Universitat de Girona
Escola Politècnica Superior

TRABAJO FINAL DE GRADO

Diseño y desarrollo de un videojuego de pádel

Alumno Sandro Bortolotti Montón

Estudio Diseño y Desarrollo de Videojuegos

Documento	Memoria
Tutor	Gustavo Patow
Departamento	Informática, Matemática Aplicada y Estadística
Área	Lenguajes y Sistemas Informáticos
Convocatoria	Junio/2022

Agradecimientos

A mi tutor, Gustavo Patow, por haberme dado ánimos y haber realizado el seguimiento de este proyecto.

A mi hermano por haberse prestado voluntario para hacer las pruebas conmigo y sugerirme mejoras.

A mis amigos y compañeros de clase que valoraron objetivamente mi proyecto y me aportaron ideas en momentos de poca inspiración.

A mi pareja que me ha animado constantemente y ha confiado en mi capacidad para afrontar este proyecto.

Lista de Figuras

Lista de Tablas

1. Introducción y objetivos

1.1 Introducción

Hoy en día, los videojuegos deportivos están de capa caída. Si bien es cierto que el número de jugadores aún es elevado, las novedades de este tipo de juegos año a año son muy escasas, dando a los jugadores la sensación que cada año están jugando de nuevo al juego del año anterior. Existen videojuegos basados en los deportes más populares, sea de forma realista o arcade, donde su gancho principal es la competitividad entre jugadores o contra la IA.

Convertir en videojuego los deportes más populares es un acierto si se mira solamente a la masa de público objetivo, pero hay muchos deportes que son muy divertidos de practicar y no son tan comunes por falta de instalaciones o por su alto precio. En este caso los videojuegos pueden ser una solución para recrear en el jugador la sensación de estar jugando a ese deporte, o quizá solo sirva para que conozca el deporte, que ya sería un gran paso.

1.2 Idea principal de Padel & Friends

Padel & Friends es el nombre del videojuego que se desarrolla para este proyecto, basado en el deporte antes mencionado, pádel. Mezclando mecánicas arcade con golpes básicos de este deporte siempre buscando el equilibrio para que el resultado sea un videojuego entretenido. Se podrá elegir el personaje con el que se quiera jugar y eso determinará el estilo de juego y la habilidad especial que se quiera usar.

1.3 Motivaciones

La motivación principal detrás de este proyecto es conseguir hacer un juego divertido, que sea competitivo, pero no frustrante. Todo esto se realizará aplicando los conocimientos adquiridos a lo largo del grado y en algunos casos aprendiendo cosas nuevas que serán útiles en el futuro. Por todo lo explicado anteriormente, he decidido diseñar y desarrollar un videojuego de pádel, aprovechando sus mecánicas divertidas y su auge en cuanto a popularidad en los últimos años. Para conservar el gancho de los juegos deportivos, se podrá jugar en multijugador local y contra la IA.

1.4 Motivaciones personales

Desde pequeño me han encantado los deportes y de mis primeros recuerdos relacionados con los videojuegos son sentado en el sofá junto a mi hermano mayor jugando a *FIFA Football 2005* (Figura 1). Enseguida me entró la curiosidad de cómo habían hecho que el árbitro pitara falta cuando lo era y, sobre todo que te enseñara tarjeta amarilla o roja cuando la falta era más grave. Esa curiosidad inicial me llevó, luego de varios años, a estudiar este grado. Es por eso por lo que he querido acabar el grado presentando un videojuego deportivo para que, en el mejor de los casos, si algún niño juega a mi juego le entre la misma curiosidad que me entró a mí.

Figura 1: Logo de FIFA Football 2005

1.5 Propósito y objetivo del proyecto

El propósito de este proyecto es diseñar un videojuego de un deporte del que no se ha hecho prácticamente ningún videojuego, con las dificultades que eso conlleva.

El objetivo de este TFG es crear un videojuego de pádel que tenga estilos de juego diferentes y que sea sencillo para el usuario aprender a jugarlo. Para llegar a ese resultado, se deberán realizar las siguientes tareas:

- Estudiar como simplificar el deporte para transformarlo en un videojuego.
- Diseñar una jugabilidad entendedora.
- Desarrollar el movimiento de los jugadores.
- Desarrollar el comportamiento de la pelota según el tipo de golpe recibido.
- Implementar el sistema de puntuación del marcador.
- Desarrollar interfaces que aporten claridad al juego.
- Modelar los jugadores y el entorno.
- Crear animaciones para el jugador.

1.6 Distribución de tareas

Al ser un videojuego creado enteramente por una sola persona, hará falta hacer tareas de todas las áreas, a diferencia de los desarrollos habituales que hay profesionales que solo trabajan en su área. A continuación, se muestra una tabla (Tabla 1) con la distribución porcentual del trabajo que se realizará en cada área de trabajo.

Narrativa	5%
Estética	25%
Mecánicas	35%
Tecnología	35%

Tabla 1: Distribución porcentual de tareas

Para comenzar, el juego tiene personajes que no son propiamente jugadores de pádel y tiene una habilidad acorde a su apariencia. Hilar personaje y habilidad tiene cierto componente narrativo.

La coherencia estética me parece importante en un videojuego, es por eso por lo que la mayoría de los entornos, personajes e interfaces son creados por mí. También quiero mencionar que el apartado artístico de los videojuegos es la parte que más me gusta y es en la que más interés tengo en mejorar y aprender.

Dentro de las mecánicas se incluyen las diferentes características que conforman la jugabilidad. En este apartado entran los diferentes tipos de golpes, el movimiento, las habilidades y las reglas deportivas, un ejemplo son las faltas de saque y el propio marcador. Al no tener referentes en juegos de este deporte, es un reto diseñar todas estas mecánicas para que sean divertidas y entendedoras. También es ese reto el que me hace mantener la motivación en este proyecto.

De nuevo, como las mecánicas son diseñadas por mí, hay muchas cosas que en mi cabeza son perfectas y funcionan a la primera, y la realidad es que no funciona así. Es más sencillo coger mecánicas existentes y aplicarlas a un videojuego, porque ya se sabe que funcionaran. De hecho, ya se han visto funcionar. En cambio, diseñarlas de cero (algunas inspiradas en otros deportes, como el tenis) es más complejo y requiere mucha prueba y error. Es por esta razón, mecánicas y tecnología son el área con mayor porcentaje de trabajo.

2. Estudio de viabilidad

En este capítulo se realizará una valoración de la viabilidad del proyecto, valorando los recursos técnicos, requisitos tecnológicos, recursos humanos y gastos económicos. Esta valoración se lleva a cabo para determinar si nuestro juego se puede desarrollar y si tiene cabida en el mercado actual.

2.1 Recursos técnicos

Este trabajo se ha realizado con un ordenador portátil con un S.O. Windows 10 Pro de 64 bits y las principales especificaciones del ordenador son las siguientes:

- Procesador: Intel(R) Core (TM) i7-10870H
- Tarjeta gráfica: NVIDIA GeForce RTX 2060
- Memoria RAM de 16GB

También se ha requerido el uso de una tableta gráfica Wacom Intuos S para fines artísticos.

2.2 Requisitos tecnológicos

2.2.1 Hardware

Para el desarrollo de este proyecto, con relación a *hardware*, sólo se ha necesitado el ordenador portátil antes mencionado y puntualmente una tableta gráfica.

2.2.2 Software

En cuanto a *software* se han utilizado variedad de programas para hacer los personajes, interfaces, el propio juego… Seguidamente se detallarán estos programas.

2.2.2.1 Unity

Unity es uno de los motores más utilizado por los desarrolladores independientes debido a sus grandes prestaciones y a su comunidad en internet que resuelve muchas dudas. Se considera el motor de videojuegos estándar para empezar a desarrollar videojuegos, aunque existen serios competidores. En este caso se ha usado la versión 2019.4.19f por que ya había trabajado anteriormente en esta versión y por comodidad.

Figura 2: Logo de Unity

Para la programación, Unity usa el lenguaje C# para crear componentes y estos componentes se pueden asociar a diferentes objetos del juego. También ofrece gran variedad de componentes preestablecidos que ayudan enormemente. En este proyecto se utilizan tanto componentes preestablecidos como componentes creados.

2.2.2.2 Visual Studio Code

Visual Studio Code es un editor de código que facilita el desarrollo a la hora de escribir. Con una interfaz simple, ayuda a visualizar mejor el programa usando distintos colores y permite ver claramente los errores. Permite instalar extensiones para facilitar aún más la tarea de programar, como por ejemplo, texto predictivo o formatear el código de forma automática.

Figura 3: Logo de Visual Studio Code

En este proyecto lo he usado como editor de código con una extensión que recomienda código para autocompletar con funciones de Unity.

2.2.2.3 Blender

Blender es el programa de modelado 3D de código abierto por excelencia. Usado por numerosos artistas debido a las posibilidades que ofrece que nada tienen que envidiar a su competencia. Esta herramienta permite principalmente modelar, esculpir, animar y texturizar objetos en 3D.

Figura 4: Logo de Blender

En este proyecto se ha usado la versión Blender 2.91.2 para realizar los personajes (modelado, esculpido, texturizado y animado) y los entornos (modelado y texturizado).

2.2.2.4 Inkscape

Inkscape es un editor de gráficos vectoriales libre y gratuito. En Inkscape se puede crear y editar diagramas, líneas, gráficos, logotipos, e ilustraciones complejas en 2D.

Figura 5: Logo de Inkscape

En este proyecto se ha usado para diseñar las interfaces del videojuego.

2.3 Recursos humanos

Habitualmente, los equipos de desarrollo están constituidos por diferentes perfiles de profesionales dependiendo de su especialización. Los perfiles recurrentes son:

- Diseñadores: responsables de idear la jugabilidad, mecánicas, narrativa…
- Programadores: encargados de implementar las ideas de los diseñadores.
- Artistas: responsables de la estética del videojuego.

Este proyecto es realizado por una sola persona que se encarga de todos los apartados.

2.4 Recursos económicos

Para el desarrollo del proyecto ya dispongo de todo el material necesario, así que no hace falta considerar ningún gasto adicional. A continuación (Tabla 2) se muestra un listado del precio de los elementos necesarios para hacerse una idea del supuesto coste en caso de no disponer de ellos.

Recurso	Coste (€)
Ordenador portátil	1100
Tableta gráfica	60
Unity	0
Visual Studio Code	0
Blender	0
Inkscape	0

Tabla 2: Desglose de recursos y su precio

Como se ha mencionado anteriormente, el proyecto será realizado por una sola persona, el autor de esta memoria, y es por eso por lo que el sueldo total no se tiene en cuenta.

2.5 Derechos legales

Casi toda la integridad de elementos usados en este proyecto son de creación propia. En caso de usar elementos de fuentes externas, siempre será con licencias de uso libre y mencionando su autoría si fuera necesario. De esta manera nos evitamos gastos innecesarios para un proyecto pequeño como este.

2.5 Estudio de mercado

Una vez sabemos que el proyecto es viable en términos económicos, es fundamental investigar el mercado actual para detectar posibles competidores. Hay que tener en cuenta aspectos como la jugabilidad, el estilo, posibilidad de jugar con amigos, etc. En función de estos parámetros sabremos si nuestro proyecto se diferencia del resto y tiene cabida en el mercado.

2.5.1 Metodología de búsqueda

Se utiliza el motor de búsqueda Google para listar los resultados más similares a nuestra propuesta. Utilizando palabras clave como las siguientes:

- Videojuegos de pádel
- Padel videogames

Los resultados son muy escasos, denotando que existe un hueco en el mercado que se puede ocupar. Aunque los resultados sean limitados, siempre es interesante investigar sus puntos débiles y sus fortalezas.

2.5.2 Juegos seleccionados

Los resultados han simplificado la tarea de selección de videojuegos, ya que solamente se han encontrado 3 resultados significativos. Simplificar la tarea de selección no es siempre algo bueno, la carencia de resultados significa un incremento en la dificultad de desarrollar mecánicas nuevas en vez de inspirarte en las existentes. Por esa razón he querido incrementar la lista de resultados buscando también referentes en videojuegos de tenis.

2.5.2.1 Juegos de pádel

2.5.2.1.1 IPadelGame

Es considerado el primer videojuego de pádel de la historia**.** Permite realizar partidos contra la CPU, así como entrenamientos. La versión actual sólo se encuentra disponible a través del App Store de Apple para iPhone, iPod Touch e iPad (ver Figura 6).

Figura 6: Representación de IPadelGame

Es un juego publicado en 2012 que cuenta con más de 100.000 descargas en AppStore. Permite hacer diferentes tipos de golpes, pero el jugador se posiciona automáticamente en la pista, cosa que hace que el juego sea aburrido.

2.5.2.1.2 Heroes of Padel

Heroes of Padel ha sido desarrollado por Zarappss Games y se puede descargar gratuitamente para Apple y Android, teniendo más de 100.000 descargas en esta última. Se nos permite elegir entre 32 jugadores con diferentes valores cuanto a velocidad, fuerza y precisión (Ver Figura 7).

Figura 7: Heroes of Padel in-game

Este juego permite customizar al personaje que controla el jugador, y tiene una estética lograda. Los controles son muy difíciles de dominar, el jugador no tiene percepción de dónde se sitúa la pelota debido a la ubicación de la cámara.

2.5.2.1.3 Padel Clash

Padel Clash también cuenta con más de 100.00 descargas en Android. Ofrece una experiencia similar al anterior juego mencionado (Heroes of Padel) con el apartado gráfico muy deficiente. También se puede golpear de diferentes maneras y personalizar al personaje.

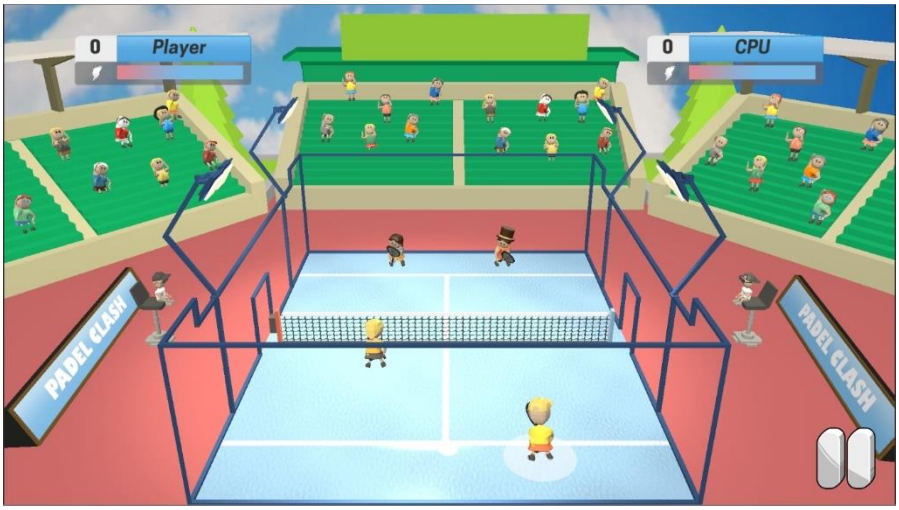

Figura 8: Padel Clash in-game

2.5.2.2 Juegos de tenis

En estos juegos nos fijaremos en como resuelven las mecánicas comunes con el pádel, como puede ser el golpeo y el movimiento, sin fijarnos mucho en los resultados en cuanto a ventas. La larga historia de los juegos de tenis hace pensar que sus mecánicas están muy depuradas y por esa razón parecen interesantes de investigar.

2.5.2.2.1 Tennis World Tour

Juego publicado en 2018 por Breakpoint, ofrece una experiencia realista en la que se recompensa el juego estratégico ante el juego preciso. El movimiento es libre después de golpear, pero en el turno de golpear el personaje es guiado con la animación de golpear para acercarlo a la pelota. La fuerza del golpe se carga manteniendo el botón apretado por lo que una buena posición es clave. A veces la pelota pasa cerca del jugador e ignora la pelota, eso genera frustración, da lugar a interpretar que no queda claro cuánto espacio abarca el jugador.

Figura 9: Tennis World Tour in-game

2.5.2.2.2 AO Tennis 2

Se trata de un juego de tenis, desarrollado por Big Ant Studios publicado en 2020. Apuntar donde el jugador quiere lanzar la pelota y mantener el botón apretado para aplicar la fuerza que se desee es la mecánica básica de este juego. También tiene en cuenta el cansancio del jugador.

Figura 10: AO Tennis 2 in-game

2.5.2.2.3 Mario Tennis Aces

La saga Mario Tennis es el juego de tenis arcade por excelencia, característico por sus personajes carismáticos complementados con sus golpes y habilidades especiales. Mario Tennis Aces publicado en 2018, ha sido desarrollado por Camelot Software Planning y distribuido por Nintendo para la consola Nintendo Switch. Su modo de juego más destacado es el multijugador online.

Figura 11: Mario Tennis Aces in-game

2.5.3 Comparación con el mercado

Nombre	Plataformas	Estilo jugable	Estilo estético	+ 1 jugador
IPadelGame	Apple	Arcade	Cartoon	No
Heroes of Padel	Apple, Android	Realista	Realista	No
Padel Clash	Android	Arcade	Minimalista	No
Padel & Friends	PC	Arcade	Cartoon	Si

Tabla 3: Comparación de productos existentes

Padel & Friends es el único juego para la plataforma de PC y también el único que ofrece la posibilidad de jugar con amigos. Estas dos características lo convierten en el producto más destacado de su sector, complementado de igual manera con las mecánicas extraídas de los juegos de tenis.

2.5.4 Modelo de negocio

Para definir nuestro modelo de negocio es necesario conocer qué modelos existen:

- De pago: El modelo tradicional, se realiza un pago único por la obtención del juego en su totalidad.
- De suscripción: Diferentes pagos sostenidos en el tiempo para conservar el acceso al juego.
- Gratuitos: El acceso al juego es gratuito, normalmente se ofrece la posibilidad de pagar para tener contenido exclusivo. Hay que evitar caer en el *pay-to-win* dando ventajas a los jugadores que han pagado.

El modelo que se ajusta más a este proyecto es el de pago tradicional, ofreciendo todo el contenido una vez se ha efectuado el pago. El modelo de hacerlo gratuito y ofrecer personajes nuevos a los usuarios que han pagado también se ajustaría correctamente al tipo de juego, pero al ser un juego con multijugador local tiene más sentido que dispongan de todos los personajes.

2.6 Público objetivo

Una forma de definir los distintos perfiles de jugador es utilizando la clasificación de Richard Bartle que propone cuatro perfiles de jugador según el tipo de jugabilidad que prefieren:

- Triunfador: Le gusta resolver retos, llegando incluso a inventarse los suyos propios y demostrar su habilidad en el juego. Requiere novedades constantes ya que abandona el juego una vez ha conseguido todos los logros.
- Explorador: Prefiere descubrir áreas y secretos escondidos, se siente restringido cuando se le impone un tiempo límite y no se le permite mirar alrededor todo el tiempo que desearía.
- Socializador: Para él lo mejor es interactuar con los demás jugadores y usa el juego como una herramienta para conocer a otros jugadores.
- Competidor: Prefiere competir contra otros jugadores antes que contra la IA.

Entre estas categorías, nuestro jugador ideal sería una mezcla entre triunfador, socializador y competidor. Como en el juego se puede jugar con amigos de forma local, los socializadores podrán jugar mientras hablan con amigos, mientras que los triunfadores y los competidores tendrán siempre alguien a quien ganar para sentirse bien.

2.7 Conclusiones del estudio de viabilidad

En el apartado técnico y tecnológico los gastos son nulos ya que los componentes utilizados no fueron comprados específicamente para este proyecto, y todos los programas usados para las distintas fases del desarrollo son completamente gratuitos.

Finalmente, en el estudio de mercado se ha observado una falta de juegos basados en el pádel, así que el proyecto representa una propuesta original y innovadora. Por lo tanto, se puede afirmar que nuestro proyecto tiene cabida en el mercado actual.

3. Planificación

Este capítulo presenta las tareas que serán necesarias realizar a lo largo del desarrollo y su temporalización, con el fin de controlar y organizar el tiempo de desarrollo. En un equipo de desarrollo convencional, se acuerdan unos límites de tiempo para sincronizarse y trabajar de forma más eficiente, pero en este caso, al tratarse de un proyecto realizado por una sola persona, la cual realizará todas las tareas, la planificación podrá variar en función de la importancia de cada tarea o de la motivación de realizarla, siempre intentando no superar el límite de tiempo establecido, pero sin ser muy estrictos.

3.1 Metodología de trabajo

En la práctica totalidad de los casos, el desarrollo de un videojuego está dividido en tres etapas fundamentales: el diseño (de mecánicas, de personajes, de estética, etc.), la elaboración (implementación en caso de programación, creación de elementos artísticos como el sonido o modelos 3D) y el testeo o realización de pruebas. Habitualmente se suele seguir este orden, pero siempre se puede volver a atrás para mejorar o rediseñar elementos del producto. En una situación ideal, el diseño y la implementación se realizarían una sola vez, pero en la práctica a menudo se tiene que retroceder para corregir alguna implementación.

De esta forma podemos decir que seguiremos un método cíclico o iterativo, en el que cada elemento primeramente se diseñará, a continuación, se creará una primera versión y últimamente se evaluará el resultado y se decidirá si es aceptable o, por caso contrario, habrá que retroceder. En cualquier caso, se podrá volver a la fase de diseño antes de haber llegado a la fase de evaluación si ocurriera un problema o si surgiera una nueva idea, tal y como se puede ver en la Figura 12.

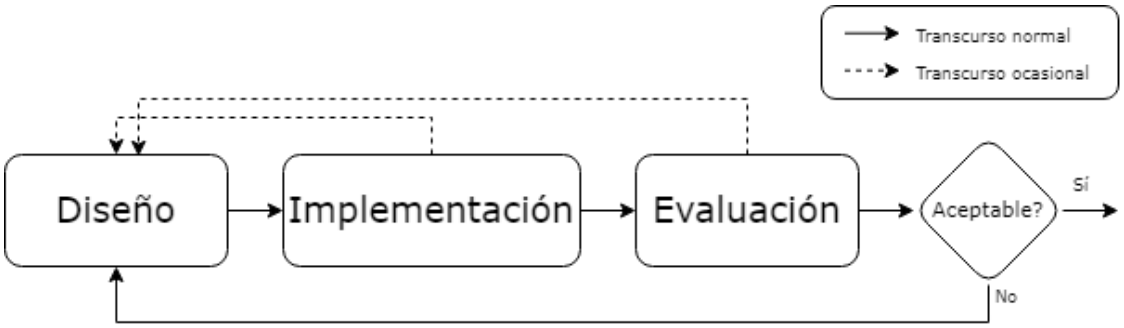

Figura 12: Mapa visual de la metodología de trabajo

3.2 Plan de trabajo

Para realizar el plan de trabajo y saber qué tareas son necesarias, he confeccionado un diagrama (Figura 13). En éste se pueden ver de manera general los principales puntos divididos en 6 categorías: Diseño, Mecánicas, Integración, Creación, Pruebas y Documentación.

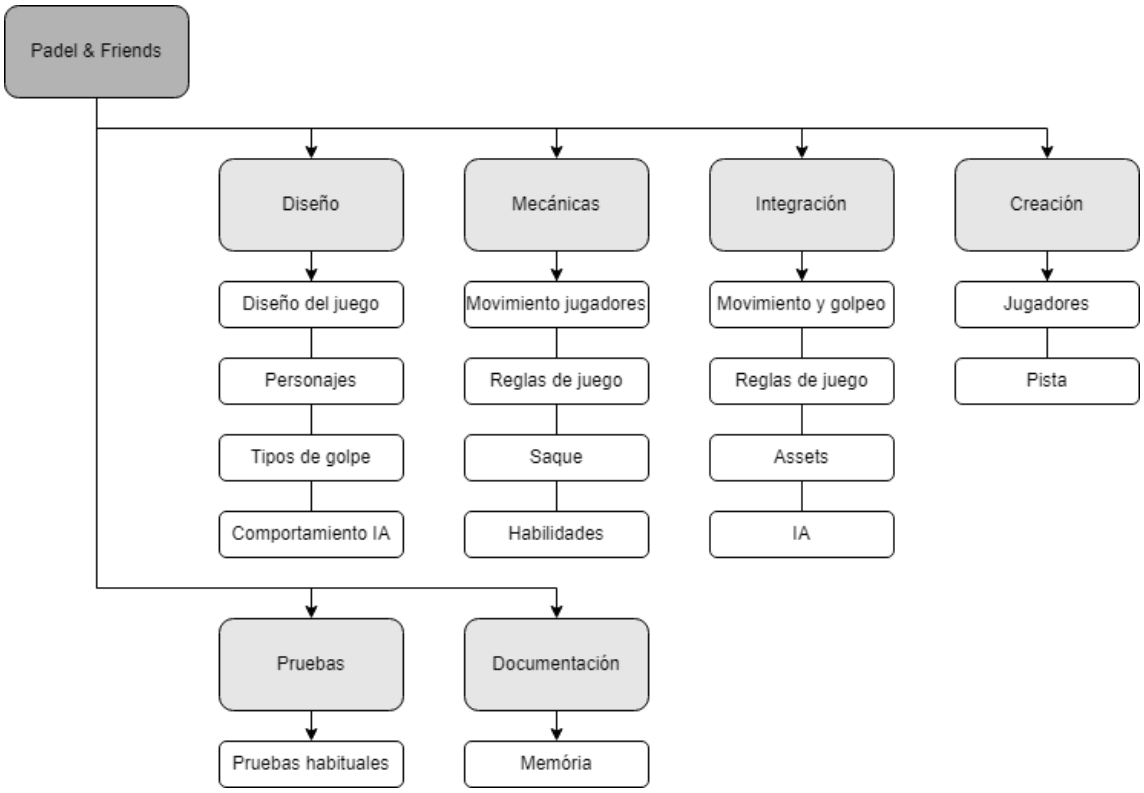

Figura 13: Diagrama de tareas según categoría

3.2.1 Descripción de las tareas

En este apartado se explica de forma breve las tareas mencionadas en el punto anterior (3.2 Plan de trabajo). También se estimará el tiempo que se destinará a cada tarea.

3.2.1.1 Diseño

- **Diseño del juego** (2 semanas): Tener una idea general del juego (modalidades de juego, controles del juego, comportamiento de la pelota, etc.)
- **Personajes** (1 semana): Características de los personajes y habilidad asociada.
- **Tipos de golpe** (3 días): Como adaptar los golpes de pádel a un videojuego.
- **Comportamiento IA** (1 semana): Simplificar el comportamiento humano en la pista y transformarlo en reglas.

3.2.1.2 Mecánicas

- **Movimiento de los jugadores** (1 semana): Vincular controles con el movimiento, establecer velocidad, movimiento según orientación del jugador.
- **Reglas del juego** (1 semana): Decidir qué reglas aplicar al videojuego y cómo reproducirlas.
- **Saque** (1 semana): Establecer ubicaciones de saque y resto[1]. Elegir ubicación según si es punto par o impar, implementar sistema de doble falta.
- **Habilidades** (1 semana): Comprobar que las habilidades son útiles y atractivas, compensar habilidades.

[1] Acción de devolver el saque

3.2.1.3 Integración

- **Movimiento y golpeo** (1 mes): Establecer valores correctos de velocidad y fuerza de golpeo según tipo de golpe, características del personaje, y establecer volumen de alcance del jugador.
- **Reglas de juego** (2 semanas): Ajustar reglas de juego al tamaño de la pista y contabilizar los puntos asignándolos al jugador adecuado.
- **Assets** (1 semana): Integrar los elementos 3D al juego, ajustar el tamaño y aplicar materiales.
- **IA** (1 mes): Crear un oponente capaz de colocarse bien en la pista y devolver las pelotas que se le lanza variando su tipo de golpe.

3.2.1.4 Creación

- **Jugadores** (2 meses): Modelado, esculpido, texturizado y animación de varios personajes.
- **Pista** (2 días): Modelado y texturizado de la pista acorde a las dimensiones reales.

3.2.1.5 Pruebas

- **Pruebas habituales** (indefinido): Siempre después de una nueva incorporación se prueba que funciona correctamente.

3.2.2 Cronograma

El cronograma de este capítulo (ver Figura 14) ilustra los tiempos de realización de las tareas y su orden cronológico. Se ubica el inicio del cronograma a principios de noviembre y el final a finales de mayo.

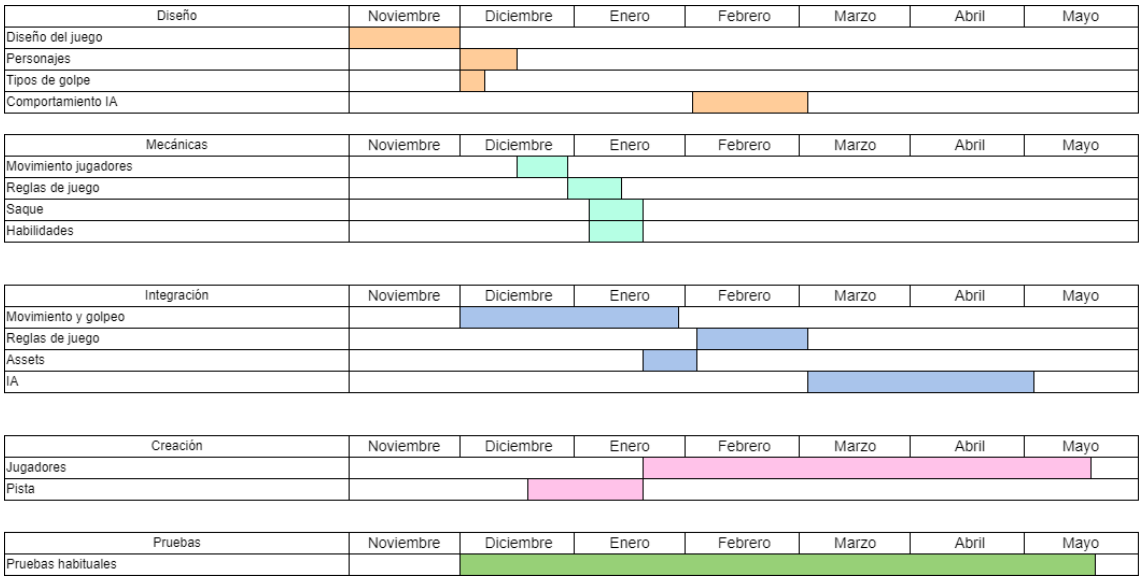

Figura 14: Cronograma de tareas según categoría

4. Marco de trabajo y conceptos previos

En este capítulo se introducirán los conceptos necesarios para facilitar la comprehensión y el seguimiento del documento de diseño del videojuego. Para hacerlo, se hablará de conceptos básicos del pádel para que todo tipo de público pueda entender el vocabulario utilizado de ahora en adelante.

4.1 Básicos del pádel

4.1.1 Puntuación

El pádel es un deporte muy influenciado por el tenis. Es por esa razón que conserva muchos aspectos comunes, por ejemplo, el sistema de puntuación:

- Se consigue un **punto** cuando se consigue que la pelota bote dos veces en el campo rival o cuando el rival no consigue devolverla. Los puntos se contabilizan en el siguiente orden: 15/30/40/Iguales/Ventaja.
- Una vez que un jugador gana 4 puntos o, en caso de empate, gane 2 puntos más que su adversario, se le otorgará un **juego**.
- El primero en llegar a 6 juegos o en caso de empate, en caso de empate a 5 juegos se deberán jugar dos más, hasta ganar por 7-5 pero si se produce empate a 6 juegos se aplicará el *tiebreak* o desempate. Al ganador se le otorgará un **set**.
- En el *tiebreak* se juega al primero que llegue a 7 puntos, en caso de empate se necesitará ganar 2 puntos más que el rival.

4.1.2 Uso de las paredes

En el pádel, a diferencia del tenis, hay paredes delimitando el campo de juego. Hay que diferenciar entre dos tipos de paredes: cristal y reja. Se diferencian en el tipo de rebote que la pelota realiza en ellos, en el cristal la pelota rebota de forma normal y en la reja rebota de forma irregular, resulta impredecible saber hacia dónde va a rebotar. La pelota puede botar en la pared siempre y cuando primero haya botado en el suelo, con una excepción, la pelota puede rebotar directamente en el cristal del propio campo del jugador, importante remarcar que sólo puede rebotar en el cristal y no en la reja.

4.1.3 Saque

En el saque, se debe sacar detrás de la línea de saque, por debajo de la cintura y la pelota tiene que botar en el cuadrado situado en diagonal respecto al lado del sacador (Ver Figura 15). El sacador tiene dos oportunidades para hacer un saque válido.

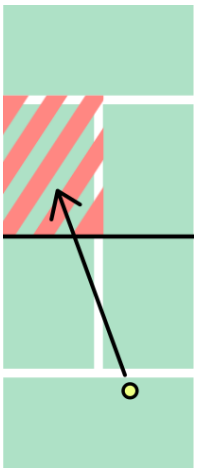

Figura 15: Esquematización del saque

En los puntos pares se saca desde la derecha y en los impares desde la izquierda. El jugador que saca se alterna en cada juego.

4.1.4 Tipos de golpe

Existen muchos tipos de golpe, pero simplificando nos quedaremos con estos 4 golpes:

- Drive o revés: Golpe plano.
- Globo: Golpe elevado, útil cuando el oponente está cerca de la red.
- Volea: Golpe plano o ligeramente hacia abajo sin que la pelota haya botado, útil cuando el jugador se sitúa cerca de la red.
- Remate: Golpe fuerte hacia abajo, con este golpe se busca que la pelota rebote en el cristal para que vuelva a tu campo o para sacarla del mismo.

Todos estos golpes, menos el remate, se puede dar de derechas o de revés, dependiendo de con que superficie de la pala se golpea a la pelota (ver Figura 16 y 17).

Figura 16: Esquematización de un golpe de derechas

Figura 17: Esquematización de un golpe de revés

5. Diseño del videojuego

En este apartado se explica el diseño de Padel & Friends, incluyendo las decisiones que se tomaron y por qué motivo. Se hablará del movimiento del personaje, el golpe, el comportamiento de la IA, etc.

5.1 Objetivos

Como ya se ha mencionado anteriormente, los objetivos del diseño de este proyecto son:

- Conseguir un resultado divertido.
- Apto para todos los públicos.
- Recrear el deporte de una forma entretenida y competitiva.

Se ha decidido usar un controlador de tipo Xbox Controller ya que los juegos deportivos son más cómodos de jugar con este tipo de controladores.

5.2 Modos de juego

El pádel se juega por parejas de modo que juegan dos contra dos, pero para este proyecto se ha optado por hacer un juego para dos jugadores como máximo para facilitar el poder jugar en forma local, dado que es más sencillo encontrar una persona para jugar que tres más. Por el otro lado, también haría falta disponer de 4 mandos para poder jugar.

Este proyecto en un inicio sólo iba a contar con el modo multijugador local, pero en una fase del desarrollo hizo falta un oponente para probar las mecánicas, y de ahí surgió el modo entrenamiento. De esta forma el videojuego es para de uno a dos jugadores.

5.2.1 Multijugador local

En este modo se puede jugar con otra persona *in situ* mediante una pantalla dividida verticalmente (ver Figura 18).

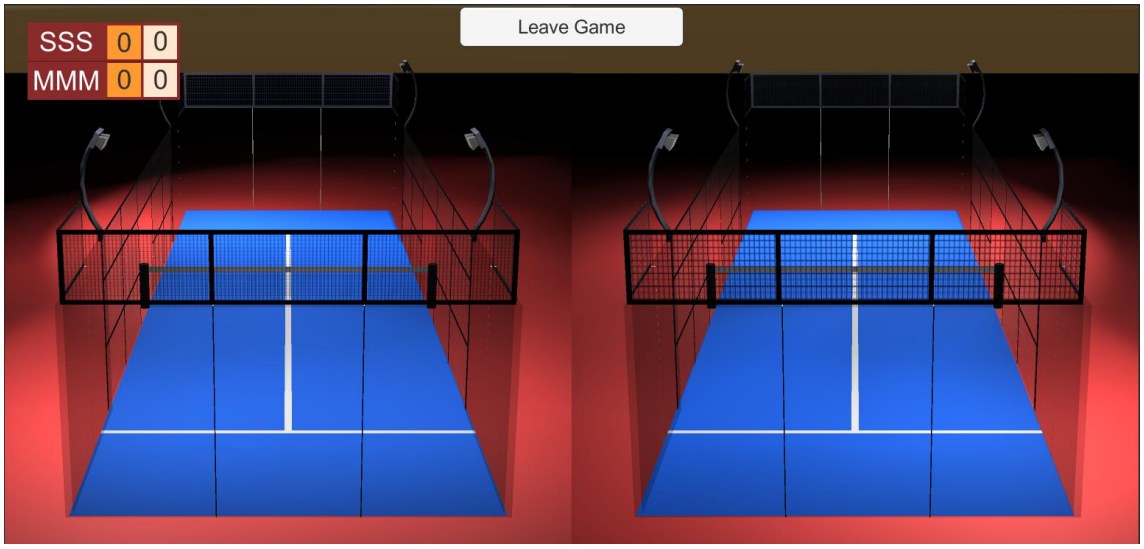

Figura 18: Pantalla dividida verticalmente

5.2.2 Entrenamiento

Este modo es individual, por lo tanto, no hace falta pantalla dividida. El rival es una IA, cuyo comportamiento se explicará más adelante.

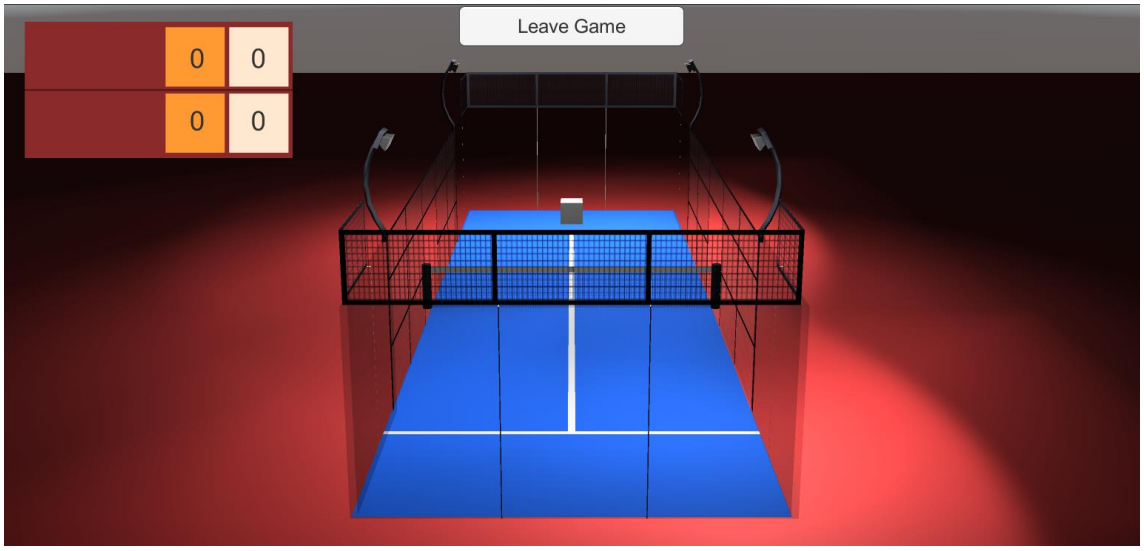

Figura 19: Vista del modo entrenamiento

5.3 Cámaras

Por falta de referentes, la ubicación de la cámara era un problema por resolver. Hacía falta encontrar una posición dónde se viera bien todo el campo, así como la posición de la pelota y los jugadores. Como inspiración, se ha tomado como referencia la posición de la cámara de la realización de World Padel Tour (WPT), que es un campeonato de pádel mundial.

Figura 20: Captura de un partido de WPT

Como se puede ver, para tapar lo menos posible se hace coincidir el inicio de la reja con el bajo de la red y el alto de la reja con la línea de saque del campo opuesto. Como se puede ver en la Figura 21, así se ha hecho.

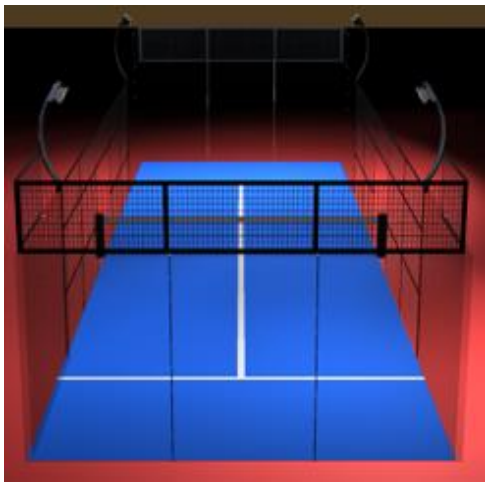

Figura 21: Posición de la cámara imitando a la de WPT

5.3.1 Cámara dividida

Por motivos de equidad, se ha debido hacer una pantalla dividida para no beneficiar a ningún jugador. Para ello es necesario usar dos cámaras, una para cada lado de la pista (ver Figura 22).

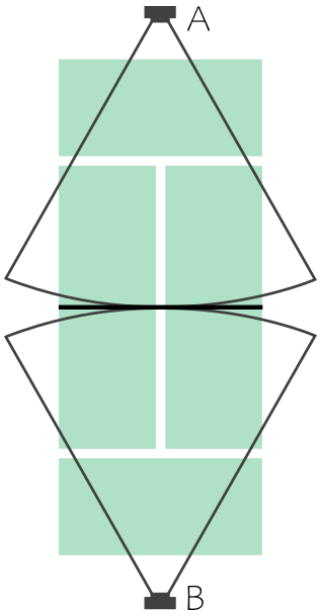

Figura 22: Esquematización de las cámaras

De ahora en adelante se hablará de las cámaras como cámara A y cámara B en relación con la Figura 22.

Unity permite dividir la pantalla en varias cámaras de manera sencilla. Seleccionando la cámara que se quiera modificar, en el *Inspector* podemos encontrar los parámetros del *Viewport Rect*, los cuales modificaremos para configurar la pantalla dividida. Estos parámetros son:

- X: Posición del vértice inferior izquierdo de la imagen, en el eje horizontal, siendo 0 izquierda y 1 derecha.
- Y: Posición del vértice inferior izquierdo de la imagen, en el eje vertical, siendo 0 abajo y 1 arriba.
- W: Ancho de la imagen.
- H: Alto de la imagen.

Entonces, para configurar la cámara A situada a la izquierda, estableceremos los valores de la Figura 23.

Figura 23: Configuración cámara A

La cámara B la situaremos a la derecha con los valores de la Figura 24.

Figura 24: Configuración cámara B

5.4 Movimiento

Para el movimiento se ha optado por un control clásico: con el joystick izquierdo se apunta en la dirección donde se quiere ir y un parámetro decide la velocidad del movimiento. Dependiendo del lado del campo donde juegue, se deberá tener en cuenta los ejes globales, ya que como las cámaras están orientadas en sentido opuesto, los ejes se invierten. Para más claridad, ver la Figura 25.

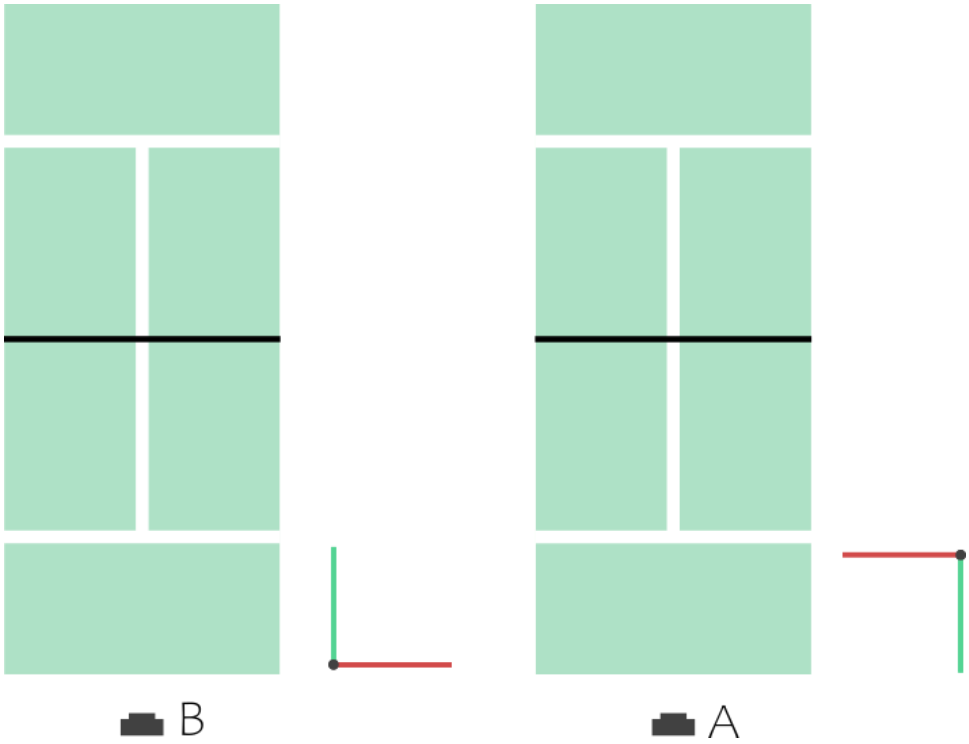

Figura 25: Esquematización del cambio de orientación de las cámaras

Como se puede ver en la Figura 25, para la cámara B ir hacia adelante es subir y para la cámara A es al revés. Lo mismo pasa en el eje derecha-izquierda. Es por esta razón por la que debemos tener en cuenta de qué lado juega el jugador, para invertir sus controles.

Los puntos de aparición de un campo y de otro, se establecen para que el jugador que aparece en el campo enfocado por la cámara A active una variable para invertir el sentido de la dirección del movimiento.

En este bloque se puede observar la variable *mirror* que es la que se establece como cierta o falsa en función del lado de aparición del jugador.

```
if(GetComponent<PlayerDetails>().playerID == 0){
    Mirror();
}
```

En este caso, el primer jugador en entrar al partido se le asignará el numero 0 como identificador y aparecerá en el campo A (enfocado por la cámara A). La función Mirror() establece la variable *mirror* como cierta.

La variable *direction* es un vector de tres dimensiones que, mediante un operador ternario, invertimos o no el movimiento hecho por el usuario. En caso de que la magnitud del vector sea superior a 1, la normalizamos para evitar ir más rápido en cualquier dirección.

```
direction = mirror ? new Vector3(-movement.x, 0, -movement.y) :
                     new Vector3(movement.x, 0, movement.y);

if (direction.magnitude > 0f && !freezed)
{
    // Evitar ir mas rapido en diagonal
    if (direction.magnitude > 1f)
    {
        direction = direction.normalized;
    }

    // Aplicar velocidad
    Vector3 _velocity = direction * speed * Time.deltaTime;

    rb.MovePosition(rb.position + _velocity);
}
```

Un pequeño detalle de este bloque es que se comprueba la variable *freezed*. Esto es porque cuando el jugador empieza a cargar la fuerza del golpe, se queda bloqueado en esa posición, pero sí que sigue rotando, mirando hacia la dirección que apunta la variable *direction* de la siguiente manera:

```
//Rotar personaje
float targetAngle = Mathf.Atan2(direction.x, direction.z) *
                    Mathf.Rad2Deg;

float angle = Mathf.SmoothDampAngle(transform.eulerAngles.y,
                                    targetAngle,
                                    ref turnSmoothVelocity,
                                    turnSmoothTime);

transform.rotation = Quaternion.Euler(0f, angle, 0f);
```

Rotamos el personaje aplicando un suavizado en el movimiento, para que no sea tan brusco e inmediato.

5.5 Golpeo

El golpeo es una mecánica base del juego y fue refinada hasta obtener un resultado simple y eficaz. Primeramente, deberemos saber si la pelota se encuentra al alcance del jugador. Este aspecto tuvo dos versiones. La primera consistía en una esfera que seguía la posición del jugador y éste podía impactar la pelota sólo si ésta se encontraba dentro de la esfera.

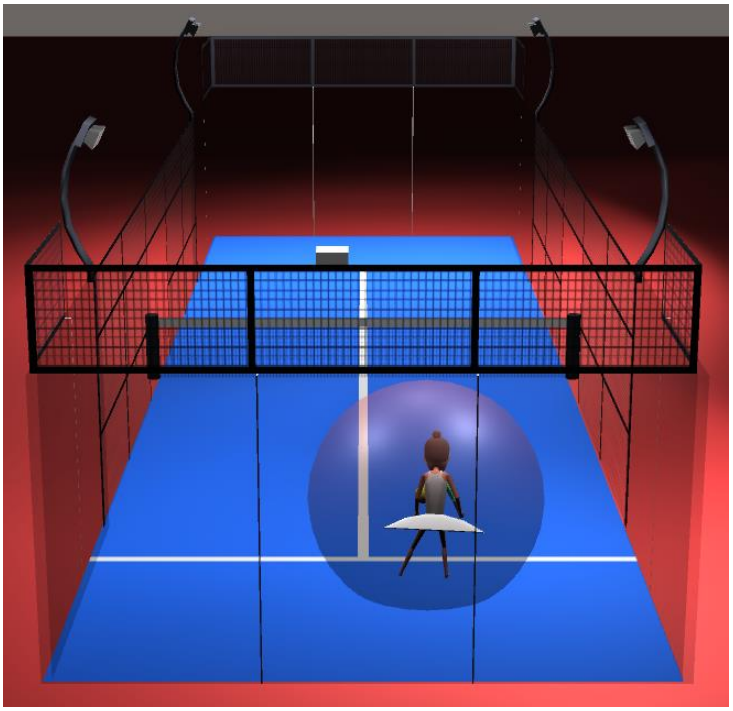

Figura 26: Primera versión del alcance del jugador

Como se puede ver en la Figura 26, no queda claro cuando la pelota está dentro y cuando está fuera. De este problema me di cuenta en la fase de pruebas y me hizo volver hacia atrás y plantear un nuevo diseño que permitiera al jugador saber cuándo la pelota está a su alcance y cuándo no. Para diseñar otra manera de solucionar el problema me inspiré en el juego Rocket League, donde solucionan un problema similar de forma simple (ver Figura 27).

Figura 27: Indicador de la posición de la pelota en Rocket League

Con esta solución en mente diseñé la segunda y definitiva versión del indicador de alcance del jugador. Proyectando verticalmente la posición de la pelota sobre el suelo y dibujando un círculo a los pies del jugador, el jugador sabrá que podrá golpear la pelota si la proyección está dentro del círculo de su personaje. A continuación, vemos explicaciones ilustradas para mayor claridad.

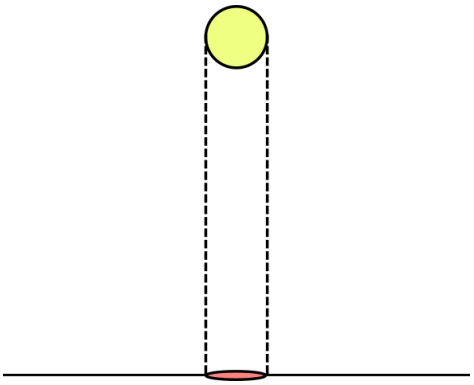

Figura 28: Proyección vertical de la pelota

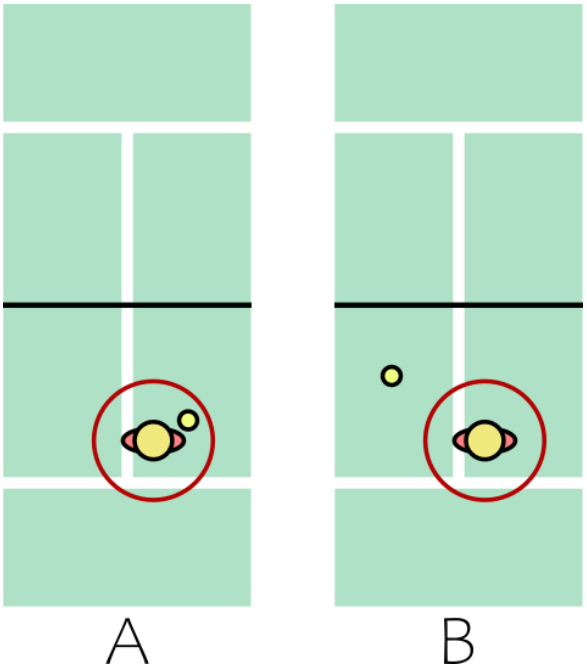

Figura 29: Ejemplos de situaciones del alcance

En la Figura 29, podemos ver que en el caso A el jugador podría golpear la pelota ya que está situada dentro del círculo. En cambio, en el caso B la pelota está situada fuera de su círculo, por tanto, fuera de su alcance. También hay que tener en cuenta la altura de la pelota, pero en esta cuestión no se ha considerado necesario implementar ningún indicador ya que es más intuitivo. En la Figura 30 se puede ver el volumen de alcance del jugador. El radio de alcance se puede configurar desde el Inspector de Unity.

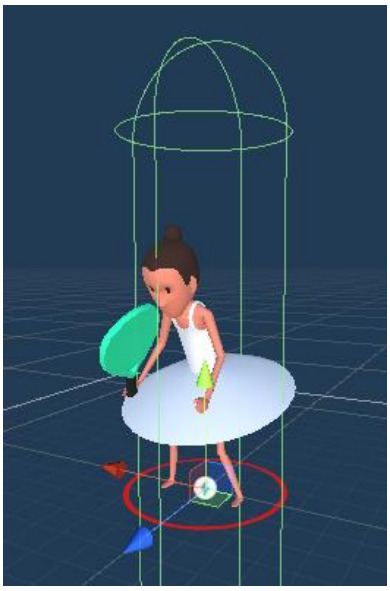

Figura 30: Área de alcance del jugador

Para poder implementar este diseño es necesario tener una imagen que siga la posición horizontal de la pelota, manteniendo la vertical a nivel del suelo. Para ello hemos creado la imagen y la hemos situado como hija de la pelota en el árbol de jerarquía de Unity (Figura 31).

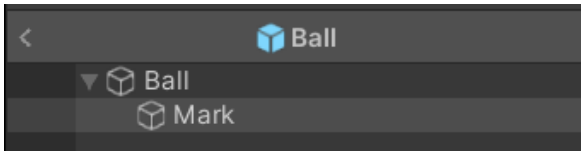

Figura 31: Árbol de jerarquía de la pelota

De esta manera la imagen seguía la posición y rotación de la pelota, pero también la componente vertical de la posición. Para corregir este problema se usó la función Update() de Unity, que se ejecuta muchas veces por segundo, para establecer la posición y la rotación correcta.

```
private void Update() {

    mark.transform.position = new Vector3(this.transform.position.x,
                                          0.1f,
                                         this.transform.position.z);

    mark.transform.rotation = new Quaternion(0.7071f,
                                             0f,
                                             0f,
                                             0.7071f);
}
```

La misma idea se aplica para el círculo que delimita el área de alcance del jugador. Como único cambio tenemos que el personaje sólo rota en el eje vertical. Por lo tanto, al ser un círculo, siempre está bien orientado.

```
area.localScale = new Vector3(range * 2f, range * 2f, range * 2f);
areaTrigger.radius = range;
```

Como se ha mencionado antes y como se puede observar, el radio de alcance es configurable. Una vez resuelto el problema de ubicar la pelota y mostrar el alcance del jugador, queda el resultado de la Figura 32.

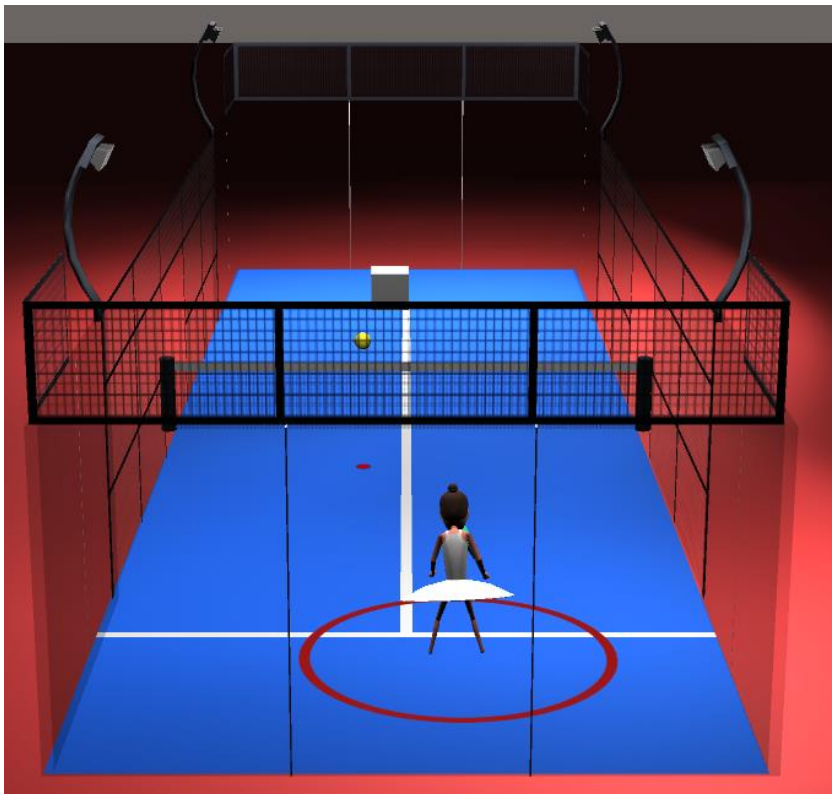

Figura 32: Resultado final del alcance del jugador y ubicación de la pelota

Para efectuar el golpeo necesitamos saber la fuerza, la dirección y la altura.

5.5.1 Fuerza

La fuerza se tendrá que cargar manteniendo el botón, cuanto más tiempo se mantenga apretado, más fuerza se aplicará al golpear, siempre respectando un mínimo y un máximo de fuerza. El incremento de la fuerza no es lineal ni exponencial, es un hibrido entre los dos. Es un crecimiento lineal hasta los dos tercios de la fuerza máxima y, pasados los dos tercios el incremento es cuadrático, tal y como se puede ver en la Figura 33.

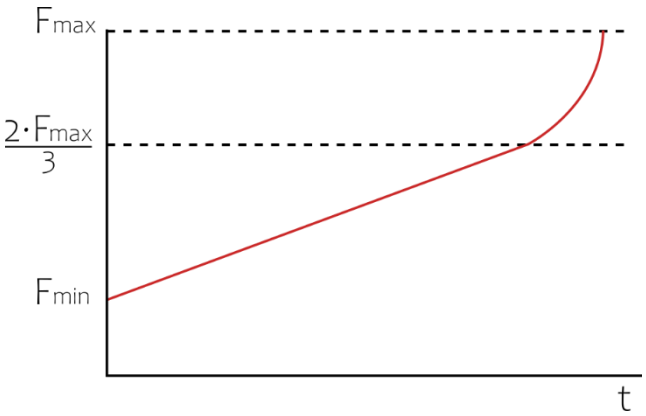

Figura 33: Gráfico representativo del incremento de fuerza

Se ha tomado la decisión de convertir el incremento en exponencial pasados los dos tercios debido a que a veces es necesario aplicar la máxima fuerza posible y no daba tiempo a cargar el golpe al máximo. Otra solución podría haber sido hacer que incrementara más rápido conservando la forma lineal, pero en este caso se perdía control sobre la fuerza que se deseaba aplicar. El jugador visualiza la fuerza que está ejerciendo en un *Slider* o barra de progreso, de esta manera es fácil ver proporcionalmente la fuerza aplicada.

```
private void chargePower(){

    if(power < powerMax * 2/3){          // crecimiento lineal
        power += powerIncrement * Time.deltaTime;
    }
    else if(power < powerMax){           // crecimiento cuadrático
        chargingTime += Time.deltaTime;
        power += Mathf.Pow((powerIncrement * chargingTime), 2);
    }

    powerSlider.value = power;
}
```

En el bloque anterior se muestra la función encargada de incrementar la fuerza y se puede observar cómo, a partir de los dos tercios, se eleva al cuadrado el incremento multiplicado por el tiempo que lleva el jugador cargando el golpe. Finalmente se asigna el valor de la fuerza a la barra de progreso.

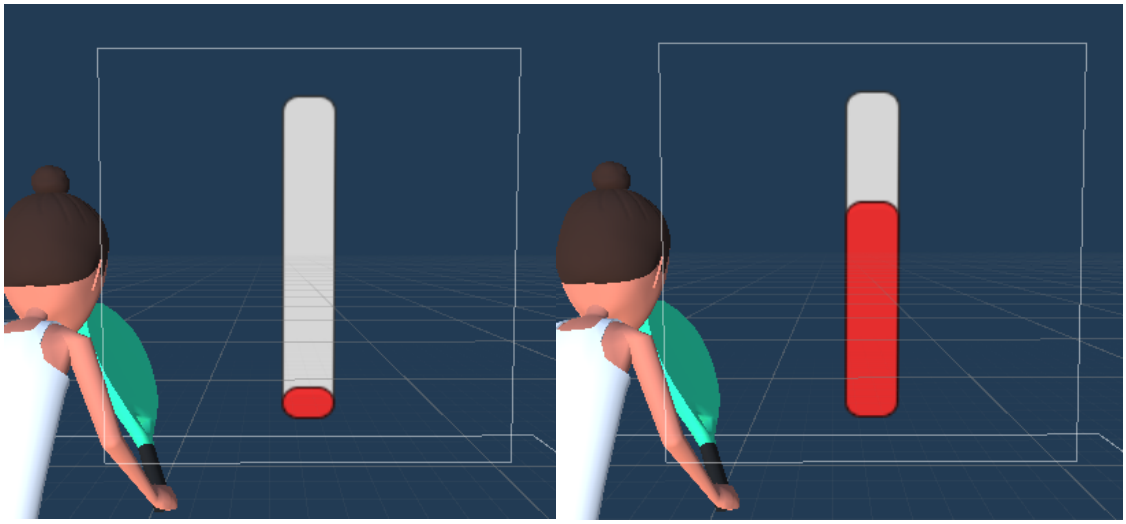

Figura 34: Barra de progreso que marca la fuerza del golpe

Es en el momento que se suelta el botón cuando se efectúa el golpe, de momento sólo disponemos de la fuerza como variable de control.

5.5.2 Dirección

La dirección se toma cuando el botón deja de ser apretado, pero mientras se mantiene apretado el personaje rota obre él mismo para ayudar a apuntar al jugador. Esta dirección se normaliza y se multiplica por la fuerza obtenida anteriormente, y aquí entra en juego el tipo de golpe.

5.5.3 Dirección de altura de tiro

Como se ha mencionado anteriormente (4.1.4 Tipos de golpe), para este proyecto se han elegido los cuatro tipos de golpe básicos del pádel. La única diferencia entre los tipos de golpe es la altura que se les aplica en el momento de golpear.

- **Drive o revés**: Se le aplica una altura razonablemente baja, lo suficiente para pasar por encima de la red.

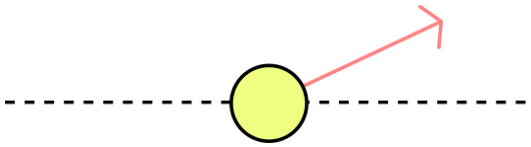

Figura 35: Representación de la altura aplicada en el drive o revés

```
Vector3 height = new Vector3(0f, 500f, 0f);
```

- **Volea**: Como es un golpe que se efectúa cuando se está cerca de la red, no es necesario aplicarle altura.

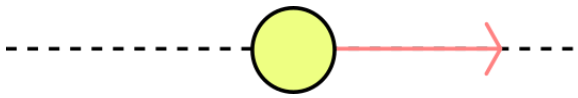

Figura 36: Representación de la altura aplicada en la volea

```
Vector3 height = new Vector3(0f, 0f, 0f);
```

- **Globo**: El objetivo de este golpe es superar por alto a tu rival, de forma que la altura aplicada en este golpe es muy significativa.

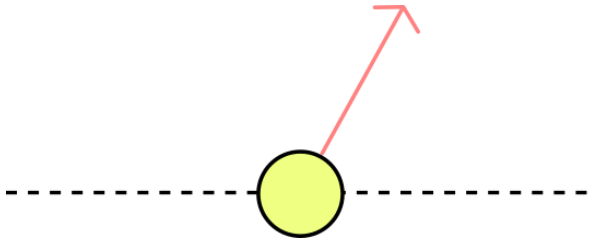

Figura 37: Representación de la altura aplicada en el globo

```
Vector3 height = new Vector3(0f, 1000f, 0f);
```

- **Remate**: Se golpea hacia abajo, por lo tanto, la altura es negativa. En este tipo de golpe se aplica siempre la misma fuerza, pero se varía el ángulo de la dirección del golpeo, variando así su altura.

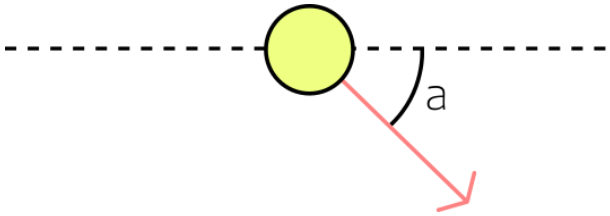

Figura 38: Representación de la altura aplicada en el remate

Este ángulo “a” se decrementa de forma lineal manteniendo el botón de rematar. Comenzando por un ángulo de 0. Esta decisión se ha tomado ya que es común en el pádel que la pelota vaya muy plana cuando se le da antes de tiempo y demasiado hacia abajo cuando se le da muy tarde. Se ha considerado oportuno implementar esta mecánica para darle importancia al temporizar bien el remate.

5.5.4 Ejecutar golpe

Una vez se dispone de la dirección, de la fuerza y de la altura, ejecutamos el golpe de la siguiente manera:

```
Shot(direction.normalized * power + height);
```

Se normaliza el vector dirección para evitar que, en direcciones diagonales, su magnitud sea mayor a 1. El vector normalizado se multiplica por la fuerza y finalmente se suma la altura. La altura no entra en la multiplicación para que siempre se aplique la altura característica del tipo de golpe, de la otra manera se podría lograr hacer un globo usando un golpe de drive.

```
void Shot(Vector3 direction)
{
    Rigidbody ballRb = ball.GetComponent<Rigidbody>();

    ballRb.velocity = Vector3.zero;
    ballRb.AddForce(direction);
}
```

Se accede al componente *Rigidbody* de la pelota, se frena la pelota completamente y se le aplica el vector que tiene la magnitud de la fuerza y dirección. El complemento *Rigidbody* es el encargado de gestionar la física del objeto.

5.6 Saque

En el saque, se ha ubicado al sacador y al restador (ver Figura 39). Antes de sacar, los jugadores se mantendrán estáticos en esa posición hasta que el sacador decida sacar. A partir de ese momento, se jugará el punto con normalidad.

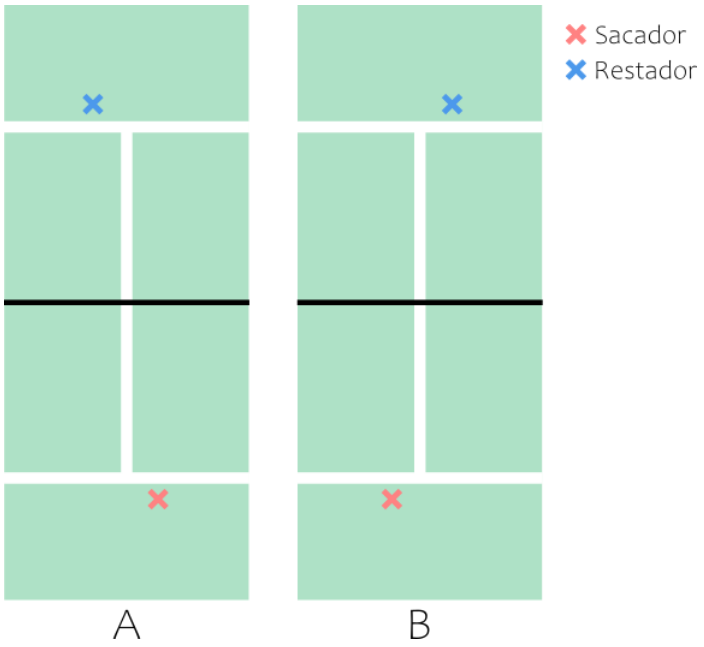

Figura 39: Ubicaciones de saque y resto

En los puntos pares, se usan las ubicaciones del caso A de la Figura 39 y en los puntos impares se san las ubicaciones del caso B.

Como se ha mencionado anteriormente, el sacador dispone de dos intentos de saque. Para detectar si el saque ha sido válido, se comprueba si el primer bote cae dentro del área permitida, en caso contrario, el saque no será válido. Si se trata del primer saque, se podrá intentar una vez más, dos faltas de saque seguidas otorgarán un punto al rival.

El juego internamente está dividido en dos estados, estado de "jugando" y estado de "saque", con el fin de gestionar de manera distinta la lógica del juego.

```
private void OnTriggerEnter(Collider other)
{
    if(firstShot){
        GestionarSaque(Globals.Instance.serveLocation, other);
    }
}
```

El bloque de código anterior es un extracto de la función OnTriggerEnter(), función que se ejecuta cuando el objeto, en este caso la pelota, entra en contacto con un *trigger* o disparador. Estos disparadores están situados por la pista con etiquetas distintivas para poder identificar dónde bota la pelota. En el supuesto que se trate del primer golpe (variable *firstShot*) se gestionará el saque de la siguiente manera:

```
private void GestionarSaque (int servePoint, Collider collision){
    // Obviar colliders no interesantes
    if(collision.gameObject.CompareTag("ResetBounces") ||
       collision.gameObject.CompareTag("Range")){
    return;
    }
```

En el proyecto se usan los disparadores para distintos elementos del juego a parte del área correcta de saque. Por ejemplo, un disparador situado en mitad de la pista, nombrado *ResetBounces*, sirve para reiniciar el contador de botes de la pelota, un disparador del rango del propio jugador sirve para saber si la pelota está al alcance del jugador. Entonces, lo primero que se hace es ignorar los disparadores que no tienen relación con el saque.

```
    // Si no choca con los elementos permitidos
    if (!(collision.gameObject.CompareTag("Area" + (servePoint)))
    {
        if(nIntentosSaque == 2){    // Mal saque
            // Punto para el rival
            Point(lastShotByTeam == 0 ? 1 : 0);
            nIntentosSaque = 1;
        }
        else{
            Point(lastShotByTeam == 0 ? 1 : 0, true);
            nIntentosSaque += 1;
        }
    }
    else{           //Buen saque
        if(collision.gameObject.CompareTag("Area" + (servePoint))){
            firstShot = false;
            nIntentosSaque = 1;
        }

    }
```

La función GestionarSaque() tiene como parámetros el identificador del lado del saque, para comprobar que el primer bote del saque cae sobre el área relacionada con la ubicación el saque, y la propia colisión, que tiene información sobre el choque, por ejemplo la etiqueta del objeto con el que se ha chocado.

En caso de que el primer bote sea en un elemento diferente a los mencionados anteriormente, querrá decir que se trata de un mal saque. Si es el segundo intento de saque, se le da un punto al jugador contrario al que ha tocado la pelota por última vez. En el caso de que sea el primer intento, se sumará uno a los intentos y se le dará otra oportunidad. Si, por lo contrario, el saque es bueno, se indica que no es el primer golpe y se reinicia el contador de intentos.

La función Point() tiene dos parámetros, el primero es el número identificador del equipo que ha hecho el punto y el segundo es un indicador de primer saque (cierto si se trataba del primer saque, falso si no. Por defecto se establece como falso).

```
private void Point(int idTeam, bool primerSaque = false){

    if(primerSaque){

        nBounces = 0;

        Globals.Instance.gameState = Globals.SERVE_STATE;
    }
    else{

        nBounces = 0;

        Globals.Instance.gameState = Globals.SERVE_STATE;

        score.PointTeam(idTeam);
        Globals.Instance.addPoint();
    }

}
```

La diferencia entre el primer saque y el segundo es que en el primer saque no se suma el punto en el marcador, pero en los dos casos se reinicia el contador de botes a cero y se vuelve a establecer el estado del juego a estado de saque.

5.6.1 Cambio de turno de saque

Al final de cada juego se cambia el sacador y la solución aplicada al código ha sido convertir las ubicaciones de resto en ubicaciones de saque y viceversa. Teniendo en cuenta el número identificador del jugador, se situará al jugador en una posición o en otra.

5.7 Comportamiento de la Inteligencia Artificial (IA)

La inteligencia artificial, de ahora en adelante nos referiremos a ella como IA, es la base del modo de juego individual en cualquier juego deportivo. Esta IA debe recrear el comportamiento de un deportista lo más parecido posible a la realidad para que el jugador tenga una buena experiencia de juego. Para este proyecto es necesario programar el movimiento y posicionamiento de la IA, así como el golpeo.

5.7.1 Movimiento de la IA

Un buen posicionamiento en pista evita se hagan puntos fáciles, poniéndole así más complicada la tarea al jugador. De esta manera el jugador tendrá un mayor reto y el juego será más entretenido.

Posicionarse bien en el pádel significa tapar la mayor área de la pista. Por lo tanto, la IA tendrá que situarse centrada en la pista respecto a la posición de la pelota. De tal manera que, si la pelota se encuentra del lado derecho de la pista, el jugador deberá estar más hacia la derecha respecto al centro de la pista. A este movimiento se le llamará basculación (Figura 40) y se aplicará cuando la pelota esté situada en el campo del contrario. En el caso de que la pelota esté en el campo de la IA, simplemente la IA irá hacia la pelota. La basculación consiste en moverse de un lado a otro girado sobre un eje. Este eje lo llamaremos pivote y estará situado centrado al fondo de la pista, en el campo de la propia IA. El jugador deberá mantenerse en todo momento en la línea que forma el pivote con la pelota. Tener al jugador en la línea no es suficiente, ya que se puede estar en infinitas posiciones a lo largo de una línea, por lo que es necesario calcular el punto exacto donde tiene que colocarse nuestra IA. El punto exacto se calcula se la siguiente manera:

```
void MovementManagement(){
  // Si la pelota está en mi campo y aún no la he tocado
  if(ball.transform.position.z > 0f && BallManager.lastShotByTeam != -1)
  {
      Vector3 pos = new Vector3(ball.transform.position.x,
                                2f,
                                ball.transform.position.z);

      // Mover hacia la pelota
      transform.position = Vector3.MoveTowards(transform.position,
                                               pos,
                                               speed * Time.deltaTime);
  }

  else{
        // Obtenemos la posición de la pelota y del pivote
     Vector3 ballPos = new Vector3(ball.transform.position.x,
                                   0f,
                                   ball.transform.position.z);

     Vector3 pivotPos = new Vector3(pivot.position.x,
                                    0f,
                                    pivot.position.z);

     // Calculamos punto medio (factor opcional)
     Vector3 midpoint = (ballPos + pivotPos * factor)/2;

     // Evitar salir del campo atravesando el cristal de fondo
     if(transform.position.z < 45 || midpoint.z < 50){

       // Mover hacia el punto medio
       transform.position = Vector3.MoveTowards(transform.position,
                                                midpoint,
                                                speed * Time.deltaTime);
     }
  }
```

En la última línea de código, se efectua el movimiento de la IA hacia el punto medio calculado previamente, usando la función MoveTowards. Esta función se encarga de calcular en que posición, entre la posición actual y la posición calculada, debe estar si tiene una velocidad en un momento exacto de tiempo La variable *factor* sirve para alejar o acercar la posición del pivote respecto al centro de la pista, de esta manera podemos calibrar si la IA juega más cerca o más lejos de la red (Figura 41).

```
if(this.transform.position.z <= netMark.position.z){
        inNet = true;
    }
    else{
        inNet = false;
    }
```

Finalmente, tenemos un condicional que indica si la IA está cerca de la red o no. Esta variable (*inNet*) modificará los tipos de tiros que haga nuestra inteligencia artificial.

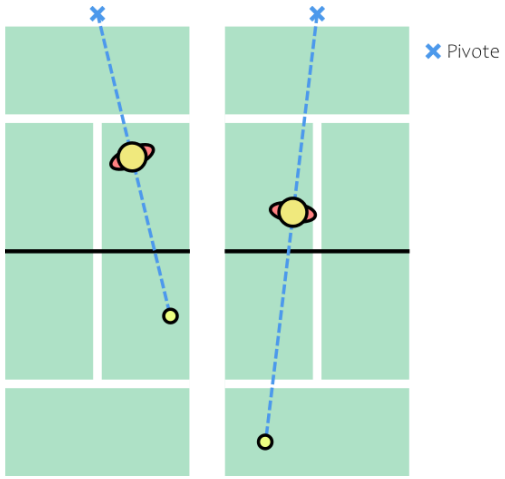

Figura 40: Ejemplos de basculación

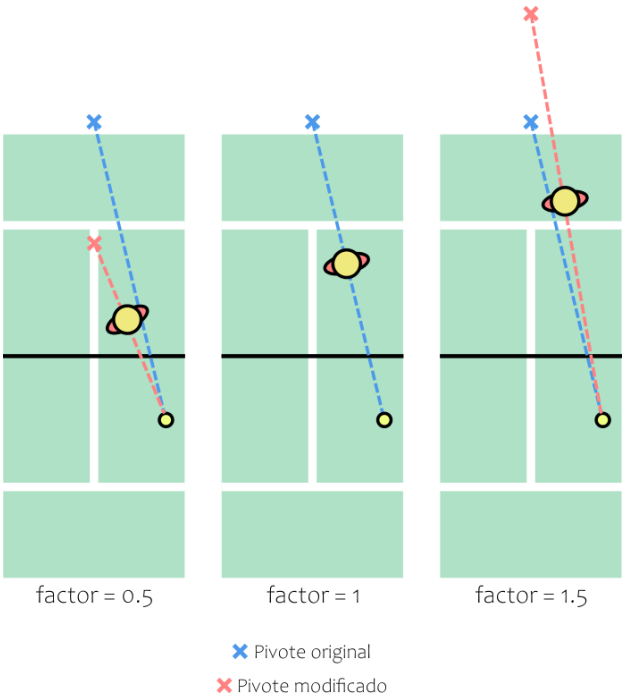

Figura 41: Basculación aplicando un factor

5.7.2 Golpeo de la IA

Para efectuar el golpeo hace falta saber los mismos tres elementos que en el golpeo realizado por un jugador, y estos elementos son: la dirección, la fuerza y el tipo de golpe.

5.7.2.1 Dirección del golpe

Durante el desarrollo del juego se fue evolucionando la metodología de elegir direcciones, dando pie a diferentes dificultades de juego.

5.7.2.1.1 Dificultad fácil

Todos los golpes se dirigen hacia el mismo punto, establecido anteriormente.

```
//FACIL
direction = (target[0].position - ball.transform.position).normalized;
```

5.7.2.1.2 Dificultad media

Los golpes van dirigidos a tres ubicaciones establecidas, eligiendo una aleatoriamente.

```
//MEDIO
int rand = Random.Range(0,3);
direction = (target[rand].position - ball.transform.position).normalized;
```

5.7.2.1.3 Dificultad difícil

La mayor parte de los golpes van centrados, ya que ajustar la pelota cerca de la pared es arriesgado. Esta es la filosofía se ha querido implementar en el estilo de juego de la IA. Para llevar a cabo esta filosofía, se ha ideado un sistema que genera la dirección influenciada de manera indirecta por la posición de la IA. Se establecen dos puntos, delimitando un rectángulo. Este rectángulo será el área donde apunte la IA, coincidiendo con el campo del rival. De manera aleatoria se elegirá una posición dentro del rectángulo y se calculará el vector que apunta desde la posición de la IA hacia el punto generado. Esa será la dirección del golpe.

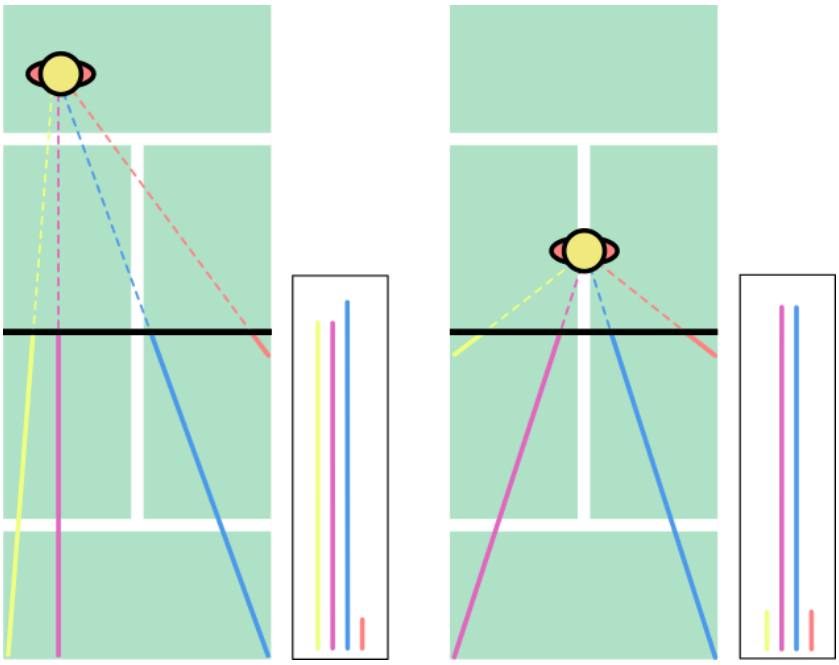

Figura 42: Esquematización de las frecuencias de direcciones

Con este modelo de generación de direcciones, es más probable que se generen posiciones en el medio. La Figura 42 lo ejemplifica, mostrando la longitud de las líneas para ver que es más probable que la dirección generada sea centrada u orientada hacia las diagonales.

```
//DIFICIL
int x = Random.Range((int)minTarget.position.x,
                     (int)maxTarget.position.x);

int z = Random.Range((int)minTarget.position.z,
                     (int)maxTarget.position.z);

Vector3 randomPoint = new Vector3(x, 0f, z);
direction = (randomPoint - this.transform.position).normalized;
```

5.7.2.2 Fuerza del golpe

Para la fuerza se establece un valor base y se la aplica una variabilidad del 25%., El valor base es un valor que, a través de pruebas, se ha llegado a la conclusión de que era el valor óptimo para jugar.

```
float randomFactor = Random.Range(0.75f, 1.25f);
hitPower = hitPower * randomFactor;
```

La variabilidad hace que los golpes sean más orgánicos y realistas. También hacen que la IA pueda fallar golpes y no sea imposible ganar contra ella.

5.7.2.3 Tipos de golpe

Este sistema se ha ideado pensando en una ruleta de la suerte, donde el premio mejor valorado aparece una sola vez. Para elegir el tipo de golpe se dispone de dos listas con nombres de golpes. La primera lista se usa cuando la IA se encuentra en el fondo de la pista y la segunda cuando se encuentra cerca de la red. En la lista aparecen los nombres de los golpes de manera repetida, que será su frecuencia. Cuantas más veces aparezca en la lista el nombre de un golpe, más probable será que sea elegido. Al poder elegir cuantas veces se repiten los nombres, es configurable la tendencia y por lo tanto se puede personalizar el comportamiento de la IA. El hecho de que haya dos listas es porque se juega diferente si se está cerca de la red o si se está lejos. Estando cerca es más común hacer voleas y menos globos, en el caso contrario pasa al revés.

```
shots = {"LOB", "LOB", "DRIVE", "DRIVE", "DRIVE"};
netShots = {"LOB", "DRIVE", "VOLLEY", "DRIVE", "VOLLEY"};
```

Una vez tenemos las dos listas, se sortea aleatoriamente un número entre 0 y la longitud de la lista (este último no se incluye) y el tipo de golpeo será el que esté situado en el índice del número que se ha sorteado.

```
int index = Random.Range(0, shots.Length);
string shotType;
if(inNet){
    shotType = netShots[index];
}
else{
    shotType = shots[index];
}
```

```
switch (shotType){
    case "LOB":
        height = new Vector3(0f, 1000f, 0f);
        break;

    case "DRIVE":
        height = new Vector3(0f, 500f, 0f);
        break;

    case "VOLLEY":
        height = new Vector3(0f, 0f, 0f);
        break;
}
```

Igual que en el golpeo realizado por un humano, se aplica la altura correspondiente al tipo de golpeo. Finalmente se efectúa el golpe.

```
Shot(direction * hitPower  + height);
```

Apliqué la solución de recrear una ruleta de la fortuna por comodidad, pero existen funciones que permites parametrizar las probabilidades de forma más sencilla, un ejemplo es la *Función de distribución acumulativa de una variable.*

5.8 Reglamento

Para este proyecto se han implementado las reglas básicas del pádel en lo que se refiere al transcurso del punto. Para determinar si un golpe es válido o no, se mirará donde rebota y se comprobaran una serie de valores.

5.8.1 Rebote en las paredes

Rebotar en las paredes incluye los cristales de ambos lados y las rejas. En el caso de los cristales distinguiremos entre el cristal A y cristal B, siendo los cristales de un lado y de otro del campo.

- **Cristal A**: Si rebota en el cristal A, se comprueba si la pelota aún no había botado en el suelo y si el último jugador en tocar la pelota no ha sido el jugador del propio campo, ya que en el pádel se permite hacer rebotar la pelota en tu propio cristal para que, con el rebote la pelota, pase de al otro campo. En el caso de que se cumplan las dos condiciones, será punto del jugador situado en el campo A.

```
if (collision.gameObject.CompareTag("WallA"))
{
    if(nBounces == 0 && lastShotByTeam != 0)
    {
        Point(0);
    }
}
```

- **Cristal B**: Pasa igual que en cristal A, pero con los participantes intercambiados.

```
if (collision.gameObject.CompareTag("WallB"))
{
    if(nBounces == 0 && lastShotByTeam != 1)
    {
        Point(1);
    }
}
```

- **Reja**: En este caso se mirará solo si la pelota ya ha rebotado en el suelo, ya que no puede tocar directamente en la reja, aunque se trate de la reja situada en el campo del jugador que golpea. En caso de tocar la reja directamente, el punto será del rival al último en tocar la pelota.

```
if (collision.gameObject.CompareTag("Reja"))
{
    if(nBounces == 0){
        Point(lastShotByTeam == 0 ? 1 : 0);
    }
}
```

Un pequeño detalle que se ha implementado es el comportamiento de la pelota en el momento de interactuar con la reja, ya que su rebote es impredecible. En el momento inmediatamente después de rebotar, se multiplica la dirección del rebote por un vector de tres dimensiones cuyos valores están entre 0 y 1, modificando así su dirección al rebotar.

```
private void OnCollisionExit(Collision other) {
    if (other.gameObject.CompareTag("Reja"))
    {
        var x = Random.Range(0f, 1f);
        var y = Random.Range(0f, 1f);
        var z = Random.Range(0f, 1f);

        Vector3 randomDirection = new Vector3(x,y,z);

        rb.velocity = Vector3.Scale(rb.velocity ,
randomDirection);
    }
}
```

5.8.2 Rebotes en el suelo

Diferenciaremos entre tres tipos de suelos: suelo A (correspondiente al campo A), suelo B (correspondiente al campo B) y suelo de fuera de la pista. Siempre que la pelota bote en el suelo, indiferentemente del tipo de suelo, se sumará un bote a la variable que cuenta los botes en el suelo. Siempre que se acabe el punto, se reiniciará a 0 el contador de rebotes en el suelo.

- **Suelo A y B**: Se comprueba que sea el segundo bote o, que sea el primer bote y el último jugador en tocar la pelota sea el del propio campo. En ese caso el punto será del jugador situado del otro campo

```
if (other.gameObject.CompareTag("GroundA"))
{
    nBounces += 1;
    if (nBounces == 2 || nBounces == 1 && lastShotByTeam == 0)
    {
        // Punto Equipo 1
        Point(1);
    }

}

if (other.gameObject.CompareTag("GroundB"))
{
    nBounces += 1;
    if (nBounces == 2 || nBounces == 1 && lastShotByTeam == 1)
    {
        // Punto Equipo 0
        Point(0);
    }
}
```

- **Suelo de fuera**: En caso de que la pelota bote fuera, se consideran dos posibilidades: la primera es que sea el primer bote y entonces querrá decir que ha botado directamente fuera, en cuyo caso será punto del jugador rival al último en golpear. La segunda posibilidad es que sea el segundo bote el que hace fuera. Esto querrá decir que el primer bote lo ha hecho dentro y la pelota ha salido del campo debido a un remate. En este caso el punto será del último en haber golpeado.

```
if (other.gameObject.CompareTag("Out"))
{
    nBounces += 1;

    if(nBounces == 1){
        // Punto del jugador rival
        Point(lastShotByTeam== 0 ? 1 : 0);

    }
    else if (nBounces == 2)
    {
        Point(lastShotByTeam);
    }
}
```

5.8.3 Otras reglas

Como es obvio, un mismo jugador no puede tocar la pelota dos veces seguidas y esto se consigue comparando el identificador del último jugador en tocar la pelota con el jugador que la quiere tocar. En caso de que sean diferentes, se podrá realizar el golpe.

```
if(BallManager.lastShotByTeam !=
gameObject.GetComponent<PlayerDetails>().playerID){

    PerformShot();
}
```

También hay que cambiar de lado de saque en cada punto, para ello se mirará si es un punto par o impar y de ello dependerá el lado del saque.

```
if(Globals.Instance.getNPoints() % 2 == 0){
    sP = servePoints[0].position;
    Globals.Instance.serveLocation = 0;
}
else{
    sP = servePoints[1].position;
    Globals.Instance.serveLocation = 1;
}
```

La variable *sP* guarda la posición de saque correspondiente.

A parte del cambio de lado, también hay que cambiar el turno de saque. El turno de saque se cambia al finalizar un juego, por lo que, igual que se hace para cambiar de lado, en este caso también se comprobara si es un juego par o impar.

```
if(Globals.Instance.nGames % 2 == 0){
    playerToServe = players[0];
}
else{
    playerToServe = players[1];
}
```

5.9 Marcador

En todo juego deportivo es necesario tener un marcador con el resultado. El resultado siempre va acompañado de un nombre o unas siglas que identifican al equipo o jugador. Teniendo esto en cuenta, en este proyecto se les pedirá el nombre a los jugadores al inicio del partido (Figura 44) para posteriormente poder identificar de quien es cada resultado. En el marcador se mostrarán las primeras 3 letras convertidas a mayúsculas a la izquierda del resultado numérico. El resultado numérico constará de juegos y puntos. Toda esta información estará encima de la plantilla de la Figura 43.

Figura 43: Plantilla del marcador

Figura 44: Escena provisional de elección de nombre

Tal y como se ha explicado en el apartado de conceptos previos, los puntos se cuentan de esta manera: 0, 15, 30, 40, Adv. Estos valores se guardan en una lista y se utilizaran posteriormente, ya que el conteo de puntos internamente se hará con valores enteros en el rango de 0 a 4.

Cuando se suma un punto a un jugador, hay que tener en cuenta dos cosas: si está en situación de jugar con las ventajas (ventaja de dos puntos) o si el jugador ha ganado ese juego. Si no ocurre ninguna de las dos, se otorgará punto al jugador sin más complicaciones. Primeramente, se

comprobará si está en situación de jugar con ventajas. Esto quiere decir que los dos jugadores han conseguido hacer tres puntos o más. En este supuesto se pueden dar tres posibilidades:

Aclaración: Se expresarán las situaciones con el formato PJ1/PJ2, siendo PJ1 los puntos del jugador 1 y PJ2 los puntos del jugador 2. Se supondrá que el jugador que hace el punto siempre será el jugador 1. La situación se refiere al marcador en el momento justo antes de hacer el punto. En el código se suma punto siempre y después se hacen las comprobaciones.

- **Situación de 40/40**: El jugador 1 sumará un punto y el marcador tendrá como resultado Adv/40.
- **Situación Adv/40**: El jugador 1 gana el juego, se suma 1 a los juegos ganados del jugador 1 y se reestablece el marcador de puntos a 0/0.

```
if(points[idTeam] > 4 && points[idTeamRival] == 3){

    restartPoints();

    games[idTeam] += 1;
    Globals.Instance.nGames += 1;
    textGames[idTeam].text = games[idTeam].ToString();
}
```

- **Situación 40/Adv**: El jugador 2 pierde la ventaja, pero el jugador 1 no suma ningún punto. Resultado finalmente será de 40/40.

```
else if(points[idTeam] == 4 && points[idTeamRival] == 4){
    points[idTeam] -= 1;
    points[idTeamRival] -= 1;
}
```

Tal y como se ha dicho antes, el jugador 1 no suma punto, es por eso por lo que se le resta 1 ya que se le había sumado previamente. Al jugador 2 se le quita la ventaja también.

Figura 45: Marcador final

5.10 Pelota

En otros juegos deportivos, la pelota es fiel a la realidad en el sentido de respetar el tamaño, material, peso, etc. En este proyecto, al tratarse de un juego con mecánicas arcade, se ha optado por incrementar el tamaño, consiguiendo así una estética más desenfadada. También se ha reducido la gravedad, haciendo que la caída de la pelota sea ligeramente más lenta para dejar más tiempo de reacción a los jugadores. En cuanto al material, existía la posibilidad de modelar la pelota y aplicarle texturas, pero a la hora de la verdad, no hacía falta tanto detalle ya que la pelota siempre se ve desde lejos y apenas se apreciarían los detalles. Es por esa razón que se ha usado una esfera del propio motor de juegos Unity aplicándole un material físico con un índice de rebote de 0.55, lo que quiere decir que la pelota en cada rebote perderá un 45% de la energía, haciendo que la pelota rebote un 45% menos. En la Figura 46 se puede entender con más claridad.

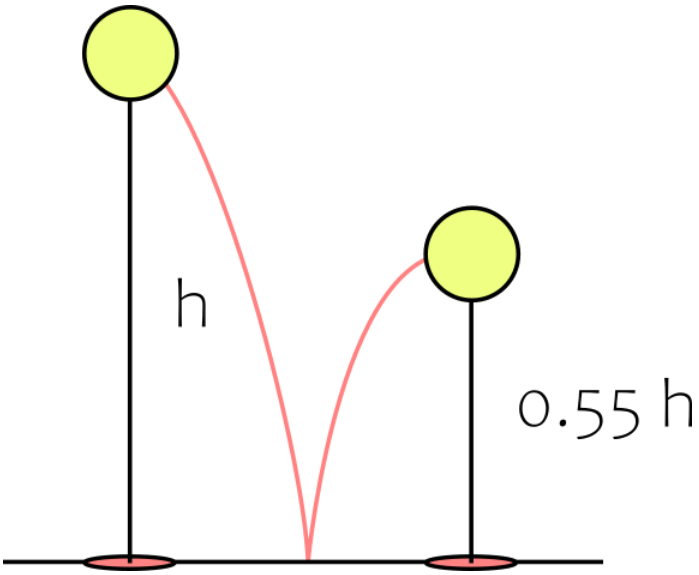

Figura 46: Explicación índice de rebote

Aclaración: Unity diferencia entre material y material físico. Un material únicamente tiene efectos en el apartado visual donde se puede elegir textura, configurar el comportamiento de la luz, etc. Un material físico afecta al apartado de la física, permite configurar índice de rebote, fricción, etc.

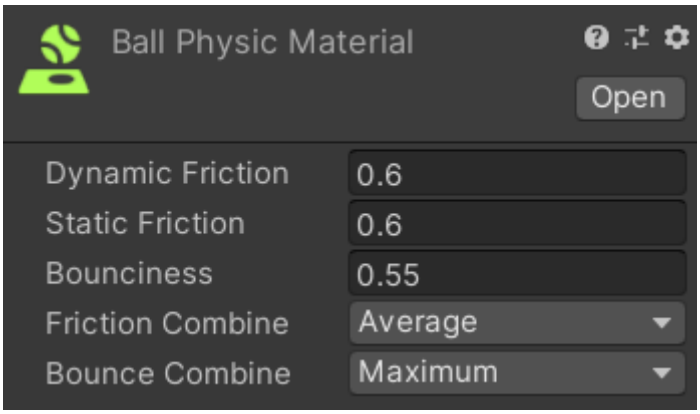

Figura 47: Configuración del material físico de la pelota

5.11 Pista

En este apartado se hablará sobre el modelado de la pista. Para empezar a modelar, hay que tener en cuenta las medidas de la pista reglamentaria. La pista común es la pista de dobles, es por eso por lo que hay que prestar atención a las medidas que se encuentran, en mi caso, en internet. A continuación, en las Figuras 48 y 49 se detallan cuáles son estas medidas.

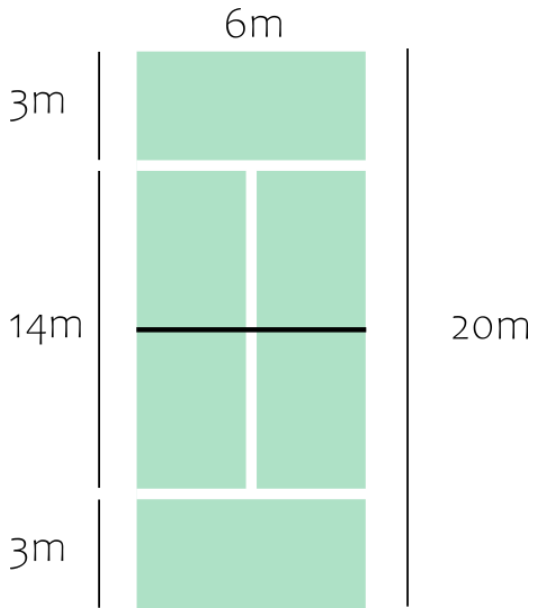

Figura 48: Medidas desde vista aérea

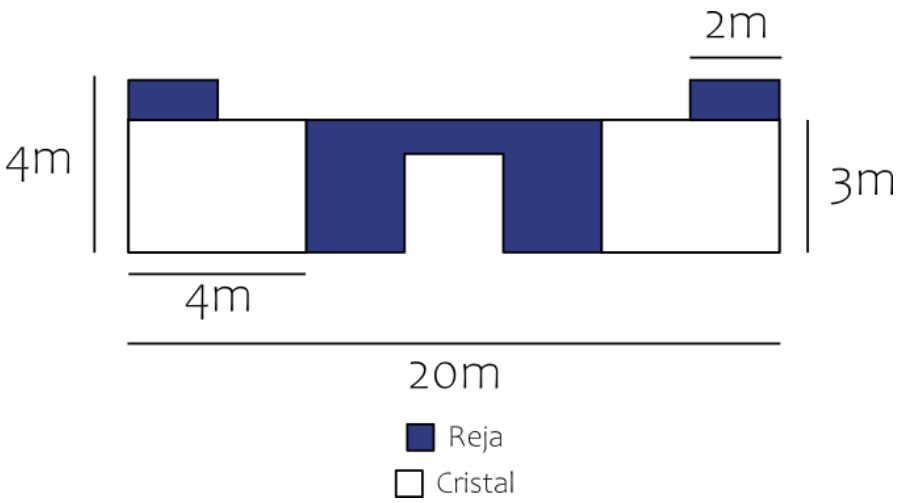

Figura 49: Medidas desde vista lateral

Una vez se han obtenido las medidas, pasamos al programa de modelado Blender. Al ser un programa más artístico que técnico, la herramienta de medir distancias no funciona como se desearía ya que se ocultan las medidas cuando se cambia de herramienta. Es por esa razón que se empezó con la base con las medidas correctas (Figura 50) y luego poco a poco se fueron haciendo las demás partes con las medidas proporcionales a la base.

5.11.1 Fotos del proceso de modelado

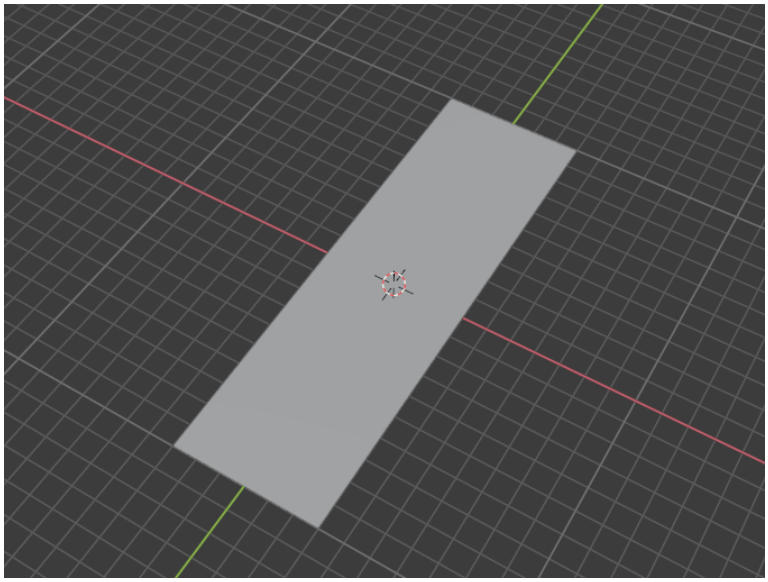

Figura 50: Base de la pista de pádel

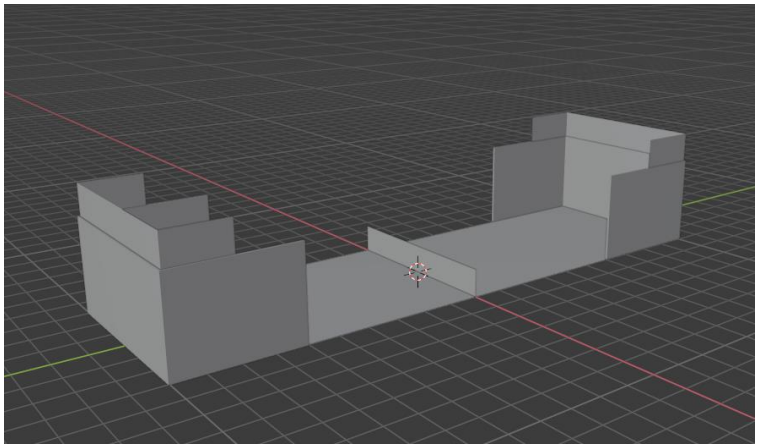

Figura 51: Cristales, rejas y red de la pista

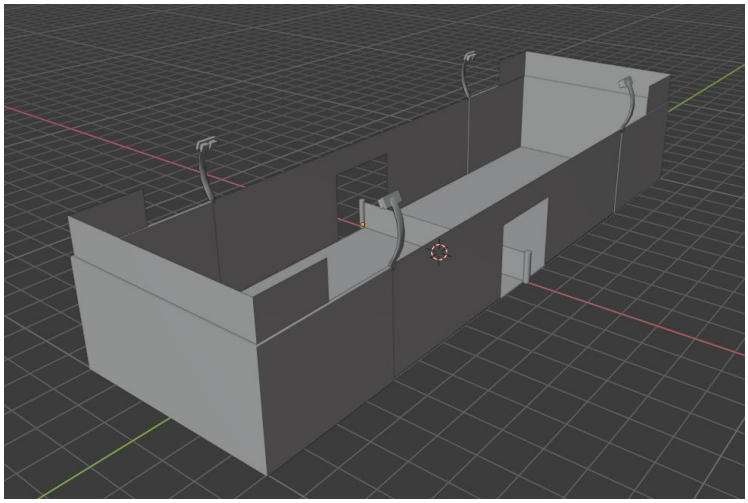

Figura 52: Modelado final de la pista

5.11.2 Materiales de la pista

5.11.2.1 Suelo

Tanto en el suelo como en las líneas, se ha utilizado un material plano sin textura, de color similar al suelo del WPT. He respetado el color del suelo ya que ofrece un buen contraste con la pelota y facilita su visión.

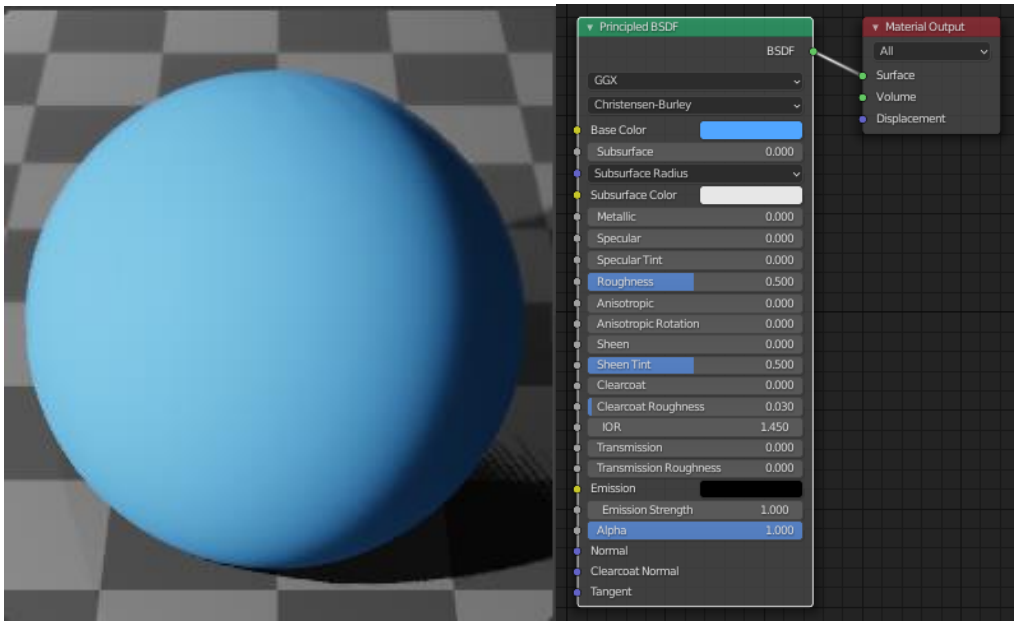

Figura 53: Material del suelo

El material de las líneas está hecho de la misma manera con la diferencia que el *Base Color* es blanco.

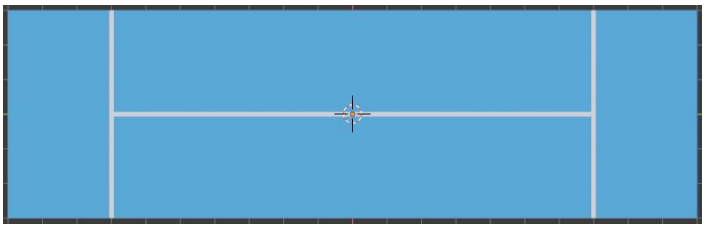

Figura 54: Resultado final del suelo

5.11.2.2 Cristales

Para los cristales se ha usado un color plano de tonalidad casi blanca y con mucha transparencia. Para simular las juntas entre cristales y dar mayor realismo, se ha usado el mismo material que para hacer las líneas del suelo.

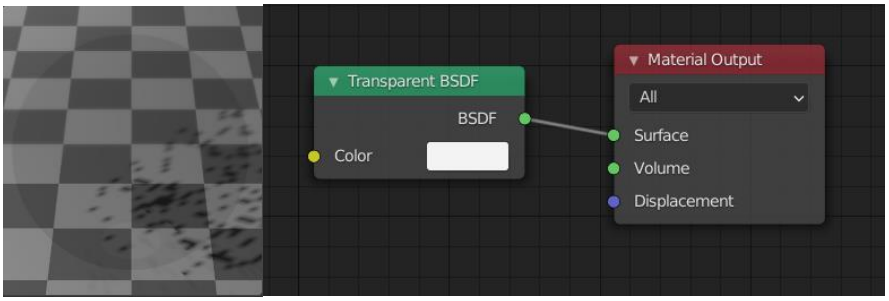

Figura 55: Material del cristal

5.11.2.3 Reja

En el caso de las rejas, se podría haber modelado el entramado metálico, pero eso supondría una cantidad de vértices innecesarios que empeorarían el rendimiento del juego. Es por esa razón que se ha usado un material con mapa de opacidad. El material en cuestión fue descargado de internet en *ambientCG*, que se trata de un catálogo de materiales gratuitos y fáciles de aplicar.

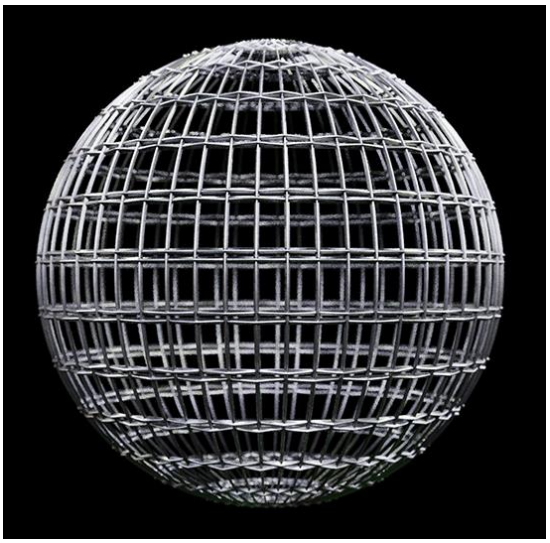

Figura 56: Previsualización del material de la reja

El mapa de opacidad actúa a modo de máscara que multiplica el valor en una posición concreta por el valor del canal de transparencia del mapa de color del material. De este modo, las partes negras del mapa de opacidad serán transparentes. En la Figura 57 se hace una recreación del proceso de multiplicación.

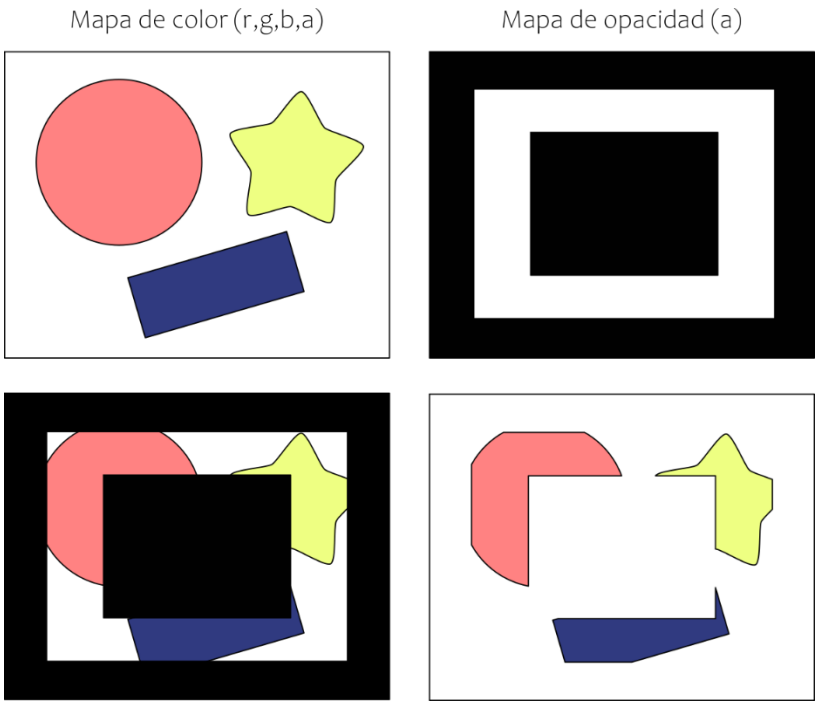

Figura 57: Recreación de la multiplicación del mapa de opacidad

En el caso del material de la reja, sus mapas de color y opacidad son los siguientes:

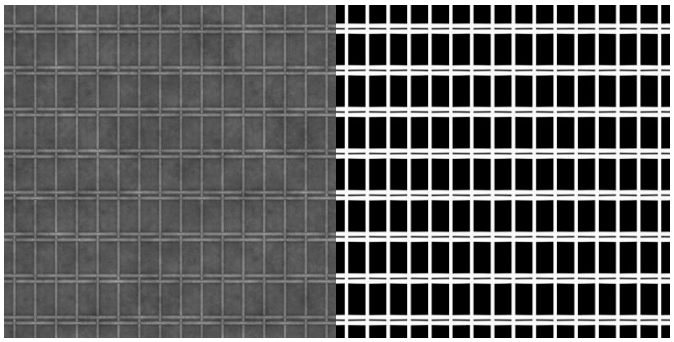

Figura 58: Mapas de color y opacidad del material de la reja

Para cambiar la tonalidad de la reja, se añade un nodo *Mix* que tiene dos imágenes de entrada y una de salida. Este nodo permite elegir la cantidad de color que deja pasar de una imagen y de otra. En el caso de la reja, no interesa el color grisáceo de la textura y es por eso por lo que solo dejo pasar el color azulado (Figura 59).

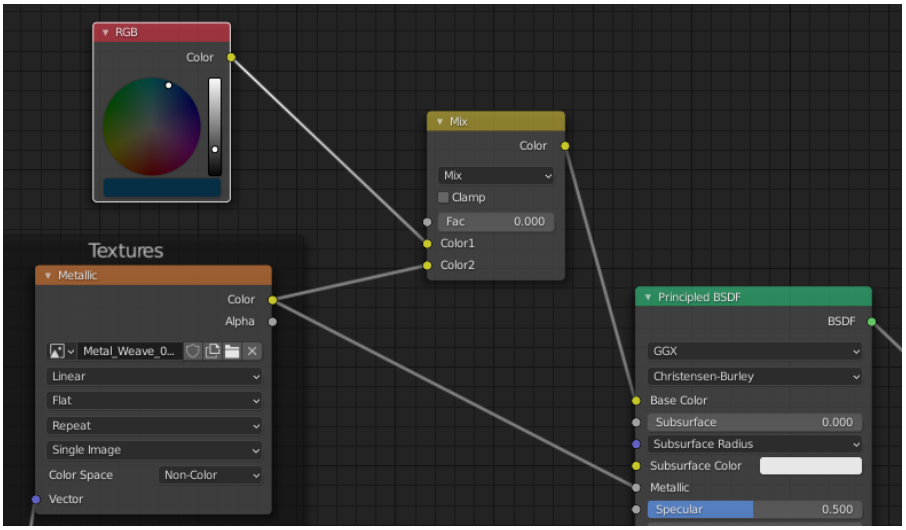

Figura 59: Funcionamiento del nodo *Mix*

Al aplicarse el material al plano, queda el resultado de la Figura 60. Para recrear con mayor fidelidad la reja, se han añadido partes estructurales hechas con un material plano, del mismo color que la reja.

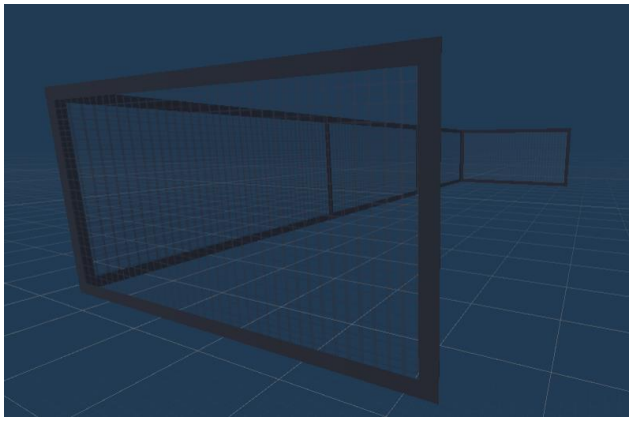

Figura 60: Resultado final de la reja

5.11.2.4 Red

Para la red se reutiliza el material usado en la reja junto al material de las líneas del suelo para la cinta. Los postes situados a cada lado de la red utilizan el material de metal, igual que los detalles estructurales de la reja.

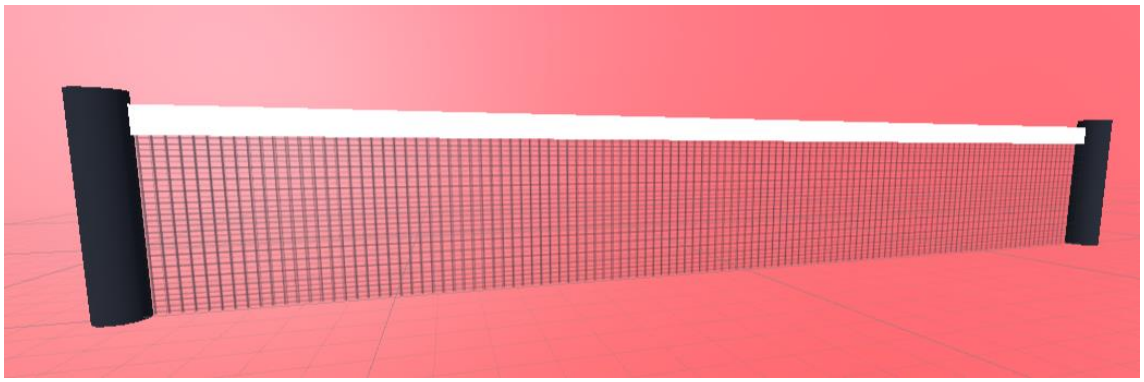

Figura 61: Resultado final de la red

5.11.2.4 Accesorios

Para adornar más el entorno y hacerlo más creíble, se han modelado unos focos puramente decorativos, puesto que no emiten luz.

Figura 62: Focos decorativos

5.11.3 Resultado final

Con todo lo explicado anteriormente, ha quedado el resultado de la Figura 63 y 64.

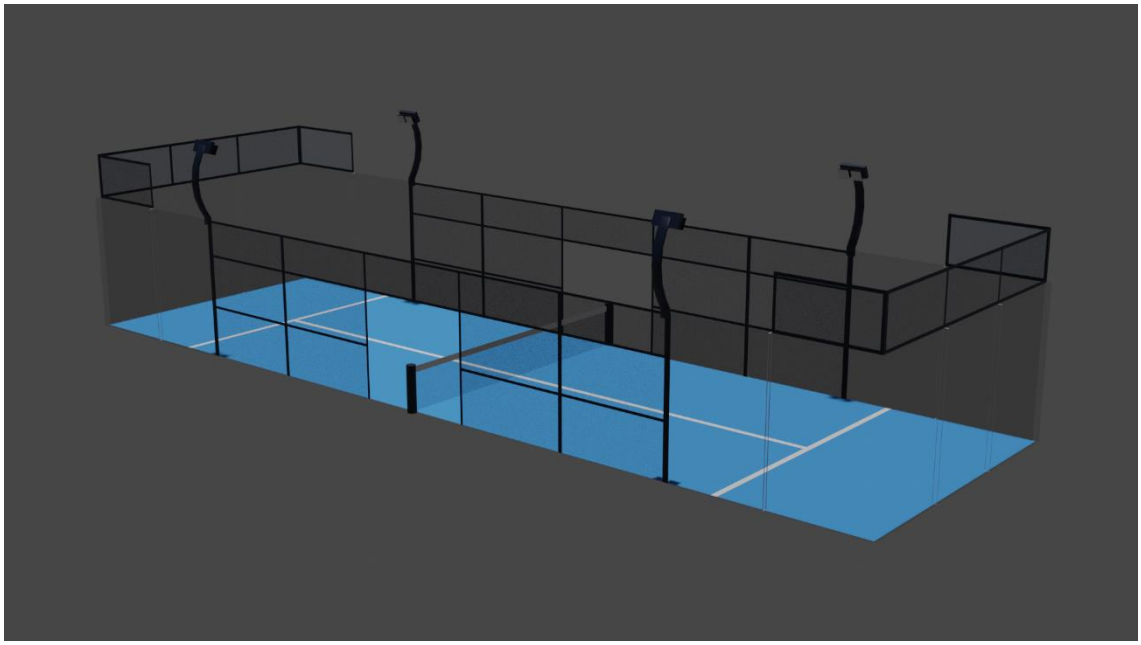

Figura 63: Resultado final de la pista

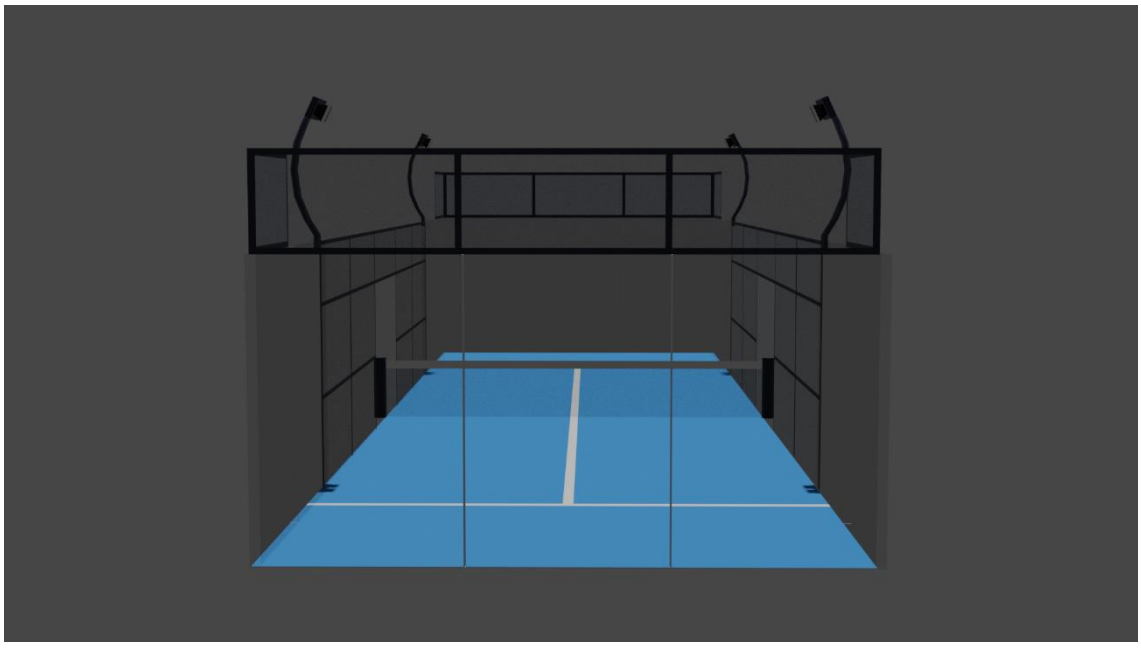

Figura 64: Resultado final de la pista

5.12 Personajes

Para este proyecto se planeó diseñar dos personajes distintos para tener variedad de habilidades y estilos de juego. Inspirado en Mario Tennis Aces, se ha decidió hacer dos personajes que mantengan la misma estética pero que no tengan ninguna relación entre ellos, en cuanto a narrativa. En lo que se refiere a la estética se ha seguido la inspiración del Mario Tennis Aces y se ha optado por una estética *cartoon* y minimalista, exagerando el tamaño de la cabeza y reduciendo el ancho de las extremidades.

Para dar variedad se ha optado por hacer un personaje más veloz pero menos fuerte y otro al revés. El personaje veloz se trata de una bailarina y el personaje fuerte se trata de un leñador. Estos personajes se han diseñado y modelado con dos metodologías diferentes.

5.12.1 Bailarina

Con la bailarina se ha querido remarcar su ligereza desde el principio de su diseño, planteando incluso hacer que caminara de puntillas, que en el mundo del ballet se llama hacer puntas. Finalmente se descartó esta opción ya que las animaciones de *Mixamo*, de las que hablaremos más adelante, al ser genéricas, no había ninguna que corriera de puntillas. La metodología de desarrollo de la bailarina ha sido (en orden cronológico): búsqueda de referencias, dibujo a mano del diseño, digitalización del dibujo, modelado, esculpido, texturizado, *rigging* y animado.

5.12.1.1 Búsqueda de referencias

La búsqueda de referencias se realizó por internet, buscando fotografías de bailarinas reales, de bailarinas en 3D y también de ilustraciones. Se prestó especial atención a la vestimenta, llamada tutú, y también al pelo, ya que estos dos rasgos son muy característicos de las bailarinas. Estos son unos ejemplos de referencias usadas para el posterior diseño.

Figura 65 y 66: Referencias de bailarina

Figura 67 y 68: Referencias de bailarina

5.12.1.2 Dibujo a mano del diseño

Con todas las referencias y la idea de bailarina que se tenía en mente, se realizó el boceto para utilizar de referencia. En el momento de realizar el dibujo aún tenía pensado que corriera de puntillas, es por eso por lo que se representa en el boceto de la Figura 69.

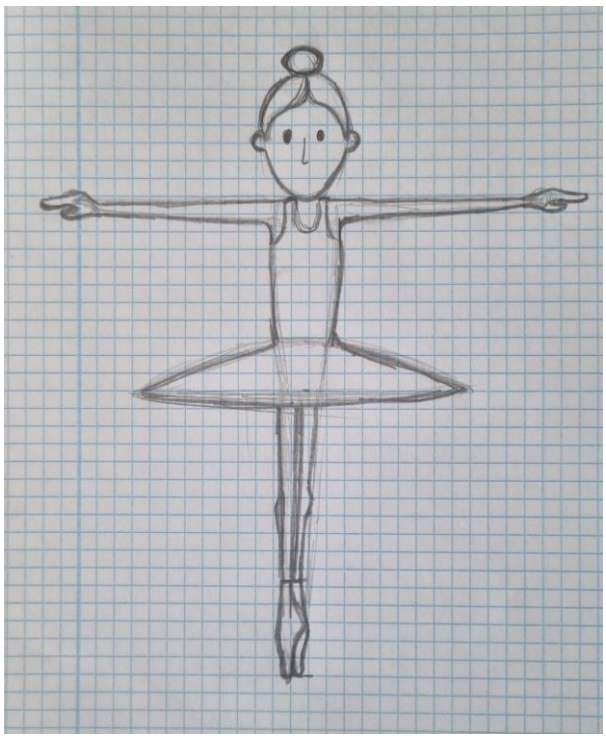

Figura 69: Boceto a lápiz de la bailarina

5.12.1.3 Digitalización del dibujo

Con la ayuda de la aplicación para móvil llamada *Notebloc* se ha digitalizado el dibujo. Esta aplicación permite hacer una foto y hace la función de escáner. Aplica las correcciones de color necesarias para resaltar más las líneas y que los blancos sean más blancos. Con la digitalización obtenemos el resultado de la Figura 70.

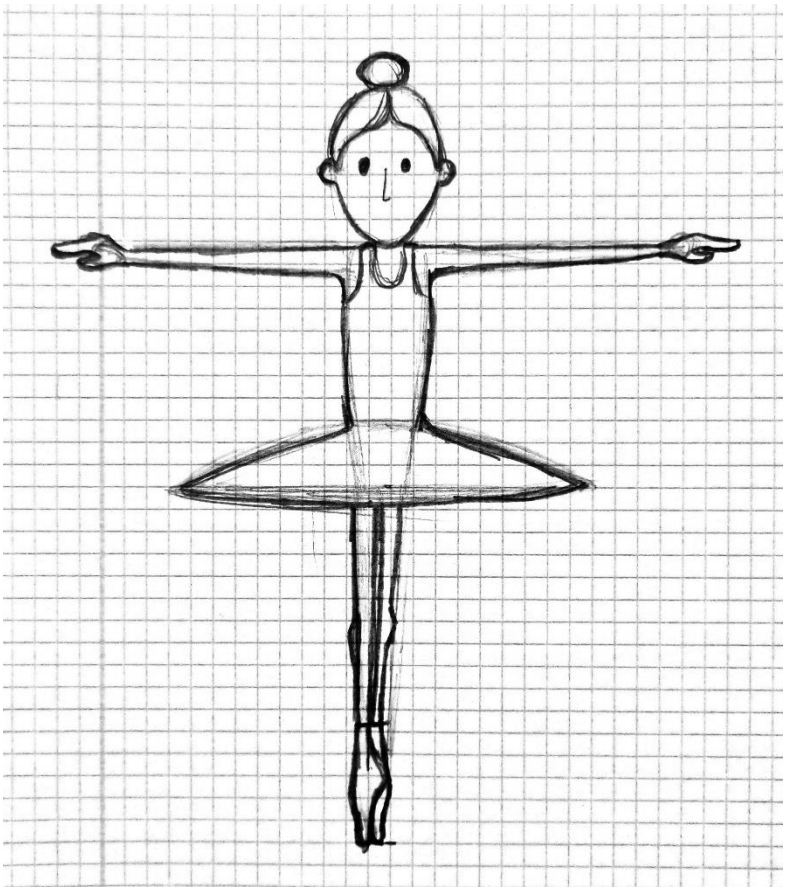

Figura 70: Boceto digitalizado de la bailarina

5.12.1.4 Modelado

La bailarina se dividió en las siguientes partes: cabeza, torso y brazos, tutú y piernas. Todas las partes menos la cabeza se realizaron solamente modelando, así que en este apartado nos centraremos en estas partes. Al ser una figura completamente simétrica, se modela la mitad de la figura y se aplica un modificador de Mirror para que con el espejo se complete el otro lado de la figura. Para empezar a modelar la figura siempre se parte de una figura simple, en este caso se parte de un cilindro al que se le van extruyendo caras nuevas y escalándolas para darle la forma deseada. Por ejemplo, en el tutú (Figura 71) se puede ver claramente como se ha trabajado con la mitad de la figura, partiendo de medio cilindro.

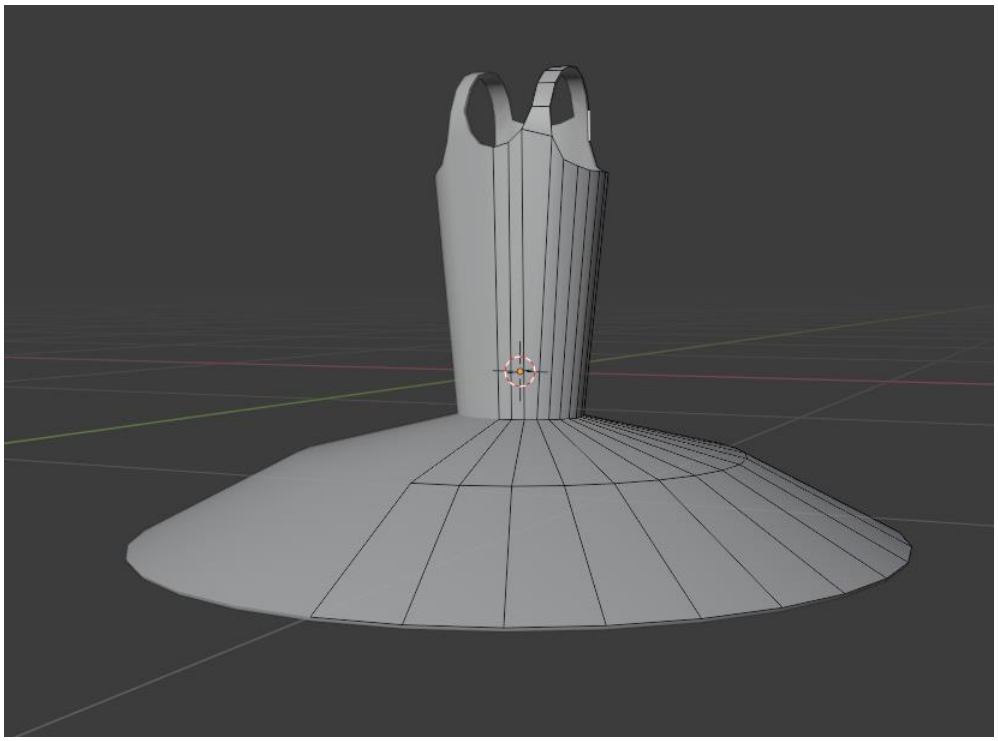

Figura 71: Modelado del tutú

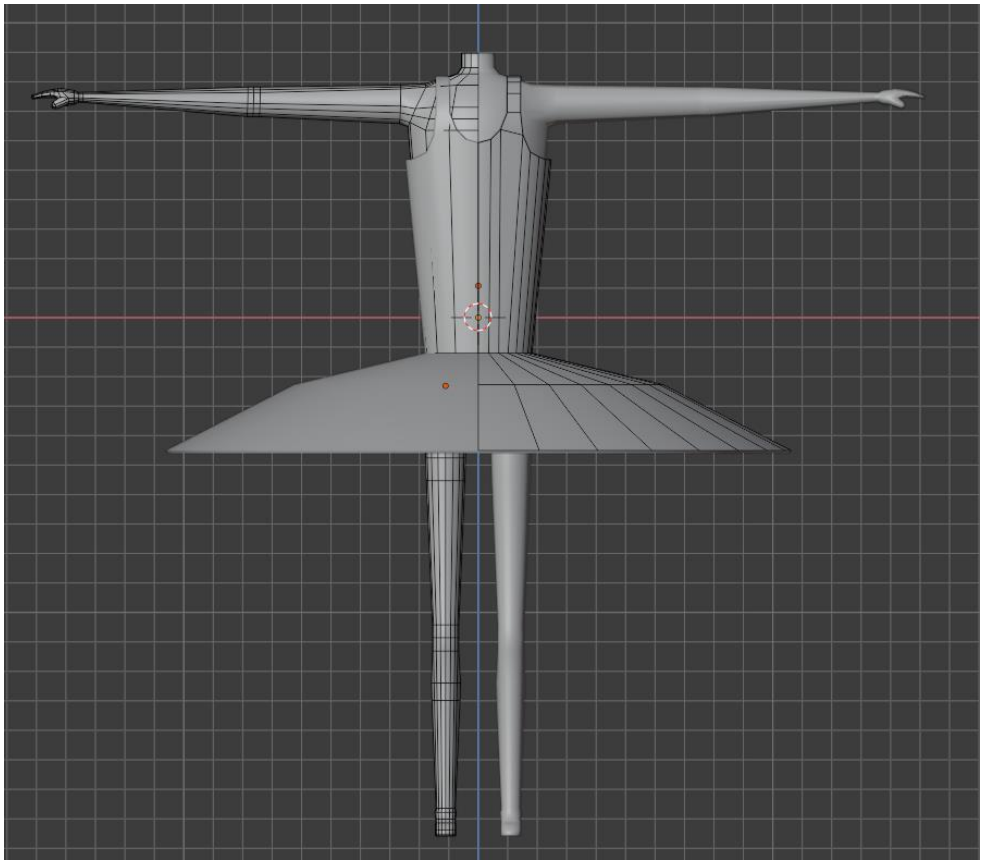

Figura 72: Vista frontal de las partes modeladas

También se tuvo que modelar la pala, utilizando referencias obtenidas en internet y de una pala que posee el desarrollador de este proyecto. Ver Figura 73.

Figura 73: Referencia de la pala

Como la pala dentro del juego no se va a ver en detalle, se han obviado los agujeros para ahorrar polígonos extras que ralentizarían el rendimiento del juego innecesariamente, como se puede ver en la Figura 74.

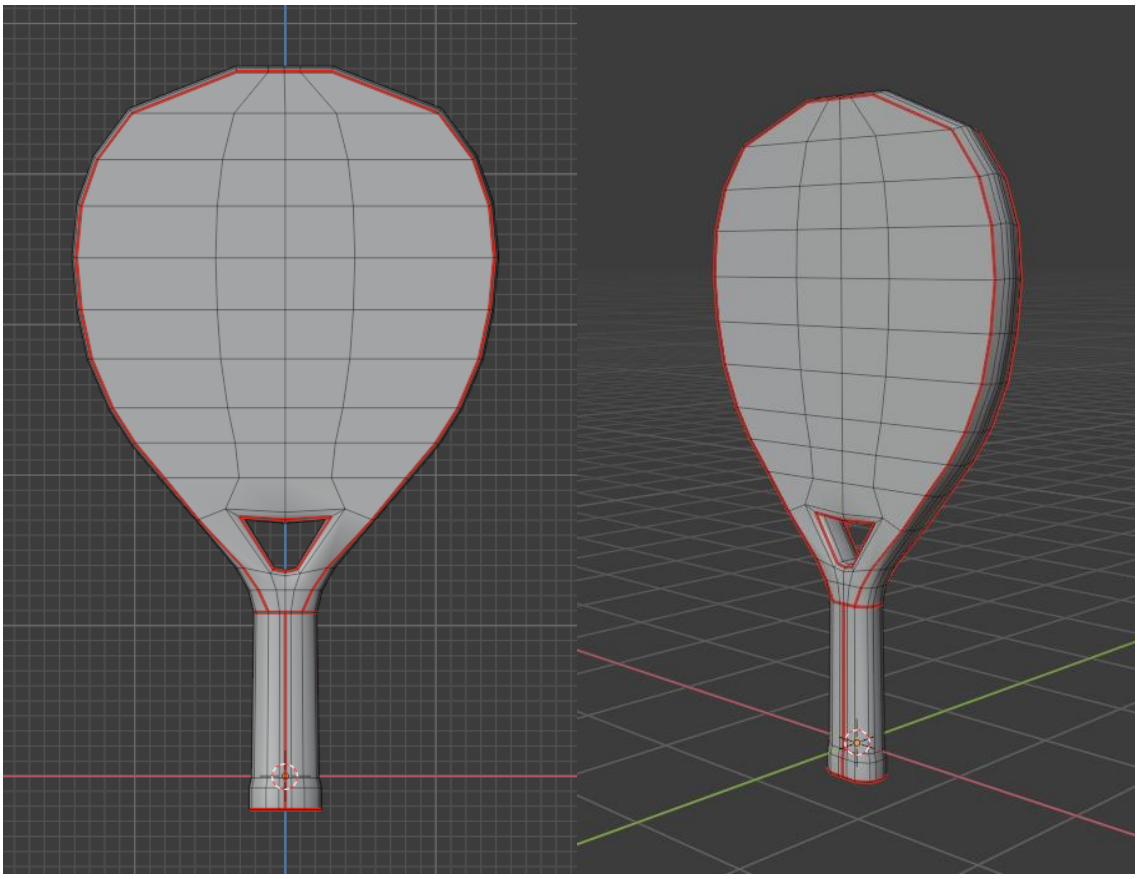

Figura 74: Modelado de la pala

5.12.1.5 Esculpido

La diferencia entre esculpir y modelar es que modelando se manipulan las caras, aristas o vértices directamente y en la escultura se usan pinceles para manipular los vértices. La escultura digital es parecida a manipular una masa de plastilina, permitiendo así hacer figuras orgánicas de forma más simple. La escultura sólo se usa en la cabeza de la bailarina, principalmente para hacer la nariz y la cavidad de los ojos. En escultura también se puede activar el modo espejo para conservar la simetría de la figura. La cabeza está formada por tres elementos: cráneo, ojos y orejas. Los ojos son esferas con la escala modificada, las orejas son cilindros también con la escala modificada y rotado de forma que quede en una posición adecuada, quedando el resultado de las Figuras 75 y 76.

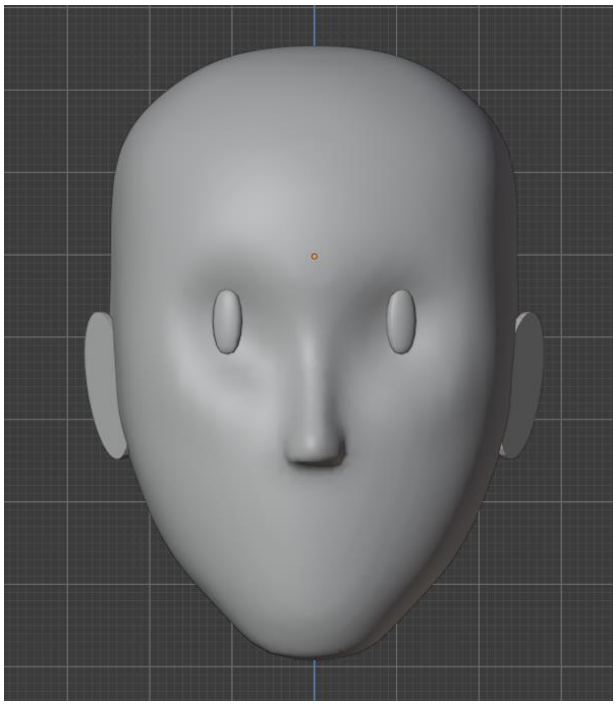

Figura 75: Esculpido de la cabeza vista frontal

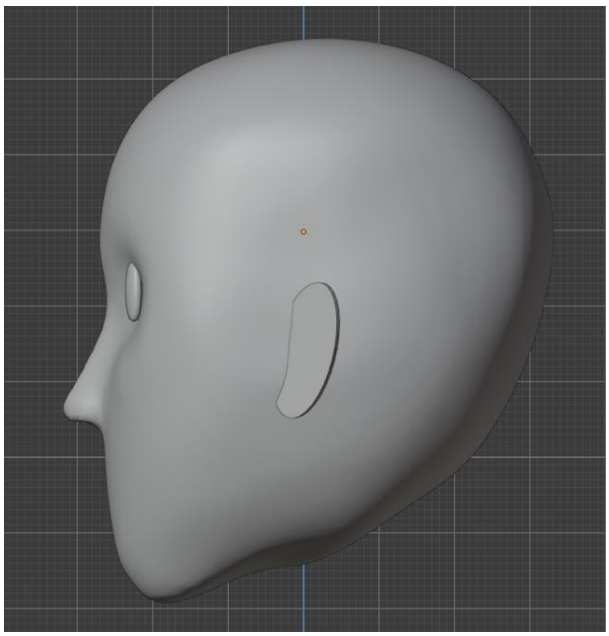

Figura 76: Esculpido de la cabeza vista lateral

Al cráneo se le aplicó un modificador llamado *Remesher* para reducir el número de polígonos, pero siempre conservando la forma. También puede ayudar a limpiar la topología de la maya.

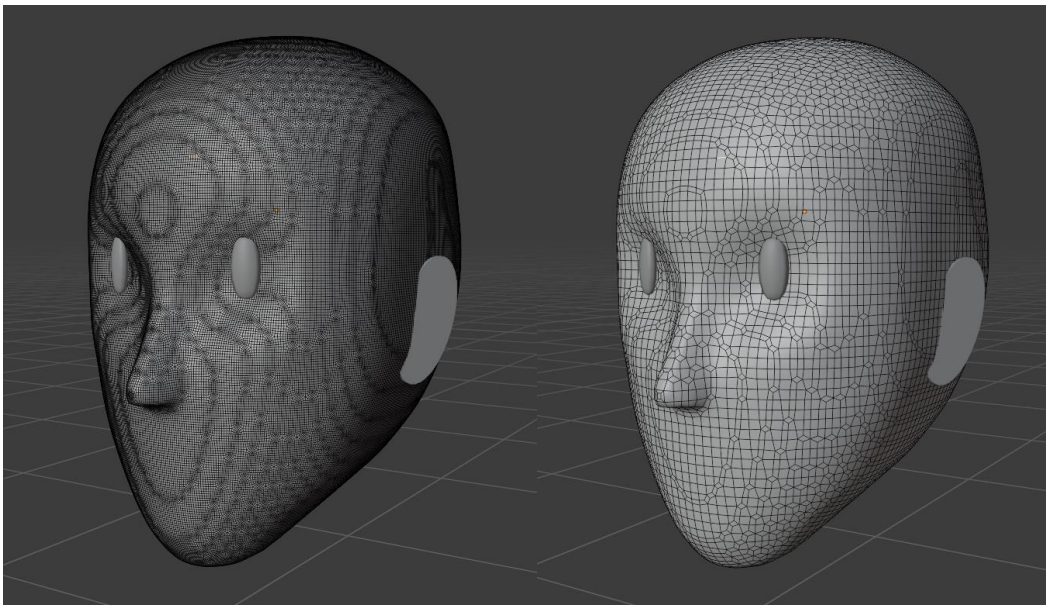

Figura 77: Comparativa de polígonos en el cráneo

El pelo también se ha esculpido simétricamente para darle forma y se le han añadido detalles con un pincel más fino, recreando la dirección de los mechones de pelo. También se le ha aplicado el modificador *Remesher* con el mismo objetivo que en el cráneo.

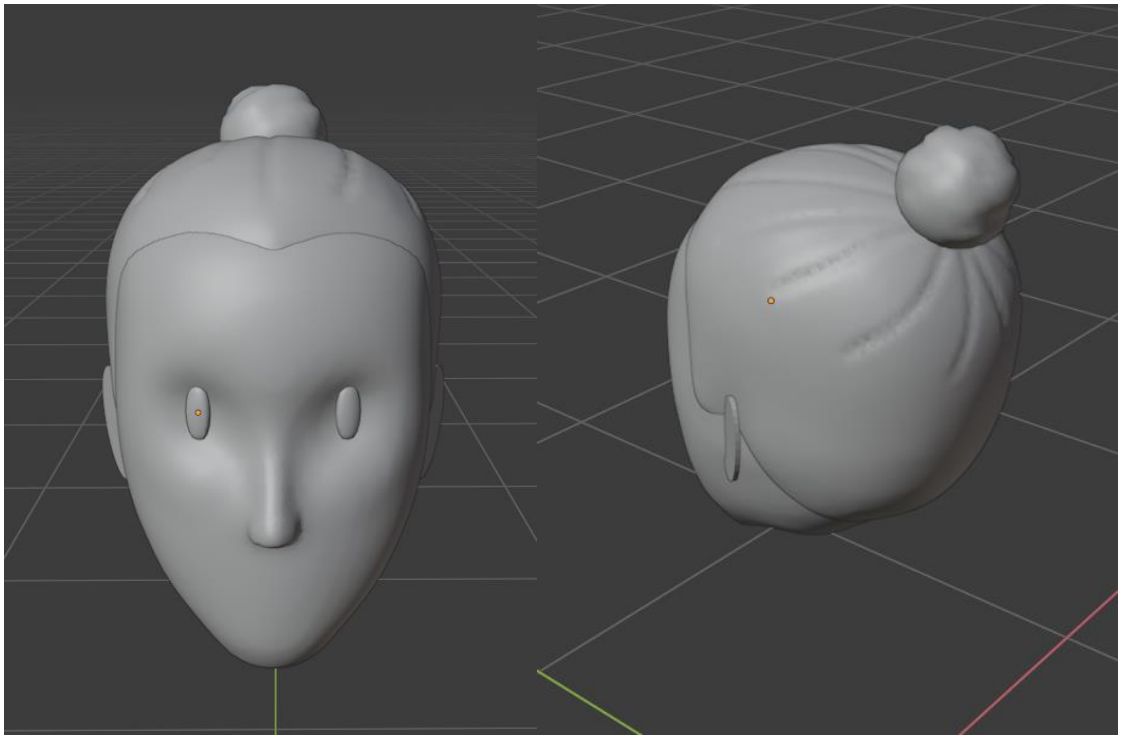

Figura 78: Resultado final del pelo

5.12.1.6 Texturización

En este caso no se han aplicado texturas basadas en imágenes, sino materiales con colores planos. Así que la tarea de texturizados se ha reducido a elegir los colores de la piel, el tutú, el pelo y los ojos. Se ha optado por colores poco fantasiosos, más bien realistas, obteniendo los resultados de las Figuras 79 y 80.

Figura 79: Resultado final de la bailarina

Figura 80: Resultado final de la bailarina

Para ver la evolución desde el boceto hasta el acabado de la texturización, se ha hecho la Figura 81.

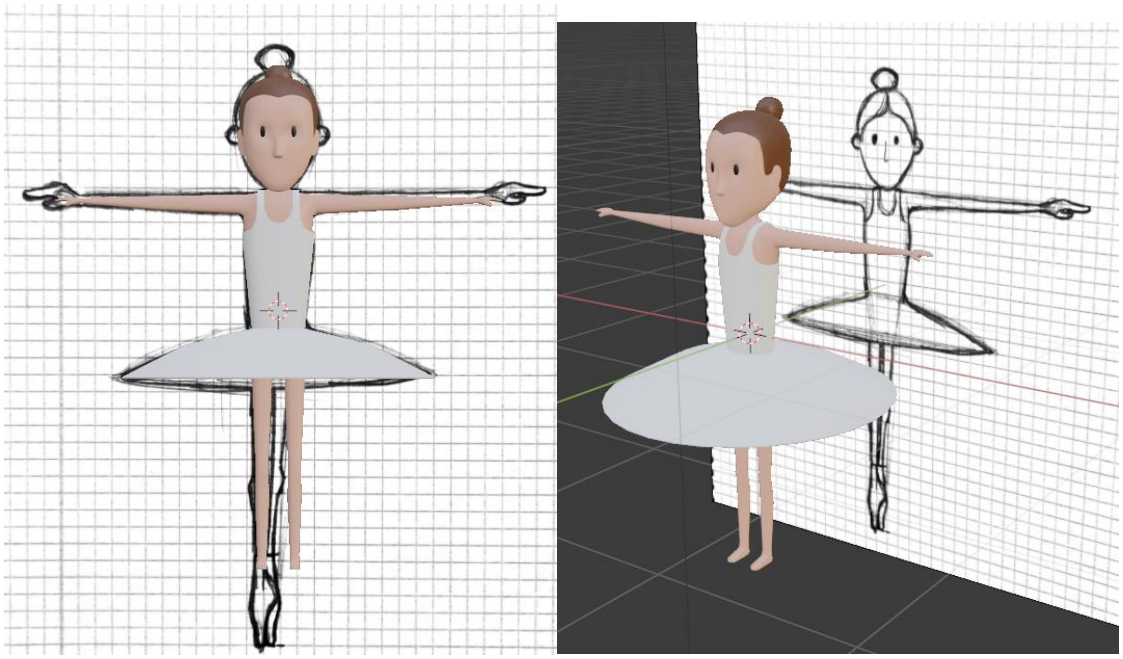

Figura 81: Comparación del modelo 3D con el boceto

5.12.1.7 Rigging

El proceso de *rigging* es necesario para aplicar animaciones sobre nuestro personaje. En este proceso se le otorgan huesos a nuestro personaje y estos huesos tienen influencia en los vértices de la maya más cercanos. Determinando así el comportamiento que tendrán estos vértices en el momento que el hueso se mueva. Al conjunto de huesos se le llama esqueleto, en este caso se trata de un esqueleto simple ya que no cuenta con huesos para cada dedo ni huesos para las expresiones faciales. El esqueleto desarrollado se puede ver en la Figura 82.

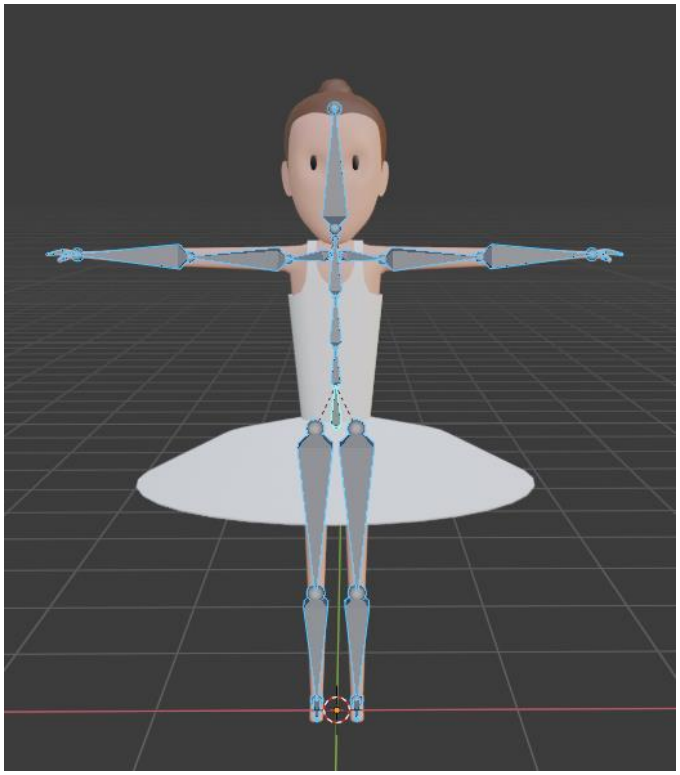

Figura 82: Esqueleto de la bailarina

Para cada hueso hace falta pintarle su influencia sobre los vértices, Blender facilita una herramienta que permite ver qué vértices están siendo influenciados por el hueso seleccionado y en qué grado están afectados, con una escala de colores típica de sensores de temperatura, siendo azul poco o nada influenciados y rojo muy influenciados, tal y como se puede ver en la Figura 83.

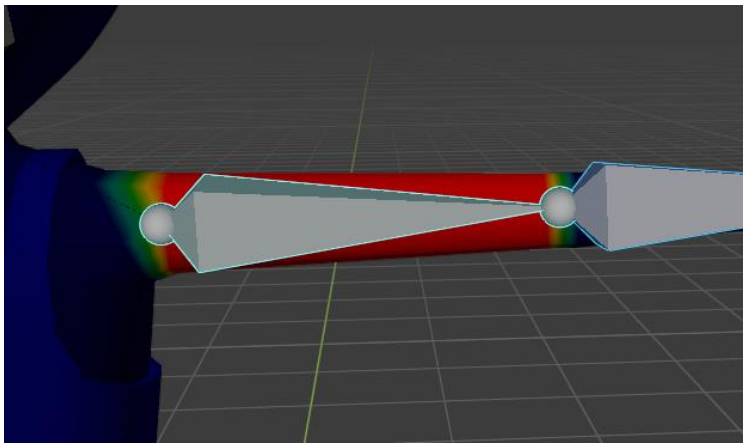

Figura 83: Influencia del hueso superior del brazo

5.12.1.8 Animación

Para las animaciones básicas como correr o estar quieto se ha recurrido a la biblioteca de Mixamo, donde se encuentran multitud de animaciones gratuitas y útiles. Pero para las animaciones más específicas como pueden ser el golpe de derechas o el golpe de revés se han hecho de cero. Para la realización de las animaciones de los golpes se buscó consejos para realizar los golpes de pádel en internet. En estos tutoriales enseñan los movimientos que hay que hacer paso por paso, estos pasos son en los que se hizo especial atención para recrearlos en los *keypoints*. Un *keypoint* es un punto en el tiempo donde se guarda la posición y rotación de un objeto. Juntando varios *keypoints* se crean las animaciones interpolando las posiciones entre los *keypoints*. En la Figura 84 se recrea esta interpolación.

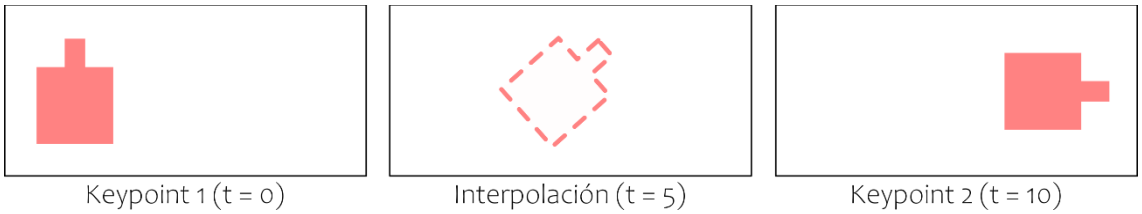

Figura 84: Recreación de una interpolación

Por el tipo de controles del juego en el momento de golpear la pelota (parar al jugador en el momento que carga el golpe), se ha visto necesario hacer las siguientes animaciones (serían necesarias más animaciones, pero por falta de tiempo no se pudieron llevar a cabo):

- **Esperar (*Idle*):** Esta animación fue cogida de Mixamo, en el pádel se espera con la pala levantada, situada a la altura de la cabeza, y las piernas semiflexionadas. Como en Mixamo no había animaciones que representaran este movimiento específico, se cogió una animación de un portero de fútbol, cuya posición se asemeja. Posteriormente en Blender se modificó para que la pala no chocara con la cabeza. En la Figura 85 se ve un momento de la animación para hacerse una idea de la posición de espera.

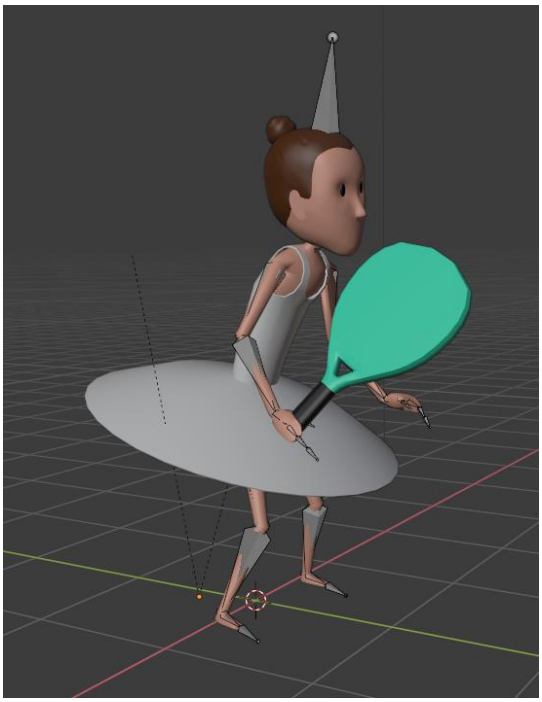

Figura 85: Animación de espera

- **Correr:** Esta animación también fue importada de Mixamo y modificada para que la pala no chocara con la cabeza de la bailarina. Ver Figura 86.

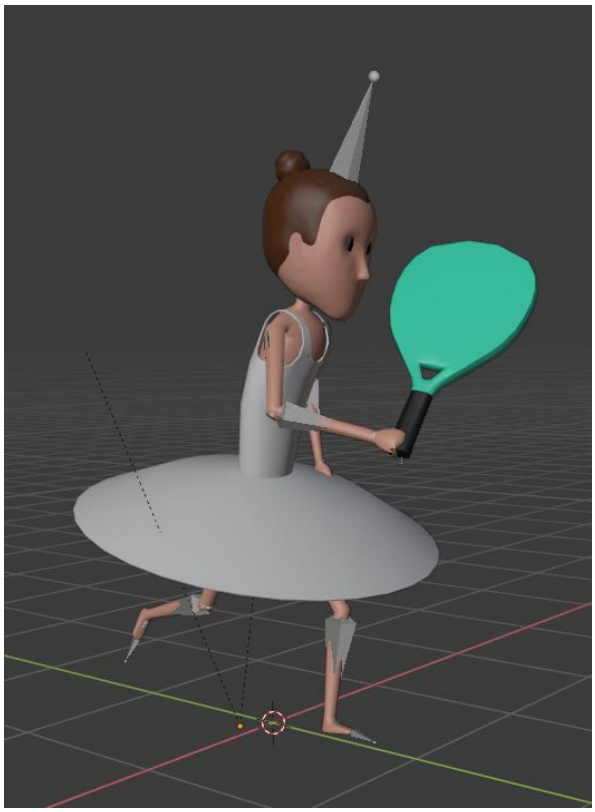

Figura 86: Animación de correr

- **Preparar drive:** En el momento que el jugador empiece a cargar el golpe, se ejecutará esta animación. Se trata de una animación que se pasa de una pose de espera a una pose de cargar el tiro (Figura 87).

Figura 87: Animación de preparar el drive

- **Drive:** Después de preparar el drive, cuando el jugador suelta el botón se ejecuta la animación del golpe de derechas, acabando el movimiento empezado por la preparación del drive. En la Figura 88 se observa la posición final del golpe de derechas.

Figura 88: Animación de golpe de derechas

- **Preparar revés:** Igual que en el drive, en el revés pasa lo mismo. Esta animación, se ejecuta cuando la pelota está situada a la izquierda del jugador. Ver Figura 89.

Figura 89: Animación de preparar el revés

- **Revés:** Se ejecuta en el momento que el jugador suelta el botón de golpear. En la Figura 90, podemos apreciar lo anterior.

Figura 90: Animación de golpe de revés

5.12.2 Leñador

El leñador sigue con la estética cartoon, agrandando la cabeza y estrechando las extremidades. A diferencia de la bailarina, el leñador no dispondrá de pala de pádel y jugará con el hacha. Esta característica da una explicación narrativa a la habilidad del leñador. La metodología que se ha seguido para desarrollar este personaje ha sido diferente, ya que, en el proceso de desarrollo de este proyecto, el autor de esta memoria aprendió nuevas técnicas de modelado de personajes mediante un curso por internet. Este curso dividía el proceso de modelado en las siguientes partes: Búsqueda de referencia, *blocking,* escultura, texturización y animación.

5.12.2.1 Búsqueda de referencia

En este curso se proporciona un *concept art* para esculpirlo. Un *concept art* es una ilustración que tiene como objetivo visualizar un posible resultado final. Así pues, el *concept art* que se usará de referencia es el de la Figura 91.

Figura 91: Referencia del leñador

5.12.2.2 Blocking

El *blocking* es una técnica de escultura que se centra en preparar la figura para posteriormente esculpir sobre ella. Se basa en hacer una figura general y aproximada a la deseada con el fin de visualizar y corregir proporciones, formas, siluetas, etc. Esta figura estará formada por volúmenes simples como por ejemplo cilindros, cubos, esferas, etc. En las Figuras 92 a la 94, se enseña la evolución del proceso de *blocking.*

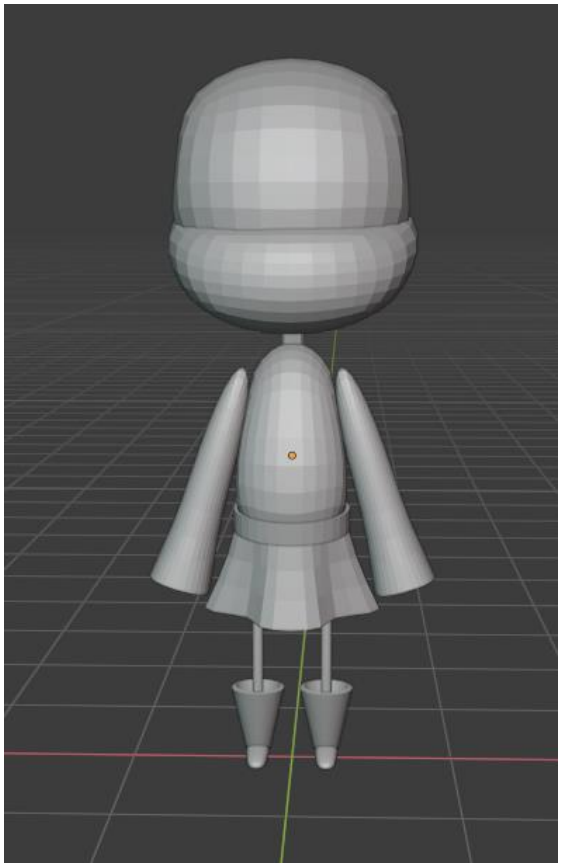

Figura 92: Primer blocking

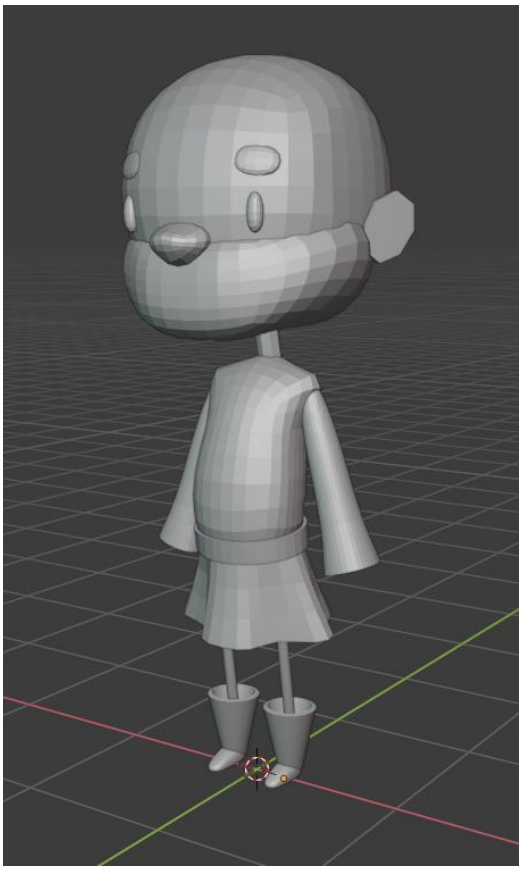

Figura 93: Segundo blocking

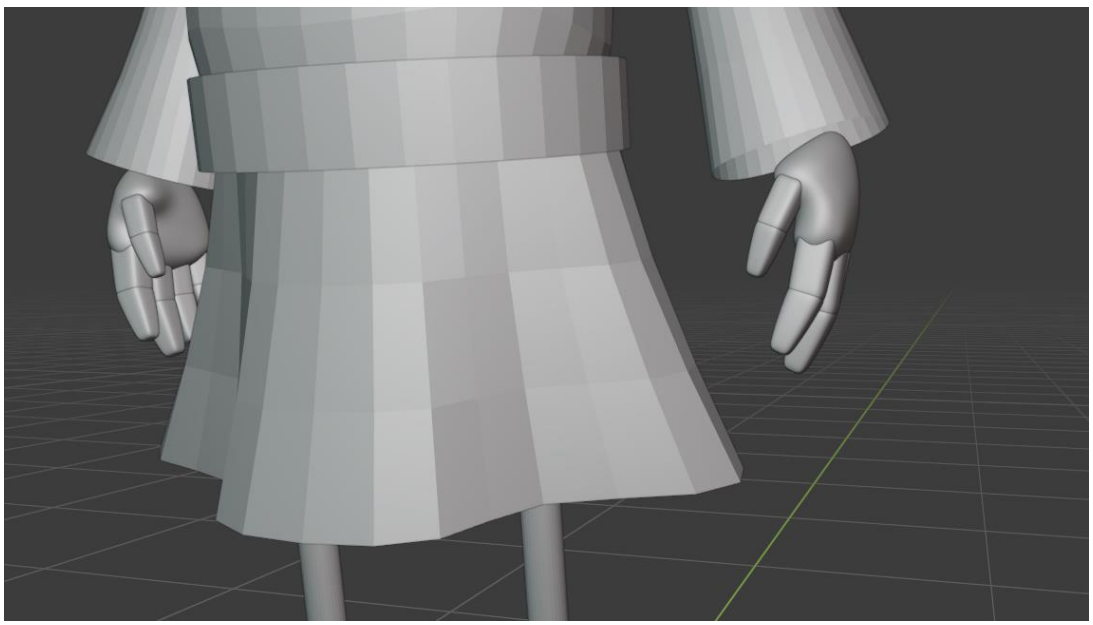

Figura 94: Blocking de las manos

Durante el proceso es buena idea cambiar el modo de visualizar el modelo, en este caso se optó por una visualización con colores aleatorios para distinguir las partes que forman al leñador, como podemos observar en las Figuras 95 y 96.

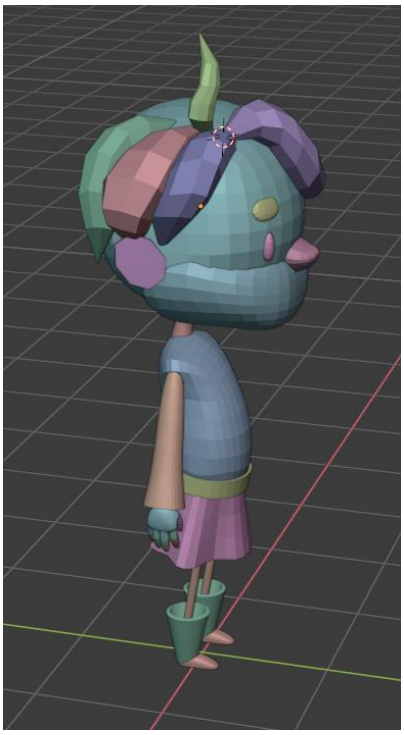

Figura 95: Cuarto blocking

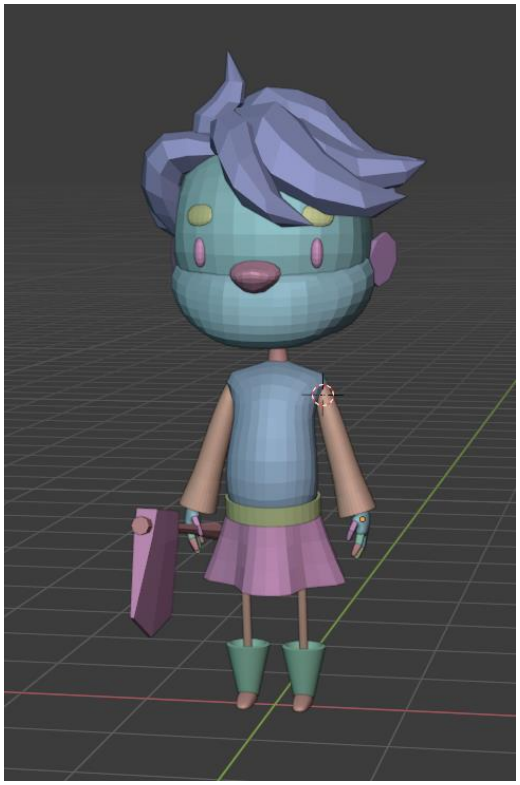

Figura 96: Blocking final

5.12.2.3 Escultura

Durando el proceso de *blocking* se utilizan técnicas de escultura, pero únicamente para dar forma, en este paso nos referiremos a esculpir para dar detalle a nuestra figura. El personaje del leñador requería muchas horas de trabajo por lo que no ha dade tiempo a acabarlo. Sin embargo, en la Figura 97 se pueden ver algunos detalles hechos con escultura sobre la cara del leñador, como pueden ser los orificios de la nariz y las orejas.

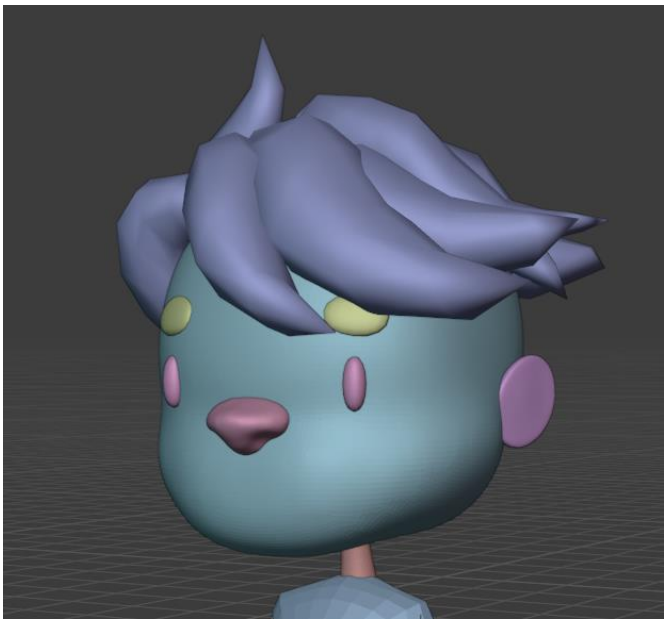

Figura 97: Escultura aplicada al leñador

5.12.2.4 Próximos pasos

Una vez se tiene al leñador completamente esculpido con todos sus detalles, de vuelve a la misma metodología seguida en la bailarina. Los pasos restantes serían: texturización, *rigging*, animación. Para la texturización, se aplicarán colores planos, igual que los que se habían aplicado en la bailarina. En la fase de *rigging* se pintarían las influencias del nuevo esqueleto y sobre este esqueleto se aplicarían las mismas animaciones presentes en la bailarina.

5.13 Herencia del jugador

Para evitar la duplicación de código, se ha creado una superclase nombrada PlayerMovement y de ella heredan las clases Bailarina y Leñador. Estas clases específicas permiten configurar aspectos como la velocidad del personaje y también cuentan con métodos propios. La parte más interesante es la parte de la habilidad especial de cada personaje. Las dos clases cuentan con un método llamado SpecialMovement, pero en cada clase se ejecuta el código necesario para cada habilidad.

5.14 Habilidades

Cada tipo de personaje cuenta con una habilidad especial, en este proyecto se han llevado a cabo dos habilidades.

5.14.1 Habilidad de Bailarina

La bailarina será capaz de realizar un salto hacia delante de forma rápida para alcanzar pelotas lejanas. Para efectuar ese salto se tendrá en cuenta la dirección hacia donde se quiere ir y un tiempo pequeño que será el tiempo que estará la bailarina haciendo el salto. Internamente, no se ejecuta ningún salto, sino que se duplica su velocidad durante un periodo corto de tiempo sin dejar que el jugador cambie la dirección del salto mientras éste se está ejecutando.

```
public void onSpecialMovement(InputAction.CallbackContext ctx){

    // Ejecta la función Dashing cuya función tiene un temporizador
    StartCoroutine(Dashing());

}

IEnumerator Dashing(){

    // Duplica la velocidad
    base.speed *= 2;

    // Espera un periodo de tiempo corto
    yield return new WaitForSeconds(dashTime);

    // Vuelve a establecer la velocidad adecuada
    speed /= 2;
}
```

5.14.1 Habilidad de Leñador

El leñador, usando su hacha, podrá “cortar la pelota en dos” y enviar dos pelotas al campo contrario. Una de estas dos pelotas será falsa y desaparecerá al cabo de un segundo. Esta habilidad se puede ejecutar en el momento que la pelota real está al alcance del leñador. La pelota ficticia se instanciará en la posición de la pelota real y tendrá la dirección apuntada por el leñador con una altura aleatoria. Esta pelota ficticia solo cuenta con un componente *Rigidbody* que se encarga de aplicarle la física a la pelota y a diferencia de la pelota real, no cuenta con colisionador, por lo tanto, no chocará con ningún objeto ni disparador.

```
public void onSpecialMovement(InputAction.CallbackContext ctx){

    if(ball == null){
        // Encontrar la pelota real para obtener su posición
        DetectBall();
    }

    // Si la pelota real está al alcance y la ficticia no se ha
       generado todavia
    if(ballInRange && !fakeBallSpawned){

        fakeBallSpawned = true;

        // Instancia la pelota ficticia
        fB = Instantiate(fakeBallPrefab, ball.transform);

        // Establecer valores de direción y altura
        Vector3 direction = new Vector3(movementInput.x,
                                        0f,
                                        movementInput.y);

        Vector3 height = Vector3.up * Random.Range(0, 1000f);

        // Disparar la pelota ficticia
        ShotFakeBall(fB, direction.normalized * power + height);

        // Borrar pelota ficticia con temporizador
        StartCoroutine(DeleteFakeBall());
    }
}

IEnumerator DeleteFakeBall(){
    // Esperar un segundo
    yield return new WaitForSeconds(1f);

    // Borrar la pelota
    Destroy(fB);
    fakeBallSpawned = false;
}

void ShotFakeBall(GameObject fakeBall, Vector3 direction)
{
    // Obtener componenete Rigidbody de la pelota ficticia
    Rigidbody ballRb = fakeBall.GetComponent<Rigidbody>();

    //Efectuar golpe
    ballRb.velocity = Vector3.zero;
    ballRb.AddForce(direction * 2f);


}
```

En el momento que se instancia la pelota, se activa la variable booleana *fakeballSpawned* para evitar que el jugador pueda instanciar más de una pelota ficticia a la vez.

5.15 Animaciones

Las animaciones creadas anteriormente se deben exportar al motor de videojuegos Unity, para gestionarlas desde ahí. Para gestionarlas, Unity ofrece una interfaz visual en forma de máquina de estados que permite establecer conexiones entre animaciones. Estas relaciones se pueden personalizar con las condiciones que se tienen que dar para pasar de una animación a otra. También se puede configurar cada animación con parámetros como, por ejemplo, que la animación se tenga que finalizar antes de pasar a la siguiente, que se repita la animación una vez acabada, incluso la curva de progresión entre animaciones (hacer más o menos brusco el cambio entre animaciones).

En este proyecto se ha hablado de cinco tipos de animaciones, estas son las relaciones entre ellas y las condiciones que se deben cumplir:

- **Esperar**

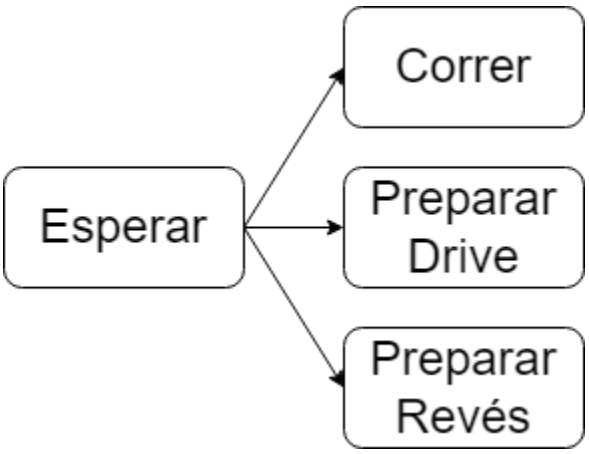

Figura 98: Relaciones de la animación de esperar

Condiciones:

+ Esperar -> Correr : isRunning = true
+ Esperar -> Preparar Drive : isPreDrive = true
+ Esperar -> Preparar Revés : isPreReves = true

- **Correr**

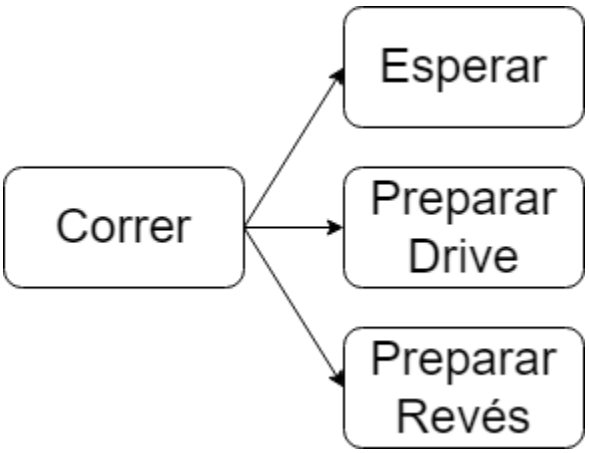

Figura 99: Relaciones de la animación de correr

Condiciones:

+ Correr -> Esperar : isRunning = false
+ Correr -> Preparar Drive : isPreDrive = true
+ Correr -> Preparar Revés : isPreReves = true

- **Preparar Drive**

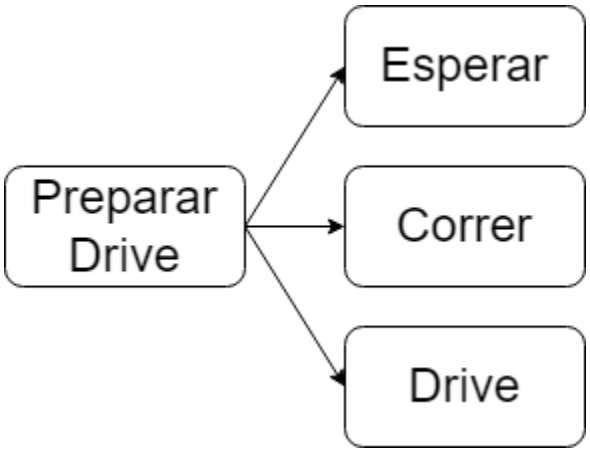

Figura 100: Relaciones de la animación de preparar drive

Condiciones:

+ Preparar Drive -> Esperar : isRunning = false
  isPreDrive = false
  isShooting = false
+ Preparar Drive -> Correr : isRunning = true
  isPreDrive = false
  isShooting = false
+ Preparar Drive -> Drive : isShooting = true

- **Drive**

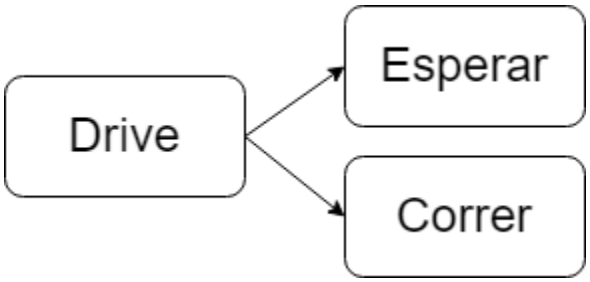

Figura 101: Relaciones de la animación de drive

Condiciones:

+ Drive -> Esperar : isRunning = false
+ Drive -> Correr : isRunning = true

- **Preparar Revés**

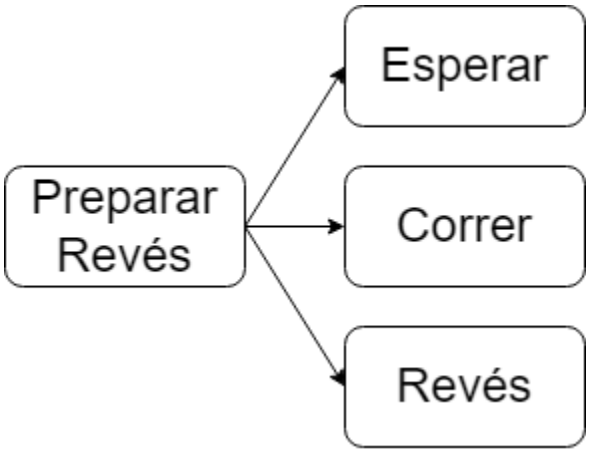

Figura 102: Relaciones de la animación de preparar drive

Condiciones:

+ Preparar Revés -> Esperar : isRunning = false
  isPreDrive = false
  isShooting = false
+ Preparar Revés -> Correr : isRunning = true
  isPreDrive = false
  isShooting = false
+ Preparar Revés -> Revés : isShooting = true

- **Revés**

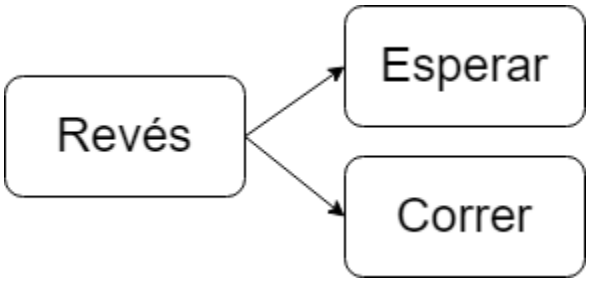

Figura 103: Relaciones de la animación de drive

Condiciones:

+ Revés -> Esperar : isRunning = false
+ Revés -> Correr : isRunning = true

En el código se activan estas variables en los momentos oportunos. Por ejemplo, en el momento que el jugador se mueve, la variable *isRunning* se establece como cierta, tal y como se puede ver en el siguiente fragmento de código.

```
if (direction.magnitude > 0f && !freezed)
{
    if (!isRunning){
        animator.SetBool("isRunning", true);
    }
}
```

Si el vector dirección tiene la magnitud mayor que 0 y si el jugador no está paralizado (cargando el golpe), se comprueba si la variable *isRunning* está ya establecida en cierta, en caso contrario se activará. La variable *animator* guarda el componente *Animator*, que es el que se ocupa de gestionar las animaciones.

En la Figura 104 se muestra el diagrama de animaciones completo desde la interfaz de Unity.

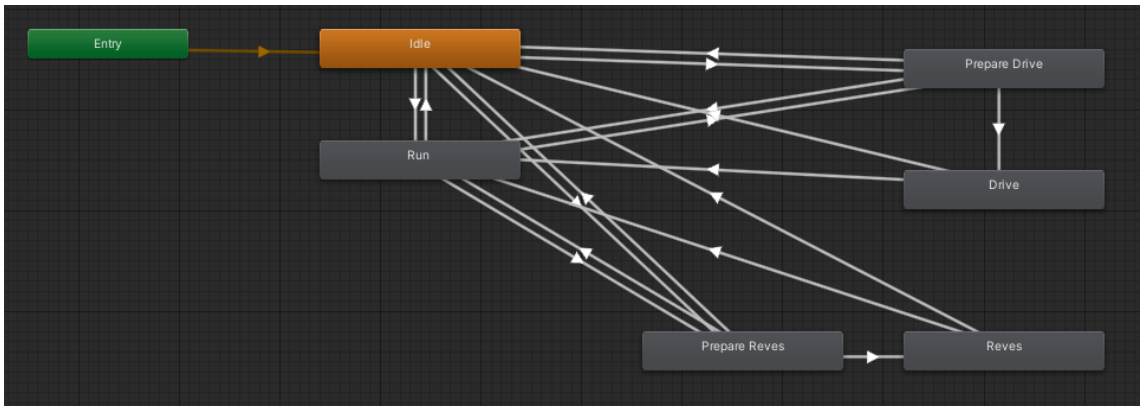

Figura 104: Maquina de estados de las animaciones

Por defecto la animación de entrada será la animación de esperar (Idle). En este tipo de diagrama de animaciones es importante cerrar los caminos en forma de bucle, para que no se quede paralizado en una animación

5.16 Controles

Al tratarse de un juego multijugador local, no basta con detectar si un botón se ha apretado. Hay que diferenciar qué dispositivo de control lo ha apretado. Para ello se utilizó el componente *Player Input Manager* en un objeto global, que diferenciará el dispositivo y el componente *Player Input* en los personajes, que invocará eventos los cuales serán los encargados de influir en la lógica del juego, ver Figura 105. El componente *Player Input* requiere un mapa de acciones, que se trata de asociar los botones del dispositivo que nos interesa con nombres de eventos (Figura 105) y, seguidamente, estos nombres irán asociados a métodos del código, como se puede ver en la Figura 106.

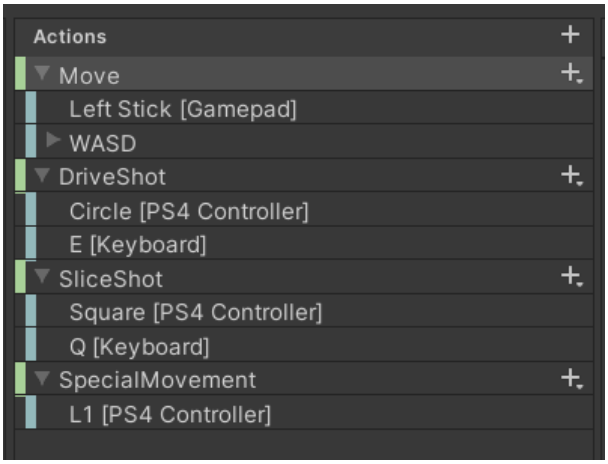

Figura 105: Mapa de acciones

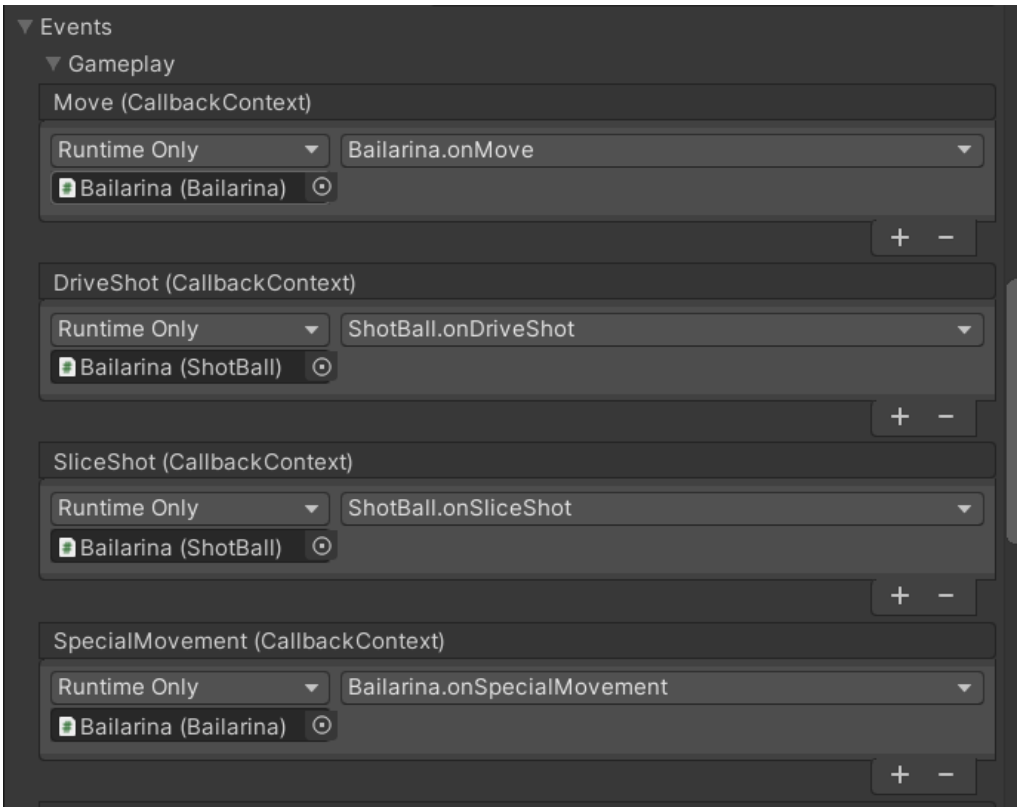

Figura 106: Eventos de controles de la bailarina

Dentro del código se obtiene el valor de la entrada (input) del jugador. Por ejemplo, en el evento *Move*, asociado al método *onMove*, se obtendrá la dirección hacia la que el personaje debe moverse, de la siguiente manera:

```
public void onMove(InputAction.CallbackContext ctx) {
    movementInput = ctx.ReadValue<Vector2>();
}
```

En otros eventos sólo interesa saber que se ha apretado el botón, por lo tanto, no interesará el valor de la entrada, como es el caso del evento *SpecialMovement*, que se trata de la siguiente manera:

```
public void onSpecialMovement(InputAction.CallbackContext ctx){
    StartCoroutine(Dashing());
}
```

En este caso no usaremos el parámetro de entrada *ctx*.

5.17 Escenas

Las escenas o pantallas existentes en este proyecto están representadas a continuación (Figura 107) en un diagrama que indica el camino para llegar de una a otra.

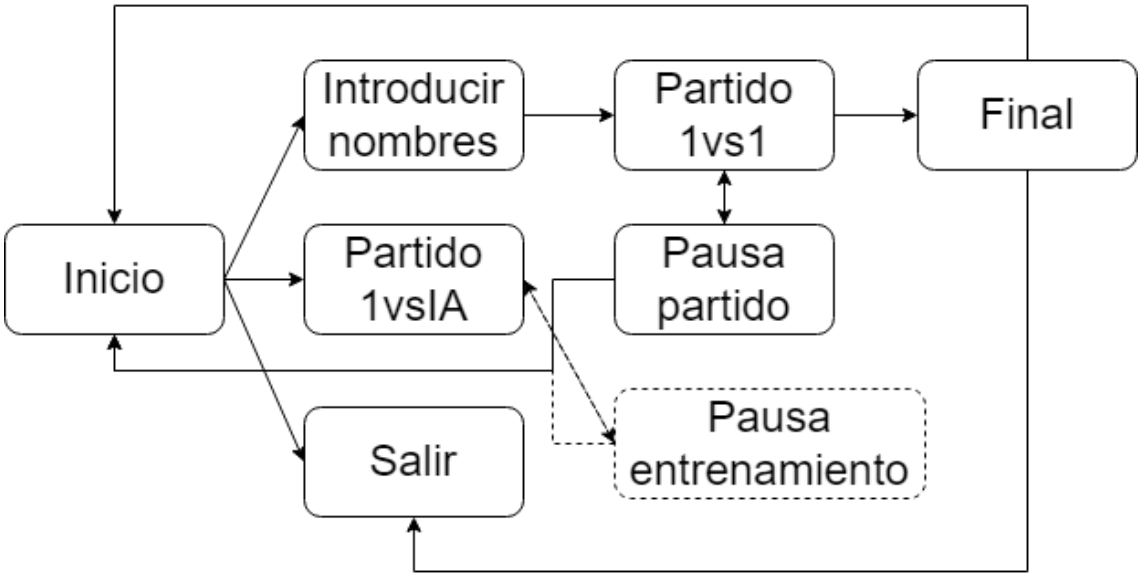

Figura 107: Diagrama de escenas

Las escenas contienen lo necesario para cumplir su función y no se ha invertido tiempo en hacer que sean visualmente atractivas.

5.17.1 Inicio

En la pantalla de inicio se elegirá el modo de juego deseado o la posibilidad de salir del juego.

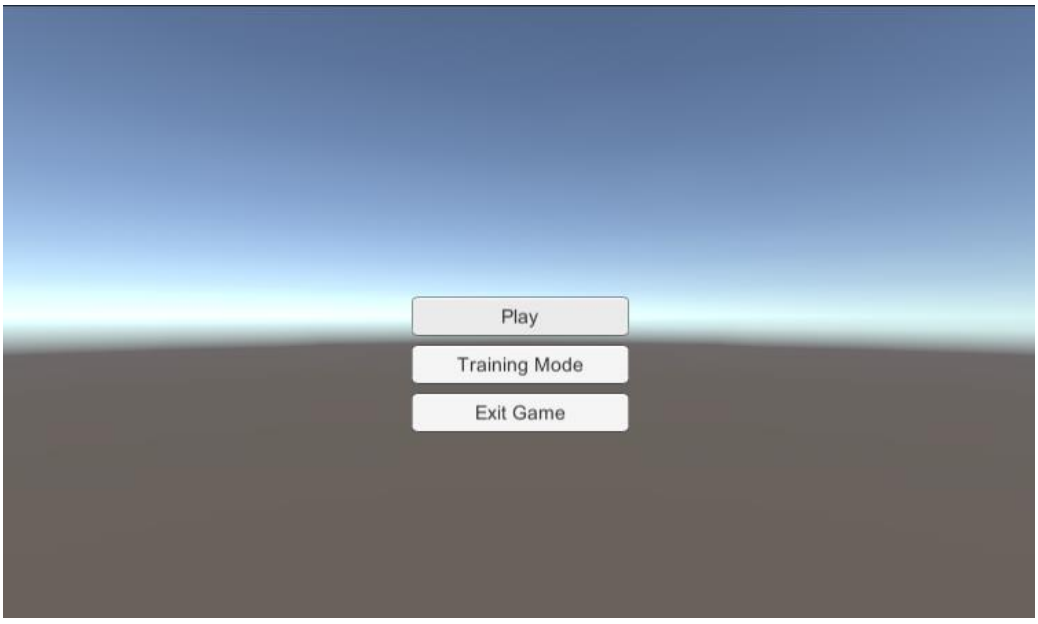

Figura 108: Escena de inicio

5.17.2 Introducir nombres

Es la pantalla que precede al partido 1vs1. En esta pantalla se introducirán los nombres de los jugadores para que posteriormente se muestren en el marcador, ver Figura 109.

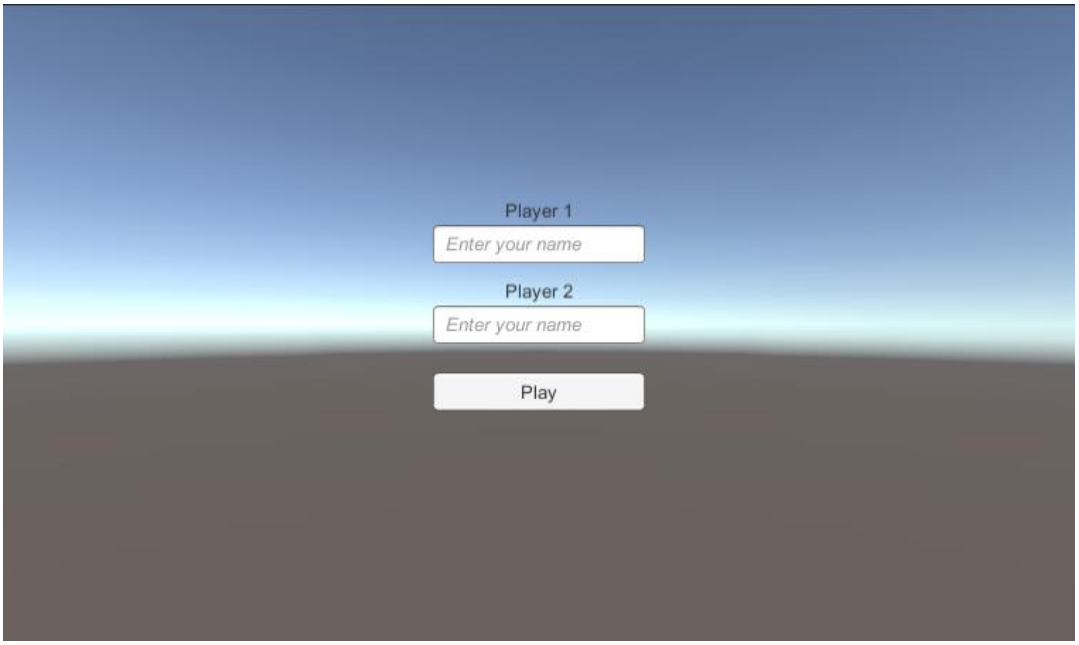

Figura 109: Escena de introducción de nombres

5.17.3 Partido 1vs1

Escena de juego principal donde se muestra el área de juego, en pantalla dividida, y el marcador, ver Figura 110.

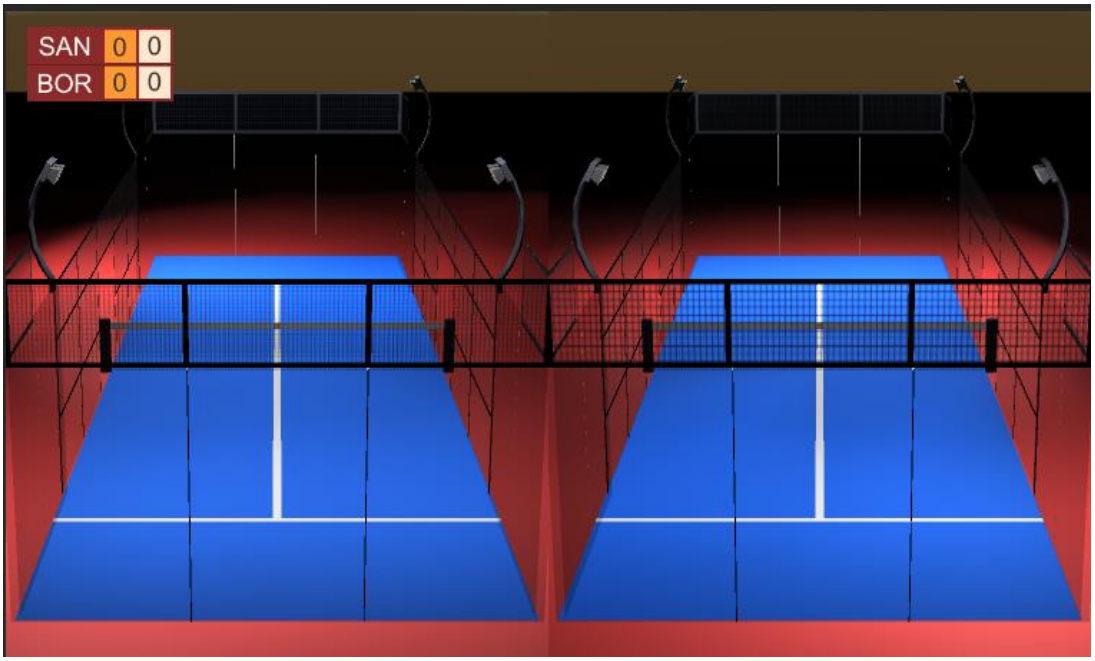

Figura 110: Escena de juego 1vs1

5.17.4 Pausa partido

No se trata de una escena en sí, más bien es una interfaz que se sobrepone encima del partido con los botones de *Resume*, que relanza el partido desde el punto donde se había quedado, y el botón de *Exit*, que permite a los jugadores abandonar el partido y volver a la escena de inicio. En la Figura 111 se pueden observar ambos botones.

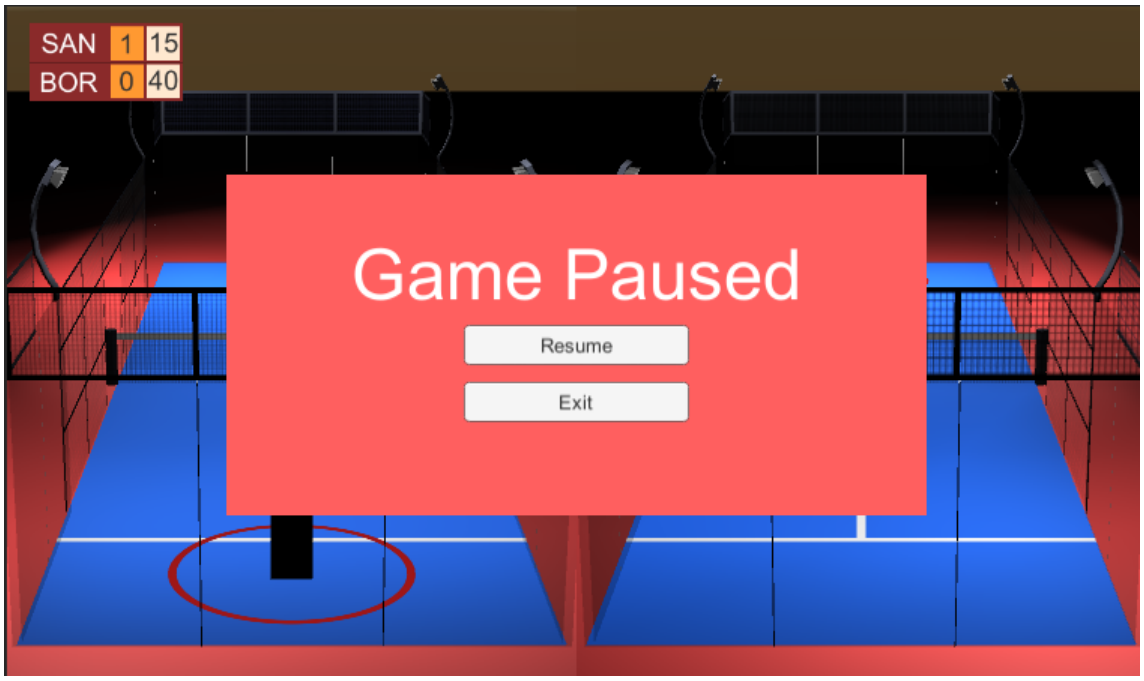

Figura 111: Interfaz de pausa

La interfaz de pausa se ha constituido de la forma que se puede ver en la Figura 112. Formada por un objeto padre cuyos hijos son los botones mencionados anteriormente, el texto y la imagen de fondo.

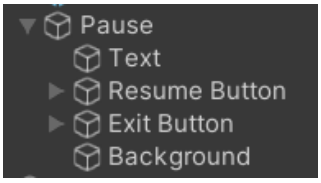

Figura 112: Jerarquía de la interfaz de pausa

Para pausar el partido, es necesario alterar la escala del tiempo y ponerla en 0, para que así no pase el tiempo. A su vez, también se debe mostrar la interfaz de pausa. Todo esto se ha hecho de la siguiente manera:

```
public void onPause(InputAction.CallbackContext ctx){

    if(pauseMenu.active){
        Time.timeScale = 1f;
        pauseMenu.Deactivate();

    }
    else{
        Time.timeScale = 0f;
        pauseMenu.Activate();
    }
}
```

Se ha creado un evento asociado a una acción del mapa de acciones y en este evento se comprueba si la interfaz ya es visible. En ese caso interpretaremos que el jugador quiere reanudar el juego. En caso contrario, interpretaremos que quiere pausarlo. La variable *pauseMenu* contiene la clase *PauseManager* la cual tiene los métodos usados por los dos botones de la interfaz y también tiene los métodos *Deactivate* y *Activate.* Estos dos últimos métodos se encargan de ocultar y mostrar los elementos hijos del objeto *Pause*:

```
public void Deactivate(){
    foreach (Transform child in transform)
        child.gameObject.SetActive(false);

    active = false;
}

public void Activate(){
    foreach (Transform child in transform)
        child.gameObject.SetActive(true);

    active = true;
}
```

Los métodos usados por los botones *Resume* y *Exit* son los siguientes:

```
public void onResume(){
    Deactivate();
    Time.timeScale = 1f;
}

public void onExit(){
    Deactivate();
    SceneManager.LoadScene("Launcher");
}
```

5.17.5 Final

En esta escena se muestra el nombre del ganador del partido junto a los botones de salir del juego o volver al menú de inicio. El nombre del ganador se obtiene de la siguiente manera:

```
public Text winnerText;

void Start()
{
    if(Globals.Instance.winner == 0){
       winnerText.text = "Winner: " + Globals.Instance.nameP1 + "!";
    }
    else{
       winnerText.text = "Winner: " + Globals.Instance.nameP2 + "!";
    }
}
```

En las variables globales se guardan los nombres de los jugadores, introducidos anteriormente y también se guarda el identificador del ganador del partido. De esta manera podemos acceder a su nombre y mostrarlo en pantalla, tal y como se puede observar en la Figura 113.

Figura 113: Escena de final de juego

5.17.6 Partido 1vsIA

Escena de entrenamiento donde se muestra el área de juego y el marcador. Ver Figura 114.

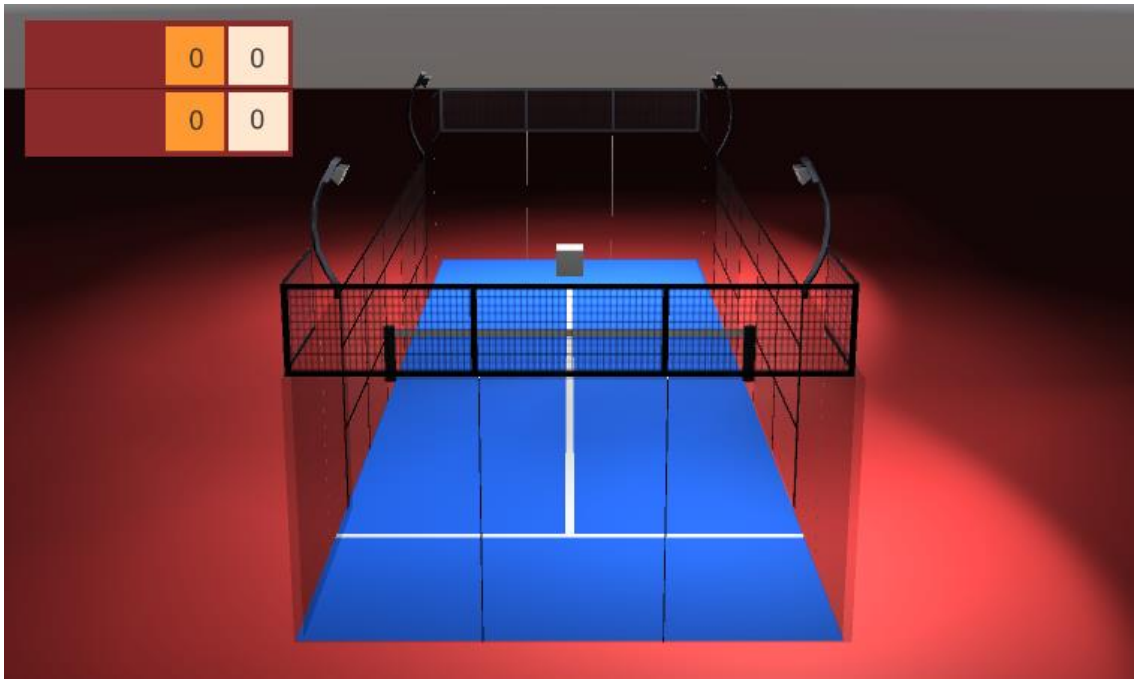

Figura 114: Escena de entrenamiento

5.17.7 Pausa entrenamiento

Igual que el menú de pausa de partido, no se trata de una escena en sí, sino una interfaz que se superpone encima de la escena de entrenamiento. Esta interfaz, al ser idéntica, no necesita más explicación que la ya explicada en el apartado 5.17.4, Pausa partido.

5.17.8 Salir

No se trata de una escena, más bien representa el estado de salir del juego. Para la claridad del diagrama se ha tenido a bien incluirlo.

5.18 Objetivo del juego

Como en cualquier juego deportivo, el objetivo es conseguir mayor puntuación que el rival. En este juego, el objetivo del jugador es ganar el partido al mejor de 3 juegos, es decir, el primero en conseguir 2 juegos, gana.

```
if(Globals.Instance.nGames == 3 && games[idTeam] >= 2){
    Globals.Instance.winner = idTeam;
    SceneManager.LoadScene("Final");
}
```

En el momento de sumar un juego a cualquier jugador, se comprueba si es el tercer juego jugado y si ese jugador ya ha conseguido 2 juegos o más. En ese caso guardaremos el número identificador del jugador en la variable global *winner* e iremos a la escena final.

6. Resultados

En este capítulo se evaluará el nivel de cumplimiento de cada uno de los objetivos marcados en el momento de plantear el proyecto. El objetivo principal del proyecto era crear un videojuego deportivo de pádel respondiendo a la falta de referentes de este género.

Para concretar más, el objetivo principal se subdividió en la siguiente lista de objetivos:

- *Estudiar como simplificar el deporte para transformarlo en un videojuego.*

El pádel cuenta con numerosas reglas complejas que agregan diversión, pero le restan sencillez a la jugabilidad. En este proyecto se han simplificado estas reglas quedándose con las más importantes.

- *Diseñar una jugabilidad entendedora.*

Con la ayuda de los elementos gráficos como la proyección vertical de la pelota o el alcance del jugador, se ha conseguido una jugabilidad intuitiva.

- *Desarrollar el movimiento de los jugadores.*

Es la mecánica básica del juego, sin ésta no sería divertida la experiencia. En el pádel es crucial moverse, pero sobre todo mover al rival para encontrar espacios. En este proyecto se ha hecho un movimiento simple y adecuado, teniendo en cuenta el cambio de orientación debido a la pantalla dividida.

- *Desarrollar el comportamiento de la pelota según el tipo de golpe recibido.*

De los numerosos tipos de golpe que existen, se han elegido los cuatro más básicos para añadir impredecibilidad y variedad al juego.

- *Implementar el sistema de puntuación del marcador.*

El marcador implementado refleja la normativa básica del pádel, quedándose fuera la normativa del desempate.

- *Desarrollar interfaces que aporten claridad al juego.*

Como se ha mencionado antes, para aportar claridad al juego se han hecho elementos gráficos que permiten ubicar la pelota con claridad, así como también se muestra de forma clara el alcance de los jugadores, ver Figura 115.

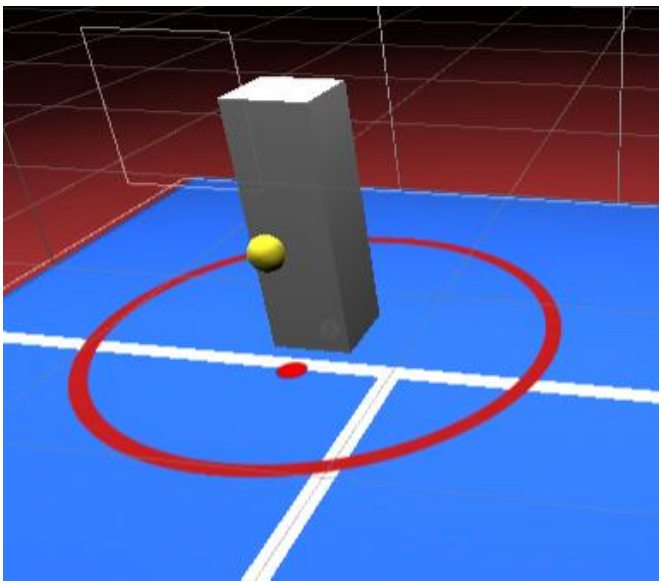

Figura 115: Interfaces de alcance y posición de la pelota

- *Modelar los jugadores y el entorno*.

Aunque la previsión inicial era acabar dos personajes, el cambio en la metodología para pasar a una metodología mejor implicó una inversión de tiempo en el aprendizaje de esta segunda metodología, trastocando así los planes previstos. El cambio en la metodología fue debido a la motivación del autor de esta memoria a querer mejorar en la escultura de figuras orgánicas y se consideró oportuno incluir esta metodología en el proyecto. La principal diferencia entre las dos metodologías es que, en la segunda, se usa un pincel para influir en la posición de los vértices, consiguiendo así, resultados más suavizados y curvados, ver Figura 116. El cambio de metodología ha servido para mejorar las dotes de escultura digital del autor de este proyecto. La pista de juego ha quedado realista aplicando técnicas adquiridas durante el grado.

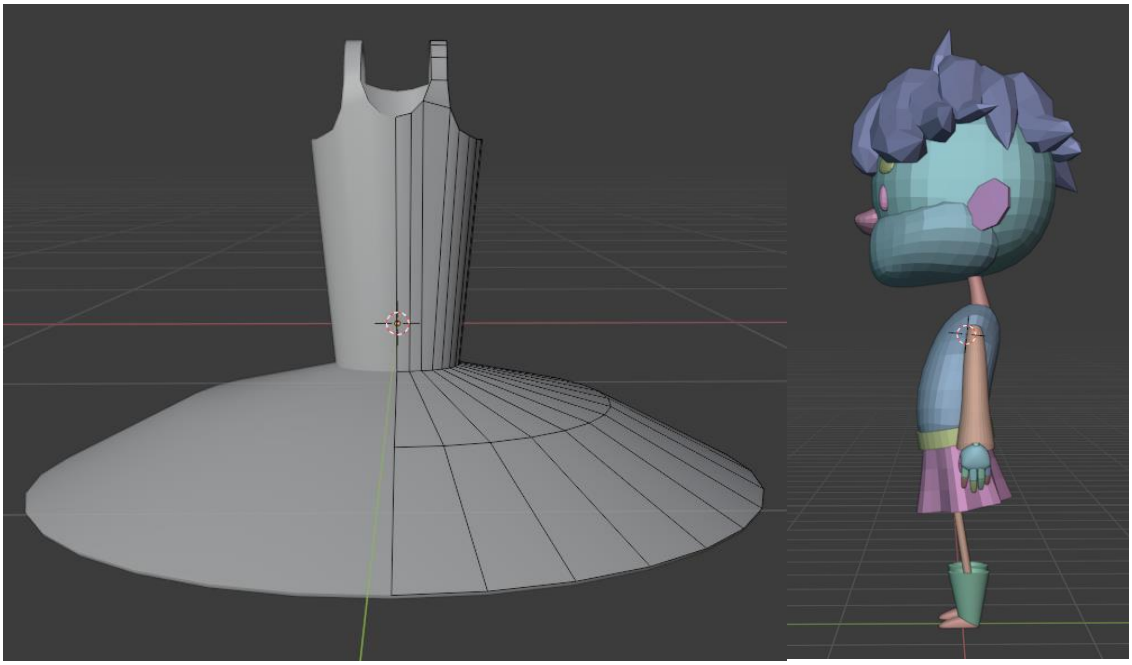

Figura 116: Comparación de topología entre metodologías

- *Crear animaciones para el jugador.*

Se han creado animaciones de dos tipos de golpe, pero se habían planeado más animaciones. Los resultados obtenidos al hacer estas animaciones han estado por debajo de mis expectativas ya que, una vez importadas dentro del juego, no se consiguió que la pala impactara en la pelota.

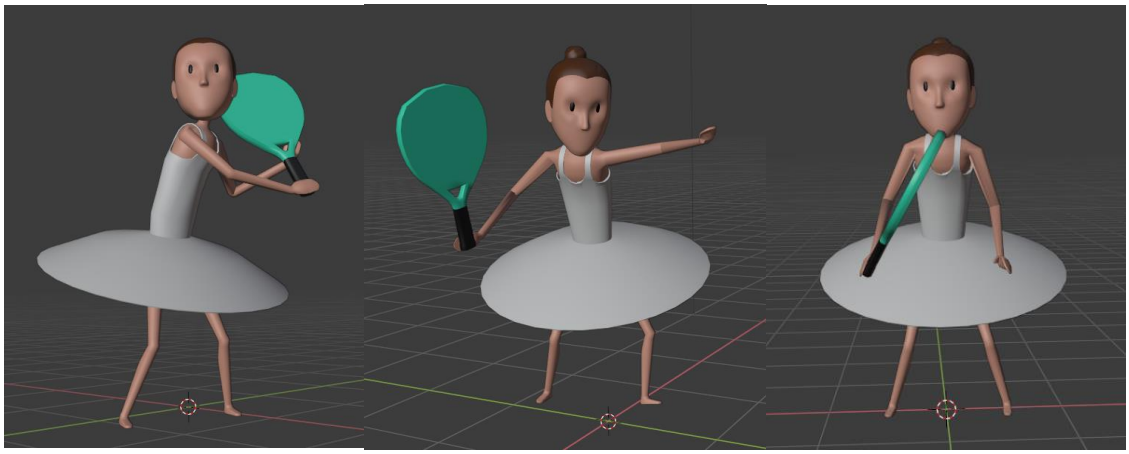

Figura 117: Ejemplos de fotogramas de animaciones

7. Conclusiones

En este proyecto se ha podido poner a prueba los conocimientos adquiridos a lo largo del Grado en Diseño y Desarrollo de Videojuegos y se ha comprobado que aún quedan muchas cosas por aprender y con casi total seguridad, siempre habrá más en este ámbito digital que no para de evolucionar.

Antes de empezar con esta idea de realizar un juego de pádel, se tenía en mente desarrollar un juego de ámbito deportivo, pero aún no se había decidido cual. Un deporte candidato era el frontenis, deporte poco conocido por la gran mayoría del público. La idea de convertir por primera vez en videojuego un deporte era muy tentadora y se pensó qué otros deportes no se habían recreado en el mundo de los videojuegos, llegando a la conclusión que el pádel no había sido recreado aun siendo un deporte en auge. La oportunidad en el mercado y la afición del autor de esta memoria a este deporte hicieron que no se diera más vueltas sobre la temática para este trabajo.

Primeramente, se tenía en mente hacer un juego realista donde se pudiera golpear a la pelota con efectos diferentes, pero rápidamente se corrigió y se optó por hacer un juego simplificando el pádel. Convertir a videojuego un deporte y querer hacerlo realista era inviable teniendo en cuenta el tiempo y los recursos de los que se disponían. Una vez se encontró la manera de simplificar el pádel, se añadieron otros aspectos como las habilidades de los personajes y así añadir contenido al videojuego y ofrecer variaciones en la jugabilidad.

Durante el proceso de desarrollo de este proyecto surgieron problemas en numerosas ocasiones, pero se pudieron resolver mayoritariamente consultando en internet, en foros donde gente plantea sus dudas y otras personas las resuelven, o en la propia documentación oficial de Unity.

Finalmente, hace falta decir que el videojuego final cuenta con los elementos básicos y más importantes de un juego deportivo, y por lo tanto se puede afirmar que los objetivos marcados inicialmente se han cumplido y se han añadido elementos extras, como por ejemplo, las habilidades de los personajes.

8. Trabajo futuro

En este apartado se muestran los elementos que no se han podido llevar a cabo y posibles ampliaciones de este videojuego.

- **Añadir un selector de personaje:** Al disponer de varios personajes, el jugador debe tener la posibilidad de elegir el personaje que más se adecúa a su estilo de juego.
- **Añadir efectos de sonido:** Para conseguir mayor inmersión sería bueno incluir sonidos en los golpes, efectos de público, etc.
- **Refinar las interfaces de usuario:** Diseñar botones e interfaces con un estilo propio.
- **Añadir efectos visuales al activar las habilidades de los personajes:** Las habilidades especiales de los personajes deberían incluir efectos de partículas para lograr un aspecto más espectacular.
- **Añadir tiempo de carga de las habilidades:** Las habilidades requerirán de un tiempo de carga para evitar el abuso de ellas.
- **Implementar el modo 2vs2:** El pádel competitivo se juega en parejas y sería bueno implementar este modo en el juego.
- **Mejorar las animaciones para sincronizarlas con la pelota:** Lograr que, en las animaciones de golpe, la pala toque la pelota en el momento de golpear.
- **Acabar todas las animaciones del golpeo:** Cada tipo de golpe dispondrá de su propia animación.
- **Modelar más escenarios de juego:** Agregar entornos diferentes que influencien el estilo de juego (viento, que la pelota bote menos, etc.)
- **Modelar más personajes y diseñar sus respetivas habilidades:** Existen muchos tipos de jugadores y por ello es necesario cubrir los estilos de cada tipo de jugador, ofreciendo nuevas maneras de jugar.
- **Añadir un tutorial de aprendizaje**: Incluir en el juego un tutorial que enseña los controles del juego y el uso de las diferentes habilidades.

9. Bibliografía

Padel, T. T. (2021, Enero 3). ¿Un videojuego de pádel? Noticias de Pádel – Todo Tu Pádel. https://todotupadel.es/videojuego-de-padel/#.YoJjOKhBxhE

Martin, R. D. (2022, Mayo 24). Reglas del Pádel (Resumen de normas básicas). PADELSTAR. https://padelstar.es/reglas-de-padel/resumen-con-las-reglas-y-normas-basicas-de-padel-perfecto-para-principiantes/

Zona de padel. (2019, Junio 11). El golpe de revés en el pádel, ejecución y mejora. https://www.zonadepadel.es/blog/2013/07/escuela-de-padel-el-golpe-de-reves/

Portillo, D. (2022, Marzo 21). Pádel individual. Cómo se juega y dónde jugar. Padel Nuestro Blog. https://www.padelnuestro.com/blog/padel-individual/

ambientCG - Public Domain Resources for Physically Based Rendering. (2021). Ambient CG. https://ambientcg.com/

10. Manual de instalación y de usuario

10.1 Manual de instalación

En este apartado se explicará paso a paso el proceso de instalación del juego.

- Descargar el archivo comprimido que contiene la aplicación.
- Hacer *click* derecho encima del archivo descargado y seleccionar la opción “Extraer aquí”.
- Entrar en el directorio extraído y hacer doble *click* en el ejecutable.

10.2 Manual de usuario

Los controles para jugar son los mostrados en la Figura 118.

Figura 118: Controles de juego