Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Nombre del Centro

Trabajo Fin de Grado

PROTOTIPO DE VIDEOJUEGO EN TIEMPO REAL DE COMBATE CUERPO A CUERPO EN 3D

Alumno: Hugo Romacho Ortega

Tutores: Prof. D. Juan Roberto Jiménez Pérez
Prof. D. José Negrillo Cárdenas
Dpto: Departamento de Informática

Septiembre, 2021

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Juan Roberto Jiménez Pérez, tutor del Proyecto Fin de Carrera titulado: **“Prototipo de videojuego en tiempo real de combate cuerpo a cuerpo en 3D”**, que presenta Hugo Romacho Ortega, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, septiembre de 2021

El alumno:

Hugo Romacho Ortega

Los tutores:

JIMENEZ PEREZ, JUAN ROBERTO (FIRMA)
Firmado digitalmente por JIMENEZ PEREZ, JUAN ROBERTO (FIRMA)
Nombre de reconocimiento (DN): c=ES, serialNumber=44259364G, sn=JIMENEZ, givenName=JUAN ROBERTO, cn=JIMENEZ PEREZ, JUAN ROBERTO (FIRMA)
Fecha: 2021.09.15 18:47:59 +02'00'

NEGRILLO CARDENAS JOSE - 77364887V
Firmado digitalmente por NEGRILLO CARDENAS JOSE - 77364887V
Fecha: 2021.09.15 18:34:57 +02'00'

Juan Roberto Jiménez Pérez y
José Negrillo Cárdenas

Agradecimientos

Este Trabajo de Fin de Grado supone el fin de varios años de gran esfuerzo y cierra una etapa que marcó mi vida.

Sin duda alguna, quiero agradecer el apoyo que me ha dado mi familia, mi novia Gema, quien además ha sido la tester oficial de este proyecto. Ellos me han dado la motivación que necesitaba para dar este pequeño gran empujón final.

Finalmente, quiero agradecer a todos los profesores que me han enriquecido con sus conocimientos y que han confiado en mí, sobre todo a mi tutores Roberto y José. Sin su paciencia, esto no habría sido posible.

Índice

Índice de figuras

Índice de tablas

1. Introducción

En este primer capítulo se va a realizar una introducción al proyecto. Se explicarán los aspectos generales del mismo.

1.1. Introducción al proyecto

Estamos en una era en la que la tecnología está presente en nuestras vidas a cada paso y cualquier forma de entretenimiento o diversión va ligada a la tecnología. No es difícil ver a niños y no tan niños con un móvil o una tableta en la mano jugando a los típicos videojuegos que podemos encontrar para dispositivos móviles o con consolas portátiles para jugar en cualquier parte.

Estamos en la época de la inmediatez y queremos poder jugar en cualquier momento, aunque esto signifique sacrificar la comodidad del hogar. Sin embargo, un gran número de personas combina estas opciones con juegos pensados para ordenadores de sobremesa o para consolas como PlayStation o Xbox, llegando incluso a preferirlo frente a otras opciones.

Además, con el acceso globalizado a Internet, una gran parte de los videojuegos de hoy en día permiten jugar en tiempo real con personas que están a miles de kilómetros de distancia. Esto alimenta aún más la necesidad de la inmediatez, ya que no hace falta hacer amigos en la vida real para tener a alguien con quien jugar, sino que solo se necesita buscar partida para que el videojuego te empareje automáticamente con jugadores totalmente desconocidos. Esta modalidad crea una inhibición del jugador con el mundo exterior y reniega de una de las características más importantes del ser humano: la capacidad de relacionarse.

Y es aquí donde se enmarca este proyecto, en los usuarios que siguen prefiriendo reunirse con su familia o amigos para jugar unas partidas en el PC. En esos usuarios que siguen apostando por el contacto humano y pese a tener muchas opciones siguen eligiendo el modo multijugador.

En el caso de este proyecto, nos vamos a centrar en el género de combate. Este género es bastante popular, entre otras razones, porque desde sus principios ha proporcionado la posibilidad de jugar entre dos personas. La adrenalina del combate y los piques constantes entre los jugadores son las principales razones que hacen este género tan popular y por las que hoy en día todavía se siguen creando modos de multijugador en local.

Una de la serie de juegos más populares de este género hoy en día es Super Smash Bros (Nintendo, 2021). Estos juegos se utilizan en convenciones de todo el mundo para que el público que asista a dichos eventos encuentre una forma de entretenimiento que pueda compartir con otra gente aficionada. La sensación de demostrar un nuevo combo o un nuevo estilo de juego con otros jugadores en persona, e incluso en ocasiones con público, es única. Algunos de estos momentos han creado amistades que de forma *online* tal vez no se habrían dado nunca.

La mayoría de desarrolladoras de videojuegos opta por la creación de estos juegos en consola, ya que tradicionalmente se han llevado en estas plataformas. Además, muchas consolas incorporan dos mandos de serie, lo que hace más fácil la posibilidad de que jueguen dos personas. No obstante, la adaptación de estos títulos a PC no ha supuesto un problema,

ya que se pueden incorporar nuevos controladores de videoconsolas o específicos para PC. Para el desarrollo de este proyecto se ha optado por esta última plataforma, ya que la disponibilidad de un ordenador de sobremesa es mucho mayor para una persona que el de una consola, pudiendo llegar a un público más casual.

A la hora de escoger una herramienta que permita el desarrollo de este tipo de videojuegos, se pueden utilizar los motores de juego existentes en el mercado o se pueden crear motores propios. Por salirse fuera del ámbito de este proyecto, se descarta la primera opción, por lo que hay que estudiar las distintas opciones que ofrece el mercado de motores de videojuegos. A continuación, se va a hablar de los más conocidos: Unity3D y Unreal Engine.

Unity3D es un motor de videojuego creado por Unity Technologies que se caracteriza principalmente por hacer que el desarrollo de videojuegos sea lo más accesible posible, otorgando las herramientas necesarias. Este motor permite la creación de videojuegos en 2D y 3D, dando soporte también para realidad virtual. Una de las principales características de Unity3D es la posibilidad de desarrollar para un gran abanico de plataformas como Windows, Linux, Mac, tvOS, PS4, Xbox One, Android, iOS y WebGL.

Tiene una licencia gratuita (exclusivamente personal) y varias de pago dependiendo de los requisitos del proyecto y, en el caso de un equipo de desarrolladores, el número de trabajadores. La licencia gratuita es ideal para desarrolladores independientes, puesto que no hay que pagar a Unity3D por los juegos que se creen y pueden ser comercializados libremente. Además, este motor cuenta con una gran comunidad, desde foros dedicados con preguntas de todo tipo, a tutoriales paso a paso sobre los aspectos y dudas más comunes de los usuarios de Unity3D.

Por otro lado, tenemos uno de los motores de juegos más conocidos: Unreal Engine. Fue creado por la compañía Epic Games y es un conjunto de herramientas de desarrollo de aplicaciones gráficas en tiempo real. Al igual que su contraparte, ofrece el desarrollo de un gran conjunto de plataformas como Windows, Linux, MacOS, Nintendo Switch, PS4, PS5, Xbox One, dispositivos móviles e incluso aplicaciones basadas en realidad virtual.

Al igual que Unity3D, Unreal Engine permite desarrollar de forma gratuita. A la hora de comercializar un videojuego, no se paga nada hasta el primer millón de dólares de beneficios. A partir de dicho límite, la plataforma cobra un 5% del beneficio en concepto de royalties. Unreal Engine también tiene una gran comunidad dedicada, aunque menor que en el caso de Unity3D, y se caracteriza por tener una curva de aprendizaje más pronunciada.

Ambos motores son grandes opciones y permiten exportar el videojuego a distintas plataformas si así se desea en un futuro. No obstante, para realizar este proyecto se va a utilizar Unity3D por ser gratuito, así como para aprovechar los conocimientos obtenidos durante el Grado sobre este motor, expandir aún más mis habilidades en él y beneficiarme de la inmensa comunidad de desarrolladores que posee.

1.2. Propósito general

El propósito de este trabajo fin de grado fundamentalmente es estudiar a fondo los distintos motores de videojuegos, para llevar a la práctica lo aprendido en el desarrollo del videojuego propuesto. Esto permitirá entender en profundidad los distintos elementos de un videojuego, como se relacionan y el trabajo que hay tras cada pequeño detalle que lo forma.

Como ya se ha mencionado, en este trabajo se desarrolla un prototipo de videojuego cuerpo a cuerpo, lo que conlleva que será un juego multijugador (dos jugadores en concreto) que pelearán entre sí.

Sin embargo, este desarrollo no solo va a centrarse en este aspecto, sino que pretende conseguir un sistema de animaciones completo, fluido y flexible, que sea válido para los distintos modelos que se quieran introducir en el juego. Además, se velará para que el juego sea en tiempo real, consiguiendo así un combate fluido y jugable.

1.3. Objetivos

El objetivo principal del proyecto es diseñar e implementar un prototipo funcional de un videojuego. Para conseguirlo se han establecido ciertos objetivos fundamentales:

- Realizar una planificación del proyecto, identificando los puntos clave del desarrollo, analizándolos y establecer fechas de entrega.
- Identificar las particularidades del juego a desarrollar, destacando las características principales del mismo, así como el público objetivo.
- Definir las mecánicas del 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 tanto las animaciones como el comportamiento asociado durante el combate cuerpo a cuerpo.
- Analizar y diseñar estructuras de datos adecuadas para detectar el daño causado.
- 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.

2. Entorno de desarrollo

2.1. Tecnologías utilizadas

A continuación, se van a enumerar las distintas tecnologías utilizadas en el desarrollo de este proyecto y se detallará brevemente el uso de cada una.

2.1.1. Unity3D

Unity3D es la principal herramienta en el desarrollo de este proyecto. Este motor de juegos proporciona todas las herramientas necesarias para la creación de un videojuego completo, desde la creación de escenas, la programación del funcionamiento del videojuego, la personalización de las animaciones o la adición de un sistema de sonido entre otras. La versatilidad que ofrece Unity3D permite que este proyecto se pueda convertir en realidad.

2.1.2. C#

C# es un lenguaje de programación moderno, basado en objetos y con seguridad de tipos. Tiene sus orígenes en la familia de lenguajes C. Los programas creados con C# se ejecutan con .NET, un sistema de ejecución virtual y un conjunto de bibliotecas de clases. Este lenguaje de programación es el utilizado en los scripts de Unity3D. Sin embargo, este no es el único uso del lenguaje, pudiéndose usar para desarrollar aplicaciones de escritorio o aplicaciones web de ASP.NET.

Para el desarrollo de este proyecto, nos beneficiaremos de uno de los componentes de .NET, llamado LINQ. Este componente es un conjunto de librerías que, en este proyecto, facilita la consulta a estructuras de datos nativas de .NET, creando un código más limpio y legible.

2.1.3. Adobe Photoshop

Adobe Photoshop es un software creado por Adobe Systems Incorporated. Este programa conocido mundialmente permite fundamentalmente la edición de imágenes digitales. Su uso en este proyecto se llevará a cabo durante la parte de diseño, durante la cual se crearán los recursos visuales necesarios para la posterior inclusión en el videojuego, como pueden ser las imágenes estáticas que se pueden ver en toda la aplicación.

2.2. Equipo empleado

Para el desarrollo del proyecto se necesitará un ordenador que posea cierta potencia gráfica para poder llevarlo a cabo de forma fluida. Se utilizará mi ordenador personal como herramienta de trabajo, puesto que es el ordenador más potente que tengo a mi disposición. Sus características principales son las siguientes:

Tipo de componente	Especificaciones
Placa base	Gigabyte GA-AB350-GAMING 3
Procesador	Asus Dual GTX 1060 OC 6GB GDDR5
Memoria RAM	G.Skill FlareX DDR4 3200 PC4-25600 16GB 2x8GB CL14 Negra
Tarjeta gráfica	GTX 1060 OC 6GB GDDR5
Fuente de alimentación	Tacens Mars Gaming Zeus 750W 80 Plus Silver Modular
Almacenamiento	Samsung SSD 870 EVO 250GB
Disipador	Enermax ETS-N31
Caja	Cooler Master MasterCase Box 5 USB 3.0 Blanca con Ventana
Monitor	MSI Optix G24C4 23.6" LED FullHD 144Hz Freesync Curva
Ratón	Logitech G502 HERO
Teclado	ACGAM AG-109R

Tabla 1. Equipo empleado

Además, para poder habilitar las partidas entre dos jugadores, se va a contar con un mando tradicional de videoconsola. En este caso se va a utilizar el Dualshock 4 de la compañía Sony.

3. Ingeniería del software

En este apartado se muestra todo lo relativo a la ingeniería del software, como es la definición del sistema ante el que nos encontramos, sus requisitos y la planificación del mismo tanto desde el punto de vista temporal como de costes.

3.1. Definición del sistema

A continuación, se va a describir en detalle el alcance del sistema y la definición de los requisitos del sistema iniciales tanto funcionales como no funcionales

3.1.1. Alcance del proyecto

En este proyecto se va a desarrollar la funcionalidad de combate de los personajes. Esto engloba la inclusión de animaciones básicas para el movimiento del personaje, la implementación de un controlador genérico que permita el manejo de los mismos y el estudio y desarrollo de las estructuras de datos adecuadas para la detección de colisiones. Además, para crear una base para el desarrollo futuro del proyecto, se implementará un sistema de eventos que permitirá tanto a los programadores y los diseñadores la introducción de efectos.

Para permitir la jugabilidad de dos jugadores en local, se debe estudiar e implementar una solución moderna que permita el uso de distintos tipos de dispositivos. Para finalizar, los jugadores deben poder elegir un personaje de entre varias opciones y poder enfrentarse en combate con otro jugador.

Se ha partido de la base de que el objetivo del proyecto es realizar un prototipo de un videojuego y no un videojuego preparado para ser comercializado. Por esta razón, se van a dejar para un trabajo futuro todas las características comunes de un videojuego comercializable y preparado para el público, como puede ser un menú principal o una sección de ajustes.

3.2. Definición de requisitos funcionales y no funcionales

Este apartado engloba todo lo relativo a la definición de requisitos que deberá cumplir el sistema.

3.2.1. Requisitos funcionales

La especificación de requisitos funcionales de un sistema debería ser completa y consistente, de forma que el sistema esté definido de una forma clara y sin contradicciones. A continuación, están numerados todos los requisitos funcionales:

- Deben poder jugar dos jugadores en local, enfrentándose entre sí.
- El juego debe ser un combate cuerpo a cuerpo.
- Cada jugador debe poder escoger el personaje que quiera para jugar.
- Este prototipo debe contener al menos dos personajes jugables.
- Los jugadores deberán poder jugar al prototipo utilizando el teclado o un mando.
- El juego deberá permitir un juego fluido, por lo que será necesaria la implementación de un sistema de animaciones que será capaz de encadenar distintas animaciones

entre sí, de forma que entre ellas no queden tiempos muertos en los que el personaje no realiza ninguna acción, aunque sí está recibiendo eventos por parte del usuario.
- El juego debe mostrar una pantalla de fin de juego cuando el combate termine.
- Una vez acabe la partida, la aplicación debe volver a dar la posibilidad a los usuarios de iniciar una nueva partida escogiendo nuevos personajes.

3.2.2. Requisitos no funcionales

Analizando detenidamente las características necesarias para un adecuado funcionamiento del sistema, se proponen los siguientes requisitos no funcionales:

- El prototipo debe ser ejecutable en Windows.
- Al seleccionar un personaje, debe tardar menos de medio segundo en aparecer en pantalla.
- La detección de colisiones debe realizarse en tiempo real, sin que el rendimiento de la aplicación se va afectado en ningún momento.

3.3. Planificación

Este apartado está centrado en la planificación de las distintas tareas que se van a llevar a cabo en este trabajo de fin de grado para la consecución de los distintos objetivos clave en la finalización del mismo.

3.3.1. Metodología de desarrollo

En este proyecto se va a utilizar una metodología incremental de desarrollo. Esta metodología obliga al equipo de desarrollo a presentar un prototipo funcional de la aplicación al final de cada iteración. Por lo tanto, se ha pensado que este método puede ser el más adecuado para el desarrollo de un videojuego por el gran refinamiento que requiere este tipo de aplicaciones.

Cada uno de los prototipos funcionales será evaluado por el cliente final (en este caso, mi novia) y se determinará si el prototipo cumple con los requisitos. Durante estas evaluaciones el cliente puede puntualizar cualquier detalle o error que deba ser corregido en la siguiente iteración, pero siempre deben respetarse los plazos de desarrollo y los objetivos que deben ser cumplidos en cada iteración.

Estos son los objetivos a cumplir en cada una de las iteraciones:
- Diseño de los personajes y sus animaciones.
- Implementación del controlador de los personajes. Ajuste de las animaciones.
- Estudio e implementación de las estructuras de datos más adecuadas para la detección de colisiones. Ajuste de las animaciones.
- Diseño e implementación de la interfaz de la pantalla de selección de personaje.
- Implementación del modo multijugador.
- Prototipo de la escena de combate. Refinamiento de las animaciones.
- Mejoras visuales y prototipo final de la aplicación.

En la primera iteración se realiza el trabajo principal de la aplicación: el diseño y la implementación de las animaciones. Este trabajo se irá ajustando gracias a la evaluación de los prototipos a lo largo de las siguientes iteraciones hasta que se consiga la calidad deseada.

3.3.2. Diagrama de Gantt

Se ha realizado un diagrama de Gantt con la planificación inicial del proyecto. La duración total del mismo es de 13 semanas.

	Duración												
	Junio				Julio				Agosto				
Iteraciones	SEM 1	SEM 2	SEM 3	SEM 4	SEM 5	SEM 6	SEM 7	SEM 8	SEM 9	SEM 10	SEM 11	SEM 12	SEM 13
Diseño de los personajes y sus animaciones													
Implementación del controlador de los personajes.													
Estructuras de datos para la detección de colisiones.													
Interfaz de la pantalla de selección de personaje.													
Prototipo de pantalla de selección de personaje.													
Prototipo de la escena de combate.													
Mejoras visuales y prototipo final.													

Ilustración 1. Diagrama de Gantt

3.4. Estimación de costes

En esta sección se va a realizar la estimación total de los costes del proyecto. Esto incluye tanto recursos software, hardware como gastos de luz y personal.

Empezando por los recursos software, la única aplicación de pago que se ha utilizado en el proyecto es Adobe Photoshop. Adobe ofrece tanto planes mensuales como anuales, siendo estos últimos más económicos. El plan anual asciende hasta un total de 290.17 €, mientras que el mensual cuesta 36.29 € al mes. Según la duración total de 3 meses del proyecto dada por la planificación inicial, la opción más económica suscribirse durante todo el período de tiempo a la opción mensual, dando un total de 108.87 €. No obstante, este programa solo se va a utilizar durante la fase de diseño del proyecto, más concretamente en la fase de diseño y desarrollo de la interfaz de la pantalla de selección de personaje, la cual solamente dura una semana. Por tanto, podemos hacer uso de la suscripción durante un mes para ahorrar costes. En el caso que haga falta más adelante, se puede volver a suscribir a la aplicación. Por tanto, el coste final de este programa será de 36.29 €.

Programa	Coste
Unity3D	0 €
Adobe Photoshop	36.29 €
Visual Studio 2019	0 €
Coste total	36.29€

Tabla 2. Costes del software

A continuación, se van a calcular los gastos materiales del proyecto. Estos incluyen el coste del equipo, la electricidad o la conexión a internet. Comenzando por la electricidad, se tendrá en cuenta un uso de 4 horas al día del equipo durante todo el período de la planificación del proyecto. Para calcular de forma exacta el consumo energético del equipo, haría falta tener un medidor físico. Como no se dispone de tal instrumento, se ha optado por realizar un cálculo aproximado utilizando una herramienta online. La potencia final que la herramienta ha calculado para el equipo es de 289W. Considerando que solo se trabaja de día, no hay

consumo de iluminación. Teniendo en cuenta que el precio actual del kW·h es de 0.14933 €/kW·h, el precio aproximado final del consumo del equipo es de 15.71 €.

El siguiente gasto fundamental es el de la conexión a Internet. La tarifa contratada durante el desarrollo de este proyecto es de 39`95 €/mes. Por lo tanto, el coste durante los 3 meses de desarrollo es de 119.85 €.

El coste del equipo de desarrollo utilizado es aproximadamente de 1231.38 €, sumando ordenador, monitor y periféricos. Finalmente hay que sumar el gasto en una libreta para el proyecto, dos bolígrafos, un lápiz y una goma de borrar, dando un total de 8.1 €.

Concepto	Coste
Consumo energético	15.71 €
Consumo de Internet	119.85 €
Equipo de desarrollo	1231.38 €
Papelería	8.1 €
Coste total	1375.04 €

Tabla 3. Costes materiales

Para finalizar esta sección, se va a hacer cálculo de los costes humanos del proyecto. Cabe destacar que el autor de este proyecto no ha realizado siempre tareas de programador, sino que también ha habido que realizar trabajos en Adobe Photoshop, que suele ser tarea del diseñador. Para el cálculo de estos costes se tendrán en cuenta las horas que el autor ha empleado en cada parte del proyecto. Los salarios utilizados en esta memoria son completamente orientativos y han sido obtenidos de Jobted.

Para el puesto de programador de videojuegos, el salario medio es de 15.2 €/hora. Para un diseñador de videojuegos, la paga media es de 14.7 €/hora. Con estos salarios, podemos calcular el coste total.

Puesto	Horas	Coste
Diseñador	182	2675.4 €
Programador	182	2766.4 €
Coste total	364	5441.8 €

Tabla 4. Coste del personal

Por tanto, el presupuesto final para el desarrollo del prototipo es de 6853.13 €.

4. Desarrollo del videojuego

En este capítulo se hablará de todos los aspectos relacionados con el desarrollo del videojuego desde que solo era una idea hasta los resultados finales del mismo.

4.1. Visión general

Se empezará dando una visión general sobre el videojuego. Llegados a este punto se van a explicar con más profundidad los detalles genéricos del proyecto.

4.1.1. Propósito del juego y audiencia

El propósito final del juego es derrotar al contrincante. La intención es que el jugador aprenda que cada personaje presenta sus ventajas y desventajas y debe desarrollar una estrategia de combate adecuada a su personaje para lograr derrotar a su enemigo. De la misma forma, también debe aprender que el personaje enemigo tiene tanto puntos fuertes como débiles y debe aprender a aprovecharlos.

A través de la variedad de movimientos implementados en los personajes, se pretende proveer tanto de estrategias agresivas como más defensivas. Se ha realizado una cuidadosa implementación de encadenamiento y cancelación de animaciones que dan la libertad al jugador de poder expresar su estilo de juego.

Este prototipo va dirigido a aquella audiencia que disfruta jugando en compañía y busca un entretenimiento divertido y desafiante. Se ha realizado el juego con el fin de promover el modo multijugador local. Siguiendo la guía de PEGI (PEGI, 2021), al ser un juego de lucha que contiene violencia que se asemeja a la realidad, la edad mínima de juego estaría en los 16 años. Además, otros juegos de una naturaleza similar del propuesto en este trabajo de fin de grado comparten el mismo límite de edad, por lo que estaría en consonancia con el resto de juegos que hay en el mercado.

4.1.2. Mecánicas del juego

En este apartado se encuentran las distintas mecánicas de juego definidas en profundidad, como son el objetivo de los jugadores, la condición de victoria y derrota, las mecánicas de los jugadores, entre otros.

4.1.2.1. Objetivo de los jugadores

El principal objetivo de cada uno de los jugadores es derrotar al personaje del otro jugador. Todas las batallas se desarrollan en una playa. Los personajes de los jugadores se sitúan a una distancia inicial, y una vez aparezcan en pantalla, los jugadores pueden controlarlos con sus dispositivos correspondientes.

Los jugadores deberán hacer uso de los movimientos que consideren oportunos en cada momento para rebajar la vida del contrincante a cero.

4.1.2.2. Condición de victoria y derrota

Mientras se desarrolla la batalla, un jugador puede realizar un ataque ofensivo. Si este impacta sobre el enemigo sin que este se proteja o se evada del golpe, recibirá una cantidad de daño que se restará a la vida del jugador.

Se determinará que un jugador ha ganado la partida si consigue rebajar los puntos de vida del oponente a 0. En el caso contrario de que el jugador pierda todos sus puntos de vida, entonces este habrá perdido el combate

4.1.2.3. Mecánicas de los jugadores

Los jugadores podrán moverse exclusivamente en un eje, de esta forma siempre se encontrarán de frente al otro jugador. Cada jugador será asignado a un lado de la escena. Esto implica que el personaje del jugador solo podrá atacar frente a él y no podrá cambiar su orientación. Todos los personajes pueden correr, andar o quedarse en el sitio. Si se utiliza un teclado no se podrá andar, puesto que no hay pasos intermedios entre pulsar una tecla o no pulsarla. En el caso de que se utilice un dispositivo con joystick se podrá regular la velocidad con la que avanza el personaje, dependiendo de la inclinación que se le dé al joystick.

Cada jugador podrá atacar con su arma, realizar volteretas hacia adelante para acercarse al jugador enemigo y preparar un ataque, hacer una pequeña evasión hacia atrás o bloquear un ataque enemigo con su arma. Los movimientos defensivos podrán ser encadenados con un ataque, pero intentar realizar un movimiento defensivo después de otro será más lento que encadenar uno ofensivo.

4.1.2.4. Sistema de detección de daño

En este juego, cada uno de los modelos de los personajes está dividido en 3 partes: cabeza, torso y piernas. Cada una de estas partes puede recibir daño de forma individual, pero solo se aplicará el daño de la primera parte golpeada en un ataque. Por lo tanto, si un personaje es atacado y existe colisión en la cabeza y en el cuerpo, solamente contará el daño recibido en la primera instancia.

Cabe destacar que la cantidad de daño recibida varía dependiendo de la zona afectada. Un golpe realizado en la cabeza de un personaje restará siempre más vida que uno realizado en el cuerpo o en las piernas. De la misma forma, un golpe recibido en el cuerpo restará más vida que si se hubiera recibido en las piernas, pero menos que en la cabeza.

Esta mecánica permite más variedad a la hora de añadir nuevos personajes, ya que se puede añadir un personaje más lento, pero con una zona de la cabeza más pequeña, siendo menos vulnerable a un golpe crítico.

4.1.2.5. Selección de personaje

Cada jugador podrá unirse al juego con un dispositivo diferente al del otro jugador. Para unirse, los jugadores deben pulsar cualquier tecla o botón de sus dispositivos y un cursor aparecerá en la pantalla con su color correspondiente. Podrán mover el cursor libremente por su mitad de la pantalla y seleccionar el personaje que quieran. Ambos jugadores pueden elegir el mismo personaje. Para empezar la partida, cuando los jugadores estén listos deben pulsar el botón Enter o Start.

4.1.3. Flujo del juego

El juego comienza en la pantalla de selección de personaje. Cada jugador debe unirse a la partida y es libre de escoger el personaje que quiera. Una vez que los dos jugadores han escogido, cualquiera de ellos puede empezar la partida.

Tras iniciar la partida, los jugadores lucharán por ganar la batalla. Cuando uno de ellos gane y después de reproducir la animación del ganador, se volverá a cargar la escena de selección de personaje automáticamente para que los jugadores puedan volver a jugar.

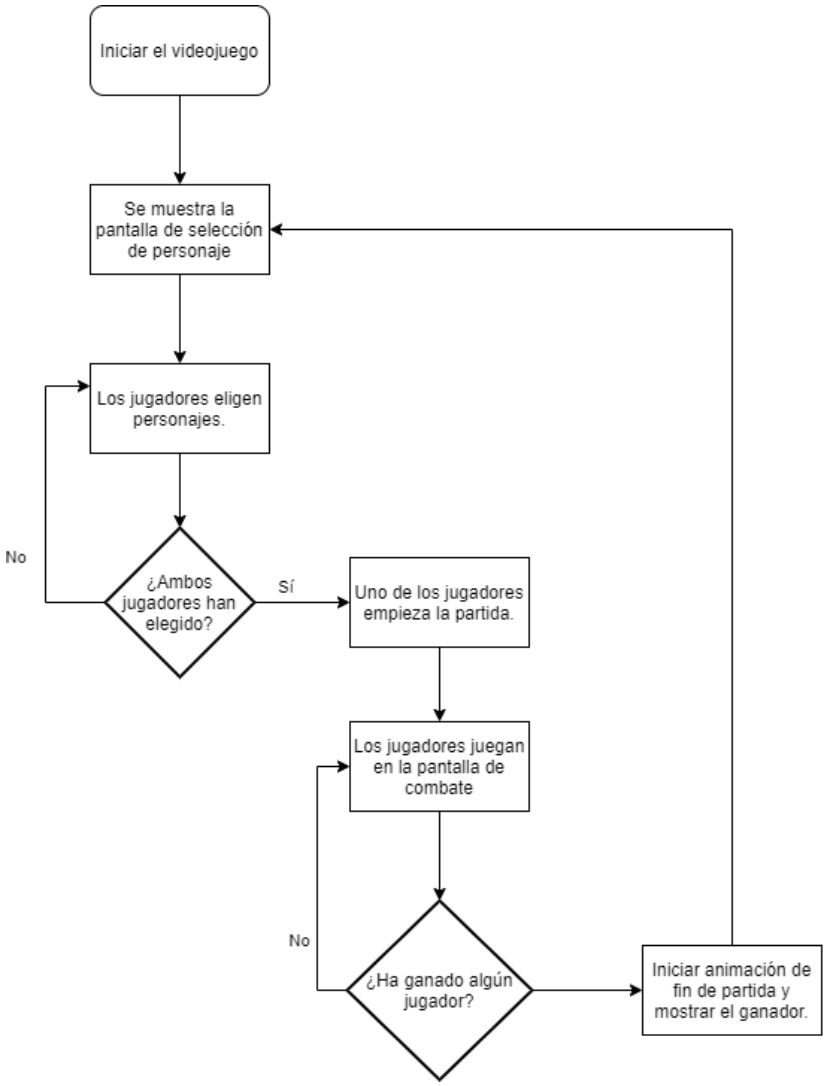

Ilustración 2. Diagrama del flujo del videojuego

4.2. Diseño del juego

4.2.1. Personajes

En esta sección se van a tratar los detalles específicos que se han tenido en cuenta para cada uno de los personajes que se han introducido en el juego y cómo ha sido su integración. Todos los personajes han sido obtenidos en Mixamo.

4.2.1.1. Paladín

Como primer personaje, se añadió Paladín. Paladín es un guerrero que lucha cuerpo a cuerpo con su espada. Se caracteriza por tener una agilidad ligeramente superior a la media y sus ataques rápidos le permiten encadenarlos fácilmente.

4.2.1.2. Mutante

Mutante se añadió al juego con la finalidad de añadir más variedad de personajes y proporcionar habilidades que Paladín no podía ofrecer. Este personaje se caracteriza por su gran envergadura. Utiliza su brazo mutado como arma y aunque sus ataques sean más lentos, gracias a su particular animación de ataque es capaz de asestar golpes en la cabeza a los personajes.

4.2.2. Animaciones

Uno de los retos que presenta este proyecto es proporcionar con facilidad a los diseñadores que puedan trabajar en un futuro en el desarrollo de este prototipo la posibilidad de añadir nuevos personajes con nuevas animaciones. Esto implica que un diseñador nunca debe tener que modificar código o el comportamiento del *Animator Controller* para añadir nuevos personajes. Por tanto, es responsabilidad del programador dejar preparada una estructura que abstraiga el comportamiento de los personajes, indistintamente de sus aspectos.

Otro reto principal ha sido proporcionar una jugabilidad fluida y satisfactoria. Esta parte es la más influyente en la experiencia del usuario, puesto que una jugabilidad mala hará que los usuarios no se sientan cómodos. Por este motivo se han realizado tantas pruebas como se ha considerado necesario para logarlo.

El mayor objetivo era ofrecer un encadenamiento de animaciones para darle a los jugadores una sensación de control sobre el personaje sin sentir que las animaciones sucedían de forma mecánica. Se han implementado mecanismos que permiten cancelar las animaciones si el usuario comanda otra acción sobre el personaje.

Todos los detalles sobre el desarrollo de las animaciones pueden verse en el apartado *Implementación*.

4.2.3. Escenas

4.2.3.1. Selección de personaje

Para el diseño de esta escena me he inspirado en otros videojuegos de la misma temática, como son *Tekken* o *Street Fighter*. En este caso se ha dividido la pantalla en dos para separar a los dos jugadores y se le ha asignado un cursor diferente a cada uno para que puedan elegir personaje. Cada uno de ellos está identificado con un color.

A cada lado de la pantalla se despliega una serie de tarjetas que se corresponden con los personajes jugables. Cada jugador solo puede mover su cursor por la mitad de la pantalla que le corresponde y puede seleccionar el personaje que quiera, sin importar que sea el mismo que ha elegido el otro jugador. Una vez seleccionado un personaje, éste aparecerá en su lado de la pantalla y realizará una animación de celebración. Una vez que los dos jugadores estén listos, pueden iniciar la partida.

4.2.3.2. Escena de batalla

En esta escena aparecen los dos personajes seleccionados por los jugadores en sus lados correspondientes. Aquí se han eliminado los cursores, puesto que no existen elementos de la

interfaz con los que se puedan interactuar. Cada uno de los personajes posee una barra de vida en la parte superior de la pantalla. La vida restante se marca en verde.

Los jugadores lucharán hasta que uno de los dos quede con cero puntos de vida. Cuando eso ocurra, la cámara realizará una pequeña animación para mostrar al ganador de la partida.

4.2.4. Interfaz de usuario

4.2.4.1. Selección de personaje

Antes de la implementación de las interfaces de usuario, se realizó un boceto para cada escena para facilitar la tarea del desarrollo posterior. En el caso particular de esta escena, el prototipo final se asemeja mucho al boceto a excepción de ciertos detalles. En el boceto se pensó añadir el nombre del personaje elegido en la parte superior de la pantalla. El texto no se añadió por su redundancia, ya que el nombre del personaje está escrito sobre la tarjeta que selecciona el usuario.

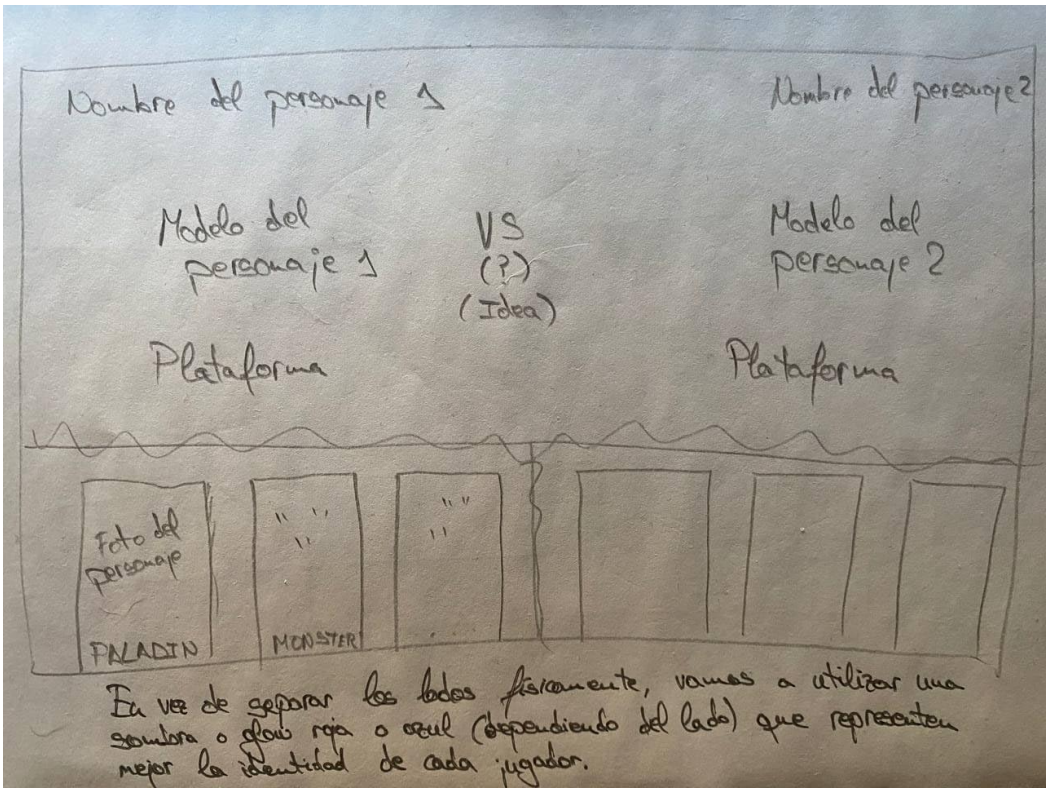

Ilustración 3. Boceto de la pantalla de selección de personaje

Como se puede observar en el boceto, la pantalla se divide en dos partes (una para cada jugador) y a cada jugador se le asigna un color representativo. Las tarjetas contienen una imagen estática de los personajes. Al diseñar esta interfaz, se pensó para que el jugador tuviera *feedback* por parte del videojuego a la hora de utilizarla. Por ello se han añadido dos efectos concretos a las tarjetas:

- ***Hover***: Al colocar el cursor sobre la tarjeta, esta debía realizar alguna animación.
- ***Selected***: Al seleccionar la tarjeta del personaje, se cambia el fondo a un color blanco. De esta forma el usuario sabe perfectamente qué personaje ha elegido en todo momento.

4.2.4.2. Escena de batalla

En esta escena, los únicos elementos que existen de interfaz de usuario son las barras de vida. El boceto realizado se corresponde bastante con el prototipo final.

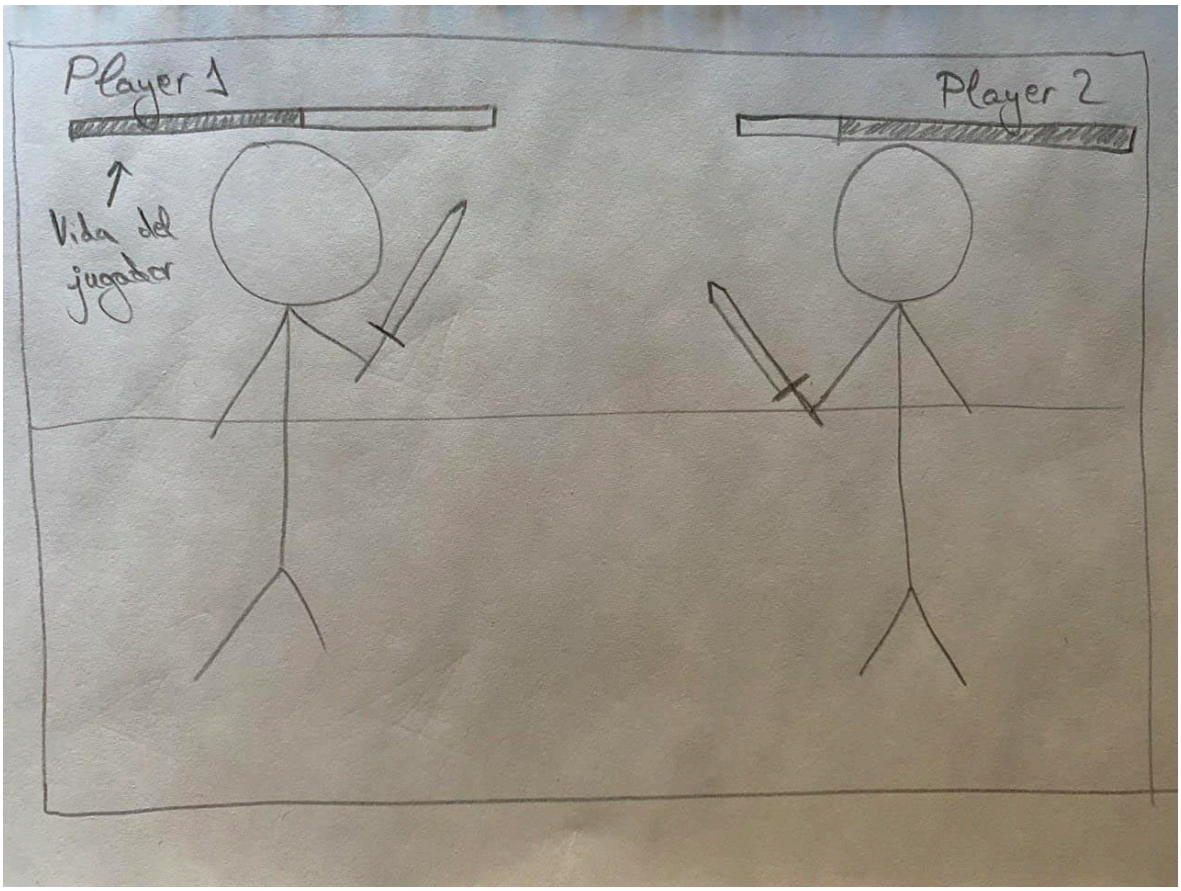

Ilustración 4. Boceto de la pantalla de combate

En el boceto se puede observar que los nombres que se sitúan sobre las barras de vida corresponden a los jugadores, mientras que en el prototipo se ha optado por utilizar el nombre del personaje del color correspondiente al jugador. La barra de vida se muestra dividida en dos zonas:

- **Zona verde:** Esta zona muestra la vida restante del personaje.
- **Zona roja:** Representa el daño que ha sufrido el personaje.

Cuando uno de los dos jugadores resulta ganador, aparece el título de final de la partida, indicando quién ha ganado el encuentro. Una vez más, se muestra el nombre del personaje victorioso en el color del jugador correspondiente, esta vez mostrando que ha ganado la partida.

4.3. Diseño del programa

Para este apartado se ha decidido realizar dos diagramas de clases. Cada uno de ellos corresponde a una de las escenas del prototipo. Se ha tomado esta decisión para evitar crear diagramas de clases gigantescos.

4.3.1. Diagrama de clases de la escena de selección de personaje

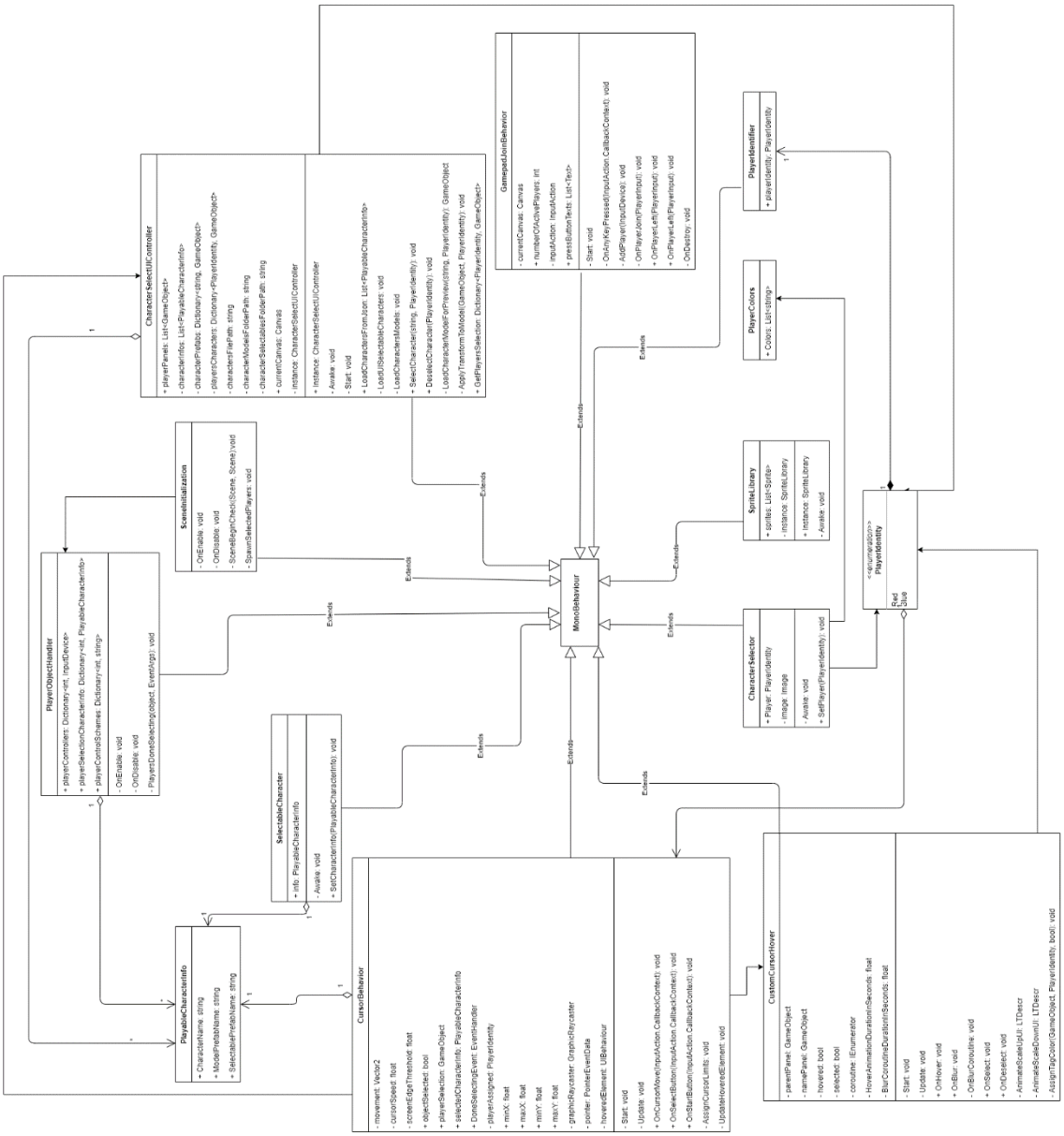

Ilustración 5. Diagrama de clases de la escena de selección de personaje

La clase principal de este diagrama es *CharacterSelectUIController*, ya que inicializa la escena, carga los prefabs de las tarjetas seleccionables y también los modelos de los personajes al seleccionarlos. La clase *CursorBehavior* utiliza la clase anterior, ya que se encarga de la los eventos de los cursores. Por lo tanto, si un jugador selecciona un personaje, debe notificarlo al controlador general.

GamepadJoinBehavior es el encargado de detectar nuevos dispositivos y agregarlos al sistema. *CustomCursorHover* es controlada por la clase *CursorBehavior* puesto que la primera maneja los eventos de las tarjetas de la interfaz. *PlayerObjectHandler* mantiene las estructuras de datos necesarias para el mantenimiento de la información de selección de personaje y *SceneInitialization* se aprovecha de estas mismas para iniciar la escena de combate.

4.3.2. Diagrama de clases de la escena de combate

Ilustración 6. Diagrama de clases de la escena de combate

Como en el diagrama anterior, existe una clase que controla el curso del juego, y esa es *GameController* y esta misma utiliza *GameplayUIController* para el control de la interfaz. No obstante, la escena parte en la clase *SceneInitialization*, como se comenta en el apartado anterior. A partir de ahí, se inicializan los personajes, comenzando por la clase *ThirdPersonController*, que controla todo el funcionamiento de los personajes. Esta clase se apoya sobre *BaseCharacter* para mantener la información relevante del combate del personaje, como la vida que tiene. La clase que se encarga de procesar los eventos del usuario es *ThirdPersonUserControl.*

Por otro lado, tenemos la interfaz *IHittable*. Las clases que la implementan serán controladores de colisiones de zonas del cuerpo del personaje para detectar los golpes. A su misma vez, la interfaz *IHittableCharacter* es implementada por *ThirdPersonCharacter* y permite ser llamado por cada una de las zonas del cuerpo que implementan *IHittable*.

Además de estos controladores, existe el de la cámara, que maneja su propio funcionamiento. CharacterSoundController se suscribe a los eventos de combate que ofrece *ThirdPersonController* y es responsable de obtener y reproducir los sonidos correspondientes de *SoundLibrary*.

4.4. Implementación

En este apartado se van a tratar todos los detalles técnicos de la implementación y cómo se han resuelto los problemas que han surgido durante la misma.

4.4.1. Personajes

4.4.1.1. Importación de los personajes

Como se ha mencionado anteriormente, tanto los personajes como las animaciones del prototipo han sido obtenidos a través de la plataforma Mixamo. Esta plataforma ofrece una gran variedad de personajes y animaciones gratuitos compatibles con *Mecanim*, el sistema de animación de Unity3D.

Para importar un personaje desde Mixamo, primero hay que descargarlo. Para ello, la propia plataforma habilita una opción para descargar los recursos adaptados para Unity3D, como se puede observar en la siguiente imagen.

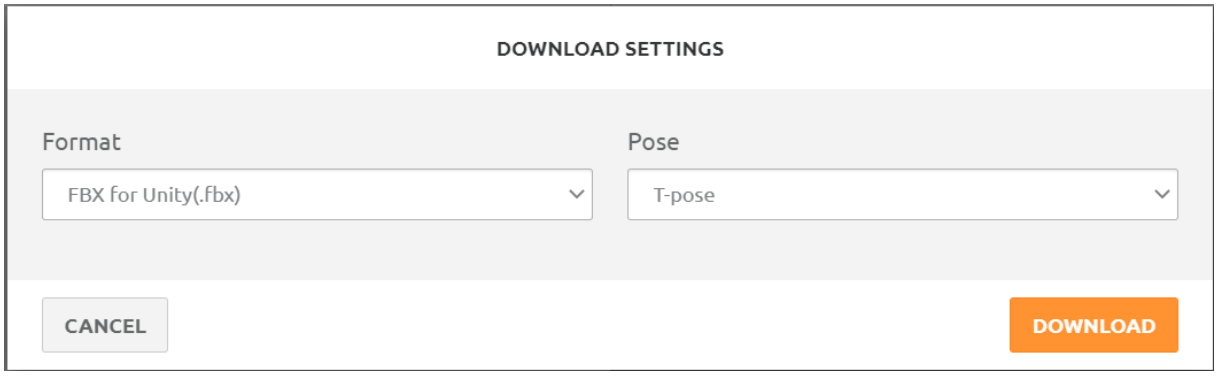

Ilustración 7. Opciones de descarga de los modelos 3D desde *Mixamo*

Una vez descargado el archivo, solo hay que arrastrarlo hasta el explorador de Unity3D y depositarlo en la carpeta correspondiente, en nuestro caso *Assets* → *Resources* → *Characters*. Algunos de los personajes que se han descargado, al visualizarlos en la escena aparecían en blanco, esto es debido a que las texturas no se cargan correctamente. Para solucionarlo, el programa permite extraer las texturas y los materiales del propio modelo descargado.

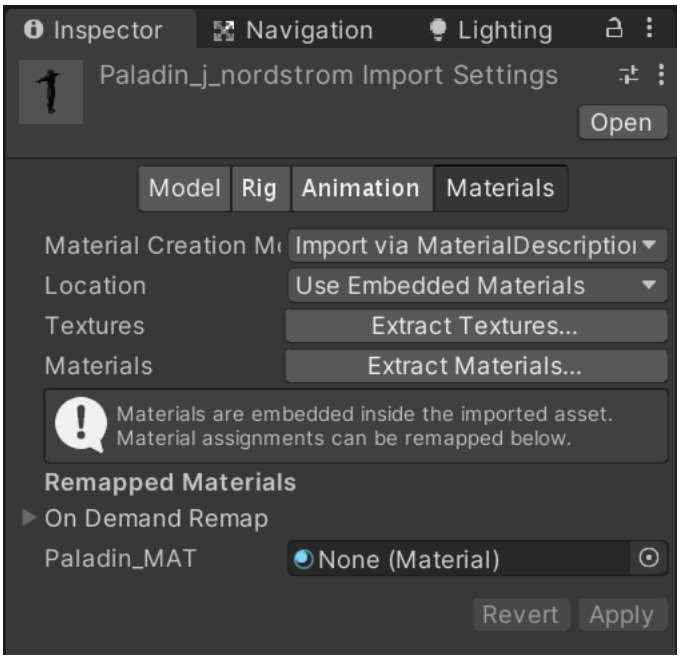

Ilustración 8. Opciones de los materiales del modelo importado

Primero hay que extraer las texturas y después los materiales que Unity3D crea a partir de las texturas. Una vez realizado este proceso, el personaje ya se puede importar en la escena en posición de T.

Ilustración 9. Resultado del modelo importado en la escena

Ahora hay que definir el tipo de animaciones que va a tener el personaje. Para este proyecto se ha definido que todos los personajes que se incluyan deben ser humanos o humanoides, para que todos puedan compartir las mismas animaciones. Para decirle a Unity3D que el modelo se va a animar es un humanoide, tenemos que configurarlo al importarlo.

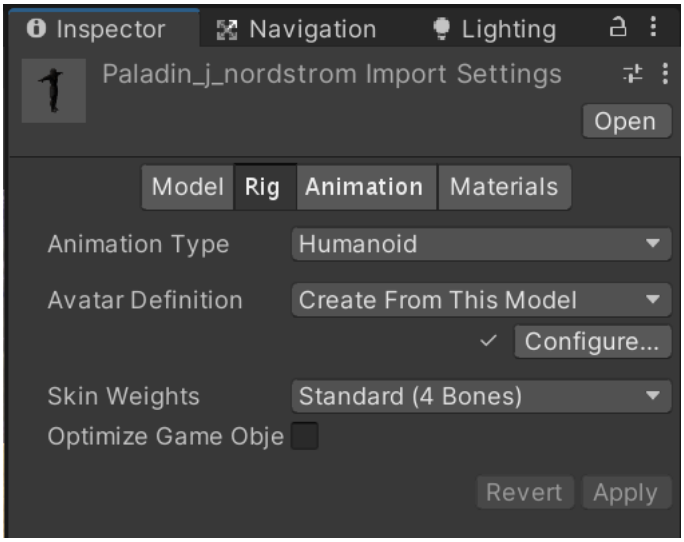

Ilustración 10. Configuración del tipo de animación del modelo

Una vez esté configurado de esta manera, tendremos nuestro personaje listo para ser animado.

4.4.1.2. Importación de las animaciones

Como se ha hablado en el apartado de *Diseño del juego* sobre las animaciones, todos los personajes comparten una serie de animaciones básicas. Esto es posible gracias a la compatibilidad de las animaciones con *Mecanim* y a que *Mixamo*, a la hora de descargarlas, hace el *retargetting* de forma completamente automática. Esta característica hace de *Mixamo* una herramienta indispensable para el uso de animaciones, ya que ahorra una gran cantidad de trabajo.

Las animaciones se han descargado de la misma forma que se ha hecho con los personajes en el apartado anterior. Para las animaciones básicas que comparten todos los personajes, se han descargado animaciones del paquete *Sword And Shield*. En Mixamo, primero seleccionamos el personaje al que le queremos aplicar la animación. Acto seguido buscamos la animación que queremos que realice y pulsamos en el botón *Download*.

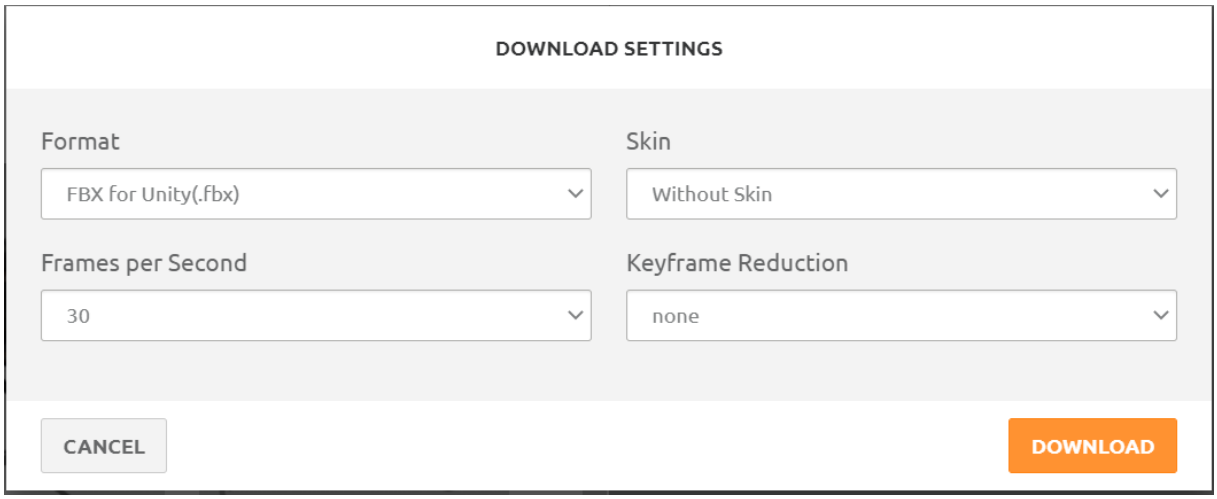

Ilustración 11. Opciones de descarga de las animaciones de Mixamo

Como se puede observar en la anterior imagen, Mixamo permite descargar las animaciones sin *Skin*, es decir, sin incluir el modelo del personaje en la descarga. Una vez descargada, el proceso de incluir el recurso en el proyecto es el mismo que para los personajes. No obstante, hay que asignarle el avatar del personaje al que se le está aplicando la animación.

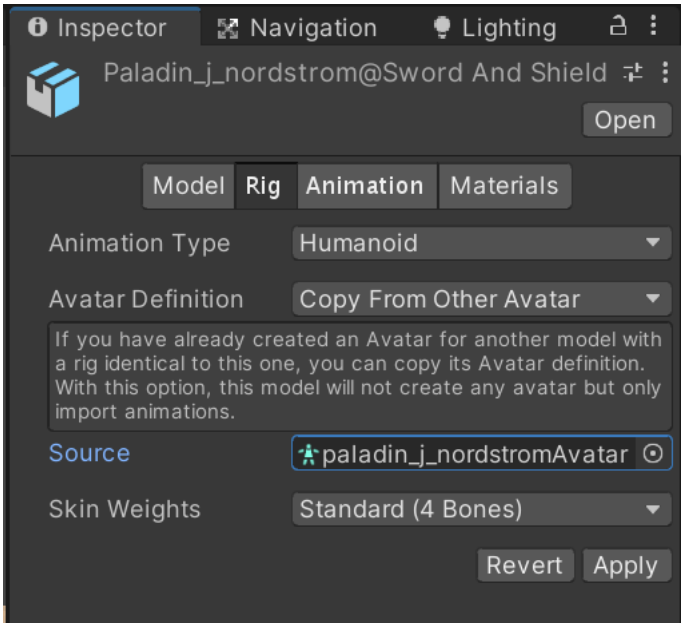

Ilustración 12. Configuración del tipo de animación importado

En el caso de las animaciones genéricas, se les ha asignado como avatar el del personaje Paladín. La razón de esta decisión es porque este personaje es el más genérico en cuanto a esqueleto humanoide, por lo que, si se quiere utilizar una de sus animaciones en otros personajes, tendrá más probabilidad de que sea compatible.

Uno de los retos que presenta este proyecto es proporcionar con facilidad a los diseñadores que puedan trabajar en un futuro en el desarrollo de este prototipo la posibilidad de añadir nuevos personajes con nuevas animaciones. Esto implica que un diseñador nunca debe tener que modificar código o el comportamiento del *Animator Controller* para añadir nuevos

personajes. Por tanto, es responsabilidad del programador dejar preparada una estructura que abstraiga el comportamiento de los personajes, indistintamente de sus aspectos.

4.4.1.3. Estructura del *Animator Controller*

Para este proyecto se ha creado un *Animator Controller* genérico, de tal forma que todos los personajes poseen los mismos estados y las mismas transiciones.

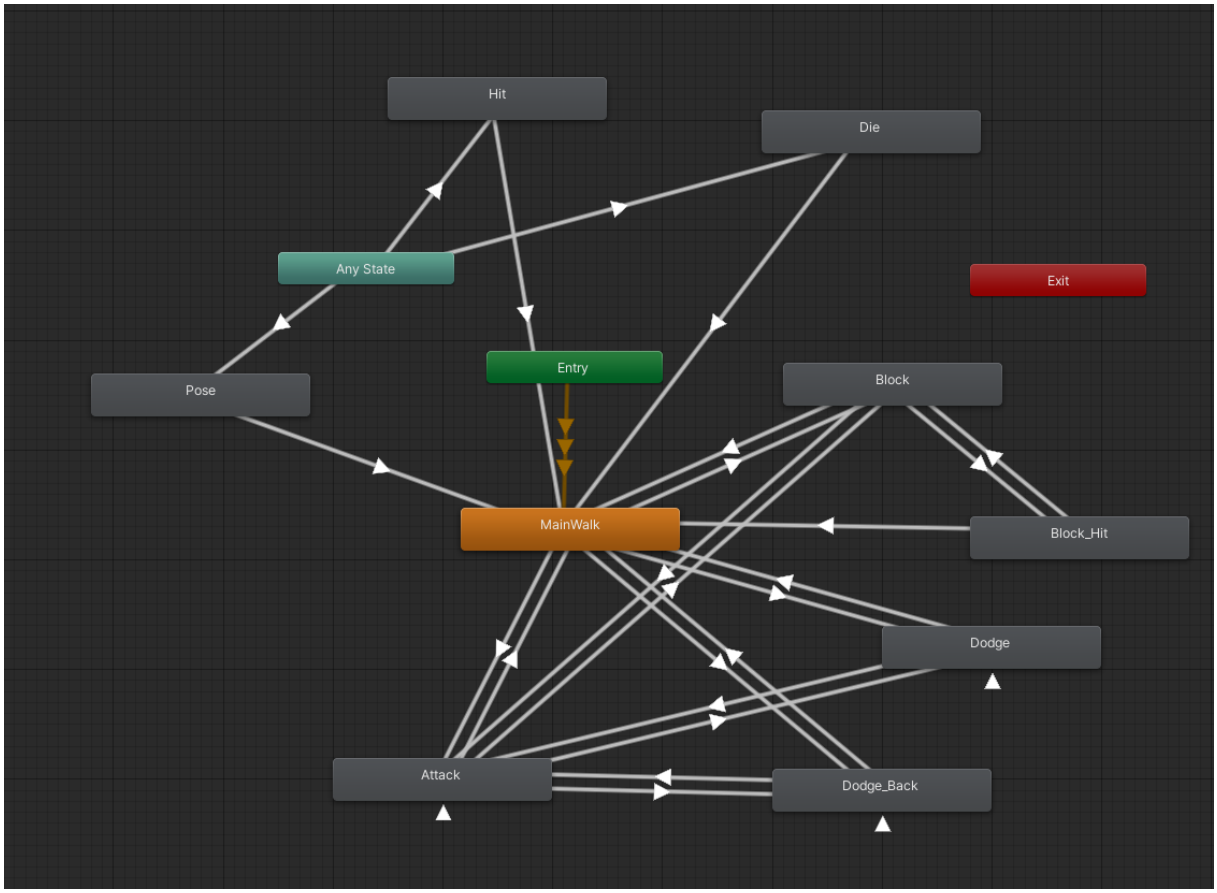

Ilustración 13. Animator Controller

A continuación, se explica uno a uno cada uno de los estados:

- ***Entry***: Es el estado de entrada al árbol. Todos los *Animator Controller* poseen este estado.
- ***Exit***: Es el estado de salida del árbol. Al igual que sucedía con el estado anterior, todos los *Animator Controller* poseen uno.
- ***MainWalk***: Este estado es un árbol de mezcla. Se encarga de realizar la animación de movimiento. Al ser un árbol de mezcla, permite interpolar animaciones y crear resultados más realistas. En este caso se ha utilizado para que el personaje tenga una animación de movimiento que se corresponda con el parámetro de velocidad que tenga asignado.

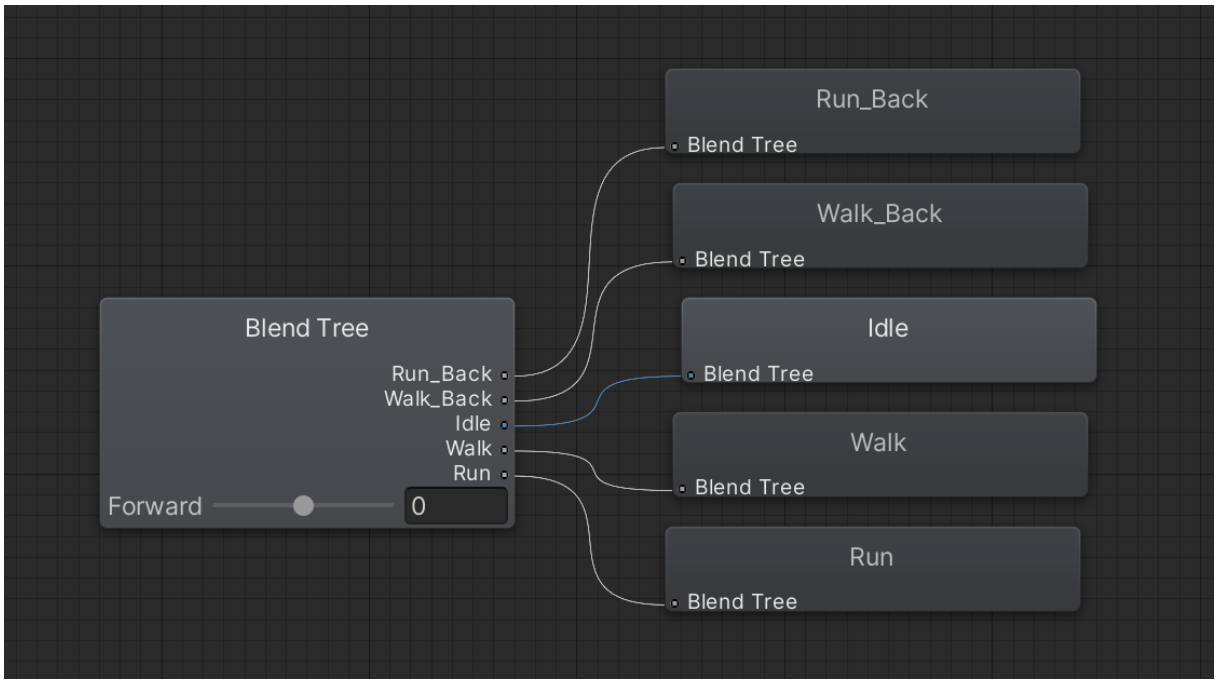

Ilustración 14. Árbol de mezcla de las animaciones de andar, correr y estar en el sitio

El valor del parámetro ***Forward*** marcará las animaciones que se ejecutarán en cada momento:

  - Si Forward es positivo, se ejecutarán las animaciones de *Walk* y *Run.*
  - Si Forward es negativo, se ejecutarán las animaciones de *Walk_Back* y *Run_Back.*
  - Si Forward es cero, el personaje permanecerá en un estado ocioso.
- ***Attack***: Representa el estado en el que el personaje realiza la animación de ataque.
- Dodge: Actúa cuando se tiene la intención de esquivar hacia delante, como una voltereta. Solo se alcanza este estado si el parámetro *Forward* es mayor que cero, lo que indica que el jugador quiere ir hacia adelante.
- ***Dodge_Back***: Similar al estado *Dodge*, permite esquivar hacia atrás. Se produce si el parámetro *Forward* es cero o menor que cero.
- Block: Estado que permite al personaje realizar un bloqueo.
- ***Block_Hit***: Estado complementario de *Block*. En el caso de que el jugador consiga bloquear satisfactoriamente un ataque, se ejecutará la animación de impacto sobre el arma.
- ***Pose***: Este estado especial es accesible desde cualquier otro estado del árbol. Se utiliza cuando se selecciona un personaje en la pantalla de selección y como pose de victoria.
- ***Hit***: Estado que se ejecuta cuando el personaje es golpeado por un ataque rival.
- ***Die***: Estado que se utiliza cuando un personaje muere para reproducir su animación de muerte.

Este árbol de estados que compone el *Animator Controller* está gobernado por un conjunto de parámetros. Dependiendo del estado en el que se encuentre en ese momento y del valor de los parámetros, se realiza una transición desde un estado a otro. Estos parámetros son manejados desde el script *ThirdPersonCharacter*, que es el encargado de gestionar el estado y las animaciones del personaje.

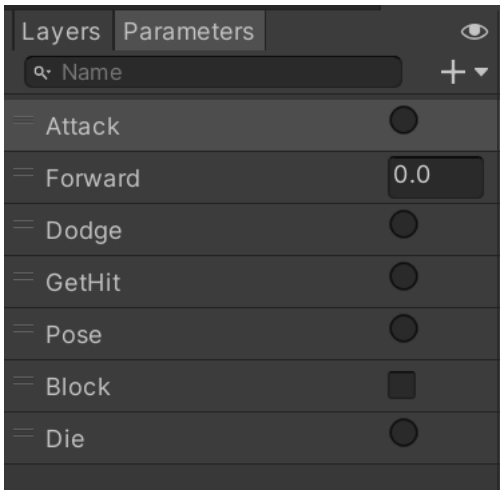

Ilustración 15. Parámetros del *Animator Controller*

Los parámetros utilizados en el proyecto pueden ser divididos en 3 tipos: números flotantes, *triggers* y booleanos.

- El único número flotante que se ha utilizado es el del parámetro Forward. Como se ha explicado anteriormente, este número indica el movimiento del personaje, tanto hacia delante como hacia detrás.
- Los parámetros *triggers* están pensados para ser activados durante una breve fracción de tiempo y ser desactivados justo después, pues representan acciones que solo deben realizarse una vez. Parámetros de este tipo son: *Attack, Dodge, GetHit, Pose* y *Die*.
- Finalmente, los parámetros booleanos representan estados que deben realizarse en bucle hasta que se cumpla cierta condición. En el caso de *Block*, mientras que el jugador mantenga el botón de bloquear pulsado, el personaje seguirá bloqueando. Cuando lo levante, el parámetro cambiará de estado y el personaje dejará de bloquear.

4.4.1.4. Encadenamiento y cancelación de animaciones.

Como ya se habló en el apartado de *Diseño del juego*, el encadenamiento de animaciones ha sido el principal reto del proyecto. Se ha aprovechado el mecanismo de los eventos en los clips de las animaciones para llamar al controlador del personaje e indicarle cuándo se puede interrumpir una animación para encadenar otra.

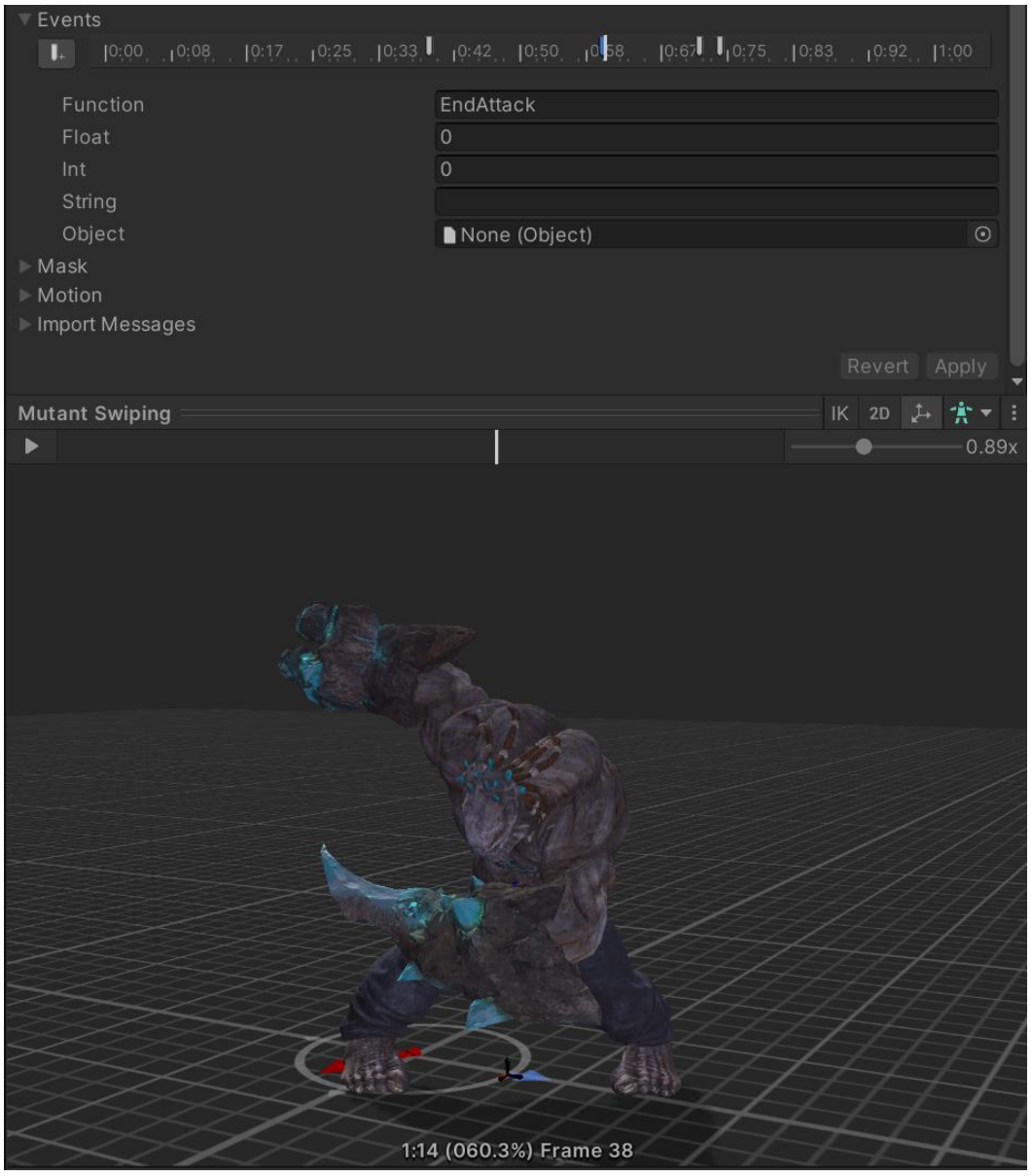

Ilustración 16. Sincronización del evento de ataque con la animación

En la imagen de arriba se puede observar el evento *EndAttack* en el momento en el que el Mutante termina de realizar el recorrido con su brazo para atacar. En el momento que se llama a este evento, el usuario puede encadenar otra acción, cancelando la animación de ataque y comenzando la otra instantáneamente. La función *EndAttack* a la que se está llamando está definida en el script *ThirdPersonController*.

```
public void EndAttack()
{
    _isAttacking = false;
}
```

Ilustración 17. Función llamada por el evento de fin de ataque

Las funciones a las que se hace referencia desde los eventos son tan simples como las de la imagen. Manejando pequeños *flags* a través de la correcta temporización de eventos en las animaciones, podemos saber si es posible realizar una acción cuando se reciben las acciones del usuario. Si el jugador quisiera atacar o realizar otra acción antes de que se llame a este evento, la acción no se realizará. Sin embargo, si se hace posterior a este evento, esta animación se cancelará, dando paso a la siguiente en una transición.

Las transiciones entre animaciones se han trabajado de forma intensiva para dar una sensación de rapidez entre animaciones, pero al mismo tiempo manteniendo una coherencia entre la postura actual del personaje y la siguiente para que la transición no se sienta antinatural.

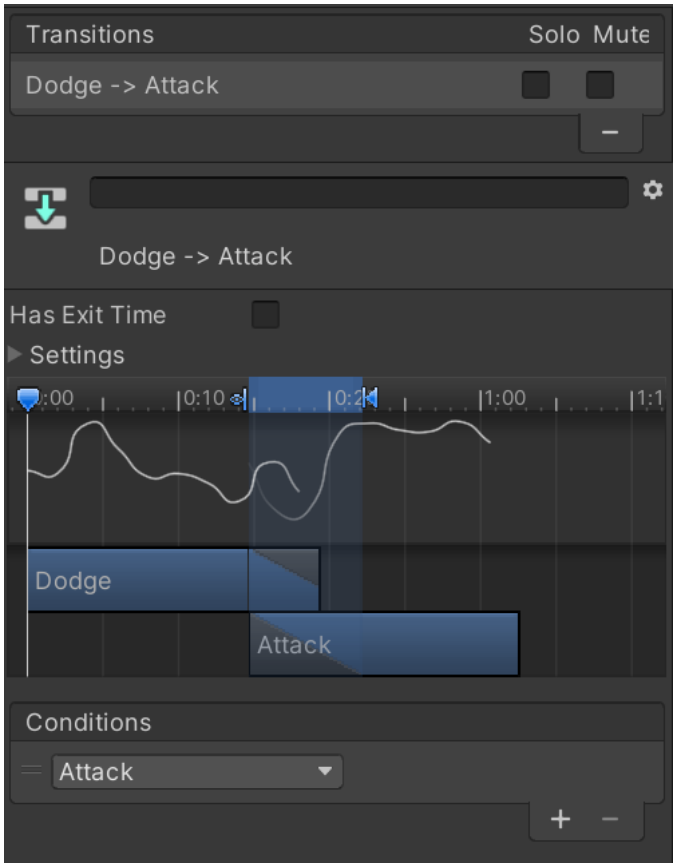

Ilustración 18. Transición entre las animaciones de dar una voltereta y atacar

En la anterior imagen se puede observar una transición de tan solo 250 milisegundos entre la animación de realizar una voltereta hacia delante y el ataque. De esta manera se mantiene un compromiso entre la velocidad de la transición y la naturalidad de la misma.

4.4.1.5. Animaciones personalizadas para cada personaje

Como se ha mencionado anteriormente, se ha abstraído el comportamiento genérico que deben tener todos los personajes en un *Animator Controller*. Sin embargo, puesto que cada personaje es diferente, sus animaciones deberían ser diferentes para darle unas personalidad y características distintas al resto.

Para llevar a cabo dicha tarea, Unity3D ofrece un componente llamado *Animator Override Controller*¸ que permite sobrecargar las animaciones del *Animator Controller* anteriormente

definidas con nuevas animaciones. Se pueden crear tantos *Animator Override Controller* como se quiera, por tanto, en el caso de este proyecto se han creado dos: una para Paladín y otra para Mutante.

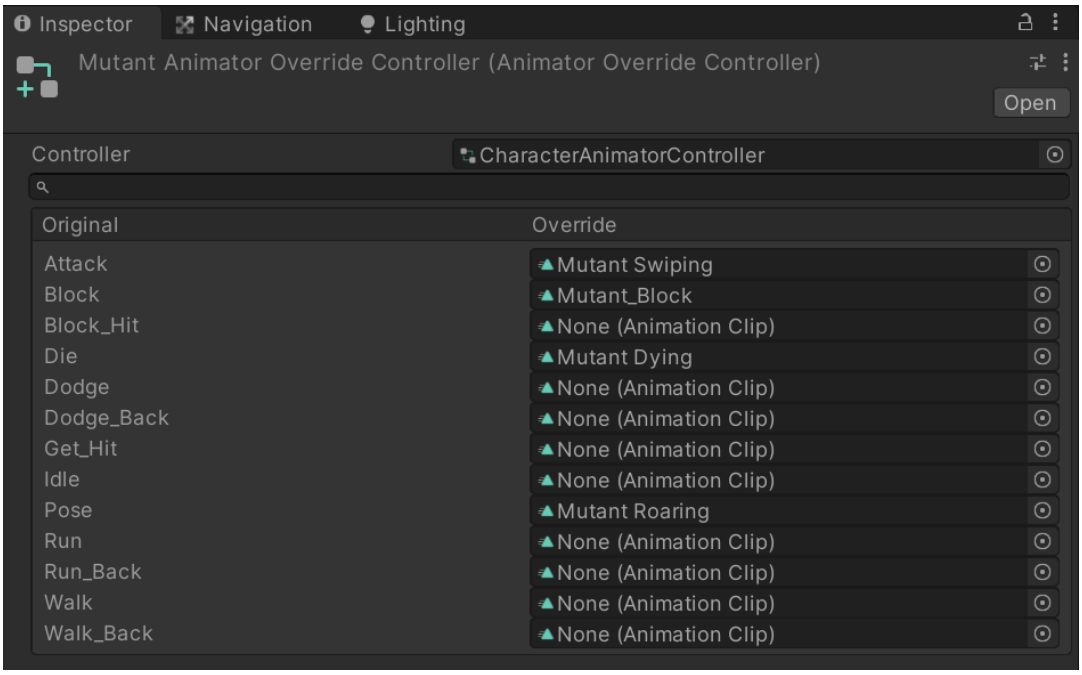

Ilustración 19. Sobrecarga del *Animator Controller* del personaje Mutante

Como se puede apreciar en la anterior imagen, se pueden sobrecargar tantas animaciones como se desee. El resto de las animaciones serán las que tenga asignadas el *Animator Controller* por defecto. Con la perspectiva puesta en el futuro de este proyecto, un diseñador podría añadir nuevos personajes y nuevas animaciones con gran facilidad.

4.4.1.6. Detección de colisiones

Otro de los puntos importantes del proyecto ha sido el desarrollo de una detección de colisiones eficiente y fiable. Al comienzo del proyecto se estudió la posibilidad de crear estructuras de datos espaciales personalizadas para una detección de colisiones fiable. Sin embargo, esta posibilidad fue descartada rápidamente, puesto que, aunque estas estructuras podían detectar con una gran precisión las colisiones entre objetos, el rendimiento que podían en una aplicación en tiempo real era realmente bajo.

Por tanto, replanteé la búsqueda de la estructura de datos adecuada con el criterio de que no necesitaba tener una gran precisión, sino que más bien buscaba una estructura eficiente. Finalmente recurrí por utilizar colisionadores de formas simples en los personajes, y para mantener el punto de precisión que buscaba al inicio, decidí dividir el cuerpo de cada personaje en tres partes: cabeza, torso y piernas. Por lo tanto, se le asignó a cada una de estas zonas del cuerpo un colisionador independiente y se ajustó su forma y tamaño de la manera más precisa posible, para que las colisiones fueran fieles a lo que los jugadores ven en la pantalla.

Ilustración 20. Colisionadores de las distintas partes del modelo de Mutante

La ventaja de estos colisionadores es que Unity3D tiene implementaciones de los mismos, por lo que no hay que desarrollar ninguna estructura espacial adicional. Además, el coste computacional de cada uno es muy bajo gracias a sus formas simples.

4.4.2. Implementación de las escenas

A continuación, se van a detallar los elementos más importantes que componen las escenas. Estos elementos ayudan a crear un escenario donde se desarrolla el combate.

4.4.2.1. Terreno

El terreno sobre el que se mueven los personajes es una isla desierta. Esta isla es bastante montañosa, aunque tiene un valle sobre el que se desarrolla la batalla entre los jugadores. Además, este valle es ideal para incluir un mar en la zona en la que acaba el terreno.

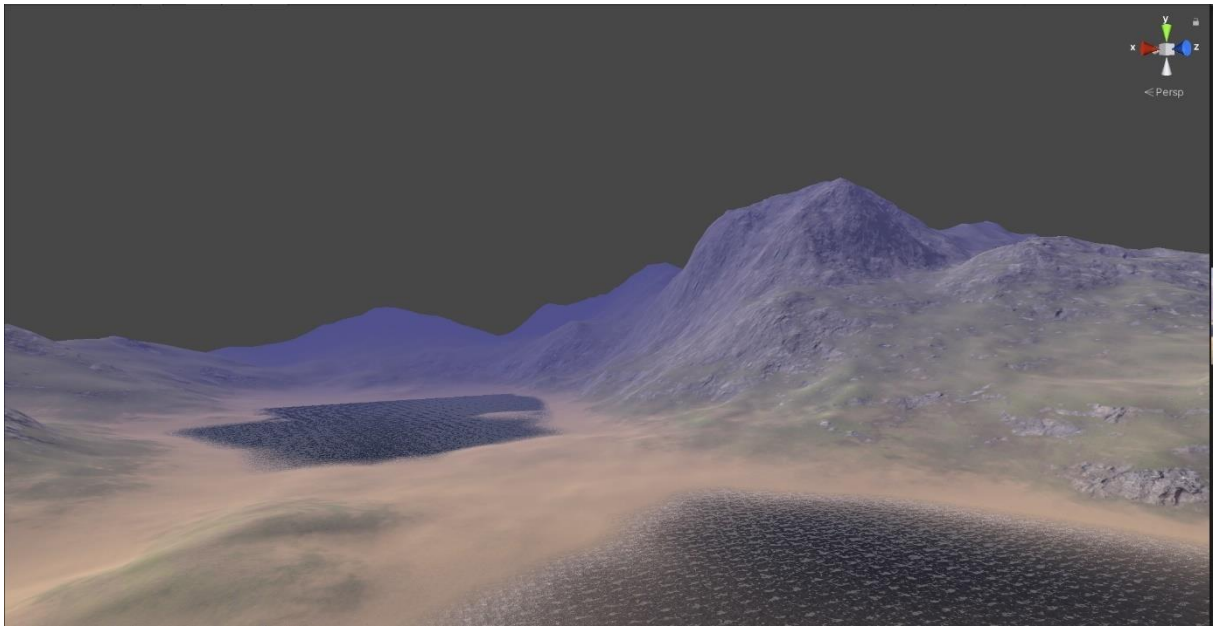

Ilustración 21. Terrenos sobre el que tiene lugar el combate

El terreno se ha descargado desde la *Asset Store* de Unity3D y pertenece al paquete *Free Island Collection* (Studios, 2021).

4.4.2.2. Agua

El agua es un elemento escénico que contrasta bastante con la aridez del terreno de la isla. En este caso se ha utilizado para crear el mar que aporta el horizonte del paisaje.

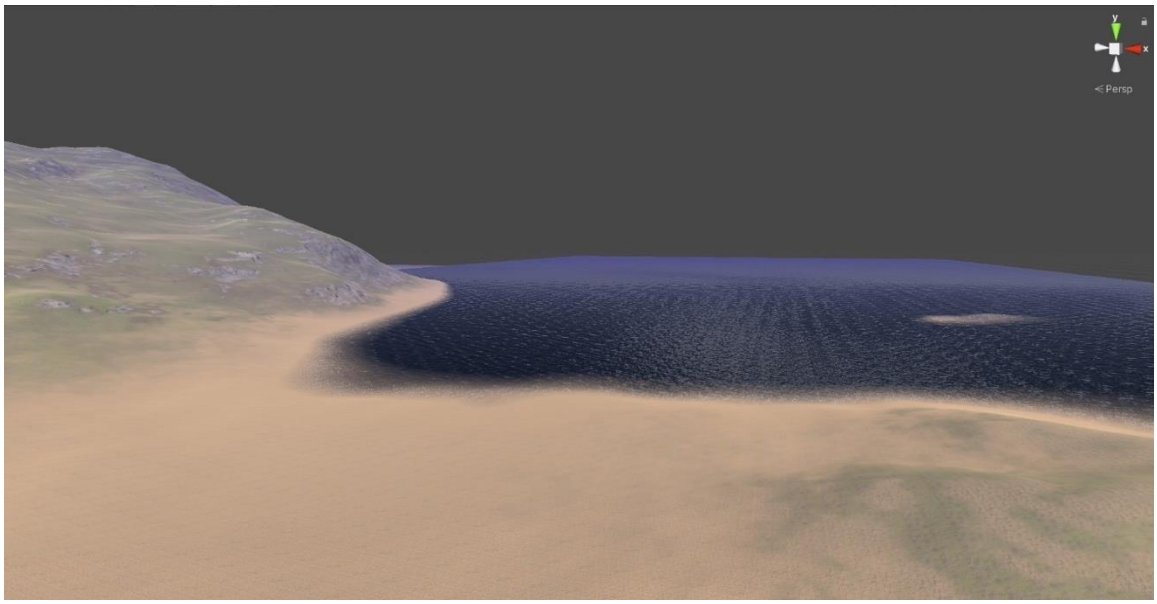

Ilustración 22. Zona de mar del paisaje donde tiene lugar el combate.

El agua se ha descargado de la *Asset Store* y pertenece al paquete *NVJOB Water Shaders V2.x* (Veselov, 2021).

4.4.2.3. Skybox

El *skybox* de la escena juega un papel importante en la ambientación de la misma y suele pasar desapercibido. Por esta razón, para este proyecto se ha querido que esta sea la excepción, estableciendo un *skybox* de un cielo en un atardecer espectacular que encaja perfectamente con el mar y la playa, dando la sensación de estar en un día de verano.

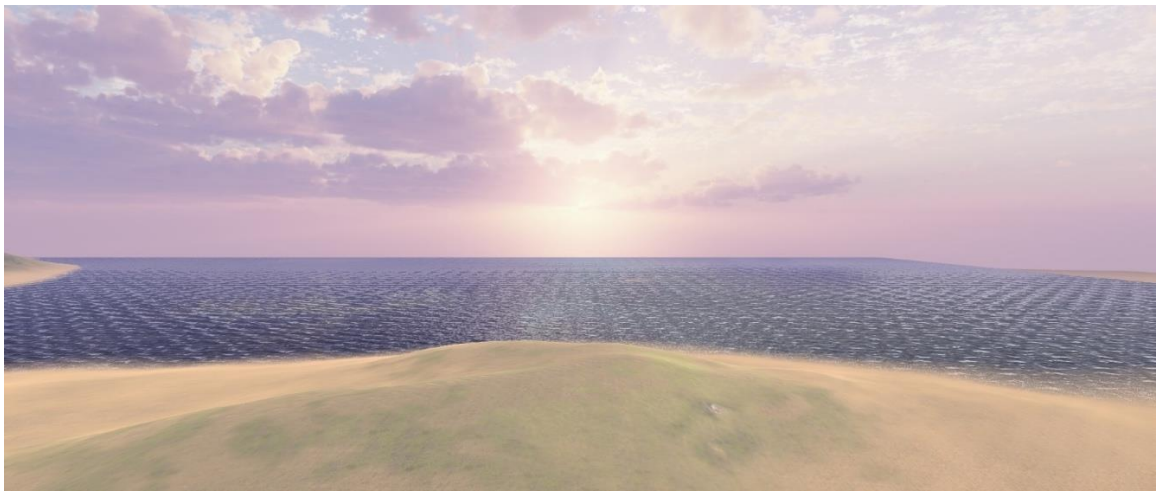

Ilustración 23. Skybox de la escena.

El nombre del paquete es A*llSky Free - 10 Sky / Skybox Set* (rpgwhitelock, 2021) y se ha obtenido de la *Asset Store*, al igual que los anteriores recursos.

4.4.2.4. Sonido

El sonido siempre juega un rol muy importante en cualquier contenido de tipo audiovisual. Los sonidos bien realizados crean en los usuarios experiencias más inmersivas y de mayor calidad, proporcionando una mejor experiencia de usuario.

Para este proyecto se han utilizado dos tipos de sonidos: los sonidos de ambiente y los sonidos puntuales. Los del primer tipo crean el entorno deseado por el desarrollador. En este caso se ha querido crear un ambiente épico, por lo que se ha utilizado música de batalla. La música utilizada para la escena de combate es *Dragon Castle* (Symphony, 2021), mientras que la escogida para la selección de personaje es *Warrior Rising* (Grimm, 2021). L

En segundo lugar, los sonidos puntuales solamente se reproducen cuando ocurren ciertos eventos en el juego. Estos sonidos están relacionados con eventos del combate. A continuación, se presenta una lista de los sonidos utilizados en cada situación:

- **Pasos**: cada vez que el personaje realiza un paso sobre el terreno, se reproduce el sonido de una pisada. Este evento está enlazado con las animaciones de correr y caminar (SonidoefectosFX, 2021). Para este evento en concreto, se han extraído 8 clips distintos de pasos y se reproducen de forma aleatoria para evitar la constante repetición del mismo sonido.
- **Ser golpeado**: cada vez que el personaje es golpeado, realiza un sonido de gruñido. Cada personaje tiene un sonido propio (FiftySounds, 2021).
- **Bloqueo con el arma**: por cada bloqueo satisfactorio de un ataque enemigo, sonará el efecto de dos armas metálicas chocando entre sí (FiftySounds, 2021).
- **Muerte**: cuando muere uno de los personajes, se reproducirá un sonido de muerte (Capcom, 2021).

Estos sonidos puntuales son ejecutados gracias a eventos declarados en el script *Third Person Controller*. El script *Character Sound Controller* se suscribe a estos eventos al inicio de la escena y obtiene los sonidos gracias a la clase *Sound Library*, que es la encargada de cargar todos los sonidos al comienzo del juego para tenerlos a disposición cuando sea necesario.

4.4.3. Cámara

Esta sección describe los distintos comportamientos de la cámara durante el combate.

4.4.3.1. Durante el combate

El papel que juega la cámara en este proyecto es fundamental para la experiencia del usuario, ya que un correcto posicionamiento de la cámara en cada momento puede crear una buena sensación sobre los jugadores. En este videojuego, la cámara no puede ser controlada por los usuarios de forma directa (es decir, moviendo la cámara o ajustando su ángulo de visión directamente en sus parámetros), por lo que es responsabilidad completa del programador que este elemento proporcione una visión correcta en cada momento, ajustando los movimientos del mismo con cada acción de ambos jugadores.

El controlador utilizado para el posicionamiento automático de la cámara ha sido creado siguiendo una serie de restricciones para la misma para facilitar el cálculo de las coordenadas en tiempo real.

La primera de las restricciones es que el campo de visión de la cámara debe ser siempre el mismo. Si se mantiene este parámetro constante, solo habrá que calcular la posición de la cámara. La segunda restricción es que la cámara siempre estará en la coordenada X intermedia entre los dos jugadores. El resultado de esto es que siempre se forma un triángulo isósceles entre los dos personajes y la cámara. Finalmente, si se establece un campo de visión de 60º, el triángulo resultante entre los dos personajes y la cámara es equilátero. Esto facilita los cálculos.

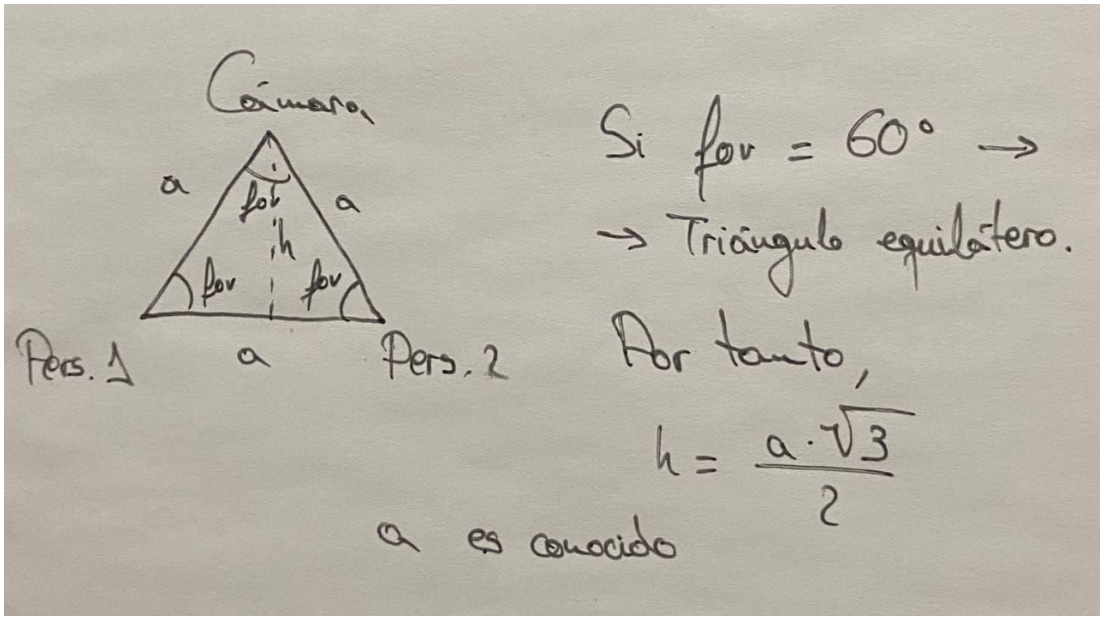

Ilustración 24. Boceto de los cálculos de la posición de la cámara

Una vez conocida la altura del triángulo, el cálculo de la posición de la cámara es trivial. Actualizando la posición de la cámara en cada *frame* de la aplicación nos aseguramos de que ambos jugadores pueden seguir la acción de sus personajes sin problemas, ya que obtienen una visión de ambos personajes simultáneamente.

4.4.3.2. Cámara del ganador

Para la cámara del ganador se ha querido realizar una transición que dé importancia al jugador victorioso. Para ello, en vez de utilizar una cámara nueva, se reutiliza la misma cámara que se utiliza durante el combate y en el momento que se declara un ganador, se realiza una transición, consiguiendo un efecto suave.

La transición de la cámara se ha realizado utilizando dos interpolaciones, una sobre la posición y otra sobre la rotación del objeto. Combinando estas dos propiedades se consigue un efecto visual bastante satisfactorio. Para realizar las interpolaciones se ha hecho uso de las funciones que ya implementa Unity3D para interpolaciones lineales.

```
transform.position = Vector3.Lerp(_initialPosition, _finalPosition, t);
transform.rotation = Quaternion.Lerp(Quaternion.Euler(_initialRotation), Quaternion.Euler(_finalRotation), t);
t += Time.deltaTime;
```

Ilustración 25. Código del cálculo de las interpolaciones de la animación de fin de combate de la cámara

4.4.4. Implementación del multijugador local

Para el desarrollo del multijugador local se ha utilizado el nuevo sistema de entrada de Unity3D que la aplicación lanzó con la versión 2019.1. Unity3D desarrolló este nuevo sistema que soluciona problemas del original, facilita otras muchas tareas como la inclusión de nuevos dispositivos en tiempo de ejecución y presenta una gran consistencia entre plataformas.

Para la inclusión del sistema de entrada en el videojuego, primero hay que descargarlo desde la *Asset Store* e instalarlo. En segundo lugar, hay que deshabilitar el antiguo gestor de entrada, ya que no pueden coexistir ambos en el mismo proyecto.

4.4.4.1. Detección de nuevos jugadores

Para empezar a detectar jugadores que se unan a la partida, primero hay que añadir el nuevo componente *Player Input Manager*, que controlará la gestión de dispositivos.

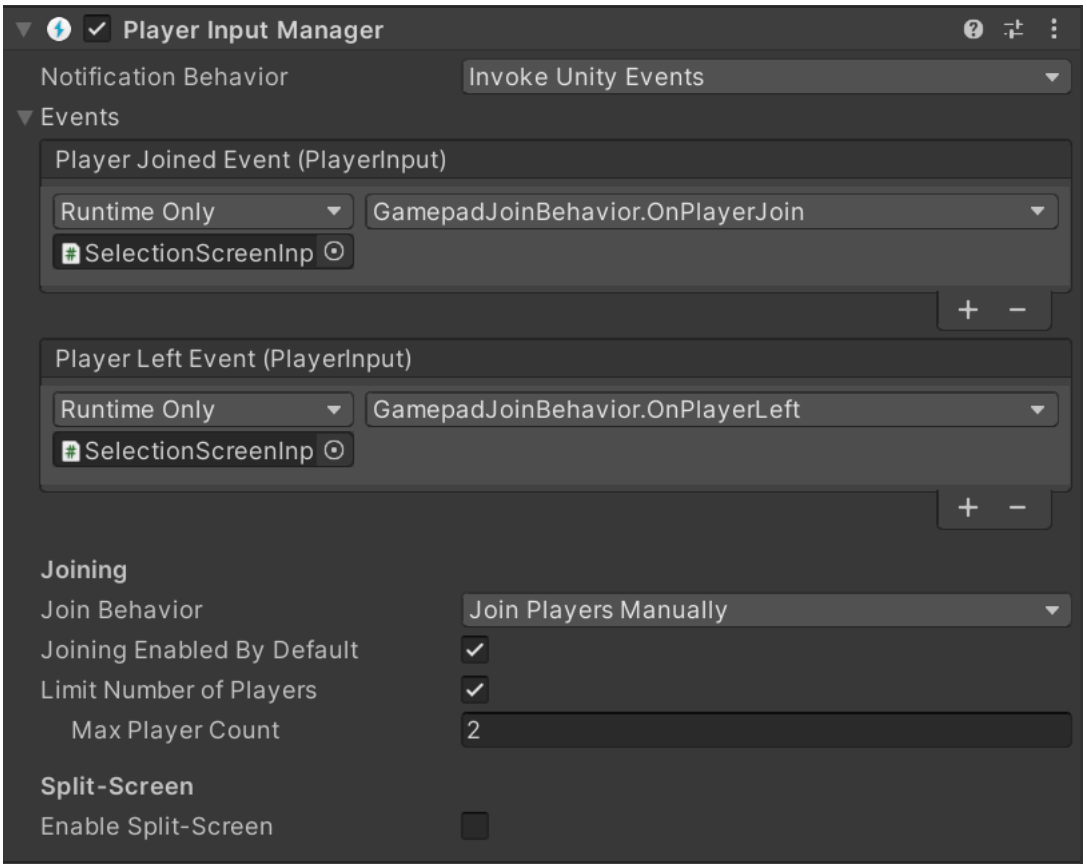

Ilustración 26. Player Input Manager

Como se puede observar en la imagen, el componente se ha configurado para que se gestionen los jugadores de forma completamente manual. Este controlador simplemente notificará cada vez que un dispositivo se conecte o se desconecte del equipo y el programador será el encargado de asignarlo o desasignarlo al jugador correspondiente.

Otra opción que se puede ver en la parte inferior del componente es la de habilitar la pantalla dividida. En un principio se consideró su uso, pero se decidió dejar desactivada la opción, ya que la pantalla dividida en este proyecto no sería de gran utilidad para los jugadores y los entorpecería.

A través del script *GamepadJoinBehavior* se establece una escucha constante de nuevos dispositivos. Para la detección de nuevos dispositivos se pueden definir botones o teclas específicas a pulsar, pero para facilitar la tarea al usuario, se decide que podrá pulsar cualquier tecla de su dispositivo para unirse a la partida.

```
//Let the players join by pressing any button.
inputAction = new InputAction(binding: "/*/<button>");
inputAction.performed += OnAnyKeyPressed;
inputAction.Enable();
```

Ilustración 27. Código que permite la detección de acciones de cualquier dispositivo

4.4.4.2. Definición de los controles

El nuevo sistema de entrada de Unity3D se caracteriza por su facilidad a la hora de definir los controles de los distintos dispositivos, pudiendo incluso crear distintos esquemas de controles para un mismo dispositivo y alternarlos en tiempo de ejecución. Para este proyecto se han creado dos esquemas de control distintos: uno para el teclado y otro para el uso de Gamepads.

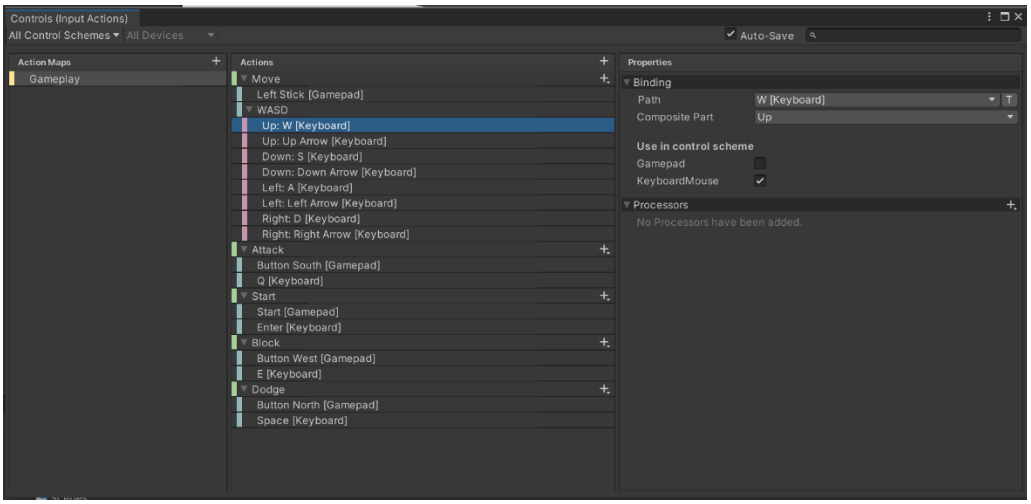

Ilustración 28. Definición de los controles

En la imagen se puede apreciar cómo, a través de la definición de acciones, se pueden asignar distintos controles para cada una. En este caso se ha asignado un control para Gamepad y otro para teclado, como se ha mencionado anteriormente. Además, para cada control se especifica qué esquema de control le corresponde.

Una vez que se ha detectado la inclusión de un nuevo dispositivo en el juego, se detecta el tipo de dispositivo. Si el tipo coincide con el de un teclado o un Gamepad, entonces se le asigna su esquema correspondiente y se instancia un *GameObject* a partir del *PlayerInput* del dispositivo gracias al método *Instantiate* de esta última clase, que pertenece al nuevo sistema de entrada. Este nuevo *GameObject* podrá manejarse con el dispositivo asignado y será su responsabilidad recoger los eventos del mismo.

4.4.4.3. Manejo de eventos de los dispositivos

El *GameObject* asignado al nuevo dispositivo tendrá el componente *PlayerInput* del que se ha hablado en el apartado anterior. Este componente es el encargado de recoger todas las acciones del usuario que se han definido en los controles y llamar a los métodos correspondientes en el controlador del GameObject. Esto permite que, dependiendo del controlador que se esté utilizando en cada momento, las acciones del usuario puedan desembocar en distintos eventos dentro de la aplicación. En cualquier caso, las funciones que son llamadas por el *PlayerInput* deben aceptar los parámetros correspondientes a cada acción.

En el proyecto se utilizan dos controladores distintos en las diferentes escenas. Para la escena de selección de personaje, el *GameObject* asociado a los controles es el cursor que maneja

cada uno de los jugadores. En el caso de la escena de batalla, los *PlayerInput* componen cada uno de los personajes escogidos por los jugadores.

4.4.4.4. Traspaso del control de los dispositivos entre escenas

El mayor reto que ha supuesto la utilización del nuevo sistema de entrada ha sido el traspaso del control de los dispositivos entre escenas. La dificultad de esta parte reside en no perder el control de los dispositivos al cambiar de una escena a otra y además, transferir el manejo de los mismos desde unos *GameObject* a otros; en este caso, desde los cursores a los personajes jugables.

Primero se debe mantener cierta persistencia entre escenas sobre la identidad de los dispositivos y a qué jugador pertenece cada uno. Para ello se ha creado el script *PlayerObjectHandler* que mantiene varios diccionarios que almacenan toda esta información, incluyendo los componentes *PlayerInput* correspondientes a cada jugador (estos componentes almacenan la información del dispositivo correspondiente) y el personaje que ha escogido cada jugador. El componente *PlayerObjectHandler* no se destruye entre escenas, por lo que esta información se mantiene a salvo en todo momento.

```
public static Dictionary<int, InputDevice> playerControllers = new Dictionary<int, InputDevice>();
public static Dictionary<int, PlayableCharacterInfo> playerSelectionCharacterInfo = new Dictionary<int, PlayableCharacterInfo>();
public static Dictionary<int, string> playerControlSchemes = new Dictionary<int, string>();
```

Ilustración 29. Estructuras que mantienen la información de los dispositivos y personajes elegidos

Una vez que los jugadores terminan de seleccionar sus personajes, se transiciona hacia la escena de combate. Por tanto, los cursores se eliminan al destruir la escena anterior, pero la información sobre los dispositivos y sus usuarios se sigue manteniendo. En el script *SceneInitialization* se aprovecha la información guardada y se instancian los personajes correspondientes a través del método *PlayerInput.Instantiate*, mencionado anteriormente. A partir de este momento el manejo de los controles ya no reside en los cursores, sino en el controlador de los personajes.

```
foreach (var player in PlayerObjectHandler.playerControllers)
{
    var playerController = PlayerObjectHandler.playerControllers[player.Key];
    var playerObjectInfo = PlayerObjectHandler.playerSelectionCharacterInfo[player.Key];
    var playerControlScheme = PlayerObjectHandler.playerControlSchemes[player.Key];

    var currentObject = Resources.Load<GameObject>($"Characters/Prefabs/{playerObjectInfo.ModelPrefabName}");
    currentObject.GetComponent<SelectableCharacter>().SetCharacterInfo(playerObjectInfo);
    currentObject.GetComponent<PlayerIdentifier>().PlayerIdentity = (PlayerIdentity)player.Key;

    PlayerInput playerInput = PlayerInput.Instantiate(currentObject, player.Key, playerControlScheme, -1, playerController);

    var inputUser = playerInput.user;
    playerInput.SwitchCurrentControlScheme(playerControlScheme);
    InputUser.PerformPairingWithDevice(playerController, inputUser, InputUserPairingOptions.UnpairCurrentDevicesFromUser);
}
```

Ilustración 30. Código que traspasa el control de los dispositivos a la escena de combate

Una vez el combate finaliza, el proceso de traspaso de control desde los personajes a los cursores de la escena de selección de personaje es el mismo.

4.4.5. Implementación de la interfaz

A continuación, se detallan los aspectos más importantes del desarrollo de esta escena.

4.4.5.1. Tarjetas seleccionables

Las tarjetas seleccionables son los elementos de la interfaz que permiten la selección de un personaje específico a los jugadores. Se han creado tantas como la cantidad de personajes existentes y cada una contiene unos parámetros de personalización diferentes. Estos parámetros definen el personaje que se muestra en la tarjeta y la identidad del jugador (rojo o azul).

Tanto las imágenes de los personajes como los fondos de las tarjetas han sido creadas en Adobe Photoshop. Después, estas imágenes han sido incluidas como *Sprites* dentro de Unity3D para su uso como componentes *Image*.

Para conseguir que los bordes de la imagen original se ajusten al tamaño de la tarjeta, se han editado las imágenes correspondientes al fondo en el Editor de *Sprites* de Unity3D.

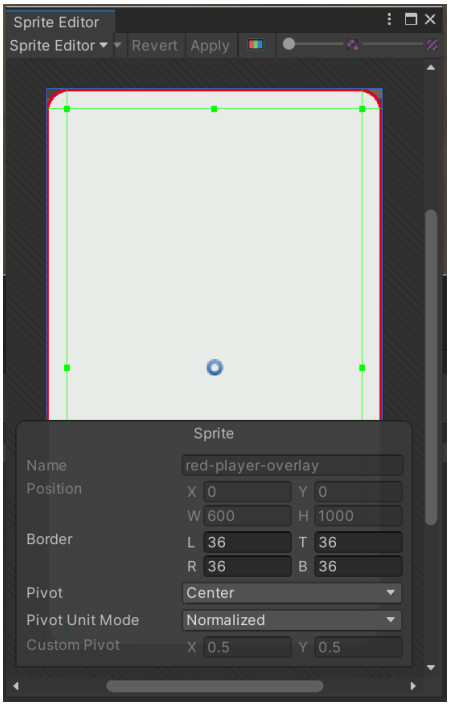

Ilustración 31. Edición del marco de la tarjeta en el editor de Sprites

Cada una de las imágenes de los personajes ha sido ajustada a mano dentro de las tarjetas para que se ajusten correctamente al espacio disponible. Además, se ha creado adicionalmente un pequeño letrero que muestra el nombre del personaje dentro de la tarjeta.

Ilustración 32. Resultado final de la tarjeta seleccionable

Como se puede apreciar, el fondo de la tarjeta de la imagen es de un tono grisáceo, mientras que el color del fondo creado en Adobe Photoshop es de color blanco. La explicación de esta cuestión está desarrollada en el apartado de *Implementación de los eventos del cursor sobre las tarjetas*.

4.4.5.2. Organización automática de las tarjetas en la interfaz

Pensando sobre el futuro de la aplicación, si se añadiera un nuevo personaje al videojuego se podría pensar que habría que mover todas las tarjetas de la interfaz para crear espacio para la del nuevo personaje. Sin embargo, si se añade una tarjeta nueva, el espacio se reorganizaría automáticamente.

Este comportamiento se puede conseguir gracias a ciertos componentes contenedores que ofrece Unity3D para sus interfaces. En este caso en concreto se ha utilizado un *Horizontal Layout Group*. Este componente agrupa los elementos horizontalmente según se le indique en sus opciones. Es un componente muy útil a la hora de añadir elementos a la interfaz dinámicamente, como es el caso actual.

Ilustración 33. Horizontal Layout Group

4.4.5.3. Implementación de los cursores

El cursor es el elemento más importante de la escena de selección de personaje, puesto que sin ellos no podría existir la interacción del usuario con la aplicación. En un juego de un solo jugador no habría sido necesario implementar un cursor personalizado, ya que se habría utilizado el cursor del propio sistema operativo. Sin embargo, ni los sistemas operativos ni Unity3D permiten utilizar más de un cursor de forma nativa. Es por ello que se ha decidido implementar uno desde cero.

El planteamiento inicial de la implementación es simple: se debe crear un elemento identificativo del jugador en la interfaz y se debe poder libremente dentro de los límites establecidos para cada jugador. Si se coloca el cursor sobre un elemento seleccionable, se debe poder seleccionar pulsando un botón. Para implementar esta funcionalidad, se decide crear un *GameObject* para cada cursor. En su forma más básica, un cursor no es más que un rectángulo controlado por un jugador que lo mueve por la pantalla.

Para que cada jugador identifique su cursor con su color correspondiente, a cada cursor se le añade una imagen de un puntero de color rojo o azul, dependiendo del lado en el que se encuentre. La determinación de qué lado le toca a qué jugador es simple: al primer jugador que se una a la partida se le asigna el color rojo y al segundo el azul.

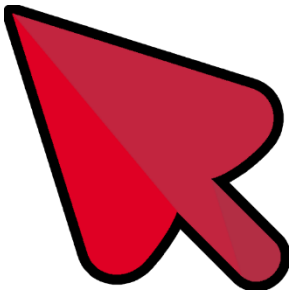

Ilustración 34. Cursor de selección de personaje.

El modelo original del puntero se ha descargado desde Klipartz (Klipartz, 2021), y se han modificado sus colores en Adobe Photoshop. El movimiento del cursor sobre la pantalla se ha implementado recogiendo los eventos de los dispositivos de los usuarios y transformándolos en movimientos sobre la interfaz. En cuanto a la acción de seleccionar una tarjeta, su implementación se explicará en el siguiente apartado.

4.4.5.4. Implementación de los eventos del cursor sobre las tarjetas

En el apartado anterior se ha explicado sobre la necesidad de crear cursores a partir de imágenes. La principal desventaja de esta decisión es que, al no tener disponible el cursor nativo de Unity3D, tampoco se disponen de eventos relacionados con el mismo. El principal evento que se echa de menos es el *hover*. El evento *hover* crea una mejor sensación sobre el usuario al darle *feedback* del elemento exacto sobre el que está posicionando el cursor.

Para no perder esta buena experiencia de usuario, se decidió implementar de primera mano el evento. Para esto se ha hecho uso del componente *Graphics Raycaster* que utiliza el *Canvas* de Unity para lanzar rayos contra la interfaz.

En cada *frame* de la aplicación, cada uno de los cursores lanza un rayo desde su posición en pantalla al *Canvas*. Si el rayo en algún momento alcanza un elemento de la interfaz, se

comprueba primero si este elemento es una tarjeta seleccionable. En caso afirmativo, se llama al método *OnHover* del componente *CustomCursorHover* añadido en las tarjetas. Este método se encarga de realizar una pequeña animación de escalado a la tarjeta en la interfaz mientras que el cursor siga sobre esta. En el momento en el que el cursor deja de posicionarse sobre dicho elemento, se llama al método *OnBlur* del mismo, que realiza otra animación de escalado, esta vez reduciendo el tamaño de la tarjeta.

Ilustración 35. Tarjetas de selección de personaje.

4.4.5.5. Barras de vida

Este elemento de la interfaz de la escena de combate es fundamental para saber cuántos puntos de vida le quedan a cada personaje. En ocasiones, un jugador puede optar por realizar estrategias más defensivas si tiene menos vida que el otro jugador para buscar un contraataque o un punto débil en su agresión. Por eso, este componente debe dar de forma fiable y clara la información que le corresponde.

Para la creación de las barras de vida se ha optado por hacer un componente compuesto por distintos elementos nativos de la interfaz de Unity3D que juntos dan la apariencia que buscamos.

El elemento clave en este apartado es el *Slider*. Este componente tiene por defecto un aspecto muy parecido al deseado, con unos bordes curvos que suavizan la apariencia. Su estructura permite modificar su color de fondo y el de la parte que rellena el fondo, por lo que lo hace lo suficientemente personalizable para las necesidades de este proyecto. El fondo se establece de color rojo para representar el daño que se le ha hecho al personaje, y el relleno de color verde, visualizando así la vida restante del jugador.

Ilustración 36. Barra de vida en combate.

4.4.5.6. Fuente de texto

La fuente de texto utilizada en este proyecto no pertenece al conjunto de fuentes que Unity3D trae por defecto. La fuente es *Astron Boy* y se ha descargado desde *wFonts* (wFonts, 2021).

4.5. Resultados

Tras superar el proceso de implementación, los resultados obtenidos se asemejan mucho a lo que se esperaba tras realizar el diseño inicial del videojuego.

4.5.1. Selección de personaje

Tras la implementación personalizada de los cursores y los eventos de los mismos, el resultado final de esta escena como del comportamiento de la misma son muy satisfactorios.

Ilustración 37. Pantalla de selección de personaje.

Como se puede observar, se diferencia claramente qué lado de la pantalla corresponde a cada jugador y cuáles son los personajes que puede elegir cada uno. Los cursores se identifican con el color de los jugadores, al igual que las tarjetas. En el estado que refleja la imagen, los jugadores todavía no han escogido personaje, y por tanto las tarjetas aparecen apagadas, pero en el momento en el que se selecciona una de ellas, se puede apreciar sin errores cuál es el personaje elegido.

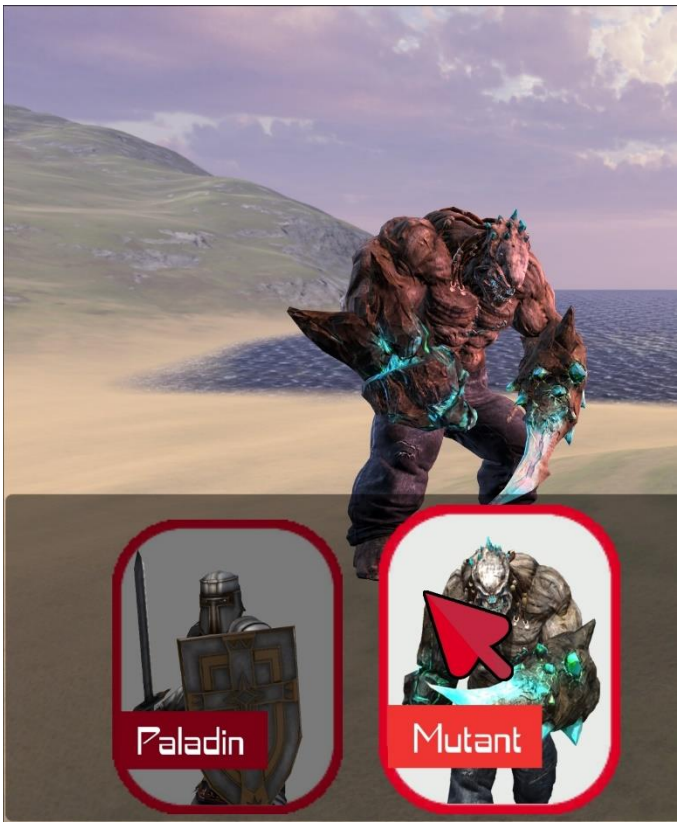

Ilustración 38. Selección de personaje con hover.

Cuando el jugador rojo selecciona a Mutante, su tarjeta parece más viva y es más grande. Quitar el cursor de la tarjeta no elimina este efecto a menos que se seleccione otro personaje.

4.5.2. Escena de batalla

Una de las principales intenciones al crear esta escena era crear un ambiente épico para que los jugadores se sientan como en una batalla real. Sin duda alguna, con el paisaje de la playa, el sonido del mar, la puesta de sol y la música épica, se puede decir que se ha logrado esta sensación.

Ilustración 39. Combate cuerpo a cuerpo

Pasando al lado de las animaciones, el resultado conseguido es bastante fluido, que es justo lo que se pretendía al principio. El jugador es capaz de encadenar acciones si las inicia en las ventanas de tiempo correspondientes. Además, la alta velocidad con la que se ejecutan las animaciones permite que los jugadores no se aburran a la hora de pulsar botones.

4.6. Pruebas

En esta sección se van a comentar las distintas pruebas que se han ido realizando a lo largo de las iteraciones del proyecto. Dichas pruebas no pudieron ser realizadas hasta la segunda animación, ya que el controlador de los personajes no se había implementado hasta ese momento. La encargada de realizar estas pruebas, como ya se ha comentado anteriormente, será mi novia.

4.6.1. Segunda iteración

En esta iteración se incorpora el controlador genérico de los personajes, por lo tanto, ya es posible la realización de pruebas, aunque será contra un objeto estático, ya que en este punto todavía no está implementado el modo multijugador.

Las pruebas realizadas en esta iteración son principalmente de fluidez de movimiento y de encadenamiento de animaciones. La probadora juega con Paladín y se enfrenta a un cilindro de prueba. Pruebas a realizar en esta iteración:

- Comprobar que funcionan todas las animaciones individualmente.
- Comprobar cuál es la sensación que ofrece la velocidad actual de las animaciones.
- Comprobar que se pueden encadenar animaciones y que la sensación de encadenamiento es fluida.

Los resultados de estas pruebas son las que se podrían esperar de la primera batería de pruebas. La probadora declara que, aunque todas las animaciones funcionan correctamente,

no se sienten fluidas. Además, añade que el encadenamiento es bastante extraño, ya que las animaciones “se cortan en mitad” y se reproducen las siguientes. Esto hace que el personaje muestre un movimiento completamente antinatural.

4.6.2. Tercera iteración

En la tercera iteración se realiza la segunda batería de pruebas de este proyecto. Durante el desarrollo de la misma se ha tenido en cuenta el *feedback* de la segunda iteración y se ha dedicado cierto tiempo a la mejora del sistema de animaciones. Para tener un mayor control sobre la cancelación y encadenamiento de animaciones, se han añadido eventos en los clips de las mismas para invocar funciones en el controlador y manejar ciertas variables que permitirán ejecutar nuevas animaciones dependiendo de su estado.

Además, se ha incrementado la velocidad de todas las animaciones para crear una sensación de agilidad. El resultado esperado de las pruebas de esta iteración es que se note una mejoría considerable con respecto al prototipo anterior. Pruebas a realizar en esta iteración:

- Comprobar cuál es la sensación que ofrece la velocidad actual de las animaciones.
- Comprobar que la sensación de encadenamiento de animaciones es fluida.
- Comprobar que el movimiento del personaje no es antinatural.
- Comprobar que se puede golpear a otro personaje en las distintas partes del cuerpo definidas.

Tras realizar esta serie de pruebas, la probadora nota una mejoría general en comparación con la iteración anterior. Según afirma, la velocidad de las animaciones le parece adecuada, aunque también puntualiza que no se podrá comprobar este parámetro con eficacia hasta que no se pueda luchar contra otro jugador.

Con respecto al encadenamiento de animaciones, la probadora declara que la sensación es mucho más fluida, y que la cancelación de las animaciones ya no son en seco, sino que se hace de una forma en la que no se nota la transición.

No obstante, existe una objeción. A veces, si el personaje está realizando una animación, al pulsar un botón tarda bastante tiempo en responder. Este problema puede ser causado por el controlador al no manejar correctamente los parámetros del *Animator Controller*, por tanto, este error será investigado en la siguiente iteración.

Finalmente, se comprueba que los personajes pueden ser golpeados tanto en la cabeza como en el torso. En las piernas no se consigue realizar la prueba, pero se da por buena, al haber sido satisfactorias las otras dos. El motivo por el que no se puede probar esta última zona del cuerpo es porque actualmente no existe en el juego un personaje que pueda golpear en esa parte.

4.6.3. Cuarta iteración

Durante la cuarta iteración, las tareas sobre las animaciones disminuyen, ya que el objetivo de la misma es el diseño e implementación de la interfaz de selección de personaje. No obstante, se dedica parte de esta iteración a la implementación de la solución del error comentado en la iteración anterior.

Tras un día dedicado a este error, el comportamiento del encadenamiento de animaciones parece normal y se decide esperar a que termine la iteración para que la probadora pueda comprobar que ya se ha solucionado.

Como se ha comentado antes, esta iteración está enfocada a la implementación de la interfaz, por lo que las pruebas a realizar no consistirán en probar funcionalidad, sino validar el conjunto de la interfaz. Conjunto de pruebas:

- Comprobar que el error de encadenamiento de animaciones se ha resuelto.
- Validar que la estructura de la interfaz sigue el diseño.

Tras realizar esta batería, la probadora confirma los dos puntos que pretenden validar estas pruebas: tanto el error de las animaciones está corregido como la interfaz sigue el diseño. Por lo tanto, no se considera necesario tener que realizar ninguna modificación para la siguiente iteración.

4.6.4. Quinta iteración

En esta iteración se implementa el modo multijugador. Esto quiere decir que dos jugadores pueden manejar los cursores de la pantalla de selección de personaje, elegir personajes y batirse en combate. Por lo tanto, surgen algunas pruebas relacionadas con este aspecto:

- Comprobar que un jugador se puede unir a la partida pulsando cualquier botón de su dispositivo.
- Comprobar que solo se pueden unir dos jugadores a la partida.
- Comprobar que se puede cambiar de personaje una vez seleccionado otro, siempre que no haya comenzado la partida.
- Probar los cursores y comprobar que los eventos sobre las tarjetas funcionan correctamente.
- Comprobar que, al iniciar partida, los personajes pueden controlarse con los mismos dispositivos que se les ha asignado.

Tras realizar la batería de pruebas, la probadora da por buenas todas menos una. En esta ocasión, los eventos de las tarjetas han dado problemas. El inconveniente ocurre porque, en ocasiones, al quitar el cursor de la tarjeta, esta no vuelve a su tamaño original, como se espera. Por lo tanto, el comportamiento es inconsistente y la probadora afirma que este error ocurre con bastante frecuencia. Por tanto, se le da prioridad a su arreglo en la siguiente iteración.

4.6.5. Sexta iteración

En esta sexta iteración ya tenemos la funcionalidad multijugador, así que se procede a implementar todo el funcionamiento del combate. Además, hay que arreglar el error del *hover* de la iteración anterior. La batería de pruebas queda de este modo:

- Comprobar que, al quitar el cursor de una tarjeta seleccionable, esta vuelve a su tamaño original.
- Comprobar que el resultado final de la escena es fiel con el diseño.

- Comprobar que cuando un jugador gana la partida, aparece una animación con el nombre del ganador y acto seguido, el juego vuelve a la pantalla de selección de personaje.
- Probar animaciones jugando con otro jugador.

Tras varias pruebas, la probadora confirma que el error de la tarjeta ya no sucede, por lo que se puede considerar como arreglado. Además, también afirma que el resultado final de la escena es bastante similar al planteado en el diseño, y espera del desarrollador que en la última iteración realice las mejoras visuales que considere oportunas. También se valida que aparece una animación de fin de partida, con el nombre del personaje y tras unos segundos el juego vuelve a la pantalla de selección de personaje.

En cuanto a la prueba de animaciones, la probadora afirma que la fluidez con la que se juega es bastante buena. Sin embargo, se encuentra un error que bloquea el control del personaje, dejándolo inutilizado para el resto del combate. Por lo tanto, la solución de este error se pasa a la siguiente iteración y se le da la máxima prioridad.

4.6.6. Séptima iteración

En esta última iteración se realizan ciertas mejoras visuales a la escena de combate, como la adición de un nuevo skybox o la adición de niebla al paisaje. Además, se consigue solucionar el error encontrado por la probadora en la iteración anterior, dejando así una aplicación aparentemente libre de errores.

La probadora da el visto bueno a los retoques finales y comprueba que está todo en orden para sacar un ejecutable de la aplicación.

5. Conclusiones y trabajos futuros

5.1. Conclusiones

Crear un sistema de animación fluido lleva casi el mismo tiempo de pruebas que de implementación como tal, ya que el factor más determinante en este caso es la experiencia del usuario. Conseguir la configuración de las animaciones adecuada es un proceso constante de prueba y error, y a veces, lo que para alguien es ideal, para otra persona está mal. Por esta razón, a veces se ha tenido que llegar a puntos intermedios de configuración de velocidad y de cancelación de animaciones para llegar a una satisfacción mutua.

Los colisionadores más simples son los que han resultado ser los más adecuados para la detección de colisiones. En un principio se pensó en utilizar estructuras de datos espaciales complejas, pero no eran adecuadas para un uso en tiempo real y tampoco se necesitaba tanta precisión. Por tanto, se optó por utilizar los colisionadores más simples para permitir su uso en tiempo real, y se ajustaron lo mejor posible a los modelos para mejorar la precisión en las colisiones.

5.2. Trabajos futuros

Al ser un prototipo, esta aplicación deja bastante margen para la creación de un juego completo y queda abierto a la inclusión de nuevas características.

5.2.1. Sistema de puntuación

Para este proyecto no se ha implementado un sistema de puntuación como tal, puesto que se ha dedicado la mayor parte del tiempo a perfeccionar el sistema de animaciones y a la detección de colisiones. Por tanto, este sistema queda pendiente para un futuro. Se podría crear una puntuación basada no solo en el daño infligido al enemigo, sino también en los ataques bloqueados satisfactoriamente. De esta forma se recompensan tanto los estilos de juego ofensivos como los más defensivos. Además, se podría añadir una lista de mejores puntuaciones. Esta puntuación podría ser un objetivo más dentro de las partidas, aportando un incentivo adicional.

5.2.2. Sistema de menús

Como se ha mencionado en el apartado anterior, las animaciones y la detección de colisiones han sido las protagonistas de este prototipo, y se han dejado de lado otros aspectos que se dan por hecho en videojuegos comercializados, como son los menús. Para que este prototipo se pueda mostrar al público, necesitaría la adición de un menú principal, donde se pueda iniciar una nueva partida y un menú de opciones, donde se puedan modificar todos los ajustes relacionados con el juego, como pueden ser el volumen, la calidad de los gráficos o incluso la modificación de los controles del juego para los distintos dispositivos.

5.2.3. Nuevos personajes

Uno de los requisitos de este proyecto era facilitar la labor de los diseñadores futuros a la hora de integrar nuevos personajes en el proyecto. Por lo tanto, uno de los trabajos futuros sin duda alguna debe ser la inclusión de nuevos personajes con características únicas.

5.2.4. Nuevos movimientos

Este prototipo permite una serie muy limitada de movimientos en los personajes. Sería ideal añadir más movimientos, en concreto, más variedad de ataques e incluso un sistema de combos para encadenar ataques. Este último punto podría aportar bastante en la idea del sistema de puntuación, obteniendo más puntos dependiendo de la calidad de los combos.

6. Anexos

6.1. Anexo I. Manual de usuario

Para ejecutar la aplicación, hacer doble click sobre el ejecutable TFG.exe. Esto debería iniciar la aplicación y llevaría directamente a la pantalla de selección de personaje.

Para que un jugador se una a la partida debe conectar su dispositivo (teclado o mando) y pulsar cualquier botón. Automáticamente aparece un cursor para el jugador. Para mover el cursor, utilizar las teclas WASD o las flechas del teclado, o el joystick izquierdo del mando. Para seleccionar un personaje, situar el cursor sobre alguna de las tarjetas y pulsar Q en el teclado o la X en el mando. Una vez que los dos jugadores escojan personaje, se puede iniciar partida pulsando Enter en el teclado o Start en el mando.

Una vez dentro de la partida, los dos jugadores deben combatir hasta que la vida de un personaje de uno de ellos llegue a cero. Para jugar, estos son los controles:

- Teclado
  - Moverse: teclas AD o flechas izquierda y derecha.
  - Atacar: Q.
  - Bloquear: mantener pulsado E.
  - Esquivar hacia atrás: Barra espaciadora.
  - Hacer voltereta hacia delante: moverse hacia delante y pulsar la barra espaciadora.
- Mando
  - Moverse: joystick izquierdo.
  - Atacar: X.
  - Bloquear: mantener pulsado el cuadrado.
  - Esquivar hacia atrás: Triángulo.
  - Hacer voltereta hacia delante: moverse hacia delante y pulsar triángulo.

7. Bibliografía

Capcom. (10 de Septiembre de 2021). *Street Fighter*. Obtenido de https://streetfighter.com/?lang=es

FiftySounds. (10 de Septiembre de 2021). *FiftySounds*. Obtenido de https://www.fiftysounds.com/

Grimm, E. (10 de Septiembre de 2021). *Tubersongs by Ean Grimm*. Obtenido de https://www.tubersongs.com/

Klipartz. (19 de Julio de 2021). Obtenido de https://www.klipartz.com/es/sticker-png-bqghh

Nintendo. (9 de Septiembre de 2021). *Super Smash Bros. Ultimate*. Obtenido de https://www.smashbros.com/es_ES/index.html

PEGI, P. E. (8 de Septiembre de 2021). *PEGI*. Obtenido de https://pegi.info/es/node/59

rpgwhitelock. (10 de Septiembre de 2021). *AllSky Free - 10 Sky / Skybox Set*. Obtenido de https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014

SonidoefectosFX. (10 de Septiembre de 2021). *YouTube*. Obtenido de https://www.youtube.com/watch?v=fDQEWDNPnZ8&ab_channel=SonidoefectosFX

Studios, B. (10 de Septiembre de 2021). *Free Island Collection*. Obtenido de https://assetstore.unity.com/packages/3d/environments/landscapes/free-island-collection-104753

Symphony, M. (10 de Septiembre de 2021). *BreakingCopyright*. Obtenido de https://breakingcopyright.com/song/makai-symphony-dragon-castle

Veselov, N. (10 de Septiembre de 2021). *NVJOB Water Shaders V2.x*. Obtenido de https://assetstore.unity.com/packages/vfx/shaders/nvjob-water-shaders-v2-x-149916

wFonts. (20 de Agosto de 2021). Obtenido de https://www.wfonts.com/font/astron-boy