Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Exploración de la realidad virtual para la recreación de escenarios en la construcción

TITULACIÓN: Grado en Ingeniería Telemática

AUTORES: David Arroyo Recio
Marc Martinez Soler

DIRECTOR: Francisco Javier Mora Serrano

FECHA: 06 de febrero del 2017

Título: Exploración de la realidad virtual para la recreación de escenarios en la construcción

Autores: David Arroyo Recio
Marc Martinez Soler

Director: Francisco Javier Mora Serrano

Fecha: 06 de febrero del 2017

Resumen

Los enormes avances en la realidad virtual en los últimos años son palpables. Tras muchos años siendo algo reservado únicamente a los más inquietos o a grandes compañías que requerían de ella para usos específicos, por fin la tecnología de los dispositivos a nivel de usuario ha alcanzado el nivel de sofisticación necesario para tener una experiencia satisfactoria a un precio asequible.

Cada semana nos encontramos, y no sólo en prensa especializada, con noticias de nuevos dispositivos, nuevas aplicaciones, nuevos conceptos que prometen revolucionar el mercado y crear soluciones a problemas reales.

El uso para el sector del entretenimiento, con videojuegos a la cabeza, pero también con otros sectores como el turismo o el cine, es el que más fuerte está pisando y probablemente el más extendido.

Hay otros campos para los que su uso no resulta tan evidente y sin embargo puede aportar mejoras sustanciales si se consiguen desarrollar aplicaciones profesionales.

Para este proyecto, nos hemos centrado en uno de esos usos: el empleo de la RV para recrear escenarios reales y dotarlos de funcionalidades que resulten útiles para el sector de la construcción.

Contando con el modelo 3D de un edificio real, el B0 de la UPC-CIMNE ubicado en Campus Nord, hemos implementado una serie de casos de uso que pueden ser extrapolables a otros escenarios:

- Libertad de movimiento por el escenario en primera persona.
- Visualización de las instalaciones a través de la pared: agua, electricidad, gas, etc.
- Modificación de texturas y colores de determinados objetos colocados en el edificio, en nuestro caso un sofá.
- Etiquetas de posición para indicar en todo momento en qué localización del edificio te encuentras.
- Inclusión de contenido multimedia externo, mostrado en un televisor ubicado en el interior del edificio. Audio 3D proporcionado por 2 altavoces virtuales.
- Iluminación realista, simulación del ciclo solar.
- Visita guiada por el edificio.

Así mismo, hacemos un estudio de los distintos dispositivos de RV que se pueden emplear para interactuar con dichos escenarios, qué nos aportan, sus ventajas e inconvenientes.

Title: Exploring the virtual reality for recreation of scenarios in construction

Author: David Arroyo Recio
Marc Martinez Soler

Director: Francisco Javier Mora Serrano

Date: February 6th 2017

Overview

The huge advances in virtual reality in recent years are palpable. After years of being reserved for the early-adopters and large companies that required it for specific uses, the technology of user-level devices has finally reached the required level of development to have a proper experience at an affordable price.

Week after week we find, and not only in the specialized press, tidings of new devices, new applications, new concepts that promise to set a revolution to the market and create solutions to real problems.

The entertainment sector use, with video games at the top, but with other sectors such as tourism or cinema also, is the strongest one, and probably the most widespread.

There are other fields for which its use is not so obvious and yet can provide substantial improvements if professional applications are developed.

For this project, we have focused on one of these uses: the use of VR to recreate real scenarios and provide them with functionalities that are useful for the construction sector.

With the 3D model of a real building, the B0 of UPC-CIMNE located in Campus Nord, we have implemented some use cases that can be extrapolated to other scenarios:

- Freedom of movement in first person view.
- Visualization of the installations through the wall: water, electricity, gas, etc.
- Modifiable textures and colors of certain objects placed in the building. A sofa in our scenario.
- Position labels to acknowledge at what location of the building you are in.
- Inclusion of external multimedia content, shown on a TV located inside the building. 3D audio provided by 2 virtual speakers.
- Realistic lighting and simulation of the solar cycle.
- Guided tour of the building.

We also do a study of the different RV devices that can be used to interact with these scenarios, what they add to the experience, their advantages and disadvantages.

ÍNDICE

Índice de figuras

Índice de tablas

Capitulo 1. Motivaciones

Estamos asistiendo a una vorágine de avances en la realidad virtual (RV), tanto tecnológica como artísticamente. Uno de los síntomas más claros puede verse en la proliferación de las noticias en la prensa, especializada o no, así como en el interés de las grandes compañías tecnológicas, que están invirtiendo cifras escandalosas, del orden de centenares de millones, siendo uno de los ejemplos más claros la adquisición por parte de Facebook de Oculus VR, por 2000 millones de dólares [1].

Sin embargo, la realidad virtual está aún en una fase de despliegue masivo. Hay aún muchos retos que afrontar y soluciones nuevas para los problemas que van surgiendo. Eso implica que las oportunidades de desarrollo para distintos sectores, tanto en los directamente tecnológicos, por ejemplo los videojuegos o el diseño gráfico, como en los tradicionales, como podría ser la construcción, el turismo o la medicina, son muy amplios.

Este pasado año 2016 hemos visto como por fin, tras distintos prototipos y versiones para desarrolladores, han salido a la venta versiones finales de dispositivos tan potentes como el Oculus Rift [2] [3] o las HTC Vive [4] [5]. De su aceptación y de la disminución progresiva de su precio así como la incorporación de nuevos competidores, síntoma de madurez del sector, dependerá en gran parte que este 2017 sea por fin el año en que la realidad virtual alcance finalmente al gran público.

A nivel personal, a la hora de elegir nuestro TFG queríamos, además de conseguir como resultado final una aplicación útil, que el tema seleccionado fuese algo que nos interesase más allá del ámbito académico, y la realidad virtual no solo nos ofrece la posibilidad de cumplir esa ambiciosa meta, sino que además es un campo que desde ya hace cierto tiempo nos ha llamado poderosamente la atención.

Para crear una aplicación de RV multiplataforma éramos conscientes que íbamos a tener que emplear un motor de videojuegos, algo con lo que no habíamos tenido oportunidad de trabajar hasta ahora y que siempre nos había resultado atractivo.

En la actualidad, la realidad virtual es un conjunto de tecnologías, la gran mayoría de ellas, tales como la computación, comunicaciones, visualización e interfaces, se encuentran en el núcleo de actividades de una universidad politécnica como la UPC y de una escuela de telecomunicaciones, además de incluir actividades de modelado 3D y simulación, disciplinas propias de CIMNE. Disponemos pues del entorno educacional y herramientas adecuadas para introducirnos de lleno en este apasionante mundo.

Capitulo 2. Introducción

En este trabajo pretendemos abordar un estudio introductorio sobre la realidad virtual a dos niveles, por un lado realizando un análisis del estado de la realidad virtual: de dónde venimos, el estado actual y las posibilidades futuras que nos ofrecen; y en segundo lugar, explorando los usos de la tecnología más allá del sector del entretenimiento, mediante la implementación práctica de un aplicación específica, en particular sobre el modelo de un edificio real donde estudiaremos algunas de estas posibilidades.

Analizamos primero distintos dispositivos de interacción con la RV, cuáles son útiles para nuestro proyecto, cuáles tenemos a nuestro alcance y cuáles serían los ideales para tener la experiencia más óptima.

Como punto de partida, se ha empleado un modelo virtual del edificio B0 de la UPC-CIMNE de Campus Nord, disponible en formato 3D Studio Max para su representación fotorrealista.

En cuanto a la presente memoria, hemos tomado las siguientes consideraciones:

- Las palabras inglesas cuya traducción al castellano puedan llevar a confusiones, o que debido al entorno al que pertenecen no tenga sentido traducirlas (por ejemplo en los nombres empleados por la herramienta de desarrollo Unity como sería *Assets*), se presentan en cursiva.
- Los nombres de funciones o clases cuya traducción no tiene sentido, por ejemplo “void main()”, están entre comillas.
- Usamos los siguientes acrónimos:
    - RV: Realidad Virtual.
    - Unity: La herramienta Unity3D.
    - HMD: Head-mounted display.
    - FPS: Fotogramas por segundo.

El documento se ha estructurado como sigue:

Capítulo 1: Motivaciones, qué nos lleva a decidirnos por la RV como tema principal del TFG.

Capítulo 2: Introducción. Estas líneas: cuatro pinceladas sobre lo que se va a encontrar el lector y algunas consideraciones.

Capítulo 3: Objetivos. Descripción del objetivo general del trabajo y su desglose en objetivos específicos.

Capítulo 4: Estado del arte. Definición de un sistema de RV. Línea temporal de la RV con sus hitos más relevantes. Primera toma de contacto con la herramienta de desarrollo que vamos a emplear, Unity.

Capítulo 5: Diseño del demostrador. Definir qué prototipos son los más adecuados para una demostración útil, qué casos de uso contemplamos y los requisitos necesarios.

Capítulo 6: Implementación. Todo el proceso de creación del demostrador planeado, importación del modelo e implementación de las características deseadas. Conexión de los dispositivos de interacción.

Capítulo 7: Resultados. Especificaciones de la aplicación final y comparación con lo planeado inicialmente.

Capítulo 8: Conclusiones. Resumen final y conclusiones que sacamos una vez terminado el trabajo.

Además del listado de referencias y un anexo que incluye la parte del código desarrollado más representativa.

Capitulo 3. Objetivos

3.1. Objetivo inicial

El objetivo inicial del trabajo era muy amplio: Explorar las posibilidades de la realidad virtual, conocer los ingredientes que la constituyen, ver en qué estado se encuentra la tecnología, el nivel de sofisticación del desarrollo y qué podemos aportar nosotros.

Tras una primera fase de estudio del estado actual de la realidad virtual, así como dar los primeros pasos en nuestro aprendizaje de desarrollo de aplicaciones con uno de los motores de videojuegos más usados, Unity, y con la aportación por parte de CIMNE del modelo 3D del edificio, decidimos enfocar este trabajo hacia el **estudio de las posibilidades de la realidad virtual para la recreación de escenarios 3D en el sector de la construcción**.

3.2. Objetivos definidos

Los pasos que seguimos para conseguir nuestro objetivo los desglosamos a continuación:

Estudio de la realidad virtual: componentes básicos

- Explorar un software básico de desarrollo de aplicaciones de realidad virtual, en este caso Unity3D, descrito más adelante, para introducirnos en el desarrollo de aplicaciones 3D enfocadas al mundo virtual:
    - A recomendación del tutor y por otras fuentes consultadas nos decidimos por Unity3D por su facilidad en el aprendizaje, soporte de la comunidad al tratarse de una herramienta gratuita y resultados vistosos.
- Creación de una aplicación que funcione de forma fluida y estable en varios dispositivos:
    - En términos de la realidad virtual, resulta imprescindible que la aplicación o aplicaciones que se desarrollen ofrezcan una experiencia de usuario satisfactoria en la mayoría de dispositivos actuales, pues si los requisitos fuesen muy elevados, se limitaría su aplicabilidad, como ya ha sucedido en el pasado.
- Explorar distintos dispositivos de RV disponibles en el mercado:
    - La experiencia RV es tanto audiovisual como de navegación del usuario por los escenarios virtuales, y existe una amplia gama de posibles dispositivos disponibles. Es importante, por tanto, probar

aquellos dispositivos que estén a nuestro alcance, ver cuáles son los más adecuados para incorporar a nuestro proyecto y cuáles serían los óptimos para una compañía con presupuestos mucho más elevados.

La realidad virtual en el sector de la construcción

- Aportar valor añadido a un modelo virtual de un edificio real al cual tenemos acceso:
    - Entendemos que el hecho de disponer de acceso prácticamente ilimitado a un edificio real del cual tenemos su modelo 3D es una oportunidad única. Es por ello que nos decidimos a explorar qué podemos aportar nosotros a este modelo, que no tenga ya, que pueda resultar verdaderamente útil y extrapolable a otros modelos.
- Explorar posibles modificaciones a realizar durante el proceso de construcción de una obra:
    - Si dispones de una aplicación que te permita visualizar y moverte por el modelo planeado antes o durante la construcción, puedes modificar dicho modelo para ver cómo quedarían los cambios antes de construirlos.
- Verificar el resultado final de la obra:
    - Dado que el modelo del que disponemos no es exactamente igual a como finalmente se construyó, nos permite ver las diferencias al visualizar el modelo virtual y compararlo con el edificio final en persona.
- Desarrollar aplicaciones del modelo útiles para inmobiliarias:
    - La posibilidad de visitar una vivienda de forma remota es muy atractiva de cara a posibles compradores, ya que se pueden hacer una primera idea bastante buena de cómo es su futura casa, (dependiendo de la calidad gráfica y la fidelidad de la recreación del modelo), sin tener que ir en persona, o como paso previo a la visita en persona. Es una cuestión de optimizar tiempo, si dispones de los modelos 3D puedes visitar varias posibles viviendas en pocos minutos.

Capitulo 4. Estado del arte

4.1. Introducción a la RV

"Virtual Reality: A computer system used to create an artificial world in which the user has the impression of being in that world and with the ability to navigate through the world and manipulate objects in the world." [6]

“realidad virtual.
1. f. Inform. Representación de escenas o imágenes de objetos producida por un sistema informático, que da la sensación de su existencia real.” [7]

“La realidad virtual (RV) es un entorno de escenas u objetos de apariencia real. La acepción más común refiere a un entorno generado mediante tecnología informática, que crea en el usuario la sensación de estar inmerso en él. Dicho entorno es contemplado por el usuario a través normalmente de un dispositivo conocido como gafas o casco de realidad virtual. Este puede ir acompañado de otros dispositivos, como guantes o trajes especiales, que permiten una mayor interacción con el entorno así como la percepción de diferentes estímulos que intensifican la sensación de realidad.” [8]

En nuestras palabras, Realidad Virtual es aquel sistema que consigue crear la impresión en quien lo usa de que se encuentra en otra realidad distinta. Para lograrlo “engaña” a los sentidos. A cuantos más sentidos afecte la simulación, más inmersiva será la experiencia.

El primer sentido, al que la inmensa mayoría de dispositivos no puede dejar de lado, es la vista. Para tener una buena inmersión es necesario que únicamente veamos aquello que se muestra a través de la pantalla de las gafas. Por ello se cubren los laterales y las zonas superior e inferior de las gafas, para que nuestro campo visual se limite a aquello que se quiere mostrar, sin interferencias externas.

A diferencia de los actuales dispositivos multimedia de consumo masivo, la realidad virtual introduce además la visión estereoscópica, que ya se ofrece en la salas de cine 3D y visión envolvente, con mayor o menos profundidad de campo en función de la tecnología.

En segundo lugar está el oído. En una aplicación en la que por ejemplo estuviésemos en un coche en movimiento sería complicado sentirnos parte de la acción si no se oyese el ruido del motor, los frenos, etc.

Para mayor inmersión se recomienda usar cascos que aíslen del ruido externo. De este modo oiremos únicamente lo que suceda en la aplicación. En este caso también disponemos, más allá del estéreo, de tecnologías de sonido envolvente y 3D.

Los dispositivos audiovisuales suponen la punta de lanza de la realidad virtual, tanto por tratarse de los principales elementos necesarios para generar la inmersión, como por disponer de la tecnología más madura. No obstante, ya existe oferta para los demás sentidos, como se describe a continuación.

Para el tercer sentido en importancia, el tacto, hay unas pocas soluciones tales como el Glove One [9], basadas en guantes con distintos sensores y actuadores vibrotáctiles, que tratan de simular el contacto con objetos haciendo presión en aquellos puntos de la mano en los cuales, en la aplicación, un objeto está tocando a la mano virtual. Una solución que también incluye los brazos es Manus VR [10].

Reproducir el sentido del tacto puede significar un gran avance, no solo en términos de realidad virtual, sino también en la manera en que tratamos con la información disponible en los ordenadores u otros dispositivos. Más allá de la experiencia táctil como tal, un buen dispositivo háptico nos podría permitir manipular objetos virtuales tal y como lo hacemos con los reales, tal y como usamos instrumentos musicales, artísticos o industriales. Las manos son nuestros principales instrumentos de interacción con nuestro entorno que, actualmente, ven muy disminuida su capacidad al tener que entenderse con el mundo digital a través del teclado y ratón.

Fig. *4.1* Glove One

En cuanto al olfato, las pocas soluciones disponibles se basan en crear olores con quemadores de esencias, que se activan cuando aparecen en pantalla determinados objetos como una flor, el mar, el césped, etc.

Algunos de los dispositivos que aportan olor a la experiencia de RV son el FeelReal [11], actualmente en preventa y que se acopla a la parte inferior de algunos modelos como el Oculus Rift o el Samsung Gear VR, o el Olorama [12], desarrollado en Valencia y que se basa en un difusor para combinar con gafas de RV pero separado físicamente de él.

Como curiosidad, el Nosulus Rift [13] es un dispositivo creado por Ubisoft para el videojuego “South Park: The Fractured But Whole”, que permite oler las flatulencias de los personajes.

Fig. *4.2* Nosulus Rift y sus consecuencias

Quizás el sentido más difícil de emular de todos, y del cual no hay a día de hoy más que algunos proyectos en curso, es el gusto. En primer lugar por la limitación física y poco higiénica de su implementación. Para una óptima percepción del gusto tiene que haber una superficie en contacto directo con la lengua, mojando con saliva cada vez dicha superficie. En segundo lugar porque no se han obtenido resultados del todo satisfactorios en la imitación de un sabor mediante la tecnología. Una opción más adecuada podría ser la que están probando un pequeño grupo de investigadores de la Universidad de Tokio [14], que se basa en engañar al cerebro mediante la vista, el olfato y el oído para que percibamos un determinado gusto.

La tecnología ha llegado a un nivel de sofisticación en el cual, con un ordenador a un precio no prohibitivo, (entre 1000-2000€), podemos obtener una experiencia fluida con una calidad en las texturas que incluso, en ocasiones, pueden hacer dudar de si lo que estamos viendo se trata de un vídeo real o una simulación en 3D.

El nivel de miniaturización de los equipos al que se ha llegado permite que con dispositivos móviles de pequeño tamaño como los *smartphone* tengamos suficiente potencia gráfica para ejecutar muchas de las aplicaciones de RV creadas y pensadas para ejecutarse en un ordenador.

Por comodidad y usabilidad, la mayoría de dispositivos de RV usados a día de hoy son del tipo HMD (Head-mounted display), es decir dispositivos colocados en la cabeza, similares a las gafas.

Dentro de los dispositivos HMD hay que hacer una diferenciación entre aquellos que requieren de un hardware externo y los que lo traen integrado en el propio dispositivo.

Los más sencillos y consecuentemente más baratos son los que dejan la computación en manos de otro dispositivo. Los más simples se pueden incluso montar con cartón, como las CardBoard de Google [15]. Se pueden encontrar fácilmente gafas de plástico a las que se les acopla tu Smartphone desde unos 10€ en tiendas de importación China como Aliexpress [16] o Gearbest [17].

A primera vista, resulta llamativo el interés de empresas como Google por dispositivos tan básicos y elementales como las gafas CardBoard, disponiendo de la tecnología, capacidad y visión para proponer *gadgets* más sofisticados, en línea con las Oculus Rift o similares. Sin embargo, la historia de la realidad virtual muestra muchos picos de entusiasmo que no han llegado a materializarse en una adopción real por parte de la sociedad, muy probablemente por sus altos requerimientos en términos de aprendizaje para su uso y coste del equipo.

Para entender un poco el contexto de la realidad virtual, veamos cómo han avanzado los dispositivos desde el primero hasta la actualidad:

- La mayoría de fuentes indica como primer dispositivo de RV el Sensorama, una máquina ideada en 1956 y cuyo prototipo fue construido en 1962 por Morton Heilig, que mostraba imágenes estereoscópicas en 3D, con sonido estéreo, silla que vibraba, ventiladores para simular viento en las escenas y difusores de olores. Para mayor inmersión introducías la cabeza en la máquina de modo que te aislaras del exterior. Obsérvese cómo en 1956 ya se tenían todos los ingredientes necesarios, ya se tenía desarrollado el concepto fundamental de la realidad virtual, pero aún faltaba una tecnología más avanzada y madura.

Fig. *4.3* Sensorama

- En 1960 el mismo Heilig patentó [18] lo que llamó “Telesphere Mask”, con un diseño sorprendentemente parecido a las gafas de RV que usamos hoy día, del tipo HMD, pero con obvia inferioridad tecnológica, que ofrecía imágenes en 3D y audio estéreo.

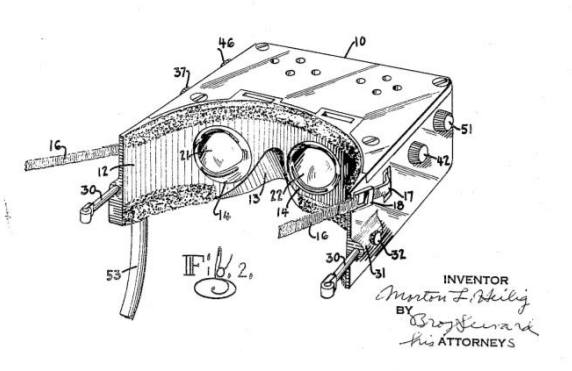

Fig. *4.4* Patente de la “Telesphere Mask”, de 1960

- Un año más tarde, en 1961, en Philco Corporation desarrollaron un dispositivo HMD que incluía detector de movimiento de la cabeza para guiar una cámara, conectada al visor mediante circuito cerrado.

  De esta manera podían tener visión remotamente de escenarios en los que es peligroso estar físicamente. Se usó, por ejemplo, para simulaciones de vuelo en oscuridad total.

- En 1965 Ivan Sutherland describió “The Ultimate Display” [19], las bases de lo que él consideraba que tenía que tener una buena recreación de RV, así como los distintos dispositivos de entrada que servían para interactuar con el mundo virtual. La tecnología disponible por el momento era insuficiente para lo que él definió, pero sirvió de base para diseñadores posteriores.

- Tres años más tarde el mismo Sutherland junto a su equipo del MIT, (Massachusetts Institute of Technology), construyó la “Espada de Damocles”, un dispositivo muy pesado, tenía que estar sujeto en el techo, del tipo HMD y que mostraba gráficos sencillos generados por ordenador.

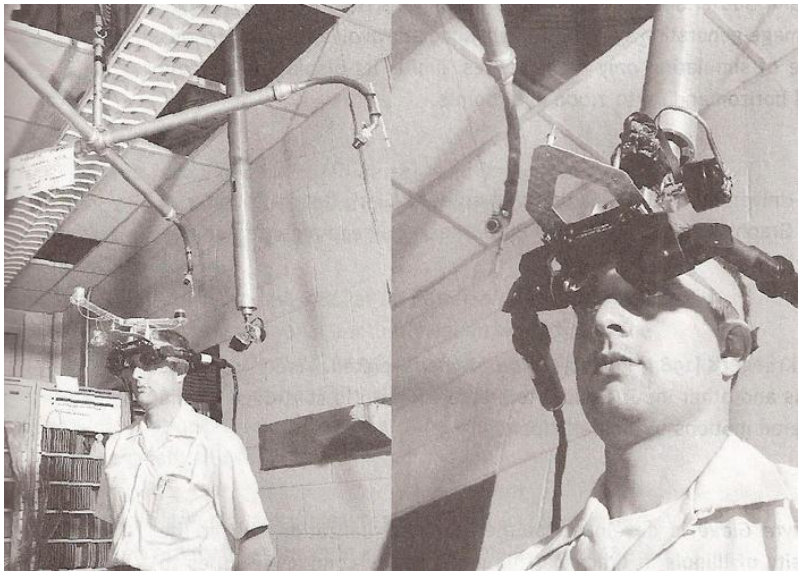

Fig. *4.5* La Espada de Damocles, de Sutherland

- En 1975 el científico y artista Myron Krueger inventó “Videoplace”, un entorno que permitía al usuario interactuar con las imágenes proyectadas, capturando con una cámara los movimientos que realizaba y procesándolos para modificar dicha imagen. Los cambios realizados los podía visualizar el siguiente usuario y modificarlos también. Es precursor de sistemas de reconocimiento de movimiento a través de la cámara en dispositivos más modernos como los EyeToy [20] para la Play Station 2 o los Kinect de Microsoft [21].

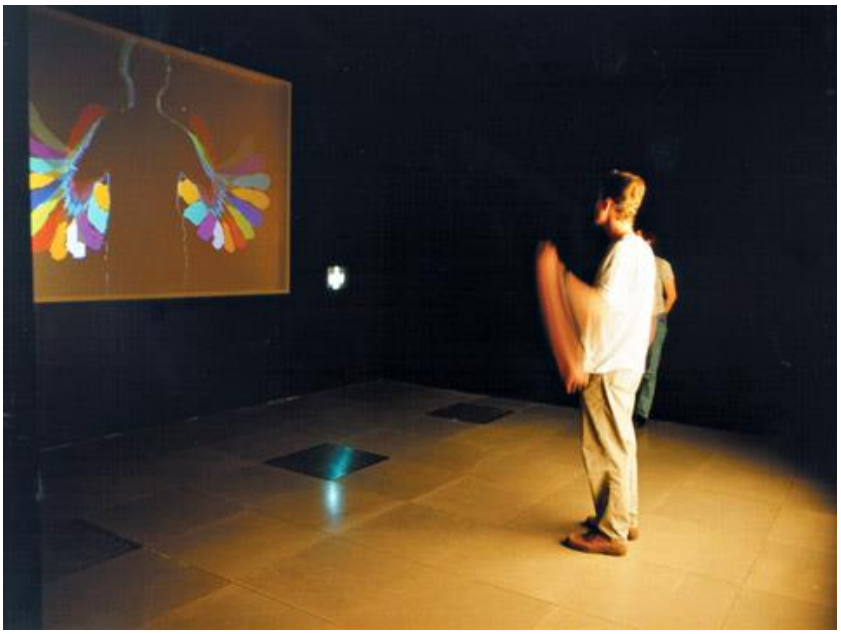

Fig. *4.6* Videoplace, captura los movimientos de los brazos del usuario y dibuja con distintos colores por donde se ha movido

- "Aspen Movie Map", en 1978 y desarrollada por un equipo del MIT permitía visitar la ciudad de Aspen, en Colorado usando tres pantallas y un mando para moverse por la ciudad. Era asombrosamente parecido al Google Street View actual e incluía características tan interesantes como información adicional de los edificios mostrados, posición actual dentro de la ciudad cambiando a un mapa 2D para mostrarla o la posibilidad de ver las fotos de la misma posición en distintas estaciones del año.

  La aplicación sirvió entre otros para el uso militar de familiarizar a los soldados con una determinada ciudad sin necesidad de estar físicamente en dicho lugar.

Fig. *4.7* Aspen Movie Map en funcionamiento

- En 1984, Jaron Lanier funda “VPL Research”, una de las primeras compañías que vendía productos de RV y responsable de productos como el “Data Globe”, un guante con sensor de movimientos que un ordenador procesaba y que tenía usos potenciales tan atractivos como la cirugía remota. A otro nivel estaba el “Data Suit”, un traje completo con sensores para medir el movimiento de piernas, brazos y tronco. La compañía quebró en 1990 y sus patentes fueron posteriormente compradas por “Sun Microsystems”. [22]

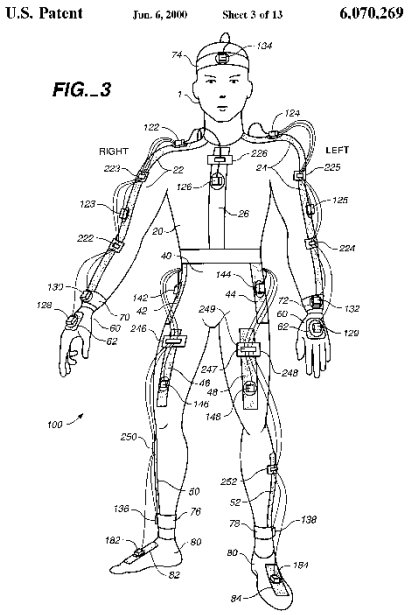

Fig. *4.8* Patente del “Data Suit”

- En 1990 y años posteriores la compañía “Virtuality Group” lanza una serie de máquinas de tipo arcade para jugar con procesado del juego a tiempo real usando un visor y un joystick. Tiene como importante ventaja respecto a otros competidores el hecho que permite el multijugador a través de otras máquinas conectadas.

Fig. *4.9* Uso multijugador de una máquina de Virtuality

- Un año más tarde Antonio Medina diseñó el sistema de RV para controlar a distancia el robot que se movía sobre la superficie de Marte. Obviamente debido a la enorme distancia entre planetas tenía unos retrasos entre que se emitía la orden de movimiento hasta que el robot físicamente se movía.

- En 1994 SEGA lanza el “SEGA VR-1”, un dispositivo HMD inicialmente pensado tanto para casa como para máquinas recreativas pero que finalmente únicamente se lanzó para recreativas. Contaba con detector de movimiento de la cabeza, sonido estéreo y juegos en 3D con un nivel gráfico muy bueno para la época.

- En 1995, Nintendo (japonesa al igual que SEGA), produce el “Virtual Boy”, con juegos en 3D (aunque monocromáticos). Fue tal fracaso en ventas que ni siquiera llegó a comercializar en Europa. Solo salieron 22 juegos a la venta y su precio de salida fue de unos 180$.

Fig. *4.10* Virtual Boy de Nintendo

- La trilogía de películas Matrix, cuya primera entrega se estrenó en 1999 trata sobre un futuro en el cuál las máquinas han esclavizado a la población y sus mentes están conectadas a una simulación de un mundo virtual generado por ordenador.

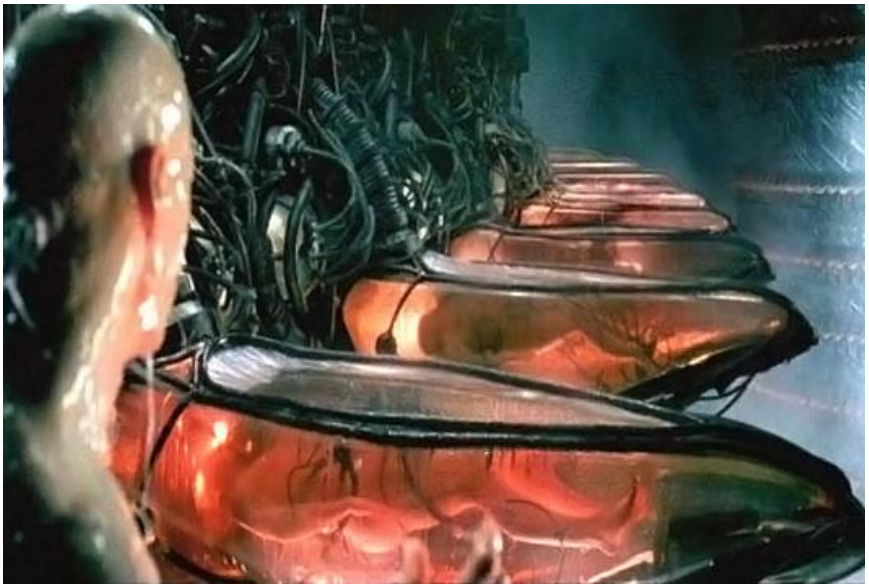

Fig 1 Humanos conectados en Matrix

- En 2010 Microsoft lanza Kinect, un dispositivo para la consola Xbox 360 que permite reconocer gestos, comandos de voz, imágenes y objetos. Compite con Wii Motion Plus de Nintendo y con Play Station Move, de Sony, pero es más sofisticado.

- 2 años más tarde Oculus lanza una campaña de financiación en Kickstarter para desarrollar unas gafas de RV muy potentes, las Oculus Rift. Recaudan 2,5 millones de dólares y realizan la primera versión, cuyo prototipo envían a los inversores que hubiesen aportado más de 300$. En 2014 Facebook compra Oculus por 1450 millones de dólares. [23] Finalmente, en 2016 salen a la venta en todo el mundo por un precio inicial de 600$.

  Los requisitos mínimos son especialmente altos, por el momento (2017) sólo son compatibles con SO Windows y necesitan como mínimo 8GB de RAM, una tarjeta gráfica Geforce GTX 970 y un procesador Intel Core i5 de 4ta generación.

- En 2016, salen a la venta las gafas de realidad virtual de Sony, las Play Station VR a un precio de 399€. Estas gafas se integran con la Play Station 4 para disfrutar de juegos adaptados al formato de RV.

  El mismo año salen las gafas de la compañía taiwanesa HTC, las HTC Vive. Igual que las Oculus Rift estas son compatibles con Windows y tienen prácticamente los mismos requisitos mínimos para asegurar un correcto funcionamiento.

- Actualidad (2017), Google presenta sus aplicaciones de realidad virtual para Android: Google VR [24] y Daydream [25]. A su vez, The Void [26] aporta escenarios reales que transforma haciendo uso de la RV.

  Uno de sus competidores directos, Microsoft, apuesta por la realidad aumentada [27], una tecnología parecida a la RV que se ayuda de la imagen obtenida a través de una cámara para añadir elementos virtuales al entorno físico real, con su dispositivo MS Hololens [28] a la vez que surgen otras nuevas empresas enfocadas específicamente en este campo, como MagicLeap [29].

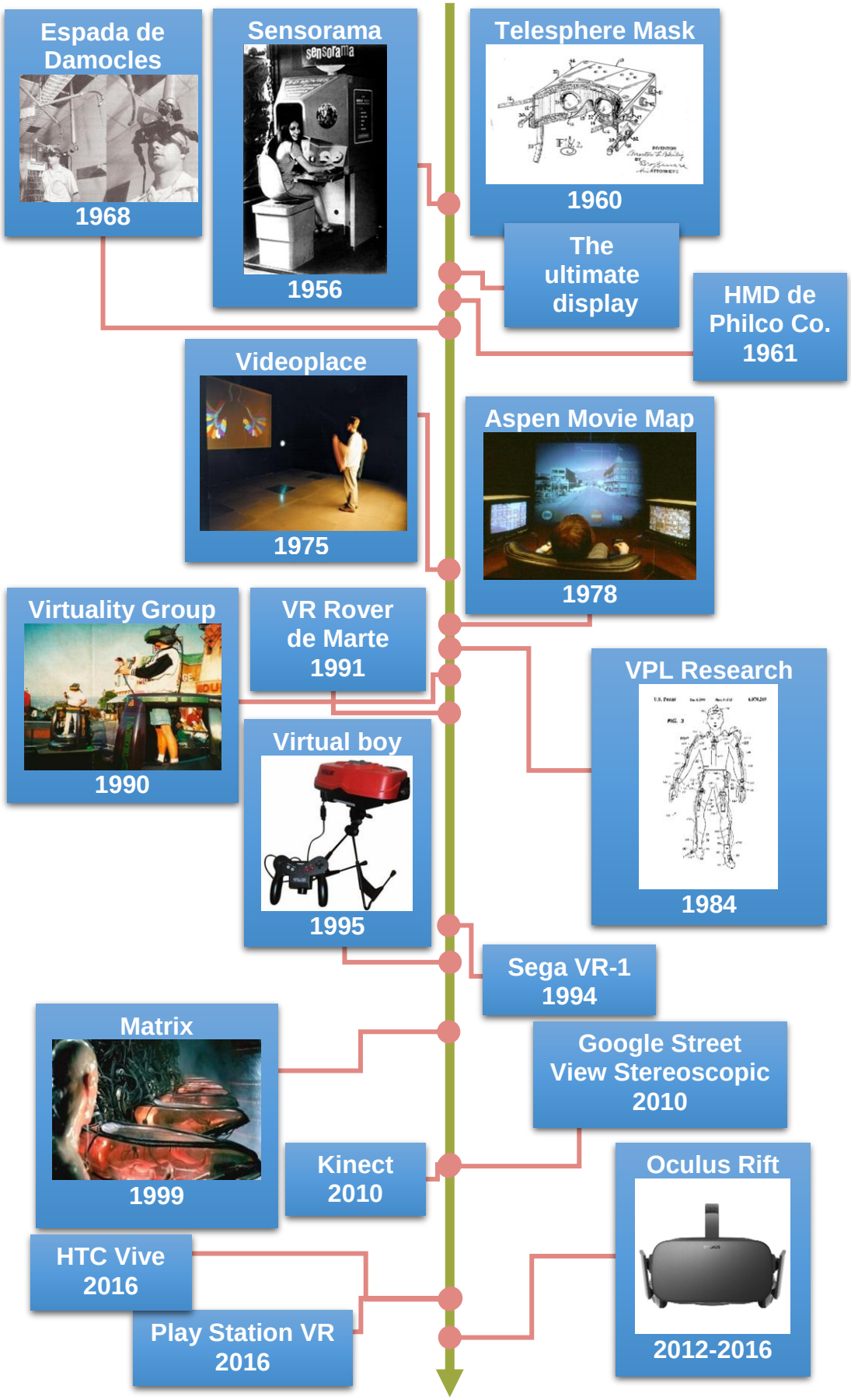

Fig. *4.11* Línea temporal de la realidad virtual (fuente propia)

4.2. Componentes y dispositivos

Para poder visualizar e interactuar con esos demostradores necesitamos diferentes piezas de hardware.

Hemos dividido estas piezas en 3 apartados: el soporte físico para la aplicación, la visualización de la aplicación y la interacción con la aplicación.

4.2.1. Soporte físico

Lo primero que tenemos que decidir en cuanto a hardware es dónde podemos hacer funcionar nuestro demostrador. Los dos soportes principales son un Ordenador y un dispositivo móvil.

Ordenador personal (PC)

La herramienta esencial para llevar a cabo este proyecto es un ordenador. No sólo como soporte para los demostradores, sino que también es el instrumento principal para desarrollar aplicaciones con Unity.

Trabajar con realidad virtual requiere mucha potencia de cálculo, tanto del procesador en sí (CPU) como de la tarjeta gráfica, y no es extraño que la configuración más apropiada coincida con la recomendada para ejecutar los videojuegos de mayor calidad, que también son exigentes en términos de memoria, cómputo y representación de modelos tridimensionales, lo que se denominan PCs para *gaming*. Los diseñadores, arquitectos e ingenieros también suelen emplear equipos similares.

Si, además, revisamos los requerimientos de dispositivos complementarios para realidad virtual (gafas, detectores gestuales, etc) y del software más habitual, encontramos que una configuración apropiada sería del tipo (a fecha de febrero de 2017):

- CPU: I7-6700K o similar (a 3.1 GHz o superior)
- GPU:
    - Compatible con DX11
    - NVIDIA GeForce™ GTX 1060, AMD Radeon™ RX 480, equivalent o equivalentes
- Almacenamiento: 256 - 512GB SSD
- Almacenamiento adicional: 1 - 2 TB HDD
- RAM: mínimo 16 GB, recomendable 32GB
- Varios puertos USB (3.0 y 3.1 en función de dispositivos, como podrían ser las MS Kinect)
- Puerto de video: HDMI 1.4 2880×1440 @ 60 Hz o HDMI 2.0 or DP 1.3+ 2880×1440 @ 90 Hz
- Bluetooth 4.0 para accesorios
- Sistema operativo: Windows 8.1, 10

Dispositivo móvil

Los dispositivos móviles actuales tienen todo lo necesario para hacer funcionar correctamente una aplicación basada en la realidad virtual.

Los teléfonos móviles incorporan, en su gran mayoría, una serie de sensores, algunos de ellos muy útiles para la realidad virtual.

Tienen un giroscopio [30], un dispositivo capaz de medir, mantener o cambiar su orientación en el espacio. Una herramienta muy útil para poder discernir en qué dirección está apuntado el móvil, por lo que se puede reproducir su rotación en el entorno virtual.

4.2.2. Visualización

Una de las bases de la realidad virtual es conseguir un sistema inmersivo, y la vista es el sentido que juega uno de los papeles más importantes.

Para “engañar” a la vista hay que conseguir generar una imagen estereoscópica Fig. *4.12*, esta imagen pretende separar lo que ve el ojo derecho del izquierdo. Son dos imágenes separadas entre sí cierta distancia y con una pequeña variación en el ángulo para reproducir lo que vería cada uno de nuestros ojos. [31]

Fig. *4.12* Imagen estereoscópica

Los dispositivos que nos permiten reproducir esta imagen de tal forma que podamos “engañar” al ojo y recrear una visión 3D para nuestro sistema VR son muchos, a continuación detallaremos los más remarcables para nuestro proyecto.

Google CardBoard

Estas pequeñas gafas de realidad virtual han sido diseñadas y son distribuidas por Google. [32]

Es la plataforma más asequible de todas ya que son de cartón, de hecho se compran desplegadas y las debes montar tú mismo. Además las puedes volver a desmontar para guardarlas.

El diseño es muy simple, tan solo constan de un soporte para el móvil y un par de lentes para convertir la imagen estereoscópica en una imagen 3D.

Diseñados principalmente para su uso con dispositivos Android e iOS, pero siempre que los dispositivos incorporen un giroscopio podrán ser usados en esta clase de dispositivos de visualización.

Fig. *4.13* Google CardBoard

VR Box

Las VR Box son la versión más lujosa de las Google Cardboard. [15]

Son algo más caras, pero son de plástico, lo que les otorga más resistencia. Tienen una cinta para que se sujeten mejor a la cabeza. Las Google Cardboard se tienen que aguantar necesariamente con, por lo menos, una mano.

Además la parte posterior es una tapa, lo que en la mayoría de teléfonos deja la cámara al descubierto. De esta forma también pueden ser usadas para aplicaciones de Realidad Aumentada, aunque también pueden servir para mostrar imágenes reales en aplicaciones de RV.

Fig. *4.14* VR BOX

4.2.3. Interacción

Hasta aquí, la experiencia ya se puede considerar propiamente de RV, pero con tan solo ver y oír no es suficiente para nosotros. El hecho de poder interactuar con la aplicación ofrece esa experiencia que realmente le da un valor añadido a nuestro proyecto.

Teclado y ratón

La opción por defecto y con más posibilidades, ya que dispones de multitud de teclas, (generalmente más de 100), completamente configurables desde la interfaz de Unity.

En cuanto al ratón puede sustituir al giroscopio pero reduce la experiencia inmersiva.

El principal inconveniente es la dificultad de uso, su poca ergonomía y funcionamiento poco intuitivo. Si la aplicación tiene distintas funciones que requieren más teclas además de las de dirección, resulta complicado encontrarlas sin poder visualizar el teclado ya que las gafas de RV nos lo impiden.

WiiMote

La WiiMote incorpora una serie de sensores que pueden ser muy útiles ya no solo para controlar aplicaciones de realidad virtual, sino para aplicaciones 3D en general.

Este incorpora un acelerómetro de 3 ejes para detectar hacia donde movemos el mando en cualquiera de las 3 dimensiones espaciales, un giróscopo para medir la orientación del dispositivo y junto a una barra de leds un detector de movimiento para capturar hacia dónde está apuntando el dispositivo.

Además tiene accesorios como el *nunchaku* que añade un *joystick* y más botones para poder añadir funcionalidades extra.

La principal desventaja que tiene para nosotros es que no es compatible con dispositivos Android con una versión superior a la 4.2 (Jelly Bean) ya que este controlador usa el protocolo L2CAP para el intercambio de datos, y este protocolo fue eliminado de las API de Google Android. Además, la conexión de la barra led para el detector de movimiento tampoco parece ser compatible.

Fig. *4.15* Nunchaku (superior) y WiiMote (inferior)

Dualshock 3/4

Este controlador no deja de ser un *Gamepad* de Sony, incorpora 17 botones y dos joysticks de 2 ejes.

Cuenta con 3 modos de vibración potenciado por 2 motores de vibración con una rueda excéntrica, Fig. *4.16*, uno a cada extremo del mando, que le permiten vibrar simulando una fuerza hacia la derecha, hacia la izquierda o central, según qué combinación de motores estén girando. Por ejemplo, si únicamente gira el de la derecha, simula una fuerza hacia esa misma dirección.

Fig. *4.16* Motor de vibración con una rueda excéntrica

Tiene un acelerómetro de 3 ejes y un giroscopio, aunque con menor precisión que los de la WiiMote.

Igual que la WiiMote, este también usa el protocolo L2CAP, por lo que tampoco es compatible con los dispositivos Android actuales.

Fig. *4.17* Dualshock 4 – Dualshock 3

Gamepad para Android

Este es muy parecido a los Dualshock de Sony, solo que están especialmente diseñados para funcionar con móviles, de esta forma, no solo son compatibles, sino que además es fácil conectarlos y usarlos.

Habitualmente, no cuentan con ningún tipo de sensor, (giroscopio, acelerómetro, etc.), ya que se supone que el dispositivo móvil queda unido de forma solidaria al mando, supliendo sus funciones de sensor.

En el caso de uso de dispositivos de RV del tipo HMD que requieren el uso del dispositivo móvil, este se acopla al propio dispositivo en vez de al mando y ya no es necesario el uso del *joystick* derecho para el movimiento de la cámara ya que el giroscopio del móvil suple su función.

Fig. *4.18* IPEGA PG-9017S

Virtuix Omni VR

El Virtuix Omni VR [33] es un dispositivo que nos facilita el movimiento del personaje dentro de la aplicación de RV.

Si bien ya contamos con un giroscopio para la visión 360 del personaje, lo ideal sería que también camináramos de verdad para mover el personaje. Este dispositivo es exactamente para eso. Consta de una cinta mecánica que registra el movimiento del usuario sin dejar que este se mueva del sitio.

Fig. *4.19* Virtuix Omni VR

Vive tracker

El Vive tracker [34] es un dispositivo que permite realizar un seguimiento de la posición y gestos que hace el usuario con las manos.

Esto es interesante en las aplicaciones de RV, ya que permite interactuar de una forma más ágil y real con el entorno virtual. Además, al saber la posición relativa de las manos respecto al cuerpo real del usuario, se pueden reproducir unas manos virtuales, de modo que el usuario las vea en el escenario virtual cómo se mueven de acuerdo a sus gestos reales. Esto último ayuda a crear una sensación aún más completa de inmersión.

Fig. *4.20* Vive tracker

4.3. Introducción a Unity

Existe una amplia variedad de software para la realidad virtual. Debido a sus requerimientos en cuanto a manipulación de escenarios tridimensionales, tratamiento de renderizado gráfico e interacción con el usuario, uno de los más populares es Unity.

Es por ello que para profundizar de forma práctica en los elementos más convencionales que forman parte de una aplicación de RV, seguiremos el desarrollo de uno de los tutoriales introductorios de Unity, que describen paso a paso el manejo esencial del programa y, por lo tanto, sus principales ingredientes.

Hay que dejar claro que Unity no es, por sí mismo, una aplicación directamente enfocada a la realidad virtual sino un motor de videojuegos que permite la recreación de escenarios 3D y la posibilidad de operar con ellos. Sin embargo, presenta opciones de configuración que permiten lanzar la aplicación en modo estereoscópico, para su uso mediante dispositivos de RV.

La sección de tutoriales de la página oficial [35] es realmente útil, ya que los vídeos del desarrollo de los distintos ejemplos no tienen cortes y explican el porqué de cada acción y el funcionamiento del código que muestran, con lo cual puedes reproducirlo tú fácilmente.

El tutorial que recomienda para los usuarios que se enfrentan por primera vez a Unity es el Roll-a-ball [36], que consiste en crear un escenario cuadrado con bordes que lo delimitan bre el cual colocamos una bola con la que recoger distintos objetos coleccionables.

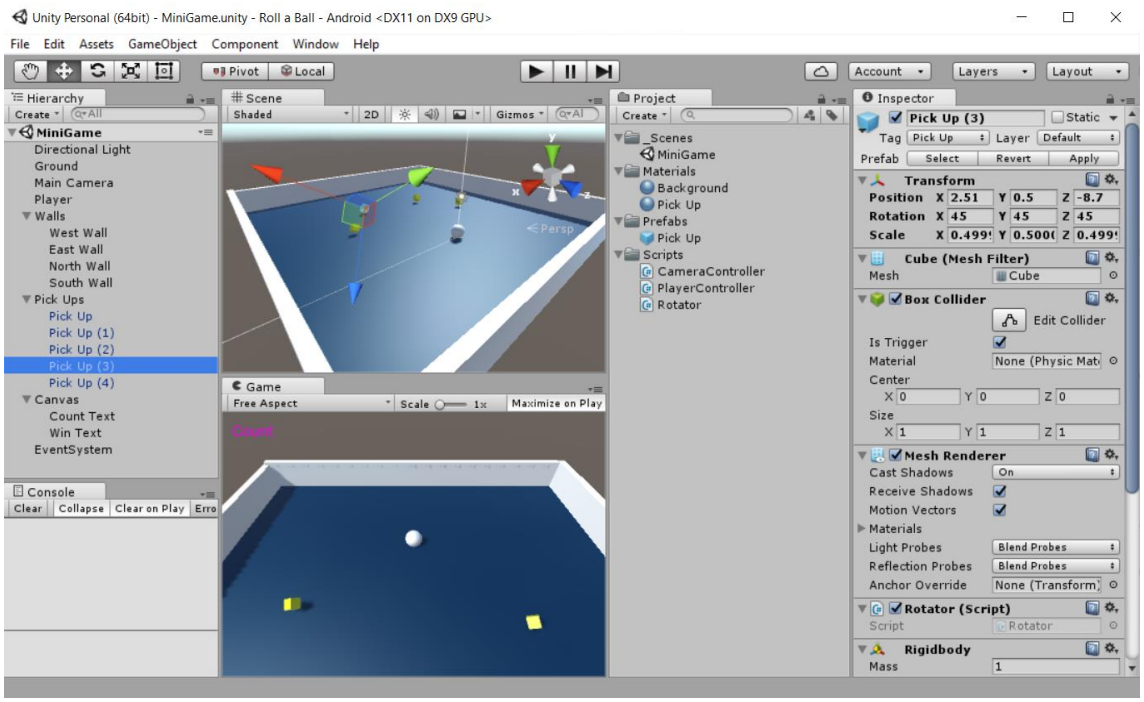

Fig. *4.21* Interfaz de manejo de Unity 3D, con sus distintos elementos

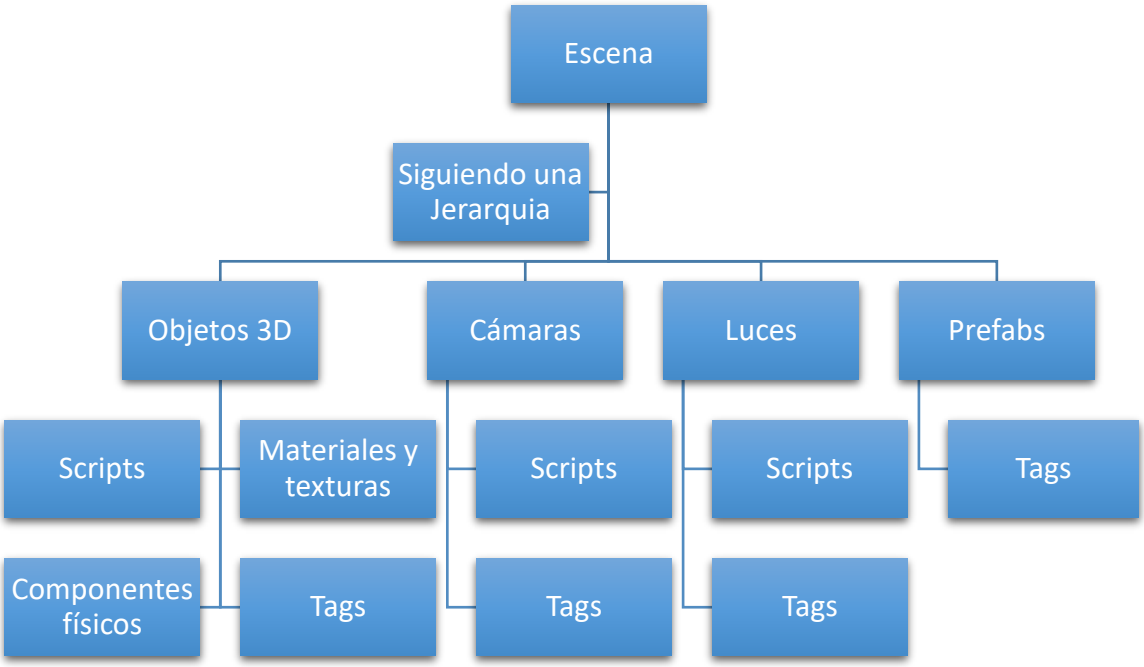

Fig. 4.22 Esquema genérico de un proyecto de Unity (fuente propia)

Empezamos pues creando el proyecto, asignándole un nombre y ruta adecuados donde se guardará.

En el tutorial veremos la mayoría de los elementos principales de un proyecto en Unity tales como las *Scenes*, las *Physics,* los *Rigidbody,* los *Scripts,* los *Objects,* los *Materials*, las *Cameras,* la *Hierarchy,* los *Prefab,* los *Collider* y los *Tag.*

Scenes: escenarios tridimensionales donde situar la experiencia

Dentro de cada proyecto puedes tener lo que Unity llama ***Scenes***, que son distintos escenarios creados para estructurar tu proyecto. Para el caso de videojuegos cada escena podría ser un nivel de dicho juego o, por ejemplo, que al cruzar una puerta cargue el nuevo escenario. Esto tiene como principal ventaja que la máquina no ha de procesar todo a la vez sino únicamente lo que hay en dicha escena. De este modo ganamos en fluidez. En proyectos relativamente grandes es imprescindible.

Para el caso de la bola, al ser un proyecto muy simple no tiene sentido tener más que la *Scene* principal así que la guardamos con el nombre que queramos dentro de una carpeta que llamamos *Scenes*, creada dentro de la carpeta del proyecto.

Object: objetos virtuales con los que interaccionar

Para crear la escena, en primer lugar insertamos un ***Object*** del tipo plano cuadrado sobre el que se moverá la bola, de este modo definimos el dominio o espacio de la aplicación. Para ello dentro de la pestaña *Hierarchy* pulsamos

Create – 3D Object – Plane. Lo redimensionamos y lo colocamos en la posición (0, 0, 0), teniendo en cuenta que los ejes son (X, Y, Z).

A continuación creamos un objeto esfera, el que será el "actor" de nuestra aplicación. En el mismo menú del plano seleccionamos *Sphere* y la colocamos en la posición (0 ,0.5, 0), ya que las esferas creadas en Unity tienen por defecto un diámetro de 1 unidad. De este modo nos aseguramos que la esfera está perfectamente posada sobre el plano con Y=0.5.

Materials: propiedades visuales de los objetos

Para cambiar de color tanto a la bola como al plano utiliza lo que llaman ***Materials***, puedes crear materiales con distintos colores y propiedades y luego asignar cada material a un objeto creado arrastrando dicho material al objeto en la ventana de la escena. De esta manera puedes tener distintas configuraciones predefinidas como podría ser un rojo metalizado, un azul semitransparente, etc.

Physics: leyes físicas, reales o ficticias, para la experiencia virtual

Para el movimiento de la bola tenemos que ajustar previamente las ***Physics*** que se le aplican. Unity emplea un motor de física que simula la gravedad, las fuerzas ejercidas por y hacia un objeto, etc. De este modo se puede simular de forma realista el comportamiento en movimiento de un objeto. Para que a un objeto se le apliquen dichas físicas es necesario emplear un componente al que llama ***Rigidbody***, el cual hay que añadir dentro del objeto y modificar las propiedades por defecto si es necesario.

Scripts: código para definir el comportamiento de los objetos

Después necesitamos un ***Script*** que defina como podemos mover el objeto, para ello creamos una carpeta *Scripts* dentro del proyecto y un *Script* llamado *PlayerController*, que añadiremos a los componentes del objeto bola. Básicamente el *Script* hace que cuando pulsemos sobre las teclas de izquierda, derecha, arriba o abajo el *Rigidbody* asignado a la bola se mueve a un lado u otro. El eje asignado hacia arriba, (en la imagen inferior el Y), tiene que ser siempre 0 en el vector con los 3 ejes que le pasamos a la función del movimiento, ya que solo nos interesa que el objeto se mueva sobre el plano, no hacia arriba y debajo de él ya que nos interesa simular un comportamiento realista de una bola.

Camera: punto de vista de la experiencia

Lo siguiente que hay que tener en cuenta es disponer de una ***Camera***, que nos muestre uno o varios puntos de vista de la acción. Para hacerlo más inmersivo, nos interesa que se mueva a la vez que movemos la bola pero también podríamos tener una cámara fija o distintas cámaras colocadas por todo el tablero que fuesen cambiando de una a otra según la posición de la bola, por ejemplo.

Hierarchy: modo en que se estructuran las relaciones entre los componentes

Los componentes de un proyecto están estructurados en base a jerarquías, (***Hierarchy)***, si un elemento está estructurado dentro de otro, de modo que lo ves al desplegar el elemento superior, decimos que es hijo del superior, por ejemplo en la Fig. *4.21* vemos como *West Wall, North Wall, East Wall y South Wall* son todos hijos de *Walls.* A su vez *Walls* es padre de los 4.

Si arrastramos la cámara principal, *Main Camera* dentro de *Object*, establecemos que *Main Camera* es hijo de *Player*, del mismo modo que *Player* es padre de *Main Camera*. El problema de esta configuración es que, al ser el objeto a seguir una bola, se mueve en sus 3 ejes al rodar y la cámara con ella de forma solidaria, de modo que es inviable.

Para el caso de un objeto que simule el movimiento de una persona no habría problema, ya que se movería en sus ejes X, Y y, al subir o bajar, en el eje Z, pero al no estar dando vueltas funcionaría correctamente.

Descartado el asignar la cámara como hijo tenemos la alternativa de crear un *Script* y asignarlo a la cámara.

Dicho *Script* se basa en calcular constantemente la diferencia en la posición en los ejes (X, Y) de la bola con la cámara y sumarle o restarle según sea necesario a la posición de la cámara. De este modo conseguimos que la cámara no rote en el eje Z sobre sí misma.

A continuación creamos un objeto para recolectar con la bola, del tipo *Cube*, lo llamamos *Pickup* y creamos y le asignamos un *Script* que hace que continuamente vaya dando vueltas, para hacerlo más atractivo.

Prefab: paquetes de componentes

Para no tener que crear uno por uno cada cubo con sus dimensiones y propiedades comunes, creamos un ***Prefab***. Un *Prefab* es un objeto creado como muestra del cual se pueden sacar copias. Una de las ventajas de usar *Prefabs* es que si modificamos una propiedad del *Prefab*, se modificará también en todos los objetos que hayamos creado usándolo como base.

Colocamos unos cuantos objetos *Pickup* sobre el tablero.

Para recolectar los objetos tenemos que ser capaces de detectar las colisiones de la bola con dichos objetos y configurar las consecuencias que conlleva. En nuestro caso nos interesa que desaparezcan al colisionar y que aumente un contador para mostrar la puntuación.

Cada objeto del proyecto, incluido tablero, paredes, bola y coleccionables puede tener un componente ***Collider***, el cuál es necesario para poder detectar colisiones. Si el suelo no tiene el *Collider* activo, por ejemplo, la bola caería al vacío nada más iniciar el juego.

De este modo en el *Script* que asignamos a la bola tiene que comprobar que al colisionar con los coleccionables, y sólo con ellos, tienen que desaparecer.

Una manera de diferenciar los objetos es con los ***Tags***. Como su nombre indica son etiquetas, que podemos crear y asignar a un objeto. En cuanto la bola haga contacto con un objeto coleccionable comparará el *Tag* del objeto con una cadena de texto concreta y en el caso que sean iguales lo desactivará ya que lo habremos recolectado.

Para llevar la cuenta, agregamos una variable de tipo *int* dentro del controlador del jugador, la cual incrementa su valor por cada objeto que recolectemos.

Por último creamos un cuadro de texto, ligado a un componente de texto en el controlador del jugador y modificamos el *Script* para que cuando se hayan recogido todos los coleccionables salga un mensaje de fin de partida.

Capitulo 5. Diseño del demostrador

En este capítulo vamos a trabajar sobre diferentes prototipos para evaluar cuáles son los más adecuados para realizar una demostración útil de la realidad virtual en la exploración de escenarios.

Para ello vamos a discutir sobre los diferentes casos de uso que podamos realizar y los requisitos que estos plantean.

Para acabar pensaremos en una solución viable sobre el diseño de estos y los retos que se nos van a plantear y cómo los podremos resolver.

5.1. Casos de uso

Nos decidimos a diseñar una aplicación que contase con funcionalidades que ayudaran a vivir una experiencia de RV más completa. Esto nos hizo plantearnos qué ingredientes nos harían falta. Por supuesto es necesario contar con el modelo 3D de un escenario así como el hardware necesario para interactuar con la experiencia de RV.

La primera idea que tuvimos relacionada con la recreación de escenarios fue conseguir, o desarrollar nosotros, el modelo virtual de algún edificio singular de Barcelona. Pensamos que sería interesante poder explorar de forma remota El Liceu o el Palau de la Música y, siendo aún más ambiciosos, poder visualizar obras o conciertos en dicho modelo.

Pronto nos dimos cuenta que, con los conocimientos y experiencia de los que disponemos, únicamente la creación de uno de los modelos nos llevaría demasiado tiempo y no teníamos garantías que el resultado final fuese a ser siquiera decente.

La posibilidad de que nos cediesen el modelo 3D “oficial”, en caso que exista, también era bastante remota.

Por suerte, a finales de verano nuestro tutor, en colaboración con CIMNE, nos proporcionó el modelo 3D del edificio B0 del Campus Nord de la UPC, en Barcelona.

Contando con dicho modelo nos decidimos a probar lo siguiente:

- Pasearnos por el edificio y sus alrededores en primera persona, de modo que podamos recorrer el edificio como lo haríamos de forma real.
- Poder visualizar vídeos en una pantalla colocada en una de las paredes, con sonido 3D.
- Mostrar las instalaciones que hay dentro de las paredes cuando nos acerquemos, (agua, electricidad, gas…).

- Etiquetas de posición, para saber en todo momento en qué parte del edificio, o exteriores, nos encontramos.
- Iluminación realista, para comprobar cómo afecta el sol a distintas horas del día en cada una de las partes del edificio.
- Simulación de una ventana en el modelo virtual cargando vídeo proveniente de una cámara colocada en la ventana real del edificio.
- Empleo de la herramienta IndoorAtlas, un servicio de localización para interiores que utiliza tecnología geomagnética para capturar la localización dentro de edificios previamente mapeados. De este modo, si mapeamos el edificio B0 real, podemos movernos por él y actualizar la posición en el modelo virtual a tiempo real.

Uno de los usos más interesantes a nivel comercial que presenta la realidad virtual y para el que nos puede ser de utilidad el modelo del edificio B0, es como herramienta comercial y promocional para las inmobiliarias.

Para ello, al modelo del edificio B0 se le pueden añadir las siguientes características:

- Añadir mobiliario al cual se le pueda modificar el color, la textura, la posición, etc. De este modo un comprador puede ver cómo quedaría su casa decorada “a su gusto”.
- Tener un recorrido previamente definido por el que se mueva el personaje de forma autónoma al iniciar la aplicación. De este modo se puede mostrar la casa de la forma en que quiere la inmobiliaria, haciendo énfasis en las zonas que más le interese mostrar e incluso añadiendo audio o texto a la presentación.

5.2. Dispositivos con los que contamos

El dispositivo imprescindible para trabajar la aplicación es un ordenador personal, preferentemente tipo *gaming*, que son aquellos con mayor capacidad de computación gráfica, y con potencia suficiente para implementar y ejecutar tal desarrollo con solvencia. Para ello disponemos de los portátiles de ambos autores.

Cuentan con las siguientes características:

Asus GL752VW:

- Procesador Intel Core i7-6700HQ, 4 núcleos desde 2,60 GHz hasta 3,50 GHz
- Nvidia Geforce GTX960M
- 16 GB de memoria RAM DDR4
- HDD de 1TB a 7200rpm
- Pantalla de 17.3” con resolución Full HD

MSI GE60 2PC Apache:

- Procesador Intel Core i7-4720HQ, 4 núcleos desde 2,60 GHz hasta 3,60 GHz
- Nvidia Geforce GTX850M
- 8 GB de memoria RAM DDR3
- SSD de 250 GB + HDD de 1TB a 7200rpm
- Pantalla de 15.6” con resolución Full HD

Para la visualización 3D de imágenes estereoscópicas contamos con 2 dispositivos móviles distintos con las características necesarias para una experiencia satisfactoria. El primero es más potente y actual, pero el segundo tiene mayor resolución de pantalla, lo cual es muy interesante al visualizar la pantalla a escasos centímetros de los ojos:

Huawei Mate 8:

- Procesador Kirin 950, 8 núcleos, 4 a 2,3GHz y 4 a 1,8GHz.
- GPU Arm Mali-T880 MP4
- 3 GB de memoria RAM
- 32 GB de memoria interna
- Pantalla de 6” con resolución Full HD

LG G3:

- Procesador Qualcomm Snapdragon 801, 4 núcleos a 2,5GHz
- GPU Qualcomm Adreno 330
- 2 GB de memoria RAM
- 16 GB de memoria interna
- Pantalla de 5.5” con resolución Quad HD (2560 x 1440px)

Estos móviles irán acoplados al dispositivo HMD de RV del que disponemos:

VR Box (tipo Cardboard).

En cuanto al controlador, disponemos de tres mandos:

Ipega PG-9025.
WiiMote.
Dualshock 3 Sixaxis.

5.3. Requisitos técnicos de la RV

Para una óptima experiencia de RV hay ciertos requisitos que nuestra aplicación deberá cumplir. Antes es necesario introducir ciertos términos técnicos:

- Fotogramas por segundo (FPS): Imágenes que se muestran por cada segundo. También conocido como tasa de refresco.
- Resolución de la pantalla: Hace referencia a la cantidad de píxeles diferentes que contiene la pantalla, expresado de la siguiente forma: número de píxeles en horizontal x número de píxeles en vertical. Ejemplo: 1920x1080 pixeles.
- Latencia: Es el tiempo que transcurre entre que nosotros pulsamos una tecla del controlador y la acción tiene lugar en la aplicación.

Según Intel [37], es necesario que se cumpla lo siguiente:

- Para percibir el movimiento totalmente natural y no como una sucesión de imágenes, se requiere una tasa de refresco de, por lo menos 90 FPS. A mayor tasa de refresco, menos posibilidades hay que el usuario padezca mareos.
- Una aplicación convencional se considera de alta definición real si cuenta con una resolución de 1920x1080 píxeles. Para RV, como hay que proyectar una imagen para cada ojo y además la pantalla acostumbra a estar mucho más cerca de los ojos, es necesario contar con una resolución mayor, en concreto no menor a 2160x1200 píxeles.
- Debido a los 2 requisitos anteriores, tanto el dispositivo de desarrollo como el de ejecución deben contar con una capacidad de procesamiento superior al que necesitaríamos para una aplicación convencional.
- Para dar una sensación de inmersión completa, es necesario que la latencia sea muy baja, del orden de 7 a 15 milisegundos. [38] Hemos de tener en cuenta que la latencia desde que, por ejemplo y en condiciones normales, mueves el ratón del ordenador y el cursor realmente se mueve es de unos 50 milisegundos, la RV tiene en este caso, una restricción muy fuerte.

Capitulo 6. Implementación

Fig. 6.1 Esquema de nuestra implementación

6.1. Importación del modelo a Unity

Lo primero que necesitamos en cualquier motor de videojuegos como Unity3D es un escenario donde poder realizar pruebas. Para pruebas más sencillas Unity incorpora una serie de objetos 3D y diversos *Assets*. Los objetos 3D básicos son, una cubo, una esfera, un cilindro, una capsula y un plano. Y como *Assets* por defecto, entre otros, un personaje en primera persona, otro en tercera persona, algunos materiales y diversos efectos visuales como partículas de fuego o reflejos.

Todas estas facilidades nos pueden ayudar para empezar a usar Unity, pero no para un entorno final, para conseguir objetos más complejos necesitamos alguna herramienta de modelado 3D como 3D Studio Max o Maya. Aunque siempre podemos explorar el *Asset* store de Unity donde hay una gran diversidad de *Assets* interesantes.

En nuestro caso tenemos el edificio B0 del campus Nord en formato .max, referente a 3D studio Max, en el que nos centraremos para explicar cómo exportarlo para poder usarlo en Unity o cualquier otro motor 3D.

6.2. Archivos 3D

Existe gran variedad de archivos 3D, y estos se pueden clasificar en 2 grupos, los archivos exportados y los archivos propietarios [39].

Los archivos propietarios de aplicaciones 3D pertenecen a un programa de modelado 3D, y cada uno tiene el suyo, por ejemplo, el 3D Studio Max guarda ficheros .max. Estos archivos están diseñados para funcionar con el programa que los generó, ya que además de los datos del modelo 3D añaden información adicional que solo el programa original puede leer y entender. Aun y así, Unity puede importar estos ficheros, ya que posee la habilidad de convertir la mayor parte de archivos 3D propietarios a un formato que el sí pueda leer.

Los archivos de intercambio estándar (a exportar) son archivos genéricos en los que se almacena únicamente la información que se va a usar de ese modelo 3D. Estos están diseñados para que cualquier aplicación 3D pueda entenderlos. Los 2 más extendidos son fbx i obj.

Unity puede importar ambos grupos de archivos, con las siguientes ventajas y desventajas:

Tabla 6.1 Comparativa

	Ventajas	Desventajas
Archivos Propietarios	Modificaciones en vivo. Si modificas el original, Unity reimporta el objeto.	Es necesaria una licencia del software originario del archivo.
	Fácil de usar.	Puede haber muchos datos innecesarios.
		Si el fichero es grande ralentiza las actualizaciones de Unity.
		Los problemas de importación son más difíciles de solucionar.
Archivos exportados	Se exportan únicamente los datos que se van a usar.	Es un proceso más costoso.
	Las diferentes capas del modelo 3D se importan como objetos separados.	Hay que volver a importar el objeto cada vez que se modifica el original.
	Se pueden importar modelos de archivos que Unity no pueda entender.	

6.2.1. Problemas con los archivos propietarios

Unity puede convertir automáticamente los archivos propietarios pero en general eso significa problemas. Sobre todo si el modelo 3D está hecho por alguien ajeno al proyecto, como es nuestro caso.

Los principales problemas que puede tener, y que tiene, nuestro modelo 3D son:

- Complementos de terceros: Estos sirven para añadir funcionalidades a los programas de modelado, uno muy usado, inclusive en nuestro modelo es V-Ray, que permite generar y renderizar materiales más detallados e iluminación más realista. Estos complementos no son compatibles con los motores de videojuegos.
- Cámaras: Para renderizar una escena en un programa de modelado 3D es necesario colocar por lo menos una cámara, que será la que indique que parte de la escena se va a renderizar. Estas cámaras Unity las intentara exportar, si son cámaras estándar no habrá problemas, pero si son cámaras V-Ray o referentes a cualquier otro complemento, Unity no lo conseguirá.
- Materiales y texturas: Igual que pasa con las cámaras, si se trata de materiales estándar este se exportara bien, pero si no lo es se exportara un material en blanco.
- Iluminación: Las luces son más problemáticas, y lo mejor es no exportarlas. Unity tiene su propio sistema de iluminación, y si no eliminas las luces intentara importarlas con resultados desastrosos. La luz del sol se convertirá en un foco, normalmente las luces de las bombillas en una luz de área y muchas otras no conseguirá llegar a importarlas.

En conclusión, es mejor las mallas del objeto con los materiales que si se puedan exportar, y todo lo demás, añadirlo desde el propio Unity.

6.2.2. Exportar a FBX

El método menos problemático para trasladar el modelo de 3D Studio Max a Unity, y en general, de cualquier software de modelado 3D a cualquier motor de videojuegos, es exportando a FBX [40], uno de los archivos exportados más polivalente.

El primer paso, después de abrir el archivo .max, es eliminar todas las luces, cámaras y demás objetos que podamos encontrar que no sean estrictamente parte del modelo 3D.

Una vez tengamos el modelo limpio, procederemos a exportarlo en FBX. En el caso de 3D Studio Max, Archivo->Exportar->FBX. [41]

En la nueva ventana se pueden escoger que elementos incluirá este FBX, hay 5 pestañas: Geometría, animación, cámaras, luces y contenidos multimedia. Para

evitar los problemas que se han tratado anteriormente deshabilitamos todas las pestañas menos la de geometría.

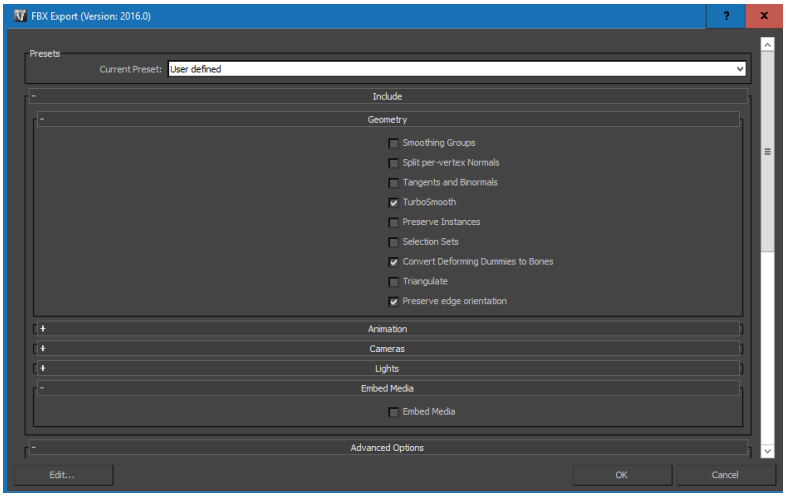

Fig. *6.2* Menu exportar a FBX

Una vez exportado el modelo ya está listo para importarlo a Unity sin problemas. Para importarlo basta con arrastrar el fichero FBX a la pestaña de proyecto de Unity.

Cuando el proceso finaliza en la vista del inspector nos aparecen varias opciones, la más significativa en nuestro caso es la de “generate *Colliders*”, esta opción crea un *mesh Collider* para cada uno de los objetos de la escena, que en definitiva, es un *Collider* con la forma exacta del objeto. Esto nos evita mucho trabajo, ya que si no deberíamos ponerlos manualmente por toda la escena. Pero tiene algunas desventajas:

- La primera es que nos generara *Colliders* en todos los objetos, incluidos aquellos donde no los necesitamos para nada, por ejemplo, no tiene ningún sentido poner un *Collider* a una bombilla o a alguna zona a la que no tengamos previsto acceder desde dentro del juego.
- La segunda es que los *mesh Colliders* pueden llegar a consumir muchos recursos si se trata de una forma muy compleja, además de que normalmente no hace falta incluir esa complejidad al *Collider*. Por ejemplo, una rejilla en el suelo, en general, no hará falta que los objetos puedas caer a través de las rendijas, y poner de *Collider* un cubo reducirá considerablemente los recursos que se asignaran a esa parte del escenario.
- Por último, hay *Colliders* que pueden provocarnos problemas inesperados, como por ejemplo los de una escalera. Si nuestro personaje se encuentra de cara con una escalera lo más probable es que colisione con el primer escalón y no la suba, para ello es más adecuado poner un *Collider* plano en diagonal que suba por la escalera, ya que es más sencillo que hacer que los pies del personaje suban realmente las escaleras.

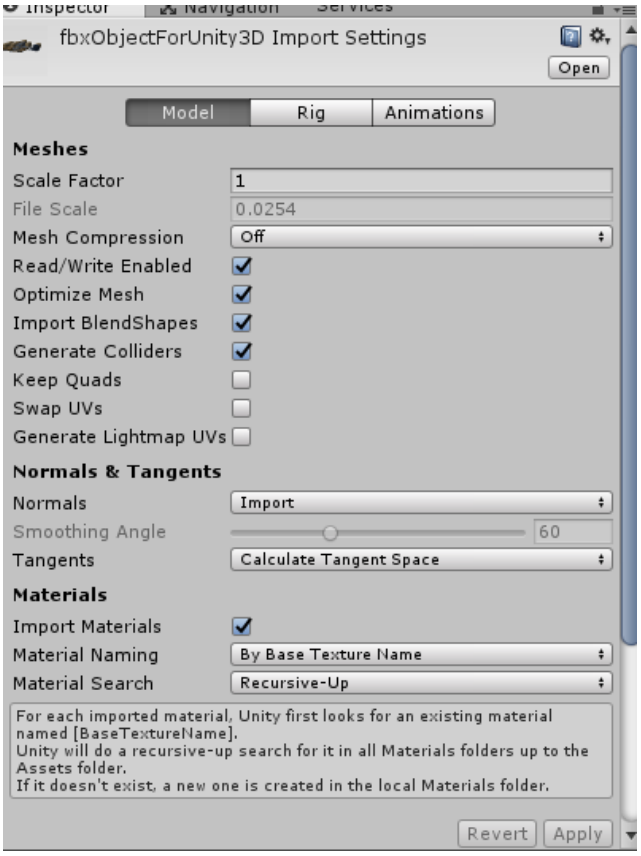

Fig. *6.3* Vista inspector modelo FBX en Unity

Aunque pueda parecer mala idea marcar esta opción por la cantidad de problemas con los que nos podemos encontrar, es recomendable generar todos los *Colliders* desde un principio, quitar aquellos que no queramos y modificar los problemáticos. De esta forma supondrá un menor esfuerzo que generarlos todos manualmente.

6.3. Conexión dispositivos de entrada con unity

6.3.1. WiiMote

Para conectar la *WiiMote* a un PC con Windows hay que seguir los siguientes pasos:

- Acceder al panel de control y abrir el icono de administración de dispositivos, haciendo clic con el botón derecho seleccionar la opción de Agregar Dispositivo.
- Empezar a apretar alternamente los botones 1 y 2 de la *WiiMote* hasta que el asistente de Windows detecte el dispositivo Nintendo RVL-CNT-01 y pulsar siguiente.

- Si nos da la opción de emparejar sin código de seguridad, continuar, sino escribir 0000, borrarlos y podremos proceder sin necesidad de facilitar ninguna clave.
- Esperar que el asistente instale los controladores necesarios y ya estará emparejado satisfactoriamente.

Una vez el mando está conectado al PC hay que conseguir que este se comunique con Unity. La *WiiMote* no es un controlador HID normal como un *gamepad* o un teclado, por lo que Windows no va a mapear ninguna de las funcionalidades del mando, para poder usarlo hay que emular el protocolo L2CAP desde Unity e interactuar directamente con el controlador.

Para ello ya existen algunas librerías preparadas para Unity. [42] Solo hay que importar el *package* e incluir las referencias en nuestros *Scripts*.

6.3.2. Dualshock 3/4

El *Dualshock* 3, aun usando Bluetooth, no se puede emparejar directamente con Windows. Para poder usarlo es necesario cumplir una serie de requisitos [43] previos, el más importante, es tener un adaptador Bluetooth 2.0 con EDR para los controladores más antiguos o Bluetooth 2.1 para los más actuales. Aunque este también se puede usar vía USB.

El método que usaremos para conectar el controlador de PS3 al PC consiste en usar la aplicación *SCPToolkit* [44] para emular el mando de PS3 como un controlador de XBOX360, que está completamente soportado desde Windows Vista. Y que además este lo mapea como un *gamepad* standard para luego poder asignar los diferentes botones en Unity.

Los pasos para conectar el mando son los siguientes:

- Habilitar el Bluetooth en caso de que este integrado en la maquina o enchufar el adaptador.
- Descargar la última versión de *ScpToolkit*
- Ejecutar el fichero *setup* e instalarlo.
- Pulsar el botón que aparece en pantalla: “Driver Installer”.
- En Windows vista marcar la opción de “Force install”, en versiones más modernas dejarlo desmarcado.
- La herramienta ya estará instalada y el controlador está siendo mapeado como un controlador de Xbox, lo que equivale a un *gamepad* estándar.

El Dualshock 4, a diferencia del 3, es *plug’n’play* en plataformas Windows a partir de Windows Vista. Se puede emparejar fácilmente mediante una conexión USB o mediante un adaptador Bluetooth estándar.

Ambos dispositivos, una vez correctamente emparejados, son inmediatamente reconocidos por Unity y ya se pueden mapear todos sus botones en el menú de *InputManager*.

Ambos dispositivos, mediante la conexión Bluetooth usan nuevamente el protocolo L2CAP, que como ya hemos comentado no es compatible con dispositivos Android modernos. Tampoco son compatible mediante la conexión USB en la mayoría de dispositivos Android por la falta de compatibilidad entre estos dispositivos y los *drivers* estándar que proporciona el *kernel* de Android.

6.3.3. Gamepad Android

Nosotros disponemos del modelo “Ipega pg-9025”, y en este apartado explicaremos como conectar este mediante su única posibilidad Bluetooth. De todos modos, todos los controladores de esta índole preparados para Android u otros dispositivos móviles funcionan exactamente de la misma forma, con la única diferencia de que existen modelos también compatibles mediante una conexión USB.

Cada modelo tiene sus propias instrucciones para ser emparejado con un anfitrión Bluetooth, en el caso concreto de nuestro mando solo hay que mantener pulsadas a la vez los botones *HOME* y *X* hasta que el móvil o PC detecte el dispositivo. Una vez detectado, para emparejar solo hay que pulsar sobre el en la lista del anfitrión, ya que la conexión no es segura ni siquiera nos pedirá un PIN para concluir el proceso de emparejamiento.

Una vez emparejado, nuevamente Unity ya tendrá control sobre el controlador, de forma que ya podremos empezar a mapear los botones hacia las funciones concretas de nuestra aplicación que queramos ejecutar.

6.3.4. Mapeando controladores con Unity

Una vez nuestro PC reconoce el controlador que queramos usar hay que indicarle a Unity que queremos que haga cada uno de los botones.

Para que no tengamos que configurar de diferente forma cada controlador según el dispositivo donde queramos usarlo Unity dispone de una característica llamada *InputManager*. Esta característica nos permite asignar cada botón a una constante que nosotros hemos definido previamente. Este proceso se conoce como mapeo del controlador.

Para ilustrarlo con un ejemplo nos podemos fijar en la Fig. *6.4*.

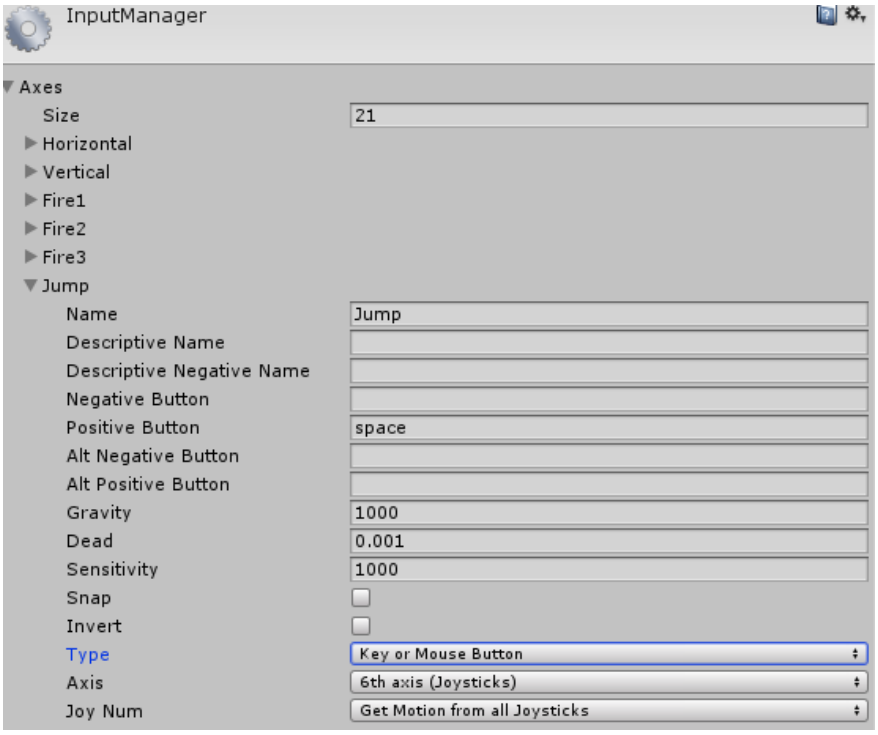

Fig. *6.4* Ejemplo de uso de la caracterisitca *InputManager*

Este menú está situado en “Edit -> Project Settings -> Input”. Este menú nos muestra de forma visual una matriz con diferentes valores. En concreto nos podemos fijar en el elemento *Jump*, En el campo *Name* podemos escribir un nombre, este servirá para poder referenciarnos a este elemento de la matriz más adelante en nuestros *Scripts*. También podemos añadirle una pequeña descripción a modo de documentación.

Por defecto se le puede asignar un botón del teclado o ratón en el campo *Possitive Button*. También cabe la posibilidad de invertir el efecto, en el caso de un *joystick* que subir sea bajar y al revés, y en el caso de un botón que pulsarlo sea en realidad dejar de pulsarlo, o al revés.

Y el campo más importante para nuestro caso es el de *Axis*, este tiene un desplegable con hasta 28 posibles opciones, que hacen referencia a cada uno de los botones del mando.

El problema que presenta este campo es que dependiendo de que controlador sea y de cómo lo reconozca el PC la asignación de botones puede ser diferente. Por ejemplo en la Fig. *6.5*, tomando como referencia la Fig. *6.4*, el botón que hemos asignado al elemento *Jump*, en el controlador de arriba se accionaria pulsando la flecha arriba y en el de abajo pulsando el gatillo trasero izquierdo.

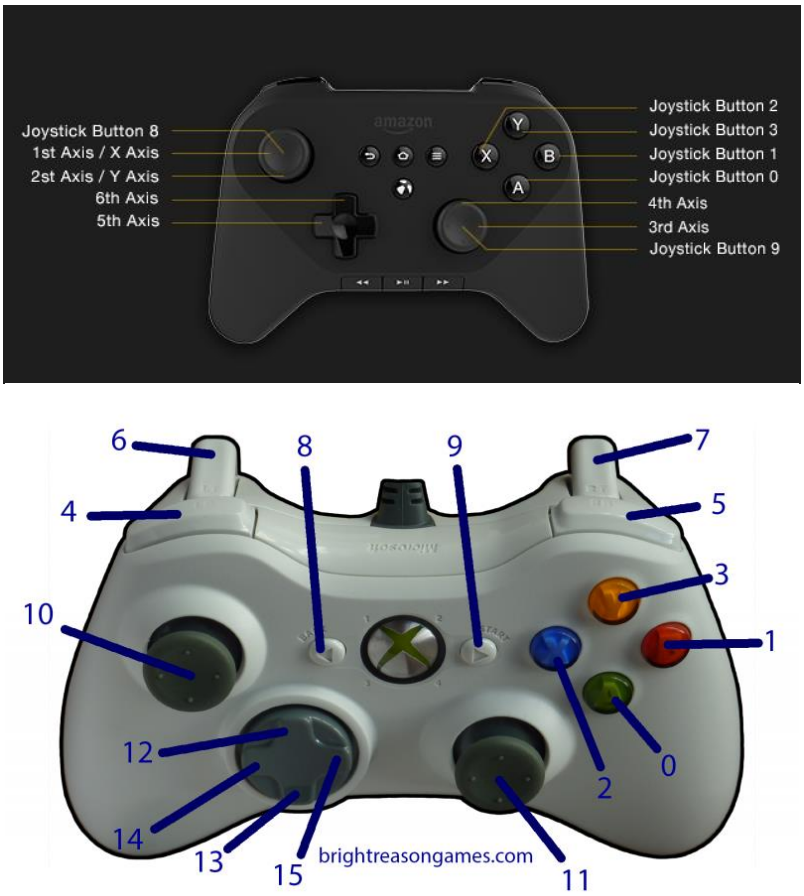

Fig. *6.5* Diferentes numeraciones de botones sobre diferentes tipos de controladores.

Para evitar confusiones es necesario disponer de una configuración específica para cada mando que vayamos a usar ya que nadie pretendería saltar con un gatillo del mando.

Una vez mapeado todo el controlador, o solo aquellos botones que vayamos a usar, para usarlo en nuestro código usaremos la clase "CrossPlatformInputManager", esta contiene la función "GetButtonDown('Jump')". Esta función devuelve *TRUE*, si estamos pulsando, en nuestro caso, el botón 6 del mando o *FALSE* en cualquier otro caso.

Con esta función nos aseguramos que sea cual sea el dispositivo que este ejecutando la aplicación, al hacer referencia al elemento *Jump*, entenderá que es el botón que hemos asignado al mando en concreto. Es decir, solo existe la necesidad de mapear los botones del mando una sola vez en un solo dispositivo, Unity se encarga por nosotros de hacer el resto.

6.3.5. Conclusión

El protocolo L2CAP y Android

Hasta ahora hemos comentado en varias ocasiones que algunos dispositivos no son compatibles con los dispositivos Android por usar el protocolo L2CAP. Pero ¿Por qué? ¿Qué es L2CAP?

L2CAP es una capa de *software* que se interpone entre nuestra aplicación y lo que es propiamente el enlace lógico entre los dispositivos Bluetooth. Es el encargado de mediar entre nuestra aplicación y el dispositivo Bluetooth para que se entiendan correctamente.

Existen otros muchos protocolos equivalentes al L2CAP como ahora RFCOMM [45].

L2CAP nunca ha sido soportado oficialmente por Android, aunque las versiones antiguas hasta Android 4.2 (Jelly Bean) lo tenían instalado en el *Kernel*, ya que usaban las librerías estándar de Linux, y estas si lo incorporan. Por lo que los modelos más antiguos de móviles Android si pueden usar los dispositivos como la WiiMote, pero estos no solían incorporar sensores como el giroscopio, además tenían muy poca potencia para aplicaciones de RV.

A partir de Android 4.4 estas librerías para el uso de Bluetooth se cambiaron por las librerías de Qualcomm, y estas nuevas librerías no incorporan el protocolo L2CAP. [46]

Aun y así, eso no significa que los dispositivos modernos no puedan implementar esa capa, al fin y al cabo es una capa de *software*, no hay ningún tipo de limitación de *hardware* para poder implementarlo.

Existen unas librerías creadas por terceros compatibles con Android [47], estas vuelven a implementar toda la pila de protocolos Bluetooth para que sean más compatibles.

El problema es que para poder instalar estas librerías hay que hacer ciertas modificaciones al dispositivo Android que están fuera del alcance del usuario medio, y que en la inmensa mayoría de los casos llevan a perder la garantía de este.

En concreto hay que conseguir acceso de *superusuario* (*root*) [48] e instalar lo que se conoce como *Custom Recovery* [49]. Pero este tema se escapa por completo a las metas de nuestro proyecto, así que no vamos a desarrollar el proceso para realizar dichos cambios.

Fig. *6.6* Capas del protocolo Bluetooth usando L2CAP (fuente propia)

Dispositivos elegidos

Después de haber probado todos los dispositivos que teníamos a nuestro alcance, nos hemos centrado en aquellos que han sido de mayor utilidad y muestran una compatibilidad completa con nuestros requisitos.

El uso de la WiiMote o de los dispositivos Dualshock nos es imposible dado que no son compatibles con dispositivos Android. El uso del teclado y el ratón no nos ha parecido el adecuado para una aplicación de RV, creemos que es más cómodo usar un *gamepad*.

Como conclusión hemos escogido el “Ipega pg-9025” para controlar el movimiento y acciones del personaje.

En cuanto a los dispositivos de visualización para RV, las OculusRift y el HTC Vive nos han parecido de gran utilidad para poder mostrar contenido en alta definición, pero al final nos hemos decidido por el VR-BOX dado que ya disponíamos de uno y así poder aprovechar las ventajas inherentes que ya hemos comentado de los dispositivos Android.

Tabla 6.2 Comparativa de dispositivos de entrada

Dispositivos	Ventajas	Desventajas
WiiMote	Sensor de movimiento, giroscopio y acelerómetro.	Incompatible con dispositivos móviles (Android/iOS) Algoritmos complejos para alta precisión
Mando PS3	Giroscopio y acelerómetro.	Incompatible con dispositivos móviles (Android/iOS)
Ipega-9025 (Gamepad Android)	Compatible con dispositivos móviles	Ausencia de sensores
Teclado/Ratón	Fácil de manejar	Poco manejable para la realidad virtual

6.4. Google VR

Para poder generar aplicaciones Unity3D de RV para sistemas Android o iOS hemos escogido usar la SDK de Google VR por la accesibilidad de las Google CardBoard.

Para empezar debemos dirigirnos al recurso de Google y descargarlo en: https://developers.google.com/vr/unity/ en la parte “Download the Google VR SDK for Unity”.

Aquí podemos usar *git* para descargar el repositorio “gvr-unity-sdk” o sencillamente acceder al enlace que nos proporcionan.

Una vez obtenemos el repositorio (y lo descomprimimos en caso de haber usado el enlace) deberíamos tener algo parecido a lo de la imagen:

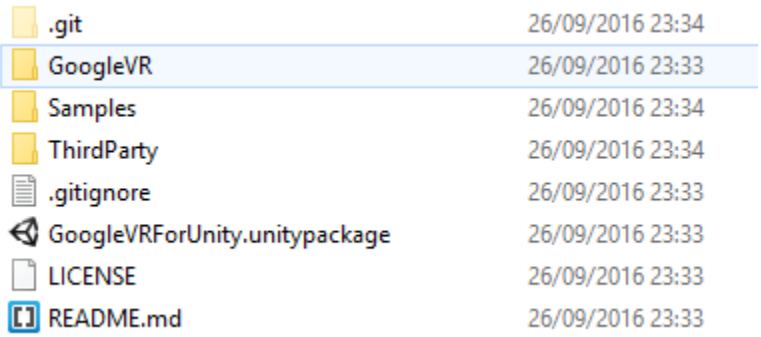

Fig. *6.7* Directorio de *Assets* incorporando GoogleVR

Ahora vamos a abrir el Unity con el proyecto al que queramos dotar de realidad virtual, o para probar, uno en blanco.

A continuación buscamos la pestaña *Project* y en la carpeta de *Assets* hacemos clic derecho, accedemos al desplegable de *Import Package* y seleccionamos la opción *Custom Package.*

Ahora navegamos hasta la carpeta anterior y abrimos el fichero "GoogleVRForUnity. unitypackage".

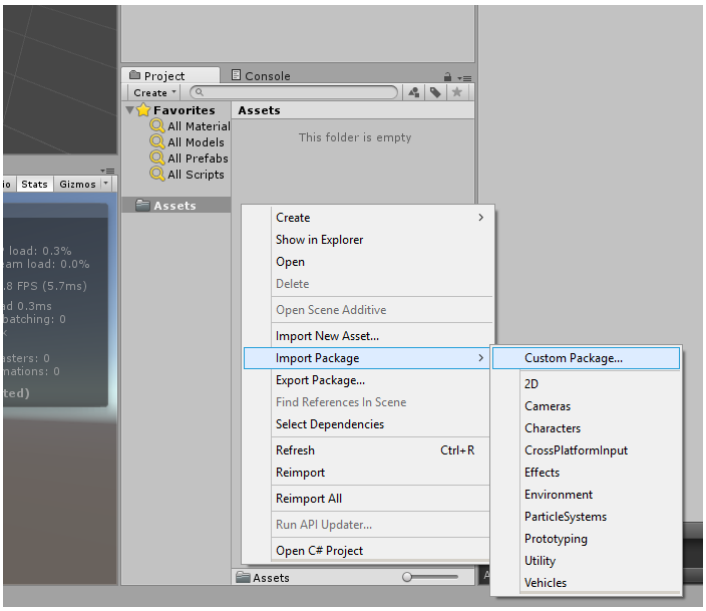

Fig. *6.8* Menú para añadir un *Custom Package*

Se abrirá una ventana nueva para que confirmar que queremos importar, nos aseguramos de tener todas las casillas marcadas e importamos. Es posible que nos advierta de que necesita otras librerías, insistimos en "Import Package", nos volverá a mostrar una ventana como la anterior, volvemos a importarlo todo.

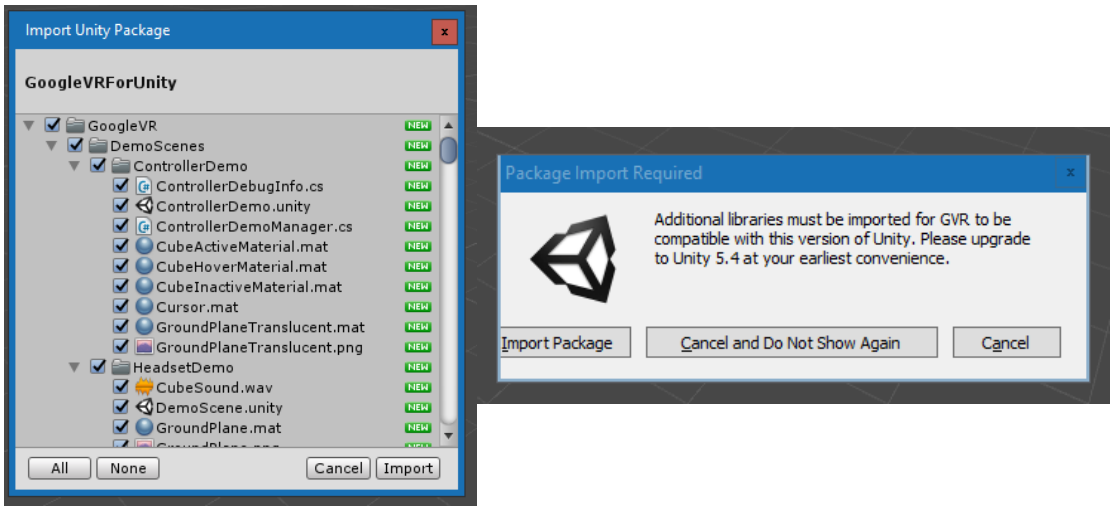

Fig. *6.9* Ventana de confirmación para realizar el proceso de importación

Si lo hemos hecho bien podremos observar un nuevo menu en la parte superior izquierda del programa, “GoogleVR”. Ahora ya estamos listos para crear nuestra primera aplicación de realidad virtual.

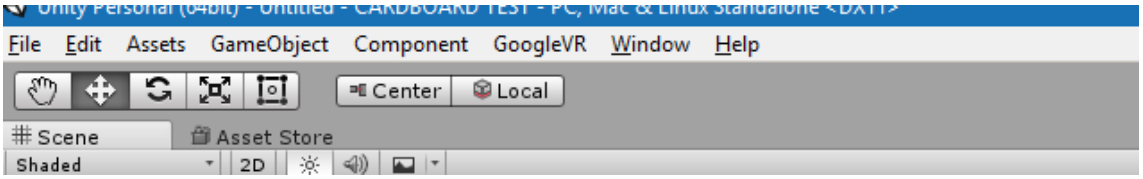

Fig. *6.10* Confirmación de instalación de GoogleVR

El paquete que acabamos de importar contiene una escena de demostración. En la carpeta de *Assets* ahora han aparecido dos más: “Plugins” y “Google VR”.

Ahora vamos a la siguiente ruta: Google VR → DemoScenes →HeadsetDemo, en esta carpeta hay una escena llamada “DemoScene”, la abrimos. Deberíamos ver algo parecido a lo siguiente:

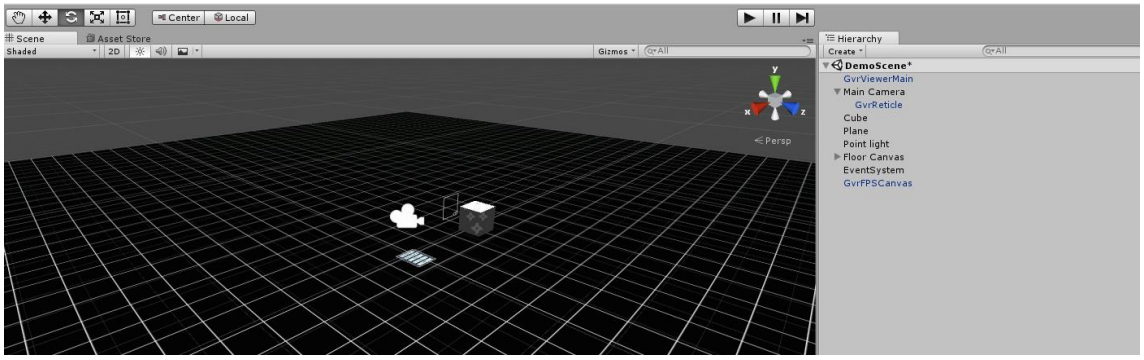

Fig. *6.11* Escena de demostración con GoogleVR

Lo que nos interesa ahora está en la pestaña de *Hierarchy*, concretamente el *Prefab* llamado “GvrViewerMain”, este es el que dotara a nuestra cámara de una visión estereoscópica, una cámara para cada ojo, además de aplicar la rotación correspondiente a los datos que reciba del giróscopo. Para verlo en acción vamos a darle al botón de Play.

Lo primero en lo que nos vamos a fijar es en que vemos la imagen dos veces, con las CardBoard esto da una sensación de inmersión muy buena. Pero vamos a mirar la pestaña de *Hierarchy*, al darle al *play* el *prefab* “GvrViewerMain” ha creado dos cámaras nuevas dentro del *Main Camera*, y si lo probamos desde un móvil con giróscopo podremos comprobar que realmente la vista gira acorde con el móvil. Este es el *prefab* que deberemos incluir en nuestros futuros proyectos para incluir esta característica.

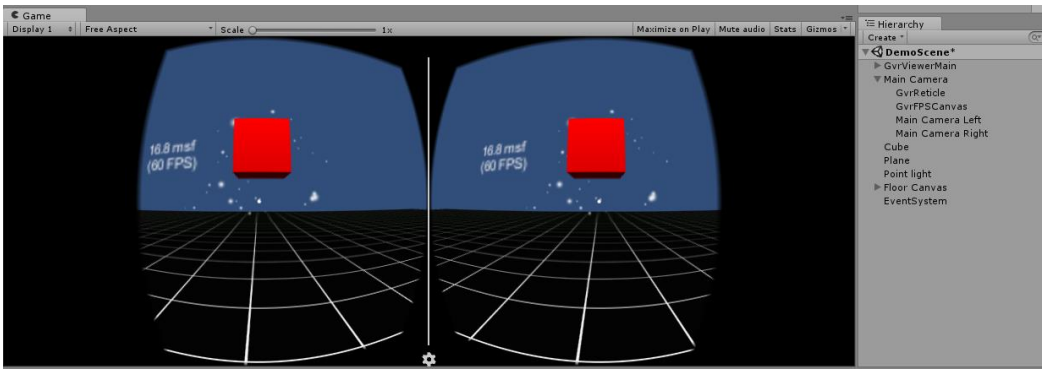

Fig. *6.12* Imagen estereoscópica generada por GoogleVR

6.5. Interacción con el modelo

Este apartado consta de dos partes para solucionar un mismo problema, ¿cómo vamos a interactuar con el modelo?

Enfocando el proyecto en el mundo de la realidad virtual, lo primero que tenemos que solucionar es como movernos por el modelo simulando una vista en primera persona. Por suerte Unity nos provee de los *Assets* necesarios para empezar.

El segundo problema con el que nos encontramos radica en cómo controlaremos el personaje. La opción más común con Unity es usar el teclado y el ratón, pero en el mundo de la realidad virtual estos controles no son cómodos. En la segunda parte de este apartado se discuten las diferentes maneras con las que podemos controlar a nuestro personaje.

6.5.1. Vista en primera persona

Uno de los aspectos clave de la RV es contar con una vista en primera persona, ya que te permite ponerte en la piel del personaje generando la ilusión de que tú eres él.

Una vez hemos conseguido importar el modelo de nuestro edificio a Unity ya tenemos el proyecto preparado para dar el siguiente paso, movernos a través de él. Unity incluye de forma gratuita unos cuantos *Assets* para ahorrarnos parte de este trabajo.

Para empezar hay que importar el paquete “Characters”, cuando el proceso finalice aparecerá una carpeta llamada *Characters* dentro de *Stadard Assets* en la vista de proyecto. Esta carpeta contiene los *Scripts* necesarios tanto para un controlador en primera persona como para uno en tercera, además de un modelo animado de una persona y unos *prefabs* listos para poder usarlos.

Para nuestro proyecto hemos usado el *prefab* “RigidBodyFPSController”, solo hace falta moverlo a la escena y colocarlo donde queramos que empiece el personaje. Este *prefab* contiene 3 componentes, un rigidbody para las físicas, un *Collider* para que no caiga en el infinito y un *Script* para el movimiento.

Con el controlador en la escena ya somos capaces de mover al personaje con las flechas del teclado, rotar la cámara con el ratón e incluso saltar con la barra espaciadora.

6.5.2. Puertas

Una vez ya nos podemos mover libremente por el escenario, el primer reto que se nos presenta es el de atravesar las puertas. Esto facilita la movilidad por el edificio y las posibilidades de una visita virtual.

Existen infinidad de métodos para abrir y cerrar puertas en Unity, nosotros nos hemos decantado por uno sencillo a la vez que eficaz.

Nuestro sistema consta de 3 partes: un detector de distancia, un accionador y una bisagra.

El detector de distancia nos permite mostrar por pantalla las instrucciones que deberá seguir el usuario para poder abrir la puerta, un mensaje como “Pulse F para abrir la puerta”. Además, este detector nos permite asignar esa acción a una puerta en concreto, sino al pulsar “F” se abrirían todas las puertas.

Para implementarlo hay dos métodos que nosotros hemos usado, ambos son válidos, pero el de la imagen es el que nos parece más visual. Este trata de un “box *Collider*” tan grande como la distancia a la queremos que se pueda abrir la puerta, en el momento en el que se detecta una colisión entre el *Collider* del personaje y este, hacemos aparecer el mensaje y activamos el accioandor de esa puerta.

El segundo método consiste en usar el método “Vector3.Distance(A, B)” para determinar la distancia entre los objetos A y B, siendo el personaje y la puerta respectivamente, acotando esta distancia podemos determinar con precisión si estamos a la distancia óptima para poder abrir la puerta y activar el accionador.

El accionador trabaja en conjunto con la bisagra. Cuando el usuario presiona la tecla F estando dentro del radio del detector de una puerta este hace girar la bisagra 90º para abrir la puerta.

Si no hubiera bisagra y aplicáramos la transformación directamente a la puerta esta giraría tomando como eje su centro, para evitar este efecto, colocamos un cubo transparente (sin *mesh renderer*) posicionando su eje central en el eje con el que queremos que la puerta gire, es decir, el margen de la puerta, donde realmente iría la bisagra, y hacemos girar este cubo invisible los 90º. De esta

forma, si colocamos la puerta dentro de la bisagra, esta girara solidaria con el cubo transparente y nos abrirá la puerta por el eje que deseamos.

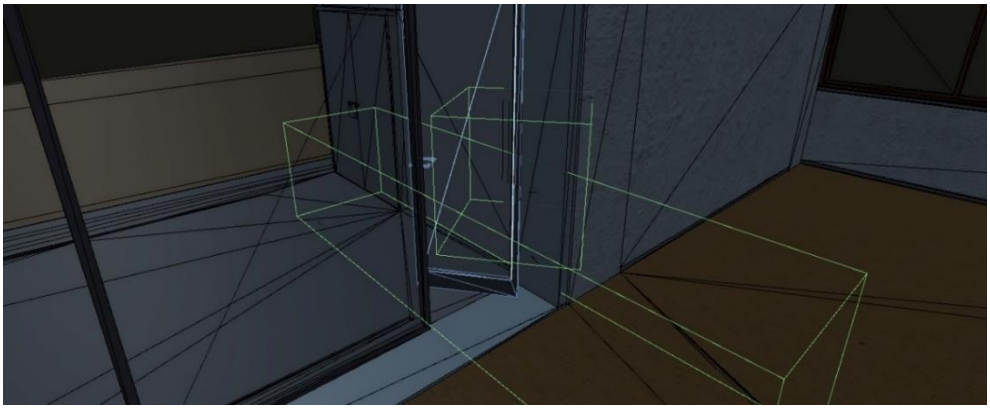

Fig. *6.13* Mecanismos para abrir y cerrar una puerta

6.5.3. Escaleras

El segundo problema con el que nos topamos son las escaleras, también esencial para poder “navegar” o desplazarnos con fluidez por el edificio.

Dado que nuestro personaje solo es un *Collider* con forma de capsula, cuando este llega al primer peldaño de la escalera no la sube, sino que colisiona como si de una pared se tratara.

La forma en que se debe solucionar este problema es cambiando esa capsula por un humanoide con piernas, generar una animación con las que el humanoide mueva las piernas de forma que pueda subir escaleras y hacerlo coincidir cuidadosamente con los peldaños de esta. Pero nuestro proyecto no está enfocado a hacer un personaje realista, así que nos hemos decantado por una opción más sencilla e igual de eficaz, siempre que la aplicación sea en primera persona.

Nuestra alternativa consiste en quitar los *Colliders* de los peldaños de las escaleras, y en su lugar, colocar un box *Collider* siguiendo la inclinación de estas, como se puede observar en la siguiente imagen:

Fig. *6.14* Comparación entre *Colliders* para subir correctamente la escalera

El efecto resultante es mucho más realista de lo que pueda parecer a simple vista, además, si volvemos a activar el *Collider* de los peldaños y bajamos ligeramente la rampa que hemos puesto para que se asome la punta de los peldaños por encima del nuevo *Collider* obtenemos un efecto parecido al que se notaría si realmente estuviéramos subiendo unas escaleras.

6.5.4. Vídeo y audio

Una vez superados los aspectos esenciales para desplazarnos por el edificio, pasamos a explorar otra serie de funcionalidades que potencien la sensación de realismo e inmersión. En particular, en este apartado abordamos el reto de conseguir un sistema de reproducción de vídeo en una pantalla virtual con un sistema de sonido envolvente en una de las habitaciones del escenario virtual. Para conseguirlo se nos presentan dos pasos intermedios por resolver: conseguir reproducir un vídeo en algún objeto del escenario y sincronizarlo con el audio envolvente.

Vídeo

Unity nos permite crear texturas cuyo contenido sea una sucesión de imágenes, este componente se llama “MovieTexture”. [50] Este componente tiene dos variantes, la primera es la que nos ofrece la versión gratuita de Unity y la otra la versión de pago.

La diferencia principal entre las dos son los formatos que admite. La versión gratuita puede reproducir una serie de imágenes, por ejemplo jpg, a la tasa de fotogramas que nosotros escojamos, por defecto 24, pero si codificamos nuestro vídeo en “ogv” también puede reproducirlo sin problemas.

La versión de pago, además, es compatible con la mayoría de formatos de vídeo e incluso a la vez con el audio del mismo. Nosotros hemos implementado la versión gratuita, ya que ofrece todo lo que necesitamos para el proyecto.

Pero este componente, en la versión gratuita y en la de pago, tiene otra utilidad también muy interesante, y es que puede mostrar un *stream* de vídeo que provenga de internet, ya sea contenido en directo de un servidor multimedia como un recurso alojado en un servidor web.
Adicionalmente también existe el componente “WebCamTexture” [51], este permite renderizar un *streaming* de vídeo en directo de la misma manera que lo hace el MovieTexture, pudiendo así mostrar en cualquier parte del escenario virtual lo que este captando la Webcam del nuestro ordenador o la cámara de nuestro dispositivo móvil.

Para empezar a mostrar un vídeo primero necesitamos un objeto donde poder renderizarlo, el más adecuado para hacer de pantalla es un plano, ya que solo tiene una superficie, si usáramos cualquier otro objeto reproduciría el vídeo en

cada una de sus caras, por ejemplo, en un cubo estaríamos reproduciendo de forma completamente inútil 5 vídeos que no queremos.

Después es necesario crear un material, dejándolo con las opciones por defecto, ya que el componente MovieTexture reemplazara aquellas propiedades que sean necesarias para ir mostrando las imágenes que al final serán nuestro vídeo , y asignar este material a nuestro plano.

Para finalizar solo resta crear el *Script* que cargara el vídeo y lo mostrara por en la pantalla virtual. Este sencillo *Script* espera que le pasemos el vídeo en formato OGG como variable que almacenara en el objeto MovieTexture, lo asigna a la textura principal del material que ya hemos creado para la pantalla virtual y lo reproduce con la función Play(). Además con la propiedad “loop” en true lo reproduce en bucle.

```
public class videoPlayer : MonoBehaviour {

    public MovieTexture movTexture;

    void Start() {
        GetComponent<Renderer>().material.mainTexture = movTexture;
        movTexture.loop = true;
        movTexture.Play();
    }
}
```

Fig. *6.15 Script* para reproducir un vídeo con el componente MovieTexture

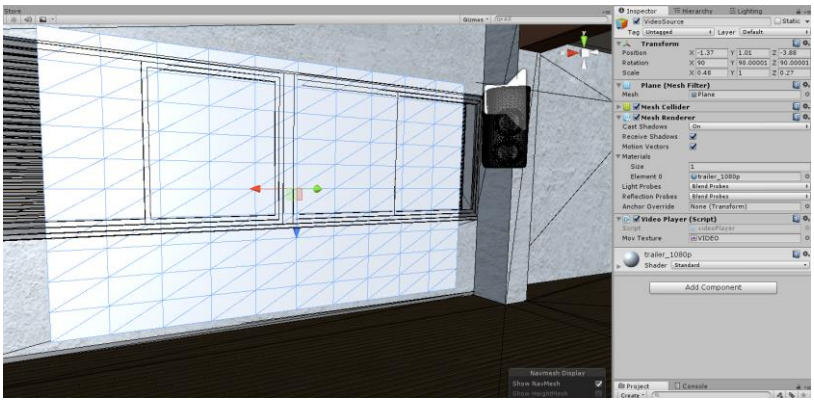

Fig. *6.16* Componentes necesarios para mostrar un vídeo

Audio

Una vez ya tenemos el vídeo vamos a añadirle el audio envolvente. Para ello usaremos el componente “Audio source” de Unity. Si el vídeo debía estar en ogg para funcionar correctamente, el audio debe estar codificado en Vorbis, siempre que usemos la versión gratuita de Unity.

Aunque existe un proyecto llamado “SECTR audio” [52] que promete una sensación de audio inmersivo muy potente, nosotros hemos conseguido unos resultados muy prometedores usando tan solo dos módulos “Audio Source” para los canales derecha e izquierda de la muestra de audio de que disponemos.

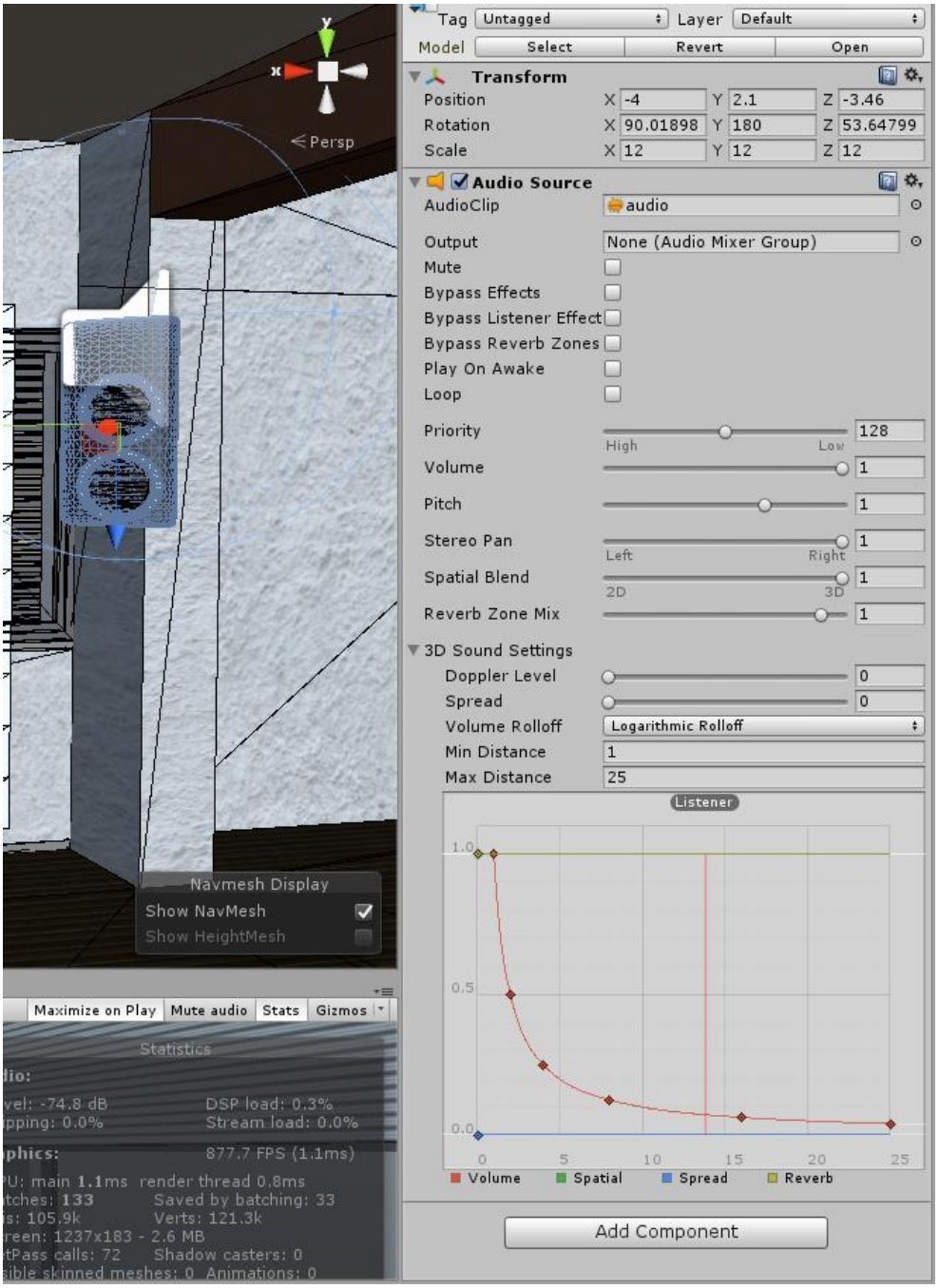

Fig. *6.17* Propiedades del módulo Audio Source

Para colocar un *Audio Source* es necesario colocar un objeto en el escenario, puede ser un objeto vacío, nosotros hemos puesto una pareja de altavoces negros para dotar al sistema de más realismo.

El primer parámetro importante de este módulo es el *AudioClip*, aquí es donde pondremos nuestro fichero de sonido codificado con Vorbis. Para nuestro proyecto es importante que este fichero este en estéreo, o también podemos usar dos ficheros diferentes, uno para cada canal.

En caso de tener la muestra en estéreo nos aparecerá una opción para el balance entre los dos canales. En este caso, para cada uno de los dos módulos, vamos a moverlo hacia la derecha y la izquierda según cada uno de los altavoces.

La última de las opciones que nos ofrece el *Audio Source* son los ajustes del sonido 3D. Básicamente nos permite definir una curva que representa el volumen con el que el *Audio Listener* (por defecto situado en la cámara principal) oirá la fuente de sonido normalizado entre 1 y 0, máximo volumen y silencio respectivamente.

También es importante la opción *Spread* [53], esta nos permite variar, entre 0 y 360, el ángulo de propagación del sonido 3D en el espacio del altavoz. Es decir, si lo dejamos a 0 grados no importara la orientación del oyente respecto a la fuente de sonido, tan solo sonara más fuerte o flojo según la curva que hayamos definido. En cambio, a 360 grados sonaran con mayor intensidad aquellos canales que estén mejor orientados a la fuente sonido, procurando una mejor inmersión acústica.

```
private GameObject speaker1;
private GameObject speaker2;
private AudioSource sp1;
private AudioSource sp2;

void Start() {
    speaker1 = GameObject.Find("Speaker1");
    sp1 = speaker1.GetComponent<AudioSource>();
    speaker2 = GameObject.Find("Speaker2");
    sp2 = speaker2.GetComponent<AudioSource>();
    sp2.loop = true;
    sp1.loop = true;
    sp1.Play();
    sp2.Play();
}
```

Fig. *6.18 Script* para reproducir audio con el componente Audio Source

El *Script* para iniciar la reproducción del audio es muy parecida a la del vídeo, una vez obtenemos acceso al objeto que contiene el módulo de audio lo iniciamos con el método “Play()”.

El sistema de sonido 3D de Unity no tiene en cuenta un factor muy significativo, el audio traspasa los *Colliders* como si no estuvieran ahí. Para solucionar este inconveniente hemos usado uno de los módulos de física que provee Unity, el *Raycast*. Este módulo se puede colocar a cualquier objeto de nuestro escenario, y calcula en línea recta hacia la dirección que escojamos cual será la primera colisión que habría en caso de seguir esa línea.

De esta manera, hemos colocado un *Raycast* en la cámara principal que, como si de una brújula se tratara, siempre está orientado al centro de la pantalla. Cuando este rayo no detecta a la pantalla como primera colisión reducimos el volumen un 50% para simular el efecto que realizaría la pared sobre el audio.

Adicionalmente, para evitar posibles fallos, dentro de la misma habitación el *Raycast* queda inhabilitado, ya que una columna o cualquier otro objeto podría obstaculizar la visión directa entre la cámara y la pantalla.

Para acabar solo queda sincronizar el vídeo con los dos canales de audio. Meramente, con ejecutar las tres instrucciones “Play()” de forma consecutiva dentro de una función “Update()” quedara nuestro sistema perfectamente sincronizado. Dado que esta función se ejecuta cada fotograma, nuestro programa está previsto que funcione a un mínimo de 60 fotogramas cada segundo, así que el error que puede llegar a haber entre las 3 ejecuciones es de menos de 17 milésimas de segundo, completamente imperceptible para un humano.

6.5.5. Iluminación

La iluminación no solo es importante como medio para acentuar la sensación de realismo y, por lo tanto, la inmersión, también puede cumplir una importante función de ayuda al diseño de interiores o de elementos como persianas, cortinas o luminarias que complementen la luz natural.

Unity solo nos provee de 4 principales tipos de luces: direccional, de área, punto de luz y foco. Estas pueden proyectar sombras al colisionar con un *Mesh Renderer* e incluso rebotar en estos. Pero Unity no incorpora un Sol, de manera que hay que aprovechar esas luces para conseguir el resultado más parecido.

Adicionalmente Unity incorpora por defecto una luz global, para que no sea necesario añadir ninguna luz para ver las cosas. Lo primero que vamos a hacer es deshabilitar por completo esta iluminación.

Las propiedades la iluminación global se encuentran en el menú “Window -> Lighting”. Si modificamos el parámetro *Ambient intenisty* a 0 la luz ambiente se apagara y veremos todo negro menos el cielo o si ya tenemos alguna otra fuente de luz.

Sol realista

Para simular un la iluminación que haría el Sol, primero colocamos en nuestro proyecto una *Directional light*, y esta la posicionamos a lo alto y en el centro de nuestra escena.

Para que haga una luz significativa, como es la del Sol, aumentamos el valor del parámetro *Intensity* a un valor cercano a 5, siempre dependiendo de la distancia a la que hayamos colocado el foco de luz direccional.
Para que las estancias iluminadas no se queden a oscuras también aumentamos el valor de *Bounce intenisty* a un valor cercano a 2.

Finalmente hay que activar las sombras, en el parámetro *Shadow Type* asignamos el valor *Hard Shadows* y vamos probando diferentes valores de *Strength* hasta que se puedan discernir claramente las sombras de la iluminación que genera el *Bounce intensity*.

Una vez hecho todo esto ya tendremos una iluminación parecida a la que haría un Sol real en nuestro escenario, pero este Sol virtual siempre está en el mismo sitio, con la misma intensidad y la misma inclinación.

Lo ideal sería hacer rotar el sol siguiendo una trayectoria elíptica con el centro más o menos en el mismo centro del escenario, apuntando siempre hacia el medio y modificando la intensidad según la parte de la trayectoria en la que se encuentre el sol.

Dada la complicación de un *Script* que realice todo este procedimiento hemos optado por una solución más sencilla e igualmente realista. Nuestro sol únicamente gira sobre sí mismo 360 grados y varia la intensidad de la *Directioanl light* según el ángulo de inclinación para recrear la iluminación que hace el sol según la hora del día que representa.

Focos realistas

A la hora de renderizar gráficos en 3D hay un factor muy importante, la iluminación, como ya hemos comentado, sin ella solo podríamos producir imágenes totalmente negras. La iluminación se encarga de determinar que partes del escenario 3D serán visibles por la cámara y de qué color se deberán ver teniendo en cuenta aquellos parámetros que hayamos definido usando materiales y texturas.

Existen dos algoritmos principales, usados por la mayoría de motores 3D. Estos son: la iluminación global [54] y el trazado de rayos o *raytracing* [55].

La iluminación global es en realidad un grupo de algoritmos, estos tienen en cuenta diversos factores. Dado una fuente de iluminación, no solo es capaz de determinar que objetos son los afectados por este haz, sino que también calcula las posibles reflexiones que pueda provocar definidas previamente en un *Shader*, una de las propiedades que tiene un material. Estas reflexiones se

calculan como un nuevo foco de luz proveniente del objeto que genera la reflexión.

Este algoritmo también es capaz de calcular diversos tipos de reflexiones y sombras. Pero la complejidad de los cálculos de los diversos algoritmos lo convierte en un sistema inadecuado para generar imágenes en tiempo real.

Aun y así, este algoritmo es usado en muchos motores gráficos, incluido Unity, aunque para que pueda generar aplicaciones a tasas de hasta 60 o incluso 120 fotogramas por segundo, muchas de las reflexiones y sombras se tienen que aproximar o incluso eliminar. De hecho Unity ofrece la posibilidad de escoger entre diversos modos de renderizado, desde *Fastest* hasta *Fantastic*. En el más rápido ni siquiera se llegan a producir sombras, la luz directa queda en gran parte difuminada y las reflexiones son casi inexistentes, mientras que en el modo con más detalle se pueden apreciar muchos detalles significativos.

Como podemos observar en Fig. *6.19*, en el modo *Fantastic* incluso aparece la reflexión de un foco apuntando a la pantalla que en el modo *Fastest* ni se tiene en cuenta.

Fig. *6.19* Comparativa de imagen entre *Fastest* (izquierda) y *Fantastic* (derecha)

La otra técnica de iluminación ampliamente usada por motores 3D, el *raytracing*, obtiene resultados mucho más agradables en comparación a la iluminación global, además por su naturaleza proporciona mucha más escalabilidad.

Este algoritmo funciona usando una técnica parecida a la que hemos hecho ya referencia anteriormente, el *Raycast*.

Se trazan diversos rayos desde el observador hacia el escenario a través del plano que la cámara pretende renderizar. El primer objeto que alcanza el rayo es el objeto visible para la cámara. Para generar una reflexión, se generan nuevamente rayos desde el objeto alcanzado hacia las diferentes partes del escenario. Del mismo modo se pueden generar difracciones dejando al rayo continuar a través del primer objeto alcanzado.

Este método es más o menos exacto dependiendo del número de rayos que se generen. A cuantos más rayos más precisión, pero a la vez mayor consumo de recursos.

A diferencia de la iluminación global, si reducimos la complejidad de este algoritmo se pierden detalles pero no se eliminan luces y/o sombras del todo. Este algoritmo no forma parte del motor gráfico de Unity, pero hay diversos proyectos de terceros intentando aplicar los algoritmos a través de diversos *Scripts* y *Assets*.

Nosotros hemos hecho uso de un *plugin* de código abierto llamado *LightShafts* [56]. En la Fig. 6.20 podemos apreciar parte del potencial de usar la técnica de trazado de rayos, respecto a lo que podíamos observar en la Fig. 6.19.

Existen otros *plugins* de pago que aplican el trazado de rayos en Unity y que se pueden obtener en la *Asset Store*. Algunos de ellos consiguen resultados aún más espectaculares que este que hemos mostrado.

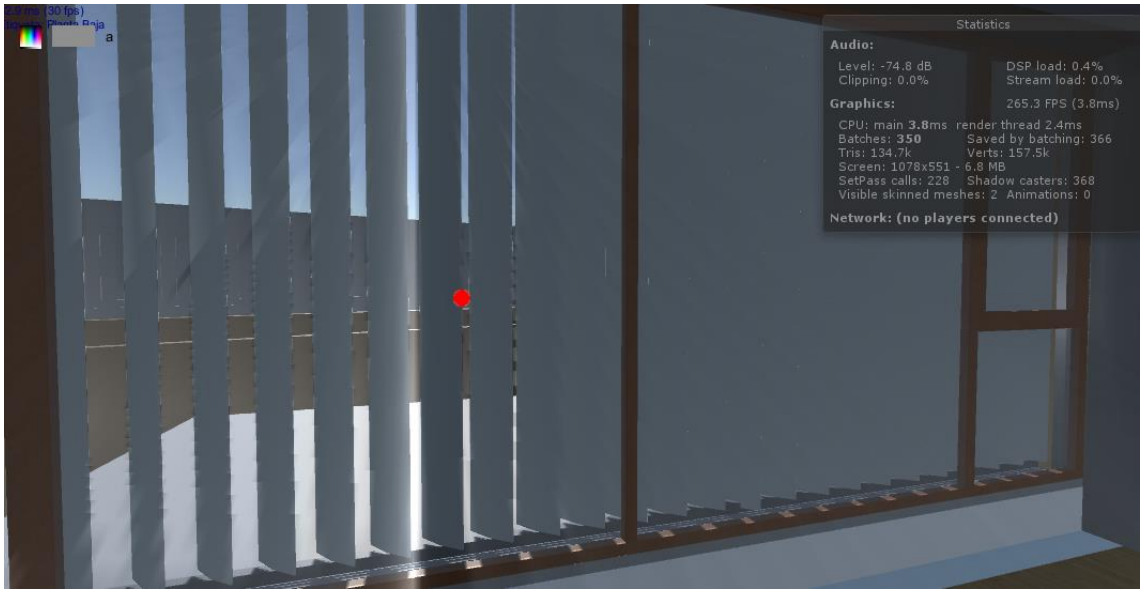

Fig. *6.20* Iluminación con trazado de rayos gracias a *LightShafts*

6.5.6. Instalaciones

Dentro de las capacidades de la modelización 3D, uno de los aspectos más potentes es la disponibilidad de todo el modelo, de toda la información del edificio, lo que facilita su acceso en cualquier momento, a diferencia del edificio real, donde las distancias y obstáculos físicos no permiten un fácil acceso a esa información. A modo de ejemplo, resulta interesante poder visualizar aquellas instalaciones aunque se encuentren ocultas por las paredes, techos o suelos, lo que simplifica tareas de mantenimiento, supervisión o ampliación.

Para conseguir que los objetos se puedan ver a través de las paredes hay que crear un material que tenga esta capacidad. Para conseguirlo hay que modificar una parte en concreto del material, el llamado *Shader* [57].

El *Shader* es una de las propiedades que tiene un material, este define el comportamiento que tendrá el material al tiempo de ser renderizado por el motor gráfico. Define las propiedades básicas del material, por ejemplo, si se trata de un color plano, si la luz se refleja, o si la textura es una imagen.

Los *Shaders* se programan con lenguajes de alto nivel diseñados específicamente para esta labor. Su complejidad varía dependiendo del nivel de detalles y efectos que se quiera generar.

En nuestro caso no queremos generar uno nuevo, solo con modificar el que Unity asigna por defecto a los nuevos materiales es suficiente. Para conseguir que el objeto se vea sea cual sea la posición de la cámara en el escenario hay que añadir una propiedad a una parte *Shader* llamada *SubShader*.

Esta propiedad se llama *ZTest*, esta define cuando la iluminación global debe "tocar" al objeto e iluminarlo. Por defecto esta propiedad tiene asignado el valor *Equal*, lo que quiere decir que si hay un foco de luz apuntando directamente al objeto este se iluminara. En nuestro caso lo hemos modificado al valor *Always*, que como indica es siempre. Es decir, aunque ni siquiera haya luz, el objeto quedara iluminado, por lo que siempre estará en disposición de verse en la cámara cuando esta apunte hacia él.

Para que los objetos con estas propiedades no confundan al usuario, las inicializamos desactivadas, de forma que no se vean. Siguiendo un procedimiento igual que el del apartado 6.5.2, detectamos cuando el usuario está cerca de alguno de estos objetos, para activarlos y así poder ver la tubería, cable o cualquier otro elemento de las instalaciones a través de las paredes u otros obstáculos.

Aprovechando que estamos en un mundo virtual, volvemos a usar la clase *OnGUI* para mostrar información adicional sobre la instalación más cercana al usuario mediante un cuadro por pantalla.

6.5.7. Materiales y texturas

Otra de las funcionalidades de este proyecto es el hecho de poder modificar los materiales y las texturas de ciertos componentes del escenario virtual.

Para conseguirlo volvemos a usar un *Raycast* conjuntamente con una pequeña retícula de color rojo para que el usuario pueda saber dónde está apuntando, Fig. *6.22*. A diferencia de antes este *Raycast* está siempre apuntando al centro de la pantalla.

De esta forma ahora sabemos a qué objeto está apuntando el usuario. Si el objeto al que apunta es un objeto modificable se activa un menú que se puede accionar con el botón asignado a *Fire1*, por defecto haciendo clic con el botón izquierdo del ratón.

A este nuevo menú, Fig. *6.21*, le hemos añadido un botón con un desplegable para modificar la textura que recubre al objeto, en ese caso un sofá, entre una textura de piel y un tapiz rayado. Gracias al *Raycast* en el momento en el que este menú se abre, al hacer clic, sabe exactamente cuál es el objeto del escenario al que nos referimos y así conseguimos cargar solo las texturas referentes a ese objeto y aplicar las transformaciones únicamente al objeto al que hemos apuntado.

Fig. *6.21* Menú de texturas

Además de modificar la textura base del objeto, también tenemos la funcionalidad de modificar el color con el cuál se renderiza esa textura. Gracias a la función “Color()” de la clase “Material” [58] que nos ofrece Unity podemos pasarle un color en formato RGBA (rojo, verde, azul y canal alfa, transparencia) y automáticamente este se aplicara al material en el siguiente fotograma de la asignación.

Para la comodidad del usuario, hemos usado un *Asset* de la tienda de Unity "uGUI Color Picker” [59], este nos muestra por pantalla una rueda de colores como se ve en la Fig. *6.23*. Gracias a este *Asset* podemos cambiar fácilmente y en tiempo real el color que se aplica al material, ya sea haciendo clic en la rueda de colores o introduciendo su valor RGBA a mano.

```
function Update () {
    var ray = cameraCafe.ViewportPointToRay (Vector3(0.5,0.5,0));
    var hit : RaycastHit;

    if (Physics.Raycast (ray, hit)) {
        if(hit.transform.name == "sofa"
            && CrossPlatformInputManager.GetButtonDown("Fire1")
            && pauseEnabled == false){
            enemy = GameObject.Find("sofa");
            EscapePressed();
        }
    }
}

function EscapePressed()
{
    if(pauseEnabled == true)
    {
        pauseEnabled = false;
        Time.timeScale = 1;
        AudioListener.volume = 1;
        Screen.lockCursor = true;
    }
    else if(pauseEnabled == false)
    {
        pauseEnabled = true;
        AudioListener.volume = 0;
        Time.timeScale = 0;
        Screen.lockCursor = false;
    }
}
```

Fig. *6.22 Script* para mostrar el menú de texturas

Para usar el *Asset* hay que colocar el *prefab* llamado “ColorPicker” que nos ofrece el paquete a en cualquier sitio del *Hierarchy*, al ser un componente “OnGUI” este se mostrara siempre al frente de todo en el campo de visión de la cámara. Una vez colocado este tiene una variable pública llamada “Receiver”, donde colocaremos el objeto al que le cambiaremos el color.

Si queremos modificar el color de más de un objeto existen dos posibilidades: colocar un “ColorPicker” para cada uno de los objetos o modificar el *Receiver* de uno solo con la ayuda del *Raycast*.

La primera opción es la más ordenada para el desarrollador, ya que podemos organizar nuestro *Hierarchy* colocando un “ColorPicker” junto a cada objeto al que vayamos a querer modificar el color. Pero por el otro lado, hay que usar el *Raycast* para habilitar solo el “ColorPicker” asignado al objeto que el usuario este visualizando.

La segunda opción es más limpia ya que solo tenemos un “ColorPicker”, pero, en cuanto al código, este se vuelve más confuso, y además, no es tan escalable como la primera opción, ya que se convierte en un “Código espagueti” [60] muy difícil de manejar.

Fig. *6.23* Menú para modificar el color de los materiales

6.5.8. Etiquetas de posición

Para poder discernir en qué lugar del escenario se encuentra nuestro personaje hemos diseñado un sistema de etiquetas. Este sistema pretende informar al usuario acerca de cuál es su ubicación en cada momento y asimismo notificar a la aplicación para que esta pueda obrar en consecuencia a la ubicación del sujeto.

Para implementar dicho sistema debemos definir una etiqueta (*Tag*) en el personaje principal, el que se mueve entre los diferentes emplazamientos del escenario, la cual contendrá el nombre de la estancia que ocupa.

Esta etiqueta se representara por pantalla mediante la clase “OnGUI”. [61] Esta clase, exactamente igual que la función de “Update()”, se ejecuta en cada fotograma, de manera que cualquier cambio en la etiqueta del personaje se refleja al instante por pantalla.


```
void OnGUI()
{
    int w = Screen.width, h = Screen.height;

    GUIStyle style = new GUIStyle();

    Rect rect = new Rect(0, 0, w, h * 2 / 100);
    style.alignment = TextAnchor.UpperLeft;
    style.fontSize = h * 2 / 100;
    style.normal.textColor = new Color (0.0f, 0.0f, 0.5f, 1.0f);
    float msec = deltaTime * 1000.0f;
    float fps = 1.0f / deltaTime;
    string text = string.Format("{0:0.0} ms ({1:0.} fps)\nEtiqueta: " + tagg, msec, fps);
    GUI.Label(rect, text, style);
    //Debug.Log("dsfsd");
}
```

Fig. *6.24* Ejemplo y código pata implementar las etiquetas de posición

Para modificar la etiqueta según cambiamos de lugar la mejor opción es poner un *Collider* transparente, con la misma idea que ya hemos comentado en el apartado 6.5.2. Este objeto tiene dos parámetros públicos, que son las dos etiquetas correspondientes a las dos zonas que este separa.

De esta forma cuando la colision con el intercambiador de etiquetas finaliza este comprueba la etiqueta actual con la primera de las suyas, si no coinciden la intercambia por esa misma, si son iguales, la cambia por la segunda.

Estos *Colliders* estan colocados de forma estrategica para que sea imposible pasar de una zona a otra sin traspasarlos, de esta forma el metodo anteriormente explicado no puede llevar a confusiones entre etiquetas.

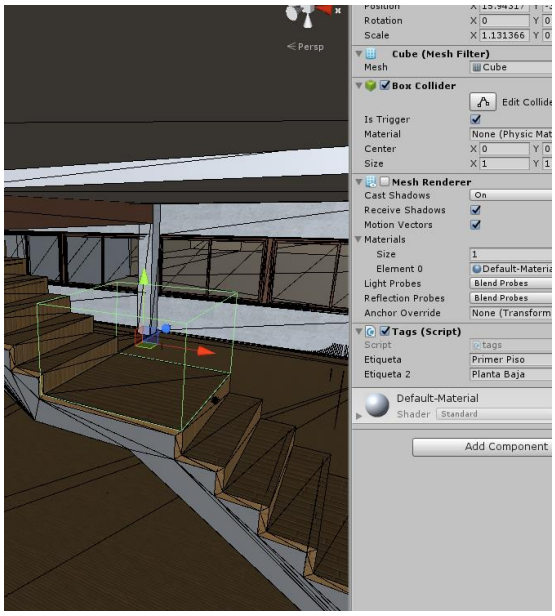

Fig. *6.25 Collider* de ejemplo para modificar las etiquetas de posición

```
public string etiqueta;
public string etiqueta2;
fps playerscript;

void Start () {
    GameObject theplayer = GameObject.Find("RigidBodyFPSController");
    playerscript = theplayer.GetComponent<fps>();
}

void OnTriggerExit(Collider other){
    if(etiqueta == playerscript.tagg){
        playerscript.tagg = etiqueta2;
    }else{
        playerscript.tagg = etiqueta;
    }
}
```

Fig. *6.26* Código de ejemplo para el *Collider* para cambiar las etiquetas de posición

6.5.9. Visita guiada

Para finalizar con la interacción con el modelo 3D del edificio B0, e introducir el último caso de uso que hemos implementado, el que hace referencia a la visita guiada en una inmobiliaria y que ya hemos empezado a desarrollar en el

apartado 6.5.7, modificando las propiedades de un sofá, hemos elaborado un sencillo recorrido por el edificio B0.

Para conseguir que el personaje se mueva completamente solo hay que editar el *Script* que trae por defecto el *Asset*. El problema principal es que si modificamos este *Script* para esta nueva funcionalidad, ya no podremos mover al personaje nosotros mismos.

Para evitar la necesidad de crear un proyecto nuevo o duplicar el *Script* podemos extender sus funcionalidades en otro *Script* nuevo. Solo hace falta escribirlo en el mismo "namespace" y requerir el componente del personaje en cuestión, como podemos ver en la Fig. *6.27*.

```
namespace UnityStandardAssets.Characters.ThirdPerson
{
    [RequireComponent(typeof (ThirdPersonCharacter))]
    public class moveAlone : MonoBehaviour {
```

Fig. *6.27 Script* para extender otro *Script*

Ahora que ya tenemos acceso a modificar el *Script* que hace mover al personaje hay que identificar cual es la función que consigue desplazar al personaje.

Esta función es la "m_Character.Move()", como parámetro espera recibir un "Vector3" indicándole en qué dirección se moverá. En la Fig. *6.28* se puede observar cómo obtener los "Vector3" necesarios para mover al personaje en las 4 principales direcciones de X e Y, además de la función "m_Character.Move()".

```
private void FixedUpdate()
{
    m_Character.Move(row, false, false);
}

void OnTriggerEnter(Collider other) {
    if(stop){
        if(other.tag == "X"){
            row = new Vector3 (1,0,0);
        }else if(other.tag == "Y"){
            row = new Vector3 (0,0,1);
        }else if(other.tag == "IX"){
            row = new Vector3 (-1,0,0);
        }else if(other.tag == "IY"){
            row = new Vector3 (0,0,-1);
        }else if(other.tag == "S"){
            row = new Vector3 (0,0,0);
            stop = false;
        }
    }
}
```

Fig. *6.28 Script* para el movimiento automático

Este *Script*, que se asigna al objeto del personaje, funciona en combinación de un *Cube* sin *Mesh Renderer* al que llamamos cubo cambia-etiquetas.

Cuando el personaje colisiona con el cubo cambia-etiquetas, este comprueba el *Tag* que tiene. Estas etiquetas pueden ser:

- **X**: Modifica la trayectoria del personaje hacia el eje positivo X del escenario.
- **Y**: Modifica la trayectoria del personaje hacia el eje positivo Y del escenario.
- **IX**: Modifica la trayectoria del personaje hacia el eje negativo X del escenario.
- **IY**: Modifica la trayectoria del personaje hacia el eje negativo y del escenario
- **S**: Detiene al personaje

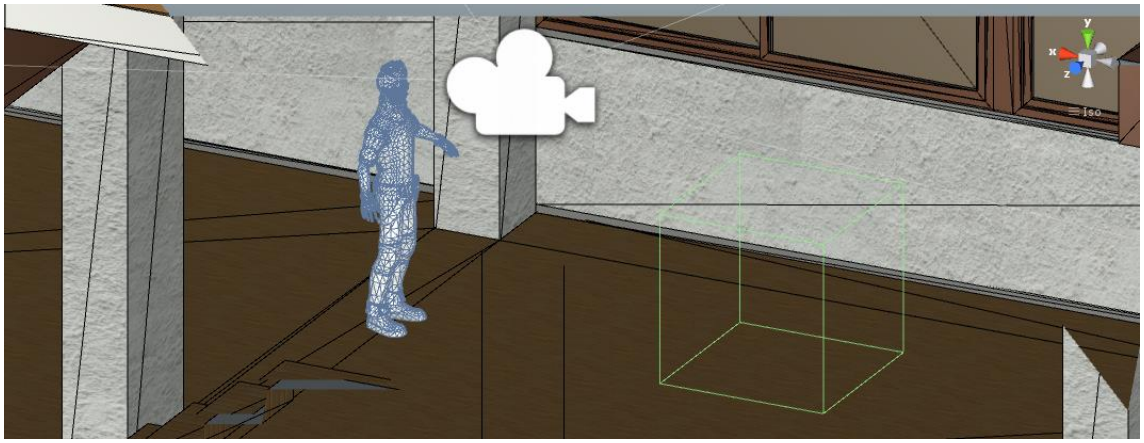

Fig. *6.29* Personaje y cubo cambia-etiquetas

Esta etiqueta hace que el *Script* modifique el valor del “Vector3”, y gracias a la función “FixedUpdate()”, al final del siguiente Fotograma se aplicara el movimiento correspondiente al personaje.

Colocando unos cuantos de estos cubos cambia-etiquetas por todo el edificio se puede hacer una visita guiada por cualquier escenario 3D en el que el usuario tan solo deberá escuchar, si hay sonido, y mirar a su alrededor para poder apreciar los detalles del escenario.

Capitulo 7. Resultados

Finalizada la etapa de implementación, procedemos a exponer los resultados obtenidos, los comparamos con el objetivo definido y exploramos las posibilidades futuras que, por falta de tiempo de tiempo y recursos no hemos tenido ocasión de implementar.

7.1. Expectativas y resultados

Creemos que hemos logrado alcanzar la mayoría de metas que nos habíamos propuesto.

Partiendo de la base adquirida durante el transcurso de nuestros estudios universitarios, nos hemos introducido en el desarrollo de aplicaciones 3D, principalmente con Unity aunque también hemos comprobado de primera mano la potencia gráfica superior de otro motor más enfocado a aplicaciones profesionales, Unreal Engine.

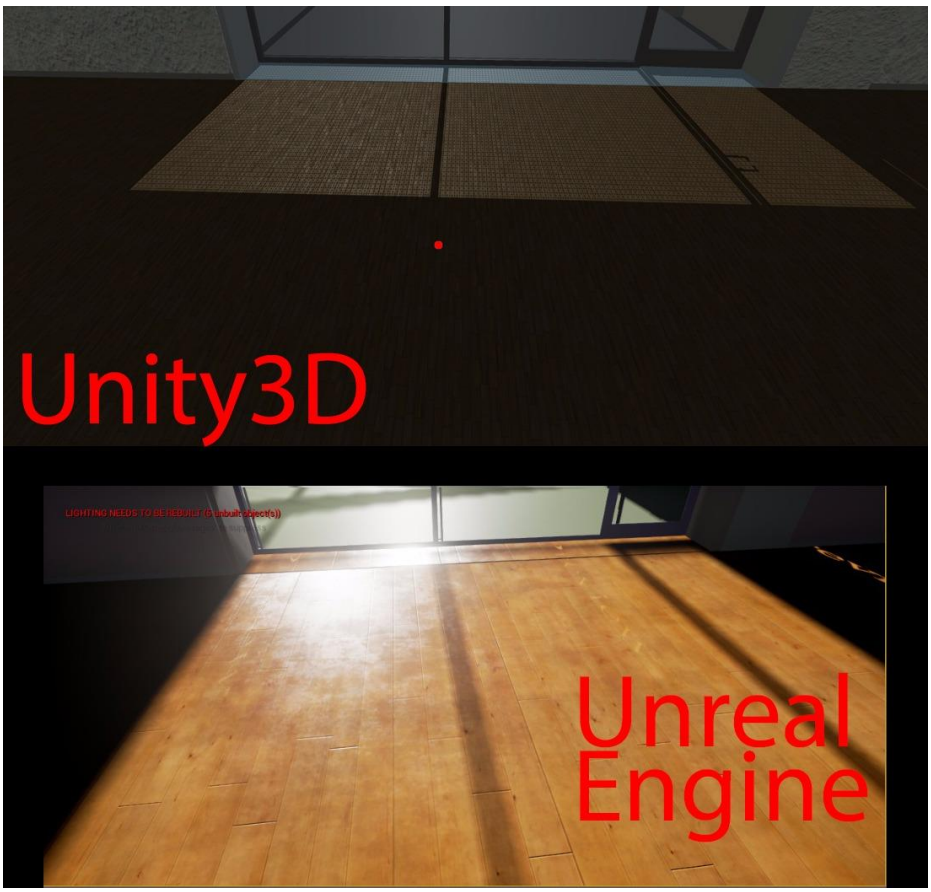

Fig. *7.1* Comparativa entre Unity3D (arriba) y Unreal Engine (abajo)

Hemos producido una aplicación que consigue llegar a los 60 FPS de forma relativamente estable en la mayoría de nuestros dispositivos, aunque como hemos comentado previamente, lo ideal serían 90 FPS constantes.

Hemos conseguido emplear todos los dispositivos de control que habíamos propuesto, aunque para la versión Android, debido a las limitaciones que este presenta para su emparejamiento, únicamente nos ha servido el mando Ipega.

Tras lograr la visión estereoscópica de la aplicación, así como su campo de visión de 360º, en conclusión, una aplicación de RV, hemos procedido a instalarla en el móvil y acoplar este a las VR Box.

Esto nos ha ocasionado un imprevisto, si bien según las especificaciones del dispositivo este es compatible con teléfonos de hasta 6 pulgadas [62], lo cual es cierto ya que físicamente cabe, no han tenido en cuenta que para móviles tan grandes como nuestro Huawei Mate 8 queda bastante ajustado y presiona inevitablemente el botón de bloqueo y encendido. Para el LG G3, aunque no es mucho más pequeño, al contar con los botones en la parte posterior del dispositivo, este funciona perfectamente.

Podemos movernos con total libertad por dentro y fuera del edificio usando un controlador externo, aunque nos hubiera gustado también aprovechar la movilidad que nos ofrece un dispositivo móvil, combinado con IndoorAtlas, 5.1, para cambiar el paradigma: en vez de controlar el personaje con un mando, sin movernos físicamente de nuestro asiento, añadir un grado más de inmersión y movernos nosotros mismos por dentro del edificio real a la vez que vemos como se actualiza la posición en la aplicación, creando la ilusión de que, realmente, nosotros somos el personaje virtual. Finalmente por falta de tiempo y cómo consideramos que ya teníamos suficientes ingredientes, no lo implementamos.

En cuanto a la posibilidad de verificar el resultado final de la obra, podemos visualizar en la aplicación cómo se planeó originalmente el edificio y compararlo en la vida real con el edificio construido. Sin embargo, como no dispusimos del modelo real as-built, (modelo final del edificio construido), hasta las últimas fases de la realización del proyecto, no hemos implementado ninguna funcionalidad especial para comparar ambos modelos de forma virtual. Si tuviéramos que hacerlo, nos planteamos dos posibles soluciones:

- Crear un proyecto con los 2 modelos de edificio, uno al lado del otro y poder moverse de uno a otro.
- Crear un proyecto con 2 escenas, una con cada uno de los edificios y poder cambiar de escena pulsando un botón y aparecer en la misma posición del otro edificio.

Para el caso de las inmobiliarias, hemos hecho una escena aparte, usando de base el mismo proyecto con las demás funcionalidades implementadas con la diferencia que en vez de controlar manualmente al personaje, este se empieza a mover de forma autónoma siguiendo un recorrido predefinido para mostrar la habitación. Es un recorrido muy sencillo pero es únicamente para demostrar que se podría extrapolar a un recorrido entero de una vivienda. Para darle más valor

a la visita podríamos implementar posteriormente audio explicativo de cada estancia proveniente de otro personaje con el rol de agente inmobiliario.

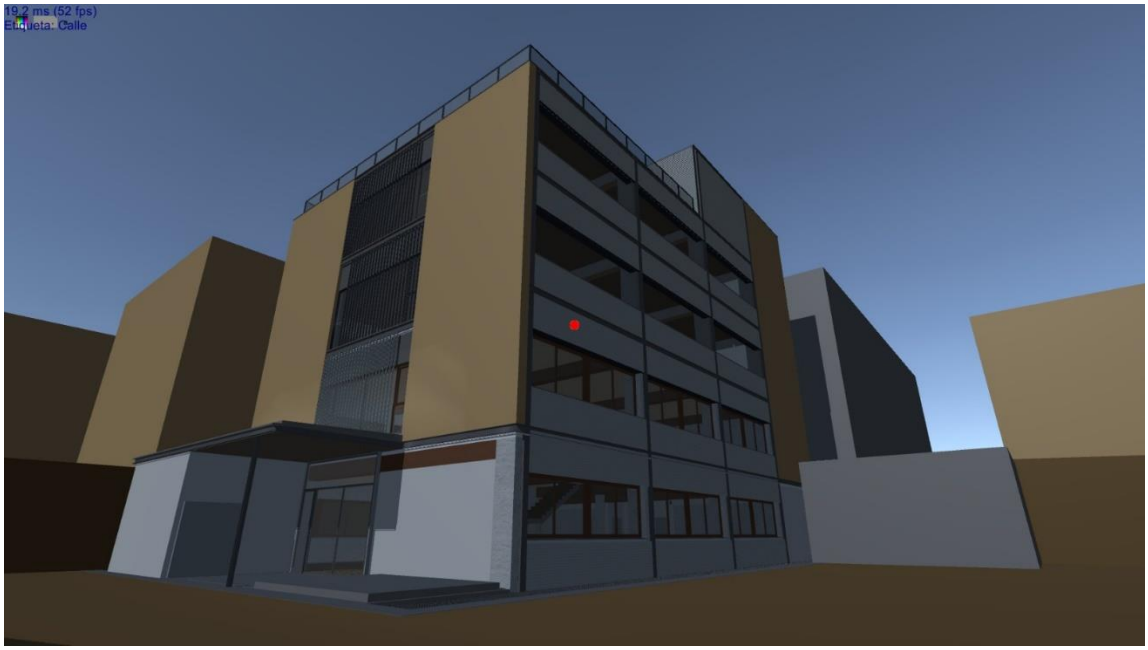

Fig. *7.2* Imagen del exterior del edificio

Fig. *7.3* Imagen del interior del edificio

7.2. Especificaciones

El proceso de desarrollo de la aplicación requiere más potencia que la ejecución posterior una vez compilada y generado el ejecutable. Si bien es cierto que la aplicación compilada va fluida prácticamente siempre en ambos portátiles, hay algunas situaciones más exigentes, como por ejemplo al salir al exterior, ya que tiene que renderizar una cantidad más elevada de gráficos, en que se reducen ligeramente los FPS si tenemos seleccionado la máxima calidad gráfica (*Fantastic*).

Se trata de ordenadores que, aun siendo portátiles, están orientados al *gaming,* es decir, cuentan con procesadores y tarjetas gráficas más potentes que la media.

Esto implica que en ordenadores más básicos puede ser necesario escoger una configuración con una calidad visual más reducida si queremos que los FPS se mantengan a niveles elevados la mayor parte de la simulación. Aunque esto reduzca ligeramente la calidad de la experiencia es preferible a una tasa más baja de fotogramas por segundo que afecte muy negativamente.

Como queda claro en la imagen comparativa Fig. *7.1*, Unreal Engine es mucho más potente gráficamente, por lo que requiere también un PC con más capacidad de procesado.

Hasta ahora los dispositivos móviles de los que disponíamos tenían la suficiente potencia para ejecutar nuestra aplicación realizada con Unity. Sin embargo, estos no están preparados para correr con suficiente fluidez la misma aplicación aprovechando las ventajas que aporta Unreal Engine como por ejemplo la iluminación o las texturas superiores.

Para visualizar estas aplicaciones en RV ya no podemos hacer uso de nuestros móviles combinados con las VR Box sino que requerirían dar el paso hacia otros dispositivos preparados exclusivamente para RV como las Oculus Rift o las HTC Vive, las cuales por otro lado requieren de ordenadores mucho más potentes de los que disponemos, comentados en el apartado 4.2.1.

Capitulo 8. Conclusiones

Después de todo el proceso de elaboración de este proyecto confirmamos aquello que nos planteamos confirmar o desmentir en un inicio: la realidad virtual, efectivamente, tiene mucho que decir y aportar a la recreación de escenarios 3D para la construcción.

Pero no solo eso, el hecho de contar con modelos reales de edificios nos abre alternativas con mucho potencial para multitud de usos como las inmobiliarias o la prevención de riesgos laborales, como ya han explorado otros compañeros conjuntamente con CIMNE y ayudándose del mismo modelo 3D del edificio.

Como conclusión, analizamos los pros y contras del empleo de modelos virtuales en la construcción:

Como ventajas, creemos que es muy interesante poder explorar la obra de forma remota sin tener que desplazarse al terreno. Así mismo se podrían facilitar las inspecciones de obra si en la recreación 3D se han añadido los datos de las distintas instalaciones que pueda haber, (principalmente agua, luz y gas, pero también otras como instalación de cableado de red o antenas). Además, como la recreación sería sobre el modelo proyectado, se podría comparar sobre el terreno los cambios en la obra construida.

Obviamente también tendría sus dificultades e inconvenientes, como la necesidad de actualizar los datos de la recreación de forma regular, así como la dificultad de aprendizaje de los usuarios para emplear las herramientas y requerir del modelo 3D adaptable a Unity o su adaptación posterior.

Nuestro compañero Arnau Rigol, en su TFG [63] ha comentado muy acertadamente lo siguiente: "*(…) No obstante, la importación no es directa y por cuestiones de compatibilidad de formatos, es necesaria una adaptación intermedia mediante 3ds Max. Una automatización de este proceso favorecería el uso de la Realidad Virtual, ya que una vez importado, el resto de acciones a realizar en Unity son fáciles de introducir mediante prefabs. (…)*".

Efectivamente, consideramos clave la necesidad de contar con una herramienta que automatice la importación de los modelos 3D y las acciones a realizar. De este modo la integración de aplicaciones como la nuestra en el campo de la construcción sería más sencilla y realista.

Capitulo 9. Referencias

[1] «TheGuardian,» 22 07 2014. [En línea]. Available: https://www.theguardian.com/technology/2014/jul/22/facebook-oculus-rift-acquisition-virtual-reality. [Último acceso: 02 02 2017].

[2] Oculus VR, LLC, «Oculus,» facebook, [En línea]. Available: https://www3.oculus.com/en-us/rift/. [Último acceso: 05 02 2017].

[3] Wikipedia, «Oculus Rift,» [En línea]. Available: https://es.wikipedia.org/wiki/Oculus_Rift. [Último acceso: 05 02 2017].

[4] HTC Corporation, «Vive,» [En línea]. Available: https://www.vive.com/eu/. [Último acceso: 05 02 2017].

[5] Wikipedia, «HTC Vive,» [En línea]. Available: https://es.wikipedia.org/wiki/HTC_Vive. [Último acceso: 05 02 2017].

[6] «Glossary of Virtual Reality Terminology,» de *International Journal of Virtual Reality*, vol. 1, 1995.

[7] «Real Academia Española,» [En línea]. Available: http://dle.rae.es/srv/fetch?id=VH7cofQ. [Último acceso: 05 11 2016].

[8] Wikipedia, «Realidad virtual,» 15 12 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Realidad_virtual. [Último acceso: 27 01 2017].

[9] «Neurodigital technologies,» [En línea]. Available: https://www.neurodigital.es/gloveone/. [Último acceso: 15 12 2016].

[10] «Manus VR,» 2015. [En línea]. Available: https://manus-vr.com/. [Último acceso: 15 12 2016].

[11] «FEELREAL,» 2015. [En línea]. Available: http://feelreal.com/. [Último acceso: 15 12 2016].

[12] «Olorama technology,» 2017. [En línea]. Available: http://www.olorama.com/es/. [Último acceso: 15 12 2016].

[13] «Ubisoft nosulusrift,» 2016. [En línea]. Available: http://nosulusrift.ubisoft.com/?lang=es-ES. [Último acceso: 15 12 2016].

[14] «Technoreeze,» 17 06 2011. [En línea]. Available: http://www.technoreeze.com/es/2011/06/17/realidad-virtual-en-los-sentidos-del-gusto-y-el-olfato/. [Último acceso: 15 12 2016].

[15] «VR-Box,» [En línea]. Available: http://www.vr-box.es/. [Último acceso: 14 01 2017].

[16] «Aliexpress,» [En línea]. Available: https://es.aliexpress.com/wholesale?catId=0&initiative_id=SB_20170119040705&SearchText=vr. [Último acceso: 16 12 2016].

[17] «GearBest,» [En línea]. Available: http://www.gearbest.com/virtual-reality-c_11366/. [Último acceso: 16 12 2016].

[18] «Mortonheilig,» [En línea]. Available: http://www.mortonheilig.com/TelesphereMask.pdf. [Último acceso: 16 12 2016].

[19] «Worrydream,» [En línea]. Available: http://worrydream.com/refs/Sutherland%20-%20The%20Ultimate%20Display.pdf. [Último acceso: 16 12 2016].

[20] «PlayStation,» [En línea]. Available: http://us.playstation.com/ps2/accessories/eyetoy-usb-camera-ps2.html. [Último acceso: 03 02 2017].

[21] «Xbox,» [En línea]. Available: http://www.xbox.com/es-AR/xbox-one/accessories/kinect. [Último acceso: 03 02 2017].

[22] Wikipedia, «VPL Research,» 16 12 2016. [En línea]. Available: https://en.wikipedia.org/wiki/VPL_Research. [Último acceso: 29 12 2016].

[23] 26 03 2014. [En línea]. Available: http://tecnologia.elpais.com/tecnologia/2014/03/26/actualidad/1395796446_034242.html. [Último acceso: 29 01 2017].

[24] Google, «VR Google,» [En línea]. Available: https://vr.google.com/. [Último acceso: 02 02 2017].

[25] Google, «VR Google,» [En línea]. Available: https://vr.google.com/daydream/. [Último acceso: 02 02 2017].

[26] TheVoid, «TheVoid,» [En línea]. Available: https://thevoid.com/. [Último acceso: 03 02 2013].

[27] «Wikipedia,» 29 11 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Realidad_aumentada. [Último acceso: 03 02 2017].

[28] Microsoft, «Microsoft,» [En línea]. Available: https://www.microsoft.com/microsoft-hololens/en-us. [Último acceso: 03 02 2017].

[29] MagicLeap, «MagicLeap,» [En línea]. Available: https://www.magicleap.com/#/home. [Último acceso: 03 02 2017].

[30] Wikipedia, «Giróscopo,» 17 01 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Gir%C3%B3scopo. [Último acceso: 18 01 2017].

[31] Wikipedia, «Estereoscopía,» 01 11 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Estereoscop%C3%ADa. [Último acceso: 20 01 2017].

[32] «Google,» [En línea]. Available: https://vr.google.com/cardboard/. [Último acceso: 12 01 2017].

[33] «Virtuix,» [En línea]. Available: http://www.virtuix.com/. [Último acceso: 22 01 2017].

[34] «Vive,» [En línea]. Available: https://www.vive.com/us/vive-tracker/. [Último acceso: 22 01 2017].

[35] Unity3D, «Unity3d, tutorials» [En línea]. Available: https://unity3d.com/es/learn/tutorials. [Último acceso: 10 08 2016].

[36] Unity3D, «Unity3d, tuorial, roll a ball» [En línea]. Available: https://unity3d.com/es/learn/tutorials/projects/roll-ball-tutorial. [Último acceso: 10 08 2016].

[37] «Intel,» [En línea]. Available: https://iq.intel.com/the-technical-challenges-of-virtual-reality/. [Último acceso: 25 01 2017].

[38] «Xataka,» 07 01 2016. [En línea]. Available: https://www.xataka.com/realidad-virtual-aumentada/por-que-un-juego-de-realidad-virtual-de-oculus-pide-un-pc-mucho-mas-potente-que-un-juego-normal. [Último acceso: 04 02 2017].

[39] «Unity3D,» 2016. [En línea]. Available: https://docs.unity3d.com/es/current/Manual/3D-formats.html. [Último acceso: 2017 01 10].

[40] Wikipedia, «FBX,» 29 12 2016. [En línea]. Available: https://en.wikipedia.org/wiki/FBX. [Último acceso: 10 01 2017].

[41] «Unity3D,» [En línea]. Available: https://docs.unity3d.com/es/current/Manual/HOWTO-exportFBX.html. [Último acceso: 10 01 2017].

[42] «GitHub,» 24 03 2016. [En línea]. Available: https://github.com/Flafla2/Unity-Wiimote. [Último acceso: 20 08 2016].

[43] «GitHub,» 3 12 2016. [En línea]. Available: https://github.com/nefarius/ScpToolkit/blob/master/README.md#installation-requirements. [Último acceso: 10 01 2017].

[44] «pcsx2,» 08 09 2015. [En línea]. Available: http://forums.pcsx2.net/Thread-ScpToolkit-XInput-Wrapper-aka-ScpServer-Reloaded. [Último acceso: 10 01 2017].

[45] Wikipedia, «Protocolos Bluetooth,» 04 07 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Protocolos_Bluetooth. [Último acceso: 24 08 2016].

[46] «Code Google,» [En línea]. Available: https://code.google.com/p/android/issues/detail?id=58164. [Último acceso: 25 08 2017].

[47] «Bluez,» [En línea]. Available: http://www.bluez.org/. [Último acceso: 01 01 2017].

[48] Wikipedia, «Android rooting,» 22 08 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Android_rooting. [Último acceso: 20 01 2017].

[49] Wikipedia, «Custom Recovery,» 13 05 2016. [En línea]. Available: https://en.wikipedia.org/wiki/Custom_Recovery. [Último acceso: 22 01 2017].

[50] «Unity3D,» [En línea]. Available: https://docs.unity3d.com/es/current/Manual/class-MovieTexture.html. [Último acceso: 11 01 2017].

[51] «Unity3D,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/WebCamTexture.html . [Último acceso: 11 01 2017].

[52] «Unity3D Assetstroe,» 21 11 2016. [En línea]. Available: https://www.assetstore.unity3d.com/en/#!/content/15325 . [Último acceso: 05 12 2016].

[53] «Unity3D,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/AudioSource-spread.html . [Último acceso: 11 01 2017].

[54] Wikipedia, «Iluminación global,» 11 01 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Iluminaci%C3%B3n_global. [Último acceso: 20 01 2017].

[55] Wikipedia, «Trazado de rayos,» 06 11 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Trazado_de_rayos. [Último acceso: 20 01 2017].

[56] robertcupisz, «LightShafts,» 06 09 2016. [En línea]. Available: https://github.com/robertcupisz/LightShafts. [Último acceso: 09 12 2016].

[57] Wikipedia, «Shader,» 25 07 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Shader. [Último acceso: 20 01 2017].

[58] «Unity3D,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/Material.html. [Último acceso: 12 01 2017].

[59] «Unity3D AssetStore,» [En línea]. Available: https://www.assetstore.unity3d.com/en/#!/content/47956. [Último acceso: 12 01 2017].

[60] Wikipedia, «Código espagueti,» 19 12 2016. [En línea]. Available: https://es.wikipedia.org/wiki/C%C3%B3digo_espagueti. [Último acceso: 12 01 2017].

[61] «Unity3d,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnGUI.html. [Último acceso: 11 01 2017].

[62] «Review-Hub,» 03 03 2016. [En línea]. Available: http://www.review-hub.co.uk/vr-box-virtual-reality-headset-review/. [Último acceso: 01 02 2017].

[63] A. Rigol Carrasco, «Posibilidades de la Realidad VIrutal para la prevención de riesgos laborales en el sector de la construcción,» Febrero 2017.

[64] [En línea]. Available: https://docs.unity3d.com/es/current/Manual/class-MovieTexture.html. [Último acceso: 1 2017].

Capitulo 10. Anexo

Obrir2portes.js

Código para abrir puertas dobles con activador vía *Collider*.

```
#pragma strict

var angley: float=-90.0;
var target: GameObject;
var target2: GameObject;
private var targetvalue: float=0.0;
private var currentvalue: float=0.0;
private var easing: float=0.02;

function Update () {
 currentvalue=currentvalue+(targetvalue - currentvalue)*easing;
 target.transform.rotation=Quaternion.identity;
 target.transform.Rotate(-89.981,currentvalue,0);
 target2.transform.rotation=Quaternion.identity;
 target2.transform.Rotate(-89.981,-currentvalue,0);
}

function OnTriggerEnter(other:Collider){
 targetvalue=angley;
 currentvalue=0;
}

function OnTriggerExit(other:Collider){
      currentvalue=angley;
      targetvalue=0;
}
```

ObrirPortaFixed.cs

Código para abrir una puerta mediante un activador por distancia pulsando una tecla para abrirla.

```
using UnityEngine;
using System.Collections;

public class ObrirPortaFixed : MonoBehaviour {

      public GameObject target;
      public float doorOpenAngle = 90.0f;
      public float doorCloseAngle = 0.0f;
      public float doorAnimSpeed = 2.0f;
      private Quaternion doorOpen = Quaternion.identity;
      private Quaternion doorClose = Quaternion.identity;
      private Transform playerTrans = null;
      public bool doorStatus = false;
      private bool doorGo = false;
      private bool showLabel = false;

      void Start() {
            doorStatus = false;
            doorOpen = Quaternion.Euler (0, 0, doorOpenAngle);
            doorClose = Quaternion.Euler (0, 0, doorCloseAngle);
```

```
            playerTrans = GameObject.Find
("RigidBodyFPSController").transform;
      }

      void Update() {
            if (Vector3.Distance(playerTrans.position,
this.transform.position) < 4f) {
                  showLabel = true;
            }else{
                  showLabel = false;
            }
            if (Input.GetKeyDown(KeyCode.F) && !doorGo) {
                  if (Vector3.Distance(playerTrans.position,
this.transform.position) < 5f) {
                        if (doorStatus) {
                              StartCoroutine(this.moveDoor(doorClose));
                        } else {
                              StartCoroutine(this.moveDoor(doorOpen));
                        }
                  }
            }
      }

      void OnGUI()
      {
            int w = 250, h = 300;
            GUIStyle style = new GUIStyle();
            Rect rect = new Rect((Screen.width-w)/2, (Screen.height-
h)/2, w, h);
            style.alignment = TextAnchor.MiddleCenter;
            style.fontSize = h * 6 / 100;
            string text = string.Format("Pulsa F para abrir/cerrar la
puerta");
            if (showLabel) {
                  GUI.Label(rect, text, style);
            }
      }

      public IEnumerator moveDoor(Quaternion dest) {
            Debug.Log(Quaternion.Angle(target.transform.localRotation,
dest));
            doorGo = true;
            while (Quaternion.Angle(target.transform.localRotation,
dest) > 4.0f) {
                  target.transform.localRotation =
Quaternion.Slerp(target.transform.localRotation, dest, Time.deltaTime
* doorAnimSpeed);
                  yield return null;
            }
            doorStatus = !doorStatus;
            doorGo = false;
            yield return null;
      }
}
```

videoPlayer.cs

Código para reproducir una fuente de video y audio almacenados como *Assets*. El video en una pantalla y el audio estéreo en dos altavoces, uno por canal.

```
using UnityEngine;
using System.Collections;
using UnityStandardAssets.CrossPlatformInput;

public class videoPlayer : MonoBehaviour {
 private bool showLabel = false;
 private Transform playerTrans = null;
 private GameObject speaker1;
 private GameObject speaker2;
 private AudioSource sp1;
 private AudioSource sp2;
 public MovieTexture movTexture;
 private bool isPlaying = false;

      void Start() {
            GetComponent<Renderer>().material.mainTexture = movTexture;
            playerTrans =
GameObject.Find("RigidBodyFPSController").transform;
 speaker1 = GameObject.Find("Speaker1");
 sp1 = speaker1.GetComponent<AudioSource>();
 speaker2 = GameObject.Find("Speaker2");
 sp2 = speaker2.GetComponent<AudioSource>();
 sp2.loop = true;
 sp1.loop = true;
 movTexture.loop = true;

 }

 void Update () {
 if (Vector3.Distance(playerTrans.position, this.transform.position) <
4f)
 {
 showLabel = true;
 }
 else
 {
 showLabel = false;
 }
 if (Input.GetKeyDown(KeyCode.F) ||
CrossPlatformInputManager.GetButtonDown("F"))
 {
 if (Vector3.Distance(playerTrans.position, this.transform.position) <
5f)
 {
 if (!isPlaying)
 {
 isPlaying = true;
 movTexture.Play();
 sp1.Play();
 sp2.Play();
 }
 else
 {
 isPlaying = false;
 movTexture.Pause();
```

```
 sp1.Pause();
 sp2.Pause();
 }
 }
 }
 }
 void OnGUI()
 {
 int w = 250, h = 300;

 GUIStyle style = new GUIStyle();

 Rect rect = new Rect((Screen.width - w) / 2, (Screen.height - h) / 2,
w, h);
 style.alignment = TextAnchor.MiddleCenter;
 style.fontSize = h * 6 / 100;
 string text = string.Format("Pulsa F para reproducir/pausar el
vídeo");
 if (showLabel)
 {
 GUI.Label(rect, text, style);
 }
 }
}
```

Realsun.cs

Código para simular el movimiento del sol y su iluminación.

```
using UnityEngine;
using System;
using System.Collections;

public class realsun : MonoBehaviour {

 public Light sun;
 public float secondsInFullDay = 120f;
 [Range(0,1)]
 public float currentTimeOfDay = 0;
 [HideInInspector]
 public float timeMultiplier = 1f;

 float sunInitialIntensity;

 void Start() {
 sunInitialIntensity = sun.intensity;
 }

 void Update() {
 UpdateSun();

 currentTimeOfDay += (Time.deltaTime / secondsInFullDay) *
timeMultiplier;

 if (currentTimeOfDay >= 1) {
 currentTimeOfDay = 0;
 }
 }

 void UpdateSun() {
```

```
 sun.transform.localRotation = Quaternion.Euler((currentTimeOfDay *
360f) - 90, 90, 0);

 float intensityMultiplier = 1;
 if (currentTimeOfDay <= 0.23f || currentTimeOfDay >= 0.75f) {
 intensityMultiplier = 0;
 }
 else if (currentTimeOfDay <= 0.25f) {
 intensityMultiplier = Mathf.Clamp01((currentTimeOfDay - 0.23f) * (1 /
0.02f));
 }
 else if (currentTimeOfDay >= 0.73f) {
 intensityMultiplier = Mathf.Clamp01(1 - ((currentTimeOfDay - 0.73f) *
(1 / 0.02f)));
 }

 sun.intensity = sunInitialIntensity * intensityMultiplier;
 }
}
```

TuberiaVisible.cs

Código para mostrar la tubería al acercarse, además de información adicional.

```
using UnityEngine;
using System.Collections;

public class TuberiaVisible : MonoBehaviour {

 private Transform playerTrans = null;
 private MeshRenderer m;
 private MeshRenderer n;
 private MeshRenderer o;
 private GameObject tuberia1;
 private GameObject tuberia2;
 private GameObject text;

 void Start () {
 playerTrans = GameObject.Find("RigidBodyFPSController").transform;
 tuberia1 = GameObject.Find("tuberia1");
 tuberia2 = GameObject.Find("tuberia2");
 text = GameObject.Find("New Text");
 m = tuberia1.GetComponent<MeshRenderer>();
 n = tuberia2.GetComponent<MeshRenderer>();
 o = text.GetComponent<MeshRenderer>();
 }

      void Update () {
 if (Vector3.Distance(playerTrans.position, this.transform.position) <
4f)
 {
 m.enabled = true;
 n.enabled = true;
                  o.enabled = true;
 }
 else
```

```
 {
 m.enabled = false;
 n.enabled = false;
                  o.enabled = false;
 }
 }
}
```

Reticle.js

Código para mostrar una retícula y modificar materiales y texturas de un elemento concreto.

```
#pragma strict

import UnityStandard Assets.CrossPlatformInput;

var crosshairTexture : Texture2D;
var position : Rect;
static var OriginalOn = true;
var cameraCafe : Camera;
var texture : Texture;
var texture2 : Texture;
var txt = false;
private var pauseEnabled = false;
var pauseMenuFont : Font;
private var showGraphicsDropDown = false;
private var enemy : GameObject;

function Start()
{
      position = Rect((Screen.width - crosshairTexture.width) / 2,
(Screen.height - crosshairTexture.height) /2, crosshairTexture.width,
crosshairTexture.height);
}

function OnGUI()
{
      if(OriginalOn == true)
 {
 GUI.DrawTexture(position, crosshairTexture);
 }
      GUI.skin.box.font = pauseMenuFont;
      GUI.skin.button.font = pauseMenuFont;
      if(pauseEnabled == true){
            GUI.Box(Rect(Screen.width /2 - 100,Screen.height /2 -
100,250,200), "Texture Menu");
                  if(GUI.Button(Rect(Screen.width /2 -
100,Screen.height /2 ,250,50), "Textures")){
                  if(showGraphicsDropDown == false){
                        showGraphicsDropDown = true;
                  }
                  else{
                        showGraphicsDropDown = false;
                  }
            }
```

```
            if(showGraphicsDropDown == true){
                  if(GUI.Button(Rect(Screen.width /2 +
150,Screen.height /2 ,250,50), "Preset 1")){

      enemy.GetComponent.<Renderer>().material.mainTexture = texture;
                   EscapePressed();
                  }
                  if(GUI.Button(Rect(Screen.width /2 +
150,Screen.height /2 + 50,250,50), "Preset 2")){

      enemy.GetComponent.<Renderer>().material.mainTexture = texture2;
                   EscapePressed();
                  }

                  if(Input.GetKeyDown("escape")){
                        showGraphicsDropDown = false;
                  }
            }
      }
}

function Update () {
      var ray = cameraCafe.ViewportPointToRay (Vector3(0.5,0.5,0));
      var hit : RaycastHit;

      if (Physics.Raycast (ray, hit)) {
            if(hit.transform.name == "sofa" &&
CrossPlatformInputManager.GetButtonDown("Fire1") && pauseEnabled ==
false){
                  enemy = GameObject.Find("sofa");
                  EscapePressed();
            }
      }
}

function EscapePressed()
{
            if(pauseEnabled == true)
            {
                  pauseEnabled = false;
                  Time.timeScale = 1;
                  AudioListener.volume = 1;
                  Screen.lockCursor = true;
            }
            else if(pauseEnabled == false)
            {
                  pauseEnabled = true;
                  AudioListener.volume = 0;
                  Time.timeScale = 0;
                  Screen.lockCursor = false;
            }
}
```

MoveAlone.cs

Código para mover el personaje de forma autónoma por el escenario.

```
using UnityEngine;
using System.Collections;

namespace UnityStandard Assets.Characters.ThirdPerson
{
      [RequireComponent(typeof (ThirdPersonCharacter))]
      public class moveAlone : MonoBehaviour {

            public bool stop = true;
 private ThirdPersonCharacter m_Character;
            private Vector3 row = new Vector3 (-1,0,0);

            void Start(){
                  m_Character = GetComponent<ThirdPersonCharacter>();
            }

            private void FixedUpdate()
            {
                  m_Character.Move(row, false, false);
            }

            void OnTriggerEnter( Collider other) {
                  if(stop){
                        if(other.tag == "X"){
                              row = new Vector3 (1,0,0);
                        }else if(other.tag == "Y"){
                              row = new Vector3 (0,0,1);
                        }else if(other.tag == "IX"){
                              row = new Vector3 (-1,0,0);
                        }else if(other.tag == "IY"){
                              row = new Vector3 (0,0,-1);
                        }else if(other.tag == "S"){
                              row = new Vector3 (0,0,0);
                              stop = false;
                        }
                  }
            }
      }
}
```

Fps.cs

Código para limitar los FPS y mostrar texto adicional en el margen de la pantalla.

```
using UnityEngine;
using System.Collections;

public class fps : MonoBehaviour {
      float deltaTime = 0.0f;
      public string tagg = "Primer Piso";

      void Awake() {
 Application.targetFrameRate = 60;
 }
```

```
	int i = 0;

	void Start(){
		tagg = "Primer Piso";
	}

	void Update()
	{
		deltaTime += (Time.deltaTime - deltaTime) * 0.1f;
		i++;
	}

	void OnGUI()
	{
		int w = Screen.width, h = Screen.height;
		GUIStyle style = new GUIStyle();
		Rect rect = new Rect(0, 0, w, h * 2 / 100);
		style.alignment = TextAnchor.UpperLeft;
		style.fontSize = h * 2 / 100;
		style.normal.textColor = new Color (0.0f, 0.0f, 0.5f,
1.0f);
		float msec = deltaTime * 1000.0f;
		float fps = 1.0f / deltaTime;
		string text = string.Format("{0:0.0} ms ({1:0.}
fps)\nEtiqueta: " + tagg, msec, fps);
		GUI.Label(rect, text, style);
	}
}
```

Tags.cs

Código para modificar las etiquetas de posición.

```
using UnityEngine;
using System.Collections;

public class tags : MonoBehaviour {
	public string etiqueta;
	public string etiqueta2;
	fps playerscript;

	void Start () {
		GameObject theplayer =
GameObject.Find("RigidBodyFPSController");
		playerscript = theplayer.GetComponent<fps>();
	}

	void Update () {

	}
	void OnTriggerExit(Collider other){
		if(etiqueta == playerscript.tagg){
			playerscript.tagg = etiqueta2;
		}else{
			playerscript.tagg = etiqueta;
		}
	}
}
```