterior, "personaje.position", hace referencia a la posición del personajes que más a la derecha se encuentre en todo momento, mientras que en la primera, se utiliza en método que se ha mencionado antes, para calcular la distancia entre dos puntos de la escena, en este caso, entre los dos personajes. De esta forma, la cámara apuntará en todo momento al punto medio entre los dos jugadores, enfocando de igual forma a ambos.

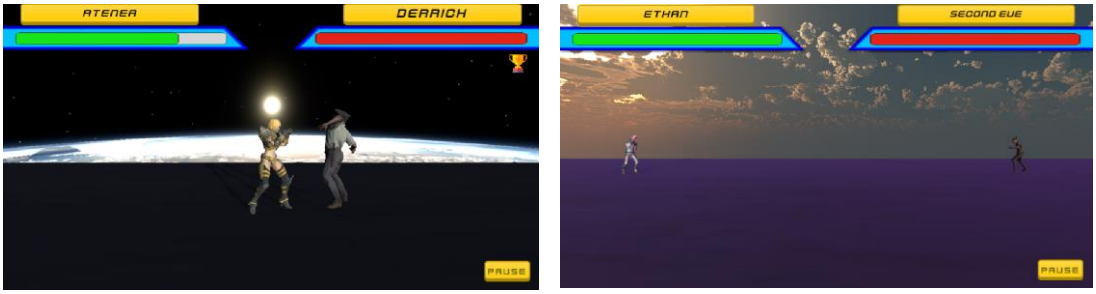

Imagen 14: GameScene y MultiplayerScene finales.

6.5 Menú de selección de escenarios

La escena *MapSelector*, que contiene el menú de selección de mapas o escenarios, fue, como se ha adelantado en su apartado de análisis, la última escena que se añadio en "Victory Royale".

Esta escena cuenta solamente con una interfaz de usuario, sin tener ningún otro GameObject fuera de la capa de la interfaz.

Lo primero que se realizó, fue introducir, como en todas las interfaces, el Canvas. Dentro de él, se han dispuesto posteriormente 6 botones, que harán referencia a cada uno de los mapas, y 2 botones más que facilitarán la navegación entre escenas, **Select**, para avanzar a la escena del combate, y **Back**, para retroceder a la escena anterior.

Cada uno de los 6 botones de los escenarios, contienen además un nuevo script llamado ***MapSelect***. Estos scripts funcionan a modo de clase, para almacenar entre otros el nombre de mapa, para mostrarlo en el texto de botón, o el material que permite cambiar el escenario. De esta forma y con este script, cuando el usuario pulsa uno de estos botones, se lanza el método **OnClickMap()** que contiene este script, el cual permite cambiar con RenderSettings.skybox la **skybox** de la escena actual, y mostrarlo antes de pasar a la escena de combate.

Otra cosa que realiza el método OnClickMap(), es modificar la variable **name**, de tipo String, del objeto con el script ***MapSelectorController***, sobrescribiéndola con el nombre del mapa del botón pulsado. Esta variable, es la que se almacena en el PlayerPrefs con la clave "map", y permite en las escenas de combate, cambiar el escenario según se haya elegido. Además, si esta variable es nula, es decir, no tiene ningún valor como por defecto, el botón Select de la escena *MapSelector*, permanecerá inhabilitado, pues no se habrá seleccionado ningún mapa aún. Todo esto, se controla desde *MapSelectorController*, donde, además, también se consulta el modo de juego, con la clave "Modo" de PlayerPres, para saber si el botón Back y el botón Select, llevarán al usuario a la escena *SelectSecondCharacter* y *MultiplayerScene*(en modo "Versus") o *SelectCharacter* y *GameScene* (en modo "Solo"), respectivamente.

Imagen 15: Escena MapSelector final.

7. Pruebas

Después de haber completado la implementación, se debe de probar el videojuego, y someterse a varios estudios tan por parte del desarrollador como por parte de terceros, para comprobar que la implementación se ha completado correctamente, y en caso contrario, solucionar los problemas que puedan haber aparecido mientras se prueba.

En este apartado, se detallarán las pruebas que se han realizado tras finalizar el videojuego y los problemas que han aparecido a raíz de estas.

7.1 Pruebas Alpha

Este tipo de pruebas son realizadas por el programador, o desarrollador del videojuego, y son un tipo de pruebas que se realizan, mayoritariamente, enfocadas en un aspecto en concreto del videojuego. En este caso, se ha realizado dos tipos de pruebas: las del movimiento de los personajes y la de los ataques de los personajes.

7.1.1 Pruebas de movimiento

Estas pruebas se han realizado tanto en el modo de combate “Solo”, como en el modo “Versus”. Tratan de probar que el personaje responde adecuadamente al control del usuario.

En estas pruebas se ha detectado que los personajes controlados por el jugador, tenían la posibilidad de atravesarse, como si no tuviesen ningún control por parte del motor físico. Esto es debido a que contienen un componente CharacterController de Unity, y no tienen un componente RigidBody, el cual permite el control por parte del motor físico.

Si a los personajes se le introdujese un RigidBody como sugiere el problema, el resto de elementos del escenario, como lo son el suelo, también deberían de portar este componente, lo cual haría que el suelo se sometiese a la gravedad, y cayese dentro de la escena junto con los personajes. Por esto mismo, lo personajes no cuentan con ningún componente RigidBody, por lo que se deberá de solucionar por medio de script.

7.1.2 Pruebas de ataques

Las pruebas de ataques, como bien indica su nombre, se han realizado para probar la funcionalidad de los golpes. Gracias a estas pruebas se han podido detectar una serie de problemas que vamos a detallar a continuación.

La primera de ellas tiene que ver con un término muy empleado en el mundo de los videojuegos, y no es otro que el "balaceo" de los personajes. Debido a los movimientos y sus parámetros, que, como ya se ha comentado en el apartado de análisis, cada personaje tenia asociados sus propios tiempos y daños para cada ataque, existía un "desbalance" entre los personajes, es decir, algunos contaban con una combinación tiempo-daño, que les hacía tener ventaja sobre los otros.

En "Victory Royale", sus personajes han sido implementados de manera que, si reciben un golpe, pasan a estar en el estado "reached", de forma que este estado, bloquea por completo el resto de acciones que puede realizar el personaje alcanzado hasta que deje de estar en el estado "reached", lo cual pasa tras unos instantes de tiempo. Si un personaje tiene muy bajos los tiempos de ataque, y a la vez, mucho daño en sus ataques, será capaz de tumbar a otro personaje sin apenas haberle dejado realizar algún movimiento. Este problema ocurría en la primera versión de "Victory Royale", por lo que se debía corregir y "balancear" los personajes, con tal de hacer el videojuego más justo.

El segundo problema que surgió durante estas pruebas, era que cuando un personaje era cortado o interrumpido realizando una animación de un golpe, es decir, que estaba realizando un golpe, pero en ese momento el enemigo le impacta antes y le interrumpe el suyo, pasando al estado "reached" y a realizar su animación correspondiente, tras este caso, el personaje interrumpido realizaba la operación de restarle vida al enemigo igualmente, pues, al parecer, y tras varias pruebas, se concluyó en que la animación se cortaba, pero la Coroutine que llevaba asociada su golpe no.

La solución a todos estos problemas se explicará en el apartado de **Resultados**.

7.2 Pruebas Beta

Por último, queda explicar los problemas obtenidos durante la fase de pruebas beta. Lo primero, las pruebas betas son un tipo de pruebas realizadas con posibles usuarios finales del videojuego. En este caso, se han utilizado un total de 3 usuarios para probar el videojuego.

Durante estas pruebas, se han detectado dos errores importantes:

El primero de ellos es un problema en la navegación entre las escenas. Al finalizar la fase de implementación, tras añadirse elementos de sonido y música, se añadió la posibilidad de emplear la navegación utilizando las teclas "Enter" y "Esc" para ir hacia delante y hacia atrás, respectivamente.

En la escena del menú principal, esta navegación no funcionaba correctamente, pues al abrir un Panel, como el del botón Credits o Controls, si se pulsaba "Esc" el juego era cerrado. Esta función debía de ser operativa solo cuando el menú principal no tuviese abierto ningún Panel, pues en dicho caso, solo se debía cerrar ese mismo Panel.

El último error del cual se informó por parte de los usuarios, era que en el modo "Solo", el jugador podía colocarse sobre Derrick de un salto y este, aparentemente,

seguir avanzando sin moverse del sitio, es decir, la animación de correr de Derrick se seguía reproduciendo aun estando estático. Esto se debe a que, como se ha comentado en su implementación, la animación por defecto de Derrick es esta, la de correr, pues carece de animación de estar quieto, como el resto de personajes.

8. Resultados

En el capítulo anterior, se han explicado los problemas obtenidos a partir de las pruebas. En este capítulo se explicarán las modificaciones y los problemas que se han resuelto en "Victory Royale", como resultado de la fase de pruebas.

8.1 Problemas con el movimiento de los personajes

Como se ha explicado anteriormente, este problema viene a raíz de que los personajes principales, no contienen una componente RigidBody. Si el problema se quisiese resolver simplemente añadiendo dicha componente, el escenario entero debería a su vez llevar una componente RigidBody, pues sino, el personaje no detectaría el suelo y caería al vacío. De esta forma, se introduce en el punto anterior, que su única solución es corregirlo a través de script, es decir, por programación.

Para corregir este problema se van a realizar una serie de cálculos que utilizarán las posiciones de los dos personajes del combate y su respectiva distancia. Como se ha comentado en el apartado de **Implementación**, Vector3.Distance(Vector3,Vector3), permite calcular la distancia entre un par de puntos. Con este cálculo, utilizando en cada personaje una referencia al personaje contrario (utilizando para ello lo expuesto respecto a la búsqueda de objetos en escena por Tag de Unity), podemos saber en todo momento dentro del método Update(), la distancia a la que se encuentran ambos personajes. De esta forma, ponemos un mínimo de distancia de 1, valor que ha sido elegido mediante prueba y error, para que el personaje bloquee su movimiento en la dirección hacia el personaje enemigo, si se encuentra a esa distancia del enemigo.

Los cálculos para la dirección del enemigo se realizan de la siguiente forma:

$$vectorDireccion = posicionEnemigo - posicionPersonaje$$

$$direccion = \frac{vectorDireccion}{vectorDireccion.magnitude}$$

Las dos posiciones que aparecen en la primera fórmula, son la correspondientes a la posición en x, y, z de los dos personajes. Con su resta, se averigua el vector dirección del personaje enemigo. Este vector aquí es almacenado bajo el tipo Vector3, el cual tiene un atributo "**magnitude**", que permite devolver la longitud del vector. Dividiendo el vector por su magnitud, obtenemos la dirección del enemigo respecto a la del personaje en el que nos encontramos. En cuanto a nuestra dirección, nos la da el usuario al pulsar las teclas de movimiento.

De esta forma, cuando se vaya a reconocer la entrada del teclado por parte del usuario, se hará bajo una sentencia "**IF**" cuya guarda será:

Si !(miDireccion != direccionEnemigo && distancia <= 1)

A esto falta añadir, que solo se consulta para los cálculos la componente x de cada dirección, al moverse los personajes en el plano XY.

8.2 "Balanceado" de los personajes

Durante las pruebas de ataque, se descubrió que algunos personajes contaban con una ventaja respecto a los demás debido a su combinación características. Tras esto, se tuvo en cuenta el tiempo de las animaciones de cada personaje para ajustar su daño dependiendo de si duraban más o menos. Cuanto menos tiempo durara su animación, menos daño recibiría el enemigo tras el golpe. De esta forma, el daño para cada uno de los movimientos de cada personaje se ajustó finalmente así (los movimientos aparecen en función a los controles del primer jugador):

Ethan: J-20, **K**-20, **U**-50, **I**-50; **Media:** 35

Atenea: J-30, **K**-70, **U**-40, **I**-40; **Media:** 45

Eve: J-40, **K**-50, **U**-40, **I**-100; **Media:** 57,5

8.3 Problema del estado "reached"

Antes, se ha comentado que cuando un personaje era alcanzado mientras realizaba un golpe, la animación de su golpe era interrumpida, lo cual estaba bien, pero seguía causando daño al contrario, lo cual no debería de ocurrir. Esto era debido a que al ser el estado "reached" un estado con transición desde cualquier otro estado (esto es, desde el estado **AnyState** que crea por defecto Unity en el Animator), la animación sí que se interrumpía, no obstante, al ser controlado el golpe por una Coroutine, esta se seguía ejecutando apartada del código principal del script *CharController*.

Para que se interrumpir la Coroutine también, se tiene que detener está utilizando el método **StopCoroutine(Coroutine)** que proporciona Unity. Para ello, se crea una variable global "**co**", de tipo Coroutine, donde se guardará de forma global al script, una referencia a cada Coroutine de cada golpe. Es decir, anteriormente para iniciar una Coroutine, se han creado métodos de tipo IEnumerator, y se iniciaban utilizando StartCoroutine(IEnumerator()). Ahora, esa región del código principal, quedará de la siguiente forma: co = StartCoroutine(IEnumerator()).

De este modo, al principio del método HurtLife(int), el método que se lanza al ser alcanzado por un golpe, se lanzará **StopCoroutine(co)**, deteniendo cualquier Coroutine que pudiese estar activa en ese momento, e impidiendo dañar al enemigo.

8.4 Problema de navegación en MainMenú

Para resolver este problema, simplemente se ha empleado una variable de tipo bool llamada "**popUpOpen**" en el script *MainMenuManager*, que inicialmente tiene el valor False. Esta variable, se pondrá a True, en el momento que se habrá alguno de los paneles de Controls o Credits. Así, se informa de cuando un Panel está abierto, permitiendo que, si esta variable se encuentra con valor True, al pulsar el usuario la tecla "Esc", se tiene que ejecutar el código del botón Back, del panel correspondiente. De esta forma, se impide que el juego se cierre mientras este uno de los paneles abiertos.

9. Conclusiones

Con este último apartado, se concluye el proyecto. Haciendo repaso de los objetivos que han sido planteados al principio de este documento, solo queda completar el último de ellos, referido al concurso de la web www.minijuegos.com. En cuanto al resto, ha sido expuesta su resolución a lo largo de este escrito.

No obstante, el ciclo de vida de un videojuego no finaliza aquí, pues ahora queda mantener el soporte al juego por parte del desarrollador, incluyendo nuevos elementos que se expondrán a continuación.

9.1 Trabajo futuro

A continuación, se hablará del trabajo a realizar, a partir de esta primera versión de "Victory Royale":

1.- Como se ha expuesto en capítulos anteriores, uno de los principales objetivos del videojuego, era tener una gran variedad de jugabilidad, por medio de diferentes personajes. Es por ello, que, con solo 3 personajes, el videojuego no puede mantenerse con la suficiente variedad indefinidamente. Por ello, uno de los primeros trabajos futuros a realizar, será la inclusión de nuevos personajes, que doten al juego de una mayor variedad, a poder ser también, con un diseño propio, esta vez sin la reutilización de modelados.

2.-Los escenarios, actualmente, constan de un fondo elegible (entre seis diferentes), y un suelo básico. Otro trabajo a realizar posteriormente, es la inclusión de nuevos escenarios, así como de dotar tanto los nuevos, como lo antiguos, de diferentes elementos que los diferencien, más allá de simplemente su fondo, ya sean árboles, obstáculos, plataformas, objetos consumibles, etc.

3.-Por último, uno de los errores descubiertos durante la fase de pruebas beta, en concreto, el relacionado con Derrick, no ha sido resuelto en la fase de **Resultados**. Esto es debido a que, junto a este error, la inclusión de nuevos enemigos en el modo "Solo", se plantea como un nuevo trabajo posterior a añadir en "Victory Royale".

Bibliografía

Anexo