Universidad Zaragoza
1542

Trabajo Fin de Grado

"Creación de un entorno interactivo para la manipulación de modelos en 3D mediante realidad virtual y destinado para fines industriales"

"Creation of an interactive environment for the manipulation of 3D models by means of virtual reality for industrial purposes"

MEMORIA

Autor/es
Pablo Díez Ruiz

Director/es
David Ranz Angulo
Ramón Miralbés Buil

Grado en Ingeniería en Diseño Industrial y Desarrollo del Producto
Departamento de Expresión Gráfica
Escuela de Ingeniería y Arquitectura

2020-2021

Resumen del proyecto

"Creación de un entorno interactivo para la manipulación de modelos en 3D mediante realidad virtual y destinado para fines industriales"

En este proyecto de fin de grado he creado un entorno virtual, al cual accederemos a el mediante realidad virtual, en donde el usuario tendrá la posibilidad de:

- Moverse libremente por la totalidad del espacio creado.
- Agarrar y manipular objetos con físicas realistas que tienen un comportamiento gravitacional (es decir, tienen peso, sus formas colisionan con los demás objetos, caen si los soltamos etc...).
- Accionar mediante botones un reproductor musical, de forma que podrán reproducir y parar la propia reproducción de una canción.
- Visualizar el funcionamiento y la explosión de los componentes de un conjunto industrial.
- Manipular los componentes del mismo y realizar diferentes operaciones que modificaran la posición y la escala de los mismos.
- Y por último, tendrá la posibilidad de observar en tiempo real, el cambio que sufrirá el entorno virtual mediante una serie de modificaciones que he realizado sobre la escena inicial.

Durante todo el proyecto he seguido una metodología de explicaciones paso a paso, en las que he ido haciendo las referencias necesarias entre los diferentes apartados de este mismo documento y a los apartados del documento de "*Anexos*". De esta forma, en caso de que alguna persona (alumno o profesor) quiera continuar con el desarrollo y la posible mejora de este software, pueda hacerlo de manera fácil, haciendo todo lo que ha estado en mi mano por explicarle de la mejor forma posible todo mi trabajo de investigación y la puesta en marcha en Unity.

Finalmente, he llegado a una serie de conlcusiones sobre la realización de este proyecto, las cuales podemos encontrar al final de este mismo documento.

Índice

Índice

0 Introducción

- 0.1 Título del proyecto
- 0.2 Objetivos del proyecto
- 0.3 Cronograma

0.1 Título del proyecto

"Creación de un entorno interactivo para la manipulación de modelos en 3D mediante realidad virtual y destinado para fines industriales"

0.2 Objetivos del proyecto

- Crear un entorno virtual e interactivo, en el cual, el usuario pueda estar inmerso en él, utilizando el equipo de realidad virtual "HTC Vive Pro".

- Que el usuario, dentro de éste entorno, sea capaz de:

• Visualizar el explosionado en tiempo real de un conjunto.
• Visualizar una animación que represente el funcionamiento del conjunto.
• Manipular, con sus propias manos y mediante el uso de unos mandos, todos los componentes del conjunto.
• Realizar las siguientes operaciones con los componentes manipulables: agrandar su tamaño, disminuirlo y finalmente, tener la posibilidad de ensamblarlo otra vez en su sitio.

- Introducir al departamento de Expresión Gráfica de la Escuela de Ingeniería y Arquitectura de Zaragoza en el mundo de la realidad virtual y su programación. Y de esta forma, apoyar a la docencia de la escuela y a mis profesores. A quienes, conocer el funcionamiento de este software, podría servirles como una herramienta muy útil a la hora de enseñar diferentes materias.

0.3 Cronograma

FASE 1: Realidad virtual, HTC Vive Pro y Unity.
Investigación sobre el campo de la realidad virtual, la programación en Unity y las posibilidades que nos ofrece el equipo de realidad virtual "HTC Vive Pro".

FASE 2: Proyecto de pruebas, investigación y aprendizaje.
Instalación y puesta en funcionamiento de HTC Vive Pro en Unity e investigación sobre la programación y scriptado necesario para poder realizar las operaciones requeridas en los objetivos del proyecto.

FASE 3: Creación de la escena virtual.
Ideación y ambientación del entorno virtual. Modelado de los componentes necesarios, exportación e importación en Unity y generación del entorno virtual.

FASE 4: Programación y funcionamiento del software
Activación de la sala de trabajo, programación de los scripts necesarios y explicación del funcionamiento del software.

FASE 5: Resultado y testeo final.

1 Realidad virtual, HTC Vive Pro y Unity

- 1.1 ¿Qué es la realidad virtual?
- 1.2 HTC Vive Pro
- 1.3 Unity

1.1 ¿Qué es la realidad virtual?

La tecnología de realidad virtual se basa en la generación de un entorno ficticio, al que, en la mayoría de los casos, se le suele dar una apariencia totalmente real. Este entorno permite al usuario trasladarse a cualquier lugar o situación posible, siempre que esta haya sido progamada anteriormente. A través de un dispositivo (gafas, cascos de realidad virtual...), podemos sumergirnos dentro de juegos donde encarnamos a un personaje determinado o viajamos sin necesidad de movernos físicamente de nuestro sitio.

Además de los dispositivos de inmersión (gafas, cascos...), existen otro tipo de dispositivos que complementan y mejoran la experiencia de la realidad virtual. Por ejemplo, los mandos, se adaptan a nuestras manos y generan unas nuevas de forma virtual, las cuales podemos visualizar en el entorno ficticio. Con estos controladores, tendremos la capacidad de manipular virtualmente diferentes elementos existentes en el entorno.

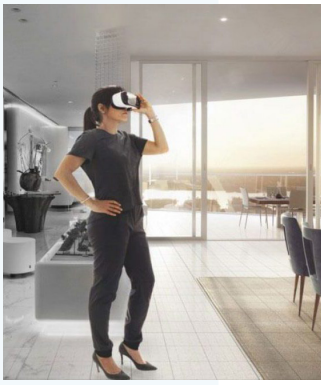
1.1.1 Realidad virtual vs realidad aumentada

La realidad virtual "VR" y la realidad aumentada "AR", forman parte de un mismo campo, la realidad extendida "XR". Ambas tecnologías se basan en crear elementos de forma virtual y nos generan la ilusión de que son reales, haciendonos creer que los tenemos justo enfrente de nosotros. Pero estas, lo hacen de forma diferente.

Realidad Virtual

Hablamos de realidad virtual cuando la totalidad del entorno (escenario, personajes, modelos 3D etc...) esta generado al 100% de manera virtual. Es decir, a través de un ordenador, se genera el entorno virtual y seremos nosotros los que estaremos inmersos dentro de él, dejándo atrás por completo el mundo real.

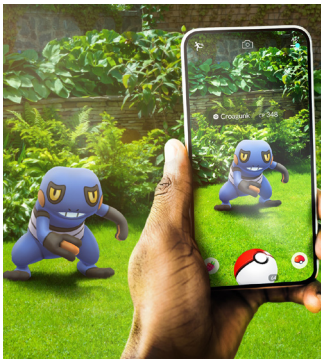
Ejemplos: juegos de consola, visitas virtuales de inmuebles etc...

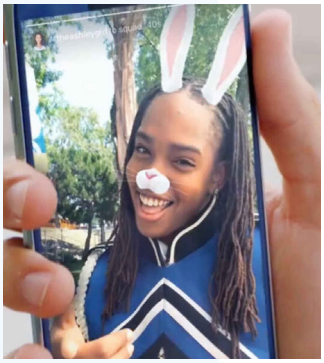


Realidad Aumentada

En el caso de la realidad aumentada, en vez de utilizarse un escenario virtual, se utiliza la escena real. Y sobre esta realidad, se añaden diferentes elementos y objetos virtuales que, de alguna forma, interactúan con el entorno real en el que están inmersos y con nosotros.

Ejemplos: filtros de Instagram, "Pokémon Go" etc...

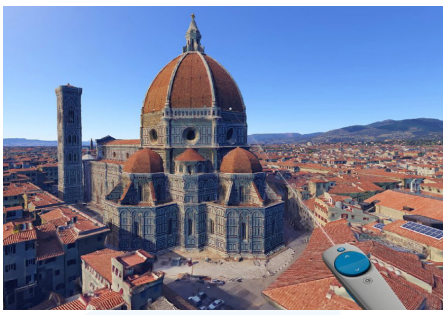




1.1.2 ¿Qué usos tiene la realidad virtual en el mundo real?

Hoy en día, esta tecnología se relaciona principalmente con los videojuegos. Pero esta es tan solo una de las diversas áreas donde podemos encontrar la VR. Otras áreas de aplicación de esta tecnología son:

Viajes

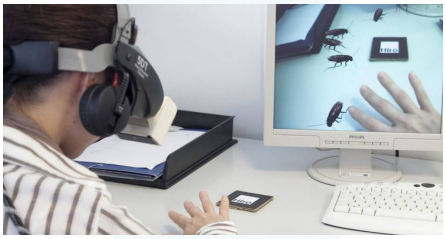


En el año 2001, la empresa Google, lanzó "Google Earth". Con ella, podíamos ver el mundo desde el ordenador, teniendo la posiblidad de viajar hasta cualquier punto del globo terráqueo. Pero en el año 2016, Google decidió mejorar su software y lanzó su nueva aplicación "Google Earth VR". La idea de este nuevo programa, es la misma que la del anterior. Pero esta vez, podremos viajar de forma virtual, dándonos la posibilidad de estar inmersos en el lugar que visitemos.

Redes sociales

Si hemos estado al corriente de los últimos avances en este campo, sabremos que la llegada de la realidad virtual a las redes sociales, ya es un hecho. Hace unos días, Mark Zuckerber (fundador de Facebook), ha anunciado de forma oficial el nacimiento del gran universo virtual "Metaverso". Un mundo nuevo en el que podremos realizar diferentes actividades utilizando la tecnología de VR.

Medicina



Existen tratmientos para tratar la depresión o la ansiedad en los que se utiliza este tipo de tecnología. Al paciente, se le expone de forma gradual a una situación que le genera pánico. Esto va haciendo que, poco a poco, vaya aumentando la confianza y la seguridad hacia esta fobia. Y todo ello, sin correr el riesgo de que pueda suceder algo inesperado.

Arquitectura

Jon Brouchoud, fundador y CEO de la empresa Arch Virtual, dice lo siguiente: "*Probablemente en los estudios habrá una sala dedicada a la realidad virtual, donde ponerse el casco y moverse por el proyecto de los edificios, y tal vez se podrán utilizar nuevas herramientas interactivas (como guantes para manipular elementos intangibles)*".

Defensa

La empresa coreana DoDAAM utiliza esta tecnología para el entrenamiento de los paracaidistas. El candidato se pertrecha y se coloca un visor de VR. Con este, obtiene una visión subjetiva y es capaz de dirigir sus movimientos en el espacio virtual sin poner en riesgo su vida.

Arte

Google ha lanzado una aplicación llamada "Tilt Brush" que es capaz de reproducir un pincel de forma virtual, con el que podemos: dibujar, pintar, exculpir... y realizar diferentes actividades dentro de un entorno interactivo en 3D.

Para mayor información sobre el uso de la realidad virtual consultar el apartado "*1.1 Implementación de realidad virtual*" del anexo "*Anexo I: Documentación e investigación*" del documento de "*Anexos*".

1.2 HTC Vive Pro

"HTC Vive" se trata de un equipo de realidad virtual que esta compuesto por: unas gafas de realidad virtual, unos auriculares incorporados y dos mandos (uno para cada mano). Aparte de esto, incorpora todo el hardware y software necesario para poder utilizarlo en un ordenador: dos sensores de movimiento, cableado necesario, progama a instalar en el pc etc...

Este equipo ha sido fabricado por las empreas HTC y Valve Corporation. La primera, se trata de una empresa tecnológica taiwanesa especializada en la fabricación de teléfonos inteligentes (smartphones). Para obtener más información acerca de esta empresa, consultar el subapartado "*1.2.1 HTC Corporation*" del anexo "*Anexo I: Documentación e investigación*" del documento de "*Anexos*".

Y la segunda, se trata de una empresa estadounidensa desarrolladora de videojuegos.

En el año 2015, HTC presentó al mundo su producto durante el "Mobile World Congress". Y un año más tarde, empezó a comercializarse.

Con el lanzamiento de este producto, HTC se introdujo en el campo de la realidad virtual. Además, HTC Vive ganó más de 22 premios al diseño y al desarrollo tecnológico. Por lo que, HTC no se quedó ahí. Con los años, la empresa ha ido sacando nuevos modelos de gafas virtuales al mercado, en donde nos presentan mejoras respecto a los modelos anteriores. Los modelos de gafas con los que cuenta la marca son: HTC Vive, HTC Vive Pro, HTC Vive Pro 2, HTC Vive Cosmos, HTC Vive Cosmos v2, HTC Vive Cosmos Elite y HTC Vive Focus.

Si nos dirigimos al subapartado "*1.2.2 Equipo de realidad virtual HTC Vive* " del anexo "*Anexo I: Documentación e investigación*" del documento de "*Anexos*", podremos ver algunas de las especificaciones de cada uno de los productos. No he descrito todas las características de cada uno de los modelos, ni tampoco he repetido las especificaciones que no han sido mejoradas en su siguiente modelo. Solamente me he centrado en las características que diferencian a los modelos tanto de sus antecesores y de sus predecesores como de otros modelos de la marca. Esto, me ha servido para comparar las "HTC Vive Pro" (equipo de realidad virtual que tengo a mi disposición para la realización del proyecto) con el resto.

Fijándonos en las especificaciones técnicas de cada producto, sí que podemos decir que existe un gran cambio en Vive Pro respecto a sus antecesoras las Vive. Pero en el caso de Vive Pro 2, no creo que haya un cambio tan grande como para plantearnos su compra. Solamente en la pantalla encontramos un cambio sustancial. Y aún así, también veo una limitación en este caso. El hecho de que Vive Pro 2 utilice una pantalla LCD en vez de una OLED, hace que esta sea mucho más rápida y que podamos ver todo con mayor calidad y mayor nivel de detalle. Pero esta pantalla, en algunos casos, podría no llegar a ofrecer las mismas tonalidades de negro intenso y la relación infinita de contraste que si que ofrece una pantalla OLED.

Comparándolas con las "Vive Cosmos" y sus predecesoras, podemos ver que la única mejora que ofrecen respecto a las nuestras, es la mejora de la interfaz de usuario. Pero en mi caso, no voy a utilizar este equipo para "disfrutar" de una experiencia virtual, sino todo lo contrario. Mi objetivo es hacer que otro usuario disfrute y pueda utilizar un software creado por mí. Por lo tanto, esto no me supone ningún beneficio.

Y por último, para el caso de las gafas "Focus", si que vemos una mejoría notable respecto a "Vive Pro". Su campo de visión aumentado, sus sensores mejorados y sobre todo, el espacio de juego tan grande que estas posibilitan, le hacen una competencia muy severa a las nuestras. Pero centrándonos en el desarrollo del proyecto, aunque si que notaríamos una pequeña diferencia, tampoco es tan grande. De hecho, es la primera vez que pruebo un equipo de realidad virtual y por supuesto, la primera vez que programo algo en un entorno virtual. Asi que, no creo que necesite unas características y especificaciones gigantes para desarollar mi software.

Por lo tanto y a modo de resumen, opino que cuento con un equipo de realidad virtual muy bueno y con mucho potencial. Por lo tanto, el hardware y el software incorporado en este producto, no supondrá ninguna limitación a la hora de llevar a cabo el proyecto.

1.3 Unity

Unity es un motor gráfico multiplataforma 2D y 3D, con el que podemos desarrollar un software virtual e interactivo. Los motores gráficos o de juegos, son aquellos softwares o programas sobre los que construimos un videojuego. Es decir, un motor gráfico para un videojuego es lo que un programa de edición para una película o un vídeo.

El desarrollador (en ese caso, yó), a través de una serie de herramientas que incorpora este motor, da forma al juego que quiere construir y va programando el resultado deseado, añadiendo todos los elementos necesarios (sonidos, modelos, transicciones etc...). Todo motor de juego debe ofrecer también un motor de renderizado para poder ver, en tiempo real, el resultado que vamos obteniendo durante el proceso de programación. Los lenguajes de programación más utilizados en Unity son C#, UnityScript y Boo.

Existen otros programas que sirven como motores gráficos para videojuegos. Pero Unity, puede diferenciarse del resto, por ser una aplicación más intuitiva y sencilla a la hora de utilizar. También destaca por la gran cantidad de herramientas que nos ofrece y por la posibilidad que ofrece de utilizarse y complementarse con otros programas de edición y de modelado virtual (como por ejemplo: ZBrush o Cinema 4D).

En el apartado "*1.3 Videojuegos desarrollados en Unity* " del anexo "*Anexo I: Documentación e investigación*" del documento de "*Anexos*", podemos encontrar información relacionada con ejemplos de juegos en Unity.

Junto con mis conocimientos básicos sobre el funcionamiento del programa y después de una investigación sobre las posibilidades del mismo, he decidio desarrollar los siguientes subapartados, ya que, para la realiación de este proyecto, he considerado de alta importancia el conocimiento de los mismos.

1.3.1 Canalización de renderizado de alta definición

Para poder desarrollar un buen videojuego, sea del tipo que sea, necesitamos obtener la mayor calidad gráfica posible. Y en el caso de diseñar un software destinado a realidad virtual, esta necesidad se amplifica. Ya que no vamos a ver el resultado a través de una simple pantalla. Sino que la forma en la que veremos este entorno, será desde un casco de realidad virtual. Por lo tanto, necesitaremos una calidad de renderizado excelente.

Unity ha llevado su canal de renderizado de alta definición (HDRP) a la realidad virtual (desde la versión 2019.3.6f1 y posterior). La realidad virtual en HDRP esta diseñada para que:

- Todas las funciones de HDRP sean compatibles con la realidad virtual.
- HDRP sea totalmente compatible con el nuevo plugin "Unity XR Framework".
- Se realice un renderizado diferido y hacia adelante.
- Dé la posibilidad al usuario de utilizar cualquier tipo de luz, sombras, calcomanías y volumétricas.
- Se puedan generar los siguientes efectos de espacio en pantalla: oclusión ambiental (AO), reflexión del espacio de la pantalla (SSR), dispersión del subsuelo (SSS), distorsión y refracción.

Para obtener más información sobre el canal de renderización de alta definición utilizado por Unity consultar el apartado "*1.4 Formas de renderizado del canal HDRP de Unity*" del anexo "*Anexo I: Documentación e investigación*" del documento de "*Anexos*".

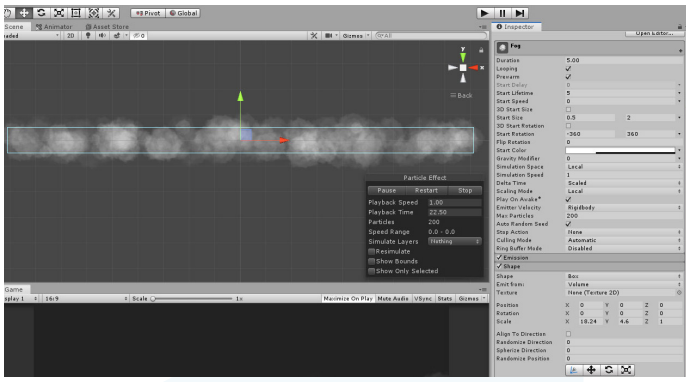
1.3.2 Sistema de partículas de Unity "Particle System"

El sistema de partículas simula y renderiza muchas imágenes pequeñas o mallas, llamadas partículas, para producir un efecto visual. Cada partícula de un sistema representa un elemento gráfico individual en el efecto, pero a su vez, el sistema simula cada partícula de forma colectiva para crear la impresión del efecto. Es decir, trabaja tanto de forma individual como de forma colectiva para así, simular de forma más precisa este efecto de movimiento de partículas.

Los sistemas de partículas son muy útiles cuando deseamos crear objetos dinámicos como el fuego, el humo o los líquidos (ya que es muy dificil representar este tipo de objetos mediante una malla (3D) o un

sprite (2D), los cuales son mejores para representar objetos de tipo sólido, como por ejemplo un coche).

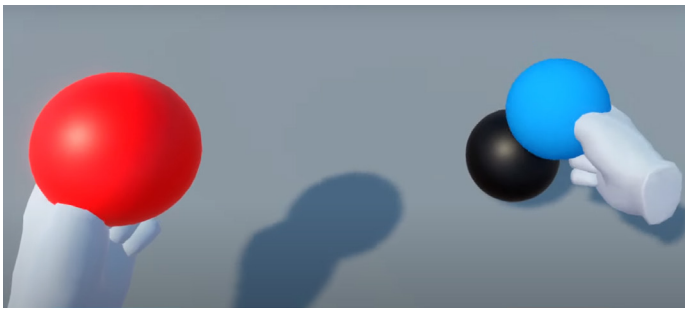
Por lo tanto, el uso de la herramienta del sistema de partículas que incorpora este motor, será una gran opción para representar objetos dinámicos que podríamos encontrarnos a la hora de representar escenas realistas para nuestro entorno de realidad virtual (por ejemplo: lluvía, agua fluyendo, nubes etc...).

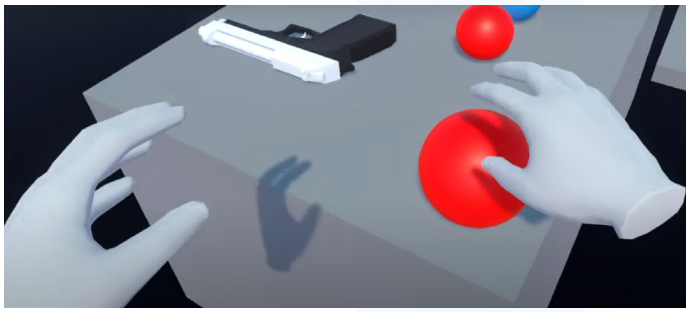


1.3.3 XR Interaction Toolkit

Se trata de un paquete instalable en nuestro proyecto, que nos proporcionará una serie de herramientas de interacción XR *[1]* y que nos permitirá agregar la interacción interactiva en nuestro juego o software AR *[2]* o VR *[3]*. Este paquete esta disponible desde la versión de Unity 2019.3 a través del package manager (administrador de paquetes importables al proyecto). Unity en su página web, nos dice que se trata de una tecnología nueva y en continuo desarrollo y que debemos tomar estas funcionalidades de forma experimental. Además, la propia página web pide a los usuarios programadores que escriban sus comentarios, experiencias y sus opiniones de mejora. De esta forma, el actual equipo de XR de Unity irá mejorando e implementando nuevos cambios en el software para conseguir poco a poco una tecnología cada vez más satisfactoria y compatible con todo tipo de ideas que tengan los usuarios.

Estas funciones nos permitirán introducirnos en nuestro videojuego y programar en él diferentes interacciones con otros modelos existentes en el mismo (desplazarnos por nuestro entorno virtual, seleccionar y agrrar objetos etc...). También proporciona una herramienta de teletransporte dentro de nuestra escena, con la que tendremos la posibilidad de cambiar nuestra posición con un simpe click en el mando.





2 Proyecto de pruebas, investigación y aprendizaje

- 2.1 Zona de trabajo
- 2.2 Primeros pasos en Unity
- 2.3 Pruebas realizadas

2.1 Zona de trabajo

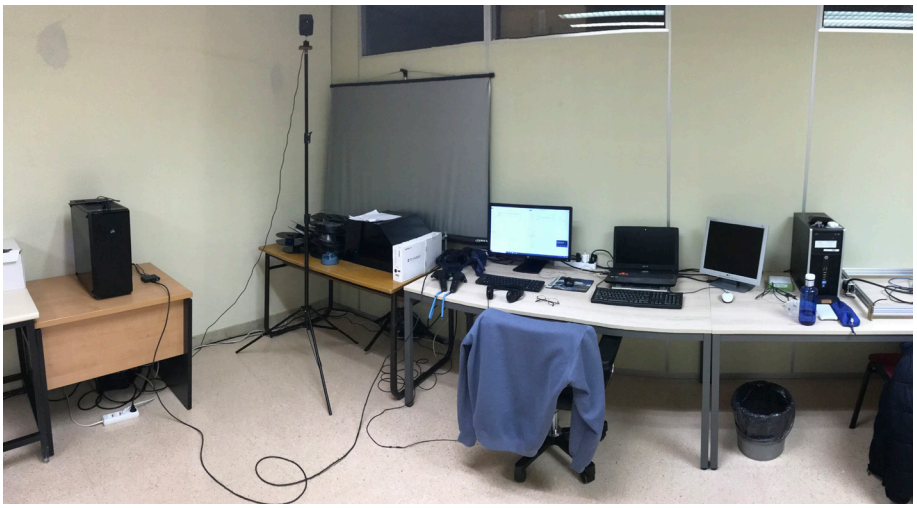
2.1.1 Montaje y preparación de mi zona de trabajo

Para poder generar el software que posibilite la interacción virtual en 3D con un ordenador, tanto Unity como el software necesario para su creación, se precisa de un equipo que cumpla con los requisitos para poder soportar la carga gráfica y operacional requerida. Y al tratarse de tecnología de realidad virtual, la potencia que necesitamos será mucho mayor.

Además, el equipo de realidad virtual no solamente consta de unas gafas de realidad virtual. Para que HTC Vive Pro funcione, necesitamos conectar dos sensores distanciados entre sí y colocados a una altura específica (por lo que necesitaremos dos trípodes de sujección). Y aparte, el equipo cuenta con dos mandos inalámbricos, con los que trataremos de realizar las operaciones de manipulación citadas en el subapartado "*0.2 Objetivos del proyecto*" de este mismo documento.

Por último, este equipo de realidad virtual pertenece al departamento de Expresión Gráfica de la Escuela de Ingeniería y Arquitectura de Zaragoza. Se trata de un equipo costoso y por norma establecida, este tipo de material no puede salir de la facultad.

Por todas estas razones, antes de comenzar el proyecto, los profesores acordaron que trabajaría en el Laboratorio 3D (situado en el mismo departamento). Allí, aparte de dejarme todos los dispositivos necesarios para utilizar el equipo de realidad virtual, también me han dejado un ordenador muy potente para poder trabajar con esta tecnología. Con el equipo de realidad virtual, los trípodes, el ordenador y las pantallas que tengo a mi disposición, y junto con mi ordenador (como herramienta secundaria) y cableado aportado por mi parte, he podido montarme una zona de trabajo muy funcional.

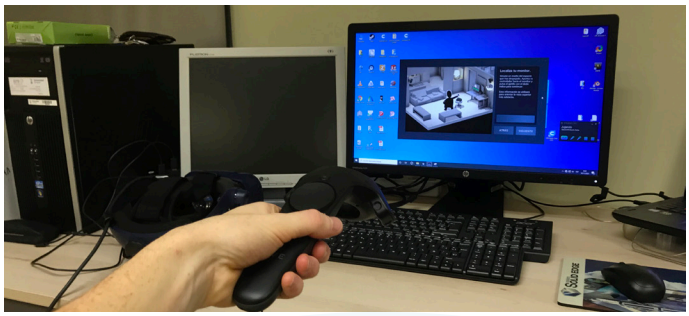


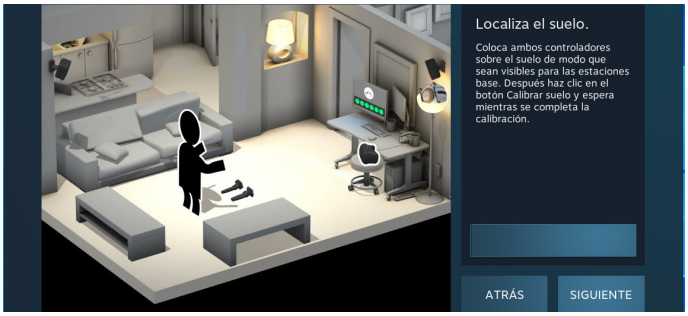
2.1.2 Preparación de HTC Vive Pro

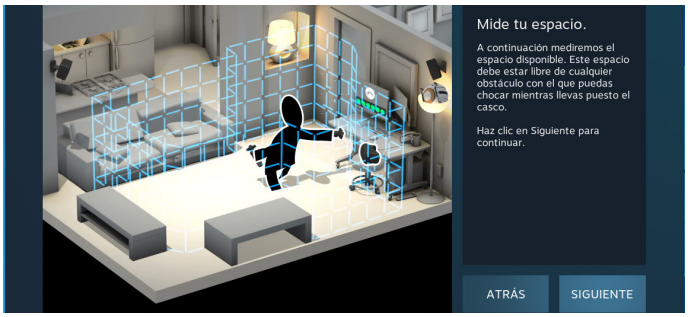
Para poder utilizar HTC Vive Pro en el ordenador, debemos seguir el manual "VIVE Pro HMD Setup Guide". Este, viene incorporado con el producto y se instala en el ordenador. Además, podemos encontrarlo de forma totalmente gratuita en la página web de HTC Vive.

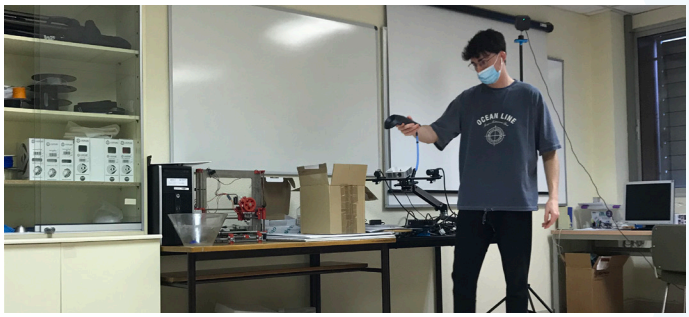
En esta guía, se detallan todos los pasos a seguir para conectar todos los dispositivos que incorpora el equipo HTC Vive Pro de forma correcta.

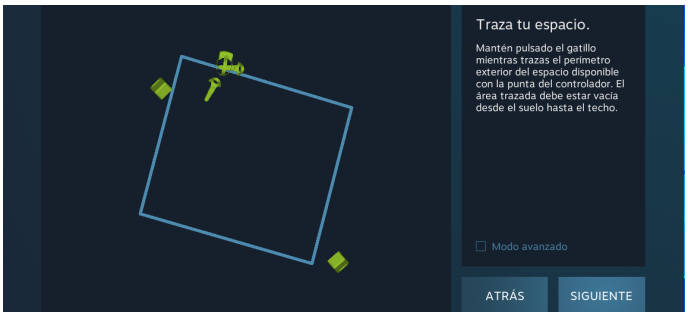
Una vez tenemos todo el hardware conectado y encendido, habrá que esperar a que el ordenador reconozca que los dispositivos han sido conectados de forma correcta. Y por último, debemos realizar la siguiente configuración:

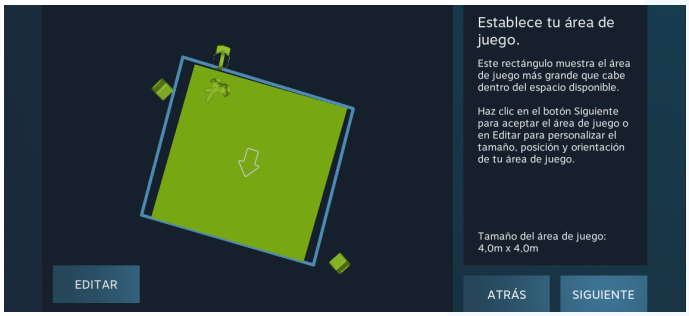












Delimitar de forma correcta y precisa el espacio libre es una tarea fundamental. Cuando tengamos el casco puesto, esta zona se delimitará con una malla roja virtual en 3D, y de esta forma, sabremos en todo momento donde están los limites de nuestra sala real. Al trabajar en este aula, la realización de esta configuración es muy importante, ya que hay máquinas y material bastante caro cerca de mí.

Por seguridad, todos los días al acabar de trabajar en el proyecto, retiro y desconecto los dispositivos del equipo de realidad virtual (sensores, casco y mandos). Y como acabo de comentar, el trazado del área libre debe ser lo más preciso posible. Por lo tanto, todos los días realizo este procedimiento al comenzar la mañana. De esta forma, reduzco la posibilidad de que algo este mal configurado o que el área delimitada en un principio, no coincida con la nueva colocación de los sensores (al tratarse de un trípode móvil).

2.2 Primeros pasos en Unity

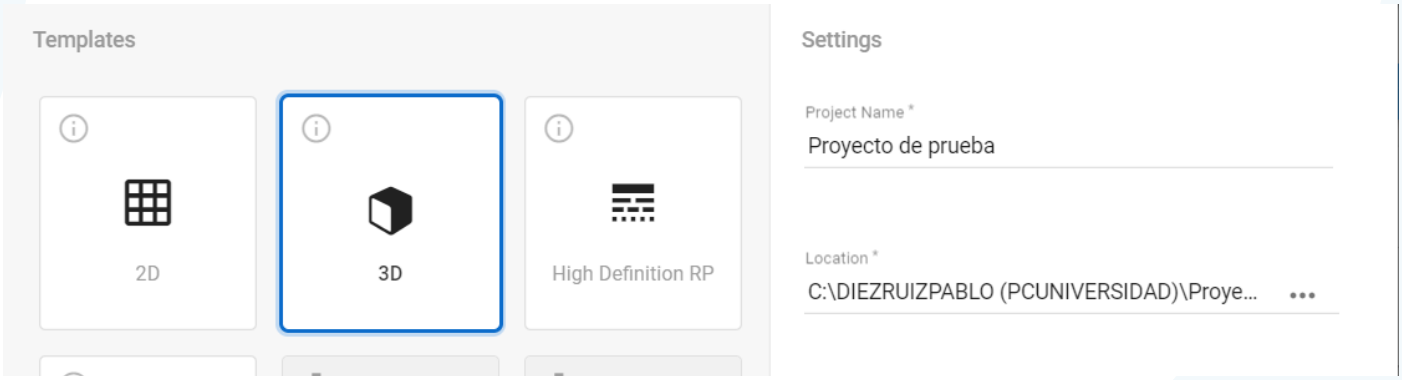
2.2.1 Creación del proyecto

En primer lugar, debemos instalar Unity Hub en nuestro equipo. Esta es la aplicación de "home" que ofrece Unity, desde la que podremos controlar todos los proyectos que estemos realizando desde nuestra cuenta personal. Después de descargar este programa, descargaremos la versión que queramos de Unity. Para realizar este proyecto, he descargado e instalado la versión "Unity 2020.3.19f1", a la que le he añadido los siguientes módulos:

- Microsoft Visual Studio Community 2019: Para poder programar en lenguaje C#.
- Android Build Support: Para una posible futura puesta en funcionamiento en un dispositivo android.
- iOS Build Support: Para una posible futura puesta en funcionamiento en un dispositivo i-phone.
- Universal Windows Platform Build Support: Para la puesta en funcionamiento desde un ordenador.
- WebGL Build Support: Para la puesta en funcionamiento a través de un enlace web.

Una vez instalado el programa y todos los módulos, he creado una carpeta personal dentro del ordenador del laboratorio. Esta, la he creado en el disco C (en el mismo en el que está instalado el sistema operativo, para que todo el software que utilice funcione de forma más rápida) y en ella, he almacenado todos los archivos relacionados con mi proyecto (archivos de Unity, Inventor, SketchUp, Adobe, Office etc...)

Por último, he creado el proyecto "Proyecto de pruebas". Por el poco conocimiento que tengo acerca de la programación en C# y el nulo conocimiento que tenía sobre como podría conectar HTC Vive Pro con el motor de juegos, he decidido realizar proyectos de pruebas en Unity. Y de esta forma, poder probar con antelación diferentes formas de introducir el equipo en el software y familiarizarme un poco con todo antes de lanzarme a crear el proyecto final.



2.2.2 Conexión HTC Vive Pro - Unity

Este motor de juegos está preparado para la integración de los dispositivos que incorpora HTC Vive Pro. Este equipo de realidad virtual llegó algo más tarde que la tecnología de Oculus Quest (otro dispositivo de realidad virtual) a Unity. Pero a partir de la versión "Unity 2018.3", podemos configurar nuestro equipo HTC Vive Pro en Unity.

Una vez instalada una versión igual o superior a la citada anteriormente, debemos seguir los siguientes pasos:

- Nuestro dispositivo HTC Vive Pro debe tener todo el software actualizado.
- Instalar el software Steam y Steam VR en nuestro ordenador.
- Añadir los siguientes paquetes:
    - XR Interaction Toolkit.
    - XR Plug-in Management.
    - Steam VR Plug-in.

Y por último, Unity recomienda al usuario que, antes de adentrarse en el mundo de la programación en realidad virtual, tenga unos conocimientos básicos acerca de su funcionamiento y sobre programación.

En mi caso, aunque tenga un conocimiento bastante escaso sobre programación, creo que con los conocimientos básicos con los que parto sobre el programa y la investigación que realice acerca de C#, podré sacar el proyecto adelante. Por lo que doy por cumplida esta última recomendación que Unity exige al usuario.

En el apartado "*1.5 Instalación del software y paquetes necesarios* " del anexo "*Anexo I: Documentación e investigación*" del documento de "*Anexos*", se encuentra de forma detallada y explicada el procedimiento seguido para realizar los pasos descritos anteriormente.

2.3 Pruebas realizadas

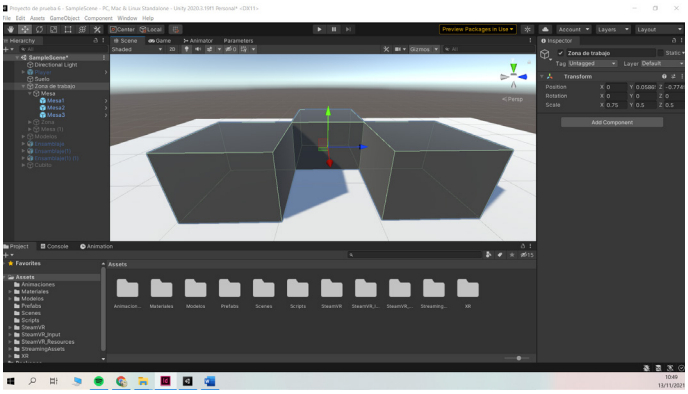
2.3.1 Montaje de la escena

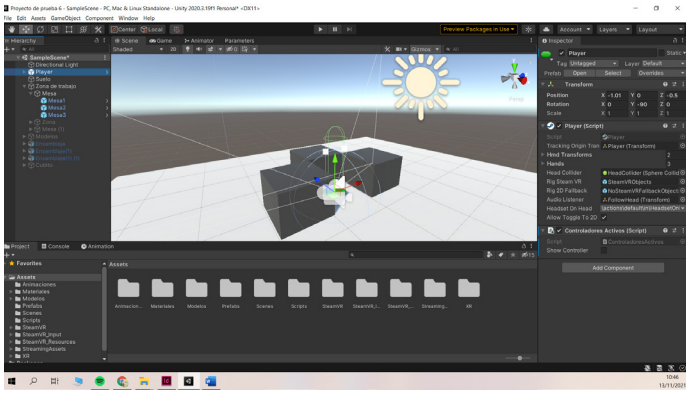
Primero, he creado una zona de trabajo virtual dentro del software, desde la que trabajaré virtualmente con los modelos 3D. Para generarla, he creado un plano y tres cubos. El plano actuará de suelo y los cubos actuarán como mesa de trabajo. Después, he añadido el componente "Player", el cual he importado desde la carpeta de Assets del proyecto. Este componente es parte del paquete de "Steam VR Plug-in" e incorpora todos los componentes necesarios para simular nuestra inmersión en el mundo virtual (cámara de seguimiento, manos de manipulación, colliders *[1]* establecidos etc...).

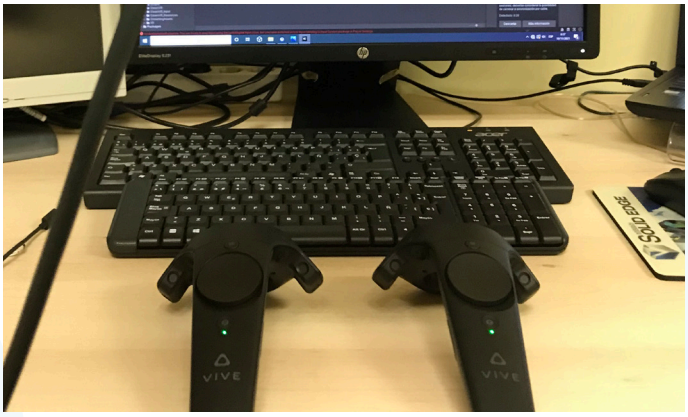
A continuación, he colocado el componente "Player" entre los tres cubos (en el medio del "escritorio virtual"). Y por último, he eliminado de la jerarquía de componentes la "Main Camera". Esta es la cámara principal que viene por defecto cuando iniciamos una escena desde cero (al igual que la "Directional Light"). Como el componente de "Player" ya tiene una cámara definida (que es la que está conectada a nuestro visor de realidad virtual), debemos eliminar la otra para que esta sea la cámara que se ejecute a la hora de inicializar el software.

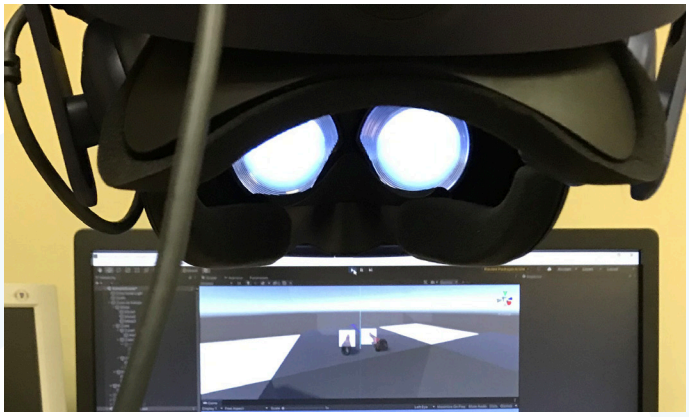
He decidido hacer esto antes de empezar, ya que a la hora de programar todo lo que viene a continuación, debo de hacer continuamente la siguiente secuencia: inicializar el software, probar, parar el software, corregir. En caso de tener que estar levantándome todo el rato y yéndome al medio de la habitación para realizar las pruebas, perdería mucho tiempo y no sería un trabajo muy eficiente.

[1] Un collider es la definición de la forma de un objeto 2D o 3D para los propósitos relacionados con colisiones físicas con otros objetos 2D o 3D. Es decir, es el área o el volumen definido por una forma determinada, que en caso de chocar con otro área o volumen con caracaterísticas similares (otro collider), colisionará con el simulando un choque. Toda la información necesaria acerca de los colisionadores en Unity se encuentra en el siguiente enlace: "https://docs.unity3d.com/es/2018.4/Manual/CollidersOverview.html".







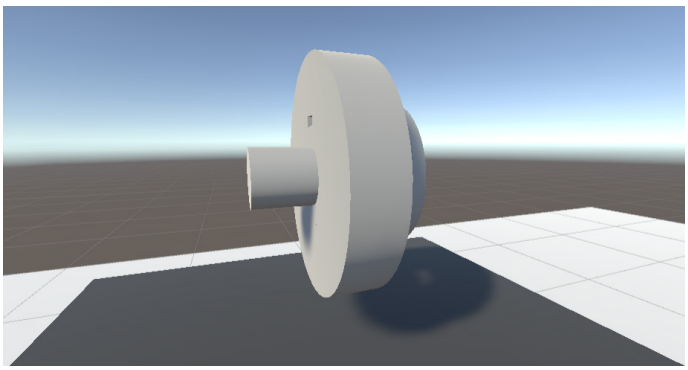
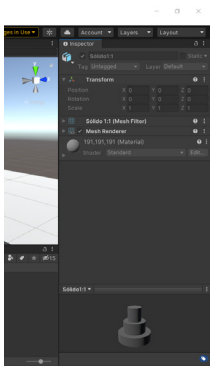
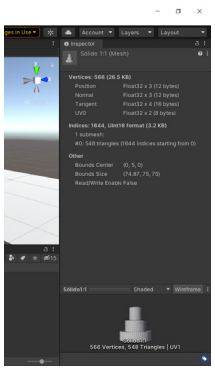
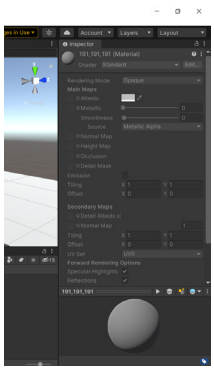


2.3.2 Modelo de pruebas

Para comenzar con las pruebas de manipulación, he partido de un modelo simple que tenía guardado en mi ordenador personal de cuando realicé algunas pruebas en mi impresora 3D. Se trata de un archivo .obj (archivo válido para la importación en Unity). Este modelo consta de tres componentes sencillos. Uno con la forma de un eje, otro con la forma de una corona (sin dentado) y otro con la forma de una chaveta. Al importar el archivo en extensión .obj (donde el modelo está ensamblado), Unity nos muestra siete archivos: dos para cada modelo y el material (que es el mismo en los tres componentes).

Los archivos de cada modelo son: un archivo que genera el modelo 3D del componente y otro archivo que genera el "Mesh" utilizado por el archivo anterior en su componente "Mesh Renderer" *[2]*. Y por último, el archivo de material aplicado a cada uno de los componentes (aplicado a los archivos que generan el modelo 3D), no podemos editarlo de forma directa. En caso de querer editar el material, debemos clickar en Ctrl+D, duplicamos el material, editamos el nuevo y lo aplicamos de nuevo a cada uno de los componentes.

[2] La traducción de "Mesh" es "Malla". Por lo tanto, el "Mesh Renderer", traducido al español como "Renderizador de Malla", se encarga de tomar la geometría guardada en el archivo "Mesh Filter" y lo representa en la posición definida por el GameObject. Por ejemplo, en el caso anterior con el plano del suelo, en la posición (0,0,0). Toda la información necesaria acerca de los colisionadores en Unity se encuentra en el siguiente enlace: "https://docs.unity3d.com/Manual/class-MeshRenderer.html".

2.3.3 Pruebas I: Manipulación

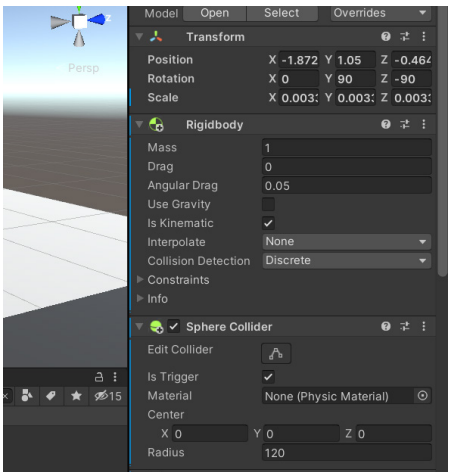
El objetivo de este primer subapartado de pruebas es conseguir poder manipular los componentes que conforman el ensamblaje. Además, estos deben comportarse de forma flotante (sin gravedad) y el movimiento debe ser fijo (es decir, que cuando transformemos su posición y orientación, estos se queden fijos en esa posición).

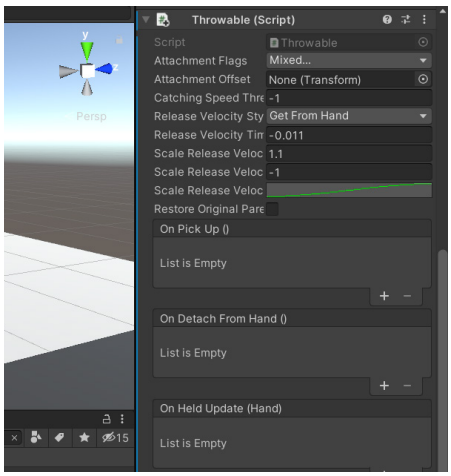
Manipulación del conjunto

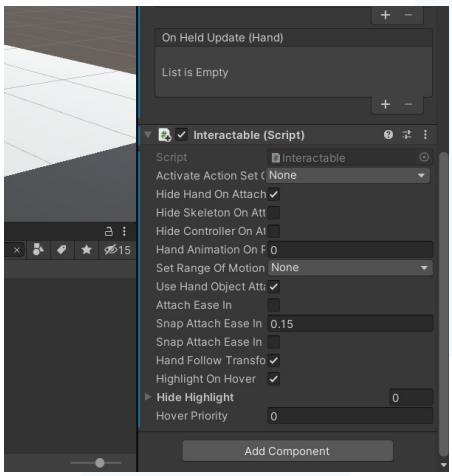
Para poder manipular (seleccionar y agarrar) el conjunto, debemos asignarle a su GameObject los siguientes componentes:

- Rigidibody: Le permitirá al GameObject actuar bajo el control de la física (peso, gravedad, velocidad etc...).
- Sphere Collider: Es un tipo de Collider de forma esférica, que permitrá al GameObject detectar el contacto físico con el collider ya programado de nuestras manos virtuales.
- Scripts (Throwable e Interactable): Los encontraremos dentro del paquete de Steam VR Plug-in.

Finalmente he ido ajustando los diferentes parámetros de cada uno de los componentes:







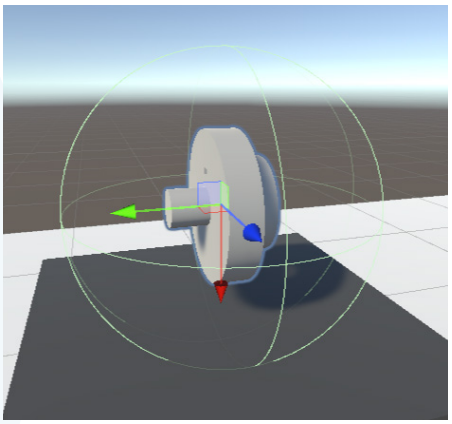
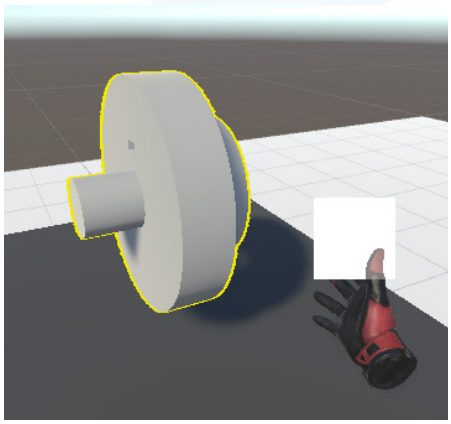
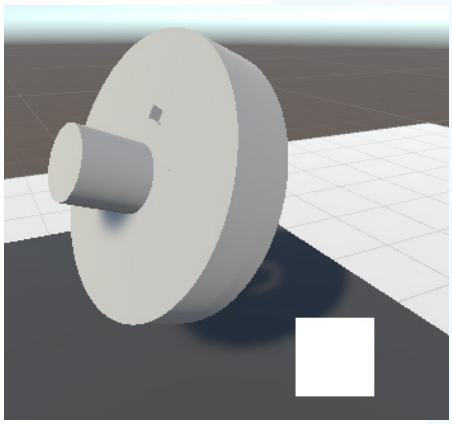
La función "Is Kinematic" dentro del componente "Rigidibody", hace posible que el GameObject "flote" y además mantenga la orientación una vez hemos modificado su posición. También hay que deseleccionar la opción "Use Gravity", ya que el GameObject adquiriría un comportamiento físico real y caería hacia abajo hasta chocar con el siguiente GameObject (más exactamente con el collider asociado a ese objeto).

El collider que le he añadido, lo he hecho mucho más grande que el tamaño del conjunto. Además, al uti-

lizar una collider en forma de esfera, tampoco me he ajustado a la propia forma del componente. Esto lo he hecho, ya que he ido viendo que era más cómodo agarrar y manipular el conjunto desde la distancia, sin interferir directamente sobre su forma. Así, mientras estamos manipulando el conjunto, podemos ir viéndolo en su totalidad.

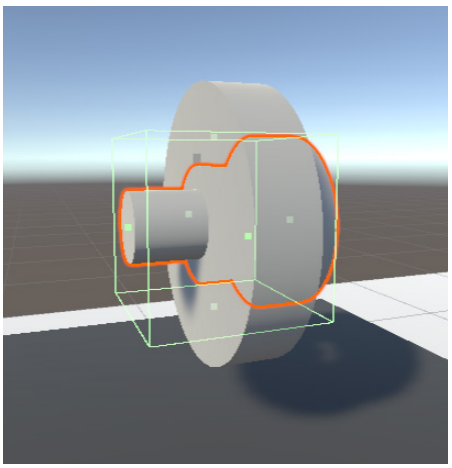
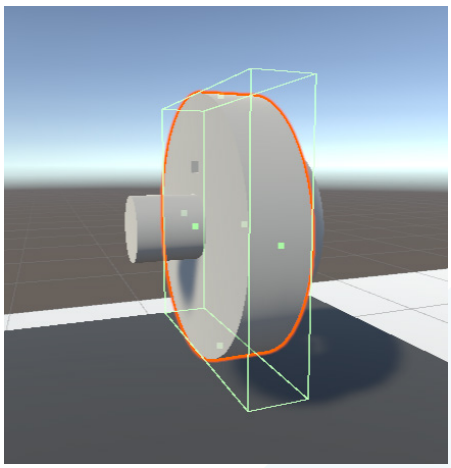
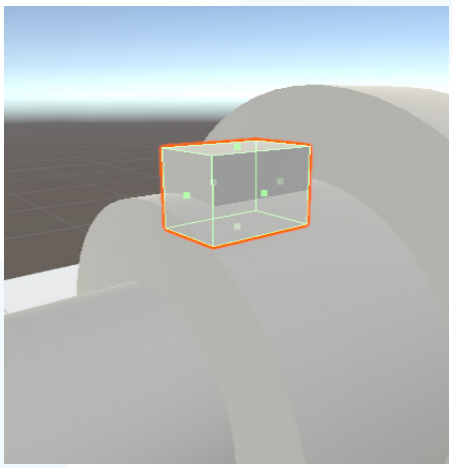
Y por último, la función "Is Trigger" dentro del componente "Sphere Collider", debe estar activada. Encontraremos esta función en cualquier tipo de collider programable de Unity. Es una función que va directamente asociada con la programación de los scripts asociados a los objetos. Cuando un GameObject choca con otro GameObject, Unity llamará a la funcion "OnTrigger". Esto es algo que vamos a ver en fases más avanzadas del proyecto, ya que gracias a esta función, podremos hacer que pase algo cuando un GameObject colisiona contra otro (por ejemplo, cuando mi mano virtual toque un botón, que se encienda un aparato de música). Pero como acabo de decir, veremos esto en profundidad más adelante.

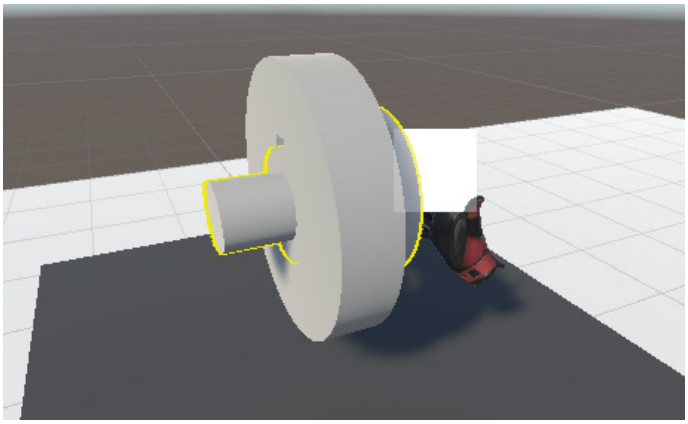
Lo que realmente nos interesa ahora de que esta función este activada dentro de nuestro collider es que, cuando nuestra mano (su collider adaptado a su forma) choca contra el conjunto (su sphere collider), Unity va a detectar que la colisión se ha producido, pero no va a generar un choque físico entre los dos componentes. Por lo que podremos atravesar el conjunto con nuestra mano sin detectar una colisión física.

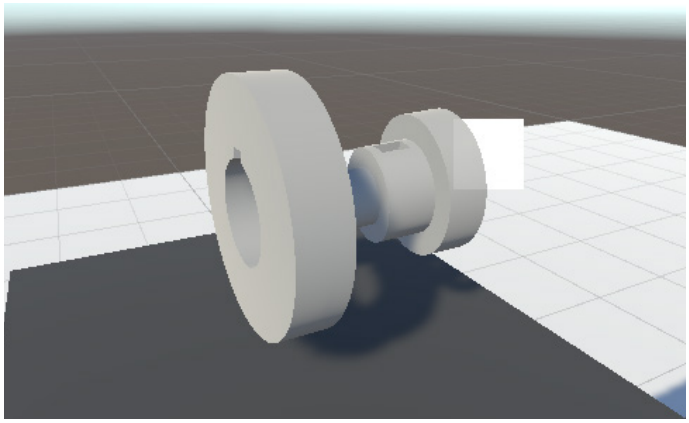




Manipulación individual

Como es de esperar, para poder manipular cada uno de los componentes que conforman el ensamblaje, debemos realizar el mismo proceso, de forma individual, para cada uno de los GameObjects asociados a estos. En este caso, el tipo de collider que he elegido es "Box Collider". De esta forma, he querido ajustar más el collider a la forma real del componente y así, no generar interferencias a la hora de escoger uno u otro. Es decir, que haya zonas en las que el collider de nuestras manos solamente detecte la colisión con uno de los otros colliders (para poder escoger entre un componente u otro).

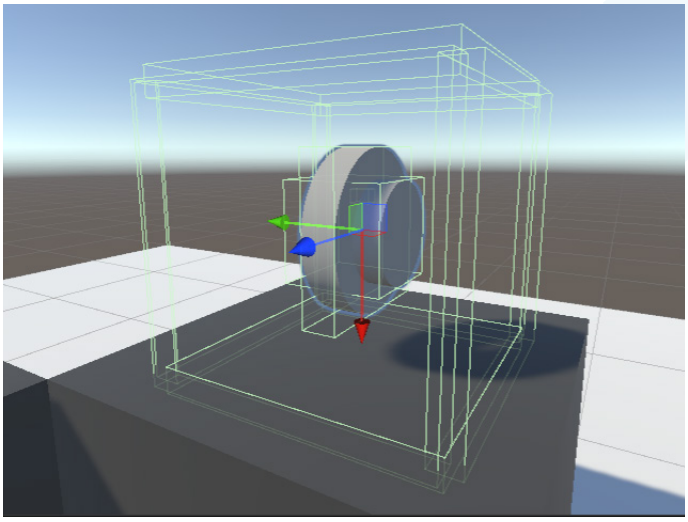




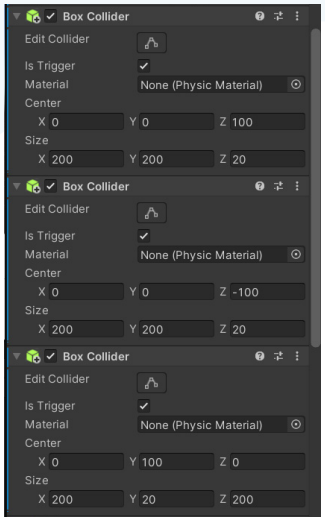


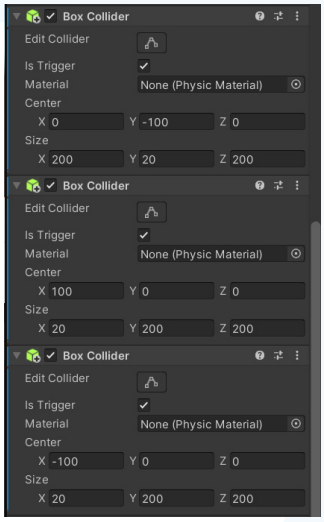


Pero si ya habíamos definido anteriormente para el conjunto un collider de tipo esférico... ¿éste no interferirá con los nuevos colliders definidos?

La respuesta es sí. Y es por eso, que se me ocurrió la siguiente solución:







En este caso, he definido seis colliders de tipo "Box Collider" con la intención de generar una forma hueca. Gracias a esta forma hueca, los colliders asociados a los componentes no interfieren con el collider asociado al conjunto.

En ambos casos (collider grande esférico o el collider compuesto) funciona correctamente. Es decir, podemos manipular el ensamblaje de forma conjunta y podemos seleccionar uno de ellos, agarrarlo y manipularlo. Pero si que es verdad que en el segundo caso, el funcionamiento es mejor. Ya que, en el primer caso, si que podemos llegar a notar algun tipo de interferencia a la hora de seleccionar un componente (algunas veces cambia con un leve parpadeo la selección del componente individual a la del conjunto entero). Esto, en el segundo caso, no sucede nunca. Por lo tanto, aunque las dos formas funcionen, el segundo método, al no generar ninguna interferencia de tipo collider componente-conjunto, es mejor. Pero, aunque sea algo mejor, terminé la realización de las pruebas utilizando el primer caso.

El hecho de que tengamos un collider hueco, lleva consigo la necesidad de generar varios colliders. Y la existencia de varios colliders, hace que Unity detecte las colisiones de forma individual y no en conjunto. Unity no sabe que ese conjunto de colliders deben actuar como uno mismo. Por lo tanto, cada vez que uno de los colliders entre en contacto o deje de estarlo con otro, como ya hemos dicho antes, el motor lo va a detectar y entonces, llamará a la función "OnTrigger". Y al realizar las pruebas para poder agrandar o disminuir el tamaño de los componentes/conjunto, me di cuenta de que es exactamente esto lo que generaba

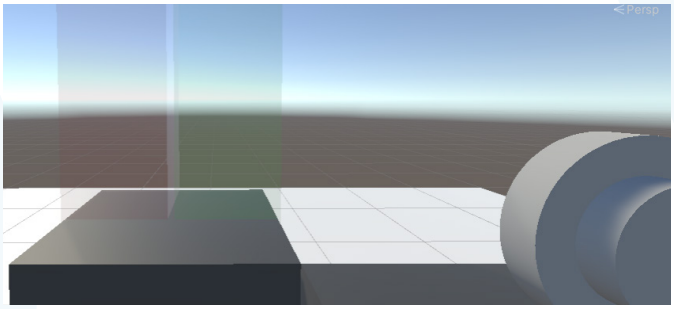
un problema e impedía seguir creciendo de tamaño al conjunto.

Explicaré todo a continuación en la siguiente sección *"2.3.4 Pruebas II: Modificación de tamaño"*.

2.3.4 Pruebas II: Modificación de tamaño

El objetivo de este segundo apartado de pruebas es conseguir que, realizando una acción determinada, el usuario sea capaz de modificar el tamaño de un componente/conjunto.

La principal función que ofrecerá el software final es la posibilidad de manipular en todo momento los componentes y modelos 3D que introduzcamos en él. Por lo tanto, la forma en la que nosotros elegimos uno de los componentes es seleccionándolo y agarrándolo. Por eso, la forma más rápida e intuitiva a la hora de modificar el tamaño del mismo, también debería seguir ese camino. Es decir, seleccionar el componente que queramos modificar sus dimensiones (por ejemplo, un tornilllo, para poder verlo de forma más detallada al tener un tamaño tan pequeño), agarrarlo y finalmente agrandar su tamaño.



Lo que he hecho finalmente es hacer dos áreas diferenciadas por colores. Se tratan de cubos 3D que han sufrido una modificación en su tamaño. Para crear el efecto transparente que tienen, he generado un material nuevo para cada uno de ellos, en los que aparte de cambiar su color "Albedo" dentro de los parámetros del material *[3]*, he modificado también su modo de renderizado a "Transparent".

Además, he creado un collider de tipo "Box Collider" para cada una de las áreas, haciéndolo coincidir exactamente con la forma del área. Además, la casilla de "Is Trigger" debe estar activada. Pero en este caso, la razón por la que activaremos esta casilla, no será solamente para que al introducir un componente dentro del área éste no choque físicamente con la misma y se le impida su paso (como hicimos anteriormente en la sección de *"Pruebas I: Manipulación"*).

Para que Unity sepa que tiene que hacer algo (disminuir o aumentar el tamaño del componente) cuando el collider de uno de los componentes entra en contacto con el collider de una de las áreas, utilizaré la función "OnTriggerEnter". Para que esta función se ejecute, deben de cumplirse dos condiciones:

- Uno de los dos GameObjects que van a colisionar debe tener su componente "Is Trigger" habilitado. Pero tan solo uno de ellos debe tenerlo activado, ya que en caso de que los dos GameObjects tengan "Is Trigger" habilitado, la colisión no ocurrirá. El objeto en el que debe estar activado, coincidirá con el objeto al que añadamos el script (en cuya definición se encuentra la función "OnTriggerEnter").
- Ambos GameObjects deben contenter un componente de "Rigidibody".

Para poder hacer esto posible, he creado tres scripts: "ModificarTamaño.cs", "AumentaTamano.cs" y "DisminuyeTamano.cs". El primero, lo he asociado a los GameObjects de cada uno de los componentes/conjunto.

[3] Los parámetros de un material son una lista de diferentes parámetros que podemos modificar a nuestro gusto y definen por completo el renderizado final de cualquier material. El parámetro "Albedo" controla el color base de la superficie. Toda la información necesaria sobre los parámetros de un material en Unity se encuentra en el siguiente enlace: "https://docs.unity3d.com/es/2019.4/Manual/StandardShaderMaterialParameters.html".

Mientras que, el segundo y el tercero, los he asociado a los GameObjects de las áreas.

Para el script "ModificarTamano.cs" he creado las variables: aumentar, disminuir (tipo booleano), x, y, z (tipo float) y he inicializado el script con aumentar y disminuir en "false" dentro de "void Start()". Para hacer que el tamaño del componente aumente o disminuya, debemos irnos a la función "void Update()". Como ya sabemos, nuestro ordenador ejecutará esta función constantemente y Unity realizará la función que nosotros hayamos programado en esta función cada vez que el ordenador sea capaz de ejecutarla.

Cada ordenador tiene un procesador específico, una tarjeta gráfica diferente, una RAM determinada instalada etc... Es por eso que, cada ordenador, tendrá una potencia y una rapidez diferente de ejecución. Por ejemplo, para medir la cantidad de instrucciones que un procesador es capaz de ejecutar en un único ciclo de reloj, se utiliza el valor IPC, con el cual se puede determinar su rendimiento. Pero... ¿Y qué pasa con todo esto? ¿De qué forma afecta a nuestro software?
Como hemos dicho anteriormente, Unity ejecutará la función "void Update()" tan rápido y tantas veces como el ordenador le permita hacerlo. Por lo que, un ordenador más potente, ejecutará esta función un mayor número de veces que en el caso de un ordenador menos potente. Esto genera un problema. Ya que si cada vez que la función se ejecuta, el componente crece una cantidad x en su tamaño, la velocidad de crecimiento dependerá del ordenador en el que estemos ejecutando este software. Es por eso, que aparece la función Time.deltaTime.

Ésta, se trata de una función ya programada por Unity, la cual se encarga de arreglar este desajuste provocado por la gran variedad de equipos existentes. Lo que hace es regular el intervalo en segundos desde el último fotograma hasta el actual. Es decir, esta función tiene establecido una cantidad de tiempo x fijada entre "veces de ejecución". Por lo tanto, si utilizamos esta función a la hora de modificar el tamaño de un GameObject en un determinado intervalo de tiempo, dará igual el equipo que estemos trabajando. La velocidad a la que esto va a suceder, es la misma.

Por último, he creado tres funciones: AumentarTamano(), DisminuirTamano() y StopTamano(). Estas serán las encargadas de cambiar el valor de los booleanos aumentar y disminuir. Por ejemplo, cuando se ejecute la función AumentarTamano(), el valor de aumentar pasará a ser "true". Al variar su estado, "if (aumentar == true)" dentro de void Update(), va a cumplirse. Por lo tanto, se ejecutará la función programada de manera permanente hasta que el valor de aumentar pase a ser "false". En este caso "if(aumentar == true) dejará de cumplirse y por lo tanto, se empezará a ejecutar la función "else". Esta función cogera el valor actual de los componentes (x, y, z) de su escala y le dará ese valor a la escala actual del GameObject. De esta forma, el objeto mantendrá la escala determinada que el usuario haya elegido.

Pero para que esto suceda, como ya hemos dicho, el valor de aumentar debe convertirse en "false". Y la única forma de que esto suceda es que una de las funciones DisminuirTamano() o StopTamano() se activen. Pero... ¿Cómo hacemos para que estas funciones se activen o se desactiven?

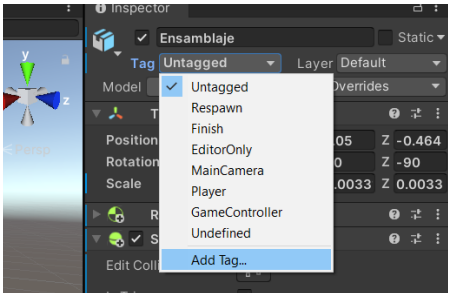
Los scripts "AumentaTamano.cs" y "DisminuyeTamano.cs" son los encargados de llamar a las funciones AumentarTamano(), DisminuirTamano() y StopTamano() definidas en el script "ModificarTamano.cs". Para ello he utilizado "OnTriggerEnter" y "OnTriggerExit". Anteriormente, ya he explicado cómo funciona y cómo se activa la primera función. En el caso de "OnTriggerExit", su funcionamiento es similar a "OnTriggerEnter". Pero en este caso, la función se va a ejecutar cuando los dos colliders dejen de colisionar (es decir, saquemos el componente afuera del área de modificación).

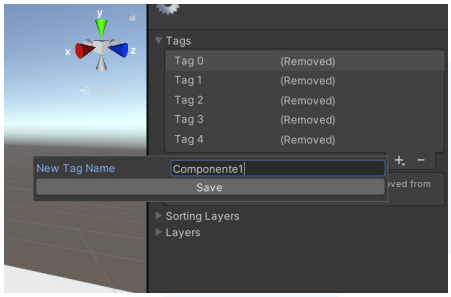
Una vez explicado el funcionamiento de ambas funciones, la siguiente pregunta que nos hacemos es: ¿qué es exactamente lo que tiene que colisionar para que esta función se active? La respuesta la sabemos: un

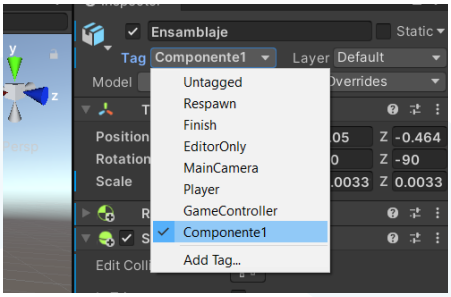
componente. Pero... ¿cómo sabe Unity que lo que ha colisionado con una de las áreas es un componente? La respuesta a esta pregunta la tenemos el los "Tags".

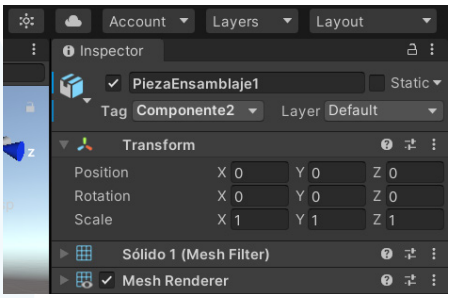
Un Tag es una palabra de referencia que se puede asignar a uno o más GameObjects. Con los tags, podemos identificar un GameObject dentro de un script. Estos se asignan de forma similar en cualquier GameObject:

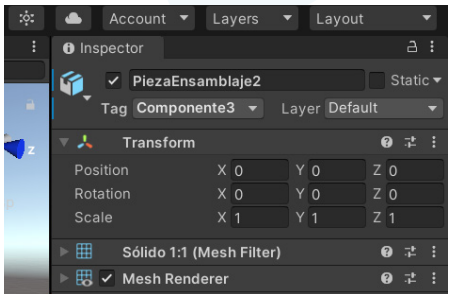
- Seleccionamos un GameObject cualquiera y vamos a la sección "Tag" (justo debajo de su nombre).
- Añadimos un tag nuevo a la lista haciendo click en "Add tag".
- Una vez añadido, volvemos a la lista anterior y escogemos el nuevo tag creado.

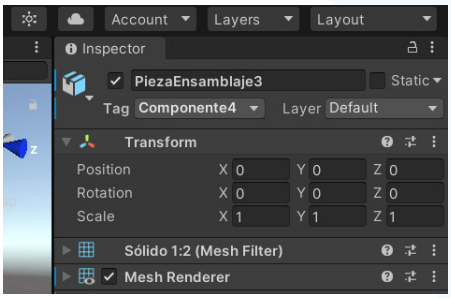




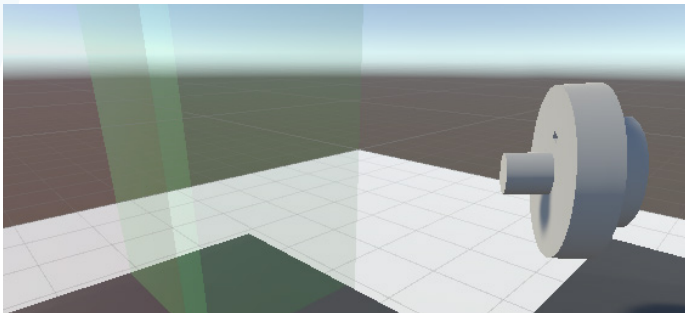


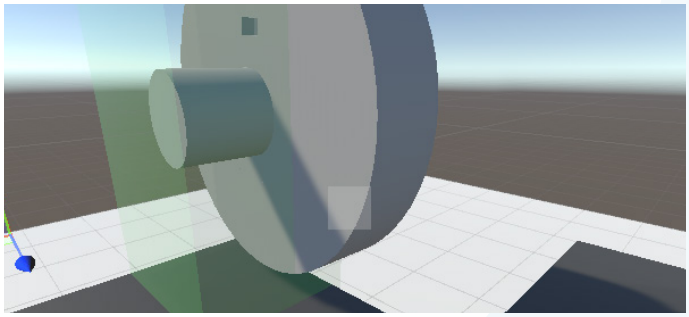


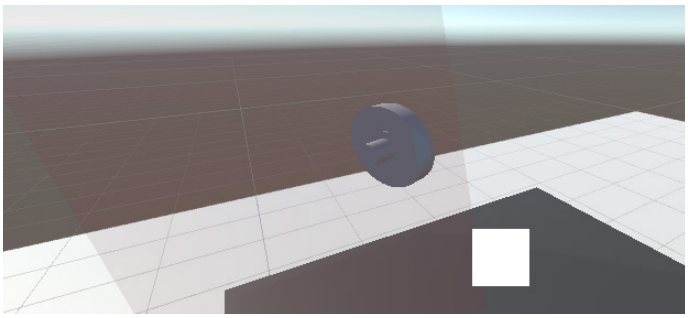


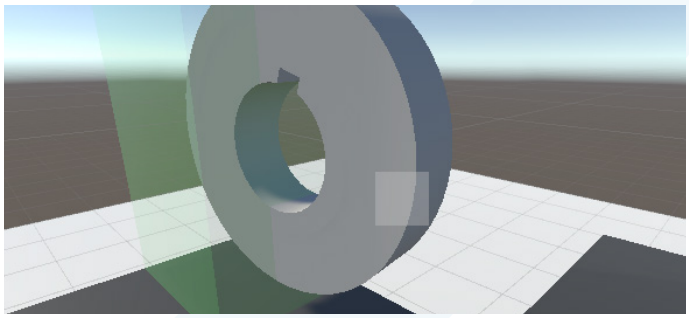


El resultado obtenido es el siguiente:









En el apartado "*2.1 Proyecto de pruebas*" del anexo "*Anexo II: Programación y scriptado*" del documento "*Anexos*", podemos encontrar el código de los scripts: ModificarTamaño.cs, AumentaTamano.cs y DisminuyeTamano.cs.

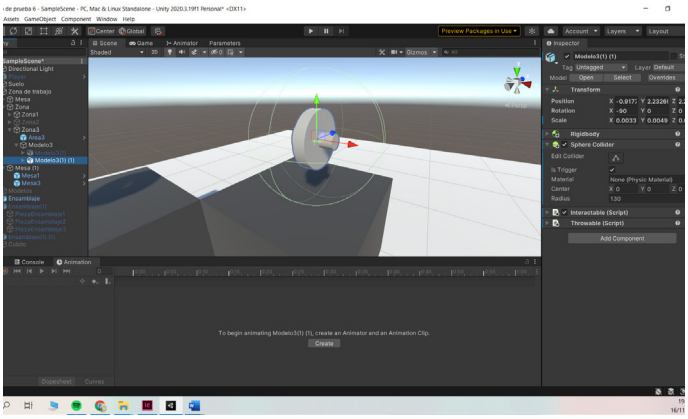
2.3.5 Pruebas III: Animaciones

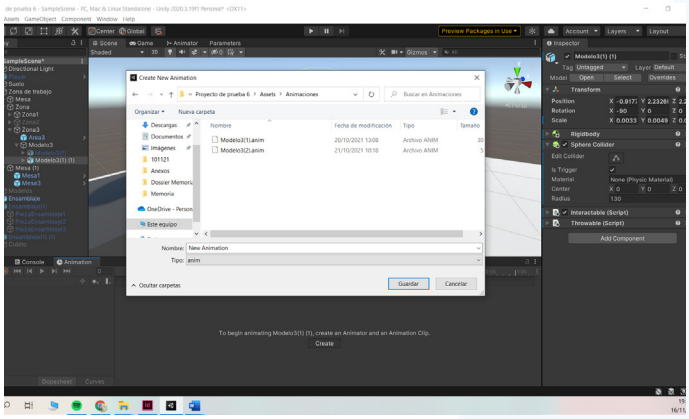
El objetivo del tercer y último apartado de pruebas es conseguir generar una animación con modelos 3D y poder visualizarla en nuestro entorno virtual. Lograr esto sería algo muy interesante y tendría una función muy útil dentro del software. Ya que, en caso de poder realizarse algo así, podríamos generar un explosionado virtual en tiempo real o representar un conjunto en funcionamiento.

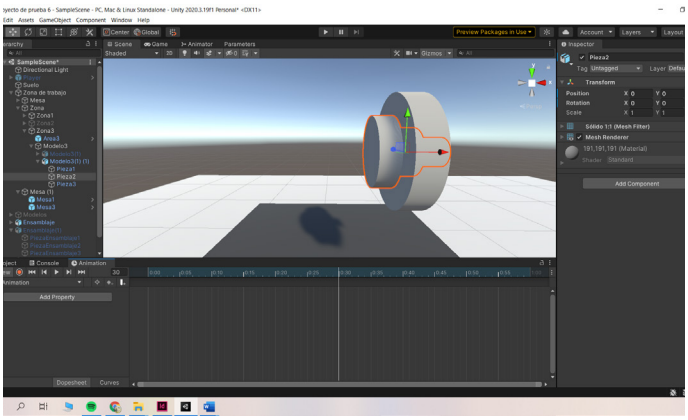
Animation Clip by Unity

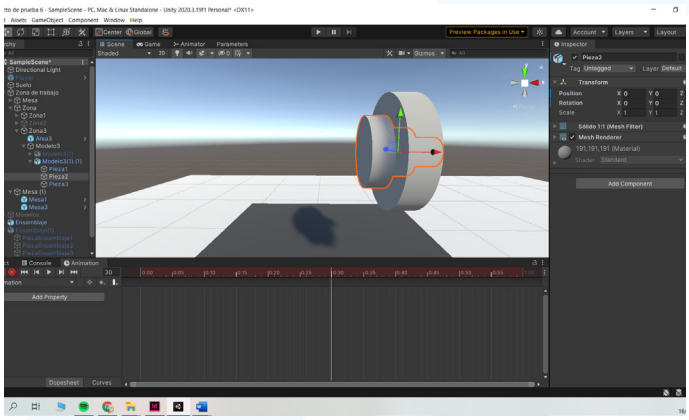
Para poder animar un GameObject dentro de Unity, el/los objeto/s necesitan tener asociado un componente denominado "Animator Component". Este componente hará referencia a un "Animator Controller", que coincidirá con el GameObject que queremos animar. Y este a su vez, contendrá referencias a uno o más clips de tipo "Animation Clip". Por lo tanto, podremos generar tantos clips como queramos, siempre que estos esten asociados a un GameObject. Y además, dentro de un mismo clip asociado a un GameObject "padre", vamos a poder animar diferentes GameObject "hijos" (lo veremos a continuación). Para crear nuestro nuevo clip deberemos seguir el siguiente procedimiento:

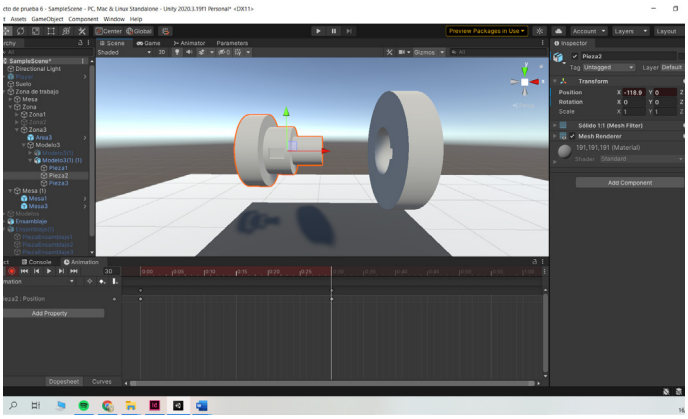
- Desde "Window" (ubicado en la parte superior) activamos la ventana de "Animation".
- Dentro del GameObject que vayamos a animar, creamos un nuevo clip. Esta se guardará en un archivo de tipo .anim dentro de nuestros Assets del proyecto. Aparte, he creado una nueva carpeta dentro de los Assets, destinada unicamente para guardar en ella las animaciones.
- Seleccionamos un GameObject "hijo" (uno de los componentes) y desplazamos la barra de tiempo blanca hasta un tiempo x. Este será el tiempo que tardará el componente en realizar el movimiento completo desde su posición incial (la actual) hasta su posición final (la que definimos a continuación).
- Ahora, pulsamos el botón de "record" situado en la parte izquierda del panel (vemos como la barra de medición del tiempo se tiñe de color rojo). Y una vez hemos entrado en el modo de grabación, movemos el componente hasta la posición que queramos (esta será su posición final del movimiento).

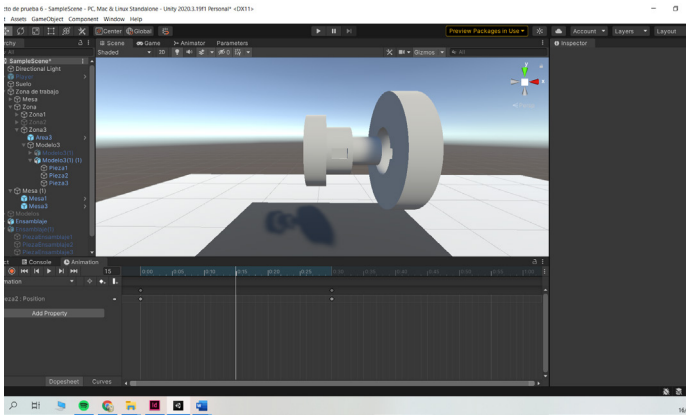








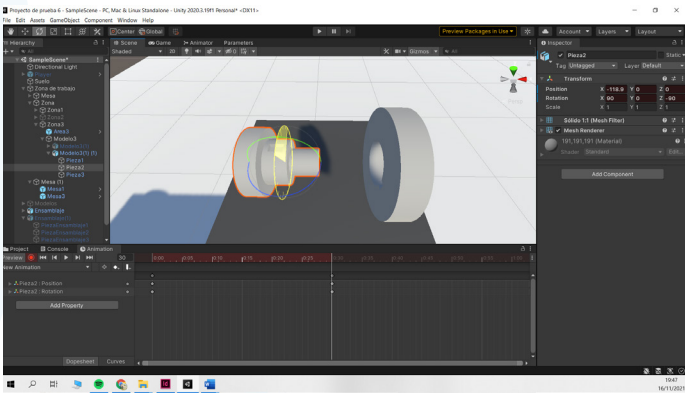


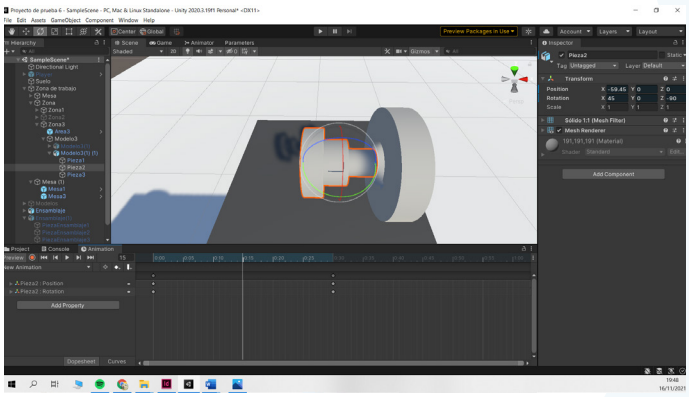


Una vez completado este procedimiento, en la barra del tiempo aparecerá un marcador (en la medición del tiempo) y este estará asociado a un indicador (en la parte izquierda). Estos indicarán lo siguiente:

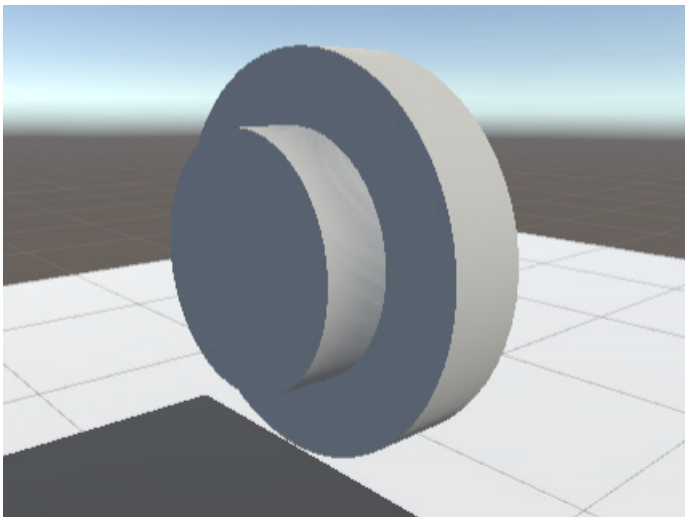
- GameObject que ha sido animado.
- Qué tipo de transformación a sufrido (posicion, giro, escalado etc...).
- El intervalo de tiempo en el que se realiza la animación.

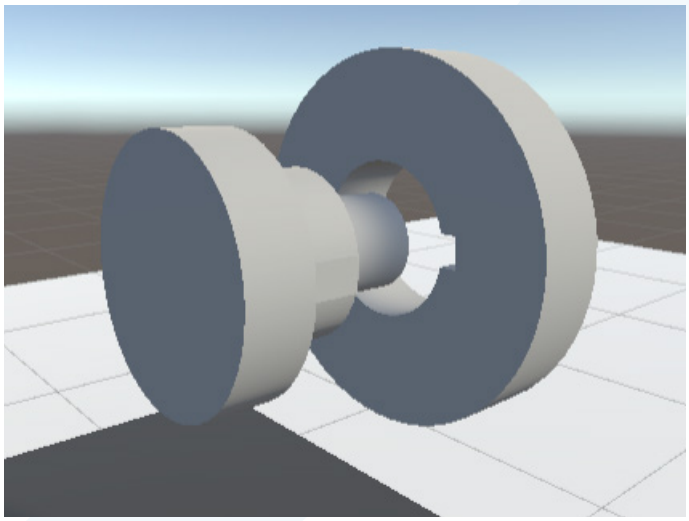
Para generar una animación giro, el prodedimiento a seguir sería el mismo. Y además, como podemos ver a continuación, podemos combinar ambos tipos de movimientos (posición y giro) en uno solo. El motor gráfico se encargará de solapar las dos transformaciones y crear la animación pertinente.

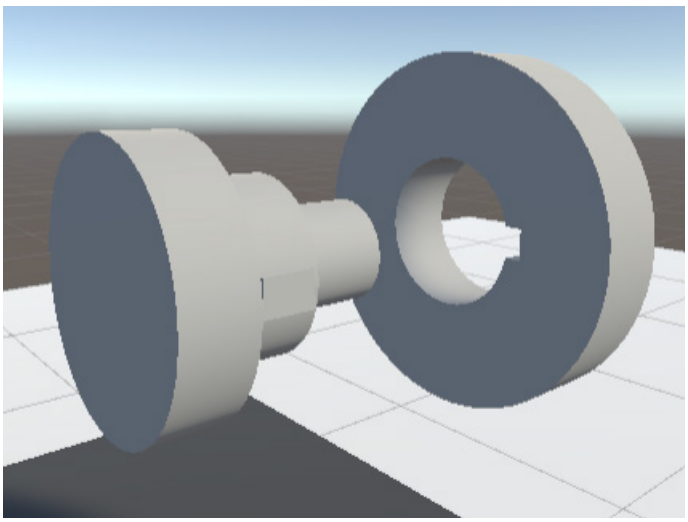
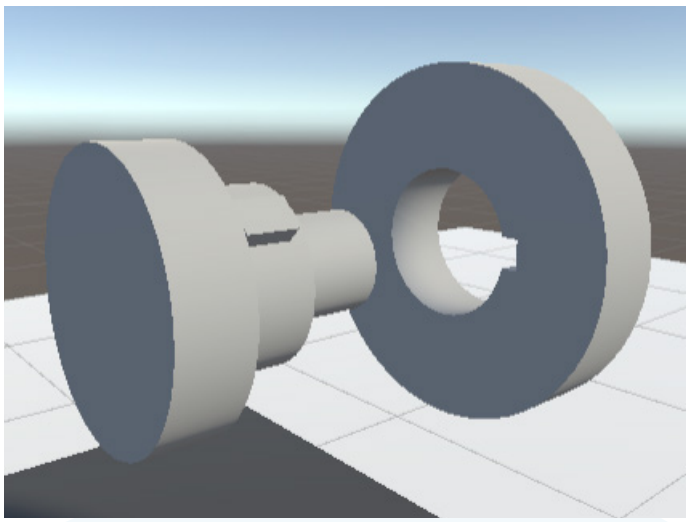
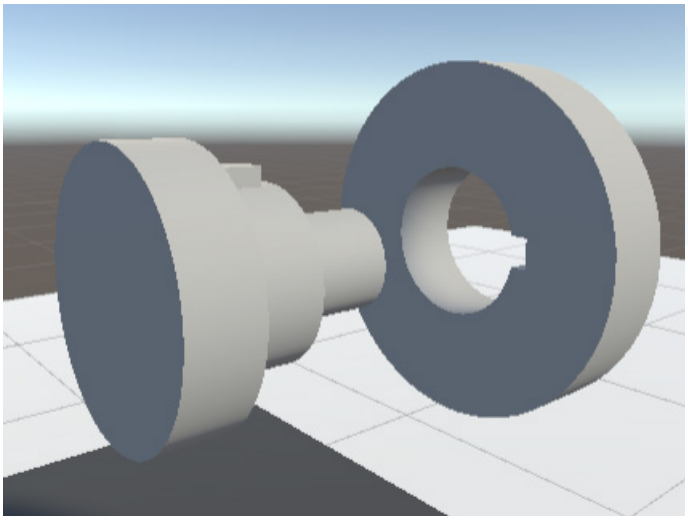
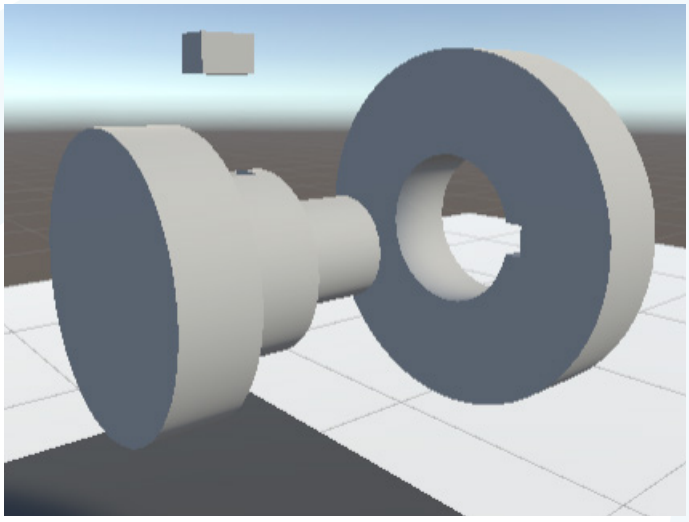




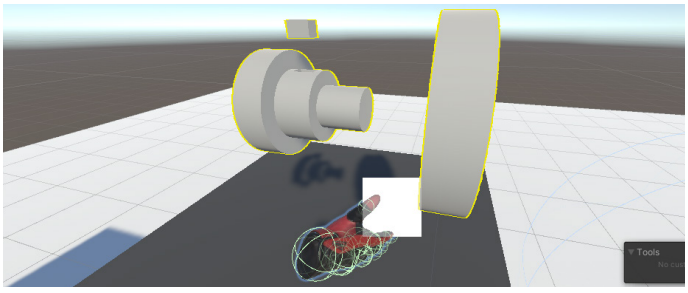
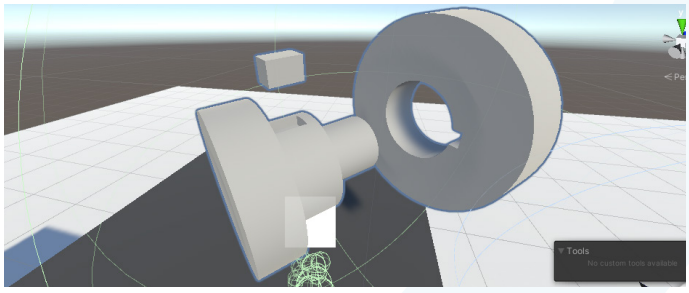
Siguiendo los pasos descritos anteriormente, he conseguido generar la siguiente animación. En ella, he realizado un explosionado virtual del modelo de pruebas.





Y para terminar, si a este GameObject le añadimos los componentes necesarios (explicados anteriormente en el subapartado *"Pruebas I: Manipulación"*), podremos manipularlo. De este modo, moviendo el conjunto a nuestro gusto, tendremos la posibilidad de ver la animación desde diferentes puntos de vista.

3 Creación de la escena virtual

- 3.1 Definición de la escena
- 3.2 Creación de la escena

3.1 Definición de la escena

Para poder empezar a construir y programar el software final, primero tengo que definir una serie de aspectos como: en qué lugar vamos a estar inmersos, qué ambientación busco para el entorno manipulable, qué tipo de interfaz se le va a ofrecer al usuario dentro del entorno, qué posibilidades va a tener etc... Por lo tanto, a continuación, voy a definir todo lo necesario para poder seguir un camino y un orden mientras voy construyendo este mundo virtual.

3.1.1 Entrada en el mundo virtual y escenario inicial

Durante la realización de la fase anterior "*FASE 2: Proyecto de pruebas, investigación y aprendizaje*", realizaba las pruebas sin tener que levantarme de mi silla. Fué posible, ya que como comenté en el subapartado *"2.3.1 Montaje de la escena"* de este mismo documento, situé al personaje virtual en el centro de la mesa de trabajo que había creado utilizando tres cubos. Esto, lo hice por simple comodidad y por no tener que estar levántandome constantemente. Pero a su vez, me dió una idea.

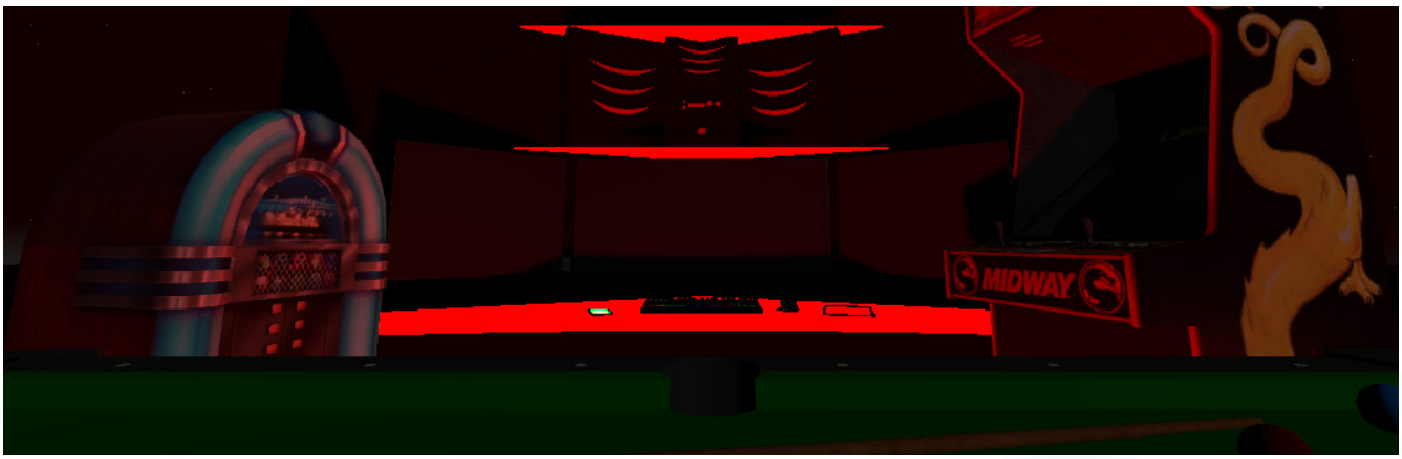
Pensé en que el software que iba a crear, a fin de cuentas, no iba a poder suplantar un entorno de trabajo por completo. Porque para ello, debería contar con una interfaz de usuario mucho más amplia, en la que por ejemplo, pudiésemos escoger entre varios conjuntos antes de tener uno cargado, pudiésemos ajustar diferentes tipo de parámetros etc... (cómo por ejemplo, las que se usan en películas de ciencia ficción). Pero generar un software con todas estas características, no se encuentra a mi alcance.

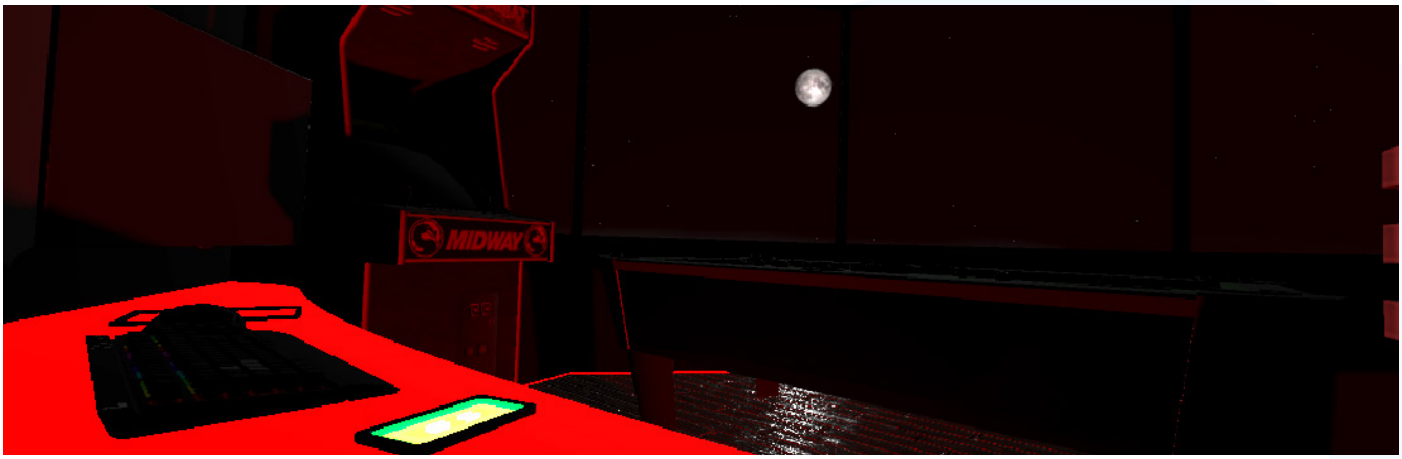
Pero por otro lado, tampoco quiero desarrollar algo que no tenga una utilidad posterior. Por lo tanto, todo esto me llevó a ampliar esa idea que había tenido en un principio como comentaba anteriormente.

La escena virtual que voy a crear, se trata exactamente de la misma que tengo delante mio en el mundo real. Pero a su vez, será un entorno completamente distinto. ¿Qué quiero decir con esto?

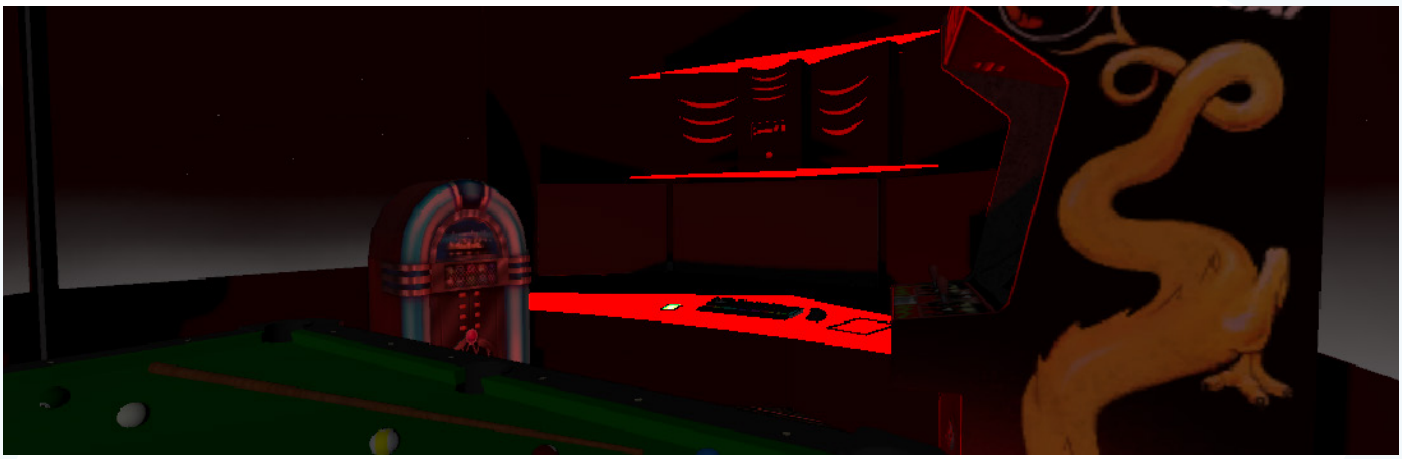
Con la idea de "mismo entorno que hay delante de mí" me refiero a que, cuando acceda a este nuevo entorno ficticio, voy a seguir sentado en mi escritorio, con mis pantallas delante y con un teclado y un ratón debajo de mis manos. Primero, en el mundo real, voy a tener una carpeta con el proyecto del conjunto. En ella, estarán todos los archivos existentes acerca de ese modelo en concreto (piezas, ensamblajes, renders, vídeos con animaciones, pdfs etc...). Pero aparte de estos archivos que siempre hemos utilizado en la escuela, tendremos uno más. Un archivo de tipo ejecutable .exe con el siguiente nombre "ConjuntoXX_VR". Este archivo, al abrirlo y tener conectado el equipo de realidad virtual en nuestro ordenador, nos creará la ilusión de seguir sentados en nuestra zona de escritorio, pero a su vez, viajaremos a un mundo diferente.

Pero ya que estamos en el mundo virtual y puedo tener lo que yo quiera... voy a hacerle algún cambio a mi set up y a mejorar un poco las vistas de alrededor.


MIDWAY


MIDWAY


MIDWAY



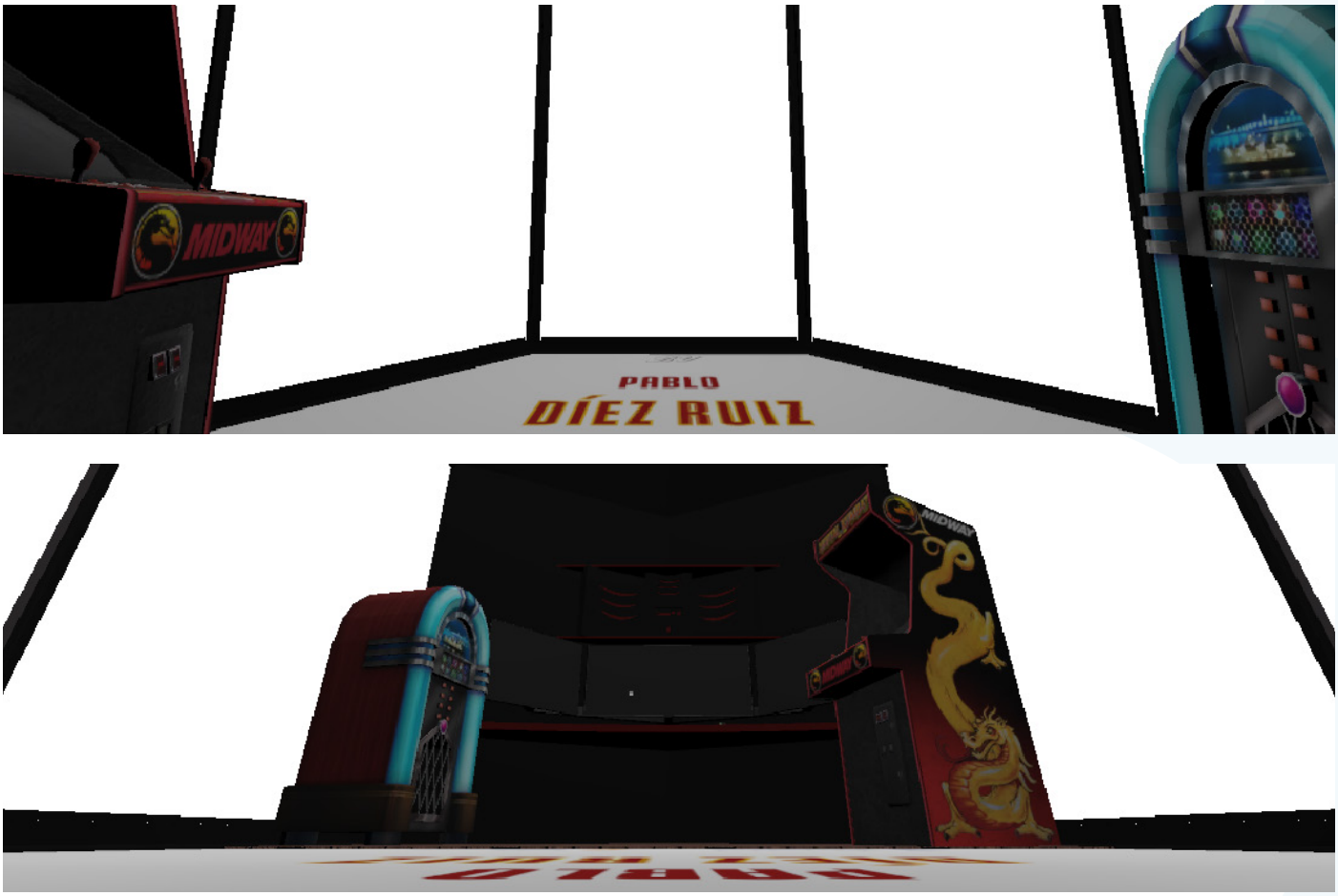
3.1.2 Sala de trabajo interactiva

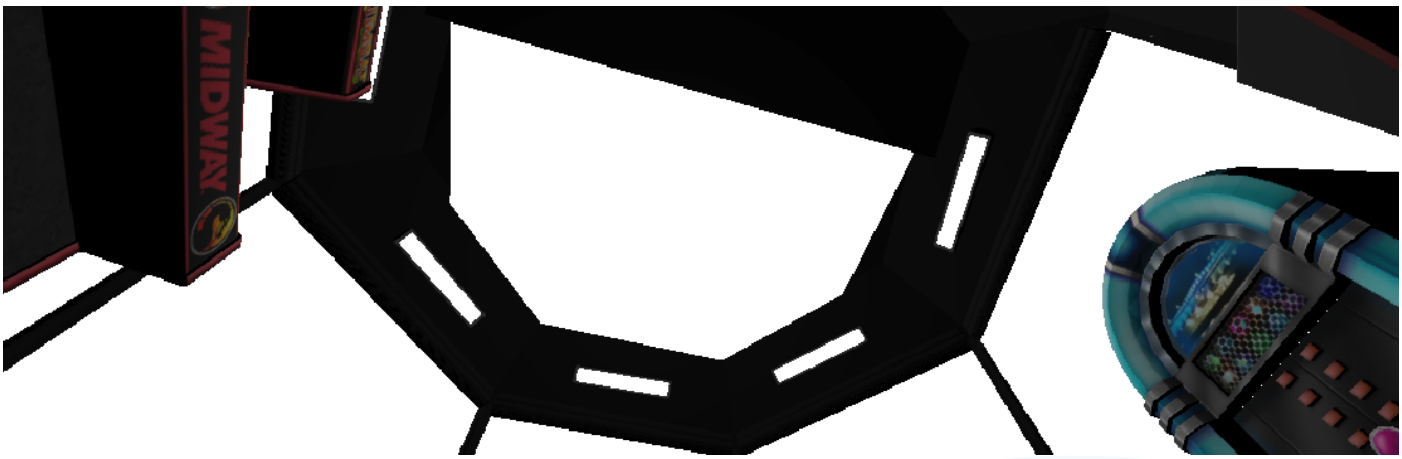
Una vez nos encontramos dentro del mundo virtual, solamente queda acceder al funcionamiento principal del software. Como he definido de forma precisa en el subapartado "0.2 Objetivos del proyecto" de este mismo documento", el usuario debe tener la posibilidad de realizar acciones como: visualizar animaciones y el explosionado de un conjunto en tiempo real, manipular con sus propias manos sus componentes y realizar diferentes operaciones sobre estos.

Es lógico pensar que, para que esto sea posible, necesitaremos un espacio libre y medianamente amplio. De esta forma, el usuario tendrá la posibilidad de moverse libremente por el entorno mientras va visualizando y manipulando los modelos 3D.

Además, la iluminación de esta zona de trabajo, también debe de ser la adecuada. No servirá de nada contar con un espacio amplio y con modelos 3D amplios, si no contamos con una iluminación correcta. El conjunto y sus componentes deben verse de la mejor forma posible y, sobre todo, debe apreciarse de forma excelente el contorno de las mismas formas que delimitan su geometría. Ya que, por ejemplo, existen componentes normalizados que tienen formas complejas y dificiles de entender a simple vista. Y, para su correcta comprensión, el usuario debe tener la posibilidad de apreciar la totalidad de su forma con el máximo nivel de detalle posible.

Por lo tanto, dentro de la misma habitación virtual, definida anteriormente en el subapartado *"3.1.1 Entrada en el mundo virtual y escenario inicial",* he creado un nuevo espacio que reune todas las características necesarias (espacio vacío, libertad de movimiento, iluminación buena, cortinas reflectantes para crear el mayor contraste posible con los modelos 3D etc...).

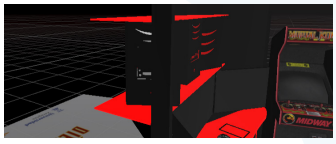


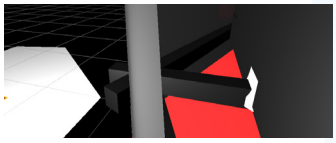


3.2 Creación de la escena

Para crear en entorno mostrado anteriormente en el apartado "*3.1 Definición de la escena*" de este mismo documento, he utilizado tres programas: Inventor Professional 2018, SketchUp 2019 (para el modelar y exportar modelos 3D en extensión .obj) y Unity (donde he importado los modelos y he ido montando paso a paso la escena).

3.2.1 Modelado de componentes



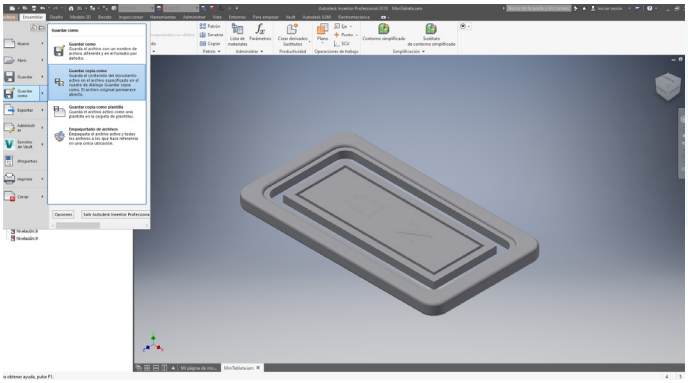
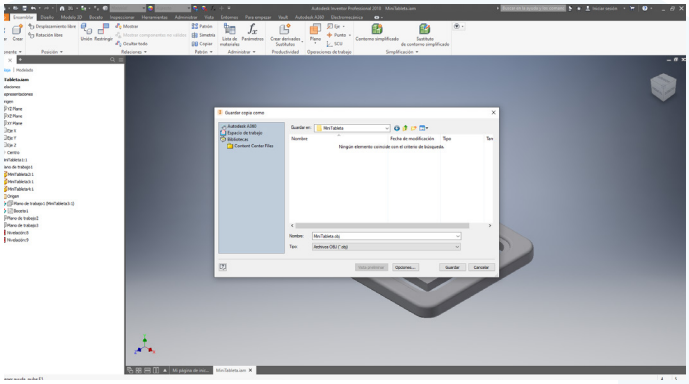
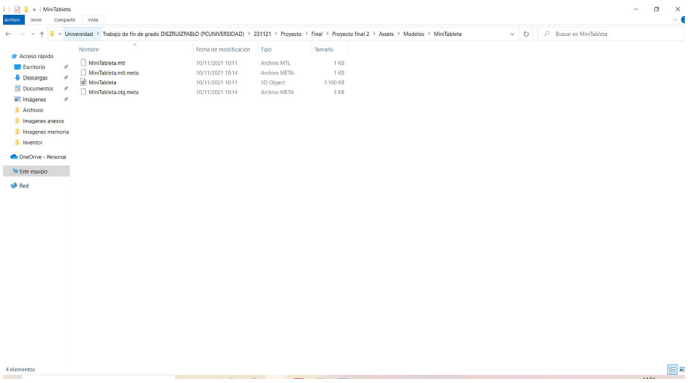
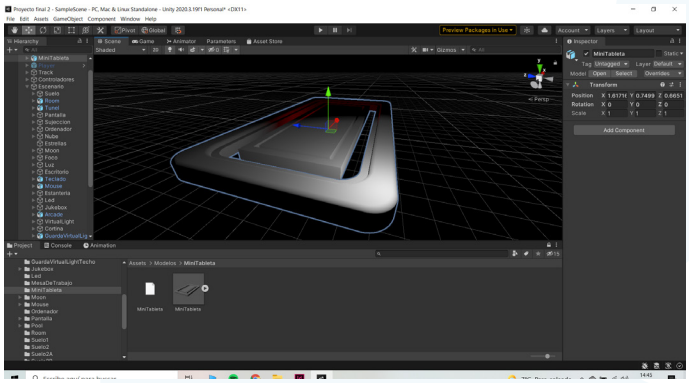
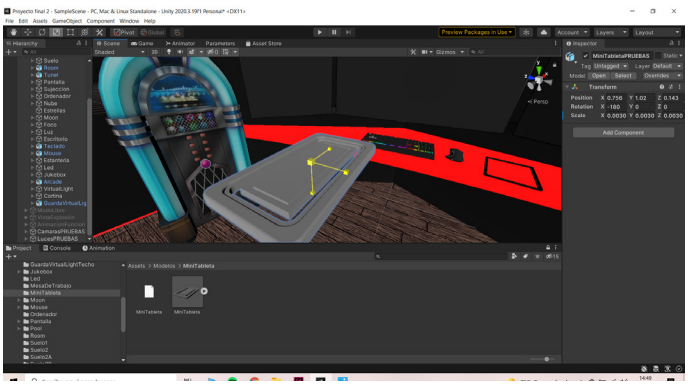
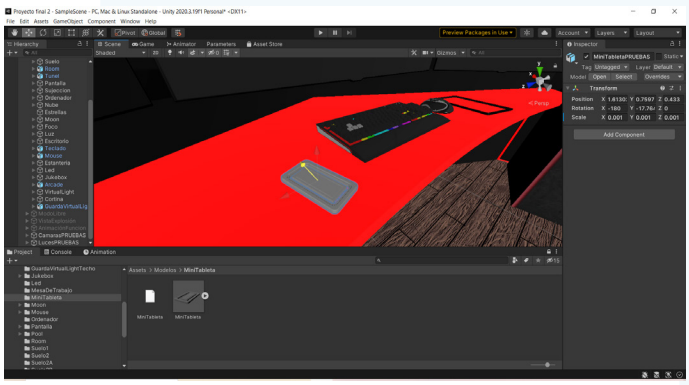


Los componentes que he creado yo mismo y he modelado en Inventor son los siguientes: Tableta, MiniTableta, Room (ensamblajes), Suelo, Tunel y Escritorio (piezas). Y por último, dentro de Unity, he creado los siguientes componentes: Estantería, Sujección, Nube y Estrellas. Los dos primeros componentes se tratan de un simple cubo reescalado y colocado en su posición. Al sobrepasar otros componentes (como la pared de la sala o las mismas sujecciones entre sí), estas partes no se ven. Y por lo tanto, estos componentes parece que tienen otra geometría distinta y que encajan en sus respectivos sitios de manera exacta.

Las nubes y las estrellas las he creado utilizando el "ParticleSystem" (sistema de partículas) de Unity. Esto, se definirá de forma detallada en subapartado "*3.2.3 Creación de materiales, iluminación y texturización*" de este mismo documento.

Para crear el modelo "MiniTableta", exportarlo en extensión .obj, importarlo en Unity y colocarlo en la posición deseada dentro de la escena, he seguido el siguiente procedimiento:

- Modelado de las piezas que componen el ensamblaje: "MiniTableta1", "MiniTableta2", "MiniTableta3" y "MiniTableta4". Después, las he ensamblado creando el ensamblaje "MiniTableta".
- Dentro del archivo del ensamblaje, exportamos el modelo en extensión .obj. Para ello, debemos guardar una copia del modelo, y ahí, nos dejará exportar el archivo con este tipo de extensión.
- Después, debemos introducir los archivos exportados en nuestra carpeta de "Assets" del proyecto. Para este proyecto, he creado diferentes carpetas dentro de los assets y así tener todo bien organizado (modelos, materiales, audio, imágenes, animaciones, scripts etc...).
- Acto seguido, vamos a Unity y arrastramos el modelo MiniTableta.obj hasta nuestra escena. El modelo se cargará en la escena con las dimensiones que tenga guardadas de forma materna. Por lo tanto, deberemos escalar y transformar su posición hasta la que nosotros deseemos.

Para crear los demás componentes y colocarlos en una posición determinada dentro de la escena, he seguido el mismo procedimiento. Para poder ver todos los modelos de las piezas modeladas, dirigirse al apartado "*3.1 Modelado en Inventor Proffesional 2018*" del anexo "*Anexo III: Modelado e importación de componentes*" del documento "*Anexos*"

¿Por qué he hecho el modelo por separado y no todo junto? La respuesta tiene que ver con la programación de los colisionadores. En este caso (MiniTableta), realmente no tiene mucha importancia, ya que, cuando ejecutamos el software, esta se encontrará encendida. Pero por ejemplo, para el caso del componente "Tableta", el cual nos ofrecerá la interfaz de usuario del software, si que tiene mucha importancia, ya que ese componente, cuando iniciemos el software, estará "apagado". Y por lo tanto, para hacer que la pantalla de la tableta se encienda, debemos controlar la activación y desactivación de uno o de algunos de sus componentes.

Pero... ¿no había dicho que el problema estaba en los colisionadores?. Pues así es, ya que, para controlar la activación y desactivación de componentes, Unity nos permite utilizar la función "SetActive(true/false)" dentro de un script. Esta función activará o desactivará el GameObject que deseemos. Pero a su vez, también activará o desactivará todos los componentes asociados a este GameObject. En el caso que nos intersa, su colisionador. Para que este no se active antes de tiempo (cuando la tableta esta apagada), podemos aprovecharnos de esto y mantenerlo desactivado hasta que llegue el momento.

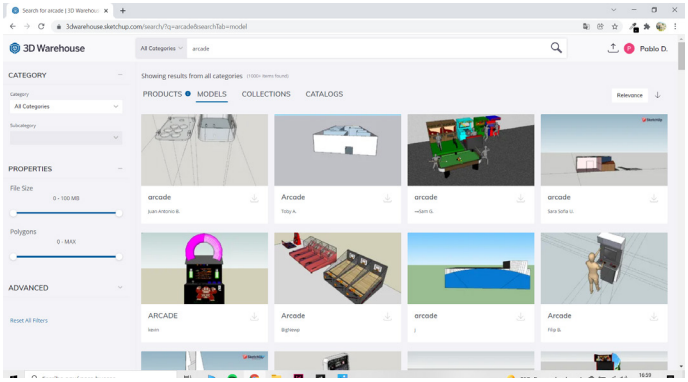
Pero creo que nos hemos adelantado un poco. Todo esto cobrará sentido más adelante y se explicará con más detallamiento durante la "*FASE 4: Montaje de la sala de trabajo interactiva y programación del software*", en donde entenderemos el por qué de la activación y desactivación de los GameObjects y el uso que le daremos para poder activar nuestra zona de trabajo interactiva.

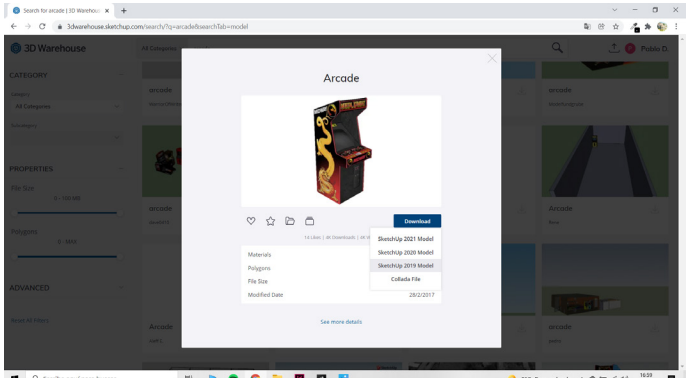
3.2.2 Importación de modelos desde la Warehouse de SketchUp

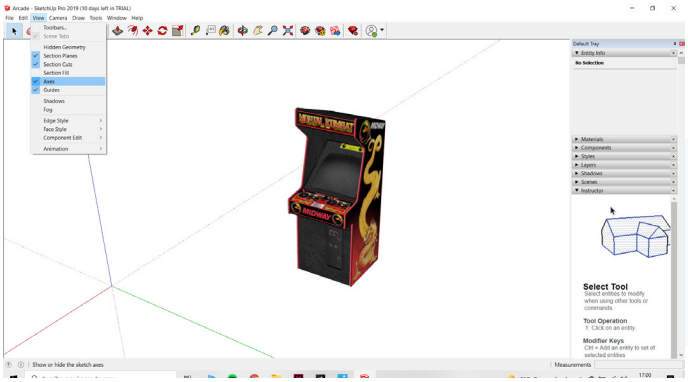
Para completar la escena, he incorporado una serie de modelos en 3D desde el siguiente enlace: "*https://3dwarehouse.sketchup.com/?hl=es*". A través de este, se puede acceder a la "Warehouse" del programa SketchUp. En esta web, podemos encontrar modelos en 3D de todo tipo (paisajes, estructuras, mobiliario, personajes, automóviles etc...). Estos modelos son creados por los usuarios y los suben a esta web para que otros puedan utilizarlo en sus proyectos.

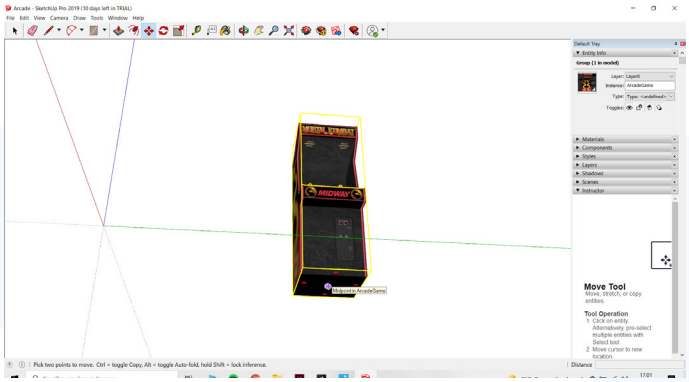
Para incorporar estos modelos en mi escena virtual, he seguido el siguiente procedimiento:

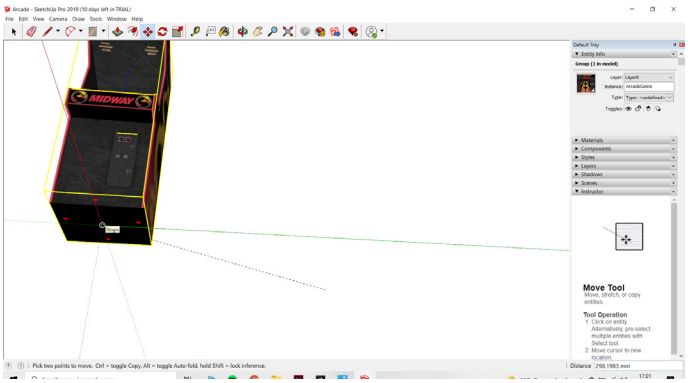
- Primero entramos en la web desde el enlace que he citado anteriormente. Una vez allí, debemos iniciar sesión en nuestra cuenta de SketchUp. En caso de no tener una cuenta, debemos crearla. De lo contrario, no nos dejará descargar los modelos disponibles en de su Warehouse.
- Después, buscamos el tipo de modelo que necesitemos, introduciendo una palabra o palabras claves en la barra del buscador y clickando en la opción "models". Y una vez hemos elegido el modelo, lo descargamos como archivo "SketchUp2019" o por el contrario, en la versión que tengamos instalada en nuestro pc.
- Abrimos el archivo y activamos la vista de los ejes de coordenadas. Después, sobre el grupo entero (click derecho y "make component"), posicionamos el centro del modelo con el centro de coordenadas. Este paso es muy útil realizarlo, ya que cuando importemos el modelo en Unity, las transformaciones que hagamos dentro del programa (posición, rotación y escala) se harán respecto a estos ejes, no respecto al modelo. Por lo tanto, es importante tener el modelo bien centrado y ajustado a los ejes de coordenadas globales.
- A continuación, exportamos el modelo en extension .dae (archivo collada). Al exportar el archivo con esta extensión, no solo nos guardará los archivos de malla del modelo 3D, sino que también nos guardará las texturas que el modelo lleva incorporadas.
- Y por último, ajustamos el modelo en su posición de la misma forma que hemos hecho para el componente "MiniTableta" en el subapartado "*3.2.1 Modelado de componentes*" de este mismo documento. También deberemos eliminar o desactivar la cámara del modelo. Esta cámara se ha importado desde SketchUp al importar el modelo en extensión .dae. Si no la eliminamos, al ejecutar el software, se activará esa cámara.

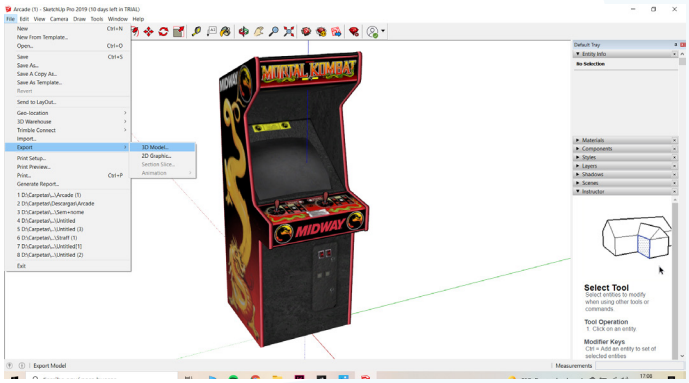


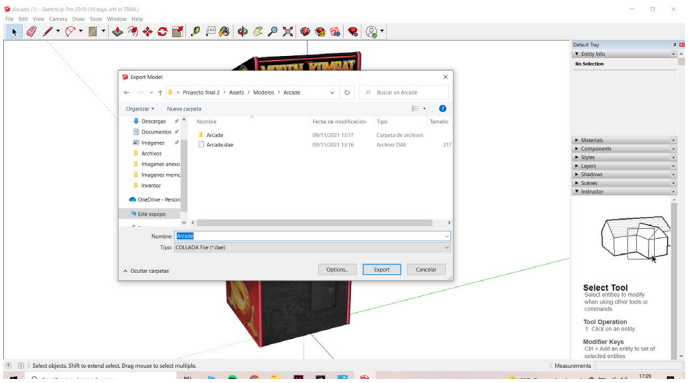


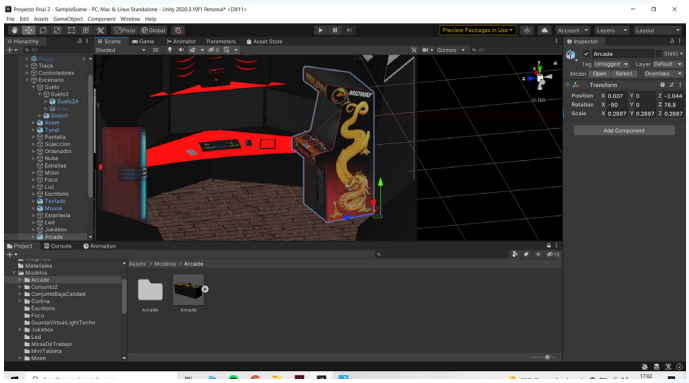








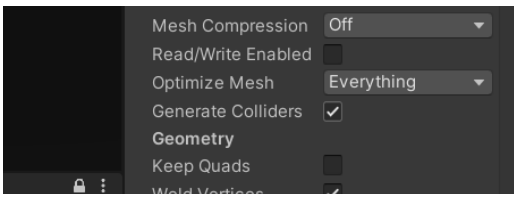




Para añadir más modelos e introducirlos en mi escena, he seguido el mismo procedimiento que el descrito anteriormente. Para poder ver todos los modelos preparados en SketchUp 2019 y que posteriormente, he añadido a mi escena, dirigirse al apartado "*3.2 Modelos exportados desde SketchUp 2019*" del anexo "*Anexo III: Modelado e importación de componentes*" del documento "*Anexos*".

Anteriormente, en el subapartado "*2.3.3 Pruebas I: Manipulación*" de este mismo documento, hemos visto como podíamos crear los colisionadores "colliders". Como ya sabemos, para que nuestras manos choquen

con la geometría de cualquier modelo, necesitamos incorporar un colisionador tanto en estas como en los modelos. Unity nos ofrece la posibilidad de generarnos automáticamente los colisionadores de los modelos que importemos, ciñéndose a la propia geometría del mismo. Para ello, activamos la siguiente casilla:



Esto lo podemos hacer tanto para los modelos importados desde SketchUp como para los modelos importados desde Inventor Professional. Gracias a ello, no tendremos que estar generando todos los colliders uno a uno para cada modelo. Pero entonces... ¿por qué no hacerlo así siempre? La respuesta es muy simple: memoria.

Tanto los archivos exportados en .obj desde Inventor Professional como los archivos expotados en .dae desde SketchUp, no tienen un gran tamaño. La geometría de los modelos que he modelado son muy simples y por lo tanto, el número de triángulos que ha generado la malla al exportarse en .obj, también es muy bajo. Y para el caso de los archivos procedentes de la Warehouse, no he buscado archivos muy simples, ya que se trata de un software en realidad virtual y la experiencia para el usuario debe ser buena. Pero tampoco he escogido modelos con un número de polígonos muy alto. Por ejemplo, haciendo pruebas, cargué un modelo de ordenador que contaba con más de 200.000 triángulos en su malla y, al ponerme el casco de realidad virtual y ejecutar el software, este se relentizaba mucho y se notaba mucho que el archivo pesaba demasiado para poder ser renderizado en tiempo real y seguir teniendo la misma fluidez durante su uso.

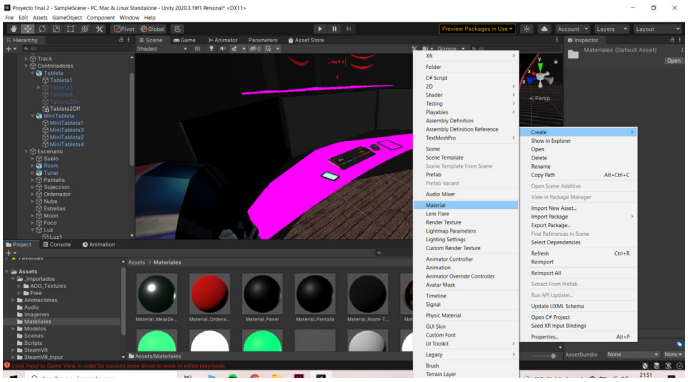
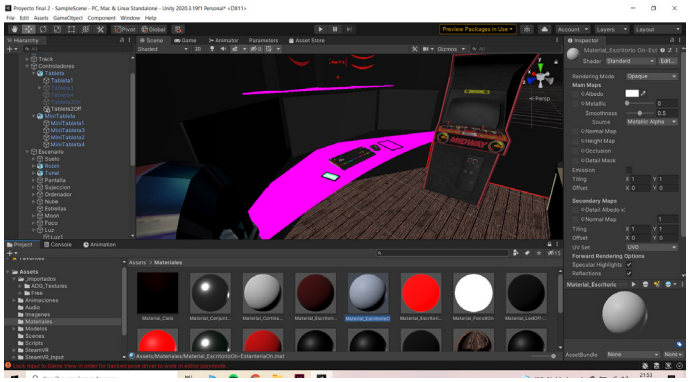
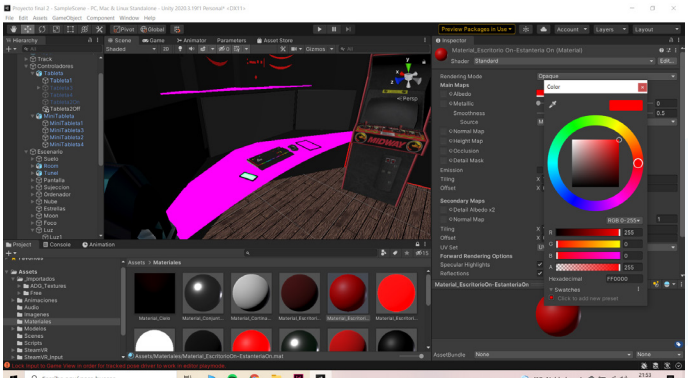
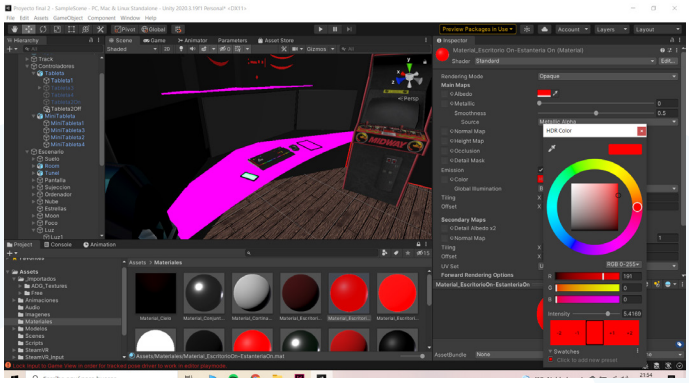
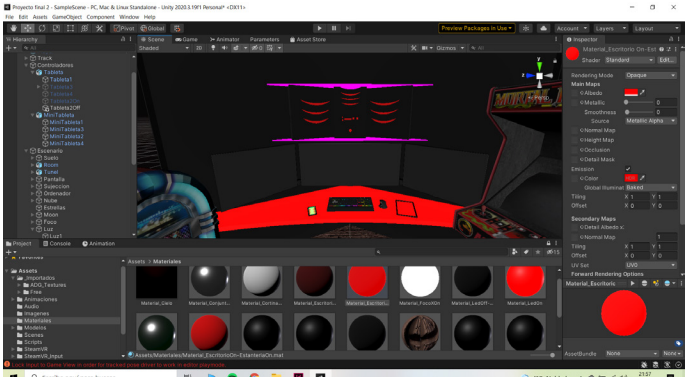
Como ya he dicho antes, Unity genera de forma automática los colisionadores ateniéndose a la geometría de propia malla del modelo. Por lo tanto, a mayor número de triángulos, mayor complejidad en la geometría de la malla que delimita el colisionador del modelo. Y es por esta razón, que Unity ofrece la posiblidad al usuario de generar sus propios colisionadores con formas geométricas muy simples (cubos, esferas, cilindros...). Y de hecho, es algo que se utiliza continuamente en el desarrollo de videojuegos. Ya que, como podemos imaginar, las mallas de los modelos 3D de algunos videojuegos muy realistas y futuristas, son enormes. Y por lo tanto, se debe simplificar la forma de sus colisionadores, porque en caso contrario, el videojuego tendría un tamaño excesivamente grande y además, ni el mejor de los procesadores ni la mejor de las tarjetas gráficas actuales, podría ejecutar todo en tiempo real y mantener la fluidez que se exige hoy en día en ese tipo de videojuegos.

3.2.3 Creación de materiales, iluminación y texturización.

Asignación de materiales

Una vez creado un GameObject o hayamos importado un modelo 3D, Unity nos da la posibilidad de añadirle un color determinado. Para ello, debemos crear un material nuevo y sobre este, ajustaremos los parámetros a nuestro gusto para conseguir el tono que buscamos (color, emisión de luz, ajuste del texturizado etc...). Por ejemplo, para darle color al escritorio y a las estanterías, he seguido el siguiente procedimiento:

- Primero creamos un nuevo material (recomendable hacerlo dentro de una carpeta de "Materiales" creada en los assets del proyecto (por organización).
- Después, le damos un nombre. Para organizar mejor los materiales, he creado la siguiente nomenclatura: Material_Componente(X1,X2...)-Componente(X1,X2)-...
- Una vez tenemos creado el material, ajustamos el color y los diferentes parámetros asociados. Además, tenemos la posibilidad de añadirle la emisión de luz del color que deseemos.
- Por último, para añadir este material a un objeto de la escena, debemos arrastrar el material desde la carpeta donde lo tengamos guardado hasta el objeto deseado. Hay que hacer lo mismo para cada objeto.


He seguido el mismo procedimiento para cada uno de los modelos y GameObjects de la escena. Para ver los parámetros de cada uno de los materiales, dirigirse al subapartado "*4.1.1 Parámetros*" del anexo *"Anexo IV: Materiales, texturas e iluminación"* del documento "*Anexos*". Y para ver el resultado obtenido de la asignación de estos materiales a sus respectivos modelos, dirigirse al subapartado "*4.1.2 Resultado*" del anexo *"Anexo IV: Materiales, texturas e iluminación"* del documento "*Anexos*".

Asignación de texturas

Aparte de poder generar un material nuevo y variar sus parámetros a nuestro gusto, Unity nos da la posibilidad de añadirle texturas a estos materiales. Una textura puede estar compuesta de diferentes canales y este programa, trabaja con los siguientes: Albedo, Metallic, Roughness, Normal Map, Height Map, Occlusion, Detail Mask. Y, gracias a estas, los materiales pueden conseguir acabados excelentes (como por ejemplo, simular rugosidades, cavidades, suciedad etc...).

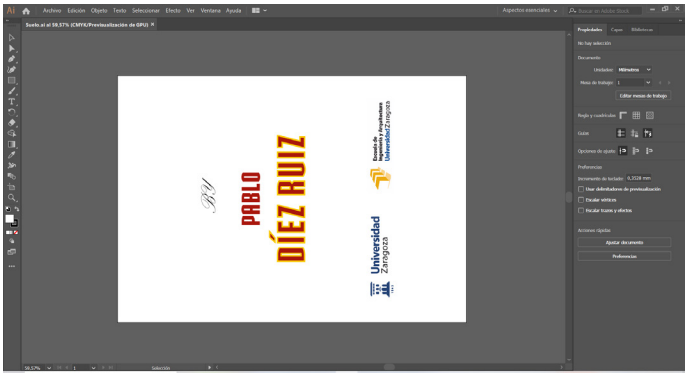
Existen programas especializados con los que podemos crear nuestras propias texturas partiendo desde cero. En ellos, crearemos los diferentes canales y los modificaremos a nuestro gusto. Por ejemplo, el programa FilterForge, no solamente nos da la posibilidad de crear nuestra propia textura, sino que también cuenta con su propia galería de filtros y texturas generadas por creadores y por usuarios especializados. Y además, podemos descargar alguna de estas texturas y editarla a nuestro gusto dentro del programa.

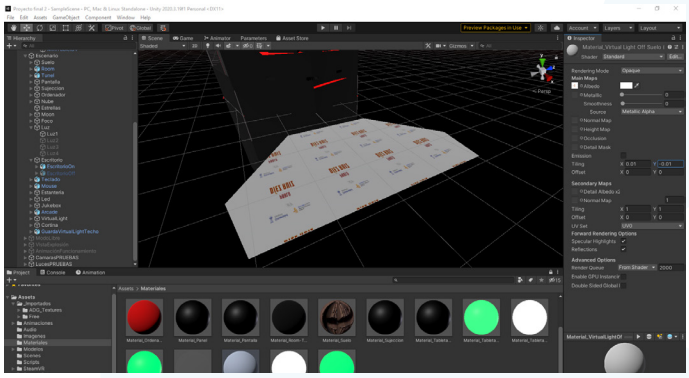
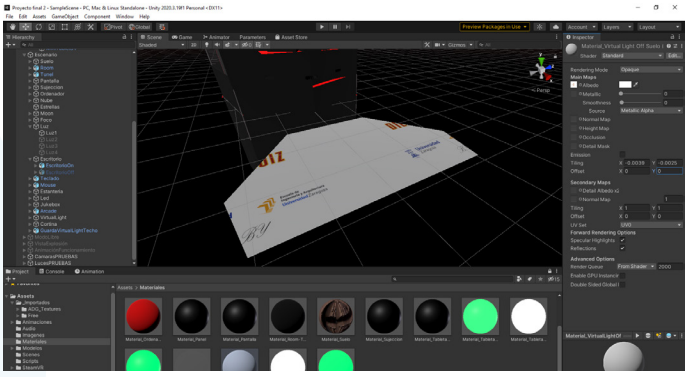
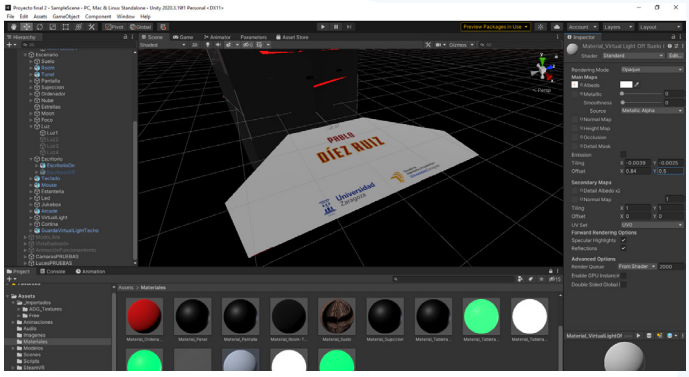
Para este proyecto, no he creado ninguna textura, ya que, salvo para el suelo, no he tenido la necesidad de simular ningún otro tipo de material especial dentro de mi escena.

La textura del suelo la he descargado desde la "Asset Store" de Unity. Es desde este mismo sitio desde donde descargamos e instalamos Steam VR Plug-in en el subapartado "*2.1.2 Conexión HTC Vive Pro - Unity*" de este mismo documento. Para obtener información sobre como acceder a la "Asset Store" de Unity, dirigirse al apartado "*1.5 Instalación del software y paquetes necesarios*" del anexo "*Anexo I: Documentación e investigación*" del documento "*Anexos*".

Esta textura denominada "Wooden Floor 3" incorpora archivos en los siguientes canales de texturización del material: Albedo, Displacement, Gloss, Metalness, Normal y Roughness. Unity nos ofrece la posibilidad de cambiar el archivo del canal que nosotros queramos. Pero para explicar esto, no retocaré ningún canal de texturización de este material, ya que me gusta como está. Para hacerlo, describo a continuación el procedimiento que he seguido para añadirle las letras y los logotipos al GameObject "VirtualLightOffSuelo".

- Primero, he creado el archivo en Adobe Illustrator 2019 y lo he exportado en jpg.
- Después, lo he añadido a los assets del proyecto (dentro de una carpeta que he creado denominada "Imagenes").
- A continuación, he seleccionado el material que había creado específicamente para este modelo. Este material, anteriormente, ya lo había asignado al GameObject. Por lo que, una vez realicemos el cambio, la actualización en la escena se producirá de forma automática.
- Una vez seleccionado el material, he buscado el archivo .jpg en su carpeta correspondiente y lo he arrastrado hasta el pequeño recuadro que hay en la parte izquierda del canal "Albedo".
- Por último, he modificado el "Tiling" y el "Offset" del material para poder ajustar con exactitud la imagen con la geometría del modelo.



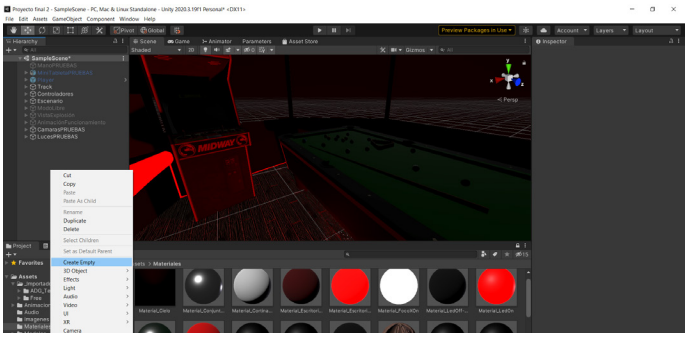
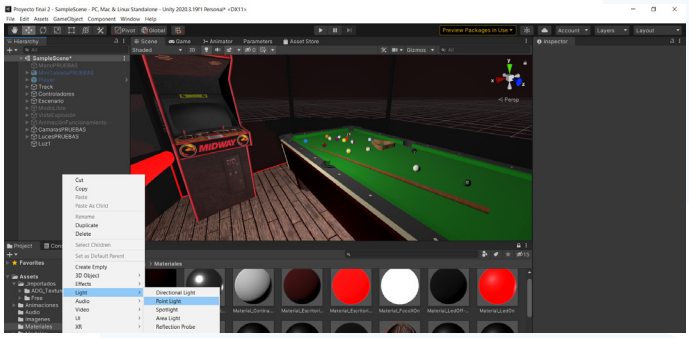
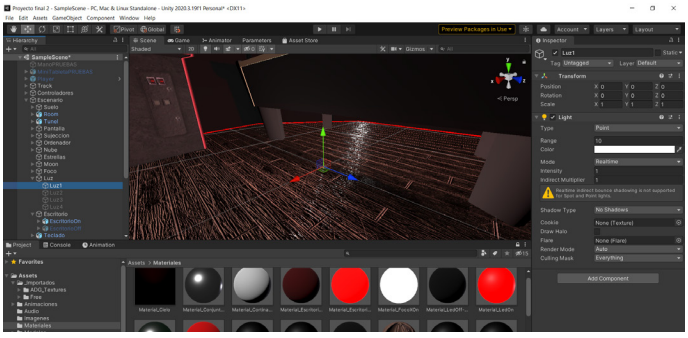
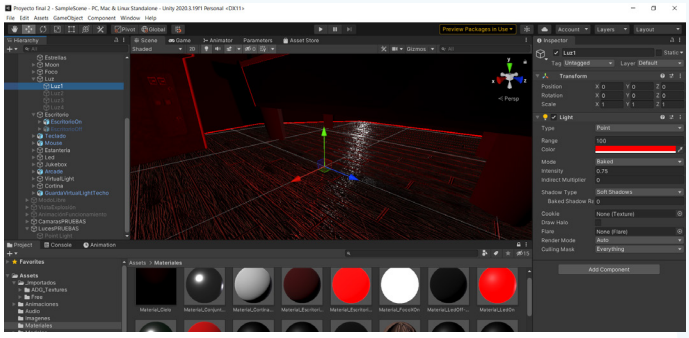
En caso de querer visualizar los canales de texturizado de las texturas utilizadas en el proyecto, dirigirse al apartado "*4.3 Canales de texturizado*" del anexo *"Anexo IV: Materiales, texturas e iluminación"* del documento "*Anexos*".

Luces e iluminación

Para añadir iluminación a nuestra escena, Unity nos ofrece diferentes tipos de luz: Directional Light, Point Light, Spotlight, Area Light, Reflection Probe y Light Probe Group. Para este proyecto he utilizado el tipo de luz "Point Light". Este tipo de luz genera un GameObject, el cual, desde un punto determinado, ilumina su alrededor, pudiendo modificar una serie de parámetros. Con estos, podremos modificar las características de la luz reflectada por el GameObject. Algunas de estas son: el rango de iluminación, la intensidad de la misma, el modo en el que lo hace, el tipo de sombras que generará sobre los modelos que esta incida etc...

Para crear la luz con tono rojizo de la escena inicial (habitación oscura con los leds de color rojo encendidos), he seguido el siguiente procedimiento:

- Primero he creado un GameObject de tipo "Empty" al que he denominado "Luz". Este, genera un objeto vacío, sobre el cual vincular otros. Se trata de algo muy útil, ya que de esta forma podemos agrupar los objetos de la misma tipología bajo uno mismo. Y a la hora de programar, si queremos que todos los objetos de esta misma tipología sigan una misma función, solamente tendremos que añadir nuestro script como componente del GameObject de tipo "Empty". Además, una vez asociados todos los objetos a un mismo "Empty", al transformar su posición, rotación y escala, se transformarán todos los objetos emparentados con este GameObject de tipo "Empty".
- Después, he creado un GameObject de tipo "Point Light", lo he llamado "Luz1" y lo he emparentado con el GameObject "Luz". Y una vez emparentado, he modificado la posición de la luz, haciéndola coincidir con un punto intermedio entre los leds que rodean la habitación, ya que, se supone que son estos objetos los que generan esta luz y no el GameObject que estamos creando ahora mismo. De esta forma, crearemos la ilusión de que son estos objetos los que están generando esta luz rojiza.
- Por último, he modificado los parámetros de "Luz1" y he ido ejecutando el software y poniéndome las gafas de realidad virtual, hasta que he conseguido el acabado que yo quería.

He seguido el mismo procedimiento para generar las otras tres luces que aparecerán en el software. Para ver los parámetros de cada una de las luces dirigirse al subapartado "*4.2.1 Parámetros*" del anexo *"Anexo IV: Materiales, texturas e iluminación"* del documento "*Anexos*". Y para ver el resultado obtenido al utilizar estas luces en la escena y el efecto producido sobre los objetos de la misma, dirigirse al subapartado "*4.2.2 Resultado*" del anexo *"Anexo IV: Materiales, texturas e iluminación"* del documento "*Anexos*".

Nubes, estrellas y Particle System by Unity

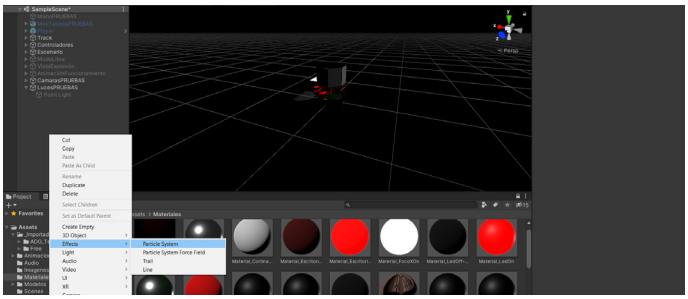
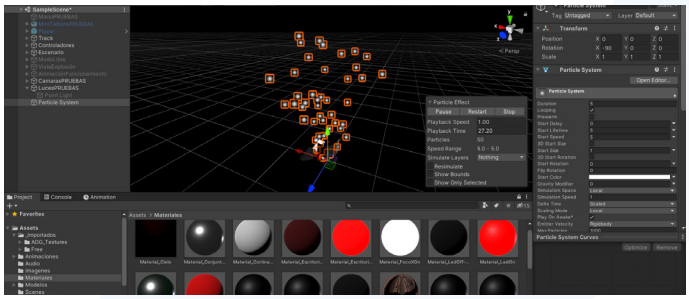
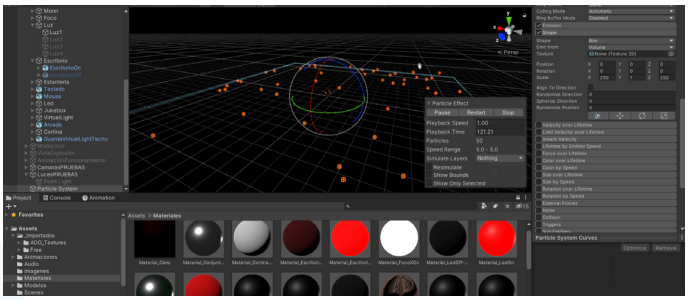
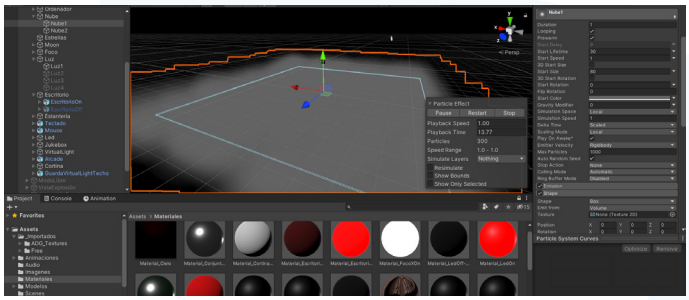
Una vez creada la escena interior, solamente falta generar una escena exterior, la cual visualizaremos desde el interior del habitáculo a través de los ventanales de la habitación. Mi intención, es la de crear un ambiente oscuro y solitario, pero a la vez bonito. Por lo que voy a crear un horizonte nocturno en el que podamos visualizar las estrellas del cielo y contemplemos el horizonte desde el nivel de las nubes. Pero... ¿cómo hacer para crear tanta cantidad de estrellas o crear infinidad de nubes espesas alrededor nuestro? La respuesta la encontramos en el sistema de partículas de Unity.

El sistema de partículas de Unity "Particle System", permite representar efectos que, normalmente, son difíciles de simular utilizando mallas o sprites *[4]*. Su modulo principal cuenta con numerosos parámetros ajustables, por lo que, utilizando este sistema, tendremos un gran abanico de posibilidades. Algunos de estos parámetros son los siguientes:

- Duración: Determina el tiempo de reproducción de cada partícula.
- Tamaño: Marca el tamaño de las partículas.
- Velocidad de inicio: Determina la velocidad con la que las partículas empezarán su movimiento.
- Play on Awake: Activado, empezará a funcionar automáticamente desde que ejecutemos el software.
- Partículas máximas: Marca el límite superior de partículas que pueden estar "vivas" de manera simultánea.

A continuación, explico el procedimiento seguido para crear "Nube1"

- Primero he creado un sistema de partículas en la jerarquía de objetos del proyecto.
- Después, he modificado el shape (su geometría) a tipo "Box".
- Por último, he ido modificando los parámetros a mi gusto hasta que he conseguido un resultado óptimo.

En caso de querer visualizar las propiedades de los otros dos sistemas de partículas creados en el proyecto, dirigirse al apartado "*4.4 Sistema de partículas (Particle System)*" del anexo *"Anexo IV: Materiales, texturas e iluminación"* del documento "*Anexos*".

[4] Los sprites son objetos 2D simples que cuentan con imágenes gráficas (llamadas texturas). Unity utiliza estos sprites por defecto cuando el motor está en "modo 2D". Cuando estos se ven en un espacio 3D, estos parecerán tener una apariencia fina, ya que no tienen ancho en el eje de coordenadas "z". Para obtener mayor información acerca de los sprites, consultar el siguiente enlace: https://docs.unity3d.com/es/2018.4/Manual/Sprites.html

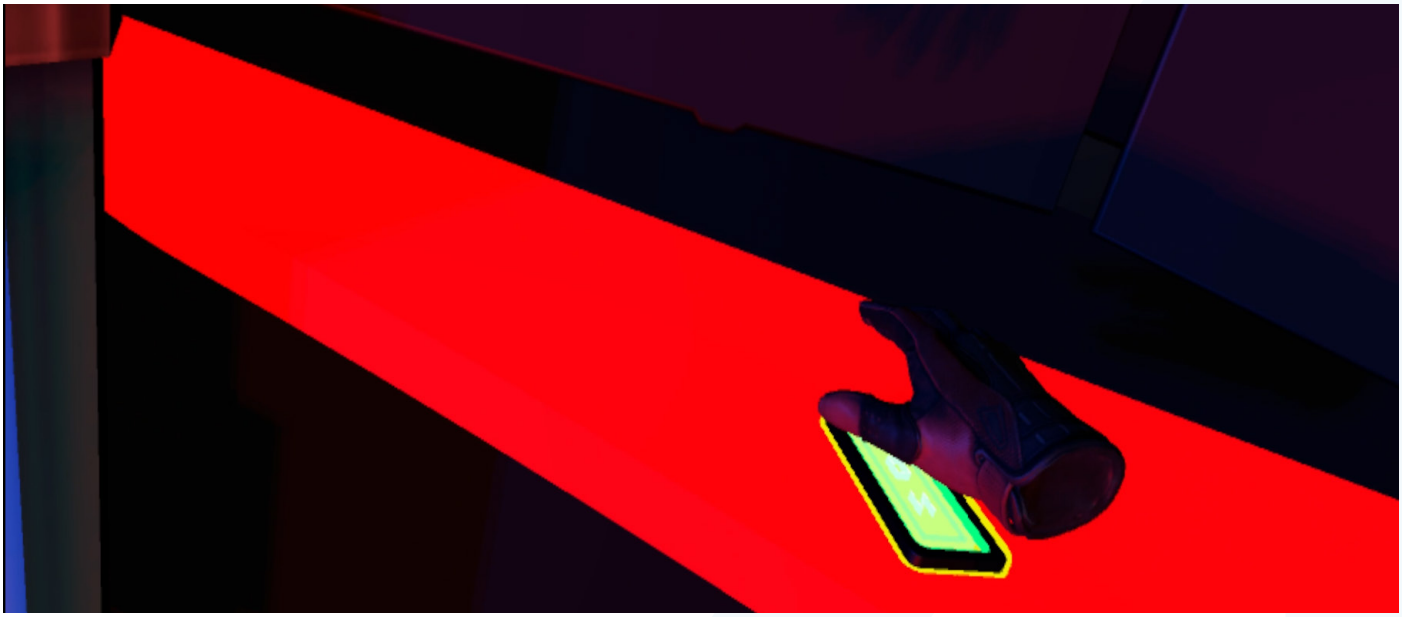
4 Programación y funcionamiento del software

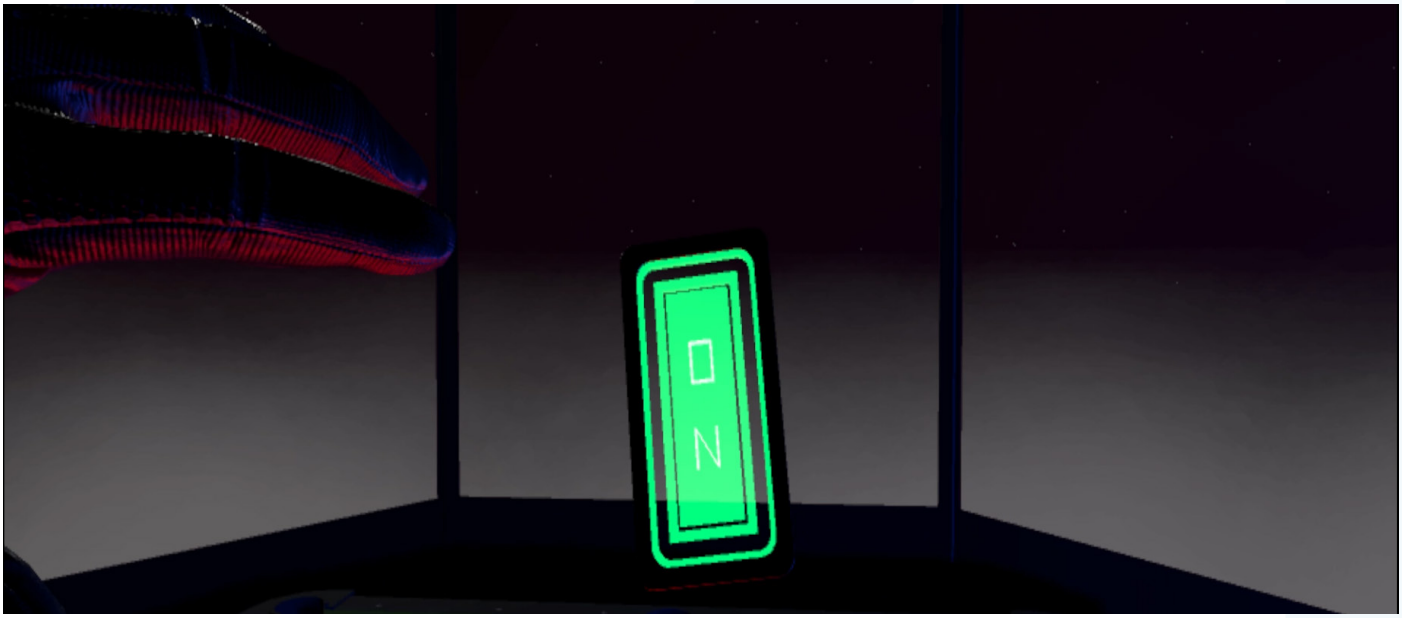
- 4.1 Interacción con el conjunto y sus componentes
- 4.2 Activación de la sala de trabajo

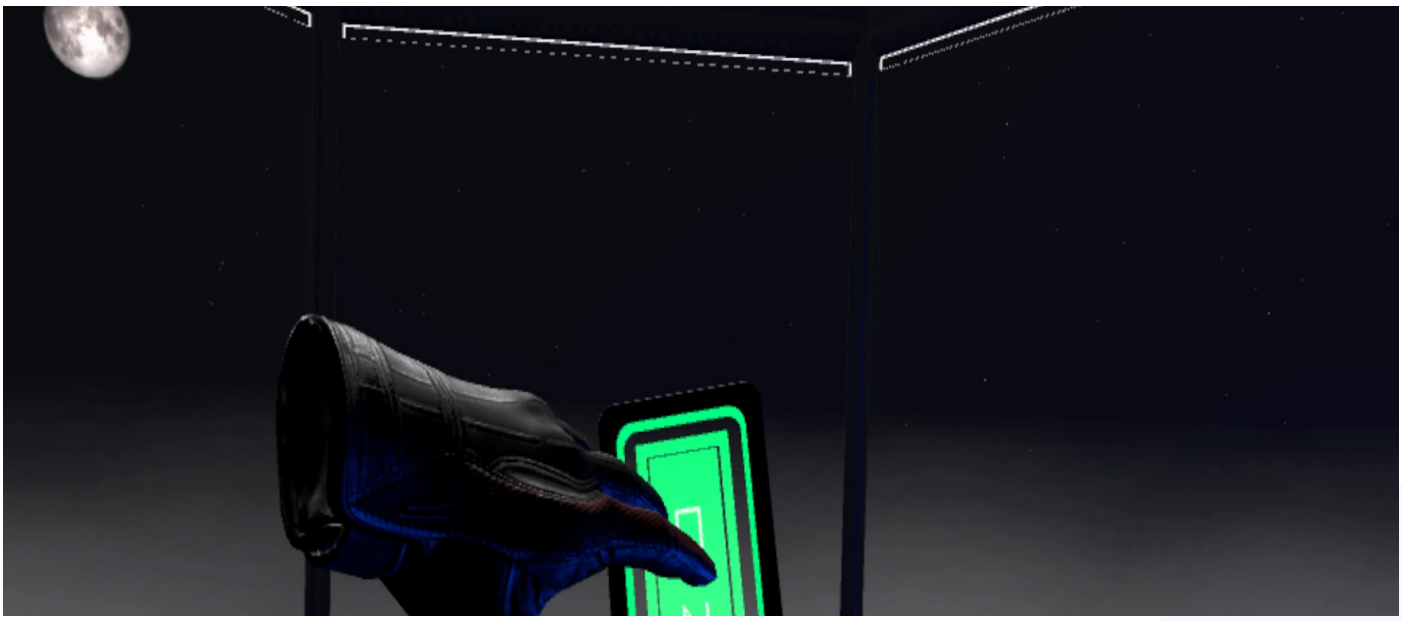
4.1 Activación de la sala de trabajo

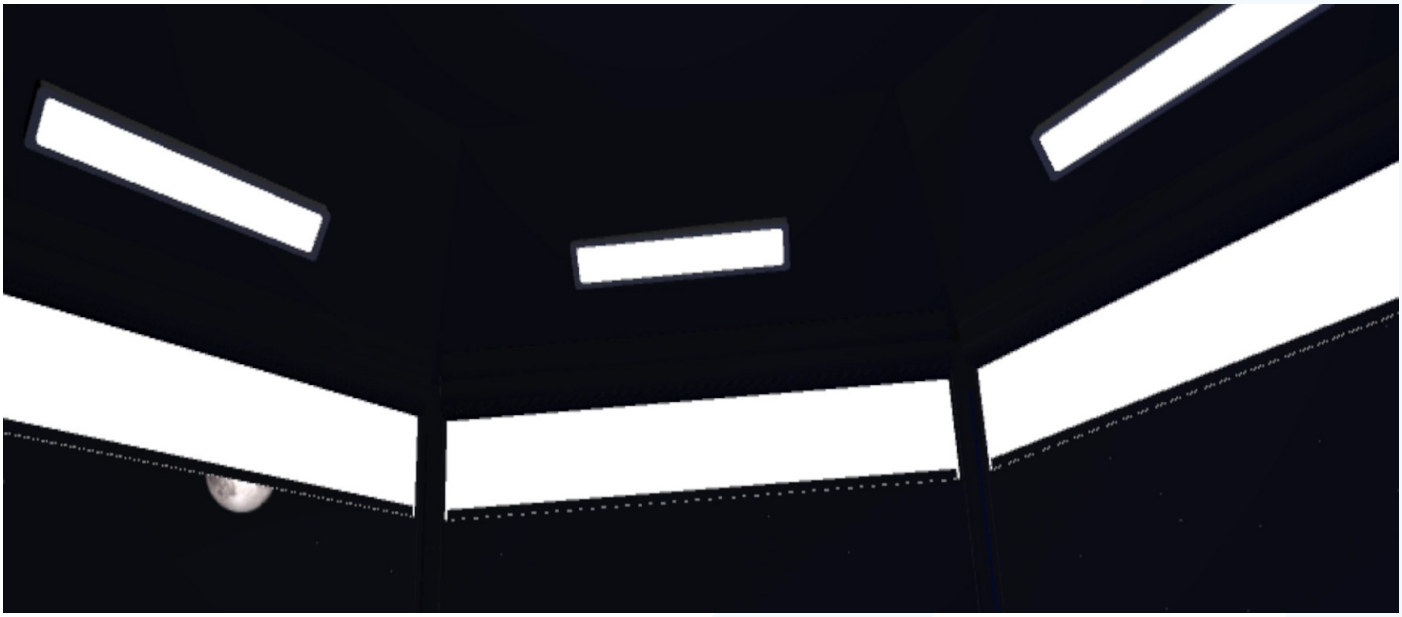
Como ya sabemos, cuando el usuario ejecute el software, este aparecerá en la escena inicial. Esta escena se trata de una habitación oscura, con luz suave rojiza y con una mesa de billar en mitad de la sala. Para activar la zona de trabajo interactiva, el usuario deberá hacer uso de la mini tableta que encontrará encendida encima de su escritorio.

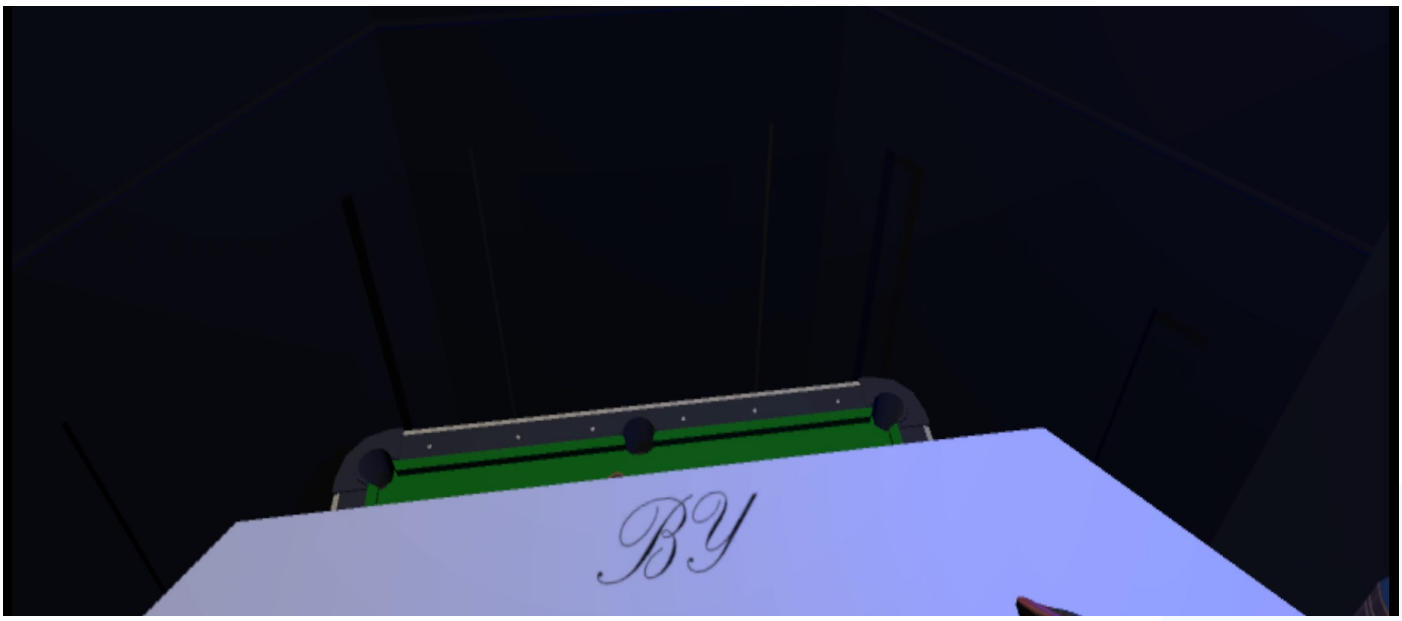
Este, cogerá el dispositivo con su mano derecha y lo accionará pulsando sobre la pantalla con su mano izquierda. Entonces, la iluminación de la habitación cambiará. Los leds, el escritorio, la estantería y las luces del ordenador se apagarán. Los focos que se encuentran en la parte superior de la habitación se encenderán. Y en este momento, el suelo empezará a bajar hacia abajo por el tunel que se encuentra en la parte inferior de la habitación, recogiendo de esta forma la mesa de billar (como si de un ascensor de automóviles se tratase). Mientras esto sucede, las cortinas bajarán de forma progresiva y un nuevo techo aparecerá poco a poco desde el interior del guardatechos. Y por último, cuando el suelo haya bajado hasta una distancia considerable y la mesa esté resguardada, un nuevo suelo saldrá lentamente delante de él.

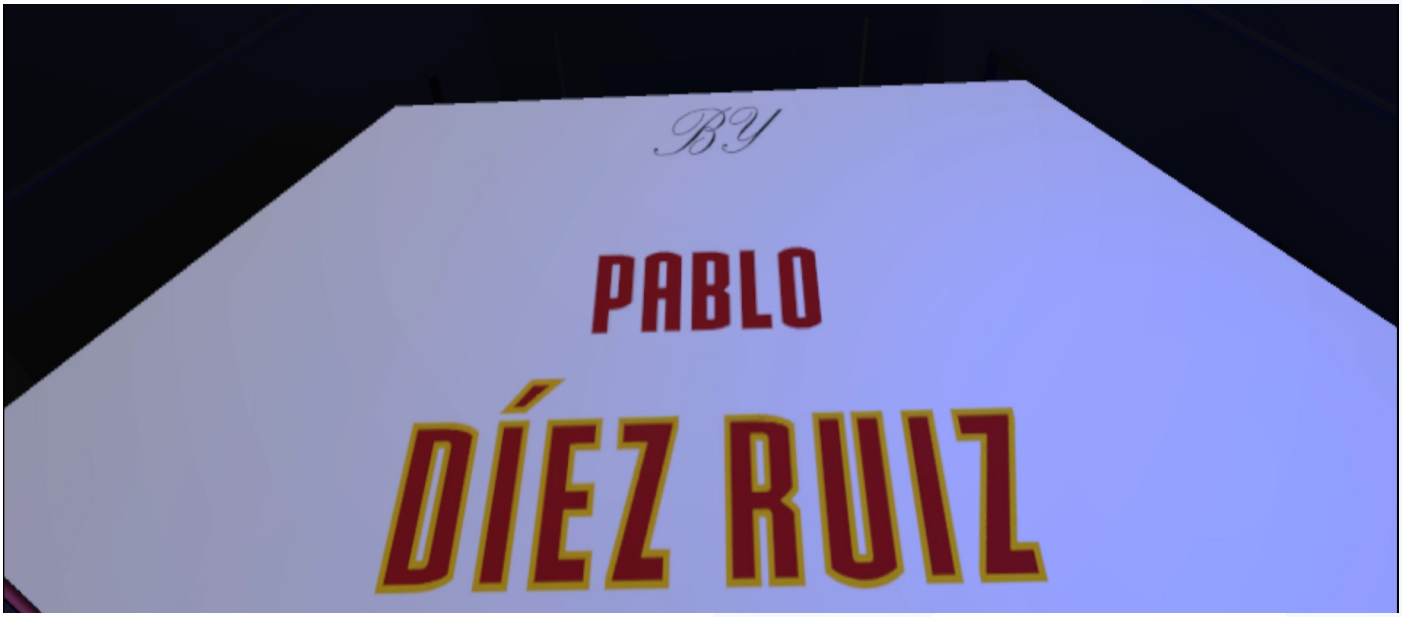








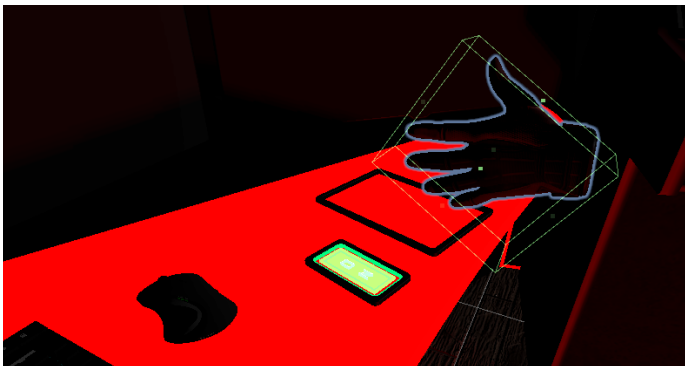

BY

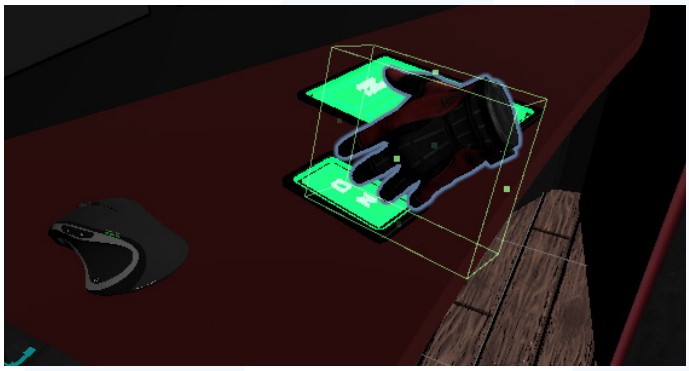

BY
PABLO
DÍEZ RUIZ


MIDWAY

Para lograrlo, he seguido los siguientes pasos:

- Lo primero de todo, es añadir los colisionadores a las manos. Sobre los GameObjects "LeftHand" y "RightHand", he añadido un colisionador de tipo "Box Collider". Este, lo he ajustado de la mejor forma posible y he activado la casilla de "Is Trigger" de ambos colliders.





- Después, sobre el GameObject "MiniTableta3", he añadido un colisionador de tipo "Box Collider" y he activado la casilla de "Is Trigger".
- Y por último, he creado los siguientes scripts: "MiniTableta3_1.cs" , "CortinaPanel_1.cs" , "Suelo2_1.cs" , "VirtualLightOffSuelo_1" y "VirtualLightTecho_1".

El colisionador de la mano izquierda, cuando entra en contacto con el colisionador de la pantalla de la mini tableta, activa la función "OnTriggerEnter" del script "MiniTableta3_1". Y este mismo script, hace que se activen y se desactiven los GameObjects que yo quiero y además, llama a las funciones de los demás scripts.

En estos scripts, se encuentran programadas las funciones necesarias para que la posición de los GameObjects que van a incorporarse a la escena, vaya modificándose progresivamente hasta llegar a una posición determinada. En este momento (cuando llegamos a una posición "x"), las funciones cambian el valor de su respectivo booleano y la posición deja de actualizarse.

En el apartado "*2.2 Proyecto final*" del anexo "*Anexo II: Programación y scriptado*" del documento "*Anexos*", podemos encontrar el código de los scripts: "MiniTableta3_1.cs" , "CortinaPanel_1.cs" , "Suelo2_1.cs" , "VirtualLightOffSuelo_1" y "VirtualLightTecho_1".

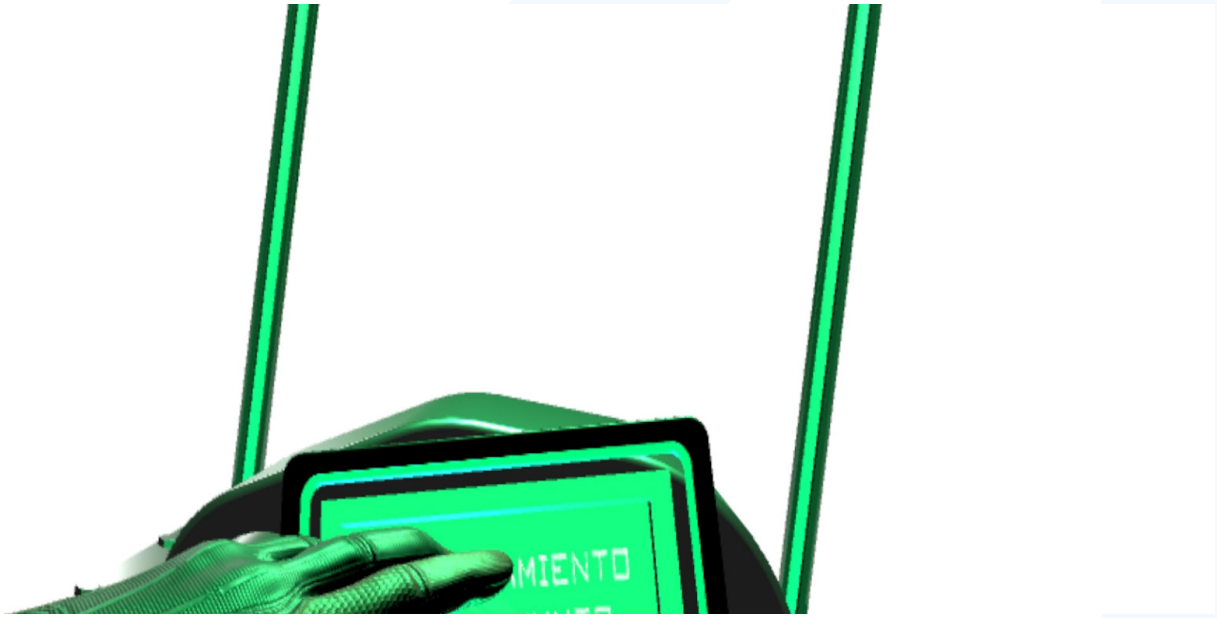
4.2 Interacción con el conjunto y sus componentes

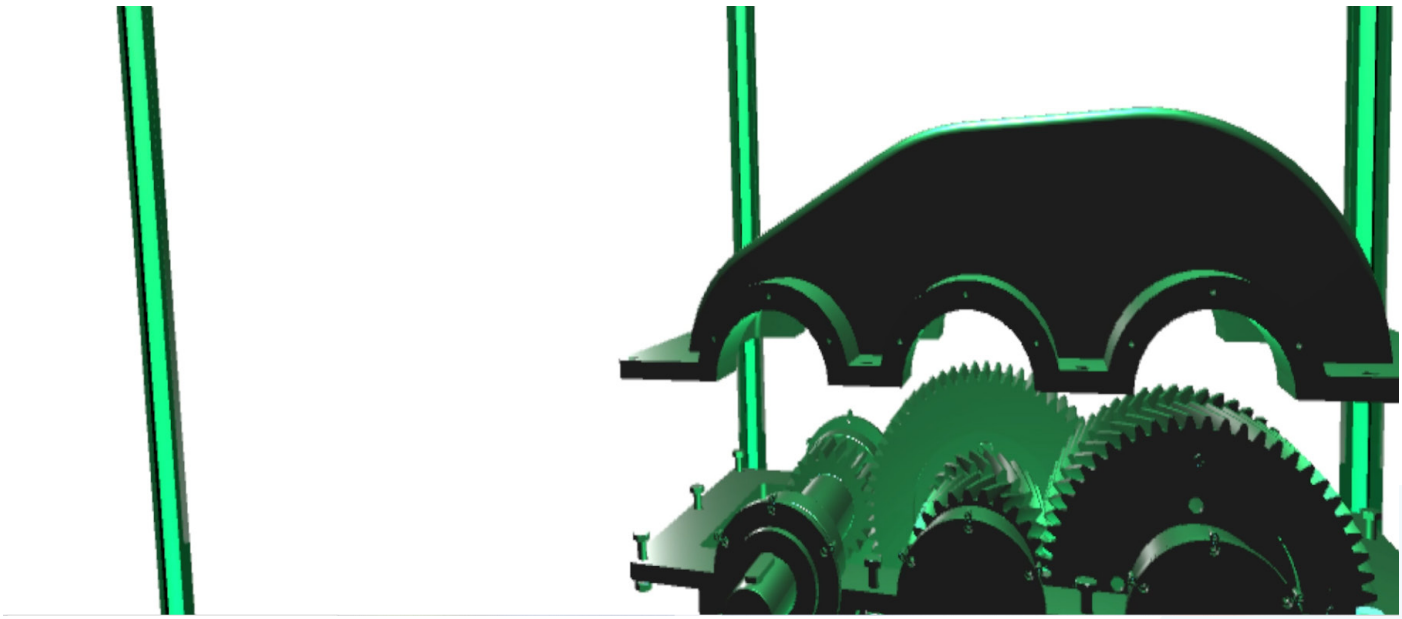
Una vez tenemos nuestra zona de trabajo activada, podemos empezar a trabajar con los modelos. Para cargar el conjunto, el usuario deberá utilizar una tableta que encontrará encima del escritorio. Este dispositivo, nos resultará familiar, ya que antes de activar la sala, éste se encontraba apagadao Ahora, la tableta esta operativa y nos ofrece tres opciones diferentes: "Funcionamiento del conjunto", "Desmontaje y explosionado" y "Manipulación de componentes".

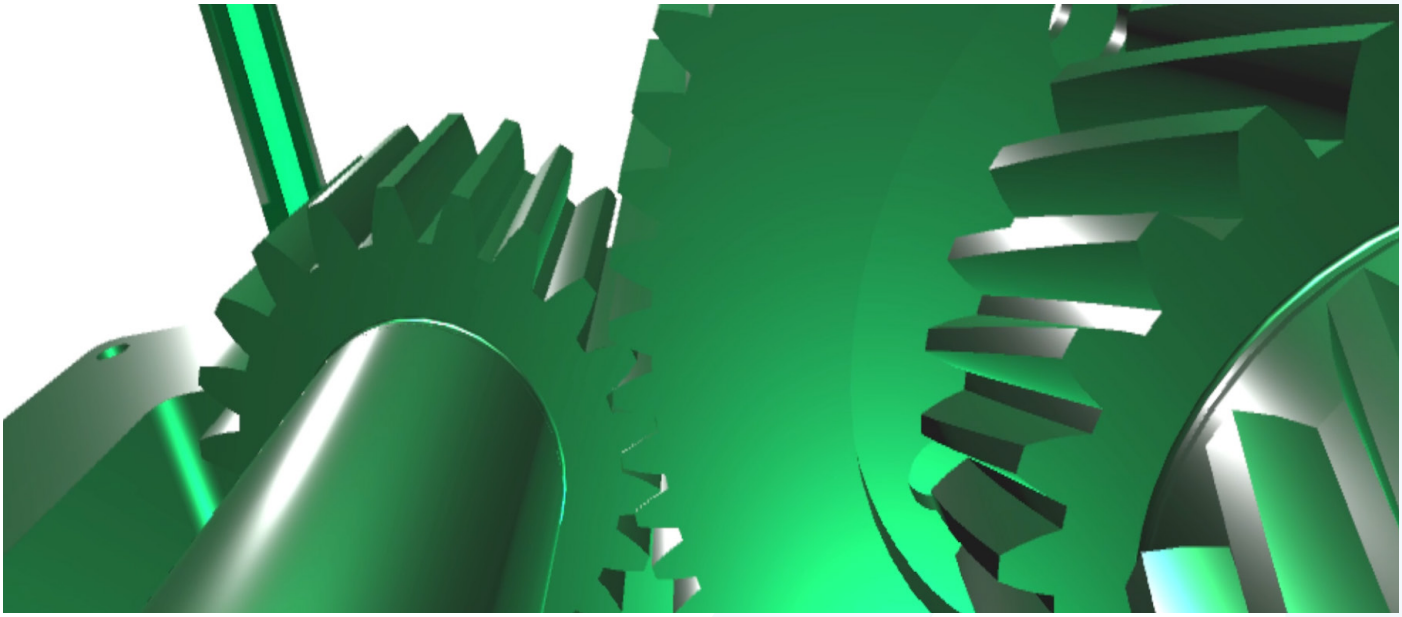
El usuario escogerá entre una de las opciones y aparecerá el conjunto delante suyo. Dependiendo del modo que haya escogido, podrá realizar una actividad u otra. A continuación describiré las funciones que podrá realizar el usuario en cada modo y explicaré cómo he conseguido programarlo.

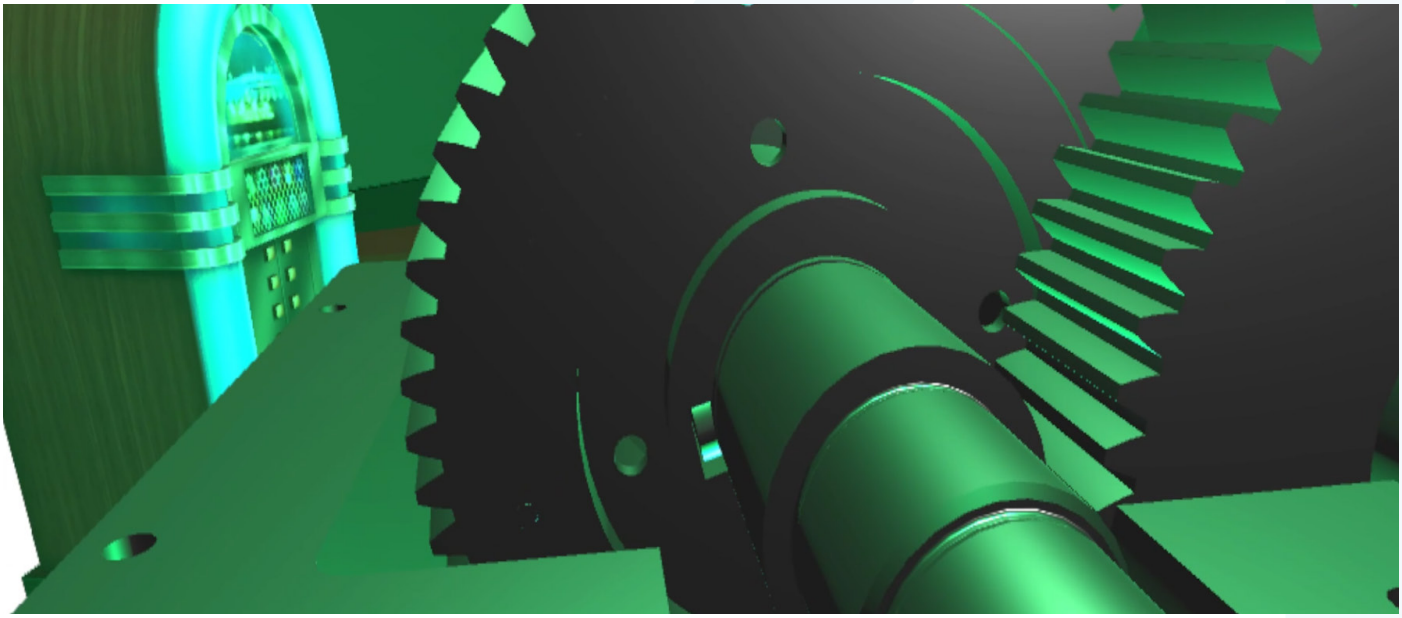
4.2.1 Funcionamiento del conjunto

En este modo, el usuario podrá visualizar el conjunto mientras este funciona. En esta animación se podrá observar como algunos componentes se eliminan de forma automática antes de que los ejes empiecen a girar. Esto lo he hecho, ya que he pensado que, de esta forma, se apreciará mucho mejor el movimiento de los ejes y la transmisión del par motor entre los mismos. El usuario observará cómo funcionan los engranajes y engranan entre sí y podrá ver cómo se transmite el giro a través de las chavetas.

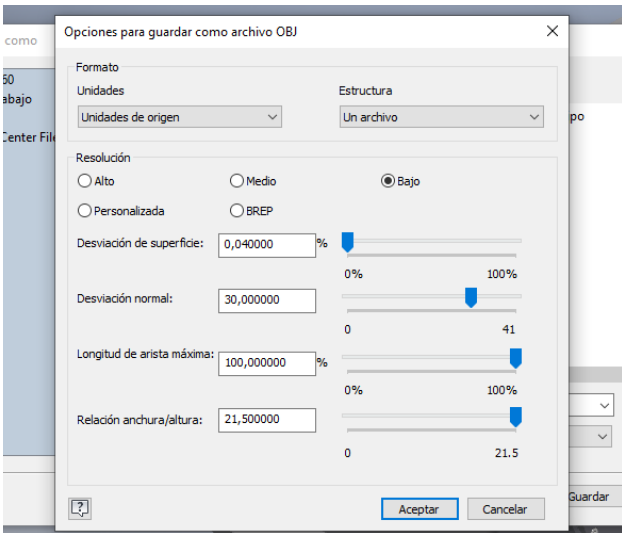








Para lograrlo, he seguido los siguientes pasos:



- Lo primero que he hecho es exportar el modelo del conjunto desde Inventor Professional 2018 en archivo .obj. Antes de exportarlo, he seleccionado una calidad baja, ya que al exportarlo en calidad alta o media, el número total de triángulos que generaban las mallas de todos los componentes, era muy alto. De todas formas, he probado a introducir ambos modelos en Unity y ejecutar el software visualizándolo con las gafas de realidad virtual. Pero en ambos casos, se colapsaba un poco la renderización y le restaba fluidez al software. Y por lo tanto, he decidido exportarlo en la calidad citada anteriormente. Además, aunque lo haya exportado con una calidad baja, el conjunto se ve con un nivel de detalle muy alto, por lo que no me ha supuesto ningún problema.

- Después, he importado el archivo en Unity. Lo he reescalado y he modificado su posición, situándolo en el centro de la zona de trabajo. Acto seguido, le he añadido a los componentes el material "Material_Conjunto(1,2,3)-MesaDeTrabajo(1,2,3)" y he emparentado al conjunto con el empty "AnimacionFuncionamiento".

Pero al empezar a programar la animación, he encontrado un problema. Al exportar el ensamblaje desde Inventor e importarlo en Unity, este último si que reconoce e identifica cada uno de los componentes por separado. Pero al exportarlo en archivo .obj desde el ensamblaje .iam, aunque luego en Unity veamos diferentes componentes asociados al GameObject que hace de ensamblaje, estos, tienen el mismo eje de coordenadas. Es decir, cada modelo al exportarse, se desvincula del resto, pero adquiere los ejes de coordenadas globales (del ensamblaje) como locales (de la pieza). Y por lo tanto, esto supone un gran problema a la hora de realizar las animaciones. Ya que, por ejemplo, si queremos hacer rotar un tornillo para simular que se está desatornillando de la carcasa, no podemos hacer que este gire en torno a sí mismo. Porque su eje "z" no se encuentra en el centro atravesando la pieza de arriba a abajo, sino que se encuentra en otra posición (en la posición en la que se encontraban los ejes de coordenadas globales del ensamblaje).

Para solucionar este problema, probé diferentes maneras de solucionarlo en el mismo programa de Unity y también traté de hacer las animaciones en Blender. Pero en Unity no pude solucionarlo y Blender me daba errores del mismo tipo (sobre las rotaciones en torno a los ejes). Entonces... no me quedo otra solución que tomar el camino difícil.

Lo que hice fue exportar todas las piezas y componentes de Inventor por separado. Y de esta forma, al hacerlo, mantendrían sus ejes de coordenadas locales. Y después, partiendo de un conjunto de referencia exportado desde el archivo "Ensamblaje", he ido ensamblando poco a poco todas las piezas y los componentes que había importado de manera individual dentro de la escena en Unity.

- Una vez ensambladas todas las piezas y componentes en su sitio, he generado la animación siguiendo el método estudiado en el subapartado "*2.3.5 Pruebas III: Animaciones*" de este mismo documento. Pero al haber ensamblado "a ojo" todos los componentes del conjunto, es imposible que estos hayan quedado perfectamente ajustados en sus posiciones. Esto, podría ser un problema, ya que todos los elementos pertenecientes a un mismo eje, deben girar en torno al mismo eje de coordenadas. Y para asegurarme de que esto sí que sucede, he creado tres GameObject de tipo "Empty": "EjeRápido", "EjeLento" y "EjeIntermedio". Todas las piezas y los componentes de un mismo eje, las he introducido en cada uno de los GameObjects de tipo "Empty" al que perteneciesen.

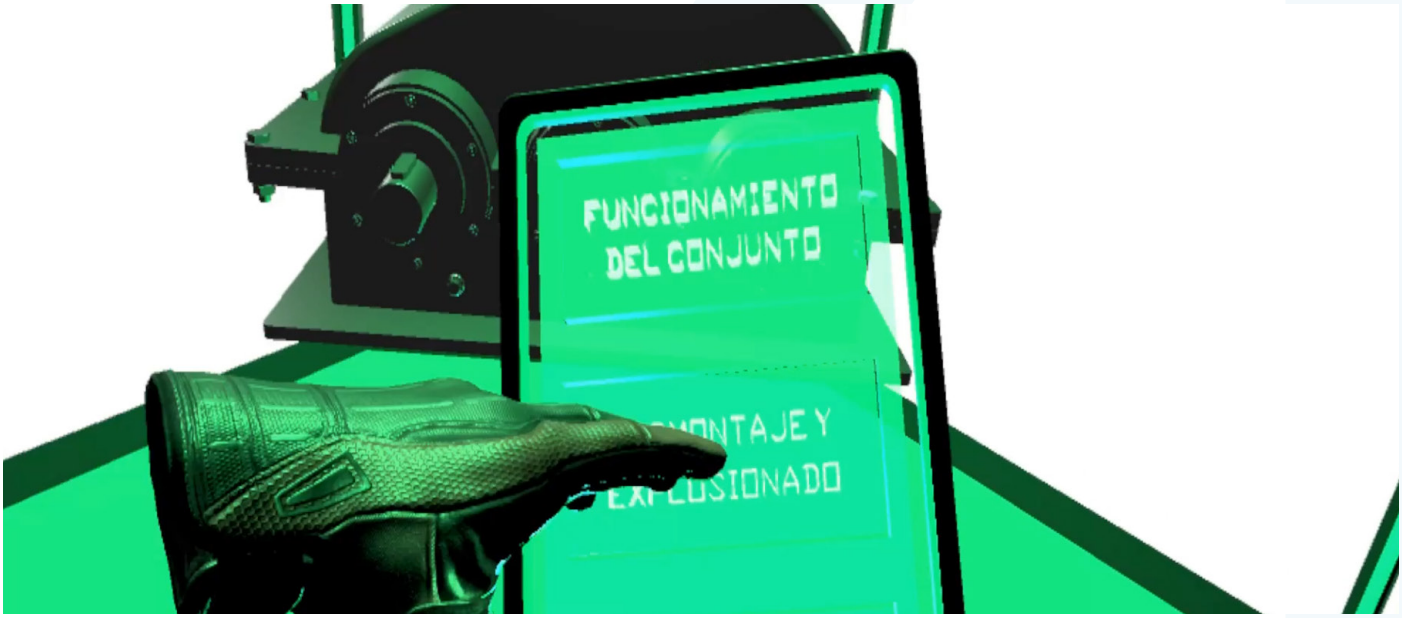
A la hora de realizar la animación simulando el funcionamiento y el movimiento de los ejes, he girado directamente el empty. Y gracias a esto, conseguimos que todos los componentes pertenecientes a un mismo eje, giren en torno al mismo eje.

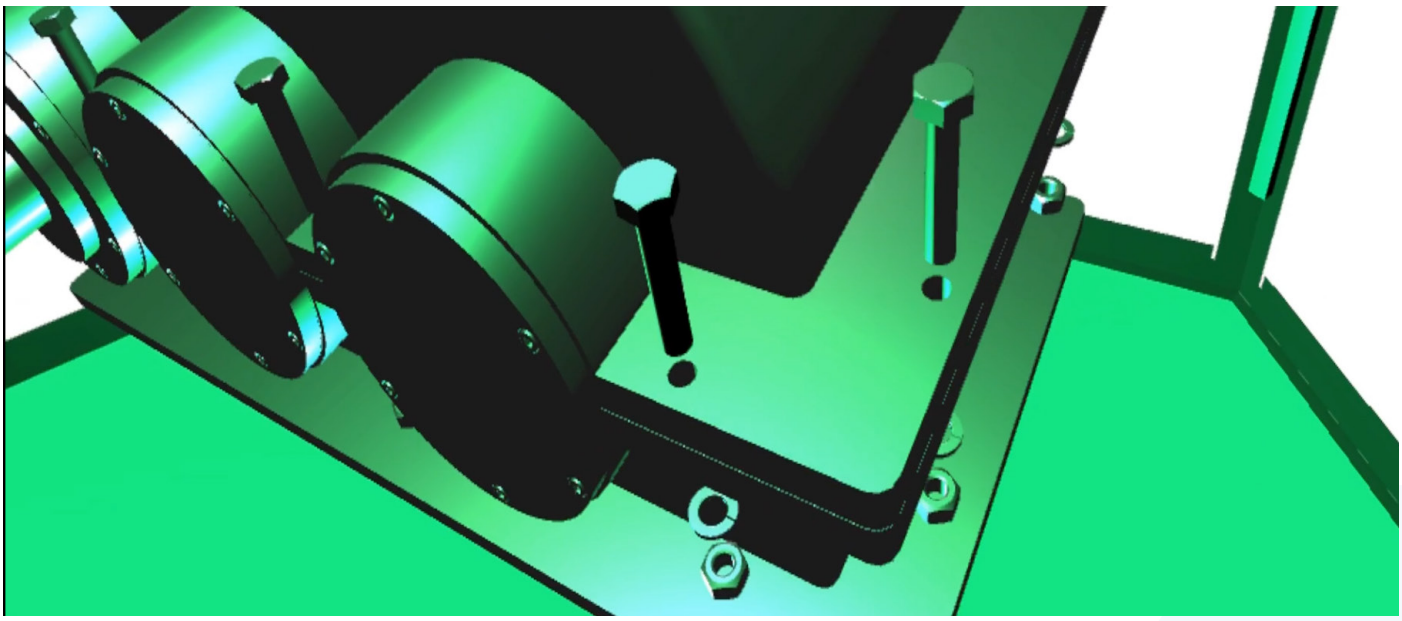
- Y por último, para hacer que se active este modo, he creado el script "AnimacionFuncionamiento_1". Este lo he vinculado con el GameObject "AnimacionFuncionamiento", al que le he añadido un colisionador de tipo "Box Collider", adaptando su geometría a la del botón del mismo. Este GameObject forma parte de "Tableta3" (pantalla de la tableta) y a su vez de "Tableta".

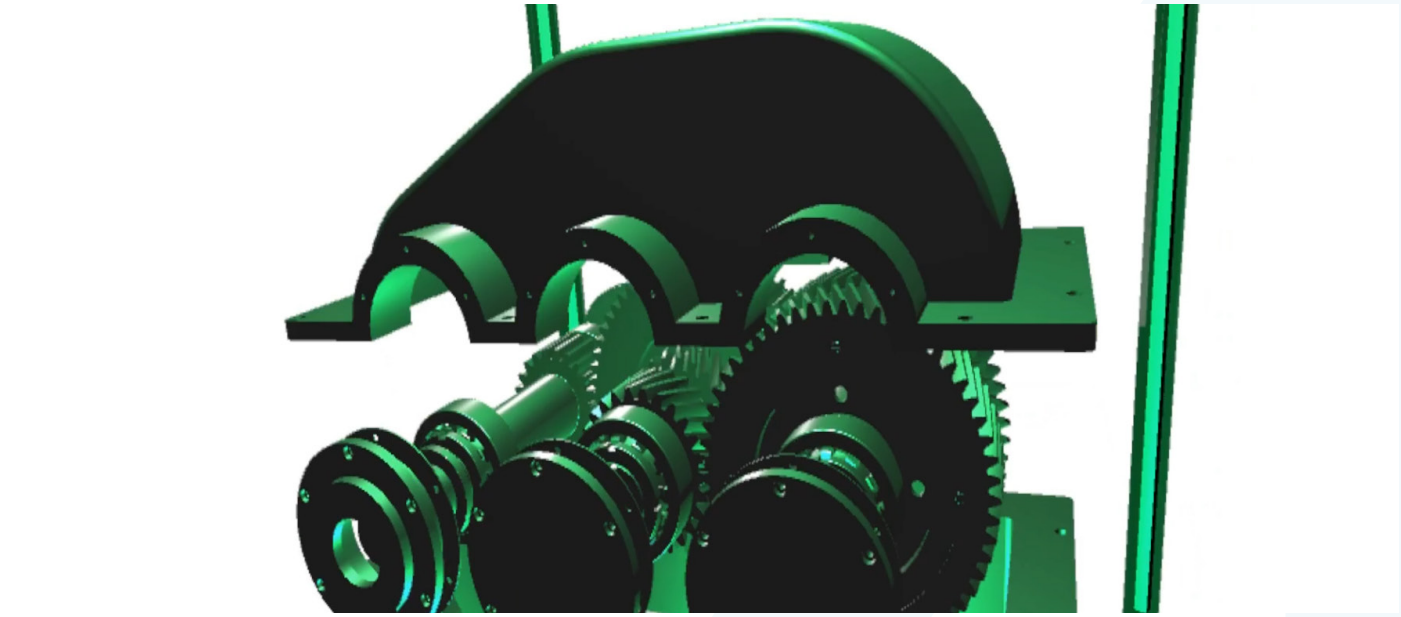
En el apartado "*2.2 Proyecto final*" del anexo "*Anexo II: Programación y scriptado*" del documento "*Anexos*", podemos encontrar el código del script "AnimacionFuncionamiento_1". Para encontrar el archivo de la animación, seguir la ruta "*Assets\Animaciones\Conjunto3*" dentro de la carpeta del proyecto de Unity.

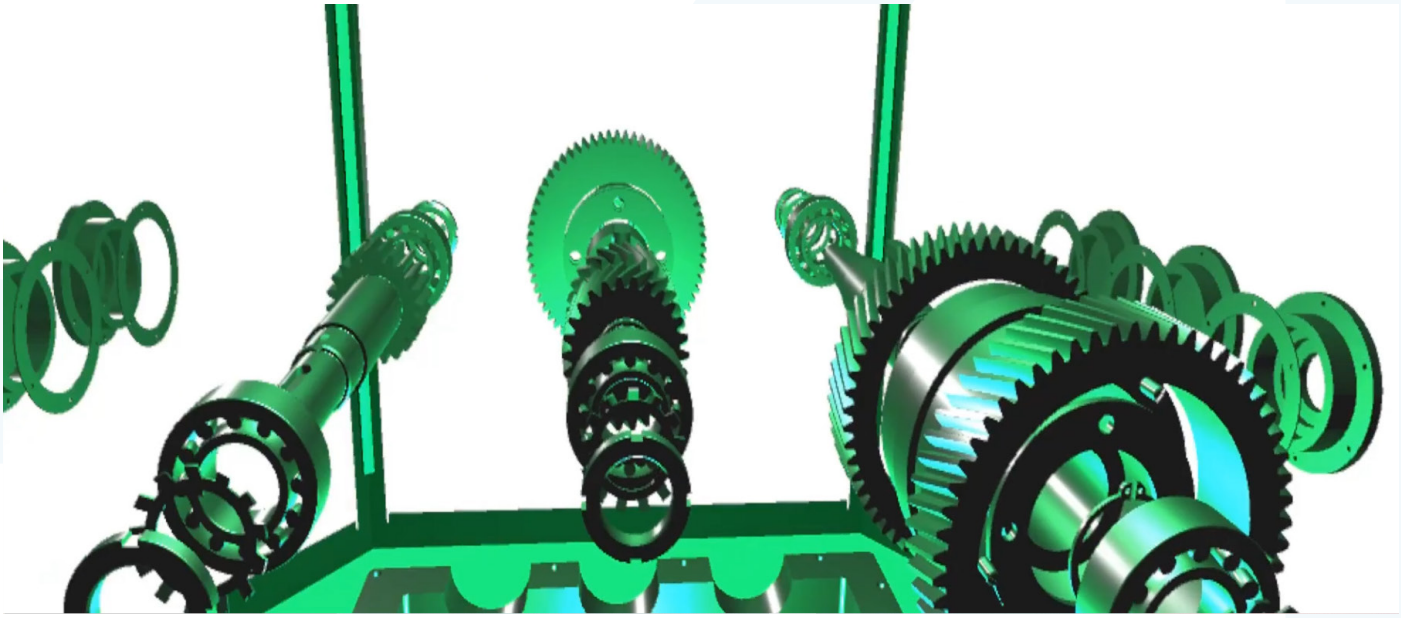
4.2.2 Desmontaje y explosionado

En este modo, el usuario podrá visualizar el conjunto mientras este se explosiona. El conjunto se inicializará ensamblado totalmente y de forma secuencial se irá desmontando delante de nuestros ojos. Una vez retirados los tornillos, arandelas, tuercas, carcasas y tapetas, la animación primero nos mostrará un despliegue de los ejes interiores, y después, todos los componentes se irán desmontando siguiendo el orden adecuado. Al finalizar la animación, el conjunto quedará explosionado durante unos segundos, ofreciéndole la posiblidad al usuario de visualizar más detenidamente la vista explosionada.



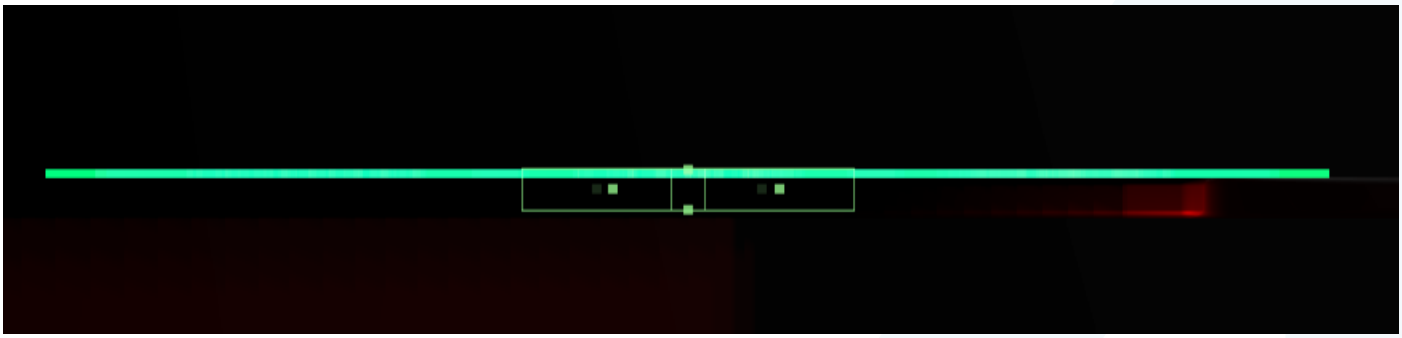






Para lograrlo, he seguido el siguiente procedimiento:

- Primero he duplicado el GameObject del conjunto anterior. De esta forma, nos ahorraremos tener que repetir todo el proceso de ensamblado, posicionamiento, escalado, selección de material de los componentes etc... Y una vez duplicado, le he cambiado su nombre y lo he emparentado con el empty "VistaExplosion".
- Después, he repetido el mismo procedimiento que en el caso anterior. Es decir, he ido creando la animación desde el inicio hasta el final, siguiendo un orden coherente e intentando que el resultado fuera lo más atractivo posible para el usuario. Al tratarse de realidad virtual, dependiendo de como hagamos estas animaciones, podemos incidir de forma muy directa en la experiencia y en las sensaciones del usuario (por ejemplo, buscar movimientos sorprendentes, combinar diferentes tipos de velocidades, realizar la animación de la forma más fluida posible etc...)
- Y por último, para hacer que se active este modo y se desactive el anterior (se desactivarán los dos restantes y en cada modo pasará lo mismo), he creado el script "VistaExplosion_1". Este lo he vinculado con el GameObject "VistaExplosion", al que le he añadido un colisionador de tipo "Box Collider", adaptando su geometría a la de su botón de la tableta. Como en el modo anterior, este GameObject forma parte de "Tableta3" y a su vez de "Tableta".



En el apartado "*2.2 Proyecto final*" del anexo "*Anexo II: Programación y scriptado*" del documento "*Anexos*", podemos encontrar el código del script "VistaExplosion_1". Para encontrar el archivo de la animación, seguir la ruta "*Assets\Animaciones\Conjunto2*" dentro de la carpeta del proyecto de Unity.

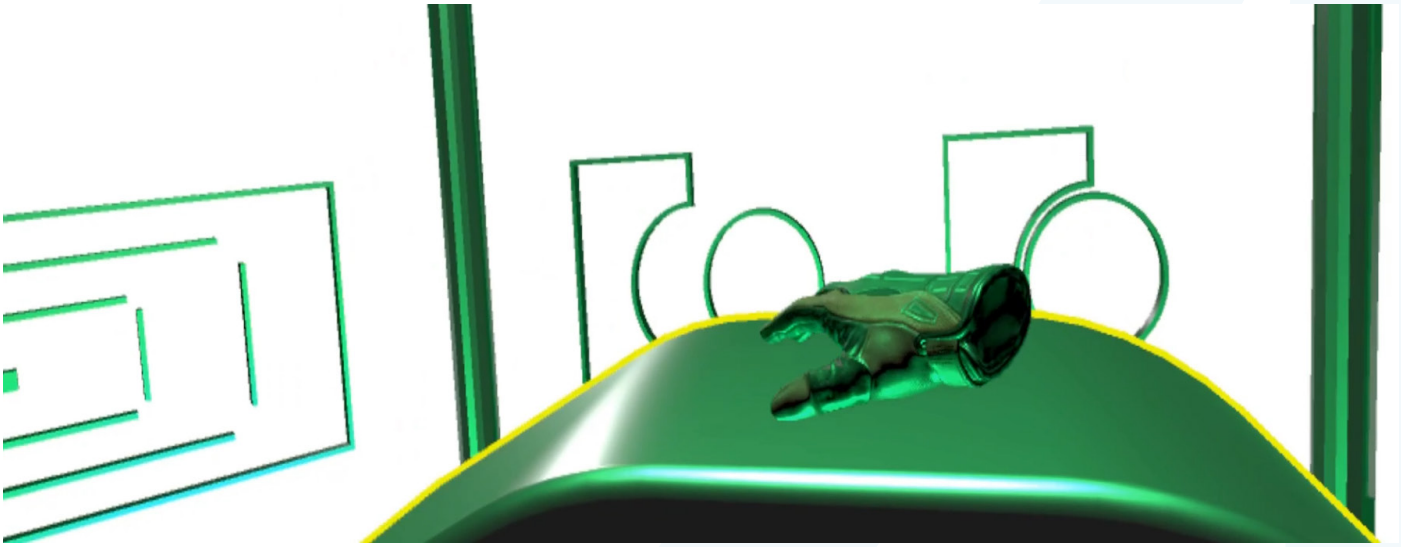
4.2.3 Manipulación de componentes

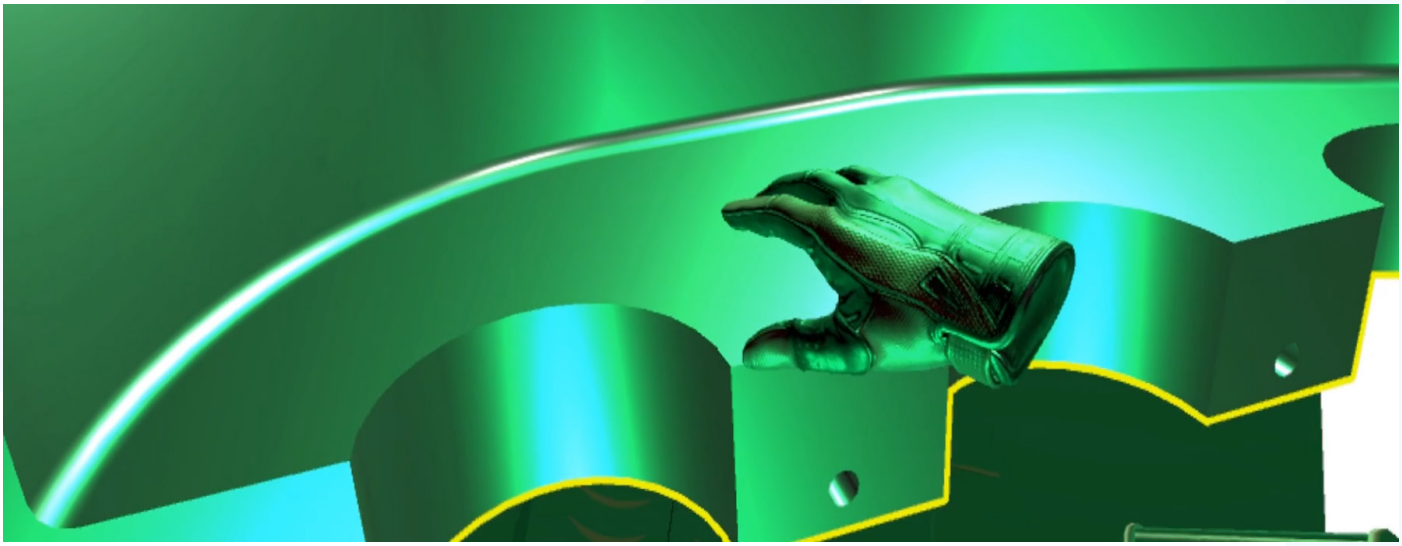
En este modo, el usuario podrá realizar las diferentes actividades/operaciones:

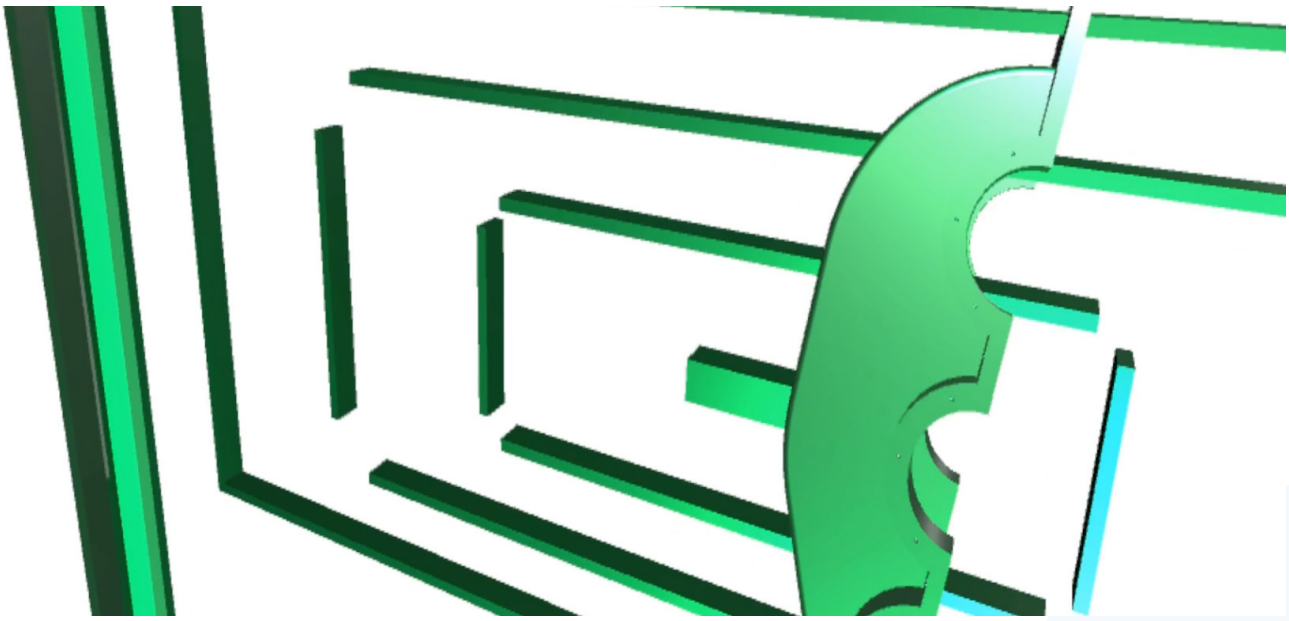
- Con sus manos, podrá seleccionar, agarrar y mover todos los componentes pertenecientes al conjunto. Gracias a ello, tendrá la posibilidad de desmontar el conjunto a su gusto, podrá visualizar las piezas de una forma más atractiva y podrá introducirse en este visualizando con detalle todos los componentes y la forma en la que el conjunto ha sido ensamblado.

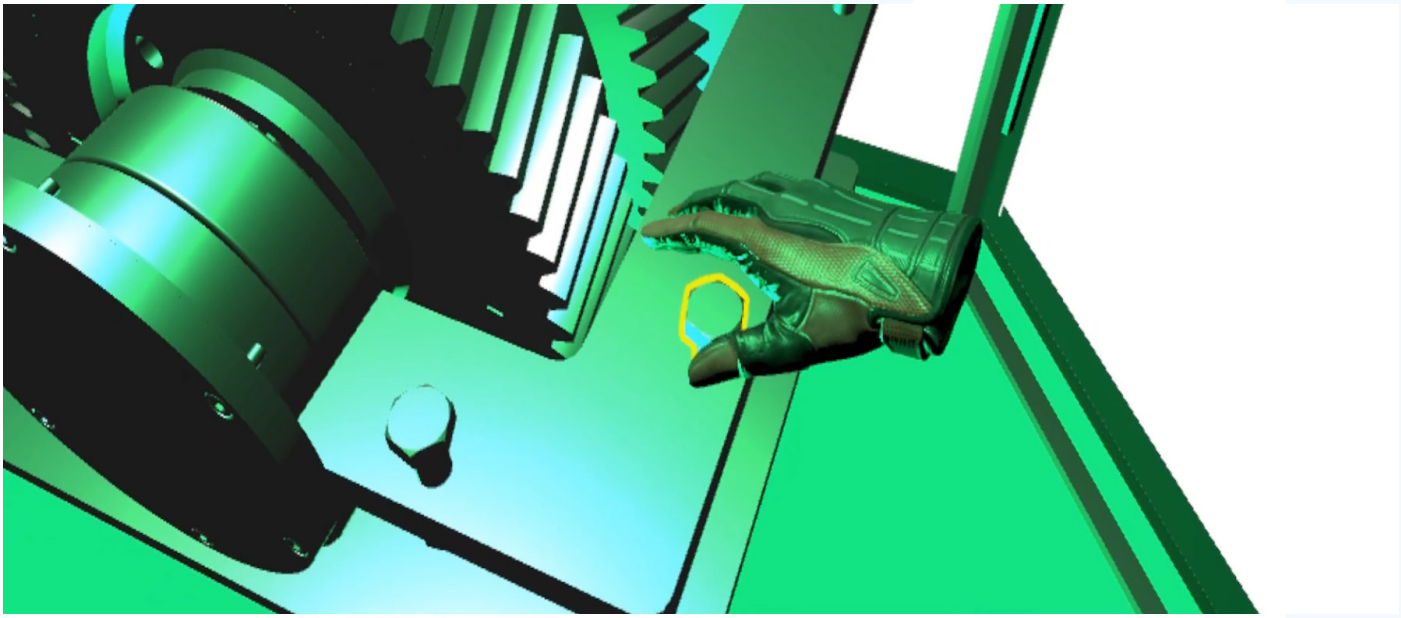
- También podrá realizar operaciones de reescalado de los componentes. Una vez seleccionado y agarrado, el usuario introducirá un componente determinado en unas zonas habilitadas. Y entonces, este sufrirá una transformación en su escala, aumentando o disminuyendo de tamaño. Esta operación le será muy útil al usuario, ya que, de esta forma, éste podrá aumentar la geometría de los componentes pequeños (como los tornillos y las arandelas), los cuales a simple vista, no podemos apreciar con facilidad todos los detalles de sus formas. Y por el contrario, para los componentes muy grandes (como las carcasas), tendrá la posibilidad de disminuir su geometría, haciendo que esta no ocupe un gran espacio dentro de la zona de trabajo.

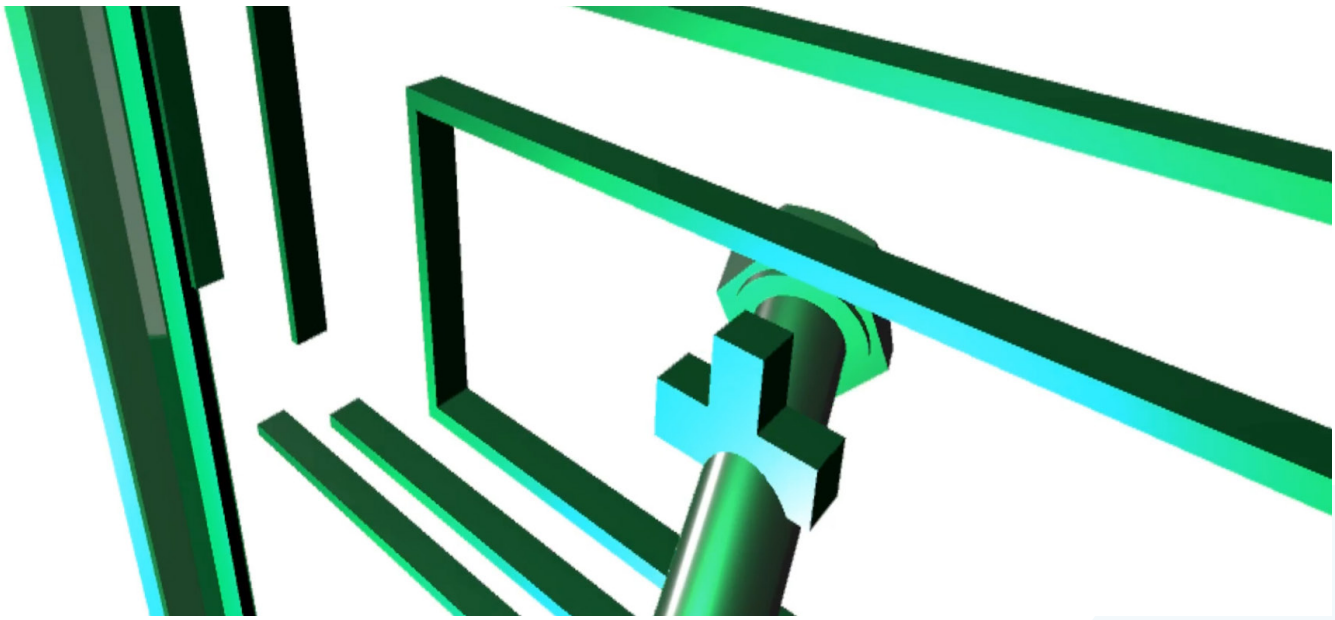
- Y por último, después de haber interactuado con alguno o varios de los componentes del conjunto y después de haber realizado transformaciones en su tamaño (aumentado o disminución de tamaño), el usuario tendrá la posibilidad de ensamblar el/los componente/s en un sitio. Habrá una tercera zona en la que el usuario, podrá arrastrar los componentes hasta ella, y de esta forma, el componente se ensamblará de forma automática en su respectivo lugar dentro del conjunto.

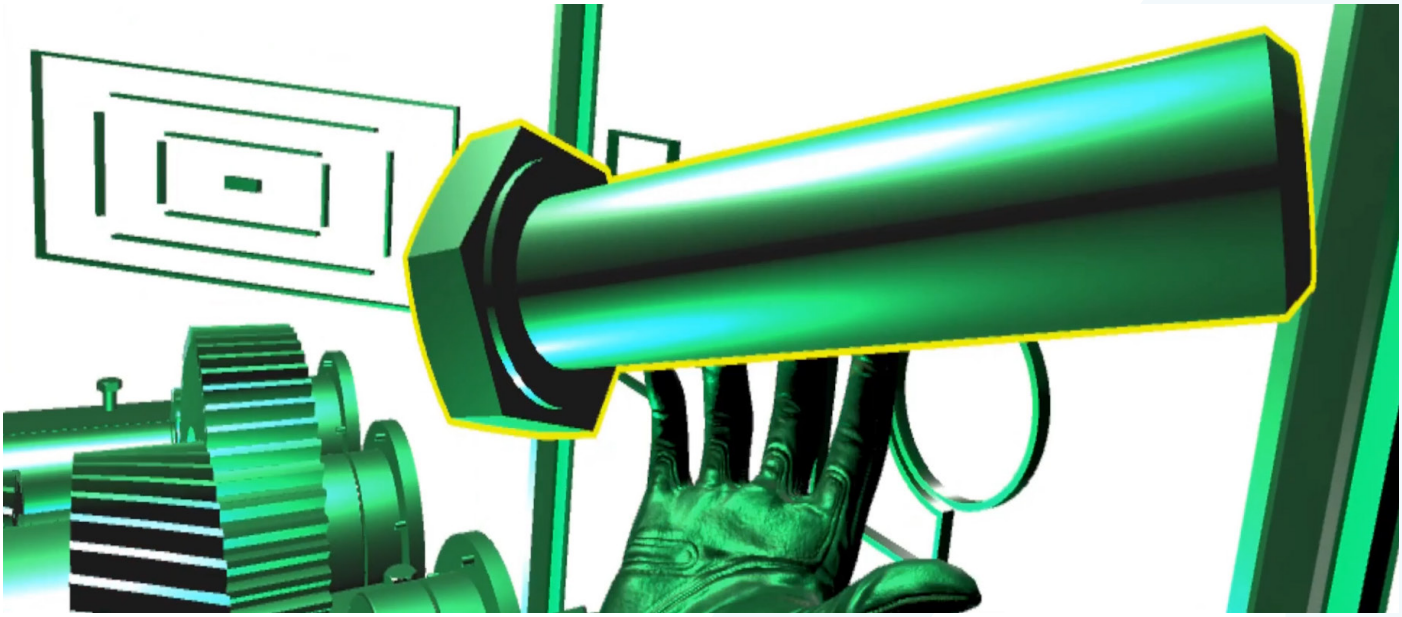


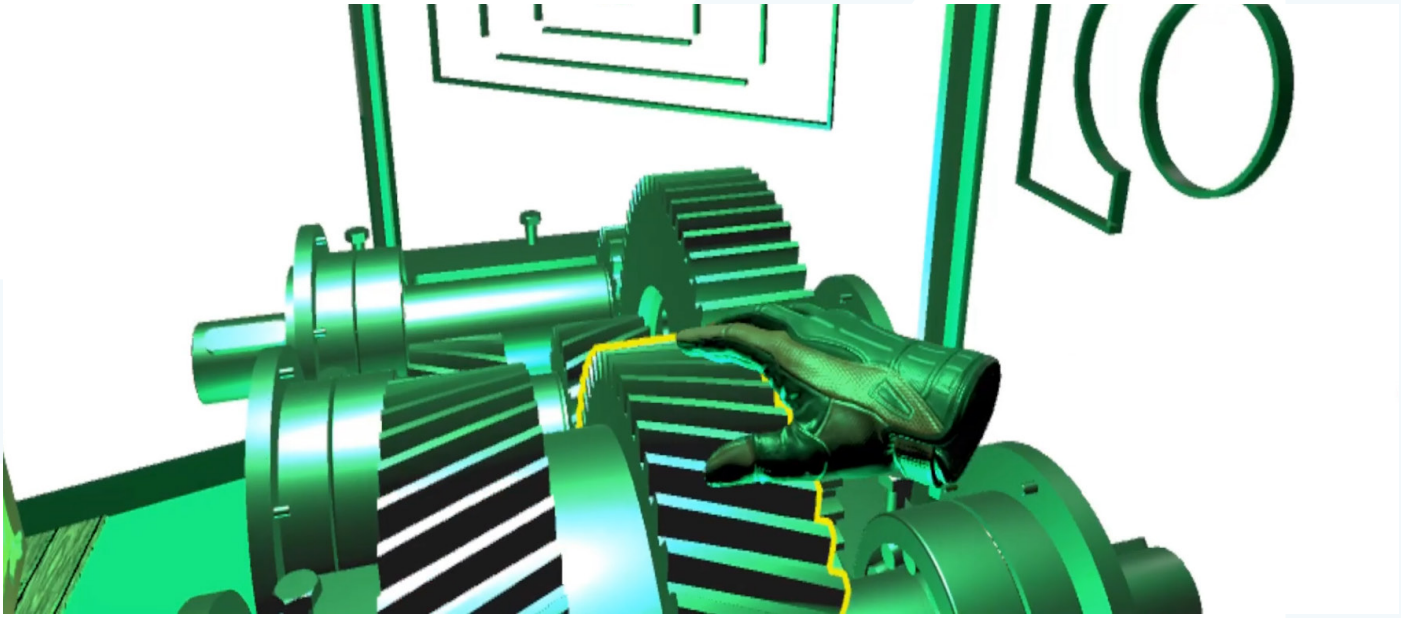


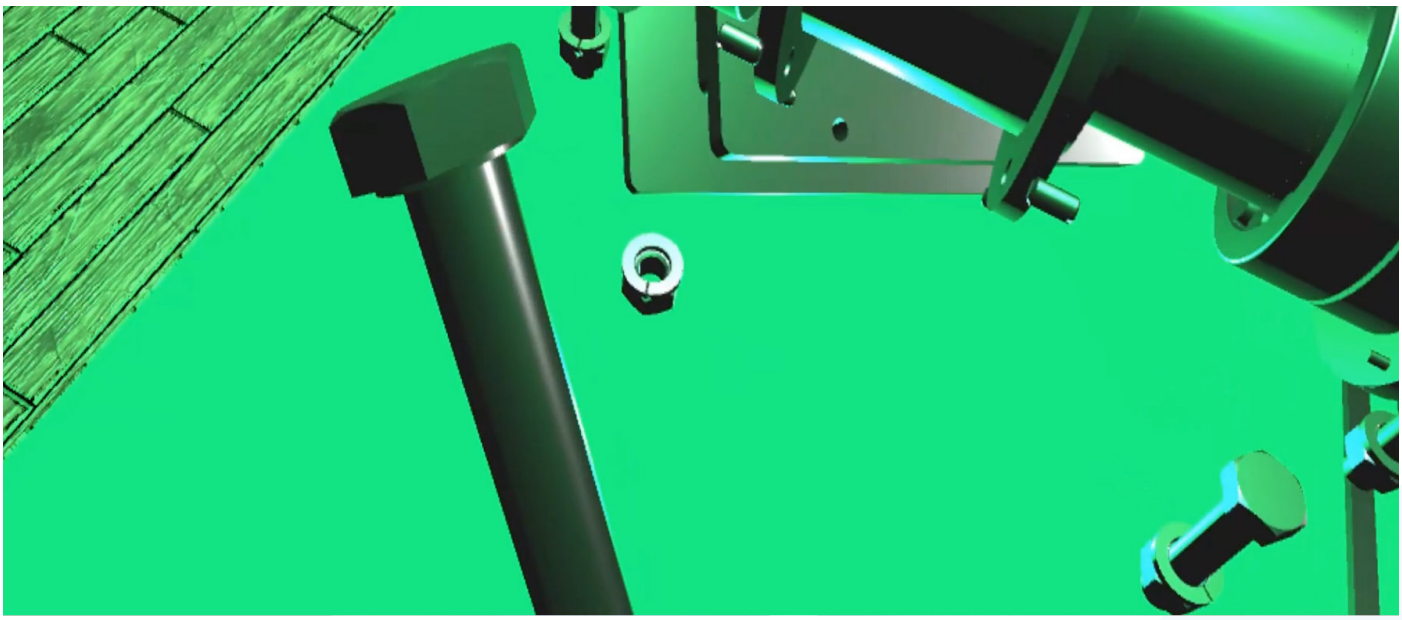


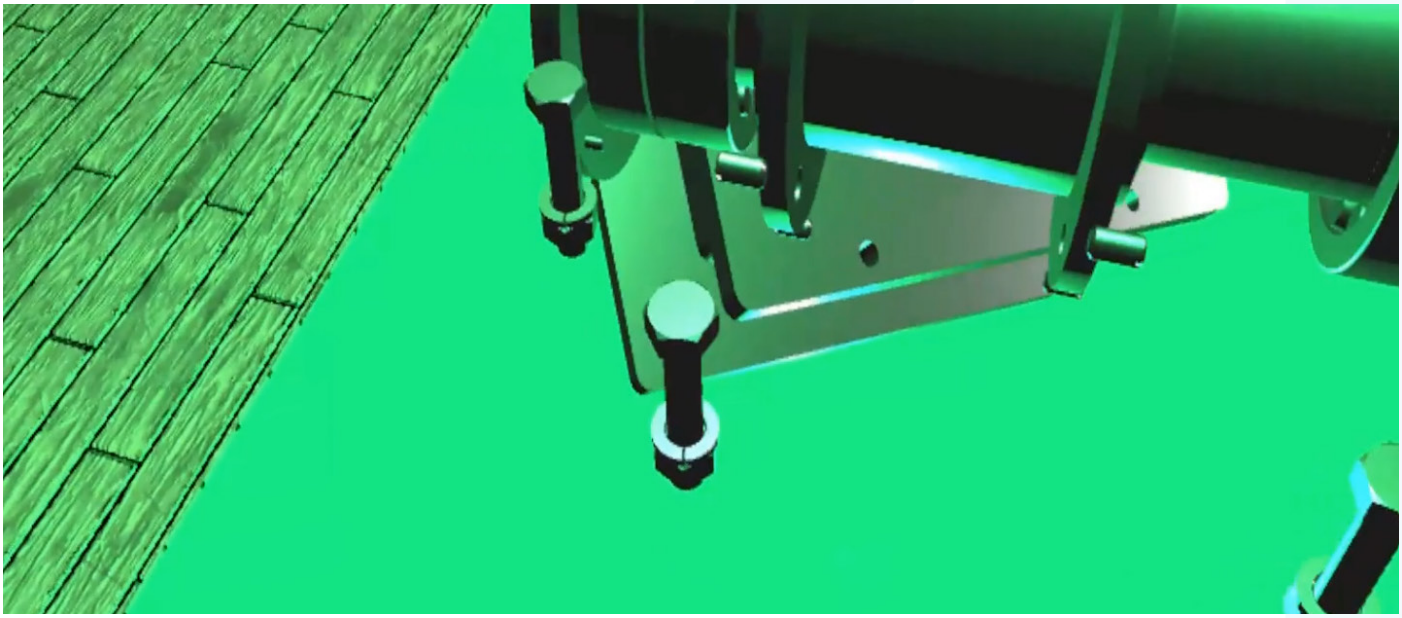












Para lograrlo, he seguido el siguiente procedimiento:

- Primero he importado el modelo del conjunto directamente desde el archivo exportado con el ensamblaje. En este caso, no vamos a realizar ninguna animación, por lo tanto, no tendremos el problema que teníamos anteriormente en las rotaciones animadas de los componentes. Por lo que es mejor importar el archivo que está perfectamente ensamblado y ajustado desde Inventor. Y acto seguido, he tenido que añadir el material otra vez a cada uno de los componentes del conjunto.
- Después, siguiendo el procedimiento explicado en el subapartado "*2.3.3 Pruebas I: Manipulación*" de este mismo documento, he programado que todos los componentes sean manipulables.
- A continuación, he creado los modelos de las zonas en Inventor Professional 2018 y los he importado en Unity siguiendo el mismo procedimiento que he explicado en el subapartado "*3.2.1 Modelado de componentes*" de este mismo documento. Y después, he añadido su material correspondiente a los diferentes modelos.
- Seguidamente, he activado la creación automática de los colisionadores de los modelos: "MesaDeTrabajo4", "MesaDeTrabajo5" y "MesaDeTrabajo6". Estos (activando la casilla "On Trigger"), me servirán como objeto colisionador para poder activar las funciones programadas en los scripts que se encargarán de modificar el tamaño (aumentándolo o disminuyéndolo).
- Una vez tenemos los colisionadores de las diferentes zonas, es hora de programar los scripts necesarios para que todo funcione correctamente. Para ello, he programado los siguientes scripts: "MesaDeTrabajo4_1", "MesaDeTrabajo5_1", "MesaDeTrabajo6_1", "ComponenteX_1" "ComponenteX_2".
- Y por último, para que este modo se active y se desactive el anterior, he creado el script "ModoLibre_1". Este lo he vinculado con el GameObject "ModoLibre", al que (como en los anteriores casos) le he añadido un colisionador de tipo "Box Collider", adaptando su geometría a la de su botón.

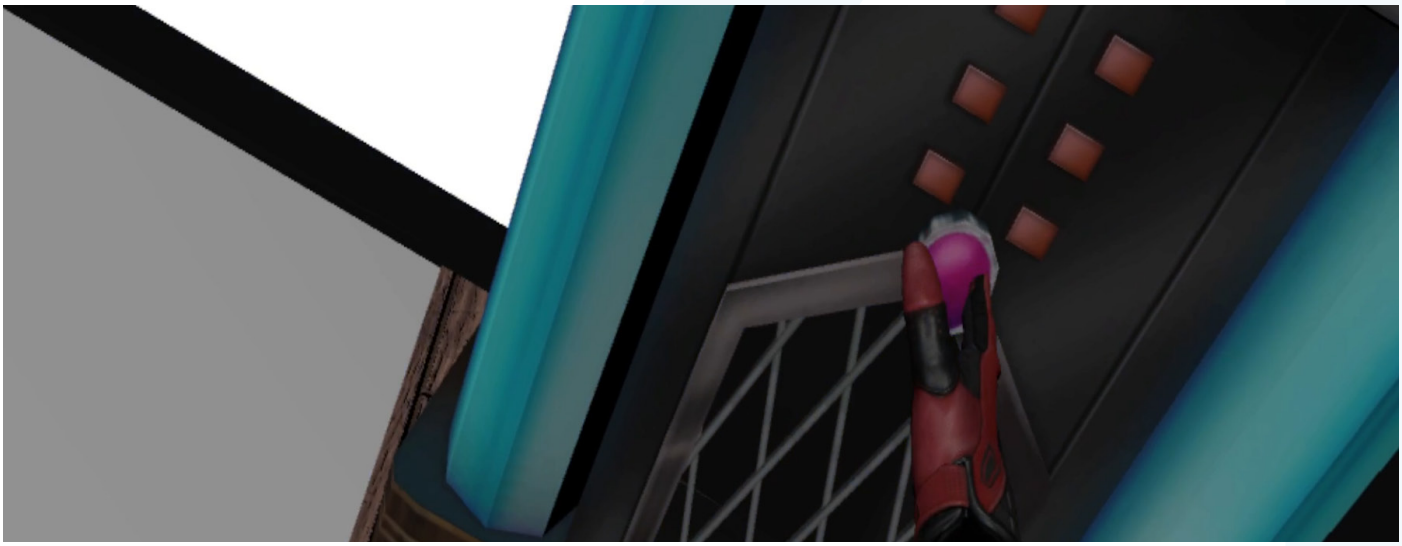
En el apartado "*2.2 Proyecto final*" del anexo "*Anexo II: Programación y scriptado*" del documento "*Anexos*", podemos encontrar el código de los scripts: "ModoLibre_1", "MesaDeTrabajo4_1", "MesaDeTrabajo5_1", "MesaDeTrabajo6_1", "ComponenteX_1" y "ComponenteX_2". Y en apartado "*3.1 Modelado en Inventor Professional 2018*" del anexo "*Anexo III: Modelado e importación de componentes*" del documento "*Anexos*" podemos encontrar las piezas modeladas: "MesaDeTrabajo4", "MesaDeTrabajo5" y "MesaDeTrabajo6".

4.2.4 Jukebox

Una de las ilusiones que tenía cuando tuve la idea de crear esta habitación en el mundo virtual, era la de añadirle elementos de mi interés personal. Por ejemplo, desde pequeño siempre me han gustado mucho los videojuegos y actualmente, me gusta jugar al billar de vez en cuando en mi tiempo libre. Es por ello que mi habitación virtual cuenta con una mesa de billar americano y con una máquina de juegos "arcade". Pero otra cosa que también me gusta mucho y que hago durante mi día a día, es escuchar música. Mientras camino hacia algún lugar, hago deporte, trabajo en mi proyecto, estudio etc... practicamente durante todo el día estoy escuchando canciones para relajarme, concentrarme, motivarme, divertirme etc...

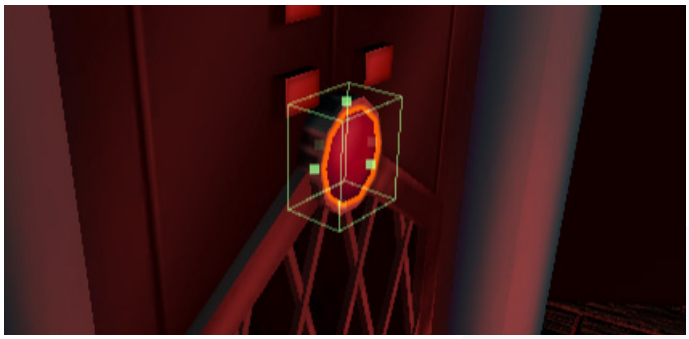
Por lo tanto, he decidido añadir una máquina de reproducción de música denominada "Jukebox".

Ponerme a jugar al billar o utilizar la máquina de arcade dentro de este entorno virtual, se trataría de un gadgect demasiado alejado del propósito principal del proyecto. Pero si en el mundo real escucho música mientras realizo múltiples actividades... ¿por qué no hacerlo también en el mundo virtual?

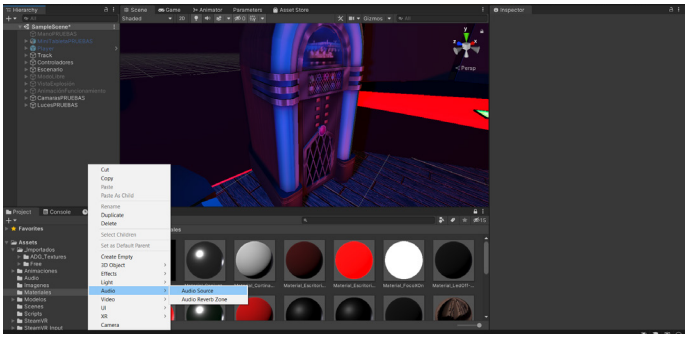


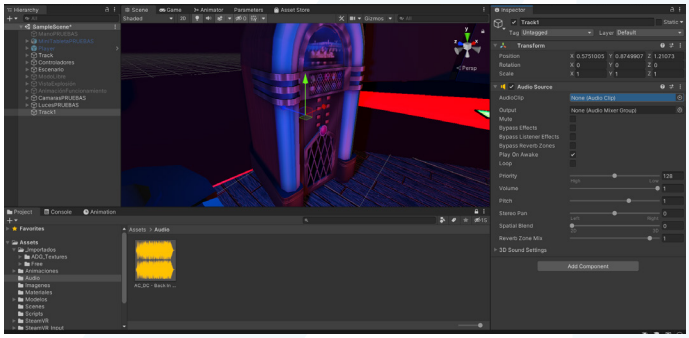
Para conseguir que la jukebox funcione, he seguido el siguiente procedimiento:

- Primero, he buscado en la jerarquía del GameObject "JukeboxModelo" los componentes del botón cuadrado superior izquierdo y del botón redondo inferior. Pulsando estos botones, el usuario podrá activar una canción y pararla cuando el quiera. Pero para ello, como ya sabemos, necesitamos crear los colisionadores. Por lo tanto, para cada uno de los componentes, he creado un colisionador de tipo "Box Collider".

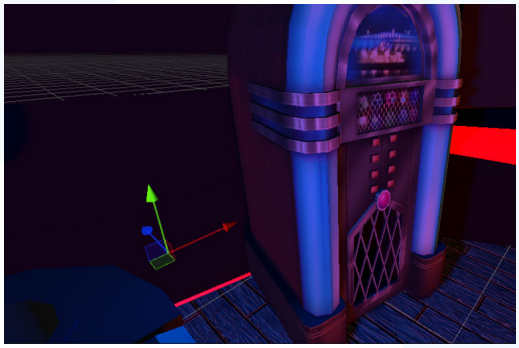


- Después, he descargado la canción "Back In Black" del grupo "ACDC" y la he añadido en formato .mp3 en los assets del proyecto (en la carpeta Audio). Acto seguido, he creado un GameObject de tipo "Audio Source", el cual he denominado como "Track1", y al que le he arrastrado el clip mp3 hasta su componente "AudioClip".





- A continuación, he creado una luz de tipo "Point Light" llamada "JukeboxLuz", para simular que la máquina se encuentra encendida mientras está sonando la canción.
- Y por último, una vez tenemos todos los GameObjects preparados, he programado los siguientes scripts: "BotonTrack1_1", "StopTrack1_1". Estos serán los encargados de activar y desactivar tanto la canción como la luz cuando los colisionadores de nuestras manos entren en contacto con los colisionadores de los botones.



En el apartado "*2.2 Proyecto final*" del anexo "*Anexo II: Programación y scriptado*" del documento "*Anexos*", podemos encontrar el código de los scripts: "BotonTrack1_1" y "StopTrack1_1". Y en apartado "*4.2 Luces*" del anexo "*Anexo IV: Materiales, texturas e iluminación*" del documento "*Anexos*" podemos encontrar los parámetros de la "JukeboxLuz"

5 Resultado y testeo final

A través del siguiente enlace, cualquier miembro que esté incluido en el grupo de la Universidad de Zaragoza de Google Drive, puede acceder a una carpeta de drive que he creado. En esta carpeta encontraremos:

- Un archivo .rar que contiene la carpeta del proyecto de Unity.
- Un vídeo en el que testeo el funcionamiento del software desde el momento de su ejecución hasta la finalización de todos las acciones posibles dentro del mismo

Enlace: https://drive.google.com/drive/folders/18zQWr3T_pUXauyRV-YYyQJo0WOykLYj-?usp=sharing

Bibliografía

1. Investigación sobre realidad virtual

https://www.adslzone.net/reportajes/tecnologia/realidad-virtual-rv/
https://www.nextpit.es/10-posibles-usos-de-la-realidad-virtual-vr
https://es.wikipedia.org/wiki/Metaverso
https://es.wikipedia.org/wiki/Tilt_Brush
https://ebac.mx/blog/que-es-unity-y-para-que-sirve
https://www.akademus.es/blog/tecnologia/desarrollo-programacion/juegos-hechos-con-unity/
https://blog.unity.com/technology/creating-immersive-photorealistic-vr-experiences-with-the-high-definition-render
https://docs.unity3d.com/Manual/ParticleSystems.html?_ga=2.206143827.158544682.1636194770-810793511.1633447384
https://www.hobbyconsolas.com/reviews/analisis-rick-and-morty-virtual-rick-ality-playstation-vr-201034
https://www.hobbyconsolas.com/reviews/analisis-rick-and-morty-virtual-rick-ality-playstation-vr-201034
https://en.wikipedia.org/wiki/Job_Simulator
https://en.wikipedia.org/wiki/The_Lab_(video_game)

2. Investigación sobre el hardware a mi disposición (cableado, ordenador etc...)

https://www.xataka.com/basics/displayport-vs-hdmi-cuales-diferencias
https://es.msi.com/blog/connect-multiple-gaming-monitors-to-laptop-guide

3. Investigación sobre HTC Vive Pro e instalación del dispositivo

https://www.youtube.com/watch?v=DcxWX7-zuAg
https://www.youtube.com/watch?v=hMV_LR2cCuI
https://www.youtube.com/watch?v=4VzCn4qI5gg
https://www.youtube.com/watch?v=dpyDt4u7Ius
https://es.wikipedia.org/wiki/HTC_Corporation
https://es.wikipedia.org/wiki/HTC_Vive
https://versus.com/es/htc-vive-vs-htc-vive-pro
https://www.sony.es/electronics/support/articles/00027623
https://es.digitaltrends.com/realidad-virtual/htc-vive-pro-2-vs-vive-pro/
https://es.wikipedia.org/wiki/Steam
https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@0.9/manual/index.html
https://docs.unity3d.com/Manual/com.unity.xr.management.html
https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647

4. Investigación sobre la importación de modelos en Unity

https://www.youtube.com/watch?v=pu_jQtn1CRo

5. Investigación sobre la forma de conexión entre HTC Vive Pro, Unity la programación en realidad virtual

https://www.youtube.com/watch?v=eja5W7bTcuk&t=138s
https://www.youtube.com/watch?v=87Br93ba83c&list=PLQ0QWapto-o_-UkCdPplNPO5MAzImzk7T
https://www.youtube.com/watch?v=SZyDInxHZto&list=PLQ0QWapto-o_-UkCdPplNPO5MAzImzk7T&in-

Bibliografía

dex=2
https://www.youtube.com/watch?v=iJ0oNYIUFJo
https://hub.vive.com/storage/tracking/index.html
https://www.youtube.com/watch?v=rbcIF_Gsnp0
https://www.youtube.com/watch?v=U8AOCx1lCww
https://www.raywenderlich.com/9189-htc-vive-tutorial-for-unity
https://www.youtube.com/watch?v=ndwJHpxd9Mo
https://www.youtube.com/watch?v=gGYtahQjmWQ&list=PLrk7hDwk64-a_gf7mBBduQb3PEBYnG4fU
https://docs.unity3d.com/Packages/com.unity.xr.openxr@0.1/manual/index.html
https://www.youtube.com/watch?v=5C6zr4Q5AIA
https://www.youtube.com/watch?v=9dc1zq8eH54
https://www.youtube.com/watch?v=eAycALcCBBI
https://www.youtube.com/watch?v=8IhBdEB3E7g
https://www.youtube.com/watch?v=furfe8E7SOA
https://www.youtube.com/watch?v=_yf5vzZ2sYE
https://www.youtube.com/watch?v=nwKvsRz12l0
https://www.youtube.com/watch?v=4tW7XpAiuDg&list=PLrk7hDwk64-a_gf7mBBduQb3PEBYnG4fU&index=8
https://www.youtube.com/watch?v=5NRTT8Tbmoc&list=PLrk7hDwk64-a_gf7mBBduQb3PEBYnG4fU&index=6
https://www.youtube.com/watch?v=VdT0zMcggTQ&list=PLrk7hDwk64-a_gf7mBBduQb3PEBYnG4fU&index=5
https://forum.unity.com/threads/xr-settings-not-showing.948544/
https://www.youtube.com/watch?v=aTiryu91jJ0
https://docs.unity3d.com/ScriptReference/Collider.html
https://docs.unity3d.com/es/2018.4/Manual/RigidbodiesOverview.html
https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html
https://docs.unity3d.com/ScriptReference/Transform.TransformVector.html
https://docs.unity3d.com/ScriptReference/Transform.TransformVector.html
https://docs.unity3d.com/ScriptReference/Transform.Rotate.html
https://docs.unity3d.com/ScriptReference/Transform.html
https://docs.unity3d.com/ScriptReference/TrailRenderer.html
https://social.msdn.microsoft.com/Forums/expression/es-ES/f708ce16-a645-4b77-b9b7-91242efa6b58/detener-o-evitar-ejecucion-en-if?forum=vcses
https://docs.unity3d.com/es/530/Manual/script-Serialization.html
https://www.youtube.com/watch?v=mZtYZ9FSw60
https://docs.unity3d.com/es/2018.4/Manual/class-Animator.html
https://docs.unity3d.com/es/2018.4/Manual/AnimationClips.html
https://www.youtube.com/watch?v=R67Xi4CH-tA
https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html
https://docs.unity3d.com/Manual/StandardShaderMaterialParameterAlbedoColor.html
https://viatea.es/curiosidades/como-calcular-mips/
https://docs.unity3d.com/ScriptReference/Time-deltaTime.html
https://docs.unity3d.com/es/2018.4/Manual/Tags.html
https://docs.unity3d.com/es/2018.4/Manual/animeditor-CreatingANewAnimationClip.html

Conclusiones

Después de haber realizado este proyecto, he llegado a una serie de conlcusiones.

En primero lugar, me he dado cuenta, todavía más, de que hoy en día, internet es una herramienta enorme con un abanico infinito de posibilidades. Cuando empecé este proyecto, nunca había utilizado nigún dispositivo de realidad virtual y el poco conocimiento que tenía acerca de programación en C#, venía de algunos ejemplos que vimos el año pasado en la asignatura optativa de "Entornos Interactivos 3D". Por lo que, una vez finalizado el proyecto, soy consciente de todo lo que he aprendido de forma autodidacta, simplemente teniendo un ordenador y conexión a internet.

En segundo lugar, he tenido una sensaciones muy grandes acerca de algo. Cuando era pequeño, me pasaba horas y horas jugando a videojuegos delante de la pantalla de mi ordenador de casa o delante del televisor de mi cuarto. A día de hoy, recuerdo que eran juegos geniales y fantásticos. Pero si buscamos en YouTube algún vídeo en el que se nos muestre el funcionamiento del mismo, veremos que la calidad gráfica y la jugabilidad estaban a años luz de la tecnología actual. Y el hecho de pensar que, estos videojuegos estaban desarrollados por empresas fabricantes de videojuegos, las cuales contaban con los mejores desarrolladores y programadores del mundo, y que yó, partiendo de un conocimiento muy básico sobre programación, haya podido desarrollar un software en realidad virtual, incrementa en mí esta sensación de la que hablaba antes.

La sensación sobre la que os hablo, no es que crea que sea un genio, para nada. Cualquiera de mis compañeros hubiera podido realizar este proyecto de la misma forma e incluso podría haberlo hecho de mejor forma. Esta idea sobre la que estoy hablando es la siguiente.

El mundo y la tecnología cambian a pasos agigantados. Y cada vez, lo hace de forma más rápida. Por lo tanto, cada día nos levantamos en un mundo que es mucho más competitivo de lo que era el día anterior. Y es por eso que, aunque algunas veces cueste y queramos dejarlo de lado, bajo mi opinión, debemos seguir trabajando de la forma más dura posible. Día a día. Porque de lo contrario, al ritmo que avanza este mundo, nos quedaremos atrás y sin opciones de seguir luchando.

Y en tercer y último lugar, decir que una de las grandes conclusiones que he obtenido realizando este proyecto, es que he disfrutado un montón haciendo lo que hacía y también de la gente que tenía a mi alrededor en el departamento.

Darle las gracias a David Ranz y Ramón Miralbés por haber confiado en mí para realizar este proyecto.

Y sobre todo, gracias Jose Antonio Gómez. Una persona que, cuando las cosas no me estaban saliendo y el entraba al laboratorio, más de una vez ha conseguido sacarme una sonrisa y hacerme sentir muy cómodo en esa lugar, dándome empujón para poder continuar probando, fracasando y volver a interarlo.

Muchas gracias Jose Antonio.