E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Aplicaciones de la realidad virtual en la manipulación y animación de modelos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor: Youssef Benbelkheir Núñez

Director: Oscar Ardaiz Villanueva

Pamplona, 8 de junio de 2023

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

AGRADECIMIENTOS

A mi familia por haberme apoyado a lo largo de mi carrera, celebrando los momentos buenos y apoyándome en los difíciles.

A la universidad por cederme el material necesario para la realización de este trabajo.

A Oscar Ardaiz por darme muy buenas ideas y ayudarme a obtener los recursos necesarios para poder seguir con el trabajo en los momentos en los que no podía trabajar desde casa.

A mis compañeros de carrera y en especial a Adrián Guiral por los buenos momentos en la carrera y el apoyo en los difíciles.

A todos los profesores del diploma de Videojuegos y Realidad Virtual de la UPNA que me han enseñado mucho de la industria y de cómo desarrollar videojuegos.

Gracias a todos vosotros he conseguido concluir un proyecto del que me siento orgulloso.

Contenido

AGRADECIMIENTOS 2
RESUMEN 6
PALABRAS CLAVE 6
MOTIVACIÓN 6
INTRODUCCIÓN 7
OBJETIVOS 7
DESCRIPCIÓN GENERAL DEL PROYECTO 8
ESTADO DEL ARTE/CONTEXTO 9
HISTORIA ANIMACIÓN POR COMPUTADORA 9
REALIDAD VIRTUAL Y ANIMACIONES 11
TRATAMIENTOS FÍSICOS Y REALIDAD VIRTUAL 13
ENTORNO DE DESARROLLO 13
HARDWARE 13
META QUEST 2 14
ORDENADOR 14
SOFTWARE 15
UNITY 15
TAREAS REALIZADAS 16
INVESTIGACIÓN SOBRE XR 16
CREACIÓN DE UN MINIJUEGO VR 16
GRABACIÓN DE ANIMACIONES EN UN SISTEMA PROPIO 18
CONTROL DE UN PERSONAJE HUMANOIDE MEDIANTE ANIMACIONES PREDISEÑADAS 19
INVESTIGACIÓN SOBRE CINEMÁTICA INVERSA Y OTRAS TÉCNICAS 20
PRUEBAS CON PERSONAJE RAGDOLL 22
INVESTIGACIÓN SOBRE LA API DE ANIMACIONES DE UNITY 22
CONTROL DE HUMANOIDE CREANDO ANIMACIONES PERSONALIZADAS 23
MEJORAS DE INTERFAZ DE USUARIO 23
IMPLEMENTACIÓN 24
INTRODUCCIÓN 24
OBTENER DATOS DEL INPUT 26
MOVIMIENTO 27
CAPTURA DE ANIMACIÓN 28
AGARRE DE OBJETOS 30
FEEDBACK PARA EL USUARIO 32
PROBLEMAS DURANTE EL DESARROLLO 37
RESULTADOS 38
POSIBLES MEJORAS FUTURAS 43
CONCLUSIONES 44
MANUAL DE USO 44

OPERABILIDAD 46
REFERENCIAS 48
FIGURAS.....51

RESUMEN

En estos últimos años se han empezado a explorar las posibilidades que nos puede traer la Realidad Virtual. En un campo en desarrollo, hoy en día se siguen buscando aplicaciones y desarrollando herramientas en Realidad Virtual que mejorarán la eficiencia de muchos trabajos que antes suponían una complejidad muy alta.

En este trabajo se han desarrollado herramientas para el control de modelos tridimensionales humanoides mediante las Meta Quest 2. Estas herramientas permiten al usuario tener un control mediante los mandos de la Meta Quest 2 de las articulaciones de los modelos 3D, permitiendo realizar animaciones de ellos, distintas poses, controlarlos mediante una serie de animaciones prediseñadas para su movimiento.

PALABRAS CLAVE

-Realidad Virtual

-Modelo 3D

-Meta Quest 2

-Animaciones

-Ragdoll

-AnimationClip

-AnimationCurve

MOTIVACIÓN

La motivación para crear este proyecto fue la necesidad de la industria de la Realidad Virtual de explorar las nuevas aplicaciones posibles de estas herramientas en relación con el manejo de modelos 3D. Estas aplicaciones potenciarán el uso de la Realidad Virtual en nuevos entornos.

Además de esto tenía mucho interés en adentrarme en un mundo mucho más técnico en el desarrollo de aplicaciones en Unity y este proyecto me gratificó porque me permitió mejorar como desarrollador y enfrentarme a algo nuevo.

INTRODUCCIÓN

En estos últimos años se ha extendido el uso de las gafas de Realidad Virtual, dispositivos que permiten introducirnos en el mundo virtual e interactuar con él. Por esto mismo es muy importante encontrar las posibles aplicaciones de estos nuevos dispositivos, desarrollar estas aplicaciones y ver hasta dónde podemos llegar con la Realidad Virtual. Las aplicaciones de realidad virtual pueden acelerar proyectos, facilitar trabajos que antes eran mucho más complejos y llevar la interacción remota a otro nivel.

La industria de las animaciones es una de las industrias más importantes a la hora de desarrollar obras audiovisuales como películas o videojuegos. Es una industria de mucha complejidad en la que es necesario disponer de muchos conocimientos psicomotores y anatómicos. La colaboración es muy importante en este sector, ya que en las grandes producciones los equipos de animaciones están compuestos por muchas personas y es de vital importancia crear herramientas que permitan automatizar el trabajo lo máximo posible.

A nivel formativo también pueden tener un rol muy importante las aplicaciones de realidad virtual ya que permiten visualizar elementos que antes solo podían ser visualizados por la red a través de una pantalla bidimensional.

Por todo esto, en este trabajo se desarrollarán herramientas que puedan permitir tanto a nivel formativo como industrial el uso de las gafas de realidad virtual. Estas herramientas permitirán acelerar el proceso de animación y enseñanza.

OBJETIVOS

Objetivos iniciales:

1. Poder controlar un modelo humanoide 3D a través de las Meta Quest 2 mediante una serie de animaciones prediseñadas que le permita moverse por un entorno.
2. Almacenar en una lista el orden en el que se ejecutan distintas acciones.
3. Reproducir la lista de acciones en su orden.

Objetivos finales:

1. Mediante los controles de las Meta Quest 2 ser capaz de controlar un modelo 3D humanoide de forma que podamos mover sus articulaciones sin que la estructura del modelo 3D se rompa. Es decir, intentando cumplir las limitaciones físicas de las articulaciones de nuestro modelo 3D.
2. Automatizar el proceso para que se pueda adaptar a cualquier modelo 3D que cumpla unos criterios de estructura básicos.
3. Poder capturar todas las rotaciones de cada extremidad en un momento determinado y almacenarlo para su futura lectura. Interactuando con el scripting API de Unity.
4. Poder reproducir una lista de todas las rotaciones de cada extremidad permitiéndonos reproducir animaciones completas en las que se puedan mover múltiples articulaciones simultáneamente.
5. Mejorar la interfaz del usuario para que el usuario sea capaz de mantener el control de lo que está haciendo en todo momento.

DESCRIPCIÓN GENERAL DEL PROYECTO

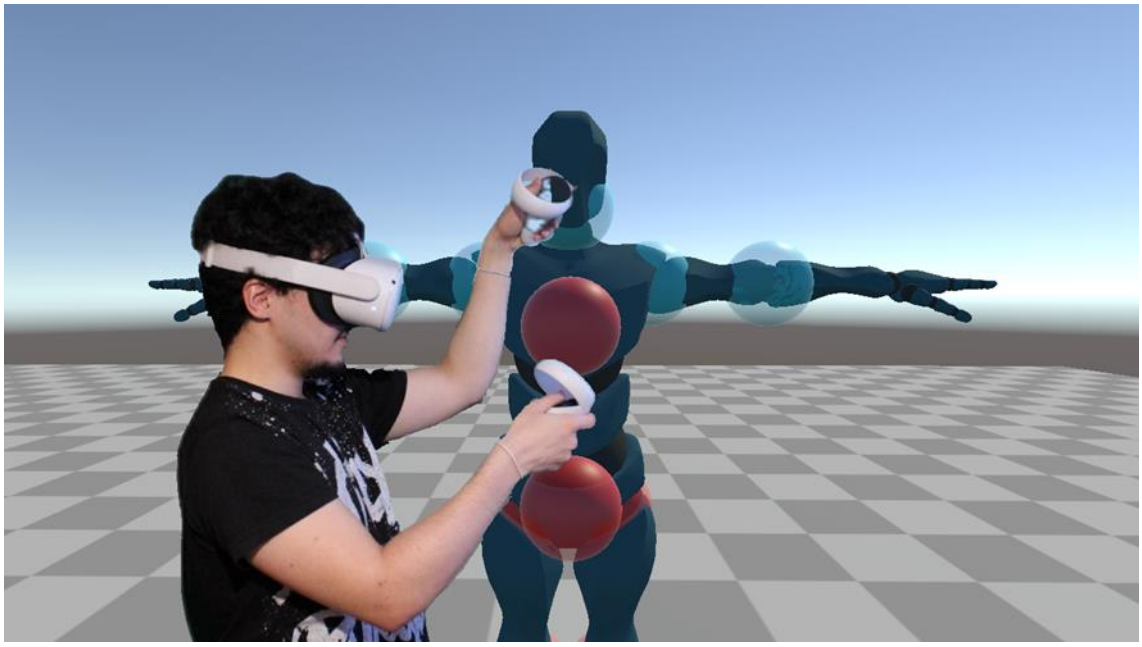

Figura 1 Representación de usuario controlando modelo 3D

Este proyecto consiste en la investigación y desarrollo de posibles herramientas para el control de modelos 3D humanoides a través de las tecnologías de Realidad Virtual.

Las herramientas desarrolladas permiten al usuario manipular y controlar mediante los mandos de Realidad virtual articulación por articulación su posición lo que le da la posibilidad de colocarlo en las poses deseadas. Estas poses serán usadas para crear fotogramas que compondrán una animación. La herramienta creada soporta la creación de estás animaciones.

Además de estas características también han sido añadidas algunas herramientas visuales que permiten al usuario tener un mayor control de lo que está realizando. Como la previsualización de dos fotogramas mientras se está trabajando con el maniquí o los objetos 3D translucidos de color azul o rojo que están colocados en las articulaciones del modelo 3D. En la anterior figura representados por las esferas azules, le indican al usuario si la articulación está bloqueada o no. Una articulación bloqueada tiene su esfera correspondiente de color rojo e impide que el Usuario mueva su articulación.

También es posible a través del agarre libre del maniquí desplazarlo como si de un muñeco de trapo se tratase. A este comportamiento se le llama “RagDoll”.

Las aplicaciones de estas herramientas pueden ser formativas, para instruir sobre fisioterapia y movimiento del cuerpo humano o para la creación de animaciones. Es por esto por lo que se ha de poner en contexto la industria de animaciones y de cómo herramientas como esta pueden suponer una ventaja en el desarrollo de estos proyectos.

ESTADO DEL ARTE/CONTEXTO

HISTORIA ANIMACIÓN POR COMPUTADORA

El desarrollo de gráficos por computadora comienza a finales de 1940 y principios de 1950 pero no fue hasta mediados de 1960 que se comenzó a utilizar de manera artística como puede ser en 1972 cuando se desarrolló una de las primeras animaciones en 3D que animaba el movimiento facial de una cara y el movimiento de una mano [1]:

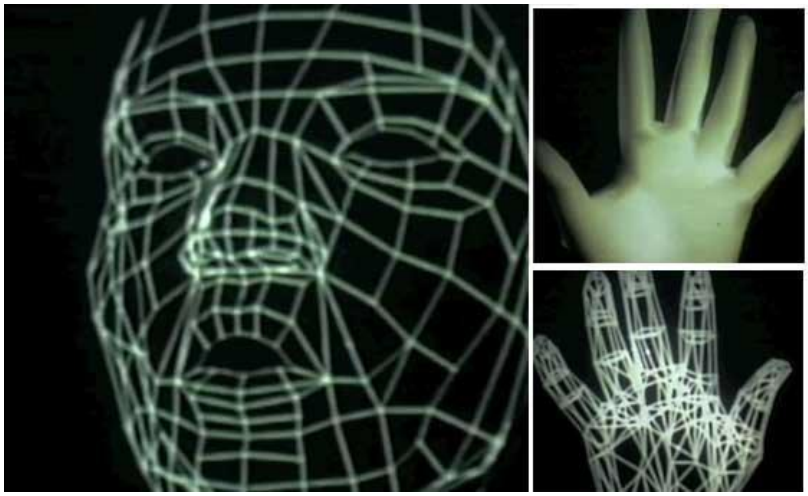

Figura 2 Primera animación 3D [2]

En 1995 se estrenó el primer largometraje por computadora, “Toy Story” que fue un gran éxito y sentó las bases de la animación por computadora.

En la actualidad gracias al avance tecnológico de los ordenadores hacer animaciones y trabajar con modelos 3D se ha convertido en una tarea que puede ser realizada en un ordenador personal. Sin embargo, los programas para realizar estas tareas son extremadamente complejos y requieren de muchos conocimientos sobre ellos.

Maya es un software 3D profesional ampliamente utilizado actualmente en la industria de animaciones para el modelado, animación y renderización de modelos 3D. Su

interfaz gráfica y su kit de desarrollo lo convierte en un producto muy completo para su uso en el desarrollo de videojuegos, material para el cine o televisión.[3]

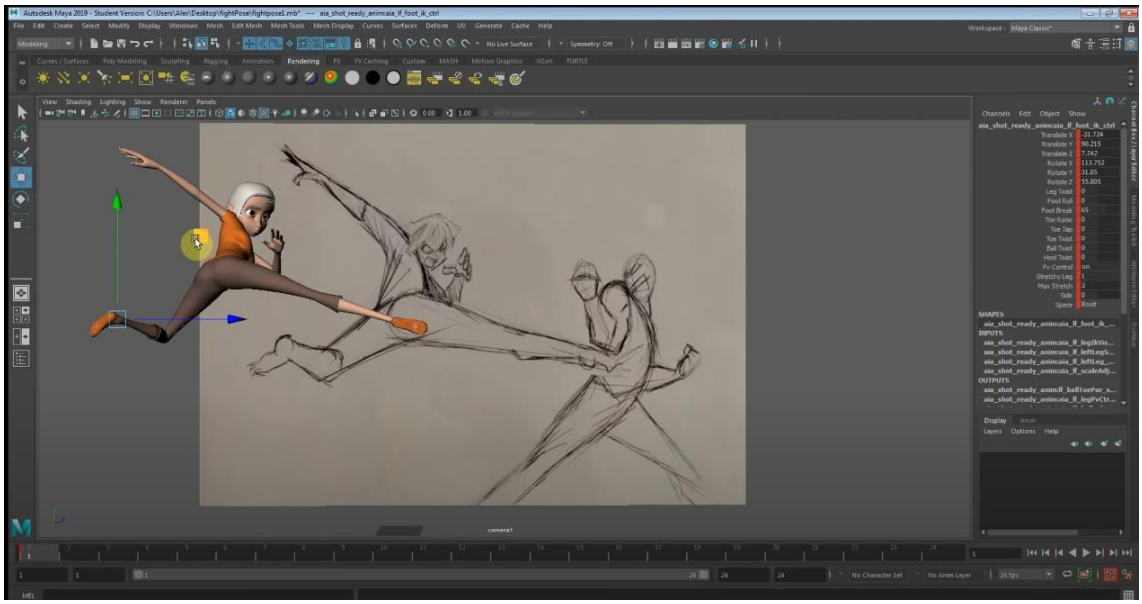

Figura 3 Creación de pose dinámica a través de un boceto [4]

En la figura 3 podemos ver cómo a través de un boceto en 2 dimensiones se está colocando un modelo 3D para que imite esta pose en la herramienta de Maya. El trabajo de colocación ha de ser hecho articulación por articulación. Controlando a través del ratón su posición espacial y su rotación todo esto a través de una pantalla de ordenador que está en 2 dimensiones. Estas capacidades de manejo de modelos podrían ser mejoradas en un entorno en el que el control sea lo más cercano a lo físico y en 3 dimensiones.

Otra herramienta: Cascadeur[5] que implementa nuevas herramientas para hacer animaciones como este sistema de puntos por el cual podemos establecer que un elemento del modelo 3D lo siga como en el caso de la siguiente figura que la mano derecha sigue esta línea de puntos mientras el modelo 3D se desplaza andando.

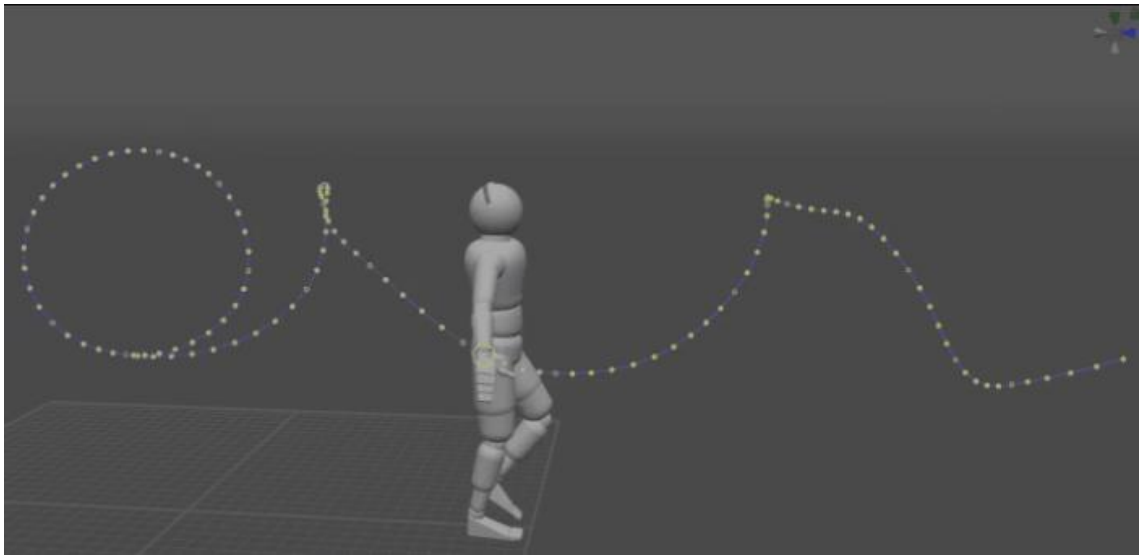

Figura 4 Animación de modelo 3D con restricciones a elementos [6]

REALIDAD VIRTUAL Y ANIMACIONES

La tecnología de la realidad virtual permite a los usuarios sumergirse en el mundo virtual y poder interactuar con él. Esta tecnología tiene mucho potencial a nivel educativo ya que nos permite crear situaciones de entrenamiento de una forma mucho más inmersiva y eficaz pero el uso industrial de estas herramientas todavía no está maduro.

Se abre una ventana de oportunidad para investigar y desarrollar las aplicaciones industriales de esta tecnología y de cómo podrían ser estás aplicaciones en la industria de animaciones. Entre las posibles ventajas que podría suponer en la industria de animaciones:

- Formativo: Enseñando a nuevos usuarios a animar modelos 3D.
- Aumento de calidad: Ya que los desarrolladores podrían visualizar las animaciones 3D de una forma muy inmersiva gracias a la Realidad Virtual.
- Aumento de productividad: Los desarrolladores tendrán más facilidades para realizar animaciones 3D de una manera más rápida.

Cabe a destacar este paper [7] que a través de un prototipo de un proyecto de investigación muestra una técnica de control de modelos 3D a través de los mandos de realidad virtual. El usuario a través de los mandos dibuja la pose que desea y el modelo 3D la imita. Esto mejora la capacidad del usuario para colocar el modelo 3D pero

realmente no se está interactuando con este modelo 3D directamente, sino que estás usando un boceto creado para que sea imitado por el modelo 3D.

Figura 5 Creación de pose a través del boceto de un usuario [7]

Un app VR para animar es Quill, en este video animan un batería [8] , pero no usan IK ni un sistema RagDoll, simplemente mueven unos “trazos” que representan brazos o piernas. Este sistema entonces no tiene la capacidad de realizar estás animaciones para modelos 3D complejos simplemente para aquellos generados con trazos en 3D.

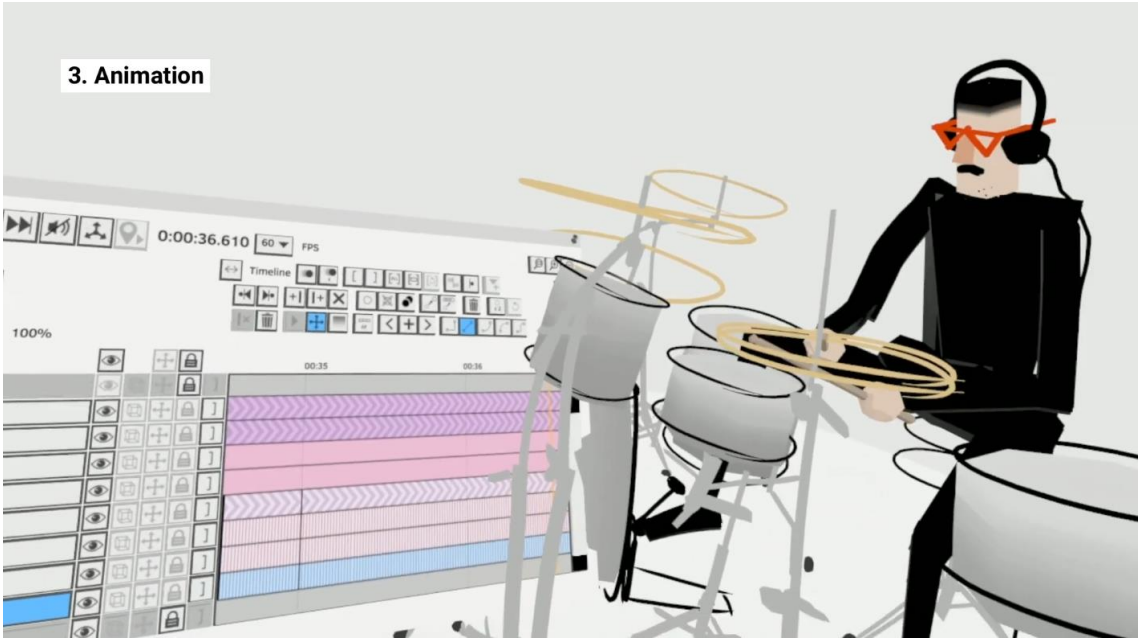

Figura 6 Animación de batería a través de Quill [8]

TRATAMIENTOS FÍSICOS Y REALIDAD VIRTUAL

Las aplicaciones de la operación de modelos 3D humanoides en realidad virtual va más allá de su aplicación en la creación de animaciones. La realidad virtual también puede ser una herramienta formativa para las rehabilitaciones o los entrenamientos deportivos.

La creación de simples animaciones a través de poses puede ser un apoyo a la formación del personal médico para la realización de terapias físicas sobre pacientes, también puede ayudar a deportistas a mejorar sus técnicas físicas a la hora de competir en el deporte.

ENTORNO DE DESARROLLO

HARDWARE

META QUEST 2

Las Meta Quest 2 son unas gafas de Realidad Virtual desarrolladas por “Reality Labs”. Tienen un entorno Android y permiten ejecutar aplicaciones de Realidad Virtual y Realidad Mixta. [9]

Meta Quest 2:

Figura 7 VR Headset Meta Quest 2 [10]

Figura 8 VR Controllers [10]

Especificaciones técnicas [10]

- Hardware: Pc opcional. Sistema propio.

- Seguimiento: Sin sensores externos.
- Controladores: Controladores Touch ergonómicos.
- Óptica: Pantalla LCD.
- 1832x1920 de resolución en cada ojo.
- Frecuencia de actualización de hasta 90 Hz.
- Almacenamiento 128 GB.

ORDENADOR

A pesar de que el proyecto se puede ejecutar y testear en las META QUEST 2 hacerlo de esta forma supondría muchísimo más tiempo en realizar builds y testear del necesario. Es por esto por lo que fue necesario utilizar un ordenador con buenas características técnicas para poder testearlo a través de OCULUS LINK que te permite ejecutar aplicaciones VR del pc sin tener que instalarlas en las Meta Quest 2 pero usando el procesamiento del ordenador.

COMPONENTES USADOS:

- Procesador: AMD Ryzen 5 5600X 3.7GHz.
- Tarjeta Gráfica: Gigabyte GeForce RTX 3060 GAMING OC 12GB GDDR6 Rev 2.0.
- Memoria RAM: Corsair Vengeance LPX DDR4 3200 PC4-25600 16GB 2X8GB CL16.
- Almacenamiento: Crucial BX500 SSD 1TB 3D NAND SATA3.
- Placa: Gigabyte B550 AORUS ELITE V2.
- Fuente de alimentación: Mars Gaming MPB1000 1000W 80 Plus Gold.

SOFTWARE

UNITY

Figura 9 Logo Unity [11]

Unity es un motor gráfico de licencia desarrollado por Unity Technologies y lanzado inicialmente en 2005. Este motor ha sido desarrollado en C++ y C#. Es un motor gráfico muy extendido en la industria de los videojuegos y ha sido utilizado en la creación de múltiples juegos exitosos [12].

Uno de los motivos por lo que este motor gráfico está tan extendido es los planes de licencias que tiene que permiten a desarrolladores pequeños usarlo de forma gratuita. Obteniendo un soporte constante de actualizaciones me permitió trabajar con la seguridad de tener librerías y plugins para el desarrollo de aplicaciones orientadas a la realidad virtual.

Este proyecto se ha desarrollado en este motor gracias al soporte gratuito, las herramientas que posee para la realidad virtual y lo extendido que está en la industria. Unity posee la capacidad de procesar el código C# por lo que este trabajo ha sido desarrollado en ese lenguaje.

TAREAS REALIZADAS

En este apartado vamos a analizar las tareas desarrolladas para la investigación y pruebas preliminares al desarrollo final de la herramienta. Algunas de estas tareas también son herramientas en sí mismas que pueden ser expandidas y desarrolladas de forma más detallada para su uso en la Realidad Virtual y otras son investigaciones sobre componentes que puedan ser usados finalmente.

INVESTIGACIÓN SOBRE XR

XR es la abreviatura de un conjunto de nuevas tecnologías creadas para cambiar la forma con la que interactuamos con el mundo digital [13].

Tecnologías como:

- Realidad Virtual: Una realidad paralela que podemos experimentar mediante el uso de las gafas Meta Quest 2. Permite sumergirte en el mundo virtual e interactuar en este.
- Realidad Aumentada: Permite plasmar el mundo digital por encima del mundo real. Como podría ser ver al mismo tiempo la sala física en la que te encuentras y un objeto 3D en este espacio.

- Realidad Mixta: Mezcla lo mejor de las dos anteriores. permitiéndote interactuar física y digitalmente con diferentes objetos o entornos.

Investigué la API de Unity de XR puesto que esta es la que se va a usar en todo el desarrollo del proyecto. Para su uso se añadió al proyecto como plugin Unity.XR y mediante las configuraciones prediseñadas en la configuración del proyecto se dispuso como target el entorno de Oculus.

Este plugin tiene la ventaja de que nos permite seleccionar la plataforma a la que lo queremos desplegar facilitando la compatibilidad de los proyectos entre distintos dispositivos.

CREACIÓN DE UN MINIJUEGO VR

Figura 10 Logo de Beat Saber [14]

Para familiarizarme con el entorno de desarrollo de Realidad Virtual en Unity se me planteó la creación de un minijuego del estilo de “Beat Saber” pero de forma que puedan jugar 2 jugadores.

Beat Saber es un exitoso juego de realidad virtual por el cual te llegan distintas figuras geométricas que debes destruir con un sable atravesándolos en la dirección correcta al ritmo de la música. Fue uno de los primeros juegos en popularizarse en el mundo de la Realidad Virtual [15].

Los dos jugadores poseen dos sables que son controlados por los mandos de Realidad Virtual.

Jugador 1: Invoca cubos cuando atraviesa con el sable la figura deseada de la pizarra.

Jugador 2: Tiene que destruir los cubos con el sable en la dirección adecuada.

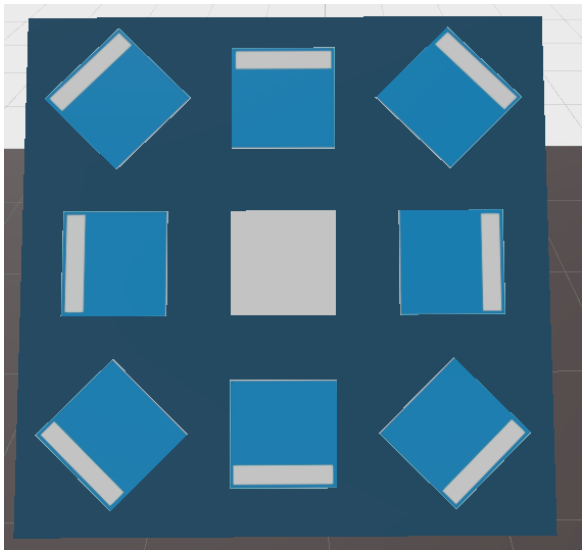

Figura 11 Pizarra de invocación de cubos usada por el jugador 1.

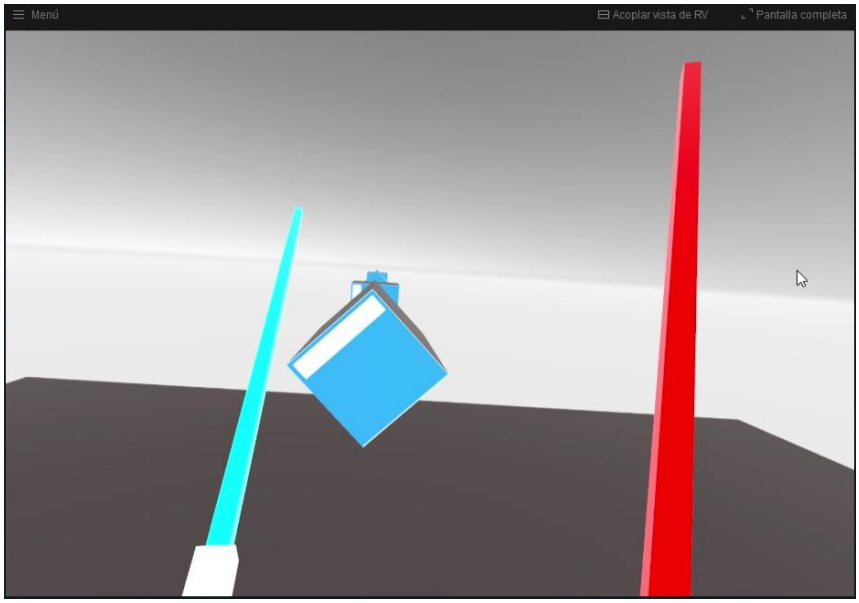

Figura 12 Ejemplo de vista del jugador 2

GRABACIÓN DE ANIMACIONES EN UN SISTEMA PROPIO

Esta tarea busca lograr la creación de un sistema propio para la captura y reproducción de los movimientos de un modelo 3D. Este sistema se desarrolló sin hacer uso del sistema de animaciones de Unity.

Para cada componente del modelo 3D a grabar se le asignará dos listas en una se almacenará una sucesión de Vectores de dimensión 3 de coordenadas y en la otra una sucesión de Cuaternión en el cual se almacenan las rotaciones.

A la hora de la reproducción se recorrerán las listas de componentes en los que estarán todos estos datos recogidos fotograma a fotograma, pero realizando una interpolación lineal que es una operación matemática para obtener el valor intermedio entre dos vectores. De lo que se trata es que en cada fotograma exista movimiento que represente correctamente el movimiento inicial (evitando que al ralentizar la animación se teletransporte el modelo), es por esto por lo que no podemos simplemente ir teletransportando el componente a reproducir fotograma a fotograma, sino que se calcula según los tiempos de ejecución y el reloj de Unity la posición deseada en ese momento.

En el caso de la lista de posiciones se realizará mediante el uso de la función de Unity [16]:

```
Vector3.Lerp(Vector3 a, Vector3 b, float t);
```

En el caso de la lista de Cuaterniones se realizará mediante el uso de la función de Unity [17]:

```
Quaternion.Lerp(Quaternion a, Quaternion b, float t);
```

Para la realización de esta tarea se ha limitado para las pruebas la grabación a 10 segundos.

La tarea fue completada con éxito y el script descrito desarrollado y puesto a prueba. Gracias a esta tarea obtuve un conocimiento mucho mayor de cómo funcionan los sistemas de animaciones.

En las siguientes tareas se planteó la aplicación de sistemas de animaciones, pero controladas mediante Realidad Virtual.

CONTROL DE UN PERSONAJE HUMANOIDE MEDIANTE ANIMACIONES PREDISEÑADAS

El objetivo de esta tarea era desarrollar una interfaz para controlar a un modelo 3D mediante animaciones prediseñadas.

Cómo animaciones prediseñadas se establecieron las siguientes:

- Caminar
- Girar izquierda

- Girar derecha
- Saltar/Levantarse
- Agacharse
- Rotar 180 grados.

Y se creó una cápsula externa. En el momento en el que se interactúe con los mandos de realidad virtual con las esferas blancas se realiza la acción descrita en las siguientes imágenes.

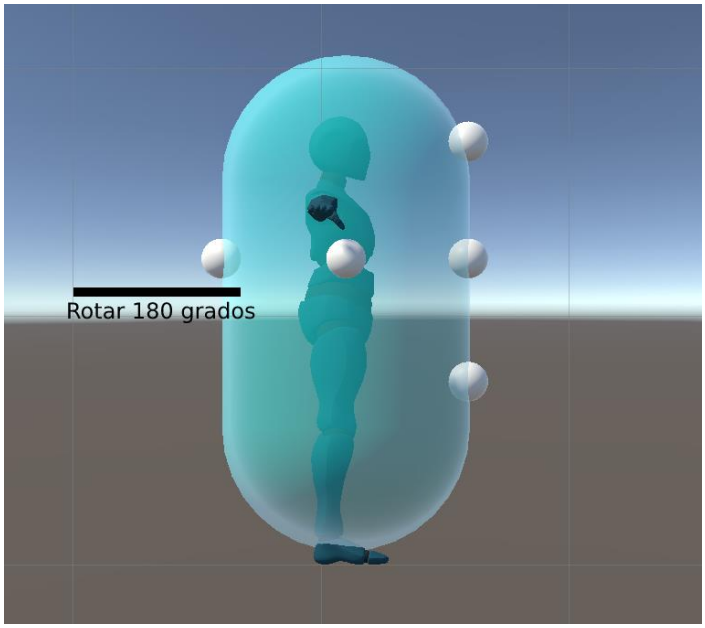

Figura 13 Vista lateral de la cápsula del modelo 3D

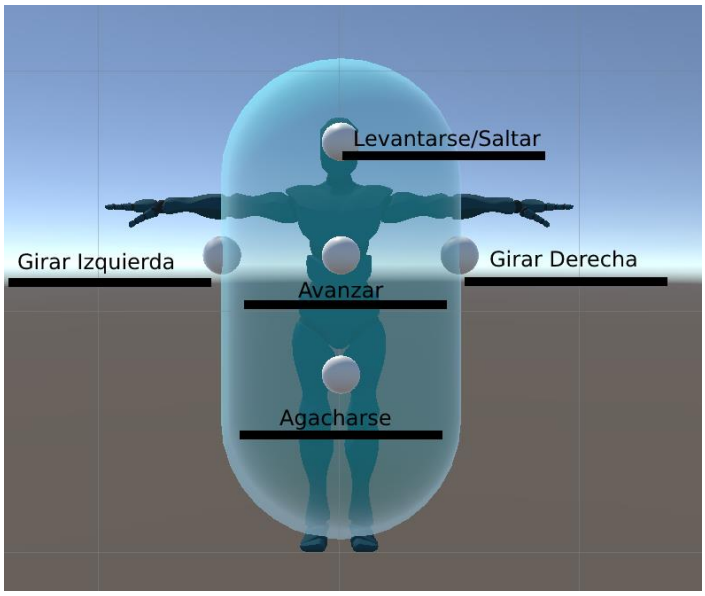

Figura 14 Vista frontal de la cápsula del modelo 3D

Estás acciones podrán ser reproducidas de nuevo en el orden realizado. Está aplicación nos da la capacidad de preparar a estos modelos tridimensionales para que completen un circuito de la forma que queramos.

Con esto ya completamos los objetivos iniciales.

INVESTIGACIÓN SOBRE CINEMÁTICA INVERSA Y OTRAS TÉCNICAS

En esta etapa del proyecto empecé a valorar el incremento del alcance del proyecto y abrir paso a desarrollar nuevas aplicaciones en VR.

Para esto comencé mi labor de investigar que son las cinemáticas inversas y otras técnicas similares.

La Cinemática Inversa es la técnica que permite simular el comportamiento de articulaciones para moverlos a una posición deseada mediante el uso de ecuaciones [18]. Las cinemáticas Inversas son muy importantes en el desarrollo de videojuegos y de animaciones. Muchas empresas han invertido muchos recursos en perfeccionar nuevas técnicas que permitan obtener resultados mucho más fieles a la realidad.

Esta tecnología me permitiría realizar una simulación del movimiento de las extremidades de un modelo 3D. Estos modelos 3D para poder procesar las cinemáticas inversas han de ser dotados por unos “huesos”.

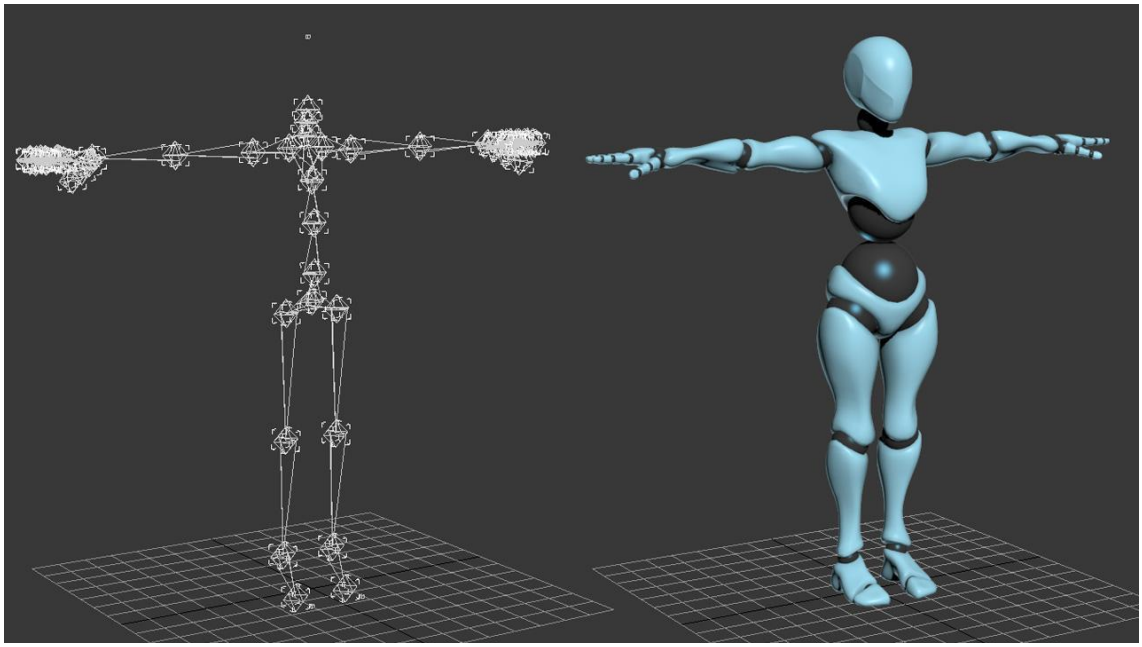

Figura 15 Modelo 3D humanoide a la derecha y la posible representación de los huesos a la izquierda [19]

El número de articulaciones es directamente proporcional al número de cálculos necesarios para la simulación de este comportamiento. Es por esto por lo que existen distintas técnicas y cada uno se adapta mejor a las necesidades de cada proyecto.

Algunos proyectos necesitan tener los cálculos en tiempo real, es decir necesitan esos cálculos para realizar otras actividades que se ejecutan en paralelo. Como por ejemplo en los videojuegos, que necesitan disponer de los cálculos en cada fracción de segundo. En cambio, otros necesitan mucha más precisión y es por esto por lo que no es necesario que se ejecuten en tiempo real. Como por ejemplo las renderizaciones de simulaciones más complejas en el mundo del cine que pueden tardar días en ejecutarse.

Unity tiene integrado en su motor esta técnica y otras de naturaleza similar como la del RAGDOLL la cual es de la que vamos a dar provecho, pero antes de empezar a trabajar con ella era muy importante entender sus bases y comportamiento.

PRUEBAS CON PERSONAJE RAGDOLL

Para integrar un sistema de control de articulaciones a un modelo 3D en Unity podemos crear un personaje Ragdoll. Un muñeco de trapo en su traducción directa al español.

Un Ragdoll es un modelo 3D que se usa mediante simulaciones físicas de elementos unidos por articulaciones para poder recrear comportamientos físicos del mundo real. Si queremos simular el comportamiento de un personaje humanoide en la simulación de explosiones, caídas, una forma de hacerlo es mediante un personaje RagDoll. Este personaje se deja manipular fácilmente por otras fuerzas externas. Así que si creamos un entorno en el que no exista gravedad y sea el usuario el que pueda manejar este RagDoll le estaríamos dando al usuario la capacidad de controlar a este modelo 3D.

A la hora de crear el personaje RagDoll en Unity se deben vincular sus articulaciones al sistema de Unity para que cree los enlaces entre estas articulaciones. También podremos controlar campos como la fuerza requerida para que el elemento sea desplazado, su masa.

Es por esto por lo que se realizaron múltiples pruebas variando estos parámetros para intentar encontrar un punto en el que el movimiento del modelo 3D sea cómodo para el usuario.

Está tarea fue un periodo de lectura estudio y pruebas de la documentación de Unity en relación con las animaciones.

Unity dispone de Animation Curve que es una clase que nos permite crear curvas de animaciones a partir de un array de keyframes. Estas curvas de animaciones están compuestas por keyframes. Las Animation Curve pueden ser representadas visualmente como en la siguiente figura:

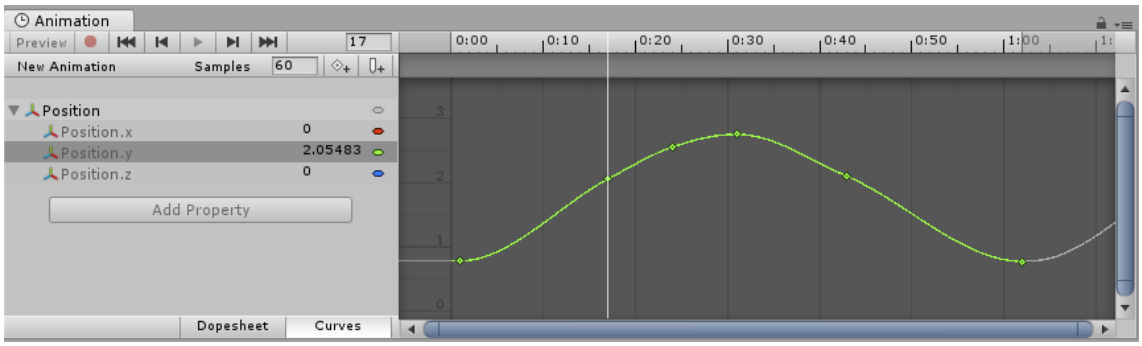

Figura 16 Ejemplo de Animation Curve [20]

Estas curvas también pueden ser modificadas según el usuario para que sean lineales o tengan otros efectos deseados por el desarrollador.

Por ejemplo, si queremos que entre dos keyframes un objeto se mueva constantemente a la misma velocidad la curva será una recta. En cambio, si queremos que este objeto acelere al comienzo y desacelere al final tendrá una forma sigmoide. Puede ser muy interesante pues crear un sistema por el cual seas capaz de definir la forma de la que quieres las curvas a través de realidad virtual.

El motor de Unity para las animaciones trabaja con Cuaterniones por lo que es importante utilizar este sistema de representación y no otro para realizar las operaciones.

CONTROL DE HUMANOIDE CREANDO ANIMACIONES PERSONALIZADAS

Una vez conocida ya la documentación de Unity y realizadas pruebas simples usando estás funciones de la API con éxito se pasó al desarrollo de una nueva herramienta.

Esta nueva herramienta permite al usuario crear animaciones sobre un modelo 3D mientras interactúa con él colocándolo en las posiciones que desee con los mandos de las Meta Quest 2.

MEJORAS DE INTERFAZ DE USUARIO

Estudio y pruebas para mejorar la interfaz del usuario.

- Menú: Varias pruebas sobre un menú. Como el uso de las gafas de realidad virtual es en 3D no se puede poner el menú estático en mitad de la pantalla ya que provoca mareos y anula la inmersión en el mundo virtual. Se planteó la posibilidad de un único menú en una mano, pero quedó descartado al necesitar 4 opciones distintas en el menú. Finalmente se decidió crear dos submenús, uno en cada mano. En una mano se dispondrán los relacionados con el trabajo de la animación como puede ser los botones “CAPTURAR” y “REPRODUCIR” y en la otra mano los botones de “TUMBAR” y “REINICIAR”.
- Sistema fantasma: Para dar información al usuario de la animación que está realizando.
- Controles: Adaptar los controles para que se adapten mejor a la naturaleza de su acción. Como puede ser poner que la acción de agarre se realice en el botón de grip y no en otros.

IMPLEMENTACIÓN

INTRODUCCIÓN

Vamos a analizar la interfaz que disponemos a la hora de desarrollar en Unity que dispone de distintas ventanas que serán usadas a la hora de desarrollar. Como se puede ver en la siguiente figura así será la vista de nuestro proyecto:

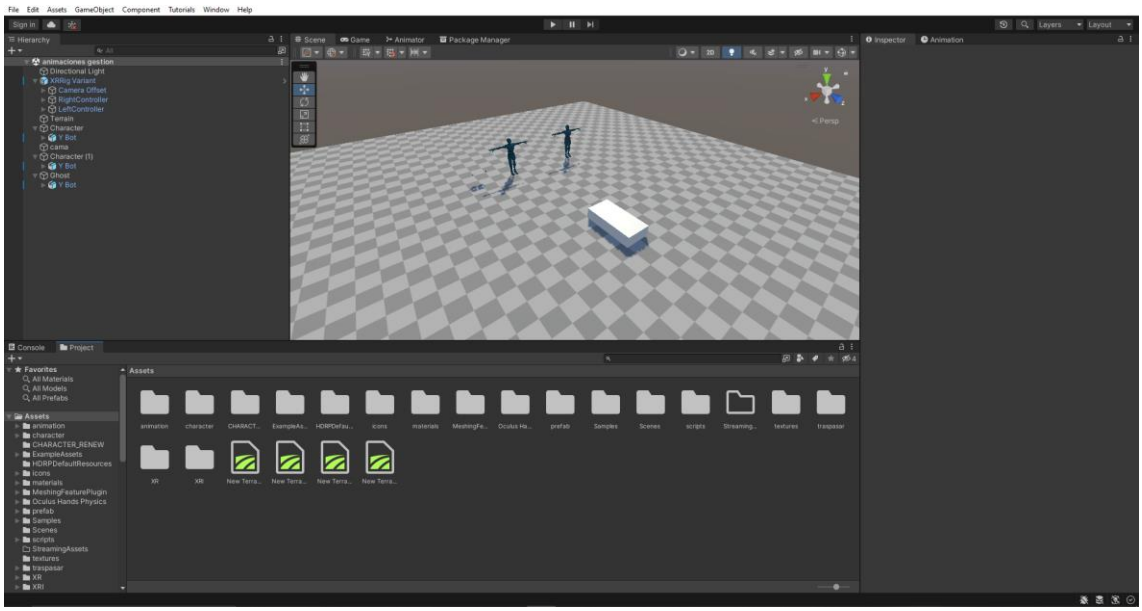

Figura 17 Captura de Unity

Como podemos ver está compuesta por las siguientes ventanas:

- Hierarchy [21]: La ventana de Hierarchy muestra todos los GameObject de la escena (un GameObject puede ser una cámara, un modelo 3D, una luz etc.). Se usa esta ventana para agrupar GameObjects a nuestro gusto, haciendo que unos sean hijos de otros de tal forma que las propiedades de unos dependan de la de los padres. Desde esta ventana también podremos seleccionar los elementos que queramos para visualizarlos en la ventana Scene o ver sus propiedades en el Inspector.

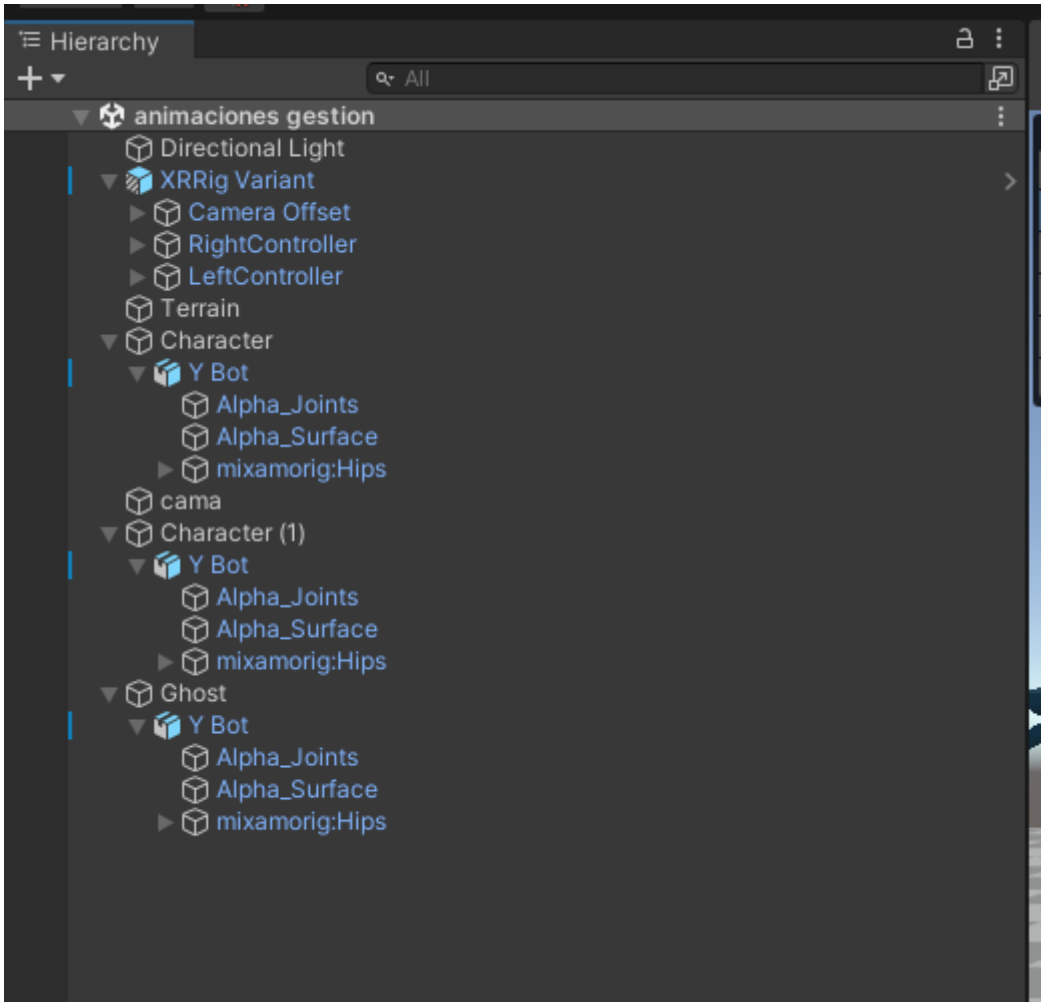

Figura 18 Hierarchy del proyecto

En la jerarquía de nuestro proyecto disponemos de dos Character, uno hará uso del primer sistema de feedback compuesto por cubos y cápsulas y el otro que hará uso del segundo compuesto únicamente por esferas. Será el Character (1) con el que podremos grabar y realizar animaciones con él.

Tenemos también un gameobject que se llama “Ghost” esté se encargará de reproducir a tiempo real las animaciones para dar feedback al usuario. Estará superpuesto en el Character (1).

Un terreno llamado “Terrain” por el cual nos desplazamos, una Directional Light para iluminar la escena y una cama en la que podremos colocar al modelo 3D para simular movimientos desde esa posición horizontal. También dispone de el XRRig Variant con los dos controllers que es la herramienta del plugin de Unity XR qué nos permite colocar al usuario de VR con todos sus componentes en la escena.

- Inspector [22]: Esta ventana se encarga de proporcionar toda la información y propiedades que tiene cada GameObject. Podemos usarla para rellenar parámetros, introducir scripts, manejar materiales...
- Project [23]: Aquí podremos ver todas las carpetas y archivos que disponemos para el uso en el proyecto. En este proyecto se ha usado un modelo 3D importado de mixamo [24], también se han usado sus animaciones para algunas pruebas.
- Console [25]: Nos permite visualizar los errores de compilación, las advertencias e imprime también los mensajes de Debug.
- Animation [26]: La ventana de animación nos permite modificar y visualizar los keyframes de una animación. Ha sido muy importante a la hora de realizar este proyecto para poder visualizar cómo se estaban creando las animaciones después de programarlas.
- Scene [27]: Nos muestra a través de una cámara interactiva ajena al proyecto la vista de nuestra escena permitiéndonos desplazarnos por ella, seleccionar elementos y modificar sus propiedades como posición, escala, rotación…

Para desarrollar en Unity usaremos C#. Incluyendo la librería UnityEngine.XR y llamando a las funciones adecuadas podremos obtener todos los datos necesarios de los controladores de las Meta Quest 2.

Cada clase creada en C# en Unity tiene las siguientes funciones base:

- Start() Se llama al comienzo de ejecución del script, antes incluso de la ejecución del Update().
- Update() Se llama cada fotograma.
- FixedUpdate() Se llama cada fotograma, pero está adaptado a operaciones que usen las físicas del motor.
- Funciones de entrada, estancia y salida en una colisión.
- Etc.

OBTENER DATOS DEL INPUT

Para poder obtener los datos de los Inputs debemos primero especificar de que input se trata mediante la función InputDevices.GetDeviceAtXRNode. Está función tiene como parámetro de entrada un XRNode que será uno de los objetos que tendremos dispuestos en el escenario en el XRRig (la cámara, el controlador izquierdo o el controlador derecho) y devuelve un InputDevice.

Mediante CommonUsages definimos qué datos queremos obtener en la llamada a la función TryGetFeatureValue a la que se le pasará este mismo y se nos devolverá mediante una variable pasada como parámetro la información deseada.

MOVIMIENTO

Creando un XRRig en el proyecto Unity XR es capaz de capturar las gafas de Realidad Virtual, los mandos y el movimiento físico del usuario en la sala en la que se encuentre. Pero para poder controlar el movimiento mediante los joysticks he creado dos clases.

Movimiento en el espacio:

Para movernos por el espacio digital mediante el joystick izquierdo he creado el script "MovementVR.cs".

- En el Start() obtenemos el CharacterController para poder llamar a la función Move más adelante.
- En el Update () Obtenemos el dispositivo mediante la función GetDeviceAtXRNode y el vector del joystick izquierdo llamando a la función del InputDevice TryGetFeatureValue y pasando como parámetro CommonUsages.primary2DAxis. De esta forma obtenemos el Vector de dos dimensiones del movimiento del joystick y se actualizará cada fotograma.
- En FixedUpdate() usamos la función CharacterController.Move y le pasamos la dirección creada a partir del joystick multiplicada por la velocidad y Time.fixedDeltaTime que es el intervalo de segundos entre los fotogramas físicos.

Movimiento de la cámara:

He creado la clase CameraMovement esta clase se encargará del movimiento lateral de la cámara mediante el input del joystick derecho, lo que le da al usuario la capacidad de rotar la cámara sin necesidad de girar físicamente.

- En el Start() obtenemos el CharacterController para poder llamar más adelante a la función transform.Rotate del character.
- En el Update () Obtenemos el dispositivo mediante la función GetDeviceAtXRNode y el vector del joystick derecho llamando a la función del InputDevice TryGetFeatureValue y pasando como parámetro CommonUsages.primary2DAxis. De esta forma obtenemos el Vector de dos dimensiones del movimiento del joystick y se actualizará cada fotograma.
- En el FixedUpdate() del Vector 2 del joystick solo usa la coordenada x. Si esta es mayor que 0 se rota hacia la derecha 2 grados por fotograma. Si es menor que 0 se rota hacia la derecha 2 gratos por fotograma y si es cero no hace nada.

CAPTURA DE ANIMACIÓN

Para la captura de la animación cree un script llamado AnimController.cs este script está encargado de todo el sistema de animaciones y de tener las funciones necesarias para que el usuario pueda controlarlo a su gusto. A la hora de capturar la animación debemos de crear en primera instancia un AnimationClip este objeto será el contenedor de todas las curvas de la animación. En Unity una animación se reproduce siguiendo el recorrido de una curva sobre una propiedad. Es por esto por lo que para poder crear una animación debemos de establecer curvas que a su vez contendrán los keyframes de cada propiedad individual de cada elemento.

- Start() Tenemos que obtener todos los hijos del elemento que queremos capturar. Puesto que para cada modelo 3D que queramos grabar necesitamos capturar individualmente cada elemento que lo compone. Usé la función this.GetComponentsInChildren<Transform>() para obtener el array de hijos del elemento.

  Recorremos mediante un bucle ese array y por cada array le añadimos 4 AnimationCurve que es la curva que definirá la propiedad a nuestro array de curvas que contiene todas las curvas del elemento a grabar. Se añaden 4 puesto que lo que deseamos capturar son las propiedades de la rotación que están especificadas en cuaternión. Cada cuaternión dispone de 4 componentes x, y, z, w. Y es usado en Unity para trabajar con las rotaciones.

- CaptureFrame() se llama cada vez que se captura un fotograma del elemento a grabar lo que hace es recorrer el array de curvas y le añade una “key” usando la función AddKey. A esta función se le pasa un keyframe que es una clave que contiene el tiempo y el valor de la propiedad. Usaremos 3 segundos como separación entre keyframes.

```
2 referencias
public void CaptureFrame() {
    int z = 0;
    for (int i = 0; i < elementChildren.Length; i+=1)
    {
        curveList[z].AddKey(new Keyframe(time, elementChildren[i].localRotation.x));
        curveList[z+1].AddKey(new Keyframe(time, elementChildren[i].localRotation.y));
        curveList[z+2].AddKey(new Keyframe(time, elementChildren[i].localRotation.z));
        curveList[z+3].AddKey(new Keyframe(time, elementChildren[i].localRotation.w));
        z = z + 4;
    }
    time = time + 3;
}
```

Figura 19 Función CaptureFrame

- GetGameObjectPath(Transform obj) Esta función fue creada para resolver una necesidad del sistema de animaciones de Unity que la documentación no cubría. Es una función recursiva que obtiene toda la ruta de un objeto. Es necesaria puesto que a la hora de procesar las animaciones Unity necesita saber toda la ruta del objeto, pero no existe ninguna función con esas características en el sistema de Unity. Para obtener la ruta debemos de obtener una estructura similar a esta "nombrePadre/nombreElemento" pero claro el nombrePadre también está compuesto por la misma estructura. Es por esto por lo que encontré cómodo desarrollar está función de forma recursiva.

```
2 referencias
public static string GetGameObjectPath(Transform obj)
{
    if (obj.parent != null)
    {
        return GetGameObjectPath(obj.parent) + "/" + obj.name;
    }
    return obj.name;
}
```

Figura 20 Función GetGameObjectPath

- DeleteRootString simplemente se encarga de eliminar la raíz principal del objeto porque en el sistema de animaciones de Unity para la raíz no es necesario introducir su nombre.
- Reproduce() será llamado por el usuario cuando quiera reproducir la animación creada. Establece las curvas en el AnimationClip y lo reproduce.

```
2 referencias
public void Reproduce()
{
    for (int i = 0; i < elementChildren.Length; i += 1)
    {
        string elementName = GetGameObjectPath(elementChildren[i]);
        elementName = DeleteRootString(this.gameObject.name, elementName);
        clip.SetCurve(elementName, typeof(Transform), "localRotation.x", curveList[i*4]);
        clip.SetCurve(elementName, typeof(Transform), "localRotation.y", curveList[i*4 + 1]);
        clip.SetCurve(elementName, typeof(Transform), "localRotation.z", curveList[i*4 + 2]);
        clip.SetCurve(elementName, typeof(Transform), "localRotation.w", curveList[i*4 + 3]);
    }
    Animation anim = GetComponent<Animation>();
    anim.enabled = true;
    anim.AddClip(clip, clip.name);
    anim.clip = clip;
    anim.Play(clip.name);
}
```

Figura 21 Función Reproduce

AGARRE DE OBJETOS

Para desarrollar el sistema de agarre lo haremos de la forma más simple posible.

El script que se encargará de manejar todos los comportamientos con relación al agarre es ObjectGrabber.cs.

Crearemos un "Sphere Collider" en cada controlador y lo activaremos como trigger. Es decir, que active las funciones OnTrigger en las interacciones con otros objetos que también dispongan de un Collider. Las articulaciones del Ragdoll dispondrán de esos Colliders así como de un RigidBody que es uno de los requisitos necesarios para que estas llamadas funcionen. Si uno de los dos elementos que interaccionan no tiene un RigidBody aunque los dos tengan colliders no se hará la llamada.

En el momento en el que un controlador del usuario y una articulación del modelo 3D ocupen el mismo espacio, se llamará a la función OnTriggerStay por cada fotograma en la que se mantenga esta condición.

Hay varias condiciones que durante el desarrollo se fueron añadiendo para facilitar la interacción en el agarre.

1. Los colliders de los controladores se hicieron mucho más pequeños para evitar comportamientos anómalos que se daban al colisionar con dos articulaciones simultáneamente.
2. Aun así, si se está colisionando con dos objetos a la vez ignora el primero con el que colisionó.
3. Añadiremos un estado "bloqueado" a las articulaciones que nos determinará si una articulación está bloqueada o no. Si la articulación está bloqueada no podemos agarrarla con los controladores y se quedará estática en el espacio. Para conseguirlo se ha de activar el bloqueo de posiciones y rotaciones en el componente del Rigidbody del elemento.

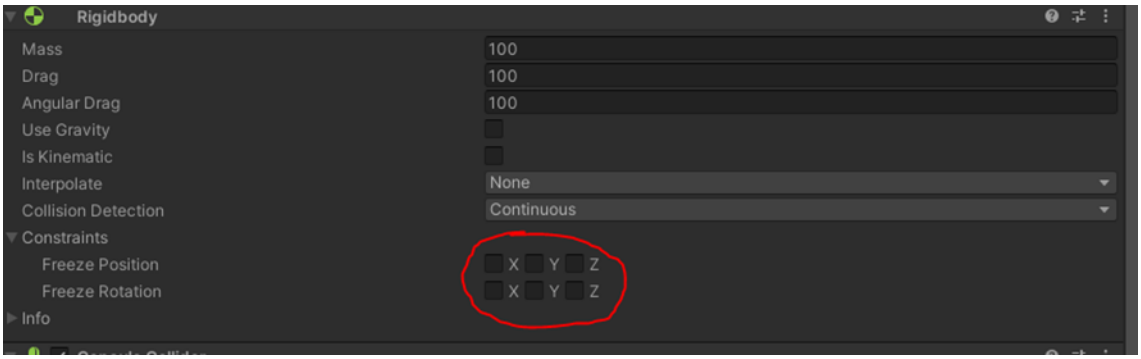

Figura 22 Rigidbody de un elemento en el que se señalan los componentes de bloqueo

Esto es muy beneficioso para el usuario para poder colocar articulación por articulación en el lugar correcto. Esta nueva funcionalidad dio muchos problemas con el sistema de RagDoll puesto que hacía que se daban varias condiciones en las que el modelo 3D se movía de forma muy errática o directamente se rompía.

Este movimiento errático se estudió para determinar su causa y se llegó a la conclusión mediante pruebas que cuando se bloqueaba un elemento intermedio de la jerarquía y se intentaba mover un elemento superior a esta el modelo 3D se comenzaba a mover de forma errática.

Figura 23 Jerarquía de nuestro modelo 3D en Unity

Por ejemplo, cuando bloqueamos el mixamoRig:LeftLeg e intentamos mover mixamorig:Hips o mixamorig:LeftUpLeg.Para solucionarlo cree la siguiente funcionalidad.

4. Bloquear una articulación bloquea todas las articulaciones padres de estas dentro de la jerarquía de Unity. Esto evitaba que se dieran las condiciones necesarias para que estos movimientos erráticos ocurran.

1. Visualización del estado bloqueado:
   Antes he explicado cómo era el sistema de agarre y he explicado del estado "bloqueado" en las articulaciones del modelo. Ahora voy a explicar cómo he implementado un sistema por el cual el usuario es capaz de ver visualmente mediante el set de gafas de realidad virtual si una articulación está bloqueada o no.

   La primera orientación fue cambiar el color de cada articulación según un cambio directo al modelo 3D. Pero esta opción quedó directamente descartada puesto que el "Skinned Mesh Renderer" que es la encargada de darle color al modelo 3D era indivisible por lo que no se podía cambiar el color a unas partes sí y a otras no.
   La Segunda orientación a la solución de este problema fue la de usar elementos primitivos como capsulas, esferas, y cuboides y que mediante su cambio de color mostrará su estado. Que casualmente son los mismos tipos de elementos que usamos para que existan las colisiones. Pero no existe ninguna forma dentro del motor de Unity para mostrarlos al jugador. Es por esto por lo que tuve que implementar mi propio sistema.

   Un sistema que analiza todos los elementos del modelo y dependiendo del tipo de collider tengan, crea un objeto 3D translúcido del mismo tipo del mismo tamaño y cumpliendo la direccionalidad y altura de esos colliders.
   El resultado de esta orientación fue el siguiente:

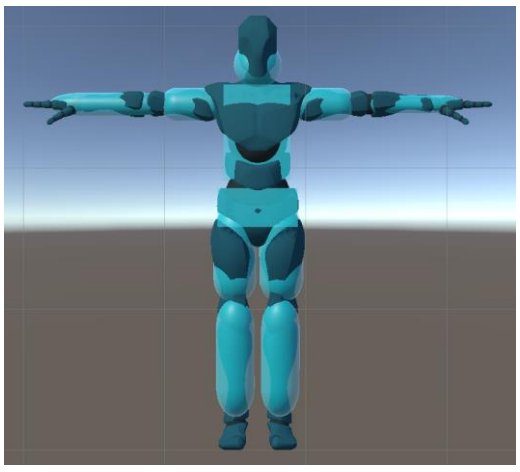

Figura 24 Desbloqueada alternativa 1

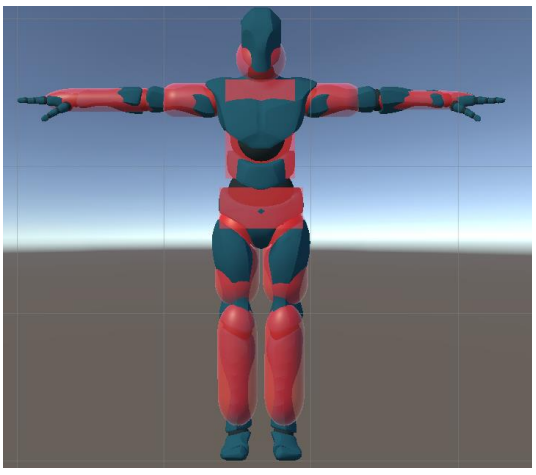

Figura 25 Bloqueada alternativa 1

   Luego creé una alternativa a este sistema con el mismo comportamiento pero creando solo objetos esféricos:

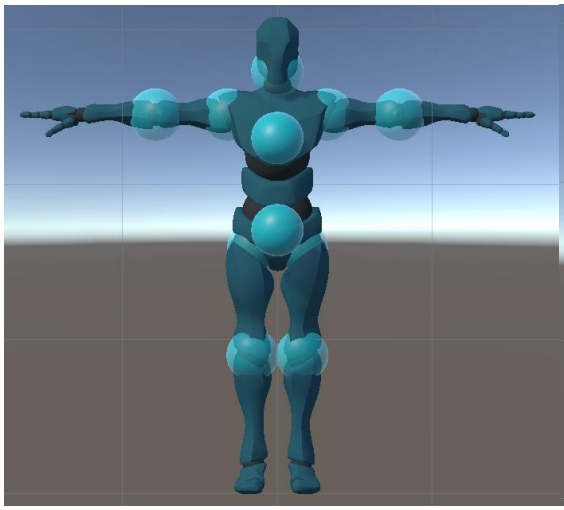

Figura 26 Desbloqueada alternativa 2

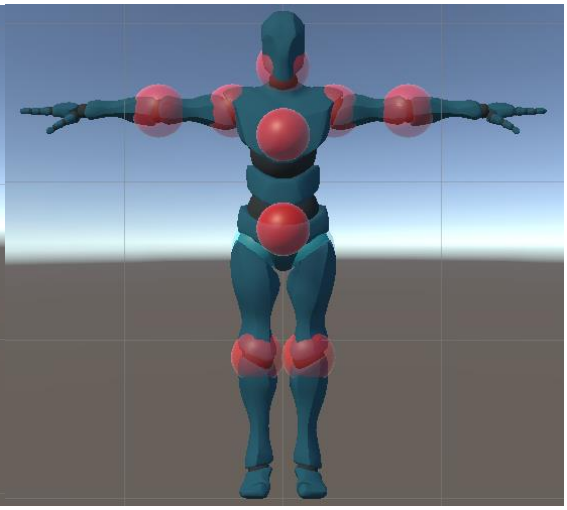

Figura 27 Bloqueada alternativa 2

2. Sistema fantasma:
   Para que el usuario pueda tener una percepción de lo que está realizando cuando captura fotogramas decidí crear una figura translúcida que replicase la animación del modelo original entre una captura anterior y la posición y rotaciones actual. Esto permite al usuario disponer de un feedback a tiempo real de su trabajo y permite correcciones y cambios a la figura para poder colocarlo de la manera que el usuario considere más correcta para la reproducción de esa animación.
   Se creó el script GhostController.cs para operar a esta figura.

   El script GhostController es muy similar al de AnimController.cs con unas pequeñas diferencias:
   - El GhostController ejecuta la animación constantemente y AnimController solo cuando el usuario lo desea.
   - La animación de GhostController solo dispone de dos keyframes. El primer keyframe introducido por el usuario o en su defecto la pose predeterminada del modelo y como segundo keyframe la posición actual del modelo.

   En este ejemplo visual se puede observar cómo funciona este sistema partiendo de la T-Pose predeterminada a una nueva pose creada por el usuario.

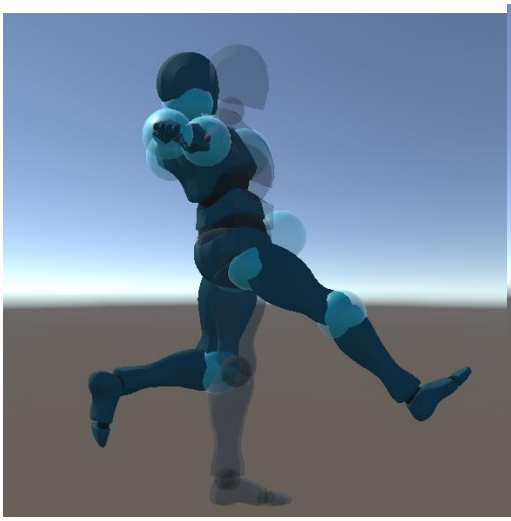

Figura 28 Fantasma 1

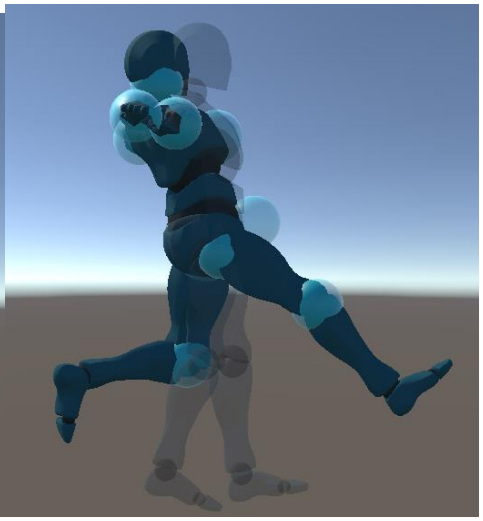

Figura 29 Fantasma 2

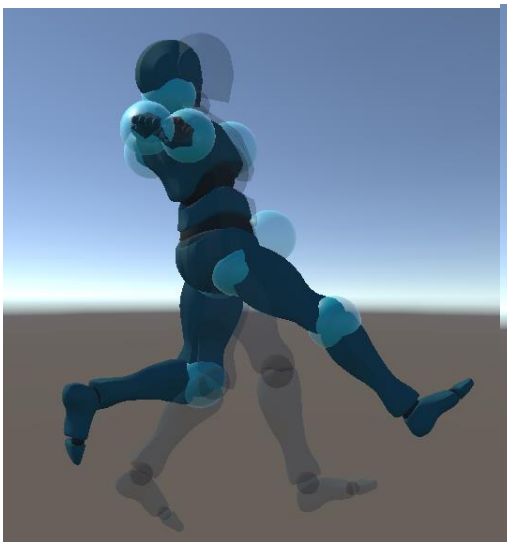

Figura 30 Fantasma 3

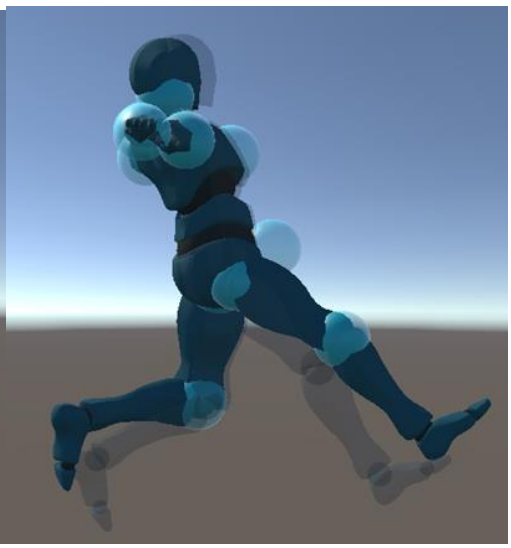

Figura 31 Fantasma 4

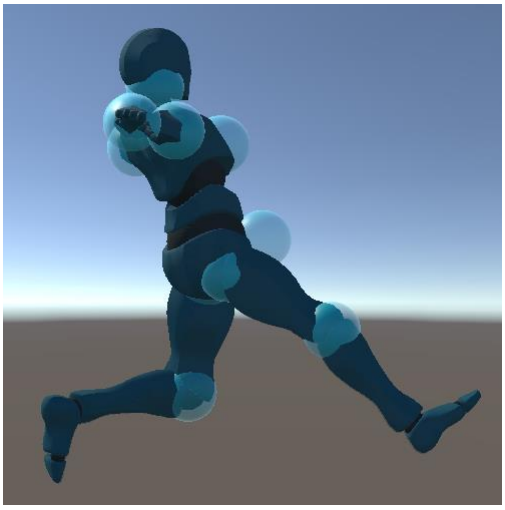

Figura 32 Fantasma 5

3. Menú:
   Hay que tener en cuenta que para desarrollar un menú en Realidad Virtual se debe crear elementos tridimensionales para no arruinar la inmersión y no provocar mareos con un menú estático.

   Para implementar el menú se crean dos planos como hijos del mando izquierdo y se crean otros dos planos como hijos del mando derecho, cada plano tendrá como tag un string que define su funcionalidad (Reiniciar, Reproducir, Tumbar, Capturar). Los tags en Unity sirven para distinguir “GameObjects” en las colisiones mediante la llamada de gameObject.CompareTag que es una función que pasándole un string nos devolverá un booleano que determine si ese objeto dispone de ese tag o no.

   A cada mano también se le vincula el script MenuManager.cs que se encargará de gestionar las llamadas que realiza el usuario al menú.

Aquí vemos el menú:

Figura 33 Menú

MenuManager.cs:

- Start(): Esta función se encarga de ocultar los menús de manera predeterminada. Para hacerlo usamos la función gameObject.SetActive(false). De esta manera el usuario no verá el menú ni podrá interactuar con él sin abrirlo.
- Update(): Obtiene el estado de los botones del menú en cada fotograma y cuando se presiona el botón del menú se activa llamando a la función Change.
- Change(): Si el menú estaba oculto lo activa y si no lo estaba lo desactiva.
- OnTriggerStay(): Las funciones onTrigger se activan cuando colisiona un objeto con rigidbody y collider con un objeto con collider trigger y en este caso en concreto se llama a esta función por cada fotograma con el que se está colisionando.

  Comparamos el tag del elemento colisionado para ver si es uno de los planos del menú. Si es uno de los planos del menú comprueba el valor del TriggerButton del mando. Cuando esté botón se deje de pulsar se llama a las funciones correspondientes según el tag de esta manera nos aseguramos de que solo se haga esto una vez.

Tag “Reiniciar”: Reinicia la escena llamando a la función de Unity SceneManager.LoadScene(SceneManager.GetActiveScene().name);

Tag “Reproducir”: Desactiva el GhostController y llama a la función Reproduce() del script AnimController.

Tag “Capturar”: Reinicia el GhostController y llama a la función CaptureFrame() del script AnimController.

Tag “Tumbar”: Mueve al modelo 3D a una posición tumbada en la camilla que se dispone en la escena.

Y para todos estos tags cuando se entra en la condición se cambia su textura para dar feedback al usuario de que está seleccionando este elemento:

Figura 34 Opción “REINICIAR” seleccionada

PROBLEMAS DURANTE EL DESARROLLO

- Falta de documentación:
  En este proyecto ha habido momentos en los que he notado que la documentación de Unity respecto a los temas de “scriptable animations” y “Quaternions” ha sido escasa y he tenido que realizar múltiples pruebas por mi cuenta para determinar el funcionamiento de muchas funciones de la API de Unity que están vagamente descritas.
- Limitaciones del sistema Ragdoll de Unity:
  El sistema Ragdoll de Unity se calibra mediante unos parámetros que el usuario introduce en las articulaciones. Estos parámetros determinan la libertad de movimientos que tienen distintas articulaciones.

  Existen movimientos que anatómicamente no se pueden realizar y han de ser tenidos en cuenta. Para calibrarlos adecuadamente se requiere de mucho tiempo sobre todo en las articulaciones de los brazos que tienen mucha más libertad de movimientos y más posibilidades que las piernas. Es por esto por lo que se ha decidido que los hombros dispongan de libertad total de movimientos, aunque muchos de estos sean anatómicamente imposibles será el animador el encargado de decidir que movimientos integrar y cuáles no. El resto de las articulaciones tienen ya unas limitaciones parametrizadas.

RESULTADOS

Vamos a realizar una simulación de una ejecución de un usuario para la creación de una animación básica de correr. A la izquierda de las Figuras estará la visión desde el editor de Unity y a la derecha la visión del usuario a través del visor de Realidad Virtual.

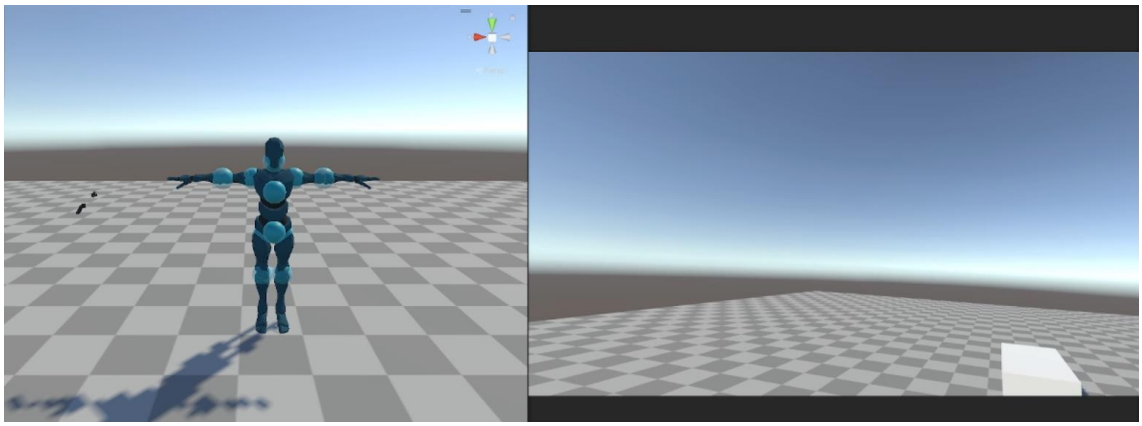

Figura 35 acercamiento al modelo 3D

Bloqueamos el pecho del modelo, en consecuencia también se bloquea la cadera por estar encima en la jerarquía.

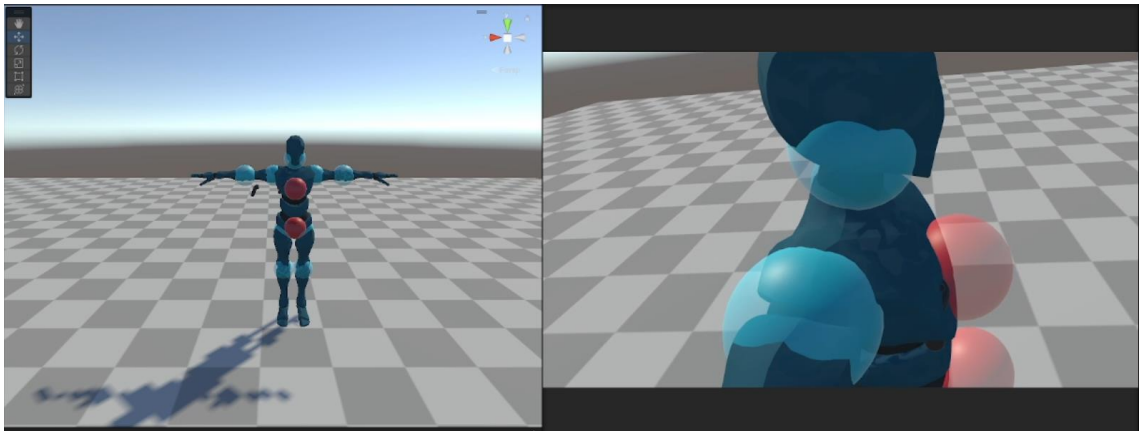

Figura 36 Colocamos el brazo derecho pegado al cuerpo

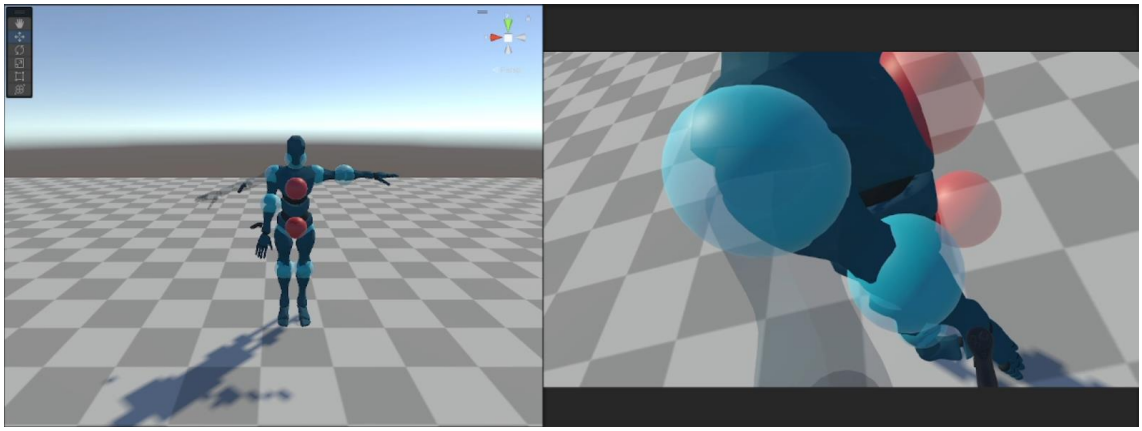

Figura 37 Realizamos la misma operación para el izquierdo y bloqueamos ambos brazos.

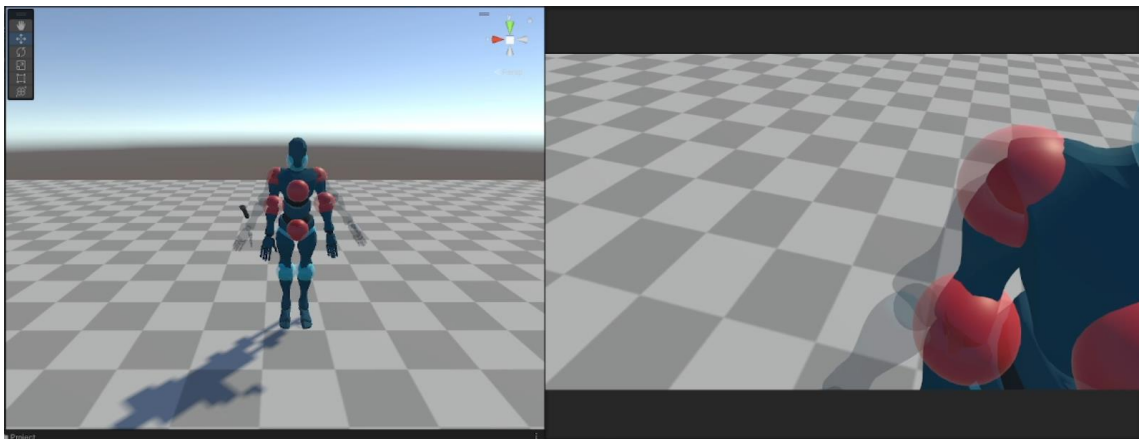

Figura 38 Fotograma inicial del modelo 3D

Hemos decidido que este va a ser nuestro fotograma inicial por lo que realizamos la primera captura abriendo el menú y dándole al botón “Capturar”. Después de esto la sombra dejará de proyectarse desde la pose predeterminada y tomará este fotograma como el inicial.

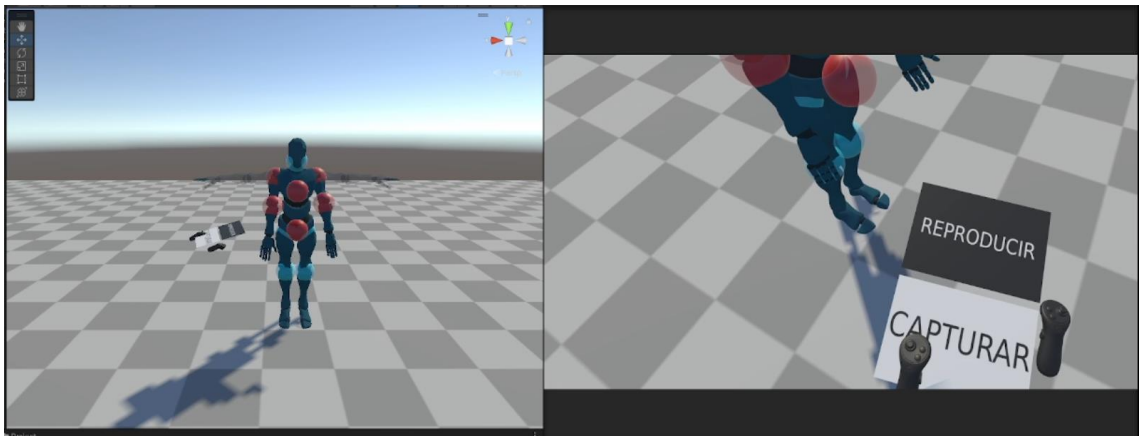

Figura 39 Capturando fotograma inicial

Desbloqueamos el brazo derecho y lo colocamos en una posición adelantada para simular que comienza a correr.

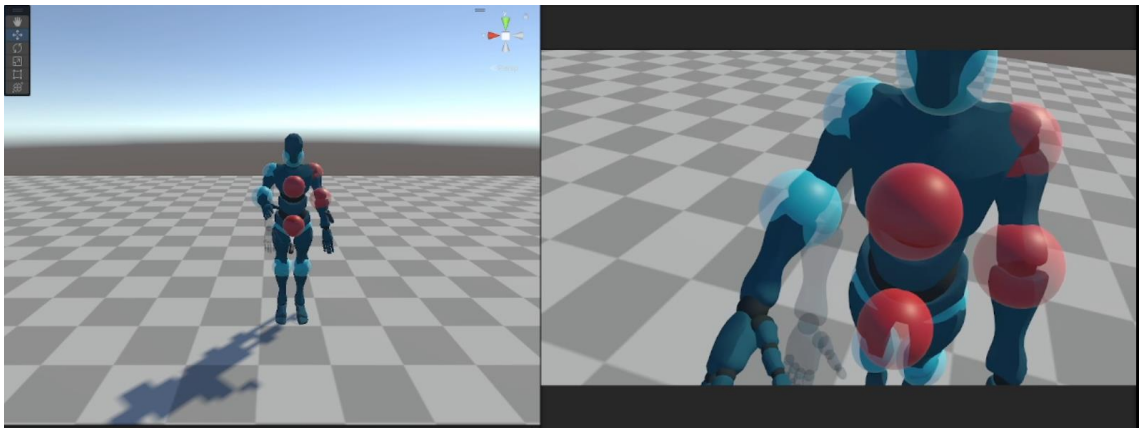

Figura 40 Colocación brazo derecho segundo fotograma

Seguidamente, desbloqueamos el brazo izquierdo y lo colocamos lo más atrás posible para obtener el efecto de correr con los brazos.

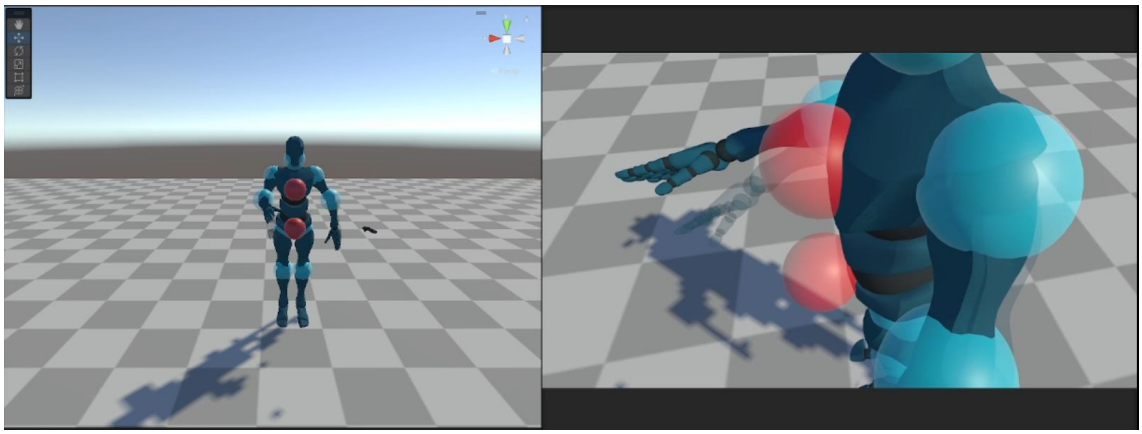

Figura 41 Colocación brazo izquierdo segundo fotograma

Levantamos ahora la pierna izquierda del maniquí.

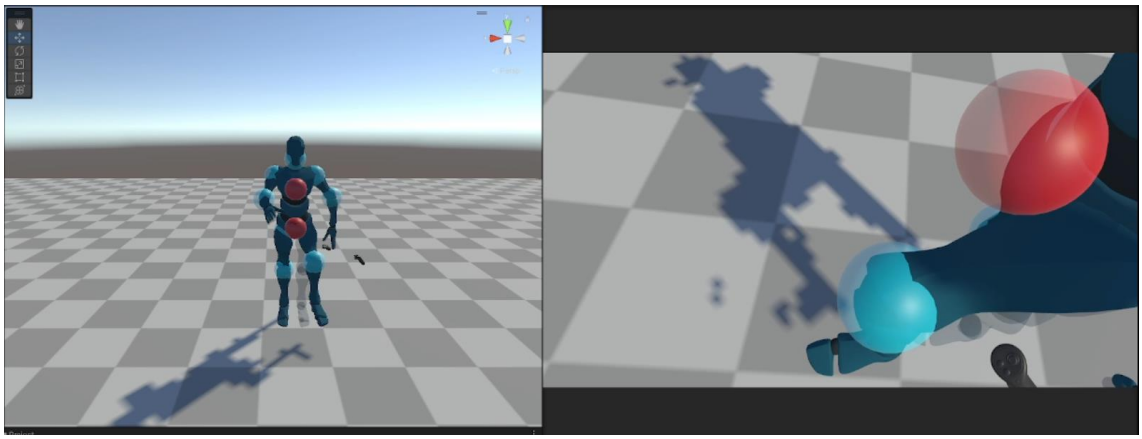

Figura 42 Colocación pierna izquierda segundo fotograma

Colocamos la pierna derecha del maniquí atrás.

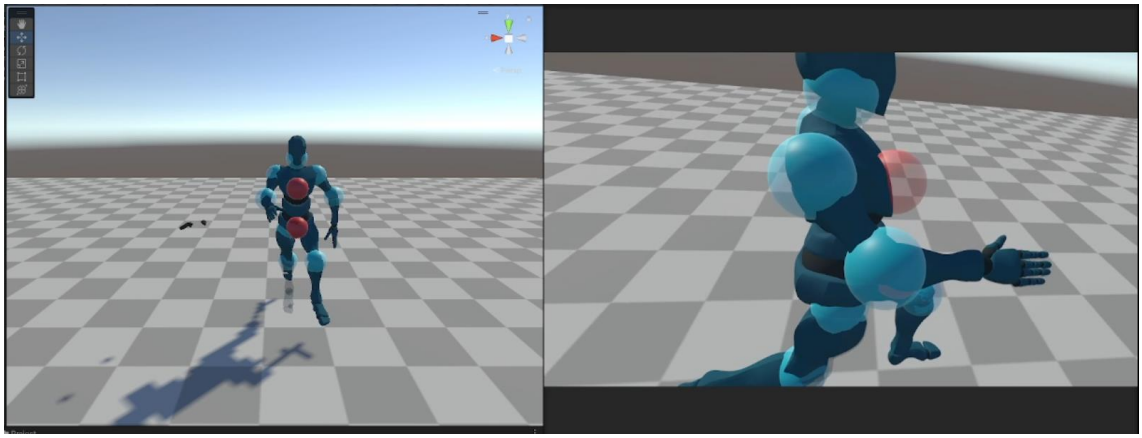

Figura 43 Colocación pierna derecha segundo fotograma

Capturamos ese instante como nuevo fotograma.

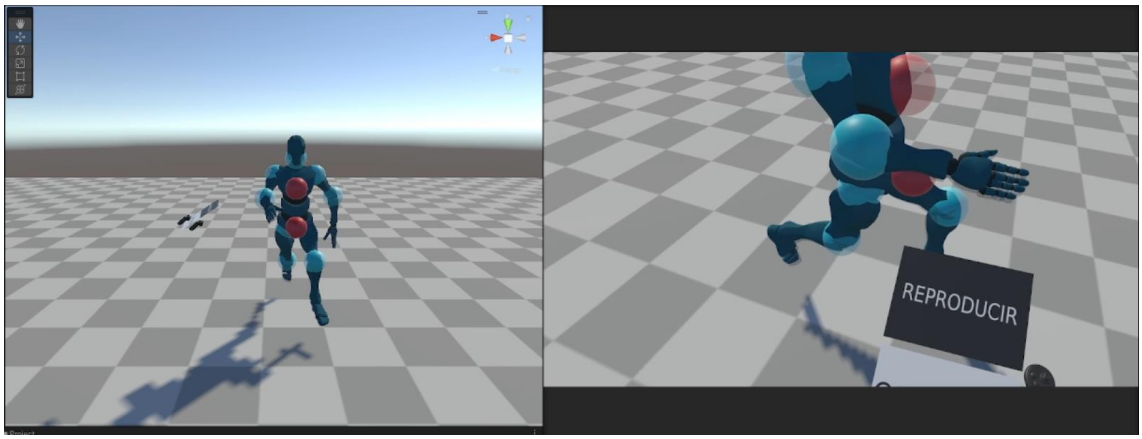

Figura 44 Captura segundo fotograma

Ahora solo nos queda realizar el movimiento inverso con piernas y brazos para acabar de realizar está animación básica.

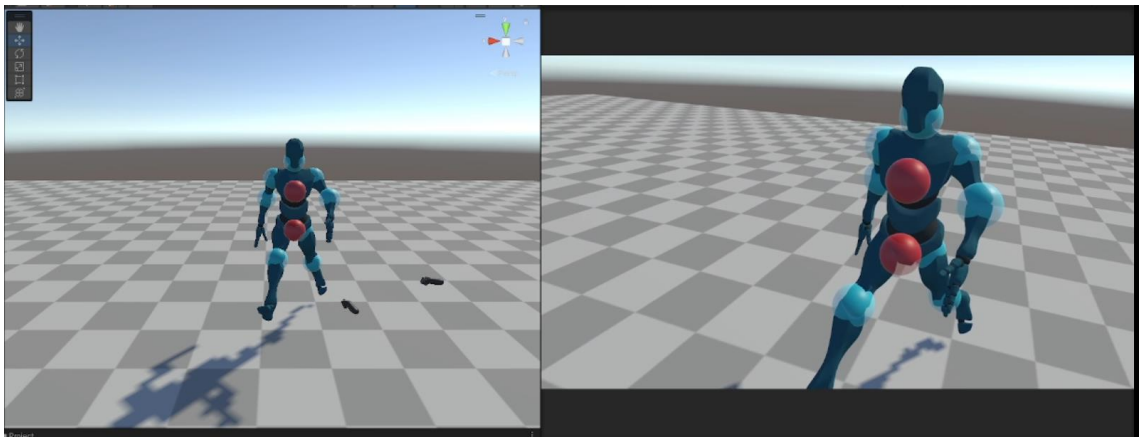

Figura 45 Último fotograma

Capturamos el último fotograma:

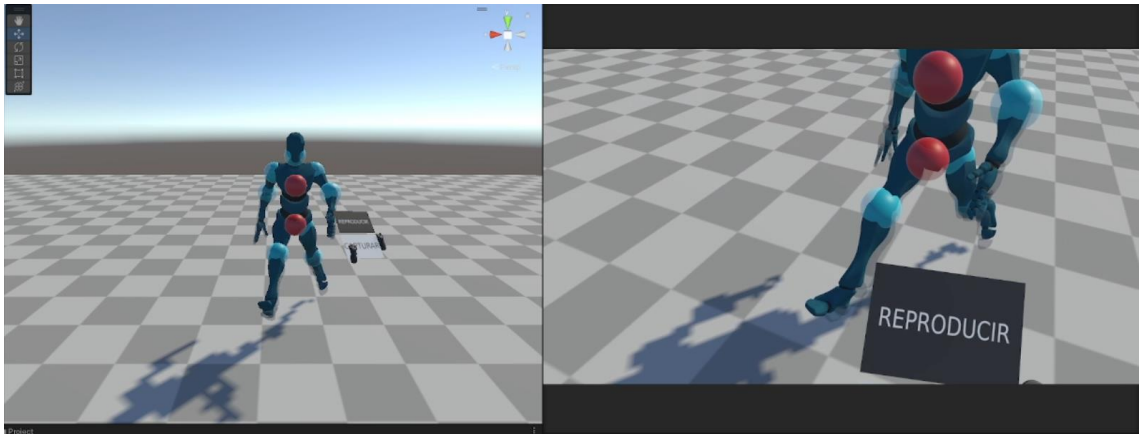

Figura 46 Captura último fotograma

Presionamos reproducir y la animación se reproducirá fotograma a fotograma con intervalos de 3 segundos.

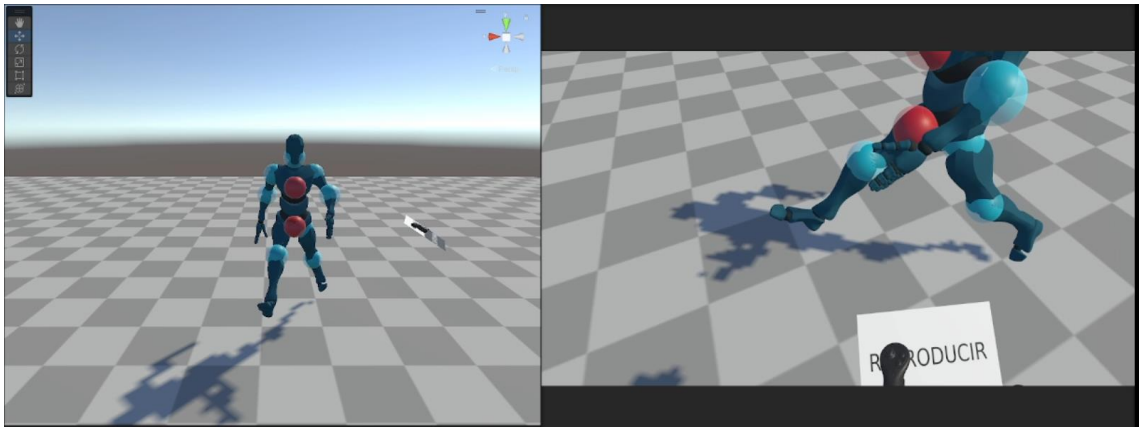

Figura 47 Reproducción de la animación

POSIBLES MEJORAS FUTURAS

Se han valorado distintas ideas cuando se redefinió el alcance del proyecto. Aunque muchas de estas ideas si se incluyen o se probaron en el proyecto. Otras se descartaron por estar fuera del alcance de este. Aquí se encuentran las posibles introducciones futuras que se podrán realizar a partir de la base creada en este trabajo:

1. Evitar que las extremidades se separen. A pesar de que no influye en el trabajo con la herramienta puesto que vuelven a su lugar de origen sin problema sería recomendable evitar que se puedan separar como en la siguiente figura:

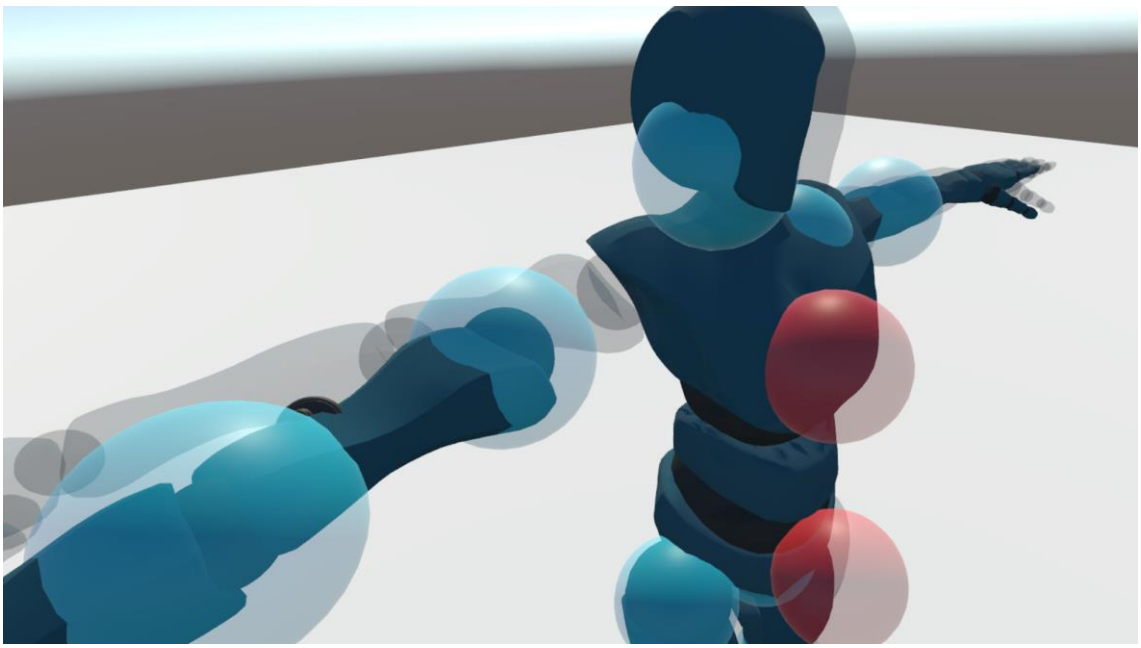

Figura 48 Extremidad separada del cuerpo

2. Soporte de captura de animaciones para las manos. Mediante el sistema de reconocimiento de manos que viene ya integrado en las Meta Quest 2 poder controlar las manos de un modelo 3D.
3. Poder interactuar con la gravedad en estos modelos 3D. Teniendo la capacidad de activar o desactivar la gravedad en nuestras animaciones para poder simular caídas, golpes con objetos, saltos etc.
4. Tener el control de las manos del modelo 3D para que se pueda adaptar a las superficies a las que se adhiera de una forma realista.
5. Poder modificar la velocidad de reproducción de los keyframes para poder acelerar o ralentizar el movimiento entre keyframes. Esto permitiría que el usuario pueda realizar las animaciones a los ritmos que desee a diferencia del sistema actual que tiene como parámetro fijo la separación entre keyframes.
6. Mejorar el sistema de agarre para ser capaz de poder rotar las extremidades a nuestro gusto y evitar problemas en las animaciones. Esto permitiría al usuario tener un mayor control de cómo quiere colocar exactamente las articulaciones de una forma mucho más rápida y sencilla.
7. Ser capaz de dar la forma de las curvas de animaciones mediante los mandos de realidad virtual para que el usuario tenga una mayor capacidad de desarrollo y control sobre la velocidad y momento del modelo 3D.

CONCLUSIONES

Este proyecto ha sido completado excediendo las expectativas iniciales de su alcance. Me ha permitido aprender a manejarme en la API de Unity cuya documentación es generalmente buena pero que en mi caso la he sentido escasa.

Al principió requerí hacer investigación y configuración de un entorno adecuado para desarrollar de la forma más óptima posible. Más adelante comenzó un desarrollo básico de pruebas para empezar a manejarme con la API tanto de Unity como de XR. Luego tuve que redefinir el alcance del proyecto puesto que ya había conseguido realizar los objetivos iniciales con éxitos y quería expandir más el proyecto.

A la hora de cumplir el nuevo alcance es cuando noté que la documentación sobre la API de Unity era más escasa de lo que me esperaba. En algunos momentos llegó a ser frustrante puesto que partía desde cero para desarrollar la captura de animaciones. Pero finalmente después de mucha investigación comencé a obtener los resultados que deseaba.

Gracias a este proyecto he podido expandir mis habilidades de investigación y programación en un entorno nuevo. También me ha permitido obtener nuevas habilidades de cómo entender al usuario que va a usar este producto y una mejora avanzada de mi conocimiento sobre Unity.

En definitiva, he conseguido realizar una aplicación funcional de la que se puede sacar partido en distintos campos.

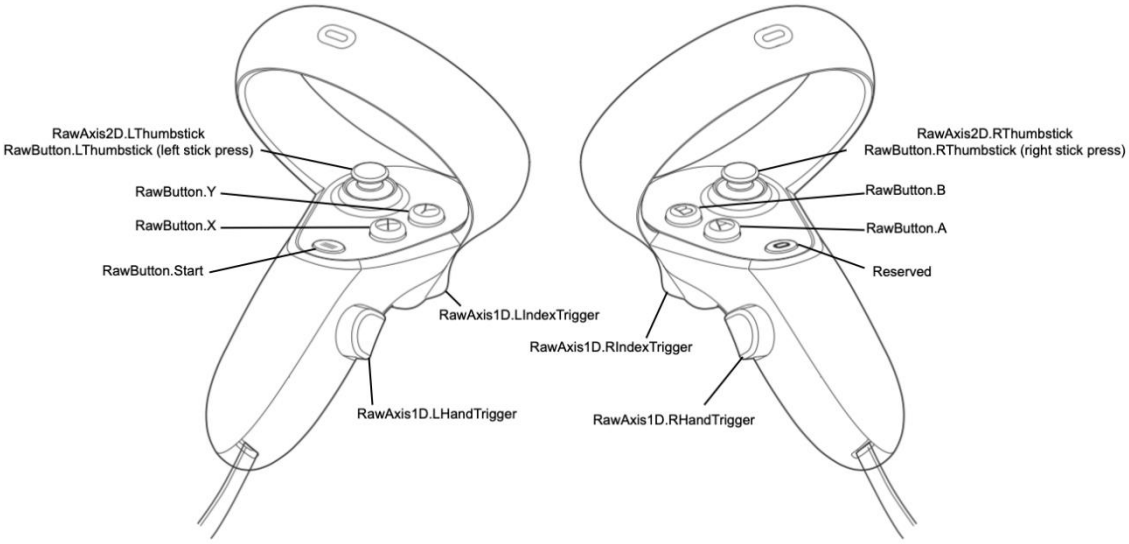

Figura 49 mapa de botones de Meta Quest 2 [28]

CONTROLADOR IZQUIERDO

- **RawAxis2D.LThumbstick**: Te permite moverte en el entorno sin necesidad de desplazarse físicamente.
- **RawAxis1D.LindexTrigger**: Te permite agarrar elementos y seleccionar elementos del menú.
- **RawAxis1D.LHandTrigger:** Te permite bloquear y desbloquear elementos. Los elementos bloqueados no podrán ser agarrados por el usuario. Bloquear una articulación bloqueará sus articulaciones padres.

- **RawButton.Y**: Te permite abrir y cerrar el menú izquierdo.

Figura 50 Menú izquierdo

CONTROLADOR DERECHO

- **RawAxis2D.RThumbstick**: Permite rotar la cámara sin necesidad de girar físicamente la cabeza.
- **RawAxis1D.RIndexTrigger**: Te permite agarrar elementos y seleccionar elementos del menú.
- **RawAxis1D.RHandTrigger**: Te permite bloquear y desbloquear elementos. Los elementos bloqueados no podrán ser agarrados por el usuario. Bloquear una articulación bloqueará sus articulaciones padres.
- **RawButton.B**: Te permite abrir y cerrar el menú derecho.

Figura 51 Menú derecho

CAPTURAR: Crea un keyframe en la que se capturan todas las rotaciones del modelo 3D. Capturar provocará que el modelo 3D translucido cambie su keyframe inicial al nuevo creado.

REPRODUCIR: Reproducirá la animación creada por el usuario.

TUMBAR: Colocará el modelo 3D en una posición tumbada.

REINICIAR: Reinicia el nivel completamente para poder empezar desde cero.

OPERABILIDAD

Cada keyframe estará separado por 1 segundo entre ellos. Un keyframe es un instante en el que se han almacenado todas las rotaciones de la jerarquía interna del modelo 3D.

El modelo translúcido nos permite tener un feedback de la animación que estamos creando a tiempo real, reproduce constantemente una animación creada con dos únicos keyframes:

- El keyframe inicial: Es el keyframe con el que empieza la animación. Si no se ha capturado ninguna animación será la posición inicial en la que se encuentre el modelo 3D.
- El keyframe final: Es un keyframe creado en base a la posición a tiempo real del modelo 3D. Este keyframe se actualiza constantemente lo que nos permite ver la transición entre estos dos keyframes constantemente.

La mejor forma de operar con esta aplicación para ir transformando al objeto 3D de una pose a otra es de dentro hacia fuera, es decir:

1. Colocamos el torso y lo bloqueamos.
2. Colocamos la cabeza y la bloqueamos.
3. Colocamos los antebrazos y muslos y los bloqueamos.
4. Colocamos los brazos y piernas y capturamos.

Repetir los pasos el número de keyframes que queramos y cuando acabemos presionar en reproducir.

REFERENCIAS

[1] Wikimedia, C. de los proyectos. (2023, January 16). *Historia de la animación por computadora*. Wikipedia. https://es.wikipedia.org/wiki/Historia_de_la_animaci%C3%B3n_por_computadora

[2] Wikimedia, C. de los proyectos. (2023, April 24). *Gráficos 3D por computadora*. Wikipedia. https://es.wikipedia.org/wiki/Gr%C3%A1ficos_3D_por_computadora#/media/Archivo:Primera_animaci%C3%B3n_3D_en_el_mundo.jpg

[3] Wikimedia, C. de los proyectos. (2023, April 28). *Autodesk Maya*. Wikipedia. https://es.wikipedia.org/wiki/Autodesk_Maya

[4]Camporota, A. (2019). How to create a dynamic pose (Maya 2019) [Video]. https://www.youtube.com/watch?v=BjLXOa_wcYw&t=218s

[5]Team, C. (2022, May 2). Cascadeur 2022: Improving Physics-Based Character Animation. *80lv*. https://80.lv/articles/cascadeur-2022-improving-physics-based-character-animation/

[6]*626ff927a2899.gif (700×339)*. (n.d.). https://cdn.80.lv/api/upload/content/58/626ff927a2899.gif

[7]Cannavò A., Zhang C., Wang W., Lamberti F.. (2020). Posing 3D Characters in Virtual Reality Through In-the-Air Sketches. 33rd International Conference on Computer Animation and Social Agents, CASA 2020 https://iris.polito.it/handle/11583/2837971

[8]Schaefer, M. (2019). VR Animation [Video]. https://www.youtube.com/watch?v=68lZJlbOGjs

[9] Contributors to Wikimedia projects. (2023, May 17). *Quest 2*. Wikipedia. https://en.wikipedia.org/wiki/Quest_2

[10] *Meta Quest 2: nuestras gafas de realidad virtual todo en uno con tecnología avanzada*. (n.d.). Meta Store. https://www.meta.com/es/quest/products/quest-2/tech-specs/#tech-specs

[11] *Brandguide*. (n.d.). https://brandguide.brandfolder.com/unity/downloadbrandassets

[12] Wikimedia, C. de los proyectos. (2023, May 16). *Unity (motor de videojuego)*. Wikipedia. https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)

[13] *Introduction to XR: VR, AR, and MR Foundations*. (n.d.). Unity Learn. https://learn.unity.com/course/introduction-to-xr-vr-ar-and-mr-foundations

[14] Wikimedia, C. de los proyectos. (2021, February 10). *Beat Saber*. Wikipedia. https://es.wikipedia.org/wiki/Beat_Saber#/media/Archivo:Beat_Saber_Logo.png

[15] Wikimedia, C. de los proyectos. (2021, February 10). *Beat Saber*. Wikipedia. https://es.wikipedia.org/wiki/Beat_Saber

[16] Technologies, U. (n.d.). *Unity*. Scripting API: Vector3.Lerp. https://docs.unity3d.com/ScriptReference/Vector3.Lerp.html

[17]Technologies, U. (n.d.). *Unity*. Scripting API: Quaternion.Lerp. https://docs.unity3d.com/ScriptReference/Quaternion.Lerp.html

[18] Contributors to Wikimedia projects. (2023, May 11). *Inverse kinematics*. Wikipedia. https://en.wikipedia.org/wiki/Inverse_kinematics

[19] *REDsdk*. (n.d.). Loading and Playing Skeletal Animations. http://www.downloads.redway3d.com/downloads/public/documentation/wf_skeletal_animation.html

[20] Technologies, U. (n.d.). *Unity*. Manual: Using Animation Curves. https://docs.unity3d.com/Manual/animeditor-AnimationCurves.html

[21] Technologies, U. (n.d.). *Unity*. Manual: The Hierarchy Window. https://docs.unity3d.com/Manual/Hierarchy.html

[22]Technologies, U. (n.d.). *Unity*. Manual: The Inspector Window. https://docs.unity3d.com/Manual/UsingTheInspector.html

[23] Technologies, U. (n.d.). *Unity*. Manual: The Project Window. https://docs.unity3d.com/Manual/ProjectView.html

[24]*Mixamo*. (n.d.). https://www.mixamo.com/#/

[25]Technologies, U. (n.d.). *Unity*. Manual: Console Window https://docs.unity3d.com/Manual/Console.html

[26] Technologies, U. (n.d.). *Unity*. Manual: Animation Window Guide. https://docs.unity3d.com/Manual/AnimationEditorGuide.html

[27] Technologies, U. (n.d.). *Unity*. Manual: The Scene View. https://docs.unity3d.com/Manual/UsingTheSceneView.html

[28] *Map Controllers*. (n.d.). Oculus Developers. https://developer.oculus.com/documentation/unity/unity-ovrinput/

FIGURAS

Figura 1 Representación de usuario controlando modelo 3D Figura 2 Primera animación 3D [2] Figura 3 Creación de pose dinámica a través de un boceto [4] Figura 4 Animación de modelo 3D con restricciones a elementos [6] Figura 5 Creación de pose a través del boceto de un usuario [7] Figura 6 Animación de batería a través de Quill [8] Figura 7 VR Headset Meta Quest 2 [10] Figura 8 VR Controllers [10] Figura 9 Logo Unity [11] Figura 10 Logo de Beat Saber [14] Figura 11 Pizarra de invocación de cubos usada por el jugador 1. Figura 12 Ejemplo de vista del jugador 2 Figura 13 Vista lateral de la cápsula del modelo 3D Figura 14 Vista frontal de la cápsula del modelo 3D Figura 15 Modelo 3D humanoide a la derecha y la posible representación de los huesos a la izquierda [19] Figura 16 Ejemplo de Animation Curve [20] Figura 17 Captura de Unity Figura 18 Hierarchy del proyecto Figura 19 Función CaptureFrame Figura 20 Función GetGameObjectPath Figura 21 Función Reproduce Figura 22 Rigidbody de un elemento en el que se señalan los componentes de bloqueo Figura 23 Jerarquía de nuestro modelo 3D en Unity Figura 24 Desbloqueada alternativa 1 Figura 25 Bloqueada alternativa 1 Figura 26 Desbloqueada alternativa 2 Figura 27 Bloqueada alternativa 2 Figura 28 Fantasma 1 Figura 29 Fantasma 2 Figura 30 Fantasma 3 Figura 31 Fantasma 4 Figura 32 Fantasma 5 Figura 33 Menú Figura 34 Opción “REINICIAR” seleccionada Figura 35 acercamiento al modelo 3D Figura 36 Colocamos el brazo derecho pegado al cuerpo Figura 37 Realizamos la misma operación para el izquierdo y bloqueamos ambos brazos. Figura 38 Fotograma inicial del modelo 3D Figura 39 Capturando fotograma inicial Figura 40 Colocación brazo derecho segundo fotograma Figura 41 Colocación brazo izquierdo segundo fotograma Figura 42 Colocación pierna izquierda segundo fotograma Figura 43 Colocación pierna derecha segundo fotograma Figura 44 Captura segundo fotograma Figura 45 Último fotograma Figura 46 Captura último fotograma Figura 47 Reproducción de la animación Figura 48 Extremidad separada del cuerpo Figura 49 mapa de botones de Meta Quest 2 [28] Figura 50 Menú izquierdo Figura 51 Menú derecho