

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

Desarrollo de herramienta software para la edición de animaciones para retargeting de personajes



Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor: Adrian Guiral Mallart

Director: Óscar Ardáiz Villanueva

Pamplona, 08/06/23

upna

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Índice

1. Agradecimientos	4
2. Palabras Clave	4
3. Motivación.....	4
4. Introducción	5
5. Objetivos	5
6. Resumen.....	5
7. Animación en la industria.....	5
8. Retargeting.....	7
9. Materiales	8
9.1. Hardware	8
9.1.1. PC.....	8
9.2. Software.....	8
9.2.1. Unity 3D	8
9.2.2. Blender	9
9.2.3. Mixamo.....	9
10. Tareas	9
10.1. Pruebas iniciales con animaciones.....	9
10.2. Selección de modelos y animaciones.....	12
10.3. Investigación sobre Unity 3D	12
10.4. Investigación sobre Quaternions y rotaciones.....	12
10.5. Investigación sobre animación y curvas de animación.....	15
11. Implementación.....	17
11.1. Introducción	17
11.2. Estructuras de datos.....	17
11.3. Captura de animación	19
11.4. Guardado de animación	21
11.5. Reproducción de animación.....	23
11.5.1. Reproducción con curvas	23
11.5.2. Reproducción manual.....	24
11.6. UI	26
11.6.1. Configuración de sliders	27

11.6.2.	Menú de elección de personajes.....	27
11.6.3.	Menú de elección de animación	28
11.6.4.	Interfaz final.....	28
11.7.	Modificación de animación	28
11.8.	Cámara y movimiento	30
11.9.	Detección de colisiones.....	30
12.	Problemas encontrados.....	32
12.1.	Documentación de Scripting API Unity 3D.....	32
12.2.	Modificación de rotaciones.....	33
12.3.	Reproducción manual de animación.....	33
12.4.	Pathing de hijos de modelo 3D	33
12.5.	Rigging modelo 3D	33
12.6.	Creación malla 3D para detección	34
13.	Posibles próximos objetivos	34
14.	Manual.....	35
15.	Conclusión	38
16.	Referencias	38

1. Agradecimientos

Antes de comenzar con la redacción de la memoria que pone punto final a mi paso por la universidad quiero aprovechar para dar las gracias.

Quiero dar las gracias a mi familia por apoyarme a lo largo de estos cuatro largos años y confiar en mí y en mis decisiones, a pesar de mis dudas.

Quiero dar las gracias a todas las personas con las que he entablado grandes amistades en estos cuatro años y que, al fin y al cabo, han sido camaradas y han compartido su paso por la universidad conmigo. No hay palabras para describir el enriquecimiento y crecimiento personal que han supuesto para mí cada uno de ellos y ellas.

Quiero dar las gracias en específico a Youssef Benbelkheir por apoyarme en mis frustraciones con el proyecto y darme ánimos para seguir adelante con el desarrollo.

Quiero dar las gracias por último a mi tutor, Óscar Ardáiz por guiarme en el desarrollo del TFG.

2. Palabras Clave

- Unity
- Blender
- Mixamo
- Rigging
- Pathing
- Mesh
- Collider
- Retargeting
- Keyframe
- API

3. Motivación

La principal motivación del proyecto es la creación de una herramienta accesible para la modificación de animaciones 3D y su 'retargeting' a otros modelos 3D de forma sencilla y rápida. Además, esta motivación parte de la familiaridad con el entorno de desarrollo de Unity, en el cual ya tenía experiencia previa en el desarrollo de videojuegos en ambos Unity 2D y Unity 3D. Asimismo, esta experiencia facilitaría el trabajo de investigación en los distintos campos que abarca el proyecto y su implementación en la herramienta.

4. Introducción

El 'animation retargeting' o reorientación de animaciones es una función que permite la reutilización de animaciones entre personajes que utilizan el mismo esqueleto, pero no comparten las mismas dimensiones [1]. Esta técnica es muy útil debido a que reduce la carga de trabajo del animador enormemente y solventa el problema de animar la misma animación para cada personaje nuevo.

Se trata de una técnica muy común y extendida principalmente dentro de la industria del videojuego, no obstante, también está presente en la industria del cine para ajustar animaciones a avatares 3D utilizados en la producción tanto de películas como series.

Para facilitar este trabajo, se va a utilizar el motor de desarrollo Unity 3D, uno de los motores más conocidos a nivel mundial y con mayor disponibilidad, teniendo a su disposición una amplia lista de herramientas y versiones.

5. Objetivos

Los objetivos de este proyecto son muy claros, en primer lugar, se busca ajustar animaciones de un modelo 3D a modelos 3D con misma estructura ósea ('rigging') pero dimensiones irregulares de manera que la animación reorientada ('retargeted') opere de forma natural sobre los nuevos modelos. Esto se consigue a través de una edición mínima de la animación para solucionar errores causados por la naturaleza de las dimensiones del modelo. Se trata de una edición mínima para no provocar cambios en la finalidad de la animación.

En segundo lugar, se persigue desarrollar una herramienta de fácil uso que permita un acercamiento accesible al mundo de la animación en el ámbito principalmente de la creación y desarrollo de videojuegos.

6. Resumen

Se busca desarrollar una herramienta en Unity 3D con C# para la edición y modificación de animaciones sobre modelos 3D. Haciendo uso de la herramienta el usuario podrá observar la animación y modificarla a través de controles tales como controles deslizantes o indicadores 3D sobre huesos del modelo. La herramienta permitirá la creación de estas nuevas animaciones y su almacenamiento con el fin de realizar un 'retargeting', es decir, ajustar la animación original a distintos modelos 3D.

7. Animación en la industria

Para entender el ámbito en el que vamos a trabajar merece la pena entender la situación actual de la animación en la industria. Cabe destacar que la animación se encuentra en varios sectores de la industria como la industria del cine, la industria de los videojuegos, tecnología industrial y prácticamente todo el sector tecnológico.

Una vez visto el alcance de la animación en la industria podemos entender por qué el mercado global de la animación se estimó en 2022 con un valor de 391.19 mil millones de dólares americanos y se espera que crezca hasta los 587.1 mil millones de dólares americanos para el comienzo de 2030. Este crecimiento en la industria se debe principalmente a los avances en las comunicaciones entre sectores tecnológicos, el creciente interés y éxito de la industria del entretenimiento y la extensiva accesibilidad a internet en distintas regiones del mundo [2].

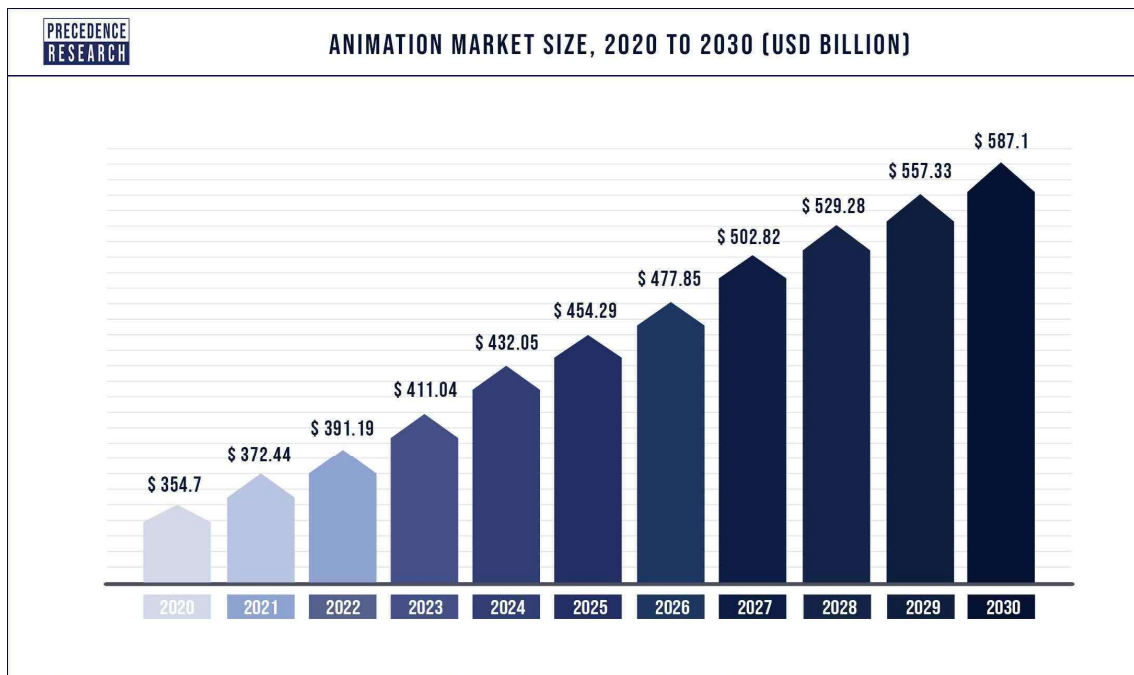


Figura 1: Estimación del crecimiento del mercado

Por otro lado, el continuo desarrollo de las tecnologías de realidad virtual y realidad aumentada también brindan nuevas oportunidades a la industria de la animación, como pueden ser el desarrollo de videojuegos en realidad virtual o el desarrollo de metaversos, tendencia en aumento dentro del sector tecnológico, aunque sin mucho éxito como por ejemplo el metaverso de Meta (antiguamente Facebook) debido a un pobre desarrollo de este.



Figura 2: Fotografía del estado del metaverso por Meta

8. Retargeting

Como se ha mencionado antes, el 'retargeting' o reorientación de animaciones es una técnica que consiste en la reutilización de animaciones entre personajes con mismo esqueleto.

El esqueleto de un modelo 3D está formado por articulaciones, esto son puntos designados a lo largo del modelo usados para realizar transformaciones como en la gran mayoría de ocasiones, rotaciones [3].

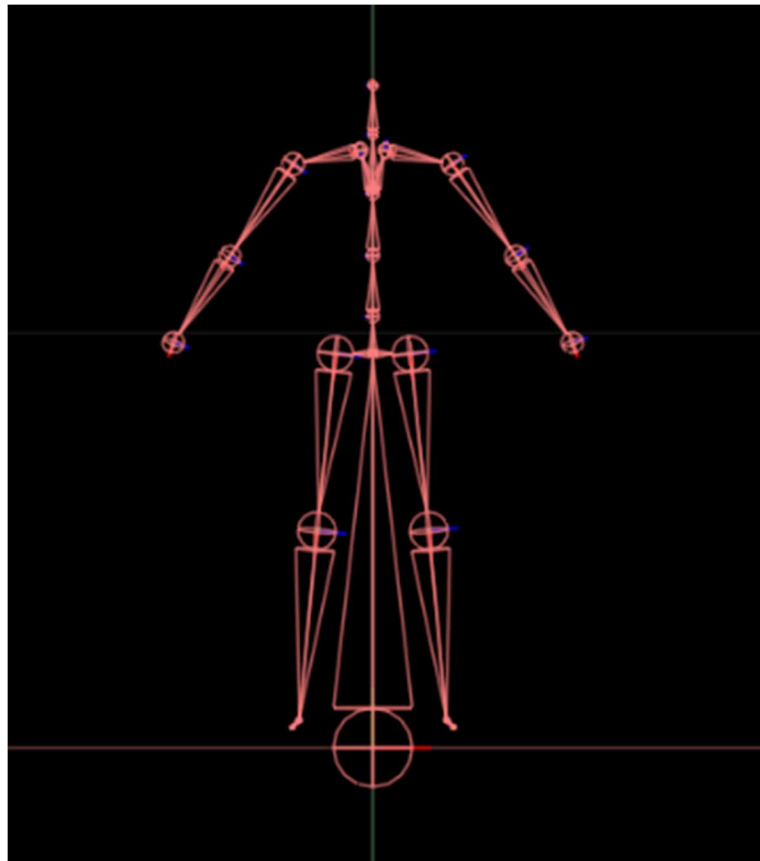


Figura 3: Esqueleto Modelo 3D [3]

Las articulaciones del esqueleto tienen una jerarquía 3D con relación padre-hijo que establece el orden de las transformaciones que se ejecutan en cada articulación para conformar una animación completa. Con esto en mente, la reorientación consigue trasladar las transformaciones almacenadas en cada articulación al esqueleto del modelo 3D que queremos animar con la limitación de que el esqueleto debe ser idéntico o casi idéntico manteniendo como mínimo el orden de jerarquía del esqueleto original.

9. Materiales

9.1. Hardware

9.1.1. PC

Para desarrollar la herramienta se ha hecho uso de varios PCs, siendo uno de ellos el PC personal y otro el PC de la sala de proyectistas. Como previsualizador del contenido desarrollado en las reuniones se ha utilizado un portátil personal. Ambos PCs, tanto el personal como el de la sala de proyectistas tienen tarjetas gráficas potentes que cumplen el requerimiento de desarrollo que expresa Unity 3D. Adjunto las especificaciones de mi PC personal ya que no tengo las especificaciones exactas del PC de proyectistas más que la tarjeta gráfica es una Nvidia GeForce RTX 3060.

PC personal:

- CPU: AMD Ryzen 5 5600X 3.7GHz
- RAM: 16GB DDR4 3200Mhz
- Placa base: MSI MPG B550
- GPU: Gygabyte GeForce RTX 2060 (Nvidia)



Figura 4: PC

Estas especificaciones son más que suficientes para desarrollar la herramienta sin problemas gráficos o de procesamiento a pesar de las limitaciones de Unity 3D.

9.2. Software

9.2.1. Unity 3D

Como se ha mencionado anteriormente, la herramienta se ha desarrollado en Unity 3D. Se trata de un motor gráfico para desarrollo de videojuegos multiplataforma cuyo desarrollador es Unity Technologies. El motor tiene soporte en Windows, Mac y Linux y fue lanzado al mercado el 30 de mayo de 2005.

La herramienta llegó al mercado con el objetivo de facilitar el desarrollo de videojuegos a desarrolladores independientes ofreciendo un motor de juego que sería muy difícil de crear por sus propios medios. De esta forma consiguieron que el desarrollo 2D y 3D fuera más accesible para personas interesadas en el desarrollo, pero sin medios para obtener un motor [4].



Figura 5: Unity Logo

9.2.2. Blender

Blender es un programa de “open source” dedicado al modelado, renderizado y creación de gráficos 3D. Se trata de un “software” multiplataforma gratuito con una gran capacidad para crear modelos y animaciones 3D, disponiendo de una amplia gama de herramientas para su desarrollo. Además, es una herramienta muy popular con una curva de aprendizaje accesible, existen una gran cantidad de videos de la comunidad de Blender sobre su uso, aprendizaje, consejos y acercamiento a la herramienta [5].

Hemos usado esta herramienta para comprobar conceptos técnicos de modelaje 3D que no se pueden modificar desde Unity 3D y que ocasionaban problemas en el motor de desarrollo, este acercamiento a Blender como una herramienta de apoyo ha resultado de gran ayuda para una mejor implementación del proyecto.



Figura 6: Blender Logo

9.2.3. Mixamo

Mixamo es una tecnología 3D para animación de personajes 3D, utiliza métodos de “machine learning” para automatizar los pasos del proceso de animación, incluyendo el modelaje 3D. Se trata de una tecnología perteneciente a Adobe y será nuestra base para recoger animaciones y modelos 3D.

Todas las animaciones y modelos utilizados serán de mixamo ya que para realizar un retargeting al menos los modelos deben compartir una jerarquía de “rigging” muy parecida y las animaciones deben saber sobre que huesos (“rig”) deben ejecutarse, por lo tanto, es importante que las curvas de animaciones conozcan los nombres de los huesos (miembros del personaje 3D) para que Unity reproduzca correctamente la animación.



Figura 7: Mixamo Logo

10. Tareas

10.1. Pruebas iniciales con animaciones

Antes de comenzar el proyecto, se decidió que sería conveniente realizar una investigación previa sobre el problema que queríamos solucionar. Para ello se cargaron distintas animaciones sobre un modelo 3D y se exageró el modelo para obtener un

personaje con piernas largas, uno con brazos largos y otro con una cabeza exageradamente grande.

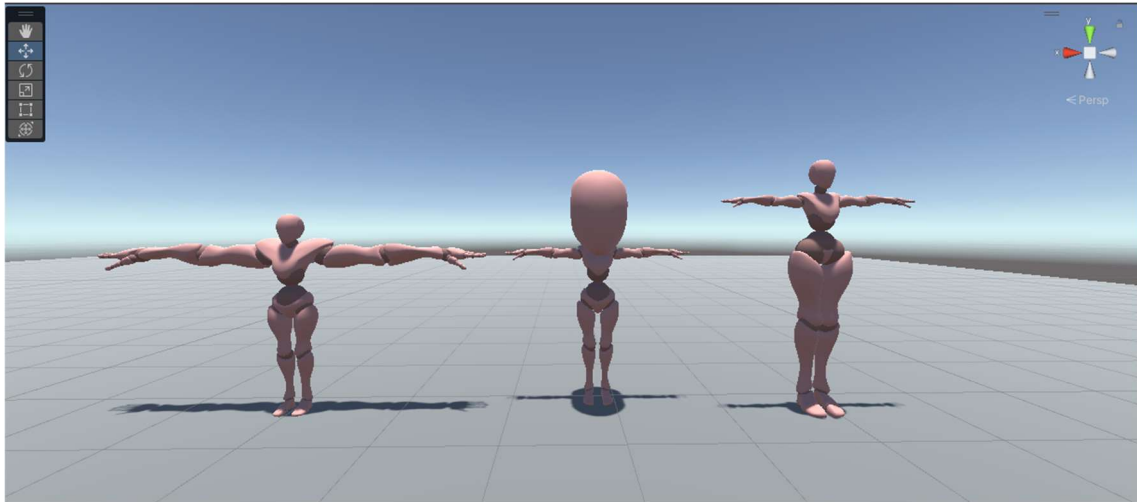


Figura 8: Personajes creados

Seguidamente, se probaron las animaciones sobre estos tres personajes para visualizar el problema existente en el retargeting que pretendíamos resolver. Efectivamente, se podía observar que las animaciones en estos nuevos personajes producían comportamientos indeseados como superposición de miembros del personaje.

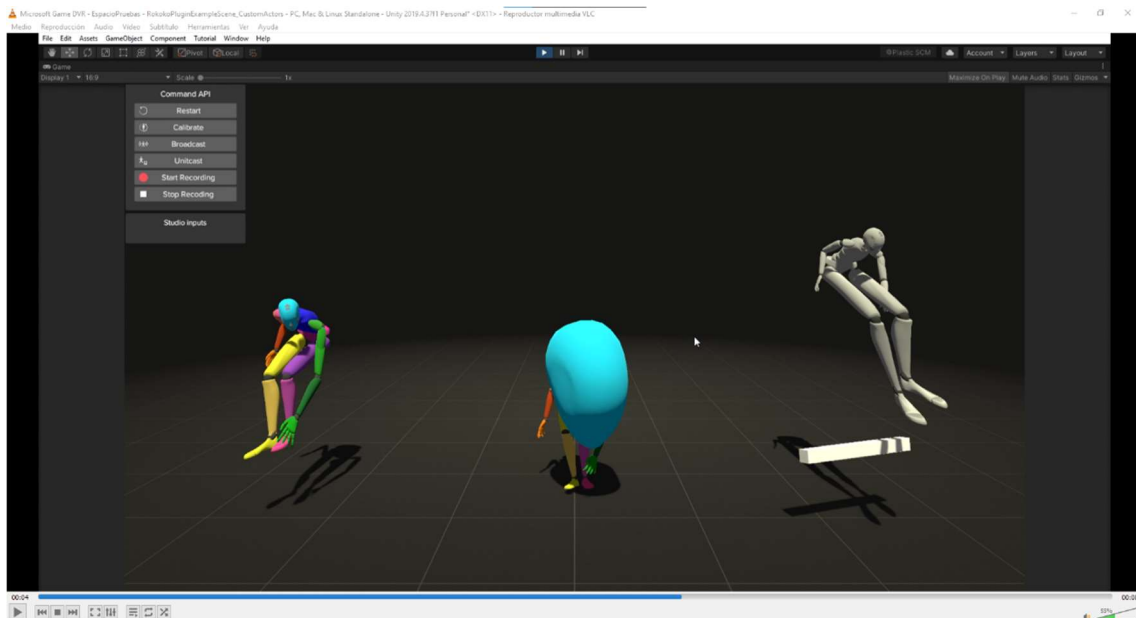


Figura 9: Problema en retargeting sobre modelos

Se hicieron pruebas con otras animaciones para confirmar la existencia del problema.

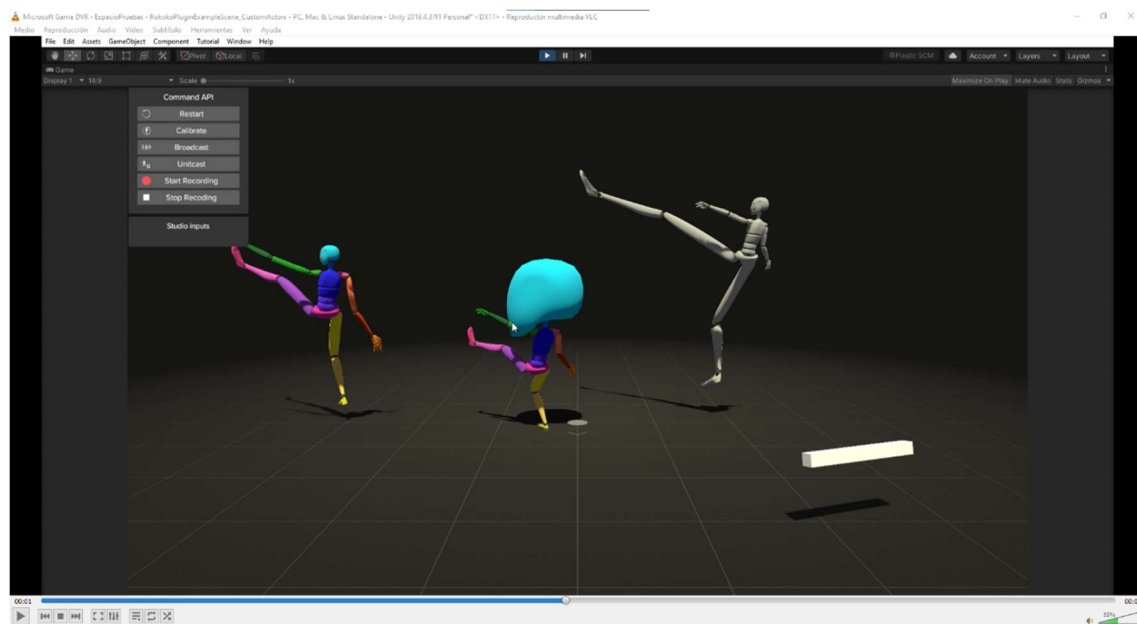


Figura 10: Problema sobre animación lanzar pelota

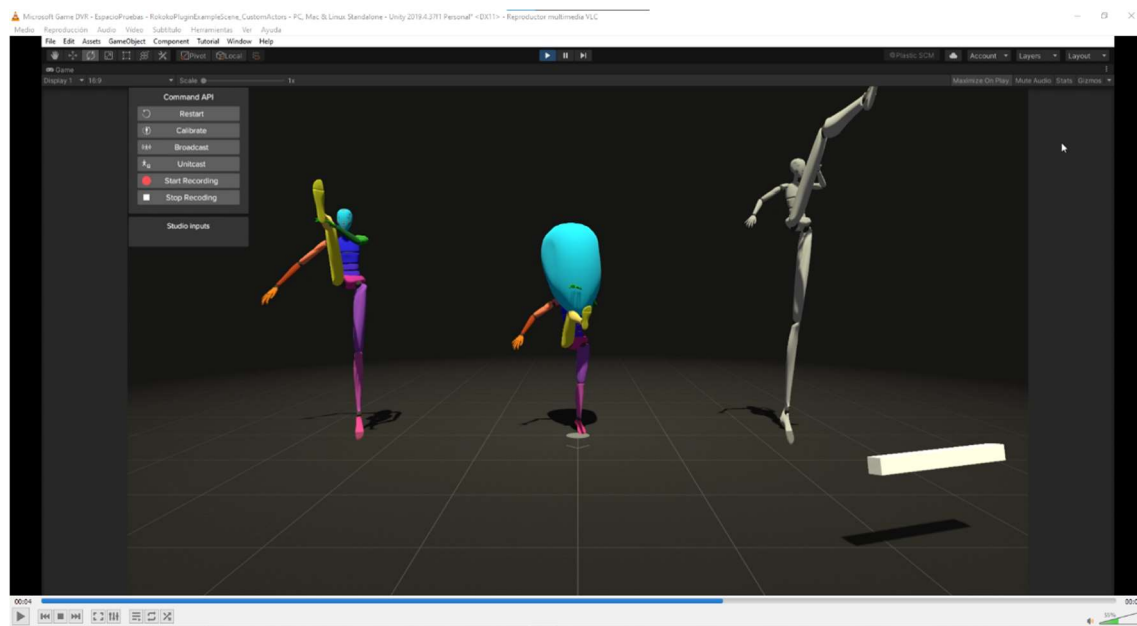


Figura 11: Problema sobre animación

Tras confirmar la existencia del problema se procedió a la implementación de la herramienta y su respectiva investigación acerca de los temas que nos interesaban para conseguir una implementación de la herramienta accesible.

10.2. Selección de modelos y animaciones

Para hacer pruebas de la herramienta a lo largo del desarrollo se realizó una selección inicial de animaciones y modelos sobre los que trabajar. El modelo elegido fue el “bot” genérico de “mixamo” y las animaciones fueron un salto hacia delante y la animación de tirar la pelota usada en las pruebas iniciales con animaciones.

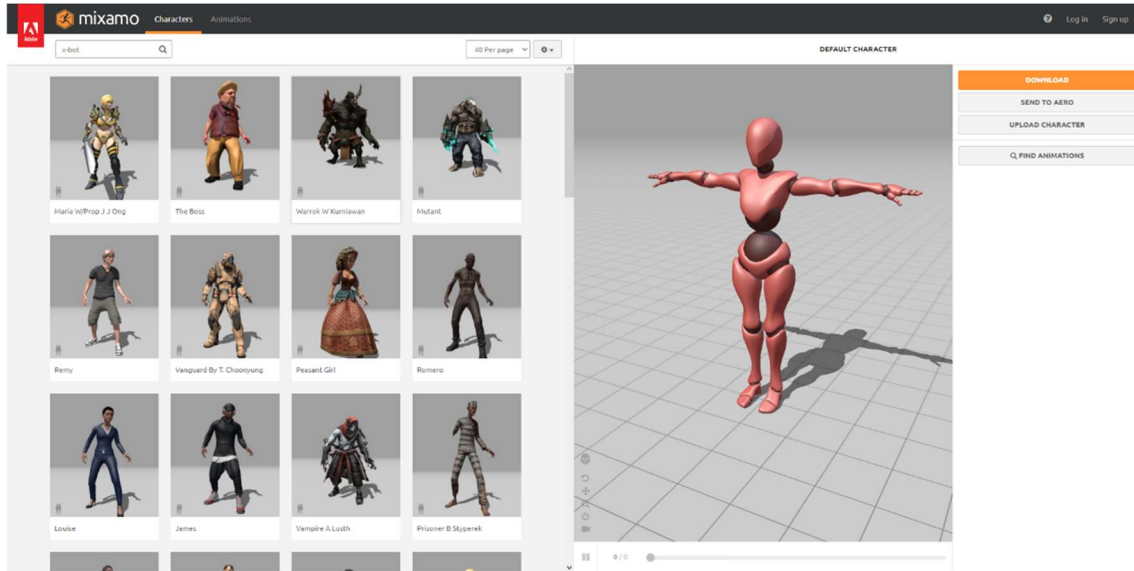


Figura 12: Modelo "mixamo" y animaciones

10.3. Investigación sobre Unity 3D

El entorno de desarrollo de Unity 3D de forma resumida está conformado por un editor, un sistema de archivos, un sistema de escenas y una jerarquía de objetos para cada escena que está formada principalmente por una cámara, una fuente de luz y un conjunto de objetos.

Unity dispone de una web de aprendizaje llamada “Unity Learn” que proporciona videos y procesos descritos paso a paso sobre distintos temas y cuestiones para comenzar a desarrollar en Unity. “Unity Learn” dispone de tutoriales y cursos categorizados por dificultad y familiaridad con el entorno de desarrollo de Unity lo cual la convierte en una web muy útil para aprender conceptos de Unity [6].

Con los conocimientos adquiridos en la titulación de desarrollo de videojuegos de la UPNa sumados a “Unity Learn” para recordar conceptos e implementaciones para montaje de escena, reproducción de animaciones y manejo del editor se ha podido adquirir un nivel más que necesario para el desarrollo del proyecto. Todo esto sumado a las investigaciones sobre conceptos de naturaleza más técnica propios del tema del proyecto que se mencionarán a continuación.

10.4. Investigación sobre Quaternions y rotaciones

Quaternion es un término matemático que hace referencia a un sistema numérico que extiende a los números complejos. Fueron descritos por primera vez por el matemático

irlandés William Rowan Hamilton en 1843 y aplicado a la mecánica en el espacio tridimensional. Hamilton definió un “quaternion” como el cociente de dos líneas dirigidas en un espacio tridimensional, dicho de otra forma, el cociente de dos vectores.

La multiplicación de “quaternions” es no conmutativa y se representan generalmente con la forma $a + b i + c j + d k$ donde a, b, c y d son números reales e i, j, k son vectores de base, conjunto de vectores B en un espacio vectorial V en el cual cada elemento de V puede ser escrito de forma única como una combinación lineal de B [7].

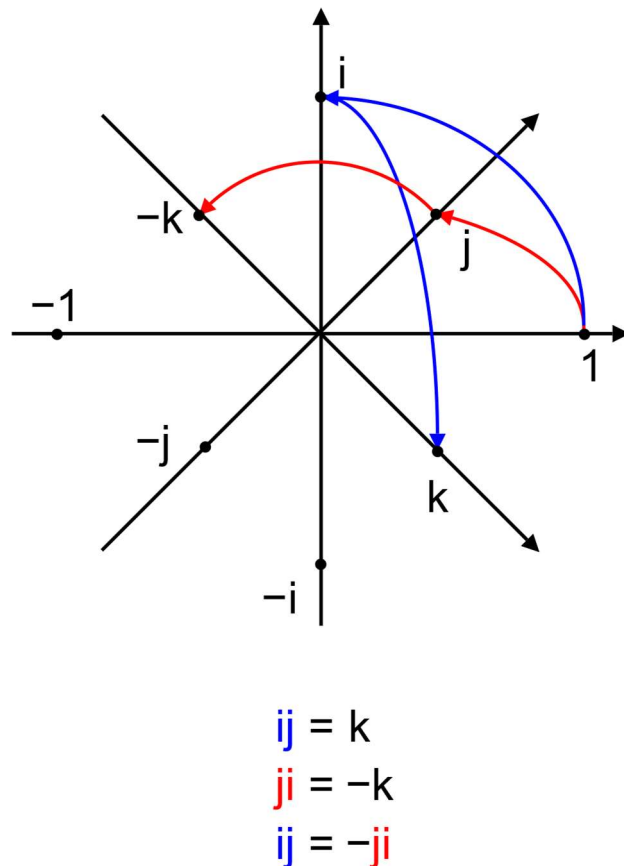


Figura 13: Quaternion

Los “quaternions” se utilizan en matemática pura pero también tienen usos prácticos en matemática aplicada especialmente en cálculos de rotaciones tridimensionales, principal tema de nuestro proyecto ya que las animaciones 3D son un conjunto de rotaciones tridimensionales y Unity maneja las rotaciones con “quaternions”. También se pueden utilizar con otros métodos de rotación como los ángulos eulerianos, tres ángulos introducidos por Leonhard Euler para describir la orientación de un “rigid body” con respecto a un sistema de coordenadas fijo [8], y matrices de rotación, matrices de

transformación utilizadas para realizar rotaciones en el espacio euclidiano [9], dependiendo de su aplicación [10].

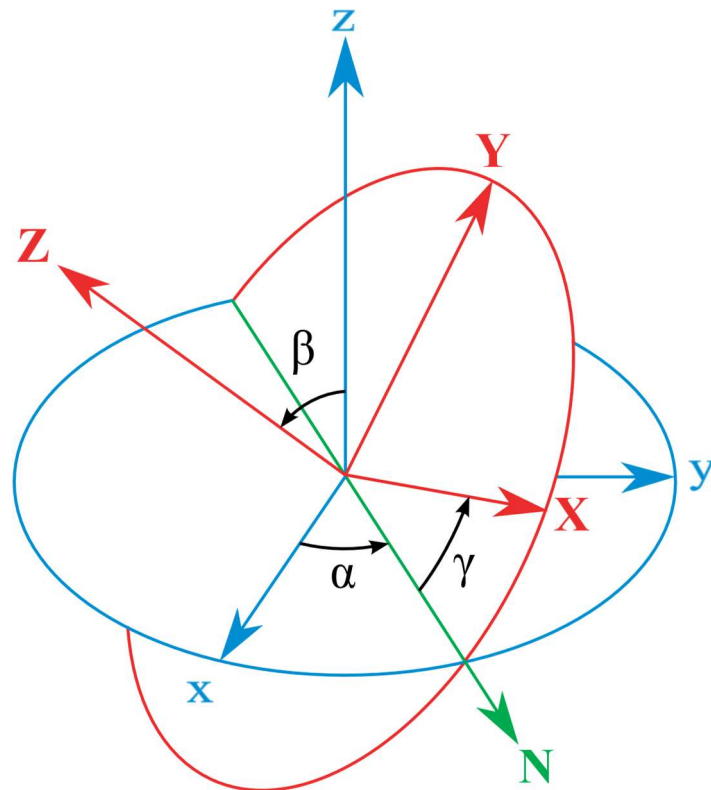


Figura 14: Ángulos eulerianos

Como se ha mencionado anteriormente, Unity 3D representa las rotaciones como “quaternions” o ángulos de Euler, aunque opera internamente con “quaternions”.

En el editor de Unity, los ángulos de Euler son representados por valores aplicados de forma secuencial en la X, Y y Z. Esto quiere decir que una rotación euleriana aplicada a un objeto sufre primero la rotación en el eje x seguida de la rotación en el eje y seguida de la rotación en el eje z. Los ángulos de Euler no son utilizados internamente por Unity porque tienen una limitación conocida como “Gimbal Lock” (bloqueo de cardán) que consiste en la pérdida de un grado de libertad, número de parámetros independientes en un sistema mecánico [11], en el espacio tridimensional que ocurre cuando dos de los ejes se alinean en paralelo de forma que las rotaciones se realizan en un espacio bidimensional [12].

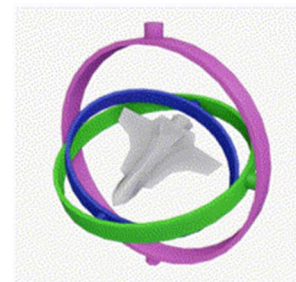


Figura 15: Gimbal Lock

En el caso de los “quaternions”, Unity los representa como cuatro números (x, y, z, w), sin embargo, estos cuatro números no representan los ejes y no se deberían manipular directamente. Un “quaternion” puede representar tanto una orientación como una rotación donde la rotación se mide en base al origen de la rotación o identidad, “quaternion” que representa la no rotación. Como la representación se mide como el paso de una orientación a otra, no puede representar una rotación mayor que 180 grados.

Aunque Unity opere internamente con “quaternions”, el editor muestra los valores como ángulos de Euler, sin embargo, dependiendo del tipo de rotación que se quiera aplicar no se pueden tener en cuenta esos valores ya que existen dos tipos de rotaciones en un objeto: la rotación local y la rotación respecto al mundo (escena) y Unity muestra la rotación global. Ambas rotaciones se almacenan en el “transform” de un objeto, componente que determina la posición, rotación y escala de este.

De cara a la programación en Unity que se realiza normalmente en C# existen múltiples métodos que nos ofrece el propio Unity para evitar manipular “quaternions” directamente. Por otro lado, Unity también nos recomienda evitar transformar “quaternions” a ángulos de Euler, modificar y volver a aplicar a una rotación ya que puede causar efectos secundarios no deseados, efectos que no son mencionados ni explicados. Los métodos propios de Unity para los “quaternions” son los siguientes:

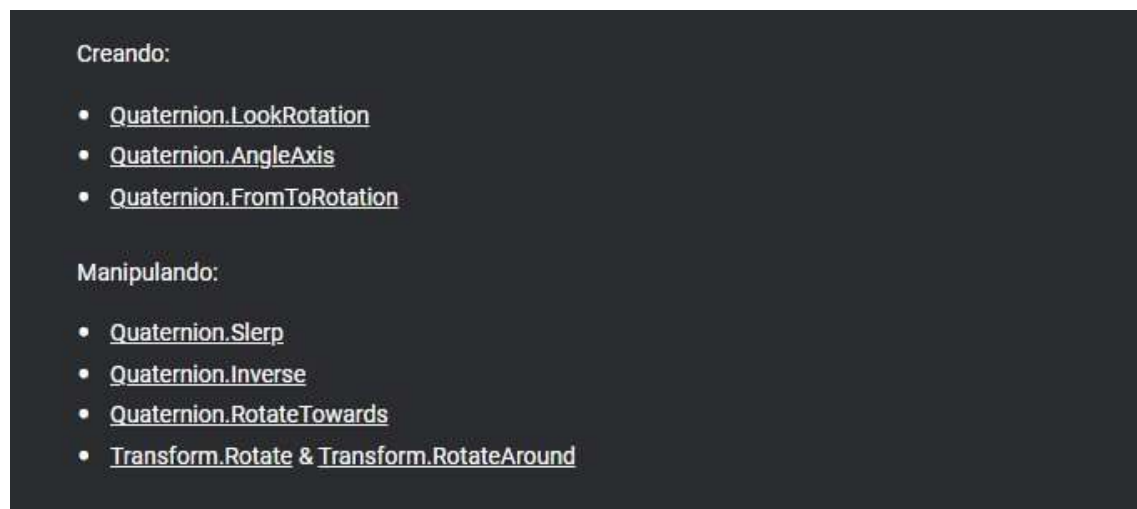


Figura 16: Métodos Quaternions

En la implementación de la herramienta se utilizará alguno de los métodos aquí mostrados y de los que se hablará en puntos de la implementación [13].

10.5. Investigación sobre animación y curvas de animación

En el ámbito de animación, Unity utiliza una clase propia llamada “Animation Clip” que representa los bloques para construir una animación en el entorno de Unity y se importan de los archivos FBX dentro del proyecto [14].

Los clips de animación están conformados por curvas de animación, cualquier propiedad animable puede tener una curva de animación dentro de un clip de animación lo cual significa que los clips controlan como estas propiedades cambian con el tiempo para formar una animación.

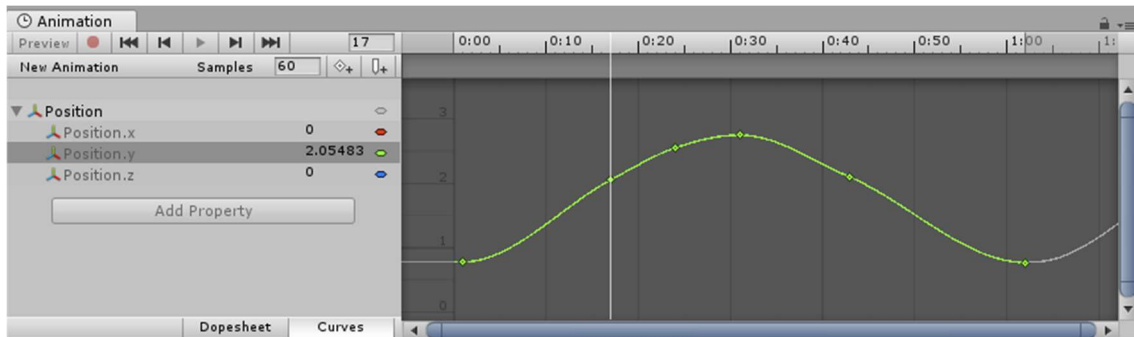


Figura 17: Animation Curve para Position.y

Las curvas de animación están compuestas por un conjunto de “keys” (claves), puntos de control que atraviesa la curva. Si estas llaves coinciden en un “frame” específico de la curva, a ese “frame” se le denomina “keyframe” (“frame” clave).

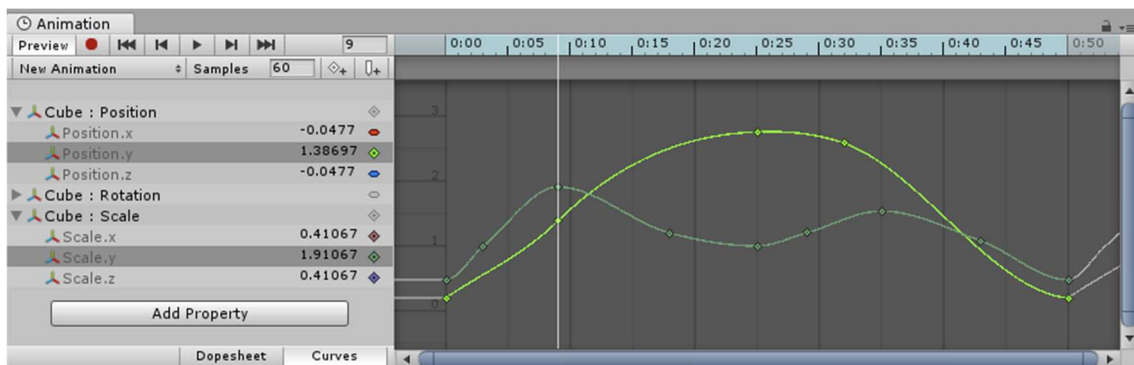


Figura 18: La línea indica un keyframe para las curvas

En el caso de animaciones complejas, hablaremos de animación compleja como cualquiera que contenga más de una curva de animación, se comparten unos keyframes específicos a lo largo de las curvas de animación que normalmente están relacionados con la tasa de refresco a la que se ejecutan las animaciones, es decir, los “frames” por segundo. De esta forma, en cada “keyframe” se actualizan los valores de las propiedades que toman parte en la animación.

A pesar de que internamente y como se ha mencionado previamente, Unity opera con “quaternions” para operar rotaciones, en el caso de las curvas de animación se puede interpolar entre dos rotaciones usando los valores del “quaternion” o los ángulos de Euler.

La interpolación usando los valores de “quaternion” genera animaciones más fluidas y evita el bloqueo de cardán, pero no puede representar rotaciones mayores que 180 grados. Esta será la interpolación que usaremos ya que no se realizan rotaciones

mayores que 180 grados con una tasa de refresco alta como la que suelen tener las animaciones y es muy importante evitar el bloqueo de cardán para obtener animaciones precisas. Las interpolaciones mayores que 180 grados generan interpolaciones más pequeñas ya que por su comportamiento siempre encuentran el camino más corto para realizar la rotación [15].

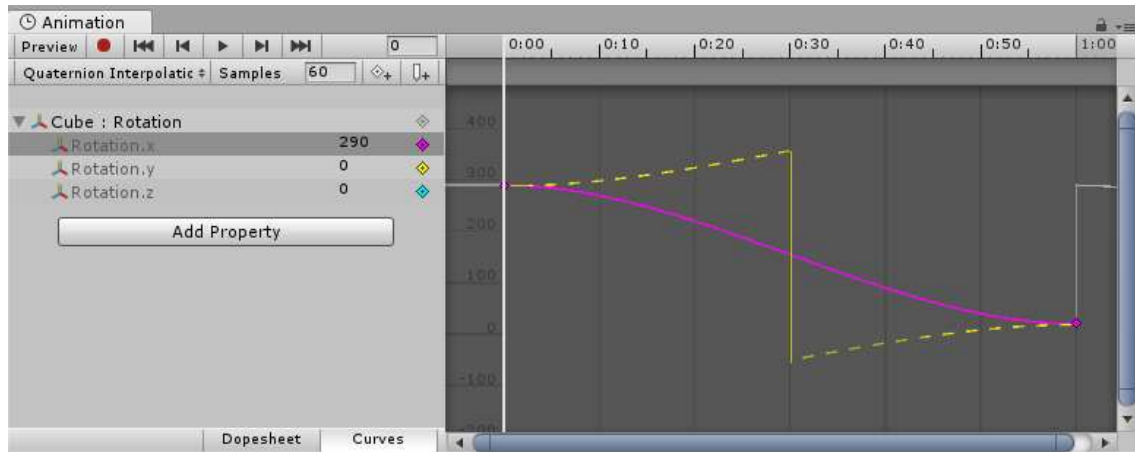


Figura 19: Ejemplo de interpolación con Quaternions mayor que 180 grados

11. Implementación

11.1. Introducción

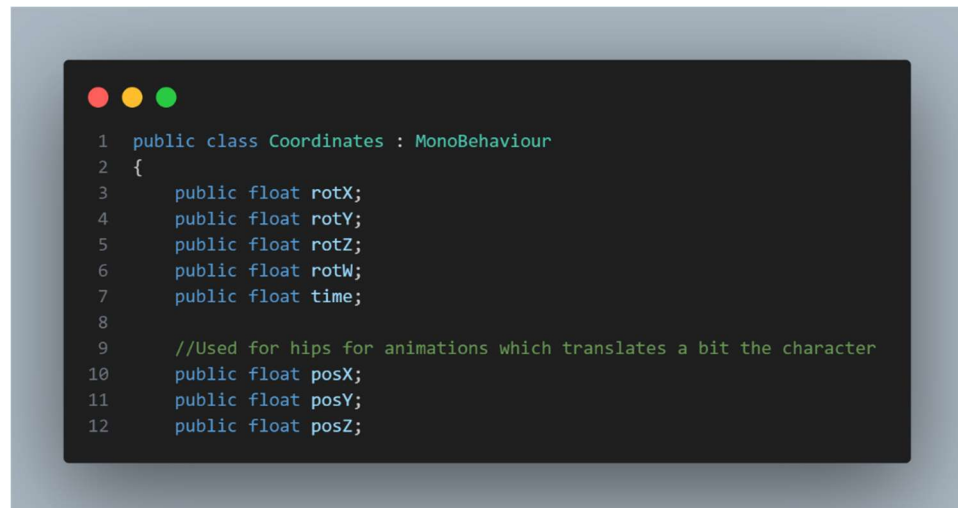
En los próximos apartados se va a describir de forma detallada los pasos, decisiones y desarrollos realizados para completar la implementación de una herramienta software para la edición de animaciones para “retargeting” de personajes y cumplir el objetivo de este proyecto.

Para iniciar el desarrollo del proyecto se requiere de la instalación de “Unity Hub” el cual maneja las licencias y versiones de Unity y el editor (versión) que se desea usar, en este caso, el editor 2021.3.19f1. Por otro lado, se precisa de la instalación de Blender por si debemos realizar cambios extremos en alguno de los modelos 3D y para investigar posibles comportamientos de estos. Por último, se necesita una herramienta de control de versiones para guardar el progreso en la implementación de forma segura, en este caso, se ha usado GitHub.

11.2. Estructuras de datos

La implementación de Unity de los clips de animación mencionados anteriormente hace imposible su lectura ya que Unity no precisa de métodos para leer las curvas o “keyframes” de una animación por lo que para poder editar una animación se llegó a la conclusión de que se necesitaba grabar la animación original para así obtener los datos de rotación de cada uno de los miembros del modelo y modificarlos posteriormente. Esta grabación nos deja con una gran cantidad de datos que manejar y preservar a lo

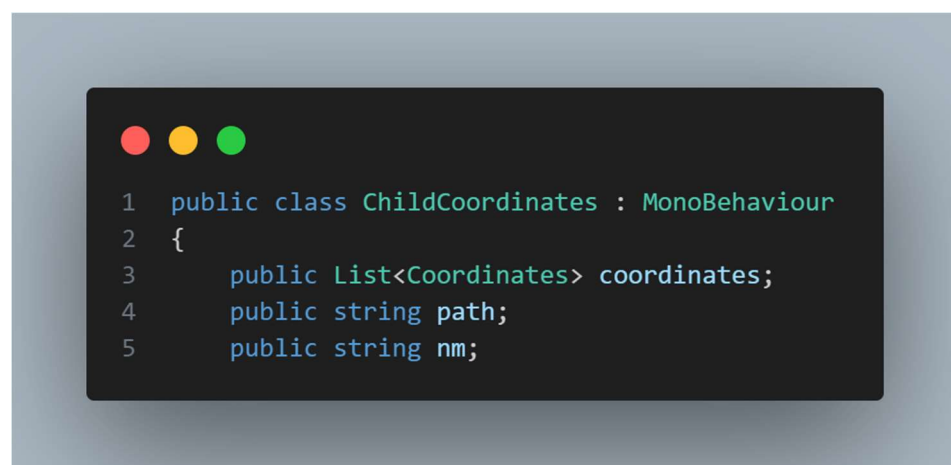
largo de la ejecución de la herramienta de forma que se implementaron dos estructuras de datos para guardar los datos ordenados y poder acceder a ellos de forma sencilla. La primera estructura guarda asociados al instante en el tiempo los datos de rotación y posición si se trata de las caderas, parte del modelo encargado de cambiar de posición si se trata de una animación con movimiento en algún eje ya sea un salto, correr, andar, etc.

A screenshot of a code editor with a dark background and light-colored text. The code defines a C# class named 'Coordinates' that inherits from 'MonoBehaviour'. It contains several public float variables: 'rotX', 'rotY', 'rotZ', 'rotW', 'time', 'posX', 'posY', and 'posZ'. A comment is present for 'posX' indicating its use for hip animations.

```
1 public class Coordinates : MonoBehaviour
2 {
3     public float rotX;
4     public float rotY;
5     public float rotZ;
6     public float rotW;
7     public float time;
8
9     //Used for hips for animations which translates a bit the character
10    public float posX;
11    public float posY;
12    public float posZ;
```

Figura 20: Estructura Coordinates

La segunda estructura almacena para cada hijo (miembro) del modelo, una lista de la primera estructura mencionada junto con el nombre del hijo y su camino, es decir, según la jerarquía del modelo, donde se encuentra el hijo. En otras palabras, la estructura resultante nos permite almacenar para cada hijo los datos de rotación a lo largo del tiempo durante la animación y su posición respectiva a la jerarquía del modelo 3D.

A screenshot of a code editor with a dark background and light-colored text. The code defines a C# class named 'ChildCoordinates' that inherits from 'MonoBehaviour'. It contains three public variables: a 'List<Coordinates>' named 'coordinates', a 'string' named 'path', and a 'string' named 'nm'.

```
1 public class ChildCoordinates : MonoBehaviour
2 {
3     public List<Coordinates> coordinates;
4     public string path;
5     public string nm;
```

Figura 21: Estructura Child Coordinates

11.3. Captura de animación

Se monta una escena inicial para comprobar que la grabación funciona, esta escena consta del modelo objetivo sobre el que se va a reproducir la animación original y el “script” de grabación junto con un plano que simulará un suelo para que se pueda percibir bien la posición del modelo.

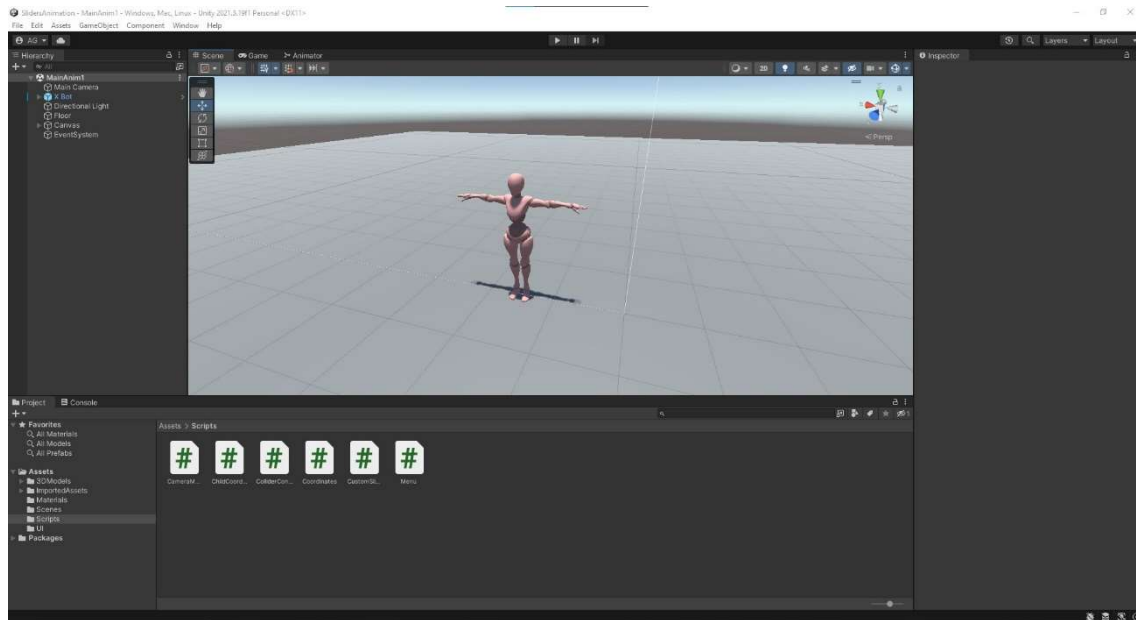


Figura 22: Escena inicial

Se asigna un componente “animation” al modelo para poder reproducir clips de animación sobre el modelo, dentro del componente “animation” introducimos la animación que hemos importado y desde script accederemos a ese componente para activar la reproducción de la animación.

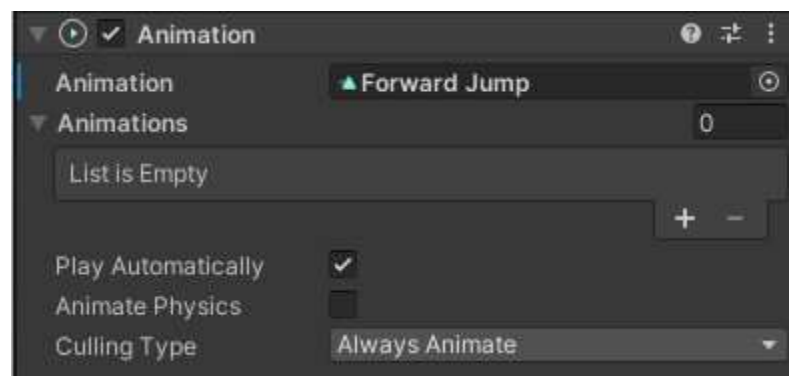


Figura 23: Componente animación del modelo

Antes de realizar la grabación de la animación y de cara al paso siguiente que será el guardado de la animación, se necesita obtener el “path” (camino), es decir, la posición de cada una de las partes del modelo respecto a la jerarquía de este. Para ello, se recorren los hijos del modelo y se ejecuta una función recursiva que construye un

“string” buscando al padre del hijo y cogiendo su nombre y si el padre de este también tiene padre se llama a sí misma hasta que llegamos a la raíz del modelo. Estos caminos se guardan en una estructura auxiliar que se usará para tener referencia de las partes existentes del modelo original y asignarlas al modelo objetivo con sus respectivos datos una vez la animación haya sido grabada con éxito.

De cara a grabar la animación se requiere capturar los datos de los miembros del modelo en los instantes que se desea, para lograr este objetivo se ha hecho uso de una corrutina. Las corrutinas en Unity son funciones que permiten su ejecución a lo largo de una secuencia de tiempo, al contrario que las funciones normales que realizan su ejecución en un solo “frame” [16].

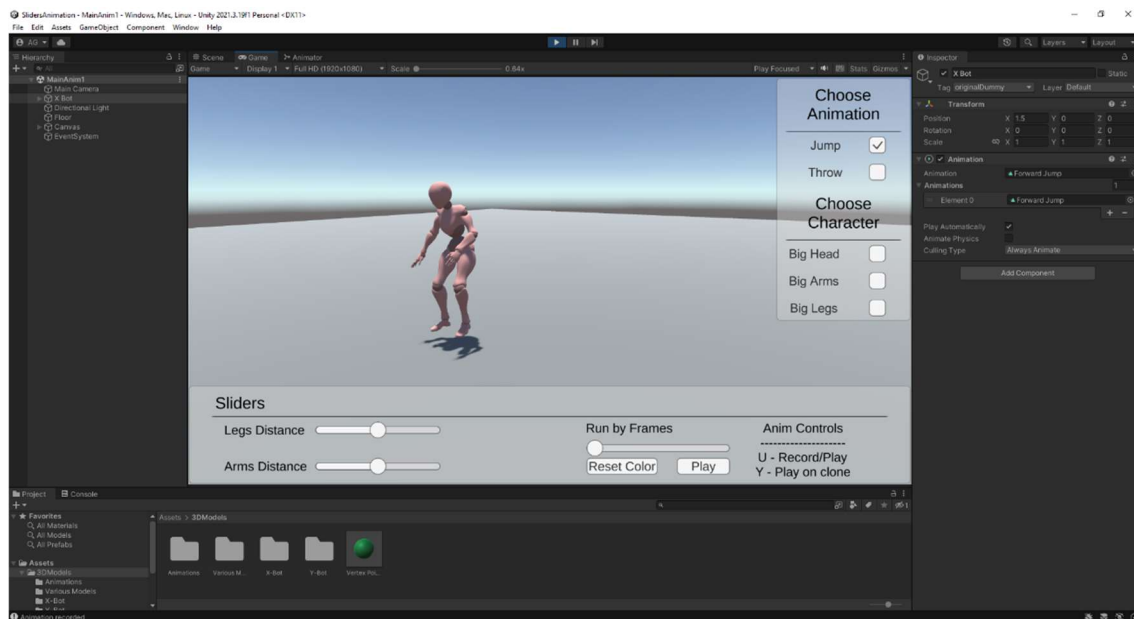


Figura 24: Animación original corriendo

De esta forma, se ha reproducido la animación original al mismo tiempo que empieza la corrutina y cada 0.02 segundos, medida elegida para conseguir 30 datos por segundo, se recorren los miembros del modelo para obtener los datos de las variables x , y , z y w del “quaternion” “localRotation”, es decir, los valores de rotación para crear un “quaternion” con la misma rotación cuando sea requerido. Finalmente se almacenan asociados al instante en el tiempo y a su posición en la jerarquía del modelo en la estructura de datos “ChildCoordinates” que como se mencionó hace uso de la estructura “Coordinates”.

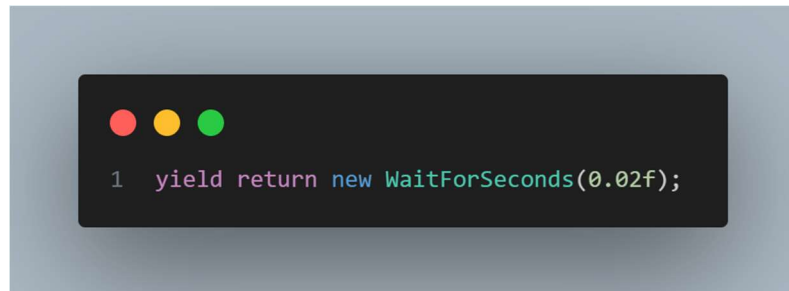


Figura 25: Función ejecutada en la corrutina para grabar datos cada 0.02 segundos

11.4. Guardado de animación

Se necesita guardar los datos como un clip de animación para usar el sistema de reproducción de animaciones propio de Unity de forma que se va a construir un clip de animación con los datos obtenidos de la grabación de la animación original.

El primer paso para construir un clip de animación es declarar un objeto “AnimationClip” vacío con el nombre de la animación que vamos a guardar. A continuación, guardar cada uno de nuestros datos en forma de “keyframes” por lo que vamos a declarar una lista de keyframes para cada dato que se ha grabado, estos son los datos de rotación: *x*, *y*, *z* y *w* del “Quaternion” y los datos de posición: *x*, *y* y *z* en caso de las caderas para, como se mencionó anteriormente, mover el modelo en caso de que se trate de una animación con eventos de traslación.

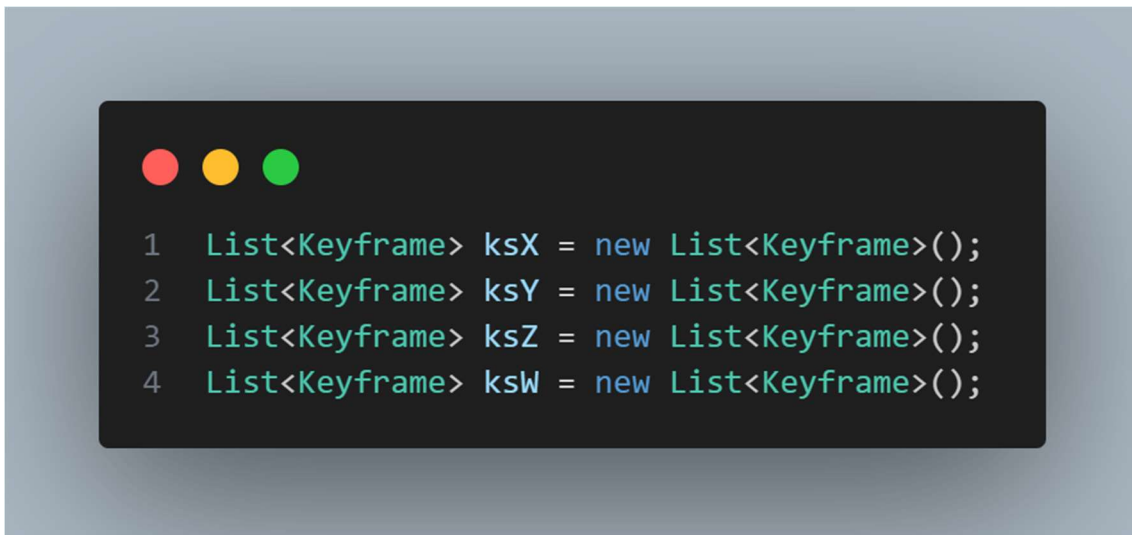


Figura 26: Listas de Keyframes

Cabe destacar que esta declaración (Figura 26) es necesaria para cada miembro del modelo ya que el clip de animación debe conocer todos los datos de todas las partes del modelo para poder realizar una animación fiel y fluida. Después de esta declaración se recorren las estructuras de datos donde inicialmente se almacenó la información de la

animación y se recorren desde el instante 0 de la animación hasta el último instante. Este recorrido se efectúa así porque los datos se deben introducir de forma ordenada.



```
1 List<Keyframe> ksPosX = new List<Keyframe>();  
2 List<Keyframe> ksPosY = new List<Keyframe>();  
3 List<Keyframe> ksPosZ = new List<Keyframe>();
```

Figura 27: Listas de Keyframes adicionales para las caderas

Los “keyframes” se añaden asociados a un instante en el tiempo y a su valor correspondiente.



```
1 ksX.Add(new Keyframe(chCoord.time, chCoord.rotX));
```

Figura 28: Ejemplo de añadir un nuevo keyframe para el instante: chCoord.time con el valor: chCoord.rotX

Una vez ya se han transformado todos los datos a “keyframes” e introducido en sus respectivas listas se crean las curvas de animación que serán las encargadas de pasar de un “keyframe” a otro de forma fluida para completar la animación.



```
1 AnimationCurve curveX = new AnimationCurve(ksX.ToArray());
```

Figura 29: Creación de una curva de animación con el listado de keyframes

Para finalizar, las curvas se añaden al clip de animación que hemos declarado nulo al principio. Las curvas se deben añadir con una referencia a la posición en la jerarquía del modelo de la parte del modelo y el tipo de dato que es.



Figura 30: Curva añadida a clip de animación, Posición del miembro: `chCoords.path`, tipo: `transform localRotation.x`

Si se desea exportar la animación como un “asset” para uso en otros proyectos o como archivo para guardar de forma definitiva se hace uso de un método de Unity para la creación de “assets”.



Figura 31: Creación de archivo .anim

11.5. Reproducción de animación

A la hora de reproducir la animación disponemos de dos formas de hacerlo: con los métodos propios del clip de animación o implementando una reproducción manual basada en transiciones entre rotaciones. Para la implementación de la herramienta se va a hacer uso de ambas con distintas funcionalidades ya que cada una tiene ventajas y desventajas que se comentarán a continuación.

11.5.1. Reproducción con curvas

Este tipo de reproducción es sencillo, el único requerimiento es guardar la animación como un clip de animación, proceso explicado en el apartado anterior, y llamar al método propio “`anim.play()`”.

11.5.2.Reproducción manual

Este tipo de reproducción es más compleja ya que requiere de un entendimiento del funcionamiento de las rotaciones en Unity y la operación con “quaternions”. La visión inicial para desarrollar esta reproducción se basa en, para cada miembro del modelo, aplicar la rotación correspondiente al instante en el tiempo de forma sucesiva hasta completar la animación.

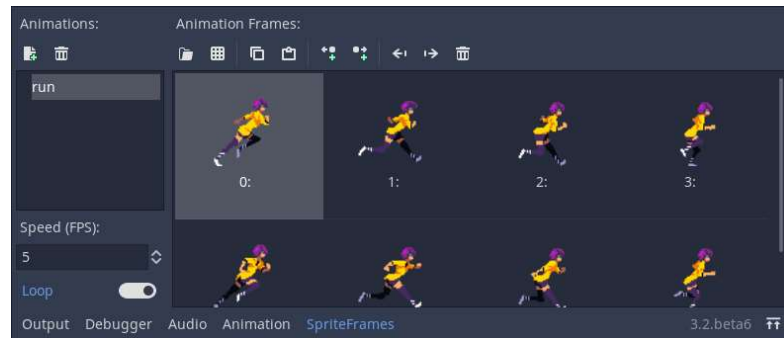


Figura 32: Ejemplo transición entre frames para animación 2D

Al igual que al grabar la animación, se debe tener en cuenta el instante en el tiempo para aplicar las rotaciones en sus intervalos correspondientes y no causar que todas las rotaciones se apliquen en el mismo “frame” resultando en efectos raros en el modelo y desde luego no consiguiendo una reproducción efectiva de la animación. De manera que para tenerlo en cuenta se va a hacer uso nuevamente de una corrutina de Unity, esta corrutina será la encargada de interpolar entre rotaciones.



Figura 33: Corrutina de reproducción


Como se mencionó en el apartado de investigación de “quaternions”, la manera correcta de operar con rotaciones en Unity es usando estos, cualquier manipulación directa de ángulos de Euler provoca efectos extraños en las rotaciones y no resultan en la rotación deseada. Habiendo recordado esto y tras muchas pruebas con los métodos de rotación de Unity se llega a la conclusión de que hay que hacer uso del método “Rotate()”, un método propio del “transform” de un objeto, explicado anteriormente es el encargado

de las rotaciones, posiciones y escalas del objeto. Este método utiliza ángulos de Euler y a que espacio se debe tener en cuenta para hacer la rotación, con estos parámetros se consigue realizar una rotación desde los valores almacenados en el “transform” del objeto hasta los valores pasados por parámetro. Esto puede resultar confuso ya que se había comentado que no se debe operar sobre ángulos de Euler, sin embargo, en este caso no se está realizando ninguna operación sobre ángulos de Euler como se va a ver a continuación con la explicación del proceso.

El proceso que se ha utilizado para interpolar entre dos rotaciones es el siguiente, primero se accede al “transform” deseado, seguidamente se crea un nuevo “quaternion” con los datos almacenados en la estructura de datos “ChildCoordinates” para ese instante. Después se aplica el método “Rotate()” con los ángulos de Euler que resultan del “quaternion” creado, para obtener los ángulos de Euler a partir de un “quaternion” se utiliza una propiedad del “quaternion” que devuelve el valor del “quaternion” traducido a ángulos de Euler lo que implica que no se realiza ninguna operación sobre los ángulos, simplemente se accede a su valor a partir del “quaternion”. Aparte es necesaria una referencia al espacio sobre la que se aplicará la rotación que será a si mismo porque todas las rotaciones se deben realizar de forma local ya que dependen de las rotaciones de sus padres si es que tienen, pero por la forma en la que se establece la jerarquía en el modelo, todos los miembros del modelo dependen de al menos un padre para efectuar sus rotaciones exceptuando del miembro raíz del modelo.

Después y como se trata de una sucesión de rotaciones, hay que tener en cuenta la rotación que se acaba de aplicar por lo que se actualiza el “quaternion” “localRotation” que almacena en el “transform” las rotaciones locales con el nuevo “quaternion”. Si se trata de las caderas se realiza un “Translate()”, método propio del “transform” que realiza una traslación en base a un vector de tres valores indicando las coordenadas y el espacio al que debe hacer referencia la traslación, con los datos almacenados y se actualiza la posición local del transform “localPosition” con esos valores.

Finalmente, este proceso se repite cada 0.02 segundos, es decir, 30 veces por segundo, medida utilizada originalmente para grabar la animación y que se mantiene para no alterar las interpolaciones y generar animaciones extrañas.

A screenshot of a code editor with a dark background and light-colored text. The code is written in C# and is numbered from 1 to 6. It shows the process of finding a child transform, creating a new Quaternion from its rotation values, and then applying a rotation to the child transform using its local rotation.

```
1 Transform child = clone.transform.Find(chCoord.path);
2 Quaternion newRotation = new Quaternion(chCoord.coordinates[i].rotX, chCoord.coordinates[i].rotY,
3   chCoord.coordinates[i].rotZ, chCoord.coordinates[i].rotW);
4
5 child.Rotate(newRotation.eulerAngles, Space.Self);
6 child.localRotation = newRotation;
```

Figura 34: Código para realizar una rotación



```
1 if (chCoord.nm.Equals("mixamorig:Hips"))
2 {
3     child.Translate(new Vector3(chCoord.coordinates[i].posX, chCoord.coordinates[i].posY,
4     chCoord.coordinates[i].posZ), Space.Self);
5     child.localPosition = new Vector3(chCoord.coordinates[i].posX, chCoord.coordinates[i].posY,
6     chCoord.coordinates[i].posZ);
7 }
```

Figura 35: Código para realizar una traslación

La implementación de una reproducción manual nos ayuda a entender el funcionamiento de las rotaciones en Unity y nos aporta una herramienta para reproducir “frames” específicos de la animación sobre el modelo objetivo de tal forma que se pueda observar en profundidad los puntos en los que el “retarget” falla y se debe arreglar. Además, nos permite crear animaciones hasta un “frame” específico de la animación original y poder observar comportamientos más pequeños.

11.6. UI

Para desarrollar el UI se hicieron pruebas con una herramienta propia de Unity llamada “UI Toolkit” basada en el desarrollo de interfaz de usuario para tecnologías web. La herramienta comprendía de un editor con elementos visuales en el que se podía arrastrar cajas y otros elementos para montar la interfaz y prometía que el trabajo desarrollado en la misma podría ser reutilizado ya que se guarda en ficheros uxml que una vez creados y editados con la misma pueden ser importados en otros proyectos para su uso. Como se trata de una herramienta de Unity y se puede usar desde cualquier proyecto a partir de la versión 2019.3 como beta y ya implementada en la versión 2020.3 también permite la modificación y adaptación de ficheros uxml desde cualquier proyecto.

Desafortunadamente, la herramienta presentó problemas al importar la interfaz de usuario desarrollada en un proyecto aparte que se estaba utilizando de prueba para comprobar la capacidad de la herramienta. Aparte, la interfaz no cargaba correctamente y no guardaba las proporciones y, además, partiendo de este problema fue imposible crear un uxml de cero en el proyecto de la herramienta para retargeting.

Después de estos problemas, se tomó la decisión de desarrollar el UI con los objetos que nos presta Unity de forma nativa y sin necesidad de hacer uso de la herramienta “UI Toolkit”.

Con la visión del proyecto en mente y unos diseños previos, se llegó a la conclusión de que se necesitarían “sliders”, botones y “checkbox” para reflejar todas las funcionalidades de la herramienta correctamente.

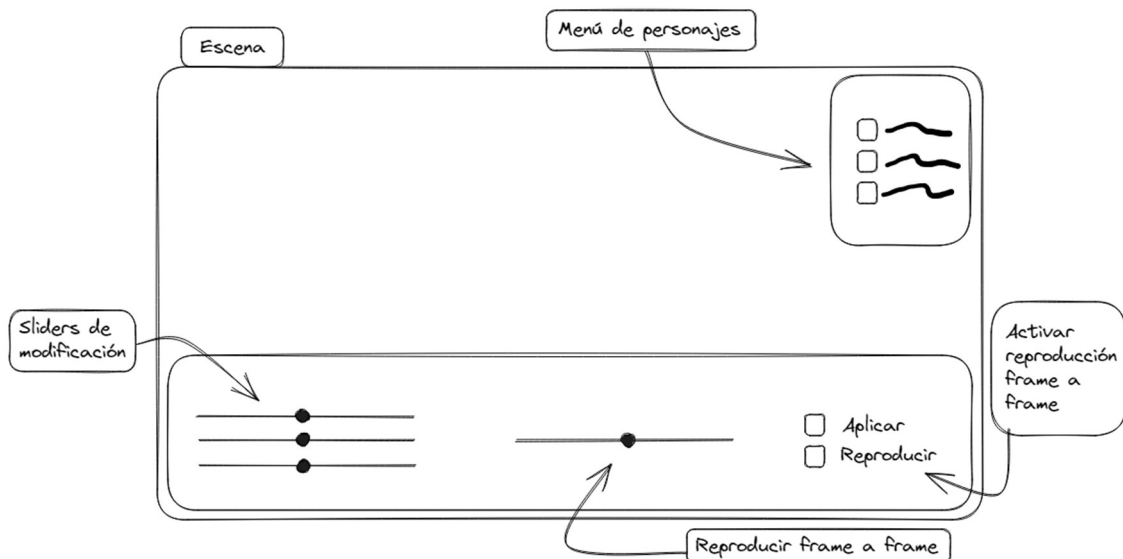


Figura 36: Primer sketch de la interfaz

11.6.1. Configuración de sliders

Para configurar los “sliders” se ha hecho uso de “listeners”, métodos que escuchan a eventos determinados y ejecutan código, para especificar su comportamiento. Se han configurado tres “sliders”, dos encargados de realizar los ajustes de distanciamiento de brazos y piernas y un tercero que se utiliza para reproducir la animación “frame” a “frame”.



Figura 37: Menú de sliders

11.6.2. Menú de elección de personajes

Para escoger entre los tres modelos disponibles (alteraciones exagerando el modelo original) se ha implementado un menú simple basado en “checkboxes” para realizar “clicks” escogiendo el modelo deseado.



Figura 38: Menú de elección de personajes

11.6.3. Menú de elección de animación

Para este menú se ha realizado la misma implementación con la diferencia de que al escoger una animación se carga una escena con la animación seleccionada.

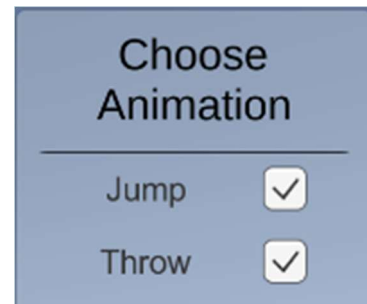


Figura 39: Menú de elección de animación

11.6.4. Interfaz final

La interfaz de usuario resultante del proceso de desarrollo de UI es la siguiente:

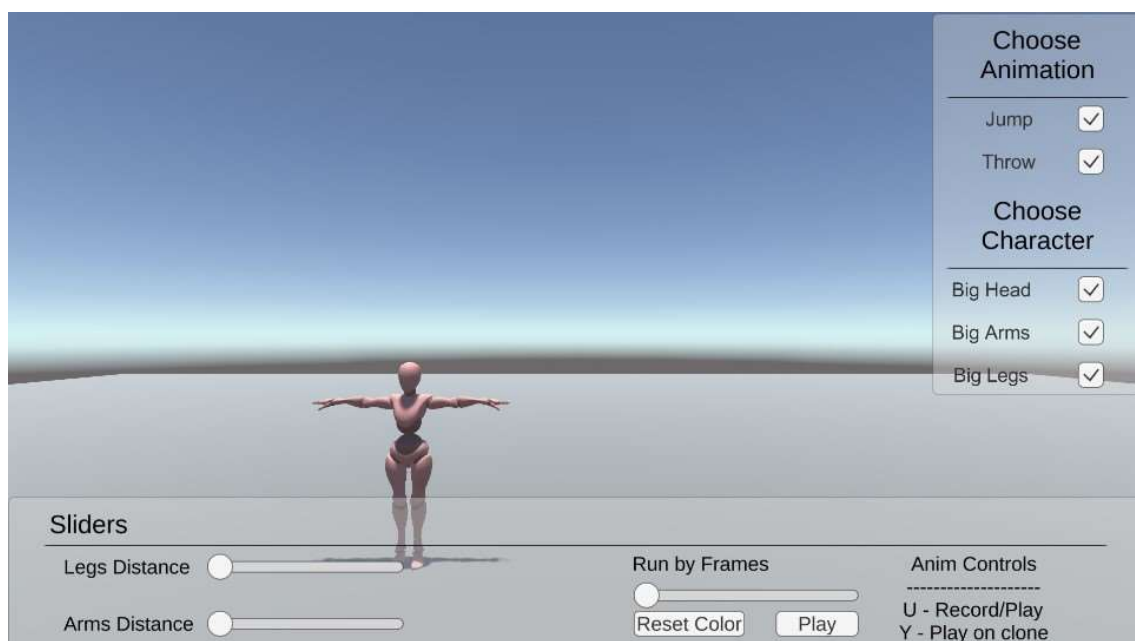



Figura 40: Diseño final de UI

11.7. Modificación de animación

Una vez grabada la animación y habiendo adaptado esta al nuevo modelo, se procede a observar la ejecución y modificarla si es necesario. Para ello se deben modificar los valores de rotación de la animación para aquellos puntos que fallen. Se llegó a la conclusión de que la mayoría de los errores llegaban de interacciones de manos y brazos con las piernas o cabeza y se decidió que se deberían realizar modificaciones en las rotaciones de esos miembros.

Para realizar las modificaciones se han hecho uso de los eventos de escucha de los "sliders" mencionados en el apartado anterior, cuando el usuario desliza el "slider" las

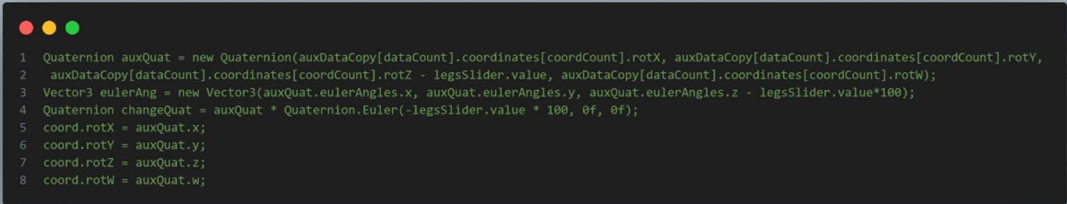
rotaciones se actualizan con el valor actual del “slider”. La modificación se ha realizado actualizando un parámetro del “quaternion”, para conocer que parámetro había que cambiar y en que cantidad se realizaron pruebas hasta que el resultado obtenido eran las modificaciones que se deseaban. Se utiliza una estructura copiada de los datos originales para mantener siempre una copia de los datos originales y no originar cambios bruscos en las rotaciones. De forma que se modifican los datos restando el valor del slider a la copia de los datos originales.



```
1 coord.rotX = auxDataCopy[dataCount].coordinates[coordCount].rotX - legsSlider.value;
```

Figura 41: Modificación del parámetro X restando el valor del slider

De igual forma también se pueden modificar los datos, creando un “quaternion” con los datos originales y un “quaternion” con los valores originales modificados y operar multiplicando los dos “quaternions”, la multiplicación es la operación correcta para sumar rotaciones.



```
1 Quaternion auxQuat = new Quaternion(auxDataCopy[dataCount].coordinates[coordCount].rotX, auxDataCopy[dataCount].coordinates[coordCount].rotY,  
2 auxDataCopy[dataCount].coordinates[coordCount].rotZ - legsSlider.value, auxDataCopy[dataCount].coordinates[coordCount].rotW);  
3 Vector3 eulerAng = new Vector3(auxQuat.eulerAngles.x, auxQuat.eulerAngles.y, auxQuat.eulerAngles.z - legsSlider.value*100);  
4 Quaternion changeQuat = auxQuat * Quaternion.Euler(-legsSlider.value * 100, 0f, 0f);  
5 coord.rotX = auxQuat.x;  
6 coord.rotY = auxQuat.y;  
7 coord.rotZ = auxQuat.z;  
8 coord.rotW = auxQuat.w;
```

Figura 42: Modificación de rotación usando multiplicación de quaternions

Ambas operaciones dan el mismo resultado así que se opera con la primera por su fácil implementación y uso. El evento de escucha del “slider” es el encargado de buscar las partes del modelo correspondientes y realizar este cambio en todos los “frames” de la animación para conseguir así el comportamiento deseado que en este caso es ampliar o disminuir la distancia entre brazos y entre piernas. Después de realizar el cambio se construyen los “keyframes” de nuevo y las curvas de animación actualizadas para poder reproducir la animación actualizada haciendo uso de los métodos de reproducción de animaciones de Unity.

11.8. Cámara y movimiento

De cara a tener una mayor visual de la ejecución, se pensó que sería conveniente que el usuario pudiera navegar por el espacio 3D para observar la animación desde distintos ángulos y comprender en que puntos hay que corregir la animación. La mejor solución a esto es implementar movimiento en la cámara para que el usuario pueda rotar y moverse cómodamente por la escena.

Se implementó un sencillo “script” de movimiento y rotación para la cámara en base al input del ratón para la rotación y el input de las teclas para el movimiento, además, para evitar usos incorrectos del UI y como Unity permite mover “sliders” con esas teclas, se implementó un bloqueo de cámara para usar el UI sin problemas. Este movimiento será explicado más en detalle en el manual.

11.9. Detección de colisiones

Para facilitar la detección de errores en el “retarget” de la animación y mejorar la experiencia de usuario se determina que debe haber algún elemento visual que indique estos fallos. Como estos fallos son colisiones entre las distintas partes del modelo y focalizando en los brazos, cabeza y piernas que son los principales causantes se llega a la conclusión que hay que hacer uso de “colliders”, esto son componentes que detectan estas colisiones y que nos brinda Unity para acompañar a los objetos.

Los modelos están formados por una malla que define su superficie, esta malla está hecha con vértices que forman triángulos dando la forma final al modelo. Existe un componente “collider” denominado “mesh collider”, este utiliza la malla del modelo para crear un “collider” sobre su superficie.

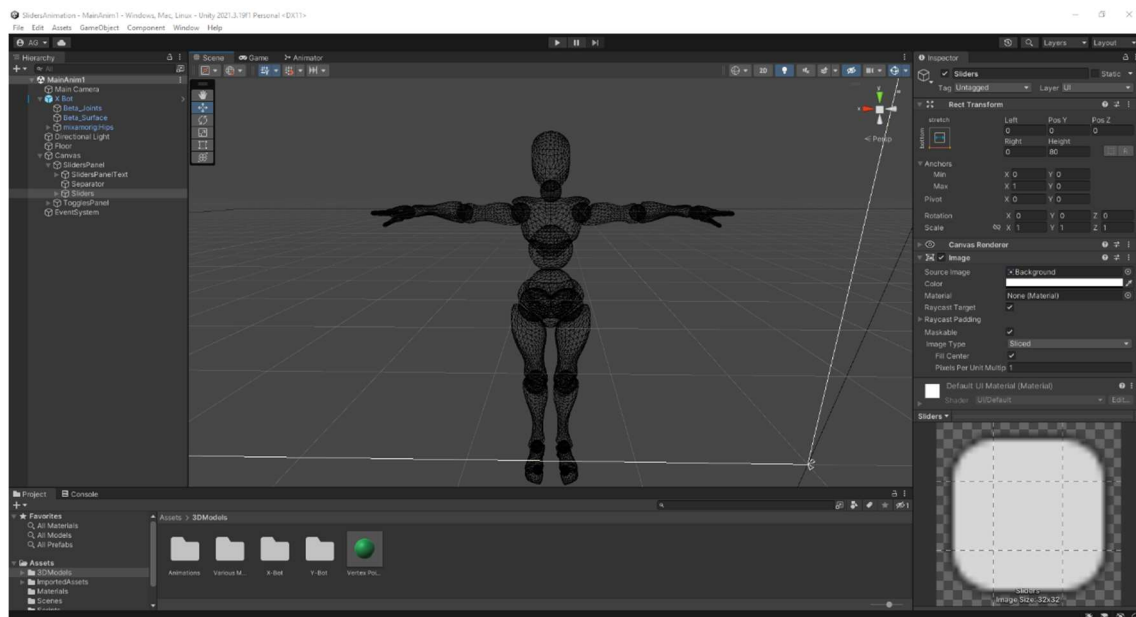


Figura 43: Malla del modelo

La malla conforma todo el modelo y se puede dividir en submallas pero para ello se necesitan saber los índices de vértices exactos y calcular los triángulos y en que orden deben formarse estos. Para evitar la complejidad de esa tarea se pensó que se podían usar las posiciones de los vértices que deseemos para instanciar pequeñas esferas con “collider” y “rigidbody” para que detecten las colisiones entre ellas. Estas esferas se mueven con el modelo y se sitúan en las posiciones de los vértices en manos, cabeza, pies y piernas.

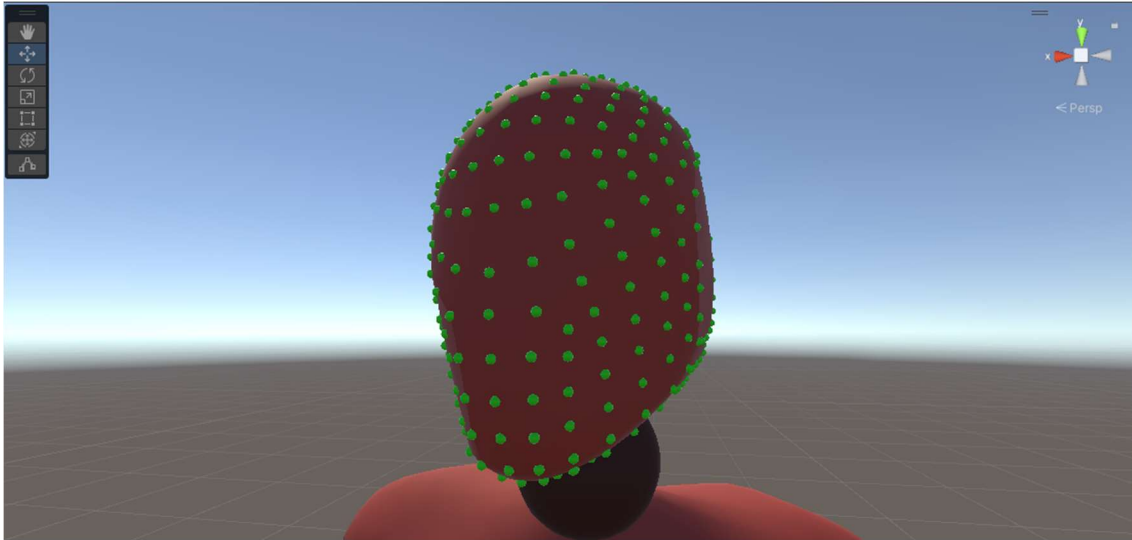


Figura 44: Esferas rodeando la cabeza actuando de detectores de colisión

Los índices de los vértices de la malla no siguen ningún orden lógico y por tanto es imposible automatizar el proceso de selección de esos índices de cara a ampliar la detección de colisiones en cualquier modelo. Se realizó un proceso manual en base a pruebas, ejecuciones e iteraciones sobre el índice para conseguir los índices exactos necesarios para realizar el seguimiento de colisiones que se deseaba.

```
1  /*RIGHT HAND*/
2      // From 7230 to 7826 --> Right Hand Thumb --597
3      // From 7230 to 7427 --> Right Hand Thumb 3 --198
4      // From 7427 to 7629 --> Right Hand Thumb 2 --201
5      // From 7629 to 7827 --> Right Hand Thumb 1 --198
6      // From 9893 to 1065 --> Right Hand Ring 3 --172
7      //From 10453 to 10625 --> Right Hand Pink 3 --172
8      //From 11013 to 11185 --> Right Hand Middle 3 --172
9      //From 11573 to 11767 --> Right Hand Index 3 --194
10     //From 7100 to 7230 and 12133 to 12330 --> Right Hand Palm
```

Figura 45: Lista de índices para la mano derecha recabada manualmente

Finalmente se consigue que las esferas adheridas al modelo en esos puntos específicos detecten cuando ocurre una colisión entre partes del modelo y actúen de indicador visual para avisar al usuario que debe usar los “sliders” para modificar el “retarget”.

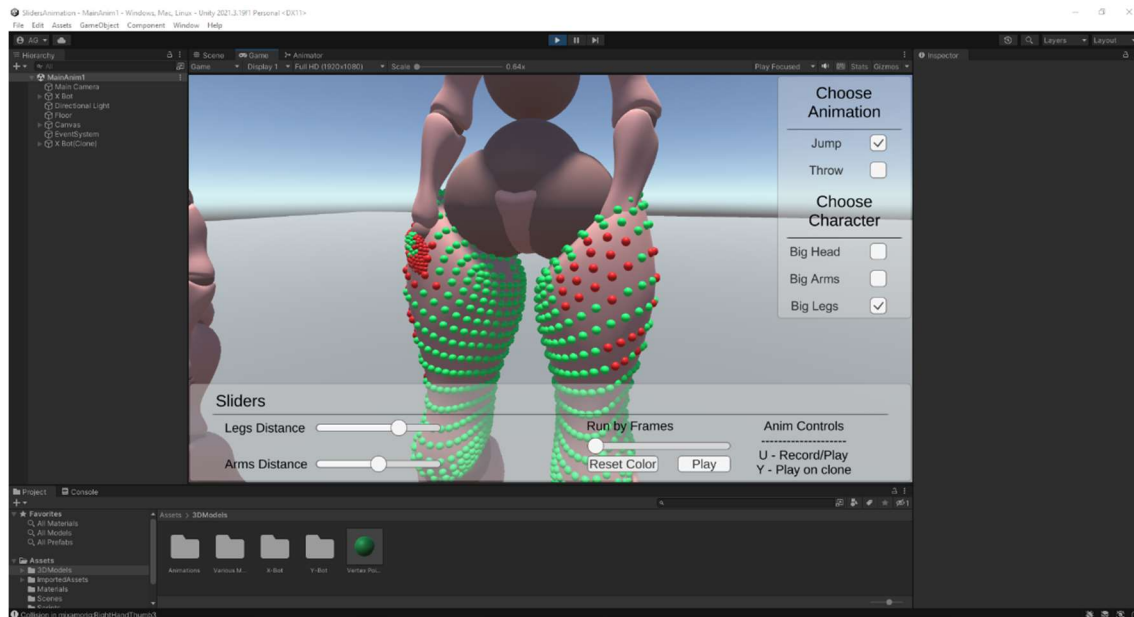


Figura 46: Las esferas rojas indican colisión

Cabe destacar que para que dos objetos con “rigidbody” y “collider” colisionen entre ellos se ha tenido que especificar que el “rigidbody” sea “kinematic”, es decir, que el “rigidbody” está bajo control directo del script de animación y no le afectan fuerzas [17]. Además, se deben permitir en las preferencias del proyecto que se detecten colisiones entre “colliders” y “kinematic rigidbodies”.

12. Problemas encontrados

12.1. Documentación de Scripting API Unity 3D

El principal problema encontrado a lo largo de todo el desarrollo de la herramienta y principal causa de frustraciones ha sido sin duda la documentación de Unity en lo referido a la programación. Quiero pensar que ha sido en gran parte por la naturaleza del proyecto ya que existe muy poca, casi nula documentación de cómo tratar animaciones desde “script” y todo lo relacionado con ello. La mayoría de los ejemplos son demasiado sencillos y no cubren el objetivo de los métodos presentes en la API. Esto en parte se debe a que Unity no nace con la idea de manipular animaciones desde script, Unity ya tiene de forma nativa una herramienta de edición de animaciones desde el propio editor y con su foco en el desarrollo de videojuegos, esta herramienta es un complemento perfecto para proveer de animaciones a personajes para implementar un videojuego ya sea 2D o 3D. Unity no plantea estos desafíos como suyos ya que sabe que herramientas como Blender, software open source, con características más propias de

una herramienta de modelaje y animado 3D, suelen ser parte del repertorio de un desarrollador de videojuegos.

12.2. Modificación de rotaciones

Desafortunadamente y por la naturaleza de nuestro objetivo, realizar una edición del retargeting para ajustar correctamente una animación a nuevos modelos, la edición de la animación requiere de la manipulación de “quaternions” de forma manual ya que no se conoce el “quaternion” objetivo para realizar una rotación de “quaternion” a “quaternion”. Esto incumple las recomendaciones de Unity, pero junto con la poca documentación sobre el “scripting” de animaciones y su modificación, esta escena se convierte en una prueba perfecta sobre el comportamiento de “quaternions” a la hora de ser modificados para alterar una animación. A lo largo del desarrollo se han encontrado problemas con las rotaciones: a la hora de referirse a los miembros del modelo se llegó a la conclusión de que se necesitaba el camino completo según la jerarquía del modelo, a la hora de realizar las rotaciones se debían hacer de forma local para tener en cuenta las rotaciones padre, los ángulos de Euler del editor, por mucho que insista la documentación en decir que representan los ángulos de Euler devueltos por “script”, no es así y un largo etcétera.

12.3. Reproducción manual de animación

El primer problema encontrado a la hora de reproducir la animación es que al igual que en la grabación, se debía tener en cuenta los instantes en el tiempo en el que se aplicaban las rotaciones. Ambos problemas se solucionaron al mismo tiempo con el uso de las ya mencionadas corrutinas de Unity. Una vez solucionado este problema se encontraron más a raíz de los métodos de rotación de Unity, realizando pruebas tanto con métodos de rotación propios del objeto “quaternion” con los cuales, ninguna rotación era la deseada y el personaje se doblaba de formas extrañas en vez de realizar la animación original como con los métodos propios del “transform”. Con este último se observó que las rotaciones tenían sentido, pero no llegaban a ser las deseadas para replicar la animación original hasta que se averiguó que se debían usar las rotaciones locales para tener en cuenta las rotaciones padres.

12.4. Pathing de hijos de modelo 3D

El único problema encontrado con relación a este tema fue que el “pathing” de los modelos es distinto dependiendo del modelaje de este y por tanto no se puede aplicar la herramienta a modelos externos a mixamo.

12.5. Rigging modelo 3D

Existe un problema inherente al modelo 3D relacionado con esta herramienta, pero primero para contextualizar cada modelo 3D se suele designar en la pose conocida como T-Pose para que al animar el modelo siempre se parta desde la misma base, esto además facilita enormemente el “retargeting”. Sin embargo, los valores de rotación de una T-

Pose pueden cambiar dependiendo del modelaje y los estándares con los que se ha realizado esa T-Pose, esto supone una gran pega a la hora de escoger modelos cuyos valores de rotación de la T-Pose pueden ser distintos, una alteración de los valores originales supone un mal funcionamiento de la herramienta. Esto se explica ya que los valores que usa la herramienta son los originales de la animación, si existe un “offset” respecto al valor inicial de la T-Pose se entiende que ese mismo “offset” se debe aplicar a las rotaciones de la animación y así se intentó desarrollar por si algún modelo de mixamo tuviera valores originales de T-Pose muy diferentes. El desarrollo dio resultados incoherentes y que no respondían a la lógica de aplicar el “offset” mencionado, sin embargo, se comprobó que todos los modelos de mixamo tienen prácticamente los mismos valores de rotación en la T-Pose.

Los valores de la pose T-Pose pueden variar entre modelos porque son valores que se fijan en el proceso de modelaje 3D, en el caso de Blender, se utiliza una herramienta para congelar los valores y rotar el modelo sin alterarlos, de esta forma se consigue la T-Pose sin alterar los valores de rotación del modelo.

12.6. Creación malla 3D para detección

El principal problema en la creación de una malla 3D ajustada a las partes del modelo que se quieren observar es que hay que partir de la malla original del modelo y de ahí separar vértices y sus correspondientes triángulos que conforman la malla en esos lugares y adaptarlos. Esto es un proceso muy complejo y prácticamente imposible de realizar por la complejidad de los índices de vértices y triángulos y sus relaciones.

13. Posibles próximos objetivos

La herramienta tiene muchas posibilidades de desarrollo a futuro y con más tiempo de investigación e implementación se podría ampliar el alcance de la herramienta, entre los posibles próximos objetivos y que creo serían buenos caminos para continuar el desarrollo están:

- Menú para crear un modelo exagerado al que ajustar las animaciones
- Que sirva para cualquier modelo (bastante difícil)
- Malla automática para detectar colisiones para cualquier modelo (bastante difícil)

Hablando de los posibles objetivos que he marcado como difíciles, en el caso de ampliar la herramienta para cualquier modelo, es una tarea muy difícil de completar ya que existen muchas variables a tener en cuenta sobre un modelo 3D, además no existe un estándar para modelar personajes humanoides 3D en el mundo del modelaje lo cual dificulta aún más encontrar un estándar para ajustar animaciones entre modelos con distinta jerarquía y política de nombrado de hijos.

En el caso de la malla automática, cada modelo 3D suele llevar asociado una malla que cubre la totalidad del modelo, y la malla está compuesta de vértices que no siguen ningún orden lógico en su colocación, es decir, no se puede identificar por el índice de vértices donde estaría localizado sobre el modelo, por tanto, diseñar mallas específicas para miembros del modelo que se quieran observar es muy complicado y requeriría del desarrollo de una IA que fuese capaz de identificar la posición de cada vértice y a que parte del modelo pertenece para asociarlos y construir una malla a partir de ellos.

14. Manual

La herramienta se controla con las teclas **W, A, S, D** o las **flechas de teclado**, con la tecla **U** e **Y** y con el **ratón** para rotar la cámara e interactuar con el **UI**.



Figura 47: Teclas usadas para controlar la herramienta

El primer paso para utilizar la herramienta es elegir la animación en la que se quiere editar el “retarget” en este caso el salto y grabarla sobre el personaje original.

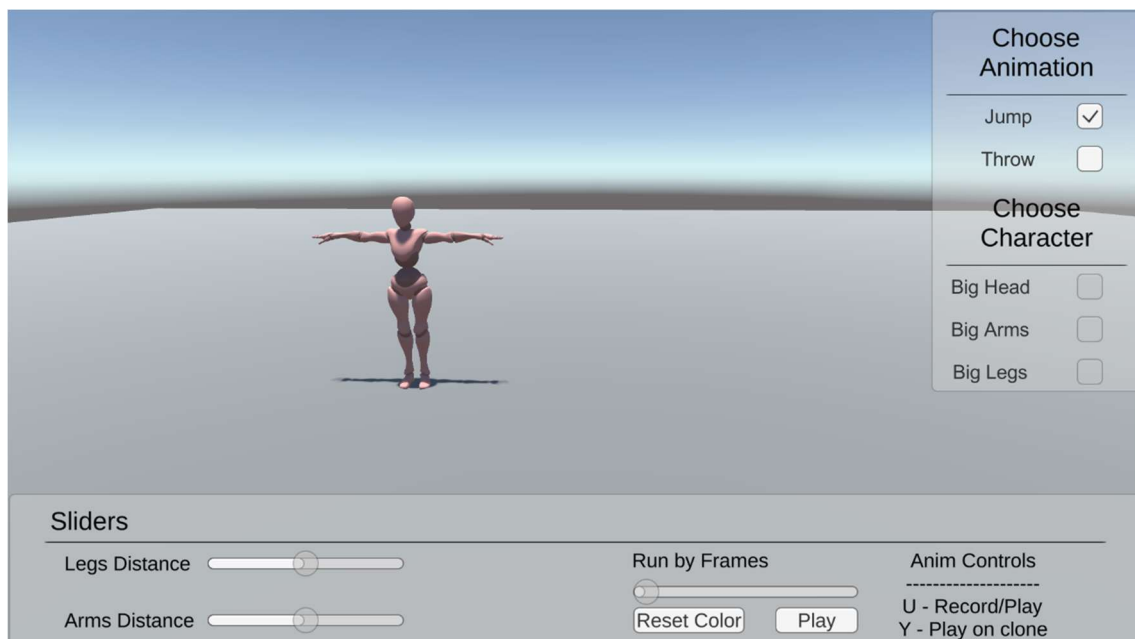


Figura 48: Herramienta construida

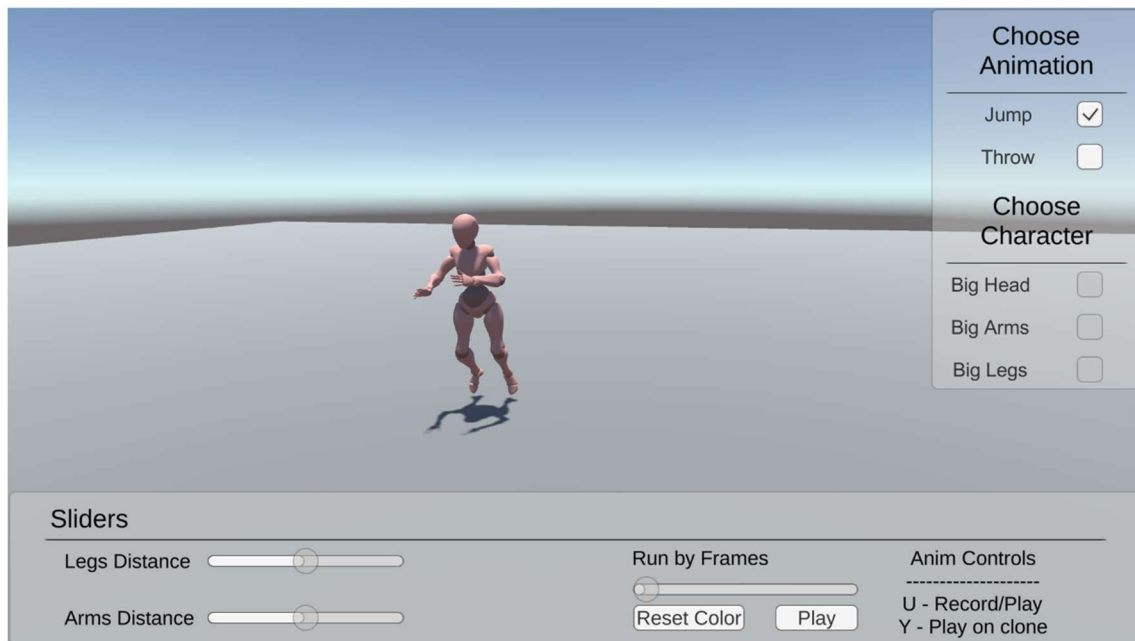


Figura 49: Grabando animación original

Después se instancia el modelo objetivo usando el menú de personajes, por ejemplo el personaje de piernas largas.

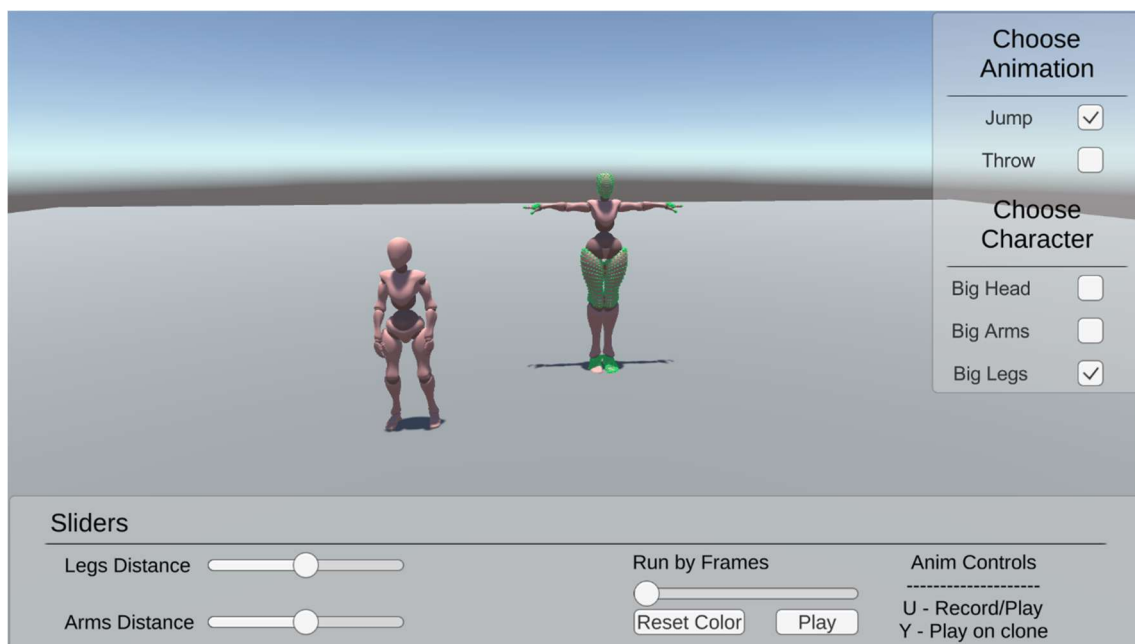


Figura 50: Modelo objetivo del retarget instanciado

A continuación, con ayuda de la Y para reproducir la animación y la reproducción por “frames”, se evalúa el comportamiento de la animación sobre el modelo objetivo. Si se presentan colisiones, serán visibles gracias a la detección de colisiones implementada.

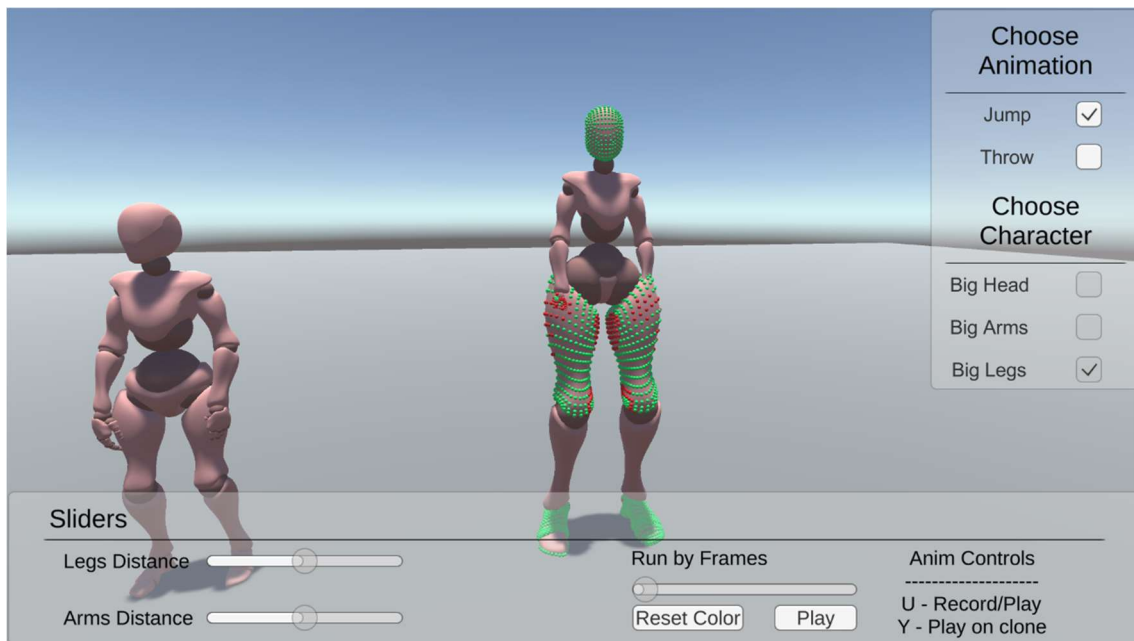


Figura 51: Colisiones (fallos) detectados en el retarget

Tras reproducir la animación se observan colisiones de las manos con los muslos del personaje (atraviesan los muslos). También se detectan colisiones entre los muslos.

Para arreglarlo se ajustan los valores de distanciamiento entre piernas y entre brazos usando los “sliders” correspondientes. Se observa que, con los nuevos valores, se ha ajustado el “retargeting” correctamente y se han eliminado las colisiones/errores que surgían al reproducir la animación sobre este modelo.



Figura 52: Retargeting ajustado

En este caso no ha sido necesaria la ayuda de la funcionalidad de reproducción por “frames” debido a la simplicidad de las colisiones encontradas, aun así, es una herramienta muy útil para observar más de cerca en que momentos ocurren las colisiones y ajustar el “retarget” de forma más precisa.

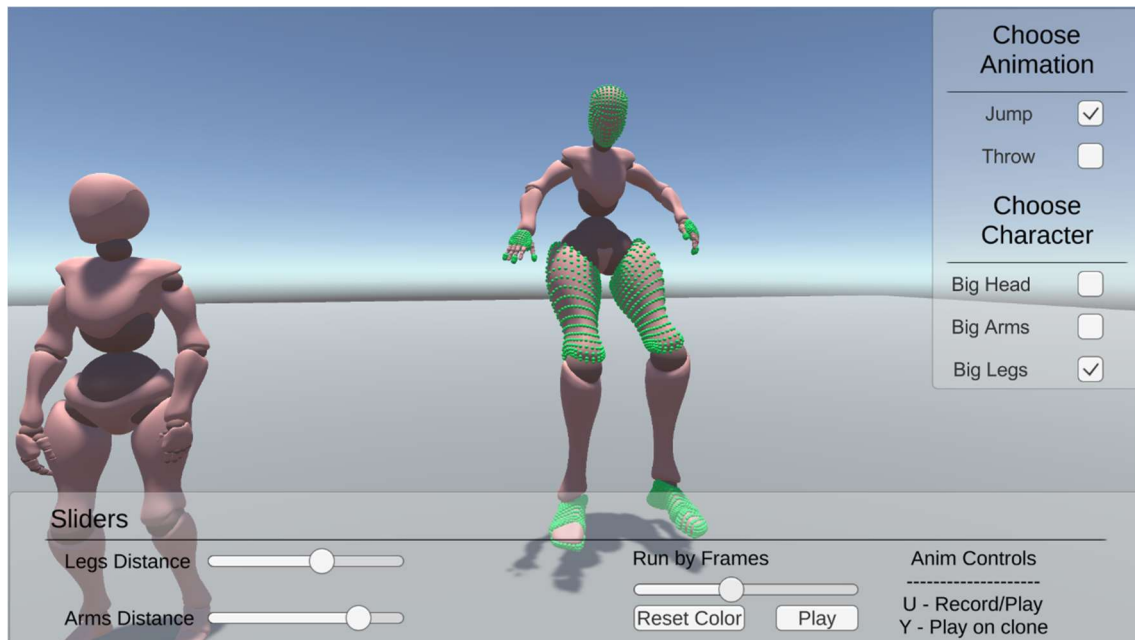


Figura 53: Ejemplo de uso de la funcionalidad de reproducción por frames

Siguiendo estos sencillos pasos, se obtiene un ajuste del “retarget” de la animación deseada.

15. Conclusión

La implementación de la herramienta se ha completado con éxito no sin encontrar bastantes problemas por el camino, que tiene muchos frentes posibles por donde continuar ampliando sus funcionalidades en un futuro, sin embargo, Unity no es el mejor entorno para desarrollar herramientas con estos objetivos ya que como he comentado en otros apartados, Unity no está preparado para implementaciones en este ámbito.

En conclusión, se trata de una herramienta muy interesante y de gran utilidad, con mucho potencial y que con ayuda de algún software externo de modelaje podría convertirse en una herramienta esencial para el desarrollo de videojuegos.

16. Referencias

[1] Unreal Engine 5. Animation Retargeting. (n.d.).

<https://docs.unrealengine.com/5.0/en-US/animation-retargeting-in-unreal-engine/>

[2] What is the current size of the global animation market? (n.d.). Retrieved May 22, 2023, from <https://www.precedenceresearch.com/animation-market>

- [3] NVIDIA Blogs. ¿Qué es Animation Retargeting? (2022, June 30).
<https://la.blogs.nvidia.com/2022/06/30/que-es-animation-retargeting/>
- [4] Unity (motor de videojuego). (n.d.).
[https://es.wikipedia.org/wiki/Unity_\(motor_de_videojuego\)](https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego))
- [5] Blender. (n.d.). <https://es.wikipedia.org/wiki/Blender>
- [6] Learn game development w/ Unity | Courses & tutorials in game design, VR, AR, & Real-time 3D | Unity Learn (n.d.). <https://learn.unity.com/>
- [7] Basis (linear algebra) (n.d.) [https://en.wikipedia.org/wiki/Basis_\(linear_algebra\)](https://en.wikipedia.org/wiki/Basis_(linear_algebra))
- [8] Euler angles (n.d.). https://en.wikipedia.org/wiki/Euler_angles
- [9] Rotation matrix (n.d.). https://en.wikipedia.org/wiki/Rotation_matrix
- [10] Quaternion (n.d.). <https://en.wikipedia.org/wiki/Quaternion>
- [11] Degrees of freedom (mechanics). (n.d.).
[https://en.wikipedia.org/wiki/Degrees_of_freedom_\(mechanics\)](https://en.wikipedia.org/wiki/Degrees_of_freedom_(mechanics))
- [12] Gimbal Lock (n.d.). https://en.wikipedia.org/wiki/Gimbal_lock
- [13] Rotación y Orientación en Unity. (Copyright © 2020 Unity Technologies. Publication 2019.4).
<https://docs.unity3d.com/es/2019.4/Manual/QuaternionAndEulerRotationsInUnity.html>
- [14] Animation Clip (Clip de animación). (© 2016 Todos los derechos reservados. Unity Technologies. Publication 5.3-Q). <https://docs.unity3d.com/es/530/Manual/class-AnimationClip.html>
- [15] Using Animation Curves. (Copyright © 2021 Unity Technologies. Publication Date: 2023-05-12.). <https://docs.unity3d.com/Manual/animeditor-AnimationCurves.html>
- [16] Technologies, U. (n.d.). Coroutines (corrutinas). Unity Manual. Retrieved May 23, 2023, from <https://docs.unity3d.com/es/2018.4/Manual/Coroutines.html>
- [17] Technologies, U. (n.d.). Unity. Scripting API: Rigidbody.isKinematic. Retrieved May 30, 2023, from <https://docs.unity3d.com/ScriptReference/Rigidbody-isKinematic.html>
- Figura 1: What is the current size of the global animation market? (n.d.). Retrieved May 22, 2023, from <https://www.precedenceresearch.com/animation-market>

Figura 3: De Unity Technologies -

<https://brandguide.brandfolder.com/unity/downloadbrandassets>, Dominio público,
<https://commons.wikimedia.org/w/index.php?curid=110761106>

Figura 4: De TM/[®]Blender Foundation - Vectorised by Vulphere from

<http://www.blender.org/> (SVG code), Dominio público,
<https://commons.wikimedia.org/w/index.php?curid=35129654>

Figura 5: By Mixamo - personally, CC BY-SA 3.0,

<https://commons.wikimedia.org/w/index.php?curid=34638391>

Figura 8: By Maschen - Own work, CC BY-SA 3.0,

<https://commons.wikimedia.org/w/index.php?curid=27793856>

Figura 9: By Lionel Brits - Hand drawn in Inkscape by me, CC BY 3.0,

<https://commons.wikimedia.org/w/index.php?curid=3362239>

Figura 10: By Drummyfish - Own work, CC0,

<https://commons.wikimedia.org/w/index.php?curid=77738933>

Figura 11: By Unity Documentation

<https://docs.unity3d.com/es/2019.4/Manual/QuaternionAndEulerRotationsInUnity.html>

Figura 12: By Unity Documentation <https://docs.unity3d.com/Manual/animeditor-AnimationCurves.html>

Figura 13: By Unity Documentation <https://docs.unity3d.com/Manual/animeditor-AnimationCurves.html>

Figura 14: By Unity Documentation <https://docs.unity3d.com/Manual/animeditor-AnimationCurves.html>

Figura 32: Animación de Sprite 2D. (n.d.). Godot Engine Documentation. Retrieved May 29, 2023, from

https://docs.godotengine.org/es/stable/tutorials/2d/2d_sprite_animation.html