UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

DESARROLLO DE UNA APLICACIÓN DE REALIDAD VIRTUAL MULTIUSUARIO PARA LA EXPLORACION DE DISEÑOS INMOBILIARIOS 3D

Fernando Bermúdez Oliver

TRABAJO FINAL DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Tutor UPV: Jorge Orta López

Universitat Politècnica de València – Campus d'Alcoi
julio 2019

Convocatoria de defensa: *julio, 2019*

Resumen

Uno de los retos para todo desarrollador de aplicaciones *software* es el de satisfacer todas las necesidades relacionadas con el usuario, cada vez más exigente y adaptado a las nuevas tecnologías. El presente trabajo final de grado trata sobre el despliegue y desarrollo de una aplicación de realidad virtual en red, ofreciendo un sistema que permita compartir y explorar de manera cooperativa con el resto de usuarios conectados un entorno virtual, el cual está orientado a diseños inmobiliarios en tres dimensiones. Con esta aplicación, se pretende proporcionar a los usuarios de la plataforma una nueva manera de interactuar, comunicarse y gestionar un contenido virtual de manera sincronizada. Con el objetivo principal de juntar la tecnología actualmente demandada de la realidad virtual junto con un entorno en red y una base de datos desde la cual almacenar la información del tiempo de conexión de cada usuario.

Dicho esto, el gran reto como desarrollador de aplicaciones es conseguir unificar todas y cada una de las tecnologías mencionadas en este resumen en una única aplicación.

Palabras clave: Realidad virtual, Inmobiliarias, Unity, VR Multiusuario

Resum

Un dels reptes per a tot desenvolupador d'aplicacions *software* és el de satisfer totes les necessitats relacionades amb l'usuari, cada vegada més exigent i adaptat a les noves tecnologies. El present treball final de grau tracta sobre el desplegament i desenvolupament d'una aplicació de realitat virtual en xarxa, oferint un sistema que permeta compartir de forma cooperativa amb la resta d'usuaris connectats un entorn virtual, el qual està orientat a dissenys immobiliaris en tres dimensions. Amb aquesta aplicació, es pretén proporcionar als usuaris de la plataforma una nova manera d'interactuar, comunicar-se, crear i gestionar un contingut virtual de forma sincronitzada. Amb l' objectiu principal d'ajuntar la tecnologia actualment demandada de la realitat virtual junt amb un entorn en xarxa i una base de dades des de la qual emmagatzemar la informació del temps de connexió de cada usuari.

Dit açò, el gran repte com a desenvolupador d'aplicacions és aconseguir unificar totes i cada una de les tecnologies mencionades en aquest resum en una única aplicació.

Paraules clau: Realitat virtual, Immobiliàries, Unity, VR Multiusuari

Abstract

One of the challenges for any software application developer is to satisfy all the needs related to the user, increasingly demanding and adapted to the new technologies. This final work deals about the deployment and development of a virtual reality application online, offering a multiplayer system that allows to share a virtual reality environment, which is oriented in real estate designs. With this application, it is intended to provide a new way to interact, communicate, create and manage a virtual content for the users of the platform in a synchronized way. With the main target of bringing together the currently technology of the virtual reality along with a network environment and a database from which to store the information of the connection time of each user.

That said, the greatest challenge as a software developer is to unify all these technologies mentioned above in one single application.

Keywords: Virtual Reality, Real estate, Unity, VR Multiuser

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1 Objetivos

El objetivo perseguido con la elaboración del proyecto consiste en completar de manera satisfactoria el desarrollo de una aplicación de realidad virtual multiusuario para teléfonos móviles con sistema operativo *Android* que sirva para la exploración de diseños inmobiliarios en tres dimensiones, con el reto de integrar en la aplicación la realidad virtual junto con un sistema multiusuario y una base de datos externa.

Se trata de todo un reto personal con el que espero poner a prueba mis competencias específicas cursadas en el grado de ingeniería informática, pero con el que también se pretende evolucionar como profesional con la adquisición de nuevos conocimientos para llevar a cabo cada uno de los hitos expuestos a continuación:

- Desarrollar la aplicación en un entorno multiusuario.
- Integrar la tecnología de la realidad virtual en la aplicación.
- Implementar la interacción con el entorno.
- Implementación del movimiento.
- Implementación de modificaciones en tiempo real.
- Conseguir la captación de datos por medio de una base de datos externa con la cual obtener el tiempo de conexión de los usuarios.

Abordando los siguientes hitos se pretende proporcionar como resultado una aplicación que permita ofrecer al usuario la sensación de participación en la escena inmobiliaria con el resto de usuarios conectados, sincronizando sus acciones de manera que puedan explorar a lo largo de la escena, comunicarse, interactuar y editar contenido.

1.2 Antecedentes

Siempre me ha fascinado la velocidad con la que avanza la tecnología y esto no es algo para nada novedoso pues ya lo tuvo en consideración *Gordon E. Moore* el 19 de abril de 1965, día en el cual formuló su propia ley empírica muy conocida y estudiada en el ámbito de la informática anunciándola de la siguiente manera [1]:

"El número de transistores por unidad de superficie en circuitos integrados se duplica cada dos años"

Ley de Moore, 19 de abril de 1965

Desde mi punto de vista y mi manera de razonar considerada científica esta situación es asombrosa pues implica que el ser humano y el usuario estén constantemente en continua adaptación y evolución. Poco tiempo ha pasado desde que apareció el primer teléfono móvil a manos de *Martin Cooper* en el año 1973 [2], el primer enrutador IP por *Virginia Strazisar* durante el 1974 [3] y el primer computador personal fabricado por la compañía *IBM* llamado el *IBM PC* en 1981 [4].

Echada la vista atrás con la cita de unos pocos inventos de una gran lista, podemos observar a día de hoy cómo ha cambiado nuestra manera de utilizar la tecnología. Cómo hemos pasado de utilizar un teléfono móvil para realizar y recibir llamadas y mensajes de texto, a poder ser capaces de poder llevar en nuestro bolsillo un mini ordenador, capaz de indicarnos nuestra ubicación geográfica e incluso de poder ver y comunicarse en tiempo real con otra persona al otro lado del planeta a través de una videoconferencia.

El avance tecnológico es capaz de transformar por completo la sociedad tal y como la conocemos, es de esta situación de donde nace mi motivación para desarrollar mi proyecto y adentrarme con mis conocimientos adquiridos en el grado de ingeniería informática, en el despliegue de una aplicación basada en una tecnología de la que todavía se desconoce y de la que considero que tiene larga proyección y mucho que ofrecer.

Muchas empresas ya se están dando cuenta de esta situación y de la oportunidad de negocio que para ellas presenta la realidad virtual, ya sea para fines de captación y formación, creación de aplicaciones o marketing digital.

En el siguiente trabajo se utiliza la realidad virtual para desplegar un entorno en red sobre un diseño inmobiliario, en el cual se puedan conectar diferentes usuarios y puedan explorar cada uno de los rincones de la casa en tiempo real.

Cada usuario se instanciará sobre la misma casa y desde ese momento podrán comunicarse con el resto de usuarios a través de la transmisión por voz IP, explorando y compartiendo de forma cooperativa un hogar donde sumergir sus sentidos en 360 grados.

La aplicación va a estar conectada a una base de datos externa con la cual obtener la información del tiempo de conexión de cada usuario por medio de su fecha de entrada y su fecha de salida.

Finalizada la puesta en escena considero importante que en primer lugar se debe hacer una introducción sobre cuáles son las tecnologías y plataformas con las que se trabaja durante el desarrollo del proyecto, de manera que se profundice en los aspectos clave expuestos en la memoria para ayudar a la comprensión conforme avance la lectura.

1.3 Realidad virtual

1.3.1 ¿Qué es la realidad virtual y cómo se representa?

La tecnología de la realidad virtual o comúnmente conocida como *VR* del inglés *Virtual Reality* es una de las tecnologías donde el componente visual toma uno de los papeles más importantes, pues a través del uso de técnicas estereoscópicas y la computación, permite la simulación digital de un entorno gráfico tridimensional. Su principal objetivo con dicha representación gráfica es el de ofrecer al usuario una sensación de inmersión total en un mundo ficticio a través del cual estrechar la frontera entre lo real e irreal, facilitando la interacción e integración con el *software*. Como base para lograr el funcionamiento de la simulación son necesarios componentes *hardware* que actúen

como núcleo del sistema, procesando y administrando las distintas operaciones que ejecute el usuario en tiempo real. Por otro lado, el *software* es el encargado de reproducir dicha interfaz virtual.

Para su utilización y representación además del *software* y el *hardware* es necesario el uso de las conocidas lentes o gafas de realidad virtual que actúen como periférico externo.

Así pues, se puede decir que para poder disfrutar y hacer uso de esta tecnología se necesitan dos dispositivos esenciales que deben ir a conjunto, por un lado, el dispositivo informático, el cual puede ser desde un computador personal a un teléfono móvil y por otro lado las lentes o gafas de realidad virtual.

Figura 1: Probando las gafas de realidad virtual en laboratorio de la universidad.

Las lentes bifocales conocidas como las gafas de realidad virtual permiten aislar al usuario del mundo exterior y acércalo al entorno gráfico, esta interfaz representada para el usuario a través del visor, otorga una vista periférica de 360 grados abarcando todo el campo visual, que junto a los sensores de movimiento del dispositivo *hardware* es posible simular los movimientos de la cabeza que el usuario está ejecutando en tiempo real. De esta manera conforme el usuario dirige la mirada y explora alrededor de su entorno, los sensores controlan el movimiento de la escena.

Como en este tipo de tecnología el oído y la vista son los sentidos que más estímulos generan en el usuario, además de las gafas de realidad virtual se pueden utilizar a conjunto otros dispositivos que ofrezcan una mayor inmersión, como en el caso de entrada de audio a través de micrófono o salida por medio de los altavoces.

1.3.2 ¿Por qué he elegido utilizar la realidad virtual en mi proyecto?

Siempre me ha gustado la idea y el concepto de la realidad virtual y de cómo hacer posible la capacidad de satisfacer nuestras ideas y sueños a través de la computación digital. Y es que la programación en general y la habilidad de programar ofrecen la posibilidad de acercar la imaginación a la realidad, plasmando todo aquello que se nos pueda pasar por la cabeza a líneas de código que lo traduzcan en digital. Es por esa capacidad de estrechar la relación entre los conceptos "realidad" y "virtual" a través del *software* por lo que considero que esta es la tecnología idónea para ofrecer un entorno virtual enfocado a la exploración de diseños inmobiliarios, en el cual sumergir al usuario y ofrecerle un entorno los más realista posible con el que poder explorar e interactuar por los rincones del hogar.

Además, para responder a esta pregunta con fundamento es necesario entender cuál es el papel hoy en día de la realidad virtual en la sociedad y qué tendencias de uso le esperan en planes futuros, ya que se trata de una tecnología en crecimiento y por desarrollarse en diferentes ámbitos sociales y profesionales como los nombrados a continuación:

Medicina. La realidad virtual en ámbitos médicos está suponiendo un potencial de desarrollo importante en diversas áreas. Está demostrado que en el campo de la psicología clínica el uso de la realidad virtual puede ser de ayuda para el tratamiento y rehabilitación de ciertos trastornos psicológicos y mentales, así como derivados problemas de ansiedad. Esta tecnología se ha estado utilizado sobre pacientes ayudándoles a la superación de ciertas fobias a través de la exposición. Una exposición que viene dada mediante la generación de entornos virtuales controlados por el terapeuta, ofreciéndole al paciente un ambiente a través del cual poder explorar sin que haya consecuencias directas [5]. También se está haciendo uso de la tecnología para ayudar a la inhibición y control del dolor de los pacientes por medio de la distracción [6]. Por último, los estudiantes de medicina también están aplicando cambios en sus metodologías educativas y de formación con el uso de diversos entornos de realidad virtual [7].

Educación. En numerosos proyectos educativos se están utilizando entornos de realidad virtual para el aprendizaje, simplificando los procesos de enseñanza y acercando al usuario contenidos académicos con el fin de despertar el interés y ayudar a conseguir asentar el conocimiento a través de experiencias sensoriales. Pudiendo servir desde la ayuda para personas con necesidades especiales, hasta para estrategias formativas en centros de estudiantes de ingeniería, ya sea para mejorar la enseñanza en conceptos complejos y abstractos, o incluso situaciones que puedan presentar un riesgo para el estudiante, como en escenarios de prácticas de laboratorio [8].

Marketing digital. El sector empresarial no deja de lado la realidad virtual para conseguir la captación del público a través del contenido multimedia, cada vez son más las empresas que utilizan este tipo de tecnología para la generación de contenidos publicitarios específicos para el usuario, consiguiendo despertar un mayor interés del consumidor con la marca de la empresa y el producto, aprovechando la realidad virtual como nuevo entorno comunicativo [9].

Ocio y entretenimiento. La realidad virtual se ha asentado principalmente en este sector y actualmente continua en crecimiento dentro del mundo del entretenimiento, sobretodo en la industria de los videojuegos. Ofreciendo la posibilidad de entretener al usuario de un modo diferente al que estamos acostumbrados. No obstante, me gustaría alejarme del concepto de estrechar la relación entre la realidad virtual y el ocio únicamente para el sector de los videojuegos. Como por ejemplo en el caso de poder experimentar situaciones virtuales a través de contenidos de vídeo como si estuviéramos en una inmersión cinematográfica en 360 grados. Pero sobretodo en el que he decidido adentrarme en el siguiente proyecto a través del mundo del interiorismo.

1.3.3 Principales compañías desarrolladoras de realidad virtual en la actualidad

COMPAÑÍA	DESCRIPCIÓN	GAFAS DE REALIDAD VIRTUAL	DISPONIBLES EN
Google	Con el objetivo de acercar la realidad virtual al usuario, *Google* lanzó la idea de ofrecer un primer visor lo más económico posible llamado *Google CardBoard*, donde su material principal es de cartón reciclable. Recientemente ha puesto en venta su nuevo modelo *Google Daydream* enfocado a utilidades más exigentes.	 *Figura 2: Gafas Google Cardboard.*  *Figura 3: Gafas Google Daydream.*	*Google CardBoard:* https://vr.google.com/intl/es_es/cardboard/developers/ *Google Daydream:* https://vr.google.com/daydream/
Samsung	Samsung a través del visor *Gear VR* ofrece un visor idóneo para iniciarse en la tecnología de la realidad virtual	 *Figura 4: Gafas Gear VR.*	*Gear VR:* https://www.samsung.com/global/galaxy/gear-vr/
Facebook	La empresa conocida por su red social compró en el año 2015 el proyecto *Oculus*, consiguiendo de esta manera abarcar un futuro virtual con uno de los visores de realidad virtual más conocidos actualmente llamado *Oculus Rift*.	 *Figura 5: Gafas Oculus Rift.*	*Oculus Rift:* https://www.oculus.com/rift-s/
HTC	A través del dispositivo *VIVE* ofrece al usuario unas potentes gafas de realidad virtual.	 *Figura 6: Gafas VIVE.*	*VIVE:* https://www.vive.com/mx/
Sony	Enfocada principalmente en el desarrollo de videojuegos para su plataforma *Playstation 4* ha puesto en venta su propuesta de realidad virtual a través del producto *Playstation VR*.	 *Figura 7: Gafas PlayStation VR.*	*PlayStation VR:* https://www.playstation.com/es-es/explore/playstation-vr/

Tabla 1: Tabla resumen principales compañías desarrolladoras de realidad virtual en la actualidad.(http://mundo-virtual.com/que-es-la-realidad-virtual/)

1.3.4 ¿Hacía que tipo de plataforma está focalizado mi proyecto de realidad virtual?

La aplicación desarrollada en el siguiente trabajo está principalmente orientada para despliegues en dispositivos móviles con el sistema operativo *Android* a través del kit de desarrollo de *software* que ofrece la compañía *Google* llamado *Google VR SDK*, el cual ofrece una *API* completa para el desarrollo de aplicaciones *Android* basadas en las funcionalidades para *Google Daydream* y *Google Cardboard.* Esta elección ha sido tomada principalmente por el hecho de que actualmente es muy elevado el número de personas que poseen un teléfono móvil con el sistema operativo *Android* y con la tecnología suficientemente avanzada como para procesar y reproducir este tipo de contenido. Además, conseguir desplegar y ejecutar el *software* de realidad virtual con un realístico entorno inmobiliario es un éxito y una manera de acercar su usabilidad y portabilidad al usuario.

La gráfica incluida a continuación muestra un reflejo interesante en cuanto al porcentaje de ventas al año por sistema operativo en el mundo durante el pasado 2018, destacando por encima de todas, las ventas de dispositivos con sistema operativo *Android*:

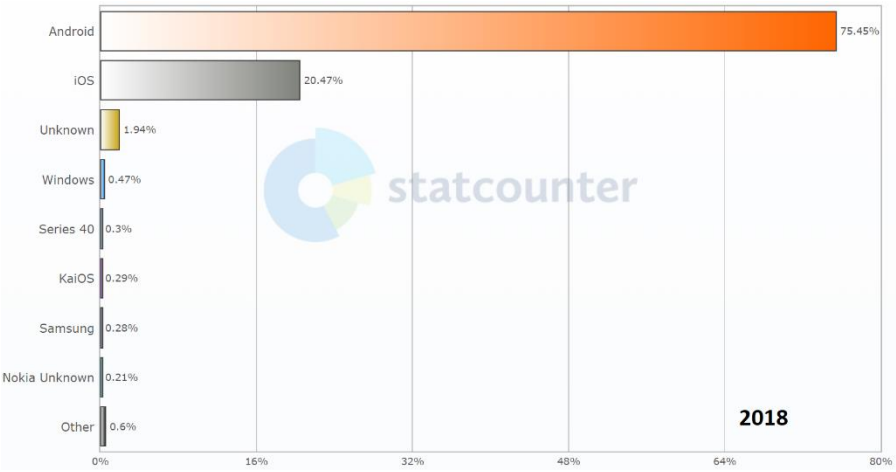


Figura 8: Porcentaje de ventas anuales de teléfonos móviles por sistema operativo en el mundo en 2018.(http://gs.statcounter.com/os-market-share/mobile/worldwide)

Como se puede observar en 2018 la cuota de mercado de dispositivos móviles en los usuarios pertenece en su gran mayoría al sistema operativo *Android*, donde abarcó un 75,45% de ventas de los terminales, por encima de su más cercano competidor IOS con un 20,47%.

1.4 Unity

1.4.1 ¿Qué es Unity?

Unity es una de los motores de desarrollo de videojuegos más utilizados actualmente dentro del ámbito 2D como 3D, creado por *Unity Technologies*, este entorno dispone de una versión básica la cual es gratuita y otra versión de pago a través de la cual se pueden obtener funcionalidades para desarrollos más avanzados. No obstante, con la versión gratuita se dispone de las herramientas suficientes para el desarrollo de aplicaciones. Así pues, *Unity* se convierte en un motor excepcional que combina diseño y programación para la obtención y desarrollo de entornos gráficos con la capacidad de compilarlos y desplegarlos en más de 25 plataformas distintas, entre las cuales se encuentran: *Android*, *iOS*, *Windows*, *Mac OS*, *Linux* y *PlayStation 4* [10].

Figura 9: Fotografía del logotipo de Unity.

1.4.2 Estructura del editor y ventanas de trabajo

Ventana de Jerarquía

Esta ventana contiene cada *GameObject* de la escena con sus respectivas relaciones de jerarquía entre objetos de padres a hijos y herencia.

Ventana de Escena

Los *GameObjects* son posicionados de manera interactiva sobre el escenario, la principal función de esta ventana es la de poder seleccionar y posicionar los elementos de la ventana de jerarquía, de manera que puedan ser pre-visualizados y editados en tamaño, escala y posición.

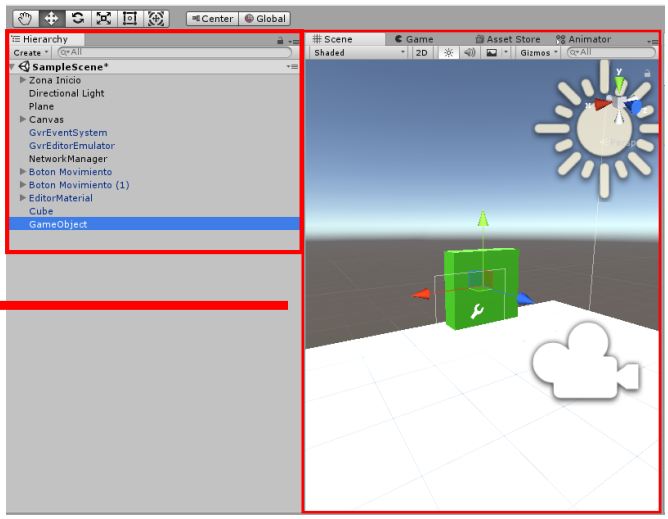


Figura 10: Ventanas Jerarquía y Escena del editor Unity.

Ventana Inspector

Contiene cada uno de los componentes asignados al *GameObject*, desde aquí pueden añadirse y ser editados los componentes del mismo y el caso de la asignación de los *scripts*, acceder a sus variables públicas, de modo que puedan ser establecidos valores por defecto sin necesidad de reescribir el código de programación.

Ventana Proyecto

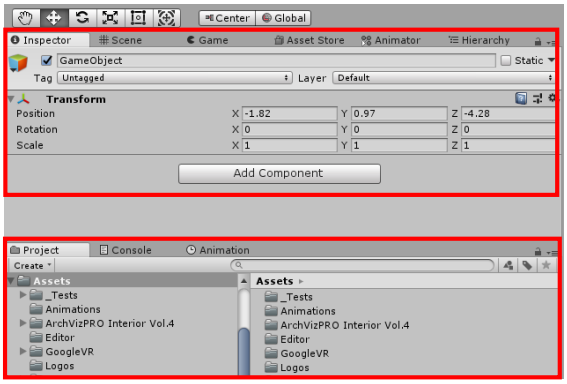


Figura 11: Ventanas Inspector y Proyecto del editor Unity.

La estructura de directorios del proyecto y los recursos del mismo se encuentra accesible desde esta ventana, desde la que es posible acceder a cada uno de los elementos con los que se trabaja en el proyecto conocidos en *Unity* bajo el nombre de *assets*. Donde es recomendable hacer una clasificación de los directorios en función del tipo de contenido almacenado, donde se incluyen las escenas del proyecto, *scripts*, clips de sonido, componentes de diseño 3D (animaciones, materiales, mayas) y los *prefabs*.

Los *prefabs* son plantillas a partir de las cuales es posible almacenar el estado de un *GameObject* con valores de componentes y propiedades determinadas, de modo que a lo largo del proyecto sea posible generar prototipos que puedan ser utilizados e instanciados en la escena con estos valores [11].

Ventana de Juego

Esta ventana del motor nos permite simular la representación de la escena a partir de la vista de la cámara principal, de modo que se pueda apreciar en el momento de edición la vista con la que el usuario va a interactuar durante la ejecución de la aplicación. Para poner en marcha esta ventana se hace uso del botón situado en parte superior de la barra de control llamado *Play Mode*, mientras este botón se encuentre activo, cualquier cambio que se produzca sobre el proyecto será temporal y volverá a restablecerse en el momento que se detenga el *Play Mode*. De esta manera *Unity* permite simulaciones de edición en tiempo de ejecución [12].

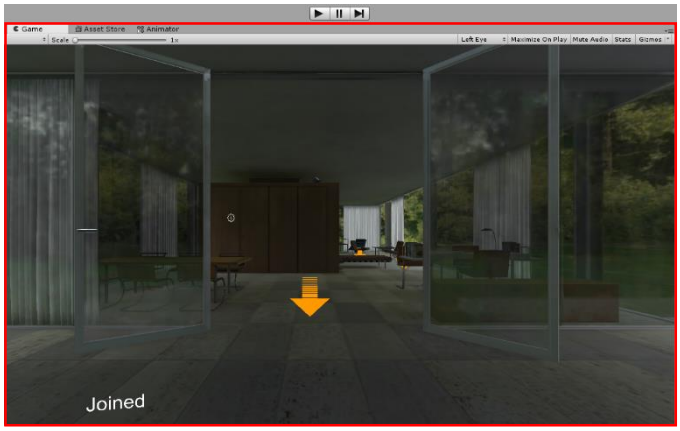


Figura 12: Ventana de Juego del editor Unity.

1.4.3 Unity y la programación

Durante mucho tiempo *Unity* ha integrado de forma nativa el *sripting* por parte de *MonoDevelop*, el cual ha sido su entorno de desarrollo por excelencia diseñado para dar soporte en el lenguaje de programación *C#* (*C-sharp*), *Boo* y *.NET*. Sin embargo, a partir de la versión 2018.1, *Unity* incluye *Visual Studio* para *macOS* en lugar de *MonoDevelop*. En *Windows*, *Unity* también deja de lado utilizar *MonoDevelop* para situar *Visual Studio 2017 Community* como el editor de *scripts* predeterminado de *C#*.

Dado que *Unity* está basando en lenguajes de programación orientados a objetos, este fundamenta la mecánica de la programación en ciertos elementos básicos de construcción, los cuales se describen a continuación:

- ***GameObjects***: Es uno de los conceptos más importantes en el editor de *Unity*, ya que todo contenido que forme parte de la escena del proyecto debe de ser tratado como un *GameObject*, ya sean personajes, luces, efectos especiales y cámaras. Los *GameObjects* no interactúan de ninguna manera con el motor de juego por si solos, para darles vida es necesario agregarle los *componentes* [13].

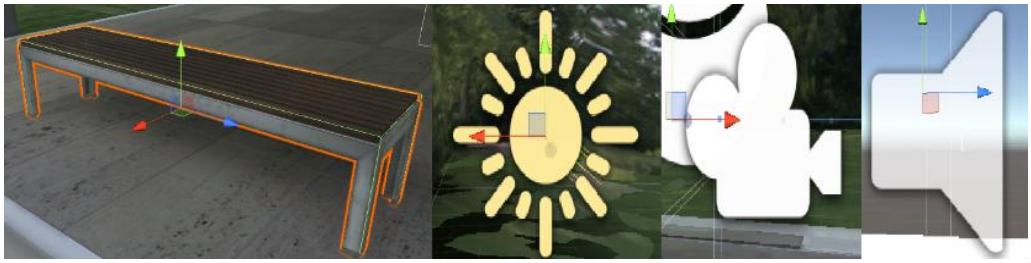


Figura 13: Fotografía de distintos tipos de GameObjects.

- ***Componentes***: Los *componentes* son conectados a los *GameObjects*, definiendo y controlando el comportamiento de los mismos a través de los *scripts*. Utilizando *scripts* puedes implementar tu propia lógica y comportamiento en la escena, es ahí donde interviene realmente la magia de la programación orientada a objetos, a través de la cual se nos permite que nuestra aplicación *software* cobre vida, enlazando el código con la mecánica de la aplicación [14].

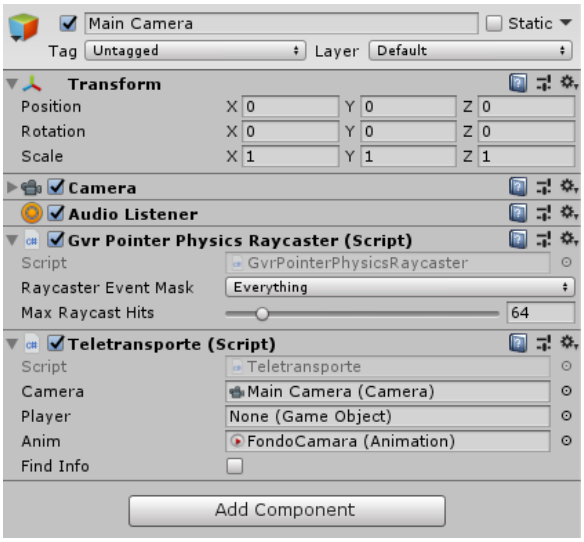


Figura 14: Ejemplo de componentes añadidos al GameObject Main Camera.

Esqueleto básico de los scripts en Unity:

Para poder profundizar más adelante en los *scripts* generados a lo largo del trabajo es necesario entender cuál es su anotomía básica en *C#*:

Esqueleto básico

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

public class NewScript : MonoBehaviour
{

	// Use this for initialization
	void Start () {

	}

	// Update is called once per frame
	void Update () {

	}
}
```

Todo nuevo *script* creado desde *Unity* deriva de la clase llamada *MonoBehaviour*, de modo que cada vez que se inserte un componente del tipo *script* a un *GameObject*, se crea una nueva instancia del objeto. El nombre de la clase viene definido por el introducido en la creación del *script*. Hay dos funciones fundamentales dentro de la clase, la función *Start()* la cual va a ser llamada en la inicialización de la aplicación y la función *Update()* que es llamada por cada refresco por segundo de ejecución [15].

Tabla 2: Esqueleto básico de un script en Unity.

1.5 Photon Unity Networking

Photon Unity Networking (*PUN*) va ser la herramienta encargada de dar soporte a la aplicación en un entorno en red cooperativo a través del *asset* gratuito proporcionado en la tienda de *Unity*, de la mano de *Photon Realtime*, se pone a disposición un servidor con la posibilidad de poder conectar de manera simultánea hasta 20 clientes en una misma sala, con la capacidad de sincronizar los objetos de la misma por medio de la *API* que proporciona una gran variedad de métodos, funciones y llamadas a procedimientos remotos (*PunRPC*). Toda la lógica de conexión está basada en la arquitectura cliente-servidor y es programada desde los *scripts* integrados en *Unity*. Todos los *SDK* de los clientes pueden interactuar entre sí sin importar el tipo, transfiriendo mensajes de red de forma síncrona entre *Android*, *Windows*, *macOS*, *iOS* y entornos web [16].

¿Por qué utilizo esta herramienta?

Unity ofrece de forma nativa su propio entorno de red interna conocido como *UNet* para conexiones de aplicaciones a través de internet *peer-to-peer*, sin embargo, no ofrece buenas condiciones de latencia y de conexión comparándolas con los servicios y ventajas que ofrece *Photon Network*, que proporciona buena latencia y un menor consumo de recursos. En *UNet* si un cliente se desconecta de la aplicación, la conexión ha terminado para el resto de clientes, sin embargo, *PUN* utiliza servidores dedicados de modo que la aplicación puede continuar independientemente de la desconexión de sus clientes, ofreciendo variedad de métodos para la reconexión y actualización de los clientes en el servidor.

De este modo con la utilización de *PUN* en mi proyecto va a ser posible la conexión en red de los diferentes usuarios instanciados en la escena inmobiliaria, de modo que se pueda obtener una completa comunicación entre cada uno de los clientes gracias al servidor, obteniendo la información y sincronización de componentes de la escena en tiempo de ejecución. Además, se le va a permitir la conexión al servidor *PUNVoice* para que proporcione a los usuarios un chat de voz de alta calidad mediante la transmisión de voz *IP*, el cual actuara como componente ideal en la comunicación de los usuarios adaptado perfectamente a la aplicación de realidad virtual.

2. PLANIFICACIÓN Y SECUENCIACIÓN DE TAREAS PREVIAS AL DESARROLLO

Como paso previo a comenzar con el despliegue y desarrollo de la aplicación ha sido necesario realizar un estudio de la viabilidad del proyecto, que implique la superación de los objetivos principales que se pretendían alcanzar para poder continuar con el desarrollo del mismo. En este capítulo de la memoria se hace reflejo de los pasos previos y objetivos establecidos como prueba que ha sido necesario fijar y superar para poder abordar el proyecto.

2.1 Diagrama de procesos

El siguiente diagrama de procesos define la planificación de las tareas a llevar a cabo durante la elaboración del proyecto:

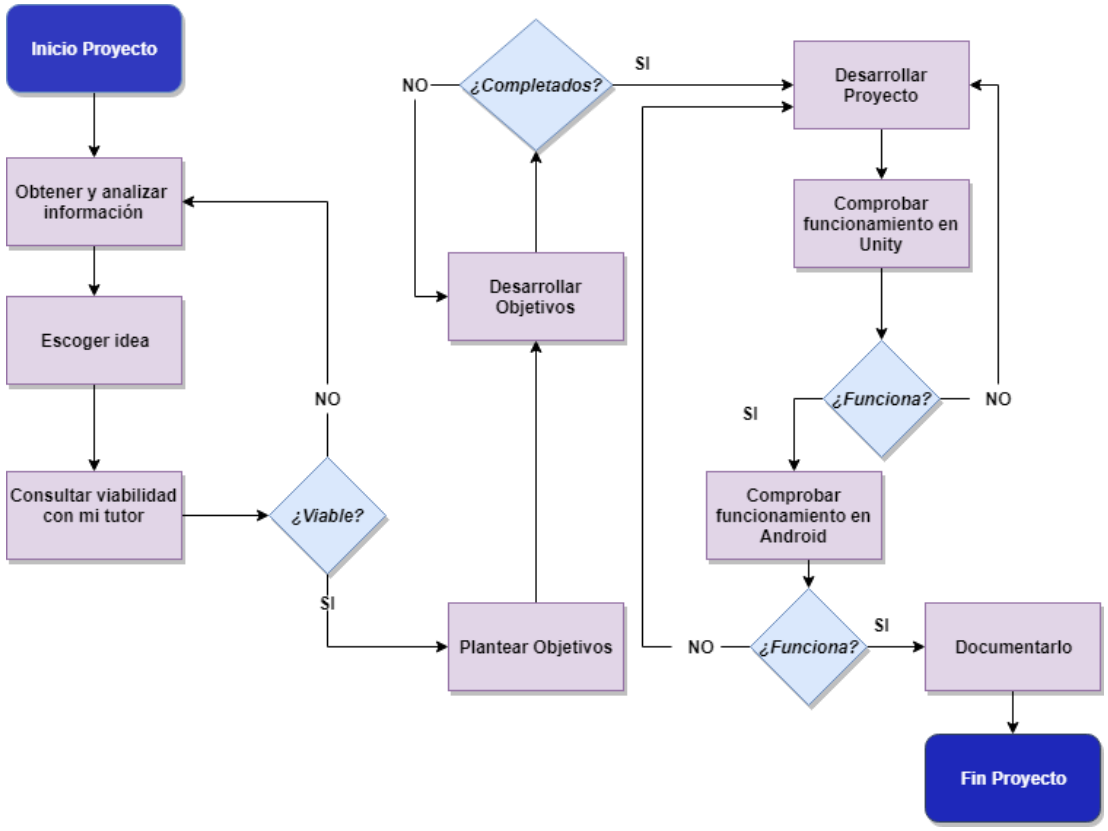


Figura 15: Diagrama de procesos.

2.2 Conseguir instanciar usuarios en red

El primer paso una vez instalado el motor de videojuegos y el más importante del proyecto, es conseguir ser capaz de instanciar y conectar diferentes usuarios en una misma escena, de modo que cuando se ejecute la aplicación se instancie el usuario que tomara el papel de usuario local y a medida que se vayan conectando diferentes usuarios a través de otras plataformas sean percibidos por el mismo.

Para el desarrollo del siguiente objetivo en primer lugar ha sido necesario importar el *asset* de *PUN* en el proyecto de prueba, la versión utilizada ha sido la v1.97. En el momento que se importan todos los paquetes en el proyecto, se ejecuta el asistente de *PUN* donde se solicita que se ingrese el *ID* del servidor al que se quiere conectar, a continuación, es el momento de configurar el servidor para que pueda establecerse la conexión a través de *Unity*. Para ello ha sido necesario registrarse en el siguiente enlace: https://dashboard.photonengine.com/en-US/account/signin y acceder al portal de *PhotonEngine* a través del cual se pueden administrar cada uno de los servidores. Se crea el servidor y se genera su *ID* para poder ser introducido en el asistente de *PUN*.

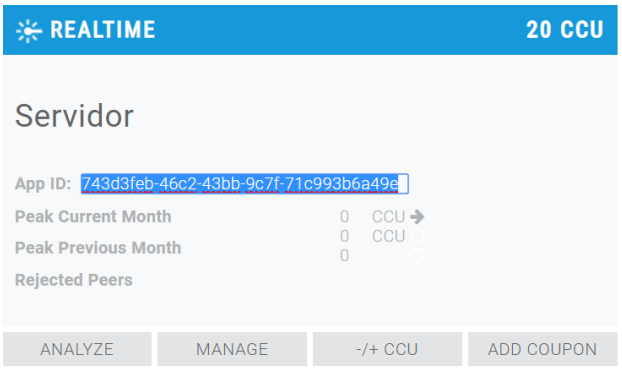


Figura 16: Servidor utilizado en la prueba de instanciar usuarios en red.

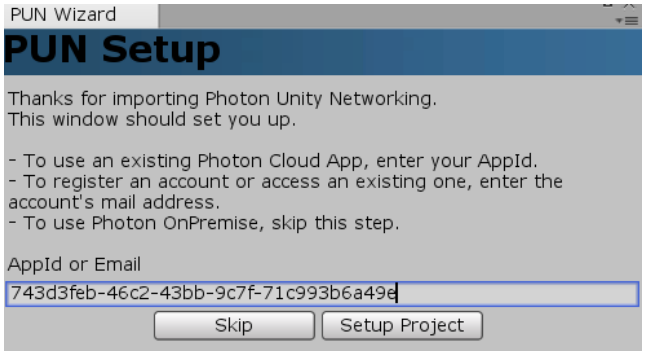


Figura 17: Asistente de configuración de PUN durante la prueba.

Realizada la identificación del servidor, el asistente almacena la información en el fichero *PhotonServerSettings* para que pueda ser solicitada en los *scripts* que se generen en futuras conexiones al servidor.

El siguiente paso consiste en la preparación del escenario de prueba, para ello se ha decido elaborar un escenario sencillo con un plano que actúe como suelo de la escena y un personaje en forma de capsula que servirá para cada uno de los usuarios conectados.

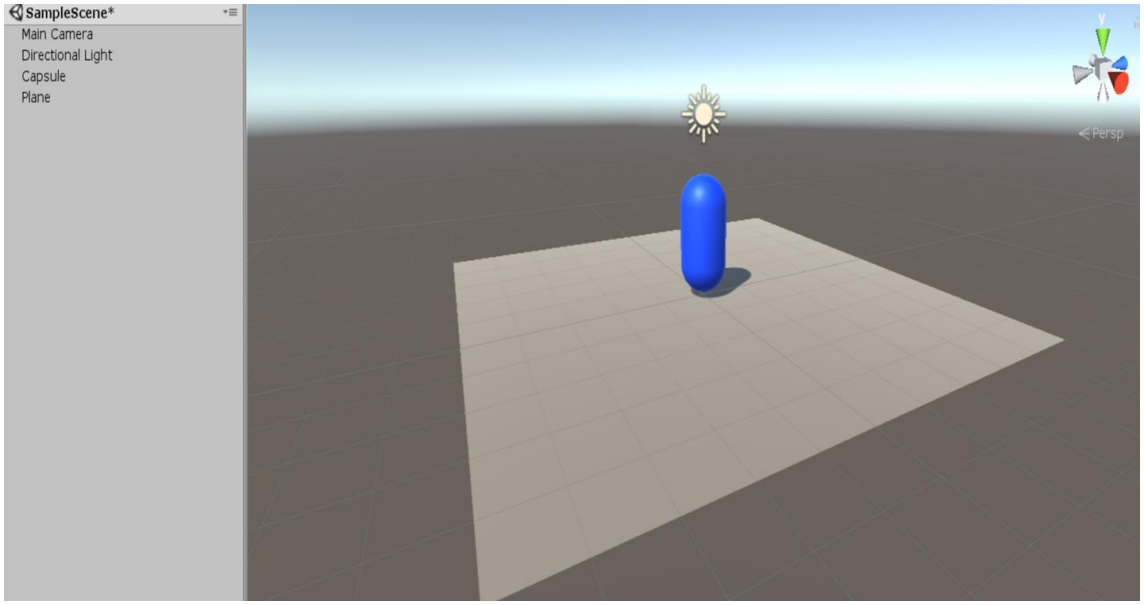


Figura 18: Escenario de prueba para conseguir instanciar usuarios en red.

A continuación, es necesario incluir en la ventana de jerarquía un *GameObject* el cual debe contener el *script* que sirva para realizar la conexión al servidor en el momento que se ejecute la aplicación. Para ello se ha incluido el *GameObject* en la escena y ha sido llamado *PruebaConexionAlServidor*, de momento se trata de un objeto vacío, pero es necesario la generación del primer *script*. Este *script* va ser el encargado de administrar la conexión al servidor por cada uno de los usuarios que se vayan conectando de manera remota.

Una vez generado el *script* se le asigna al objeto vacío *PruebaConexionAlServidor* y se le introducen los valores de las variables públicas *Version* y *TextoDeConexion*.

Para el caso de *TextoDeConexion* se ha añadido en la ventana jerarquía un *GameObject* del tipo *Text*, el cual ha sido ajustado en la posición inferior de la cámara. De este modo el estado de conexión podrá ser visualizado en todo momento por el usuario.

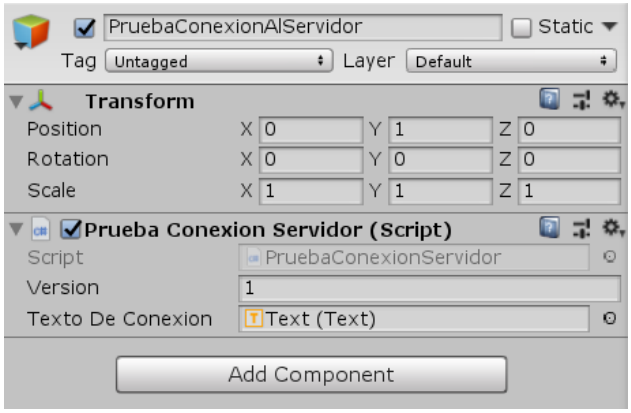


Figura 19: Componentes del objeto ConexionAlServidor.

El siguiente paso a llevar a cabo consiste en la generación del *script* que controle la lógica del usuario instanciado. En el caso de prueba únicamente interesa diferenciar entre cual es el usuario local en la escena y cuál es el global.

PruebaUsuarioConectado.cs

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

public class PruebaUsuarioConectado : Photon.MonoBehaviour {

    public GameObject _Usuario;

       void Start () {
        if (photonView.isMine)
        {
            gameObject.name = "Usuario Local";
        }
       }
}
```

Objetivo del código: la funcionalidad del siguiente *script* es únicamente la de pasarle como parámetro otro nombre al objeto *_Usuario* si se cumple la condición, de modo que cada vez que se instancie un usuario localmente en la sala del servidor, su nombre cambiará de *Usuario Global* a *Usuario Local*. Con el objetivo de poder diferenciar cual es el usuario local.

Tabla 3: Explicación del script PruebaUsuarioConectado.

Se añade el *script PruebaUsuarioConectado.cs* al *GameObject Usuario Global* y se genera un *prefab* en un nuevo directorio llamado *Resources*. Además, es necesario añadirle el componente *PhotonView* para que el *GameObject Usuario Global* pueda ser sincronizado en red.

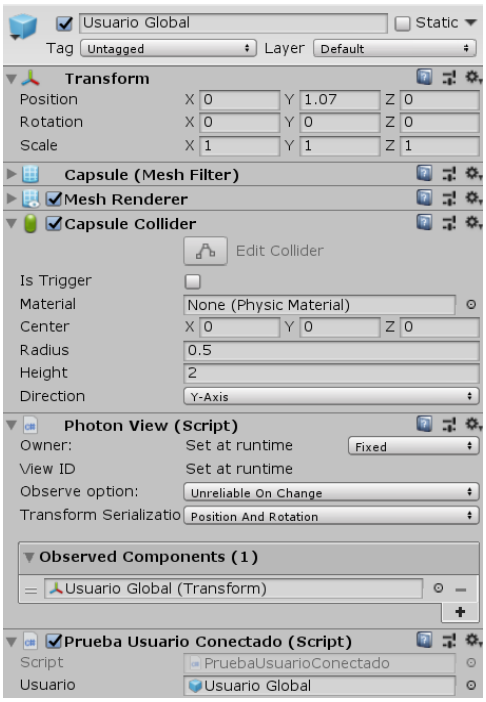


Figura 20: Componentes del GameObject Usuario Global.

En la siguiente figura se puede observar como finalmente se ha cumplido con éxito el objetivo de instanciar usuarios en red, ya que son instanciados de la manera correcta y cumpliendo con las condiciones de los *scripts* generados durante la prueba. Conectando en una misma sala diferentes usuarios.

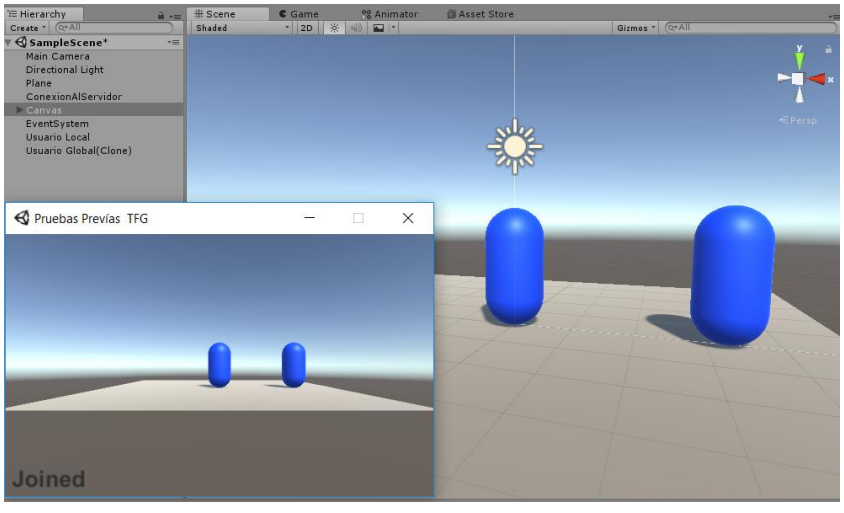


Figura 21: Usuarios conectados en red con éxito.

3. DESARROLLO DEL PROYECTO

Este capítulo es el cuerpo de la memoria y donde se va explicar con detalle cada uno de los apartados elaborados minuciosamente para completar el despliegue y desarrollo de una aplicación multiusuario, focalizada en la exploración de diseños inmobiliarios en tres dimensiones.

3.1 Preparación del entorno de trabajo

Todo el desarrollo del proyecto se va a llevar a cabo en mi ordenador personal de trabajo, en mi caso dispongo de un portátil ligero con limitaciones gráficas de la marca *Lenovo* y sistema operativo *Windows 10* de 64-bits, con las siguientes especificaciones:

Figura 22: Especificaciones de mi equipo de trabajo.

3.1.1 Instalación del motor Unity

Para poder empezar a trabajar en el proyecto ha sido necesario instalar en el equipo el motor de desarrollo de *Unity*, en mi caso he instalado la versión de *Unity 2018.2.12f1*, la cual era la última versión en el momento de la instalación, pero la actualización de versiones de *Unity* se realiza en cortos periodos de tiempo y actualmente la última versión disponible para descargar es la versión *Unity 2019.1.6*, esta actualización se puede obtener desde el 10 de junio de 2019, no obstante, si se desea descargar una versión de *Unity* en particular se puede acceder al siguiente enlace https://unity3d.com/es/get-unity/download/archive y obtener la deseada del repositorio.

Completados los pasos del proceso de instalación de *Unity* en el ordenador, es momento de ejecutar la aplicación y comenzar con la creación del proyecto:

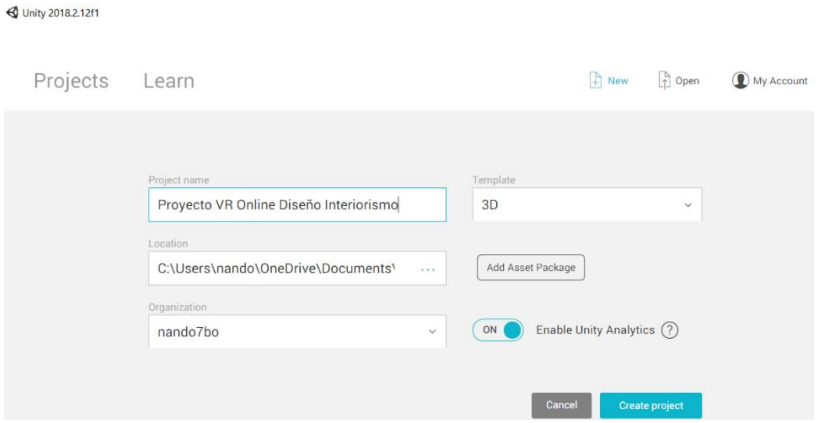


Figura 23: Creación del proyecto 3D.

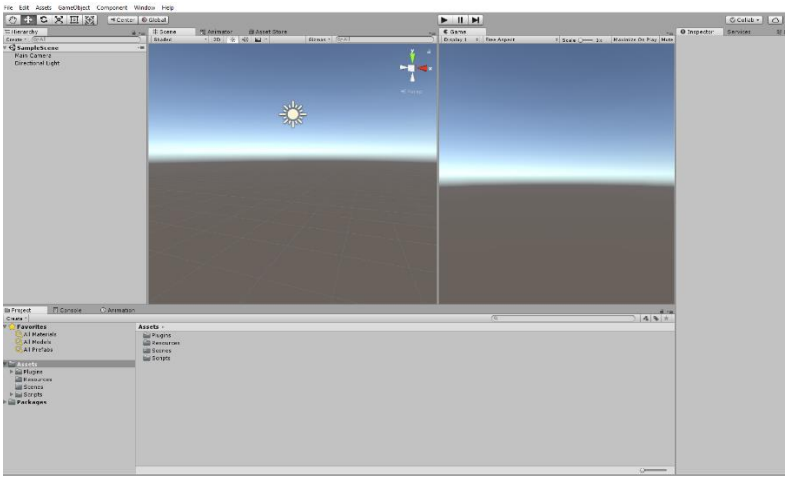


Figura 24: Proyecto creado.

3.1.2 Instalación de Android SDK y JDK

Para empezar con la configuración de trabajo en dispositivos móviles es necesario configurar el equipo con la instalación del *SDK* de *Android* y el *Java Development Kit* (*JDK*), de esta manera se podrán realizar las compilaciones desde *Unity* a dispositivos móviles con el sistema operativo *Android*, si no se realiza este paso no va a ser posible compilar la aplicación en *Android*.

He comenzado con la instalación de *Android Studio* para mi equipo *Windows 10* (de 64bits), necesario para configurar el *SDK* de *Android*, disponible en el siguiente enlace: https://developer.android.com/studio

Platform	Android Studio package	Size
Windows (64-bit)	android-studio-ide-183.5522156-windows.exe Recommended	971 MB
	android-studio-ide-183.5522156-windows.zip No .exe installer	1035 MB
Windows (32-bit)	android-studio-ide-183.5522156-windows32.zip No .exe installer	1035 MB
Mac (64-bit)	android-studio-ide-183.5522156-mac.dmg	1026 MB
Linux (64-bit)	android-studio-ide-183.5522156-linux.tar.gz	1037 MB

Figura 25: Descargables de Android Studio.

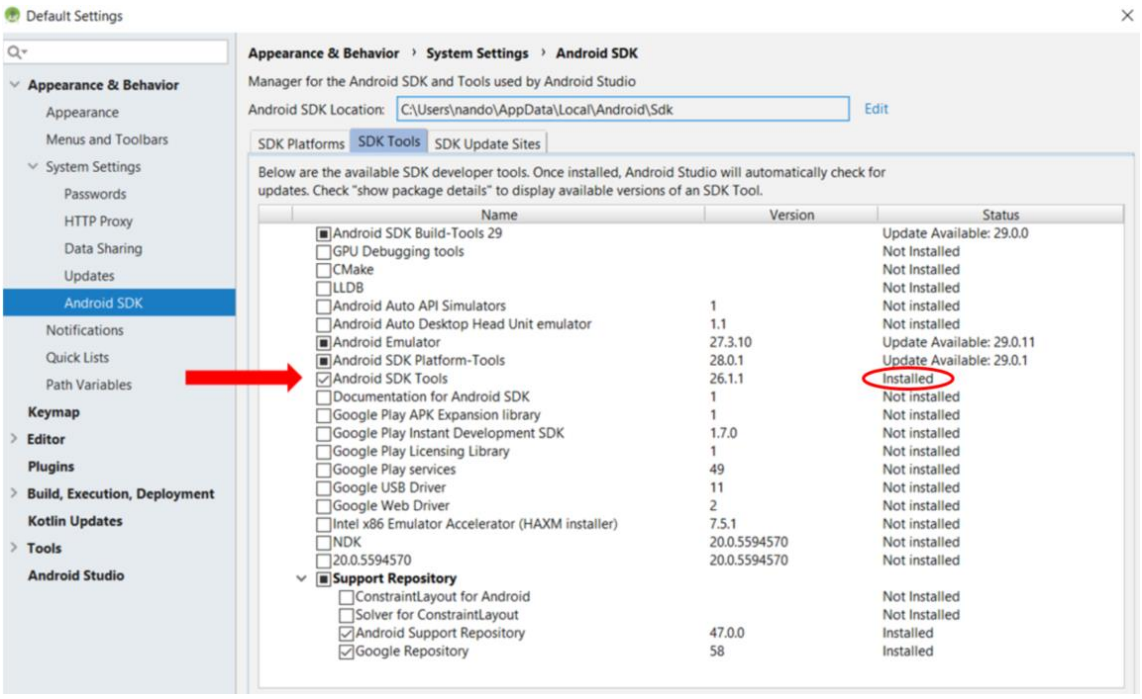


Figura 26: Android SDK instalado.

Una vez instalado el *SDK* de *Android*, se procede con la instalación de la última versión disponible en la fecha de instalación del *Java Delevopment Kit* en su versión *jdk1.8.0_191*, como se puede observar en la siguiente figura:

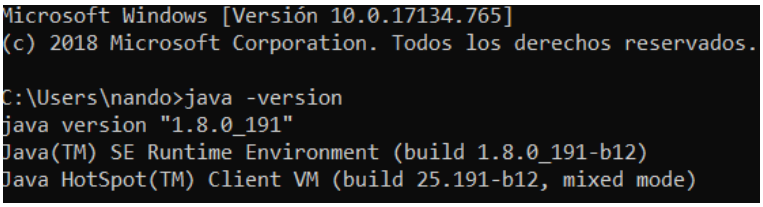


Figura 27: Versión Java Development Kit.

El siguiente paso consiste en indicarle a *Unity* las rutas en las que se encuentran el *SDK* de *Android* y el *JDK* de *Java*, para ello se accede desde la ventana *Edit >> Preferences >>External Tools* y se introducen las rutas a los directorios:

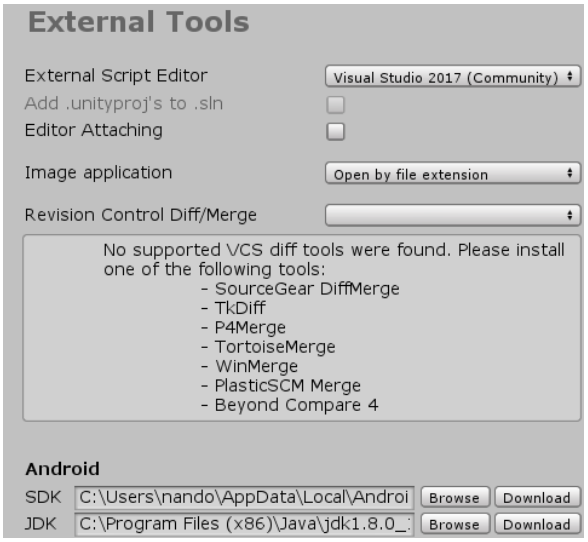


Figura 28: External Tools.

3.1.3 Configuración final para la compilación en Android

A continuación, es el momento de realizar los ajustes desde el proyecto para configurar la compilación en mi dispositivo *Android*, para ello se le indica el nivel de *API* mínimo o versión mínima de *Android* requerida por la aplicación. Para despliegues que vayan a utilizar *Google CardBoard* y *Google Daydream* el *API* mínimo compatible es el 21 (*Lollipop*). En mi caso he utilizado el *API* 22 (*Lollipop*) para versiones de *Android 5.1*.

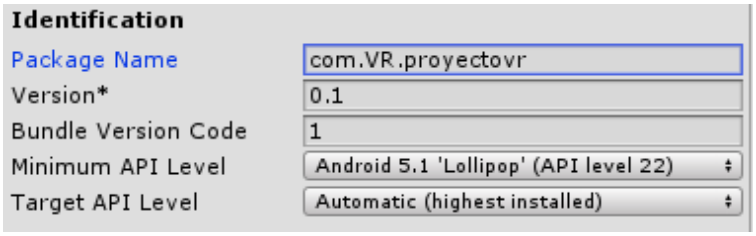


Figura 29: API mínimo de la aplicación.

Continuo con la conexión de mi dispositivo móvil por *USB 3.0* al ordenador, es importante habilitar la opción de *Depuración USB* en el teléfono móvil para permitir compilaciones desde el ordenador:

Figura 30: Opciones de desarrollador del teléfono móvil.

Posteriormente, verifico que es detectado correctamente a través del *SDK* de *Android* y *Unity*. Como se puede observar el entorno de trabajo ya está listo para comenzar a trabajar en *Android*:

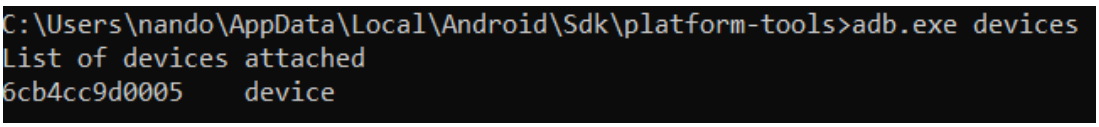


Figura 31: Conexión del teléfono móvil al ordenador.

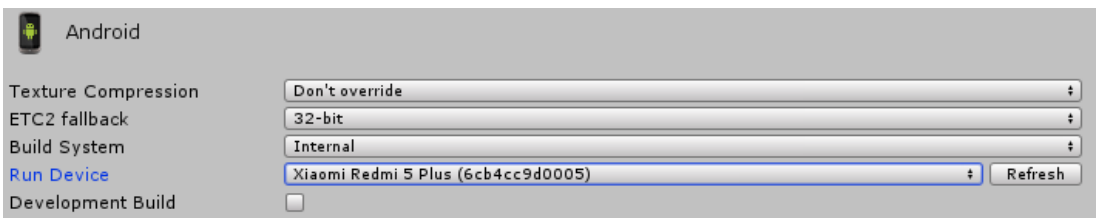


Figura 32: Teléfono móvil listo para compilar desde Unity.

3.1.4 Configuración del entorno para trabajar en realidad virtual

A continuación, se detallan los pasos realizados para poder configurar el proyecto de manera que se comience a trabajar desde un entorno de realidad virtual en dispositivos *Android*.

En primer lugar, es necesario descargar el paquete *Google VR SDK* para *Unity*, en mi caso he utilizado la última versión de las disponibles llamada *GoogleVRForUnity_1.200.0*, *Google* pone a disposición de manera gratuita todas sus versiones accesibles online a través del siguiente enlace: https://github.com/googlevr/gvr-unity-sdk/releases , una vez descargado el paquete es necesario importar la *API* de *Google* en el proyecto de *Unity*.

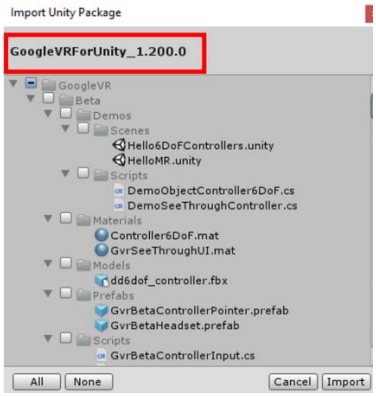


Figura 33: GoogleVRForUnity_1.200.0.

Toda la lógica de visualización de la escena va a correr a manos de la cámara principal, la cual va a ser parte fundamental para el desarrollo en realidad virtual. Es necesario habilitar desde la configuración de *Unity* la opción que permita visualizar contenido en realidad virtual a través del *XR Settings* e incluir a continuación el *SDK GoogleVRForUnity_1.200.0* descargado, desde ese momento cada vez que la aplicación se ejecute desde un dispositivo *Android* el contenido se reproducirá en realidad virtual.

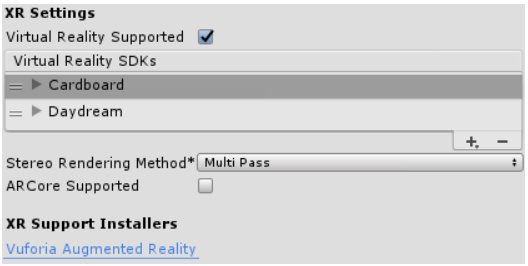


Figura 34: Habilitar contenido en realidad virtual por el SDK.

El proyecto ya está listo para poder trabajar con esta tecnología por lo que se realiza una primera compilación en el dispositivo *Android* para comprobar que el *SDK* de *Google* importado en el proyecto de *Unity* funciona correctamente:

Figura 35: Visor de realidad virtual desde dispositivo Android.

Como se puede observar en el momento que se ejecuta la aplicación desde el dispositivo móvil, se habilita el visor para la reproducción de contenido en realidad virtual.

3.1.5 Ajuste del modelo inmobiliario 3D a mi proyecto

El modelo inmobiliario utilizado durante el desarrollo del proyecto se implementa utilizando como base un *asset* en tres dimensiones que ofrece un entorno con un alto nivel de realismo a mi proyecto, el *asset* denominado *ArchVizPRO Interior.vol4* se puede obtener a través de la tienda de *Unity* accesible desde el entorno de trabajo. Este *asset* proporciona un modelo en tres dimensiones de una casa futurista que va a ser utilizada para desarrollar la mecánica de mi proyecto, con el principal objetivo de ofrecer el entorno explorable en realidad virtual para conexión de los usuarios conectados en red.

Una vez el entorno de trabajo ya está listo para poder trabajar en realidad virtual y en mi dispositivo móvil, es necesario realizar los ajustes necesarios para la escena del modelo inmobiliario ya que se trata de una escena con altos niveles gráficos de realismo e iluminación, por lo tanto, de no ser ajustada no va a ser posible compilarla desde un dispositivo móvil.

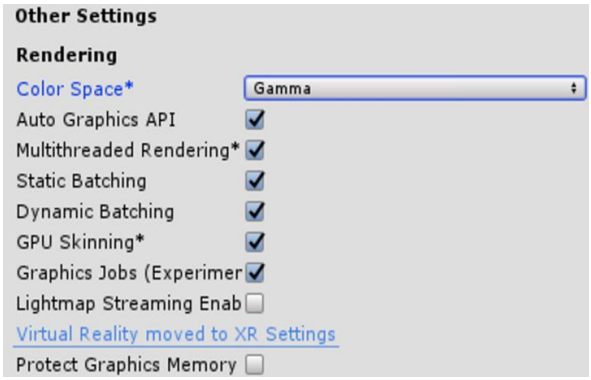


Figura 36: Ajuste para Android 1.

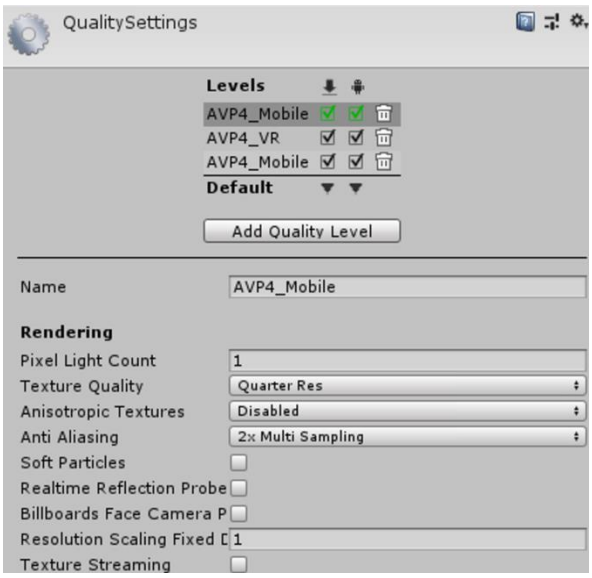


Figura 37: Ajuste para Android 2.

3.2 Configuración y conexión del servidor

Una vez completada la preparación del entorno de trabajo para utilizarse en *Android* a través de la realidad virtual, se incluye el contenido elaborado para conseguir la conexión online de los usuarios en la aplicación. De modo que ofrezca un servicio cooperativo que permita la comunicación y sincronización de los mismos a través del paso de mensajes entre cliente y servidor.

El primer paso para la implementación del modelo de red del proyecto consiste en la creación y configuración del servidor, el inicio de este proceso se realiza con un procedimiento similar al que se ha detallado en el apartado 2.2 de esta memoria (donde se realiza la prueba para la conexión de distintos usuarios en red). El proceso se lleva a cabo importando el paquete *Photon Unity Networking* (*PUN*) disponible desde *Unity* al proyecto de la aplicación, la versión utilizada es la v1.97, el paquete proporcionado por *Unity* llamado *PUN* va a servir para implementar la comunicación de los usuarios permitiendo desplegar la aplicación en un entorno cooperativo.

Una vez importado el paquete *PUN* al proyecto en *Unity*, se lanza automáticamente el asistente de *PUN* solicitando que se le ingrese el *ID* del servidor al que se quiere conectar. Este *ID* se utilizará para filtrar y separar el tráfico de las aplicaciones cliente conforme se realicen conexiones al servidor. A continuación, se realiza el acceso al portal de *PhotonEngine* que permite la creación y administración de los servidores para la creación del servidor de trabajo:

Figura 38: Servidor utilizado para la conexión de usuarios en red.

Posteriormente el *ID* del servidor se introduce en el asistente de *PUN*:

Figura 39: Asistente PUN para la conexión al servidor.

Una vez introducido el *ID* del servidor, se almacena automáticamente la información en el fichero *PhotonServerSettings* para permitir las conexiones al servidor creado:

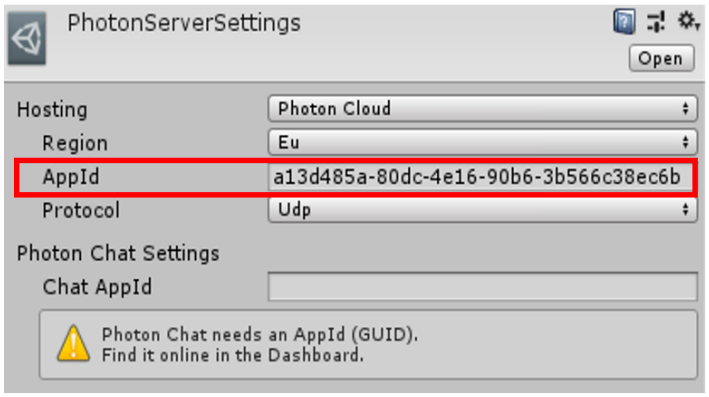


Figura 40: Fichero PhotonServerSettings.

Almacenada la información en el fichero *PhotonServerSettigns*, se realiza la creación de un *GameObject* vacío en la escena del proyecto, tal y como se hizo durante la prueba de conexión. Este *GameObject* ha sido llamado *ConexionServidor* y se le ha asignado el componente de tipo *script ConexionServidor.cs*. El objetivo de este *GameObject* va a ser el de encargarse de gestionar la lógica de la conexión con el servidor al ejecutarse la aplicación por cada cliente:

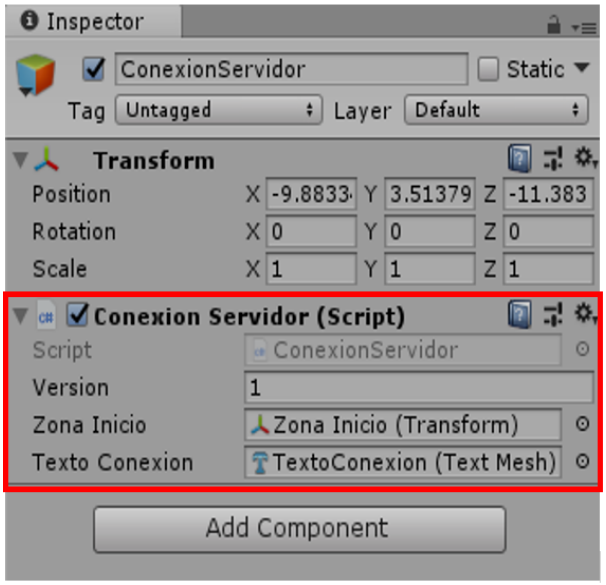


Figura 41: Componentes del GameObject ConexionServidor.

Script: ConexionServidor.cs ver código en el [Anexo 1]
Principales características y funcionalidades implementadas en el *script* asignado al *GameObject ConexionServidor*: **Objetivo del *script*:** El objetivo del siguiente *script* es el de gestionar la conexión con el servidor, de manera que se establezca la conexión y se produzca la creación de la sala en red donde se van a instanciar cada uno de los usuarios que se vayan conectando. Toda clase que requiera la conexión y sincronización en red por medio del paquete importado *PUN*, debe de contener la extensión *Photon.MonoBehaviour* en un interior, de esta manera se hace posible que cuando el *script* sea ejecutado, sea interpretado como código a gestionar por el servidor. **Funcionalidades del *script*:** • Implementación de la gestión y conexión al servidor • Implementación de la zona de inicio donde instanciar los usuarios conectados • Implementación del estado de conexión • Asignación del nombre de usuario

Tabla 4: Resumen del script ConexionServidor.cs.

Implementación de la gestión y conexión al servidor:

La gestión de la conexión al servidor es llevada a cabo por medio de los métodos proporcionados por el paquete *Photon Unity Networking (PUN)*, los cuales han sido utilizados para establecer los parámetros de conexión al servidor y la creación de la sala utilizada como entorno inmobiliario donde instanciar cada uno de los usuarios conectados. A continuación, se describe la funcionalidad de dichos métodos para ayudar a su comprensión y que papel tienen en cada una de las variables utilizadas en el *script ConexionServidor.cs*:

- *OnConnectedToMaster()*: función de *PUN* que es llamada cuando se realiza la conexión al servidor, de manera que en su interior utilizo la función para la creación de la sala donde instanciar todos los usuarios y le establezco como opciones una capacidad de conexión para 20 usuarios.

- *OnJoinedRoom()*: función de *PUN* llamada cuando se establece la conexión a la sala creada por el servidor. La utilizo de modo que cuando se realice la conexión posteriormente se instancie el usuario en la posición capturada por la variable *_ZonaInicio*. Una vez instanciado el usuario se le asigna el nombre almacenado en la variable *_NombreUsuario* para ser identificado.

- ConnectUsingSettings(): Esta función es utilizada para evaluar los parámetros de conexión al servidor, se le pasa la variable *String _Version* como parámetro. Esta variable ha sido creada para gestionar cada uno de los usuarios conectados en función de la versión de conexión al servidor.

Implementación de la zona de inicio donde instanciar los usuarios conectados:

Para la implementación de una zona inicial donde instanciar los usuarios conforme se conecten en la aplicación se hace uso de la variable *_ZonaInicio,* a la cual se le ha asignado un *GameObject* vacío con nombre similar llamado *Zona Inicio*. Este *GameObject* ha sido posicionado en las coordenadas del vector tridimensional (x, y, z) de la escena donde instanciar cada uno de los jugadores conforme se vayan conectando en la aplicación, las coordenadas del componente *Trasform* y su respectiva posición en la escena pueden observarse en la siguiente figura:

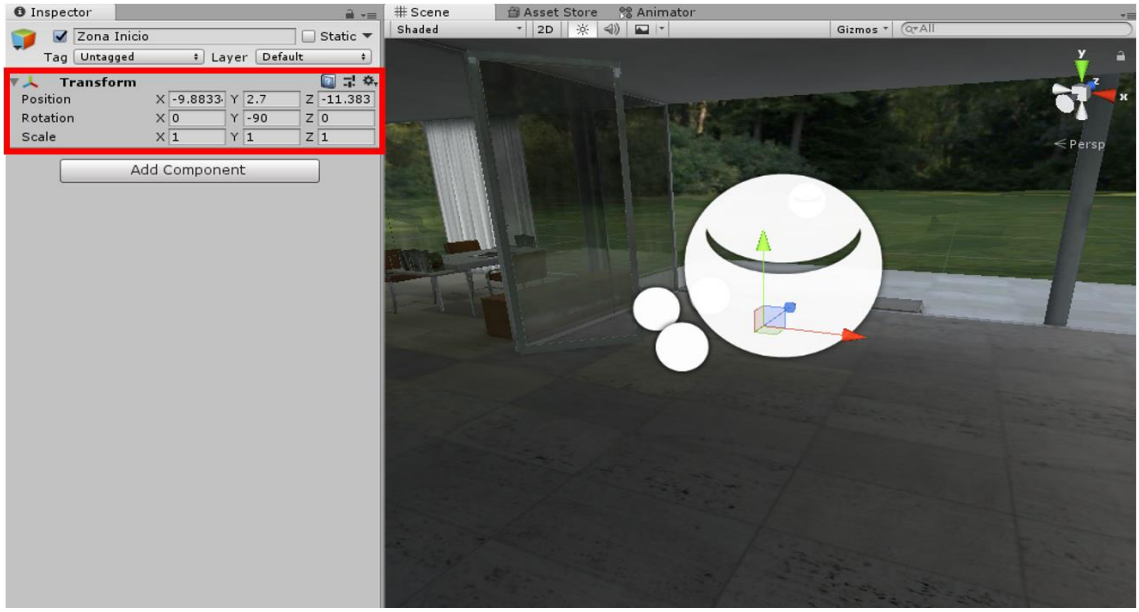


Figura 42: GameObject Zona Inicio.

La jerarquía de herencia del *GameObject Zona Inicio* se muestra a continuación:

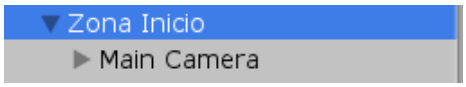


Figura 43: Jerarquía del GameObject Zona Inicio.

Donde el objeto *Zona Inicio* es padre de la cámara principal de la aplicación, de esta manera, la cámara hereda la posición y rotación local de su objeto padre. Esta situación me ha permitido resolver uno de los problemas que me he encontrado a lo largo del desarrollo que me impedía obtener la posición y vista deseadas de la cámara en el momento que se ejecutara la aplicación.

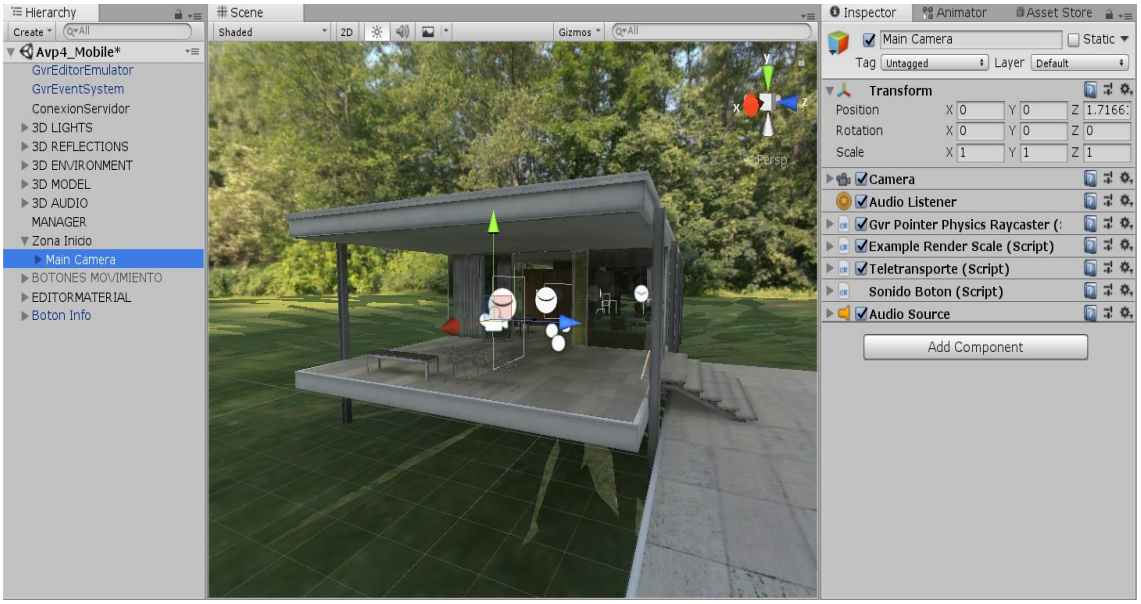


Figura 44: Solución al problema de la posición de la cámara y posición deseada.

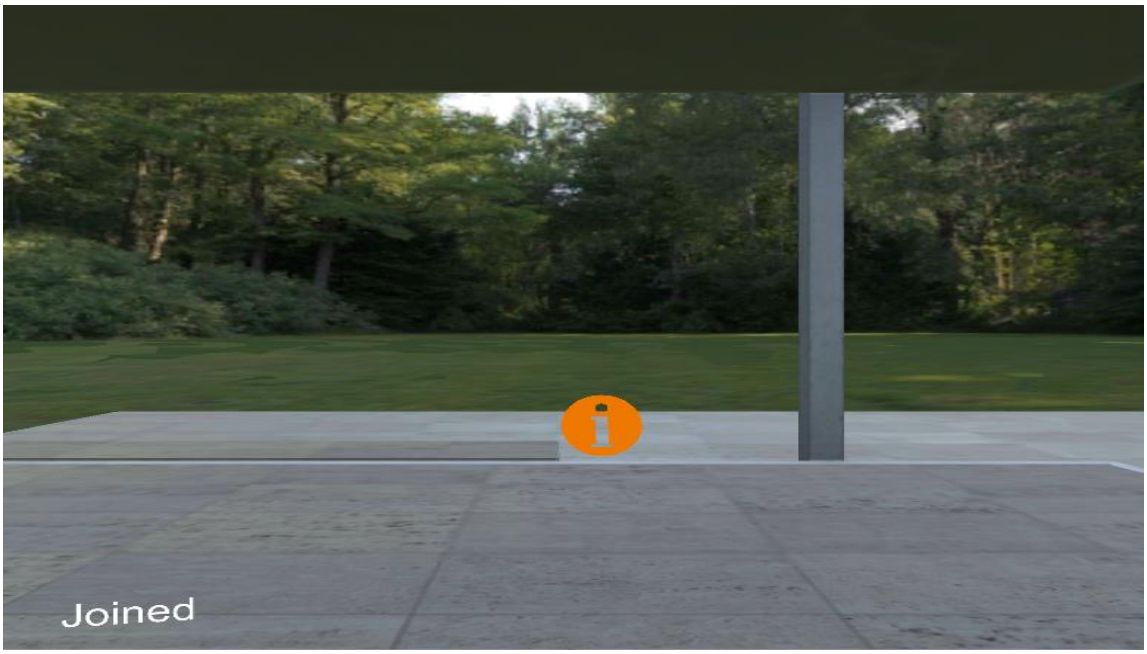


Figura 45: Solución al problema de la posición de la cámara y vista deseada.

De no ser así e introducir esta jerarquía de objetos se reescribía su posición en las coordenadas del vector (x=0, y=0; z=0,08), devolviendo una posición para nada deseada en la ejecución de la aplicación, ya que la cámara debe obtener la posición del usuario en el momento de ejecución ofreciendo una vista de realidad virtual en primera persona y no la posición y vista mostrada en las siguientes figuras:

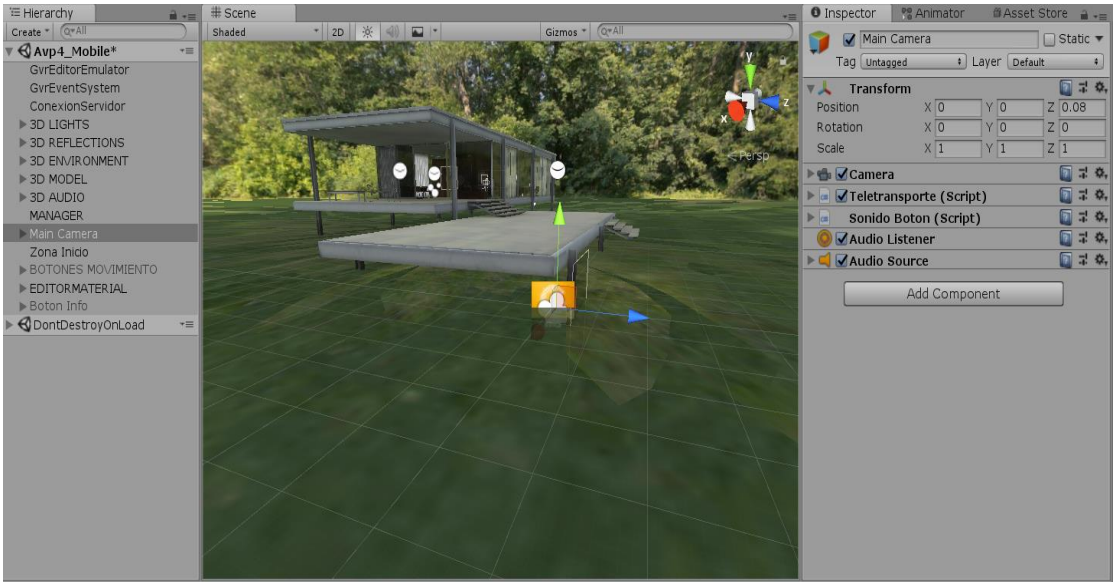


Figura 46: Posición no deseada de la cámara.

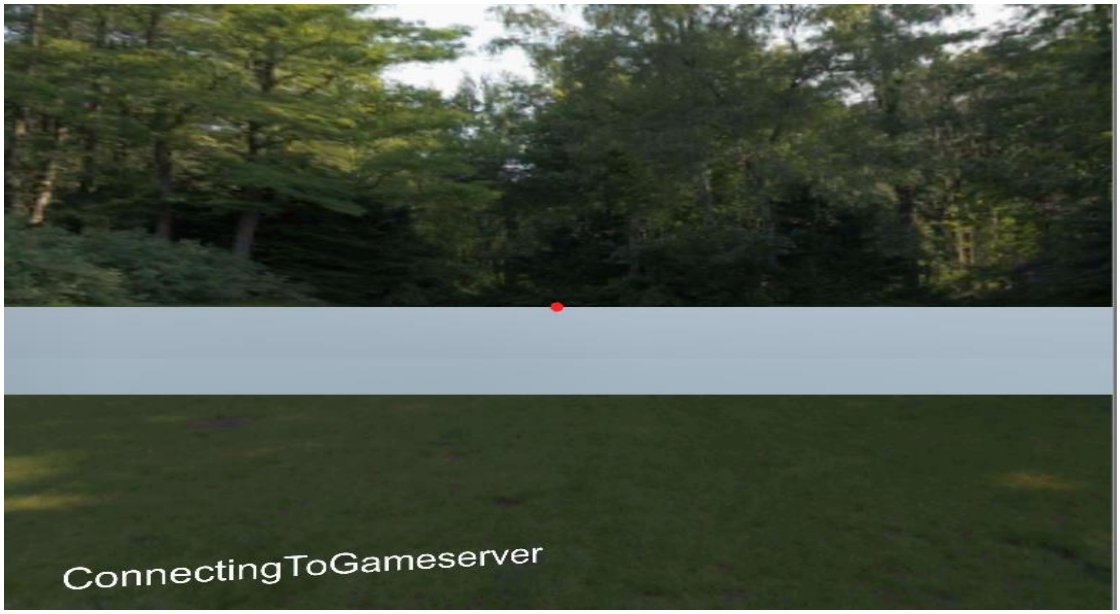


Figura 47: Vista no deseada ofrecida por la cámara.

Se puede observar cómo no se estaba cumpliendo la funcionalidad deseada en el *GameObject Zona Inicio*. Esto ocurría debido a la utilización de la realidad virtual en mi proyecto, en concreto al *Google VR SDK* que por defecto anula la posición de la cámara estereoscópica de la escena independientemente de donde se encuentre en la jerarquía de objetos, con el objetivo de emular la posición y rotación de la cámara siguiendo el modelo de rotación del cuello de *Google Daydream* y *Google Cardboard*.

Implementación del estado de conexión:

Se ha llevado cabo mediante la utilización en el *script Conexión.Servidor.cs* de la variable *_TextoConexion* y la creación en la escena del proyecto del *GameObject TextoConexion*, al cual hace referencia.

Fragmento de código del *script* ConexionServidor.cs

```
    void Update()
    {
        //Obtención del estado de conexión.
        _TextoConexion.text = PhotonNetwork.connectionStateDetailed.ToString();
        if (_TextoConexion.text == "Joined")
        {
            _TextoConexion.color = Color.Lerp(_TextoConexion.color,
Color.green, Time.deltaTime);
        }

    }
```

Objetivo del código: obtener el estado de conexión al servidor y poder ser mostrado a través de un texto en la vista del usuario.

Tabla 5: Código de programación que permite obtener el estado de conexión.

Para la creación del *GameObject TextoConexion* de manera que quede posicionado siguiendo la posición y rotación de la vista del usuario, se ha incluido como un objeto hijo de la cámara principal de la aplicación. Además, como se trata de un objeto del tipo *TextMesh* debe de ser tratado como todos los objetos que pertenecen a la interfaz del usuario (*UI*) de *Unity*. Necesitando la creación de un objeto del tipo *Canvas* llamado *CanvasCamera* que sea el hijo directo de la cámara. La jerarquía y herencia establecida de los objetos sigue la mostrada a continuación:

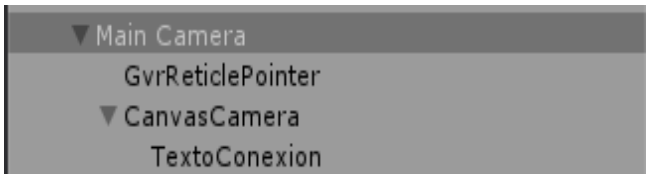


Figura 48: Jerarquía del GameObject TextoConexion.

El *Canvas* creado ha sido posicionado delante de la cámara y puede ser visto en la siguiente figura:

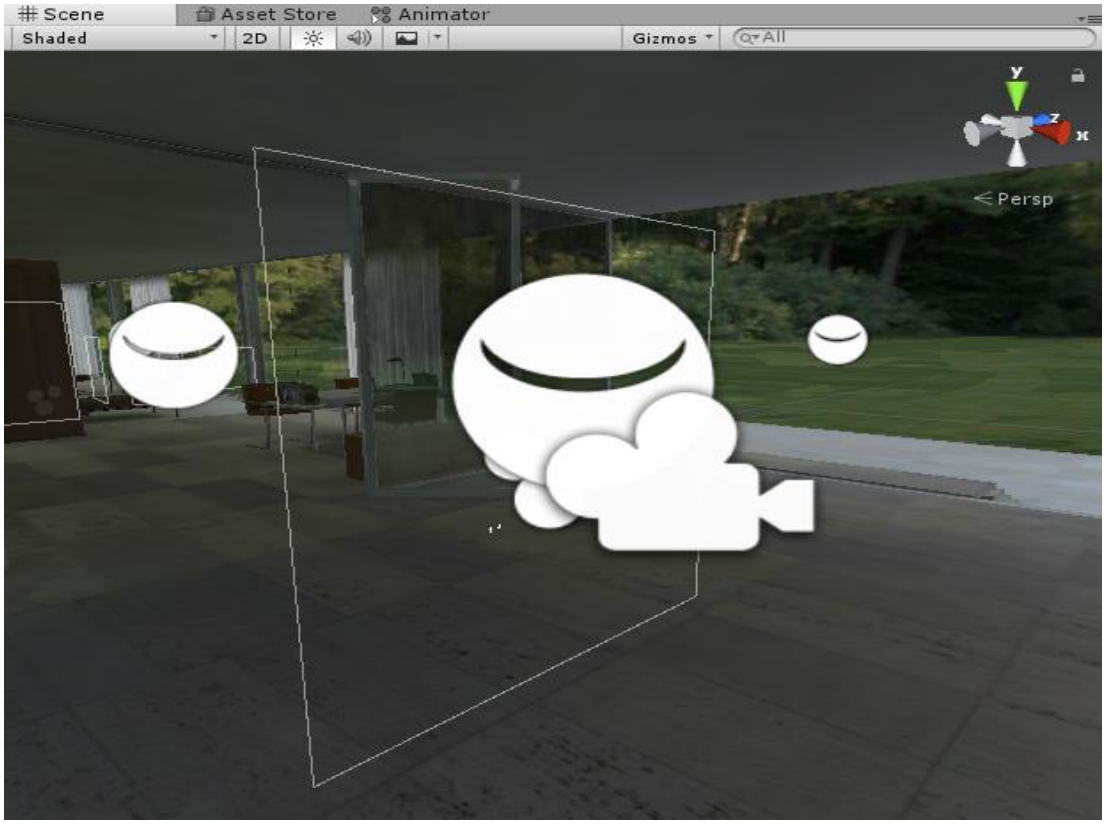


Figura 49: CanvasCamera desde la ventana Escena.

Los estados de conexión capturados a través del *script ConexionServidor.cs*:

Figura 50: Secuencia de estados por los que pasa el GameObject TextoConexion.

Como se puede observar el valor del *GameObject* pasa por diferentes estados de conexión en el momento que se ejecuta la aplicación, hasta llegar al estado final *Joined* donde el usuario finalmente ha sido conectado al servidor e instanciado en la escena.

Asignación del nombre de usuario:

El objetivo desarrollado con esta implementación es el de identificar a cada uno de los usuarios que se conecten en la sala inmobiliaria, para ello se les asigna un identificador que sirva como el nombre de usuario, por medio de la variable *_NombreUsuario*. El valor de la variable será representado en la escena de la aplicación por medio del *GameObject Nombre*. Este *GameObject* obtiene el valor asignado por la variable *_NombreUsuario* del *script ConexionServidor.cs* y representa el nombre asignado a cada usuario en la aplicación, situando encima del personaje utilizado un texto con el objetivo de poder identificar y diferenciar los usuarios conectados.

Figura 51: Nombre del usuario.

3.3 Creación del personaje y su configuración en red

A partir de este momento es necesaria la creación del personaje utilizado por el usuario, el cual va a ser instanciado cada vez que se vayan conectando diferentes usuarios en la aplicación. El personaje utilizado para representar al usuario ha sido llamado *Avatar* y va a ser generado como un *prefab* para poder ser accedido a través del *script ConexionServidor.cs* utilizado en la conexión al servidor. Para ello se ha realizado la creación del directorio *Resources* e introducido el *prefab Avatar* en su interior.

Figura 52: Contenido del directorio Resources.

3.3.1 Evolución del personaje a lo largo del proyecto

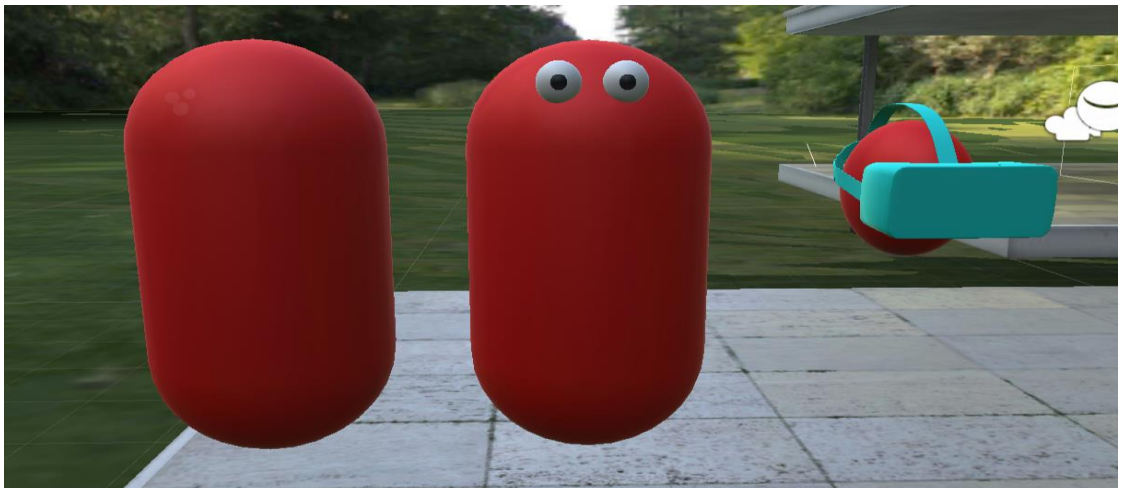


Figura 53: Evolución del personaje.

A lo largo del proyecto el personaje utilizado para el usuario se ha visto sometido a diversas transformaciones de diseño y funcionalidad, dado que en un primer momento se trataba de una capsula con la cual únicamente se hacían pruebas de conexión para la sincronización de usuarios en red, posteriormente se le decidió de dotar de jerarquía de objetos para poder abordar las problemáticas surgidas durante su instanciación en la escena y sincronización de la rotación. Posteriormente se desarrolló el modelo situado más a la derecha de la imagen, con el objetivo de simular la cabeza del usuario y dotar de un mejor aspecto en cuanto a diseño del avatar.

Finalmente, el avatar desarrollado presenta la siguiente estructura de jerarquía de objetos y herencia, con capacidades de funcionalidad y diseño más avanzados:

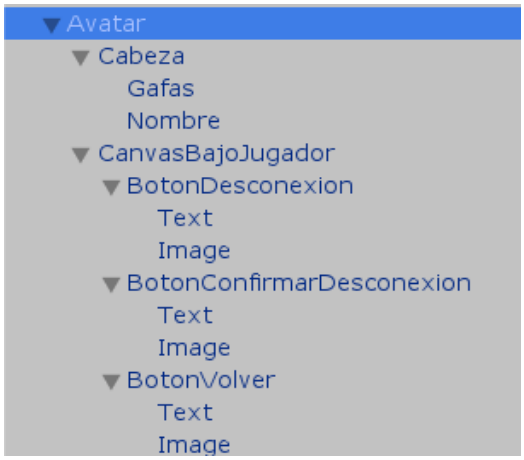


Figura 54: Jerarquía del GameObject Avatar.

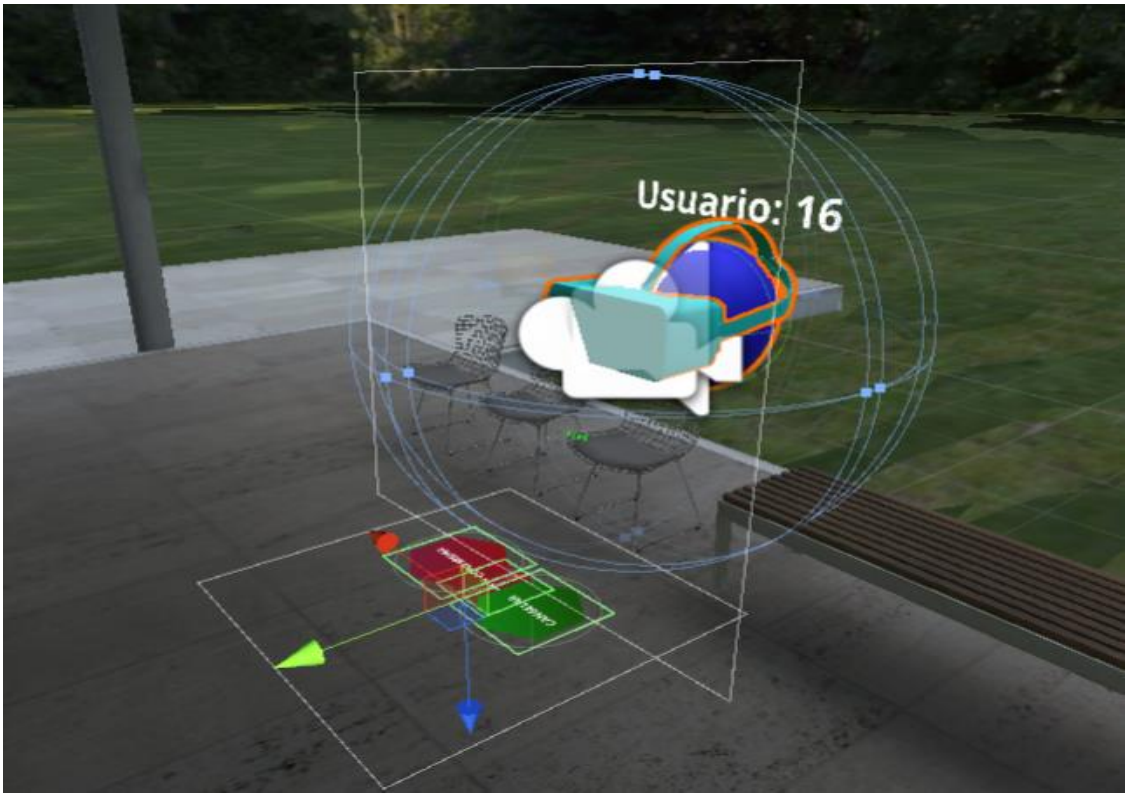


Figura 55: Personaje final Avatar para representar al usuario.

En ambas imágenes puede observarse como el personaje *Avatar* utilizado para cada uno de los usuarios que se conecten en la aplicación se compone de una amplia jerarquía de objetos que debe ser controlada y gestionada desde la programación por medio de diferentes *scripts*, de modo que cada uno de ellos se focalice en el comportamiento de los objetos en diferente nivel de jerarquía.

3.3.2 Avatar

Es el objeto padre de toda la jerarquía del personaje y el cual es utilizado para implementar la lógica que forma parte de la conexión relacionada con los usuarios y su configuración en red para su sincronización y comunicación. Además, es el encargado de controlar la jerarquía y comportamiento de los objetos que intervienen en él, cuando se realiza su conexión en la escena inmobiliaria de la aplicación.

El *GameObject Avatar* tiene asignados los siguientes componentes donde el encargado de gestionarlos es el *script UsuarioConectado.cs*:

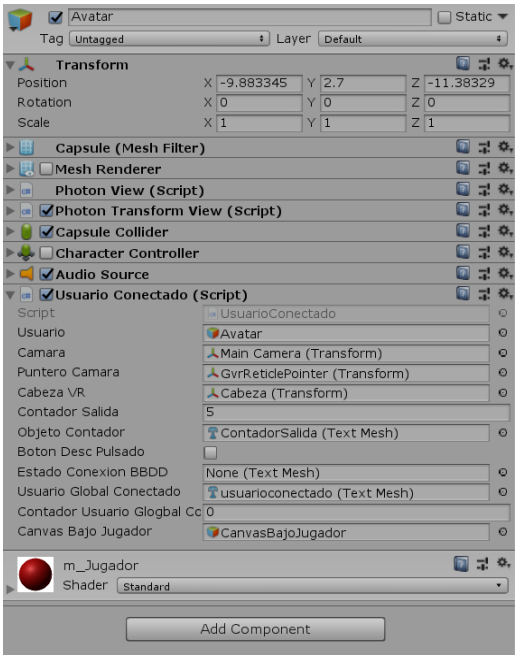


Figura 56: Componentes del GameObject Avatar.

Script: UsuarioConectado.cs ver código en el [Anexo 2]
Principales características y funcionalidades implementadas en el *script* asignado al *GameObject Avatar*: **Objetivo del *script*:** El objetivo del siguiente *script* es el de gestionar la lógica que forma parte de la conexión de los usuarios en la aplicación y controlar el comportamiento del usuario una vez es instanciado en la escena, además de establecer la sincronización y configuración del servidor utilizado para la transmisión de voz entre los usuarios y la conexión con la base de datos externa para la inserción de datos relacionados con el tiempo de conexión. **Funcionalidades del *script*:** • Distinguir entre el usuario local y el usuario global • Obtener en forma de evento cada usuario que se conecte en la aplicación • Gestión de la desconexión del usuario • Registrar la salida del usuario en la base de datos • Registrar la entrada del usuario en la base de datos

Tabla 6: Resumen del script UsuarioConectado.cs.

Distinguir entre el usuario local y el usuario global:

El *script UsuarioConectado.cs* gestiona la lógica de comportamiento de la aplicación en función de si el usuario se trata de un jugador local o no, ya que se trata del mismo avatar utilizado para todos los usuarios, pero el comportamiento no ha de ser el mismo.

De manera que en el momento que se conecte un usuario local se le cambia en primer lugar el nombre al objeto *Avatar* por *Usuario Local* y de esta manera en el caso de que vayan conectándose otros jugadores en la escena sean percibidos con el nombre de *Usuario Global* para todos ellos.

Cuando se conecta *Usuario Local* y es instanciado, se establecen relaciones de jerarquía y herencia con el resto de objetos que pertenecen a la estructura del *Avatar* que permiten ajustar la rotación y posición de la cámara en función de la posición del usuario. De manera que en el momento que se ejecute la aplicación y se conecte como *Usuario Local*, la cámara principal capturada por medio del objeto *_Camara* pase a ser hija del *Avatar* y obtenga la posición del objeto padre. Consiguiendo desde ese momento que la posición de la cámara se vaya desplazando en función de la posición del *Usuario Local*.

Además, se controla la rotación del *GameObject Cabeza* encontrado a través de la variable *_CabezaVR*, de modo que este objeto que simula la cabeza del personaje, se quede como objeto hijo de la cámara. De esta manera cuando rote la cámara al girar la cabeza por medio de los sensores de movimiento del dispositivo móvil en realidad virtual, también gire el *GameObject Cabeza* simulando nuestros movimientos como si de nuestra cabeza se tratase.

Figura 57: El GameObject Avatar pasa a ser llamado Usuario Local y se establece la jerarquía de objetos.

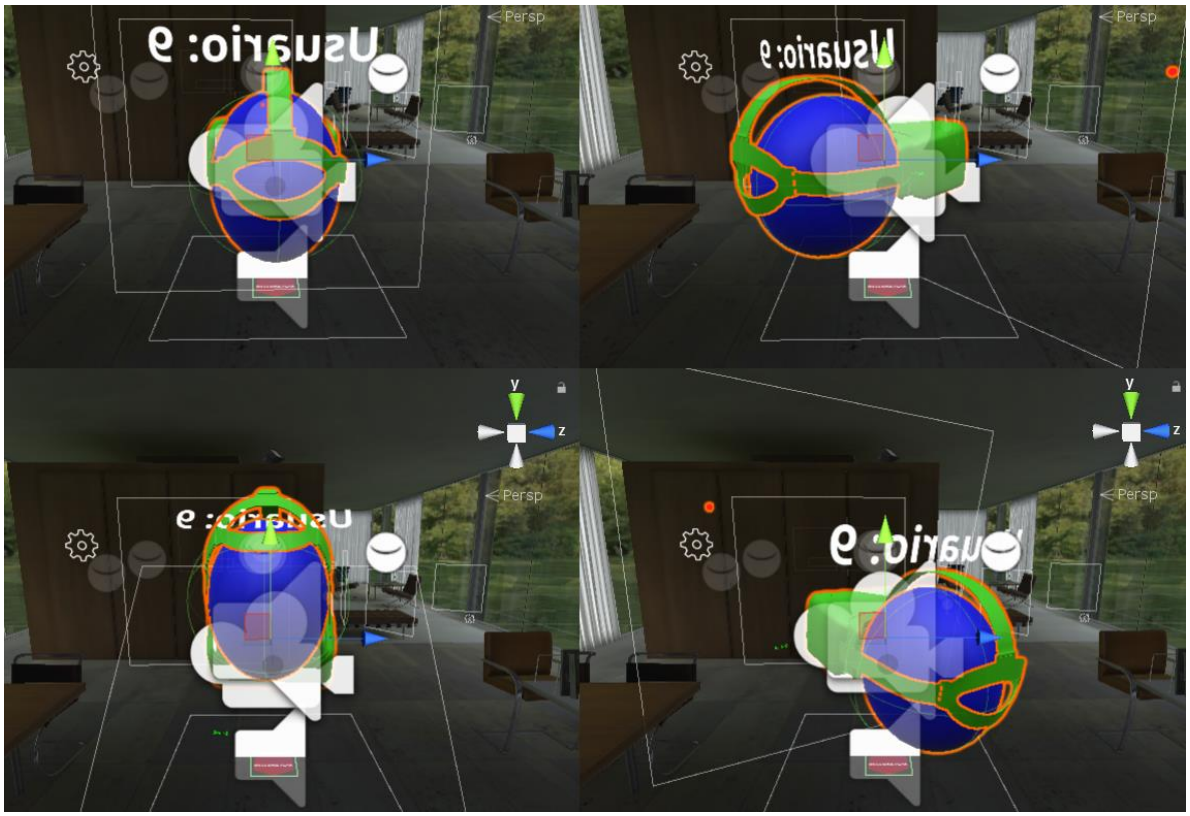


Figura 58: Rotación del GameObject Cabeza en relación con la rotación de la cámara de realidad virtual.

El *script ConexionServidor.cs* también oculta el menú creado para la desconexión del usuario en el caso de no ser *Usuario Local* ya que no debe de ser un contenido que pueda ser visualizado por el resto de usuarios que se conecten. Para ello en el caso de no ser usuario local se inactiva el objeto *_CanvasBajoJugador* en el *script*, el cual hace referencia al menú creado para la desconexión llamado *CanvasBajoJugador*.

En las figuras mostradas debajo puede observarse como el Usuario *Local* (de color azul) no puede observar el menú utilizado para la desconexión del *Usuario Global (de color rojo).*

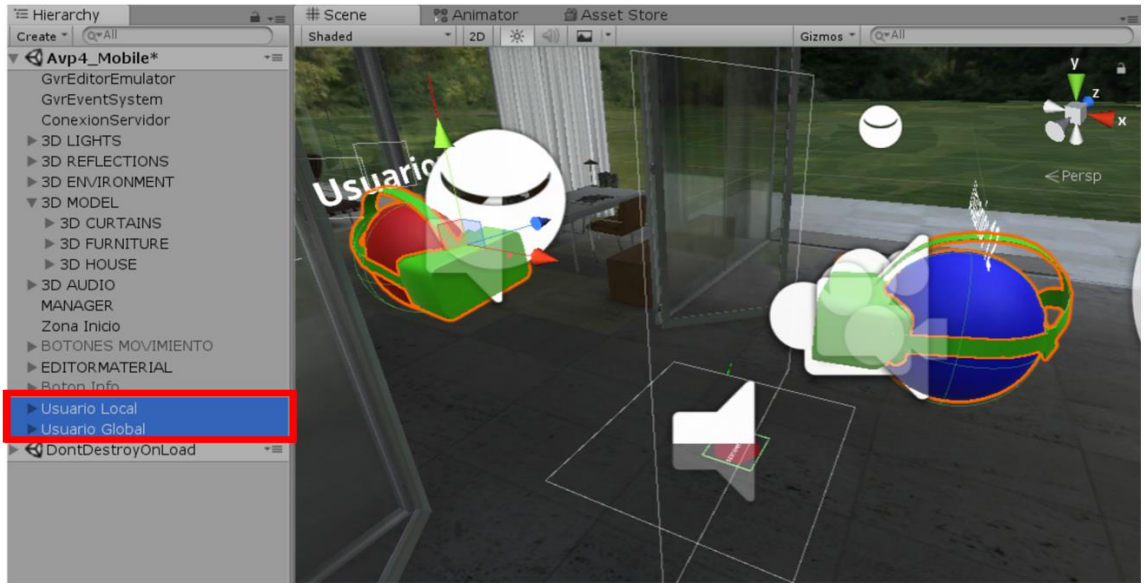


Figura 59: Inactivación de CanvasBajoJugador desde la ventana Escena de Unity.

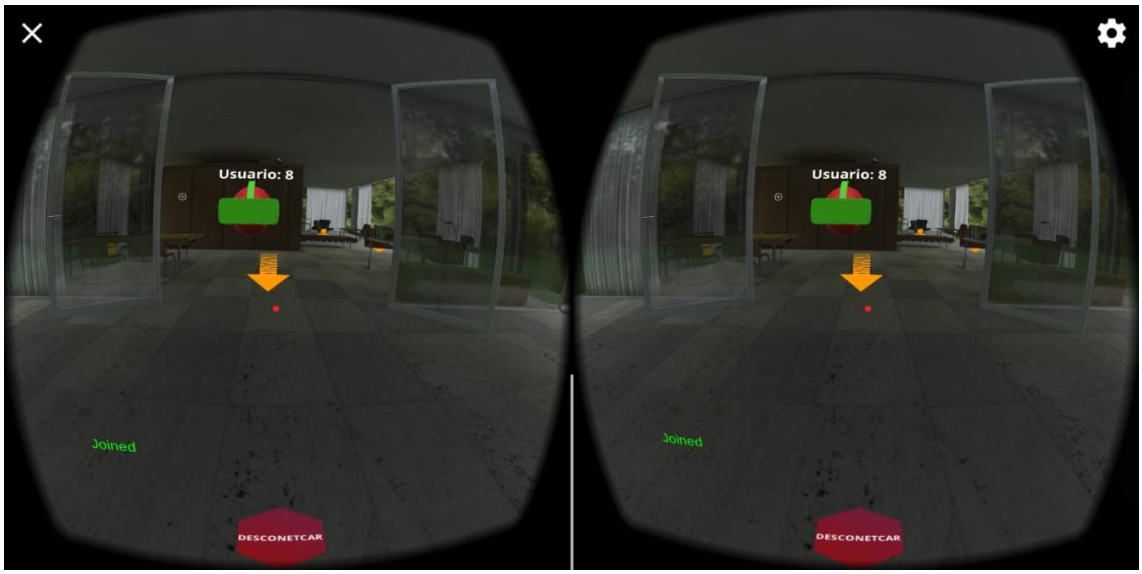


Figura 60: Inactivación de CanvasBajoJugador desde Android.

Obtener en forma de evento cada usuario que se conecte en la aplicación:

Se realiza una actualización a través del método *Update()* que capture en forma de evento cada vez que se conecte un usuario en la aplicación. Con el objetivo de poder obtener la información como usuario local, cuando otro usuario se conecte en la misma sala. Esto es realizado asignado a la variable *_UsuarioGlobalConectado* de tipo *TextMesh* el nombre de *Usuario Global* que se ha conectado, ejecutando dicha acción por cada *Usuario Global* que se encuentre en la lista *PhotonNetwork.otherPlayers*.

De modo que por cada *Usuario Global* que este en la lista, se realice la acción de asignación. Esto además es controlado por un contador de modo que el contenido desaparezca en forma de desvanecimiento pasado cierto tiempo de ejecución de la aplicación y se obtenga el evento en forma de un texto de color verde.

Fragmento de código del *script* UsuarioConectado.cs

```
foreach(PhotonPlayer _Usuario in PhotonNetwork.otherPlayers)
        {
            _ContadorUsuarioGlogbalConectado += Time.deltaTime;
            if (_ContadorUsuarioGlogbalConectado > 3.5f)
                {
                UsuarioGlobalConectado.text = "";

            }
            else
            {
                UsuarioGlobalConectado.text = "Conectado con: "
+_Usuario.NickName;
                UsuarioGlobalConectado.color =
Color.Lerp(UsuarioGlobalConectado.color, Color.green, Time.deltaTime);
            }

        }
```

Objetivo del código: implementar la funcionalidad que capture en forma de evento cada usuario que se conecte en la aplicación.

Tabla 7: Código de programación que obtiene el nombre de usuarios globales conforme se conectan.

Figura 61: Evento que muestra el nombre del Usuario Global al conectarse.

Gestión de la desconexión del usuario:

Para poder ofrecer la funcionalidad de desconexión del usuario se ha creado un menú situado bajo el personaje llamado *CanvasBajoJugador*. De manera que no ocupe un espacio visible que pueda ser molesto para la vista ofrecida en realidad virtual y en el momento que se quiera proceder a la desconexión, el usuario únicamente tiene que desplazar la mirada a sus pies y podrá observar el botón de desconexión que habilita el menú:

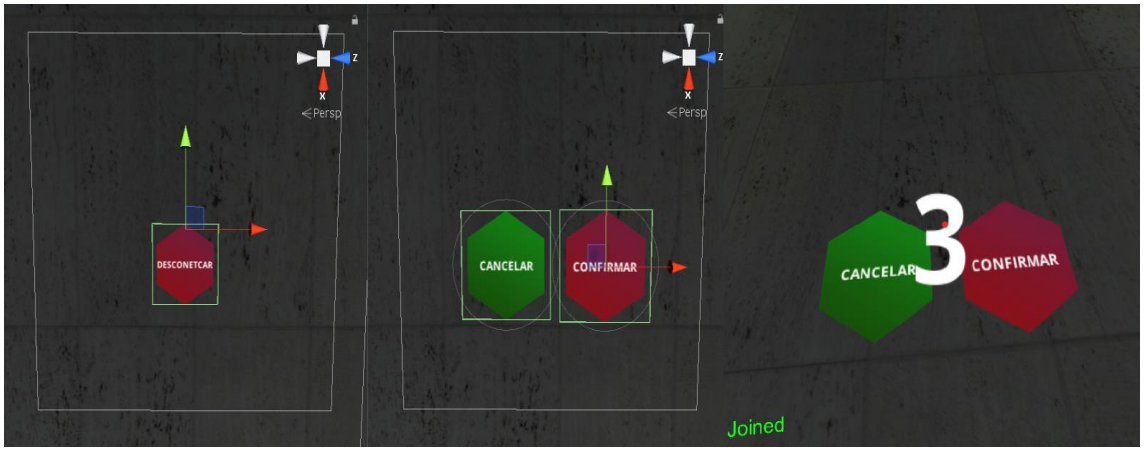


Figura 62: Menú CanvasBajoJugador.

El comportamiento del menú es controlado por el *script ConexionJugador.cs*, en concreto por el método *Desconectar()* el cuál es ejecutado al pulsar sobre el botón *Confirmar*, es entonces cuando la variable *booleana _BotonDescPulsado* se establece como *true* y ejecuta la condición introducida en el método *Update()*, que irá descontando el valor del contador flotante inicializado a 5 en la variable *_ContadorSalida*, dicho valor se irá reduciendo cada refresco de ejecución por segundo hasta llegar a 0 y es obtenido en la pantalla de realidad virtual por la variable del tipo *TextMesh _ObjetoContador*. En el momento que el contador llega a 0 el usuario es desconectado y se cierra la aplicación.

Fragmento de código del *script UsuarioConectado.cs*

```
    public void Desconectar ()
    {
        if (photonView.isMine)
        {
            _BotonDescPulsado = true;
        }
    }
```

Objetivo del código: creación de un método público que pueda ser llamado en el momento que se presione el botón *Confirmar* estableciendo la variable *booleana* a *true*. En el momento que la variable *_BotonDescPulsado* se establece como *true* se ejecuta el siguiente fragmento de código mostrado en la fila inferior:

```
    void Update () {

        if (photonView.isMine) {
            if (_BotonDescPulsado)
        {
            if(_ContadorSalida > 0.0f)
            {
                _ContadorSalida -= Time.deltaTime;
                _ObjetoContador.text = _ContadorSalida.ToString("0");
            }
            else
            {
                _ObjetoContador.text = "0";

StartCoroutine(RegistrarSalidaUsuario(PhotonNetwork.player.NickName));
                PhotonNetwork.Disconnect();
                Application.Quit();
            }
        }
         }
    }
```

Objetivo del código: implementar la funcionalidad que permita gestionar la desconexión de los usuarios.

Tabla 8: Implementación de la desconexión de los usuarios.

Registrar la salida del usuario en la base de datos:

Con el objetivo de obtener un control de las desconexiones realizadas por cada uno de los usuarios, se ha implementado la funcionalidad en el *script UsuarioConectado.cs* por medio de *RegistrarSalidaUsuario()*, de manera que permita enviar la información en el momento de la desconexión a *RegistrarSalidaUsuario.php* y se realice la inserción de los datos en una base de datos externa.

Para ello, en el momento que se ejecuta *RegistrarSalidaUsuario()*, esta llama al fichero externo en *PHP* y le envía a través de un *String* el nombre del usuario que se ha desconectado de la aplicación, almacenándolo en la variable del fichero *PHP $UsuarioRegistrado*. Posteriormente *RegistrarSalidaUsuario.php* es el encargado de proceder a la inserción del nombre del usuario junto con la fecha actual en la base de datos creada en *MySQL*, esta función es lanzada cuando el contador que gestiona la desconexión es igual a 0. En el caso de no haber conseguido insertar los datos se devuelve el error desde el servidor *Apache*.

Fragmento de código del *script UsuarioConectado.cs*

```
StartCoroutine(RegistrarSalidaUsuario(PhotonNetwork.player.NickName));
```

Objetivo del código: ejecución de la función en el momento que se realiza la desconexión del usuario en la aplicación, pasándole el como argumento el nombre de usuario.

```
    IEnumerator RegistrarSalidaUsuario(string _NombreUsuario)
    {
        WWWForm form = new WWWForm();
        form.AddField("UsuarioRegistrado", _NombreUsuario);
        using (UnityWebRequest www =
UnityWebRequest.Post("http://127.0.0.1/Unity/RegistrarSalidaUsuario.php",
form))
        {
            yield return www.SendWebRequest();

            if (www.isNetworkError || www.isHttpError)
            {
                Debug.Log(www.error);
            }
            else
            {
                Debug.Log(www.downloadHandler.text);
            }
        }
    }
```

Objetivo del código: ejecución de la función encargada de enviar la información al fichero *RegistrarSadidaUsuario.php* para la inserción de datos en la base de datos *MySQL*.

Tabla 9: Implementación de la funcionalidad que permite registrar la fecha de salida en la base de datos.

Registrar la entrada del usuario en la base de datos:

En el momento que se instancia el usuario local en la escena de la aplicación se llama a la función *RegistrarEntradaUsuario()*, almacenando en la variable utilizada en el fichero externo *PHP $UsuarioRegistrado* el nombre del usuario que se le es asociado al jugador local. Posteriormente a través del fichero *RegistrarEntradaUsuario.php* se introducen los datos en la base de datos.

Fragmento de código del *script UsuarioConectado.cs*
`StartCoroutine(RegistrarEntradaUsuario(PhotonNetwork.player.NickName));`
Objetivo del código: ejecución de la función en el momento que se realiza la conexión del usuario en la aplicación, pasándole como argumento el nombre de usuario.

`IEnumerator RegistrarEntradaUsuario(string _NombreUsuario)`
`{`
`WWWForm form = new WWWForm();`
`form.AddField("UsuarioRegistrado", _NombreUsuario);`
`using (UnityWebRequest www = UnityWebRequest.Post("http://127.0.0.1/Unity/RegistrarEntradaUsuario.php", form))`
`{`
`yield return www.SendWebRequest();`
`if (www.isNetworkError || www.isHttpError)`
`{`
`Debug.Log(www.error);`
`}`
`else`
`{`
`Debug.Log(www.downloadHandler.text);`
`}`
`}`
`}`

Objetivo del código: ejecución de la función encargada de enviar la información al fichero *RegistrarEntradaUsuario.php* para la inserción de datos en la base de datos *MySQL*.

Tabla 10: Implementación de la funcionalidad que permite registrar la fecha de entrada en la base de datos.

3.3.3 Cabeza

El objeto que forma parte del personaje llamado *Cabeza* es el encargado de representar la cabeza del usuario. De modo que reproduzca las rotaciones que ejecute el usuario a través de los sensores de movimiento del dispositivo móvil en la escena de realidad virtual. Para dotarle de diseño he decidido crear modelo en tres dimensiones sencillo que conste de una esfera que simule la cabeza y unas gafas de realidad virtual incorporadas.

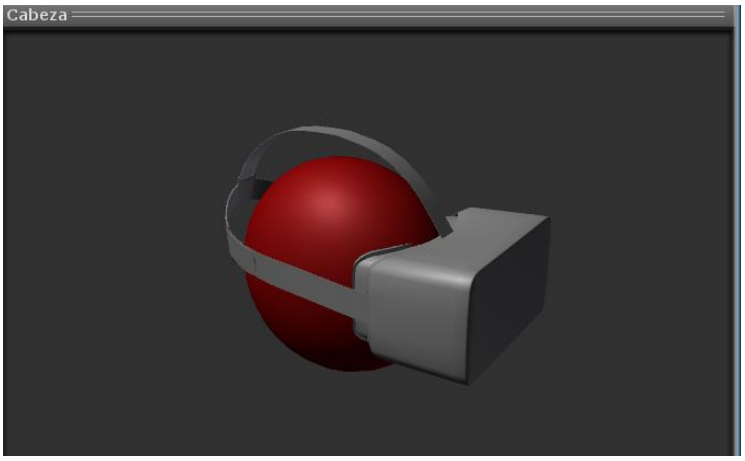


Figura 63: GameObject Cabeza.

El *GameObject Cabeza* tiene asignados los siguientes componentes donde el encargado de gestionarlos es el *script GiroCabeza.cs*:

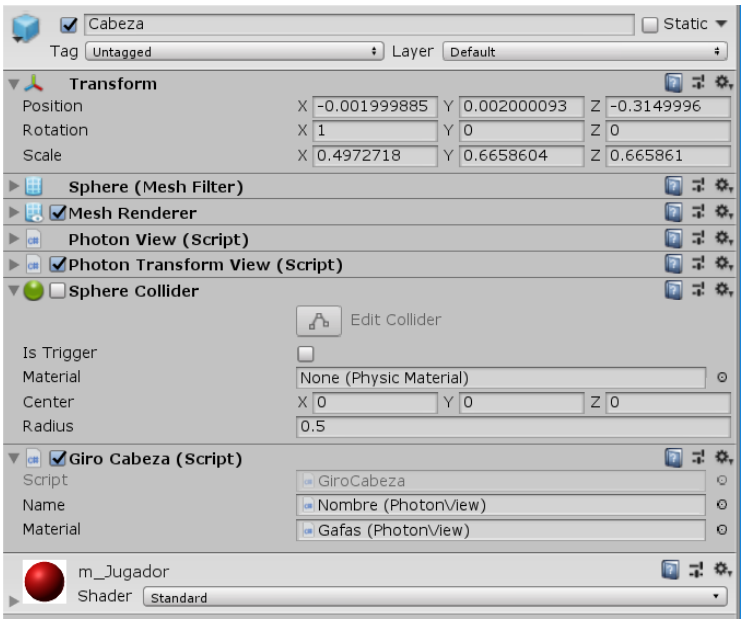


Figura 64: Componentes del GameObject Cabeza.

Script*: *GiroCabeza.cs ver código en el [Anexo 3]
Principales características y funcionalidades implementadas en el *script* asignado al *GameObject Cabeza*: **Objetivo del *script*:** El objetivo del siguiente *script* es el de sincronizar la rotación de la cabeza a través del servidor, de manera que la rotación del usuario pueda ser percibida por el resto de usuarios en tiempo real. Además de encargarse de la sincronización del nombre de usuario y de asignar un color aleatorio a las gafas de realidad virtual del personaje, haciendo uso de llamadas a procedimientos remotos en otros *scripts*. **Funcionalidades del *script*:** • Asignar color de personaje azul al usuario local • Sincronización de la rotación de la cabeza en red • Sincronización del nombre de usuario en red • Asignación del color de las gafas del personaje

Tabla 11: Resumen del script GiroCabeza.cs.

Asignar color de personaje azul al usuario local:

Como forma de diferenciar al usuario local del resto de usuarios, se le asigna al *GameObject Cabeza* un color azul:

Fragmento de código del *script GiroCabeza.cs*

```
private void Start()
{
    if (photonView.isMine)
    {

        this.GetComponent<MeshRenderer>().material.color = Color.blue;
    }
    else
    {
        return;
    }
}
```

Objetivo del código: diferenciar durante la ejecución de la aplicación al usuario local del resto por medio de la asignación del color azul para el local.

Tabla 12: Implementación del color azul para el usuario local.

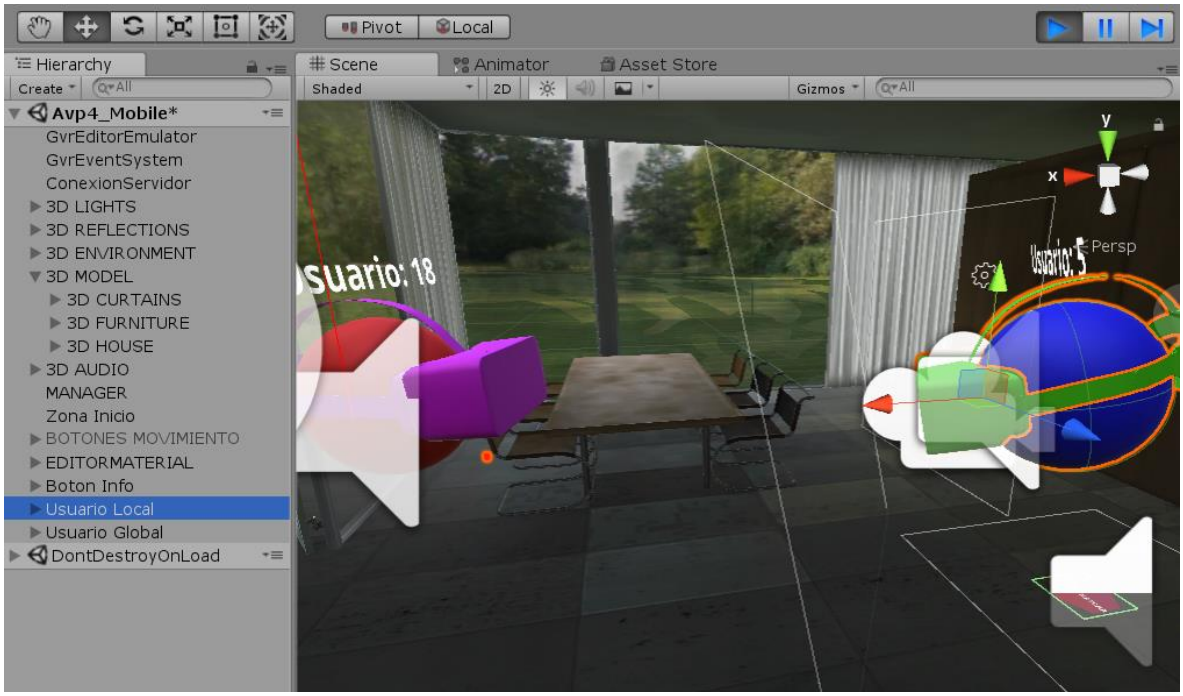


Figura 65: Color azul para el usuario local y rojo para el resto de usuarios.

Sincronización de la rotación de la cabeza en red:

En el *script GiroCabeza.cs* se ha implementado la funcionalidad de capturar la rotación del usuario a medida que este desplaza la mirada alrededor de la escena de realidad virtual, de manera que pueda ser percibida por cada uno de los usuarios en tiempo real observando cada uno de sus movimientos y hacia donde dirigen la mirada. Con el objetivo principal de ofrecer una simulación virtual más realista.

De no ser implementada esta funcionalidad la rotación de la cabeza únicamente podría ser percibida por el usuario local, ya que la información de la rotación del usuario local no es enviada al resto de clientes conectados en la aplicación al mismo tiempo.

Fragmento de código del *script GiroCabeza.cs*

```
    void Update () {
        if (photonView.isMine)
        {
            return;
        }
        else
        {
            this.transform.localRotation =
Quaternion.Lerp(this.transform.localRotation, _RotacionCabeza, .1f);
        }
    }
    void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
    {
        if (stream.isWriting)
        {
            stream.SendNext(this.transform.rotation);
        }
        else
        {
            _RotacionCabeza = (Quaternion)stream.ReceiveNext();
        }
    }
```

Objetivo del código: el siguiente código envía la información de la rotación del *GameObject Cabeza* en el caso de ser el usuario local, en caso contrario, se recibe la información almacenándola en la variable *_RotacionCabeza* y es aplicada en cada refresco por segundo de la ejecución de la aplicación al resto de usuarios no locales.

Tabla 13: Código que implementa la sincronización de la rotación de la cabeza en red.

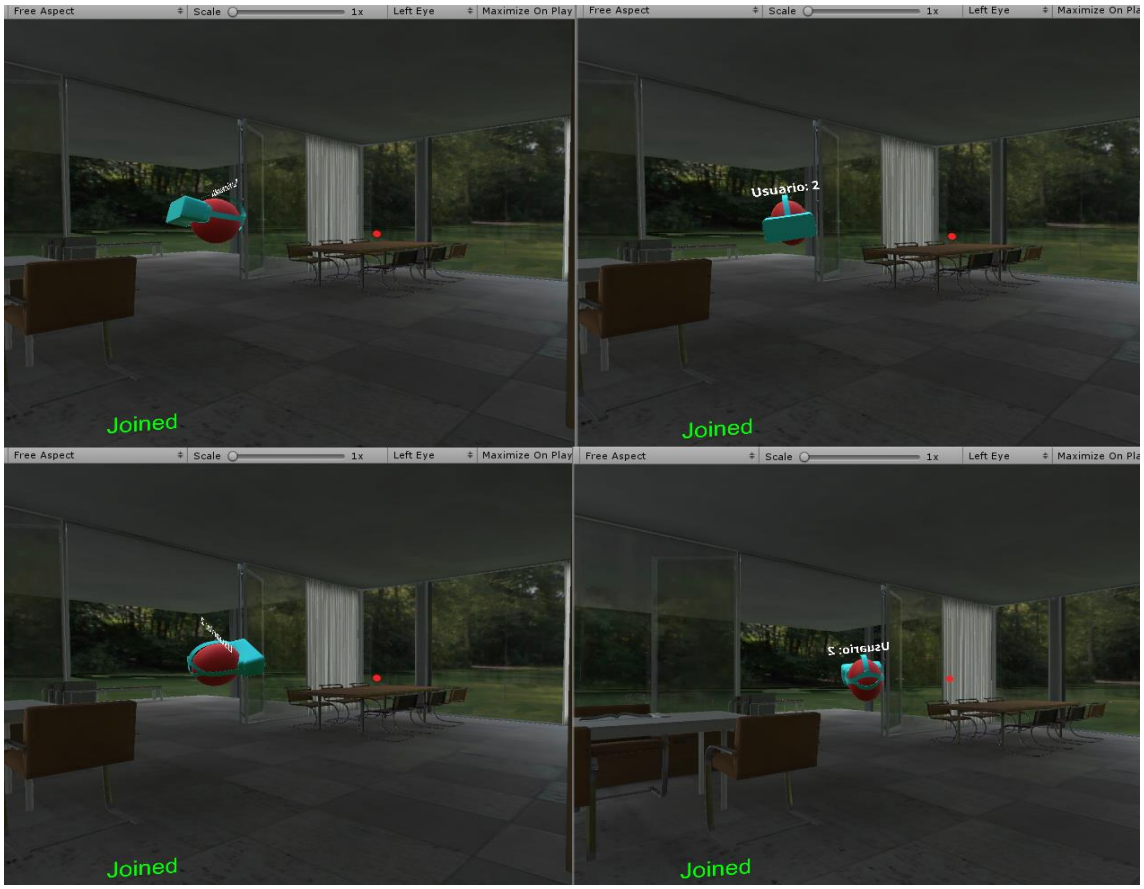


Figura 66: Rotación del Usuario Global sincronizada y observada por el Usuario Local.

Sincronización del nombre de usuario en red:

Para la sincronización del nombre de usuario en red se ha hecho uso de la llamada a procedimientos remotos *(PunRPC)* disponible en el paquete *Photon Unity Networking (PUN)*. Esto permite que el resto de clientes puedan sincronizar el nombre de usuario creado para cada uno de los usuarios que se van conectando en la aplicación.

Fragmento de código del *script GiroCabeza.cs*

```
    public PhotonView _Nombre;

     private void Start()
     {
         if (photonView.isMine)
         {
             _Nombre.RPC("ActualizarNombre", PhotonTargets.AllBuffered,
PhotonNetwork.playerName);
         }
         else
         {
             return;
         }
     }
```

Objetivo del código: realizar la llamada al procedimiento remoto contenido en el *script NombreUsuario.cs* para la sincronización del valor que almacena el nombre de usuario.

Tabla 14: Llamada al procedimiento remoto para la sincronización del nombre en red.

Como se puede observar en la siguiente figura, el *GameObject Cabeza* contiene como objeto hijo al *GameObject Nombre*, el cual almacena el valor del nombre de usuario.

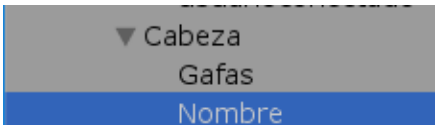


Figura 67: Jerarquía del GameObject Nombre.

Es necesario la creación de un nuevo *script* para el *GameObject Nombre* que almacene el valor del nombre de usuario obtenido en el momento de la conexión al servidor. De

este modo el *GameObject Nombre* contiene el *script* que almacena la función a ser llamada y ejecutada de manera remota desde *GiroCabeza.cs:*

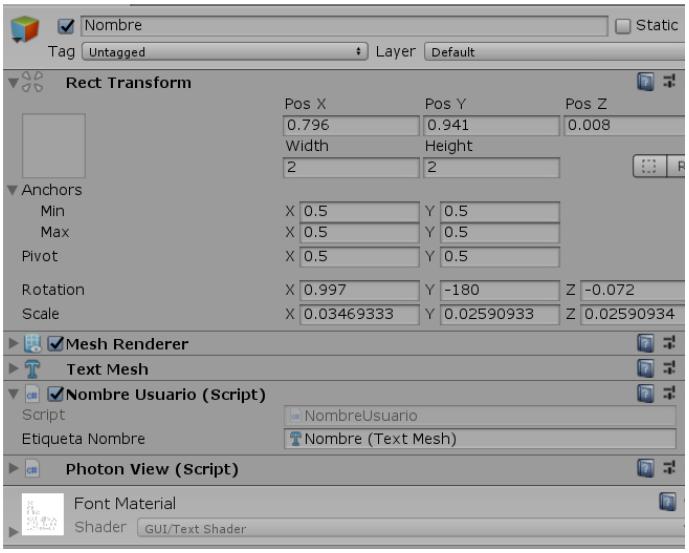


Figura 68: Componentes del GameObject Nombre.

Script: NombreUsuario.cs ver código en el [Anexo 4]
Principales características y funcionalidades implementadas en el *script* asignado al *GameObject Nombre*: **Objetivo del *script*:** El objetivo del siguiente *script* es el de sincronizar el nombre de usuario de modo que pueda ser observado por el resto de usuarios de la aplicación. **Funcionalidades del *script*:** • Sincronizar por medio de la llamada a procedimiento remoto el nombre de usuario.

Tabla 15: Resumen del script NombreUsuario.cs.

Asignación del color de las gafas del personaje:

Como forma de obtener un color de gafas aleatorio para uno de los usuarios que se vayan conectando en la aplicación, se ha desarrollado el *script ColorGafas.cs*, de esta manera cada vez que se conecte un usuario tendrá un color de gafas diferente.

Para ello se han generado los siguientes materiales para que puedan ser utilizados por el *script ColorGafas.cs*:

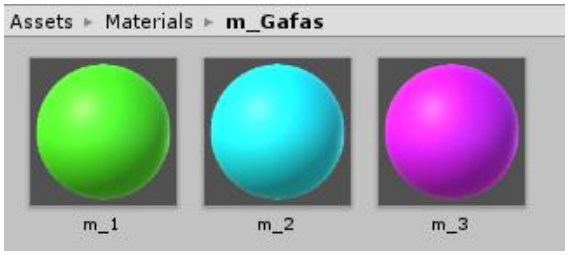


Figura 69: Colores disponibles para las gafas del personaje.

Así pues, cada vez que un usuario se conecte en la sala inmobiliaria, el color de las gafas del personaje toma un color aleatorio de entre los materiales creados.

Script: ColorGafas.cs ver código en el [Anexo 5]
Principales características y funcionalidades implementadas en el *script* asignado al *GameObject Gafas*: **Objetivo del *script*:** El objetivo del siguiente *script* es el de personalizar el color de las gafas de cada usuario de manera aleatoria conforme se conecten en la escena inmobiliaria. **Funcionalidades del *script*:** • Sincronizar por medio de la llamada a procedimiento remoto el color de las gafas con el resto de usuarios.

La función *SincronizarMaterial()* es llamada desde el *script GiroCabeza.cs* a través del uso de (*PunRPC*), asignando el color aleatorio a las gafas de cada usuario.

En el siguiente fragmento de código del *script GiroCabeza.cs* se realiza la llamada al procedimiento remoto alojada en el *script ColorCabeza.cs*:

Fragmento de código del *script GiroCabeza.cs*

```
private void Start()
{
    if (photonView.isMine)
    {
        _Material.RPC("SincronizarColor", PhotonTargets.AllBuffered);
    }
    else
    {
        return;
    }
}
```

Objetivo del código: realizar la llamada al procedimiento remoto para la obtención del color aleatorio de las gafas de los personajes conforme se conecten los usuarios.

Tabla 16: Llamada al procedimiento remoto para la asignación del color de gafas.

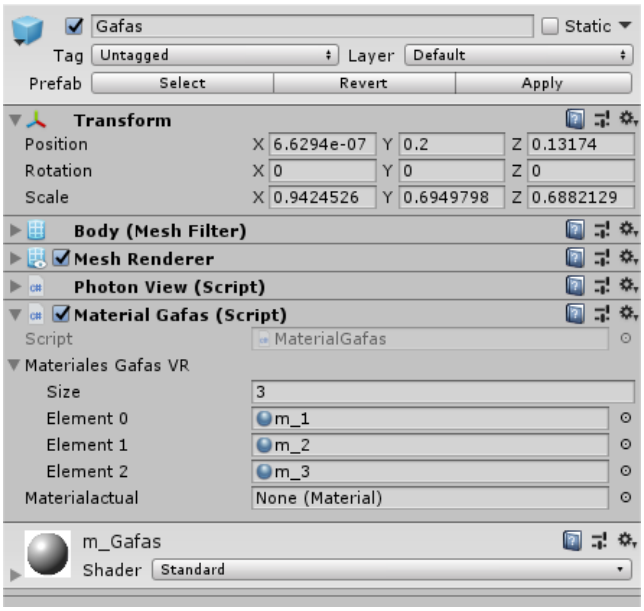


Figura 70: Componentes del GameObject Gafas.

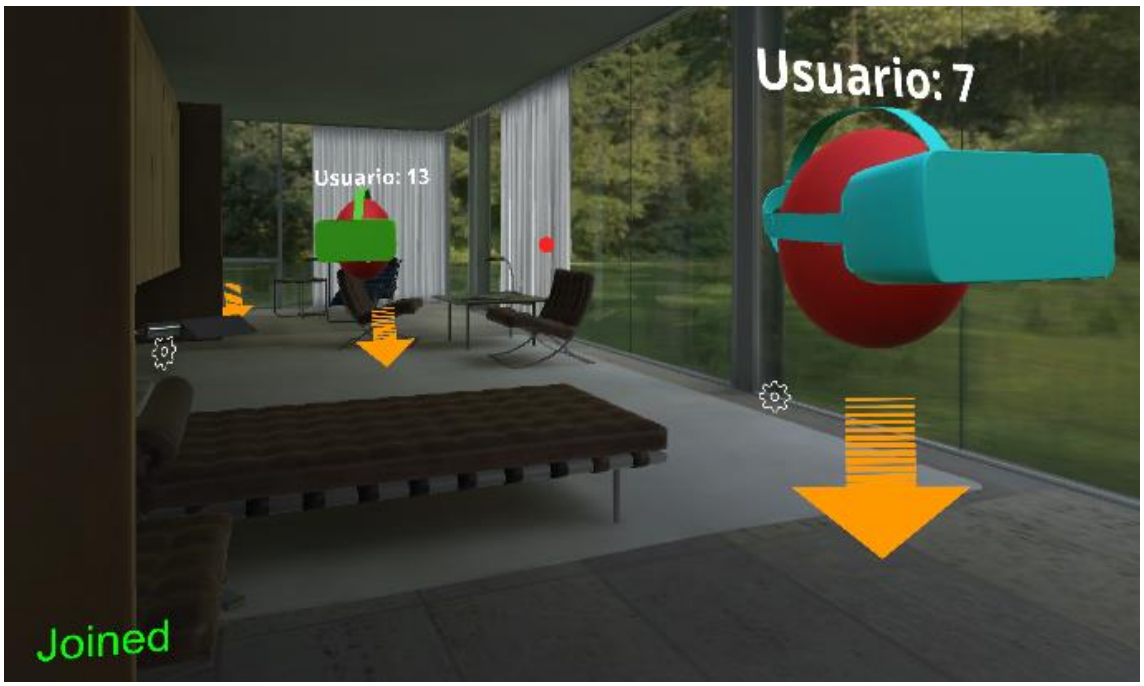


Figura 71: Color de las gafas aleatorio para cada personaje desde la vista del personaje.

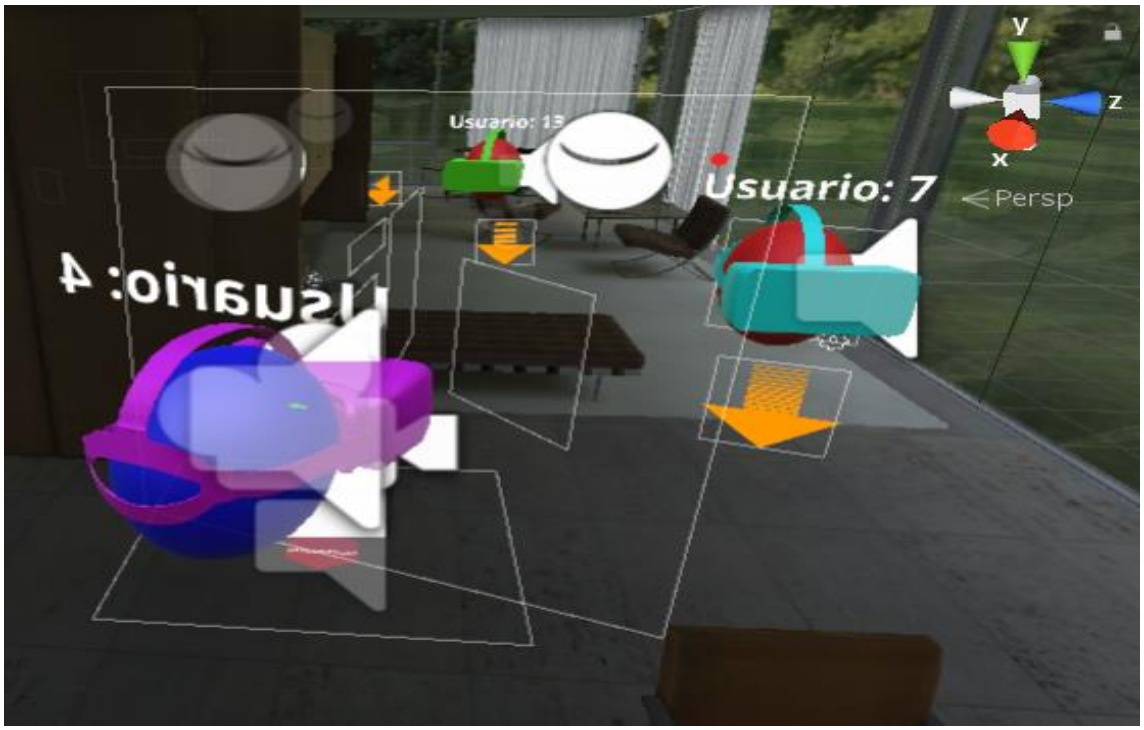


Figura 72: Color de las gafas aleatorio desde la Ventana Inspector de Unity.

3.4 Implementación de la interacción con el entorno

Uno de los principales problemas encontrados con el desarrollo de aplicación en un entorno de realidad virtual ha sido el desarrollar la manera de cómo hacer posible la interacción del usuario con el entorno tridimensional sin la necesidad de utilizar ningún periférico o controlador externo, requiriendo únicamente el dispositivo móvil para realizar toda la interacción de la aplicación.

Es por ello que he utilizado un sistema que únicamente requiera la utilización de la vista para poder interactuar con los objetos, llevándolo a cabo por medio de dos *scripts*, un *script* llamado *ClickRadialPuntero.cs* que permita realizar la función de pulsar los botones de la aplicación y otro *script* llamado *MostrarTexto.cs* que permita visualizar la información en forma de texto apuntado los objetos por el puntero de realidad virtual.

3.4.1 Implementación que permite pulsar botones en la aplicación

El comportamiento del *script* está basado en un contador controlado desde el puntero de realidad virtual de la cámara principal. De manera que, a través de este, se obtenga la información del tiempo que se mantiene observando un objeto y efectué la llamada al evento *pointerClickHandler* proporcionado por *Unity*, el cual actuará como si se hubiese pulsado un botón, o en el caso de dispositivos móviles, como si se hubiese detectado un evento en la pantalla táctil del mismo.

Script: ClickRadialPuntero.cs ver código en el [Anexo 6]
Principales características y funcionalidades implementadas en el *script* asignado al *GameObject Main Camera*: **Objetivo del *script*:** El objetivo del siguiente *script* es el de implementar la interacción que permita pulsar los botones diseñados para su utilización en la aplicación, de modo que no se requiera ningún evento capturado por medio de ningún periférico o controlador externo. **Funcionalidades del *script*:** • Permitir pulsar los botones de la aplicación

Tabla 17: Resumen del script ClIckRadialPuntero.cs.

Permitir pulsar los botones de la aplicación:

En el momento que el puntero es dirigido a través de la mirada del usuario y este detecta un objeto diseñado para su interacción, se activa el contador por medio del evento *OnPointerEnter* de *Unity*, junto con una barra de progreso circular alrededor del puntero que sirva como referencia para determinar el progreso hasta que sea iniciado el evento pointerClickHandler y se apriete el botón deseado.

Para la creación de la barra de progreso se ha creado una imagen circular a la que he llamado *BarraProgreso* y la he incluido en el *Canvas* de la cámara principal de la aplicación, de modo que quede centrada en el centro de la vista ofrecida por la cámara.

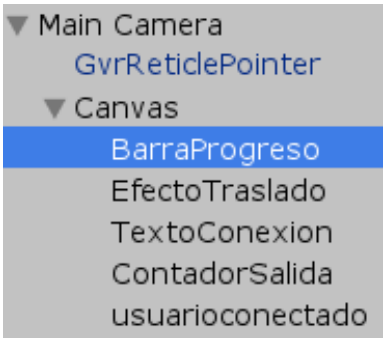


Figura 73: GameObject BarraProgreso hijo del Canvas de la cámara principal.

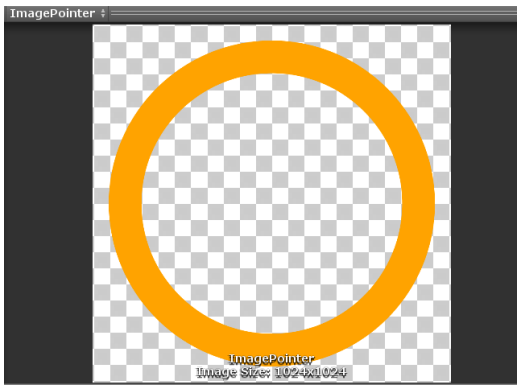


Figura 74: GameObject BarraProgreso.

La imagen se ha configurado con las características específicas para que su contenido pueda ser mostrado de manera incremental durante la ejecución de la aplicación. *Unity* permite utilizar esta opción asignando a la imagen la propiedad *Filled*, como se muestra en la siguiente figura:

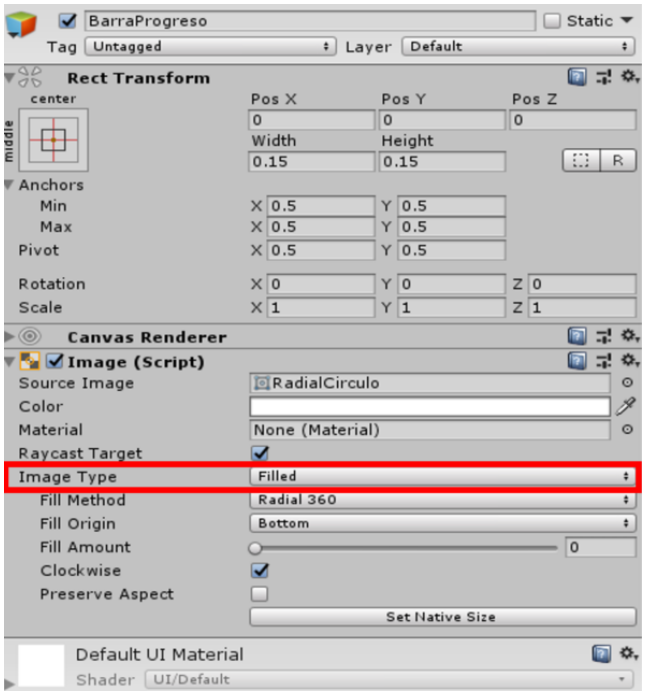


Figura 75: Componentes del GameObject BarraProgreso.

De manera que el contenido de la imagen es controlado por medio del *script ClickRadialPuntero.cs*, que incrementa su contenido en función del valor del contador y lo resetea cuando se ejecuta tanto el evento *pointerClickHandler o OnPointerExit* de la mano de *Unity*.

Fragmento de código del *script ClickRadialPuntero.cs*

```
        if (_ObjDetectado)
        {
            _Contador += Time.deltaTime;
            _BarraProgreso.GetComponent<Image>().fillAmount = _Contador / 3;
            if (_Contador >= _ClickAl)
            {
                ExecuteEvents.Execute(gameObject, new
PointerEventData(EventSystem.current), ExecuteEvents.pointerClickHandler);
                Debug.Log("CLICK");
                ResetearContador();
            }
        }
```

Objetivo del código: detectar objetos de la escena incrementado visualmente la barra de progreso *BarraProgreso* para realizar la interacción con el objeto.

Tabla 18: Implementación para el incremento de la barra de progreso.

En las figuras mostradas a continuación se puede observar como en el momento que se posiciona el puntero de realidad virtual sobre el botón que habilita el menú para la desconexión, la barra de progreso comienza a incrementar hasta llegar a completar el circulo de la imagen a modo de progresión radial. En el momento que la barra de progreso se completa se lanza el evento *pointerClickHandler:*

Figura 76: Proceso de interacción del GameObject BarraProgreso.

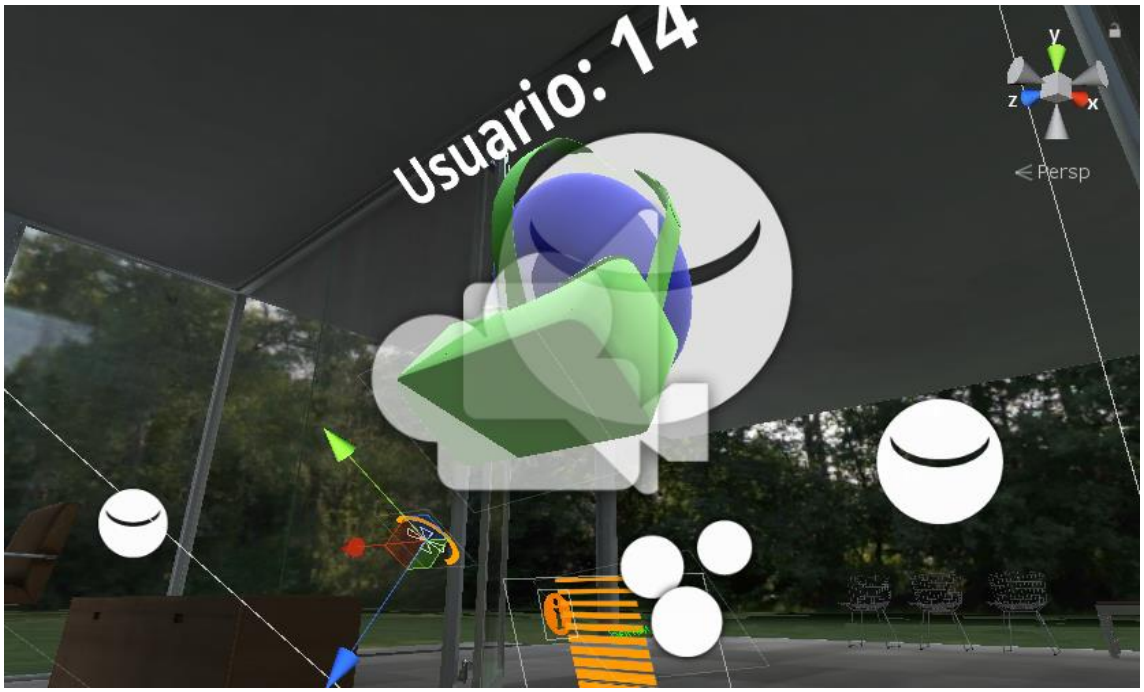


Figura 77: Barra de progreso incrementando desde la ventana Inspector.

1.4.4 Implementación que permite mostrar información de los objetos en forma de texto

Script: MostrarTexto.cs ver código en el [Anexo 7]
Objetivo del *script*: El objetivo del siguiente *script* es el de mostrar en forma de texto el nombre de cada uno de los objetos diseñados para su utilización, de forma que sirva para mostrar información relacionada con el objeto. **Funcionalidades del *script*:** • Permitir mostrar información de los objetos

Tabla 19: Resumen del script MostrarTexto.cs.

Permitir mostrar la información de los objetos:

Se ha desarrollado esta funcionalidad con el objetivo de brindar información al usuario cuando se observe sobre determinados objetos, de manera que sea mostrada y ocultada por los eventos *PointerEnter()* y *PointerExit()* de *Unity* que han sido configurados para habilitar y deshabilitar el *script*.

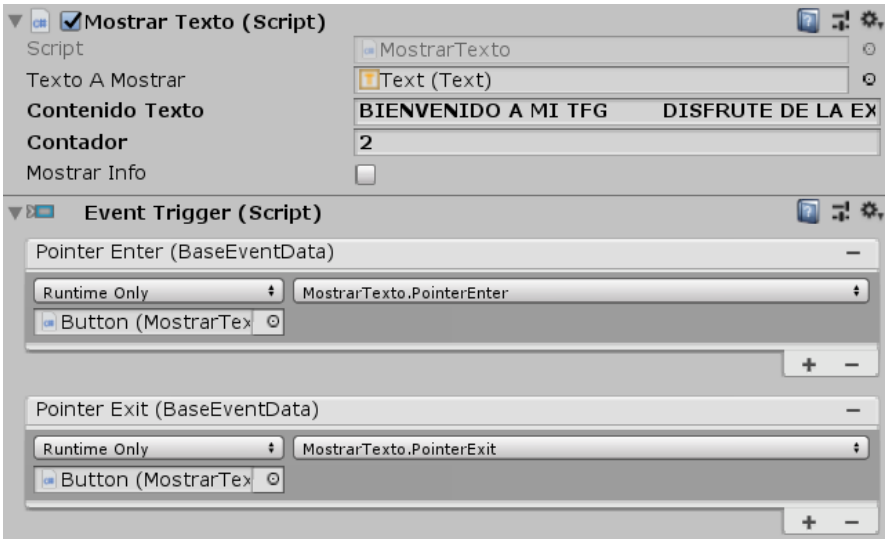


Figura 78: Ejemplo de objeto con el script MostrarTexto.cs y los eventos que lo activan y desactivan.

En las figuras mostradas a continuación se puede observar el proceso del funcionamiento al ser observados distintos objetos que contienen el *script MostarTexto.cs*:

Figura 79: Mostrando información del objeto 1.

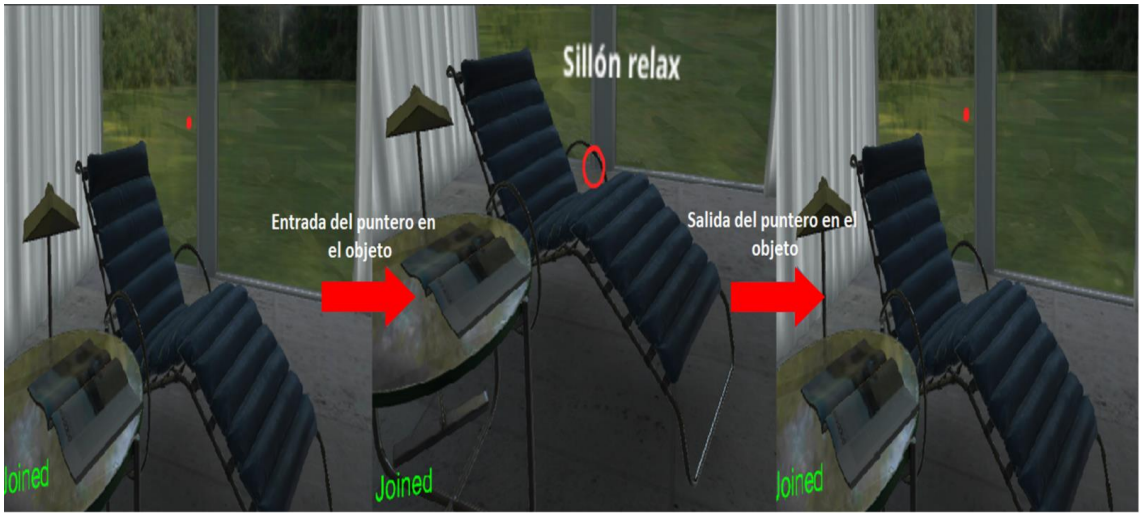


Figura 80: Mostrando información del objeto 2.

Puede observarse como en el momento que el puntero de la cámara principal (utilizado para la realidad virtual) se sitúa sobre los objetos que tienen el *script MostarTexto.cs,* el texto se habilita mostrando la información al usuario.

3.5 Implementación del movimiento

Otro de los grandes retos a resolver durante el desarrollo de la aplicación ha sido proporcionar un sistema de movimiento sin la necesidad de utilizar ningún controlador externo conectado al dispositivo móvil que me permita desplazarme por la escena inmobiliaria. Inicialmente analicé un desarrollo que se llevara a cabo por medio de un desplazamiento que me permitiera andar libremente por la escena a través de la utilización de un controlador o mando a modo de videojuego, pero al poner a prueba esta solución no me acabó de convencer por las sensaciones de mareo que presentaba al realizar las pruebas en realidad virtual. Obteniendo información y estudiando posibles opciones observé que hay estudios científicos que demuestran que ciertas implementaciones de movimiento pueden producir sensaciones de mareo sobre el usuario que está haciendo uso de la realidad virtual. Esto es debido a que el usuario está recibiendo constantemente señales de movimiento, pero en cambio realmente no se está realizando ningún cambio en la postura ni desplazamiento de la persona, aunque se esté simulando, está información recibida por el cerebro causa un desequilibrio en el sistema vestibular provocando sensaciones de desorientación y mareos [17].

Solución propuesta:

La solución propuesta pasa por la implementación de un sistema de movimiento que implique el mínimo desplazamiento para minimizar las sensaciones de mareo, simulando un transporte del usuario desde un punto a otro. Esta implementación ha sido desarrollada por medio del *script Teletransporte.cs* acompañado de una animación que simule un desvanecimiento en negro conforme se ejecute el desplazamiento, de modo que la sensación de desplazamiento sea más confortable.

Script: Teletransporte.cs ver código en el [Anexo 8]
Objetivo del *script*: El objetivo del siguiente *script* es el de implementar el movimiento del usuario en la escena inmobiliaria ofreciendo un sistema de desplazamiento sin necesidad de utilizar mando o controlador externo. **Funcionalidades del *script*:** • Implementación del movimiento en la escena inmobiliaria

Tabla 20: Resumen del script Teletransporte.cs.

Implementación del movimiento en la escena inmobiliaria:

El *script* está diseñado para ser asignado a la cámara principal de la aplicación, ya que el funcionamiento del *script* se basa en la generación del método *DispararRaycast()*, el cual al ser llamado dispara un vector o comúnmente conocido en *Unity* como *raycast* desde el centro de la cámara y se obtienen las coordenadas del objeto colisionado. Una vez obtenidas las coordenadas de la colisión, se desplaza al usuario a ese punto manteniéndolo siempre en el eje de abscisas tridimensionales *Y* constante, para evitar desplazamientos por encima del suelo de la casa inmobiliaria.

La siguiente figura representa un esquema sencillo para ayudar a la compresión de la mecánica de funcionamiento del *script*:

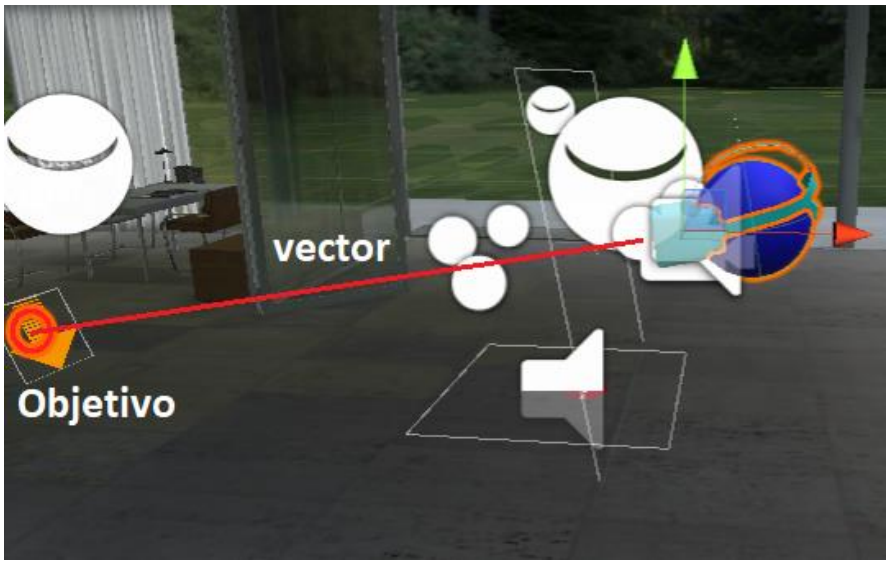


Figura 81: Esquema de funcionamiento del script Teletransporte.cs.

Fragmento de código del *script Teletransporte.cs*

```
public void DispararRaycast()
    {
        Ray _raycast = _Camara.ViewportPointToRay(new Vector3(0.5F, 0.5F, 0));
        RaycastHit _objetivo;
        if (Physics.Raycast(_raycast, out _objetivo))
        {
            if (_objetivo.transform != null)
            {

                Vector3 newLocation = new Vector3(_objetivo.point.x, 2.7f,
_objetivo.point.z);
                Debug.Log(newLocation);

                _Usuario.transform.localPosition = newLocation;

                _ATeletransporte.Play();

            }
        }
    }
```

Objetivo del código: generar un vector desde la posición central de la cámara que colisione con el objetivo deseado y obtenga las coordenadas de su posición para efectuar el traslado del usuario a las nuevas coordenadas.

Tabla 21: Implementación para el traslado.

Para la generación de los objetos objetivo de los cuales obtener las coordenadas para el traslado se han creado los siguientes botones llamados *botones de movimiento*:

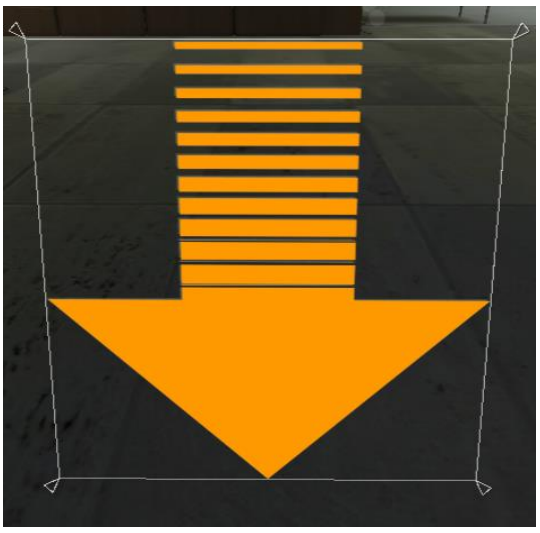


Figura 82: GameObject Botón Movimiento.

Los botones están disponibles en diferentes posiciones del entorno inmobiliario con el objetivo de ofrecer la posibilidad de desplazar al usuario a estos puntos. Estos botones de movimiento vienen implementados con la interacción desarrollada en el *script ClickRadialPuntero.cs* de modo que cuando sean apretados se realice el desplazamiento.

Componentes asignados al *GameObject Boton Movimiento:*

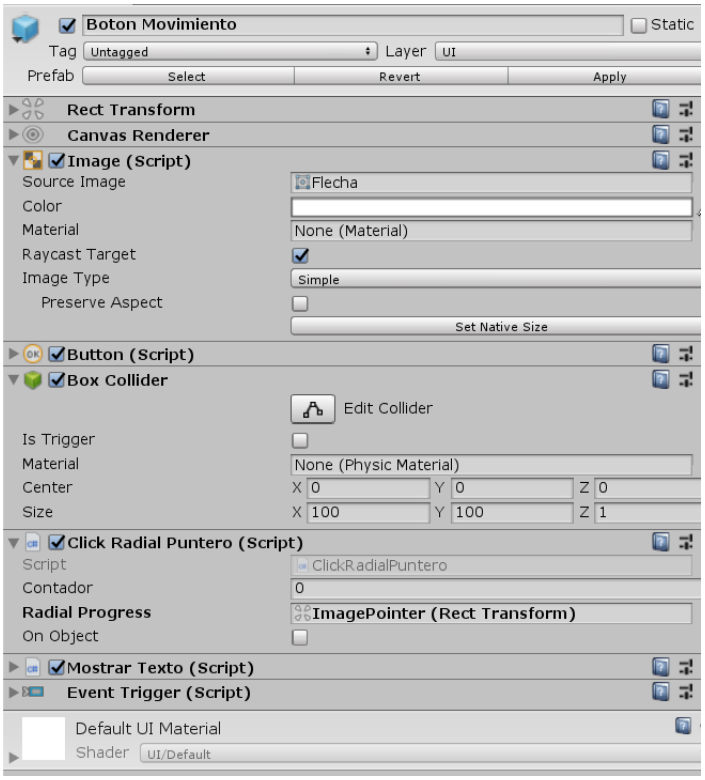


Figura 83: Componentes del GameObject Boton Movimiento.

La animación creada es reproducida en el momento que se realiza el desplazamiento realizando un desvanecimiento en negro en la pantalla del usuario:

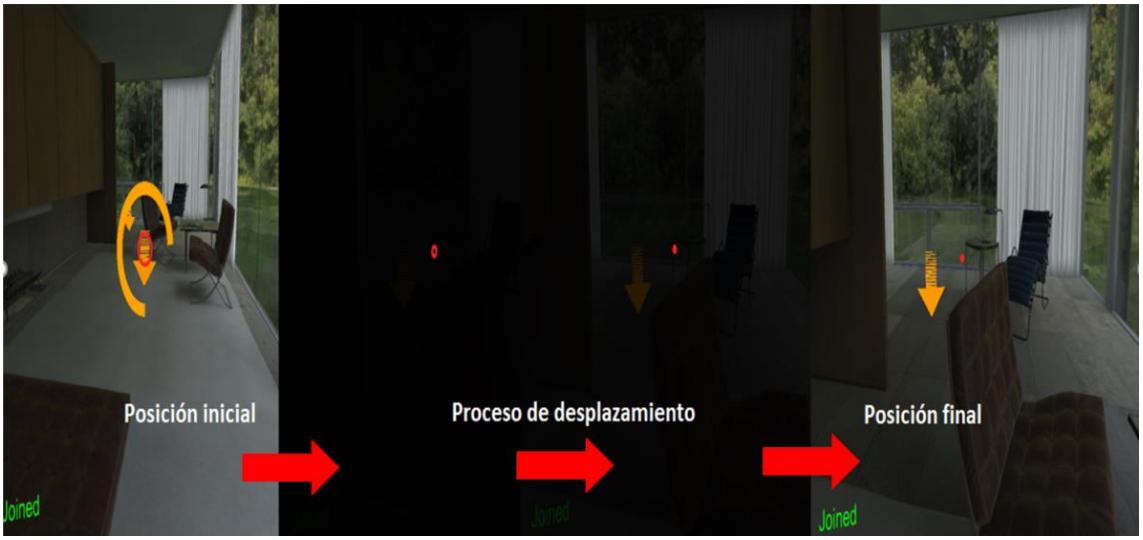


Figura 84: Proceso de desplazamiento del usuario.

3.6 Implementación de modificaciones en tiempo real

Con el principal objetivo de ofrecer al usuario la posibilidad de poder realizar modificaciones del entorno inmobiliario en tiempo real, he desarrollado la funcionalidad que permita la edición del color de cada uno de los muebles disponibles en la casa, para ello he diseñado un menú con diferentes paletas de colores llamado *Editor Mueble*, donde de esta manera se le proporcione al usuario la capacidad de escoger entre diferentes colores y personalizar el color de los muebles del entorno inmobiliario, escogiendo entre las diferentes opciones de colores disponibles.

El diseño del *Editor Mueble* puede observarse la figura mostrada continuación:

Figura 85: Menú utilizado para la edición del color del mueble.

La estructura de objetos y herencia creada para el menú es la siguiente:

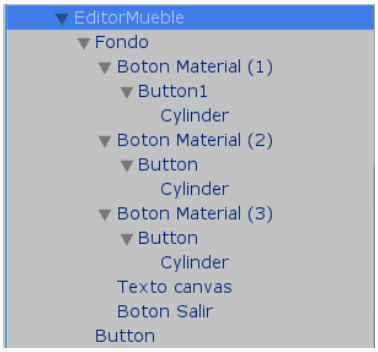


Figura 86: Jerarquía del GameObject EditorMueble.

Los botones que conforman la estructura del menú de selección de colores pueden ser presionados gracias a la implementación desarrollada para la interacción con el entorno de la aplicación. Accesible en el capítulo 3.4 de esta memoria.

El menú de selección de colores se encuentra disponible para el usuario a través del botón que habilita y activa el menú, ya que el menú se encuentra desactivado hasta que el usuario no haya presionado este botón:

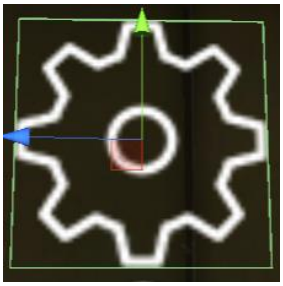


Figura 87: Botón que permite habilitar el menú EditorMueble.

De igual manera que se dispone de un botón para activar el menú, se dispone del botón que permite cerrar el menú una vez que el usuario haya decidido terminar con la edición del color del mueble:

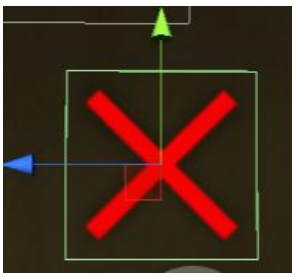


Figura 88: Botón que permite cerrar el menú EditorMueble.

El contenido principal del menú se encuentra disponible a través de los botones que obtienen visualmente la paleta de colores disponible para la edición del mueble:

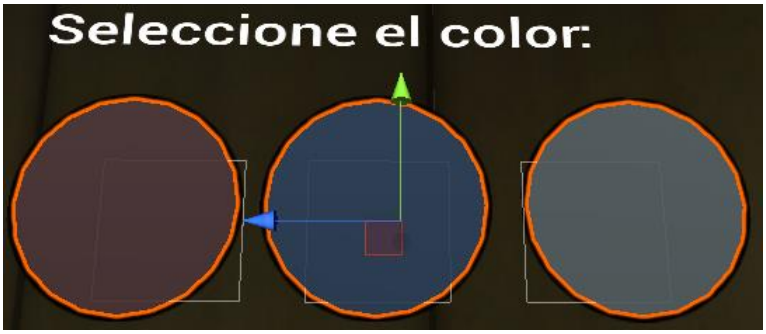


Figura 89: Botones para la selección del color del mueble.

La programación que permite realizar el cambio de color sobre el mueble deseado viene implementada por medio del *script ColorObjeto.cs,* cuyo funcionamiento permite que en el momento que el usuario haya presionado sobre uno de los colores disponibles, se realice un cambio del color sobre el mueble que contiene el *script*.

Script: ColorObjeto.cs ver código en el [Anexo 9]
Principales características y funcionalidades implementadas en el *script* asignado a los *GameObjects* que conforman los muebles de la aplicación: **Objetivo del *script*:** El objetivo del siguiente *script* es el de implementar la opción que permita la selección del color del mueble, de manera que el cambio no sea únicamente percibido sobre el usuario local, sino que el color sea sincronizado y percibido en tiempo real por cada uno de los usuarios conectados en la aplicación. **Funcionalidades del *script*:** • Implementación del cambio de color • Sincronización del color en red

Tabla 22: Resumen del script ColorObjeto.cs.

Implementación del cambio de color:

El *script* almacena la información de cada uno de los colores sobre los que se quiere realizar el cambio y en función del parámetro pasado en el método *CambiarColor()* se realiza el cambio de color sobre del objeto portador del *script*, en este caso el mueble.

El parámetro es capturado por medio de la programación de los eventos disponibles en *Unity* a través del botón, cuya configuración para cada uno de ellos sigue el esquema utilizado en el siguiente botón de selección de color de ejemplo:

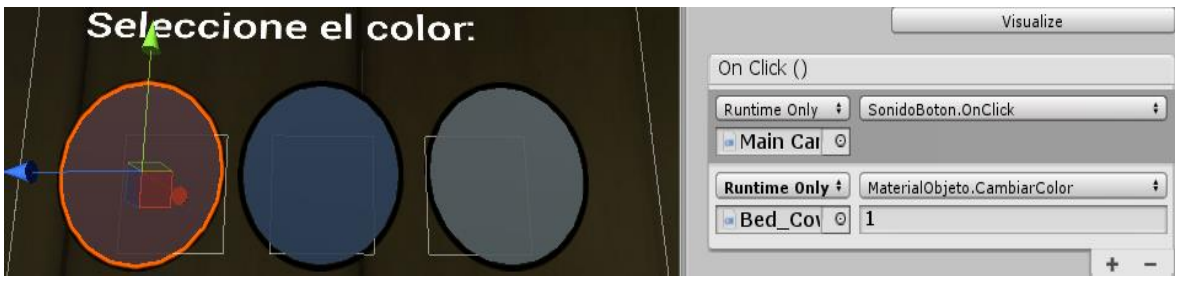


Figura 90: Introducción del parámetro que habilita el color escogido.

Fragmento de código del *script ColorObjeto.cs*

```
public void CambiarColor(int i)
    {
        switch (i)
        {
            case 1:
                this.GetComponent<MeshRenderer>().material.color = _Color1;
                break;
            case 2:
                this.GetComponent<MeshRenderer>().material.color = _Color2;
                break;
            case 3:
                this.GetComponent<MeshRenderer>().material.color = _Color3;
                break;
        }
    }
```

Objetivo del código: evaluar el parámetro utilizado para la selección del color y en función de este aplicar el color al objeto portador del *script*.

Tabla 23: Implementación del código del cambio de color del objeto.

En la figura mostrada a continuación puede observarse el proceso de la implementación para el cambio de color de la cama, desde la activación del menú que habilita la paleta de colores, hasta la elección del color y su posterior cambio en la cama.

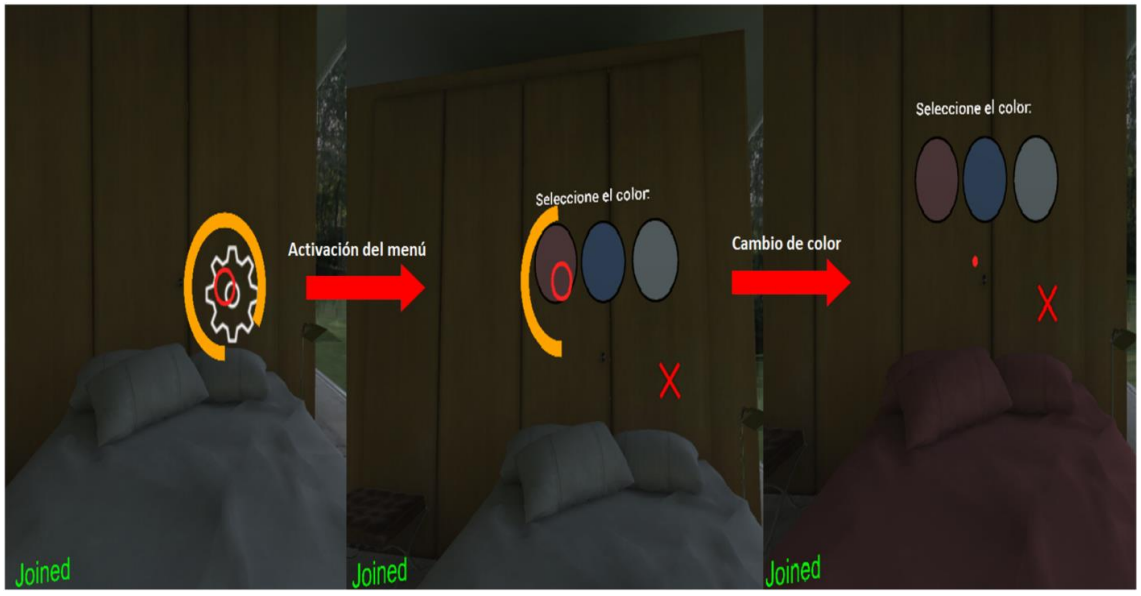


Figura 91: Proceso del cambio de color de la cama.

Pero como se refleja en las figuras inferiores este cambio solo es percibido por el usuario local, ya que la información no es enviada y sincronizada por el resto de usuarios conectados en la aplicación. Es por tanto necesario desarrollar una implementación que permita la sincronización del color del objeto en red.

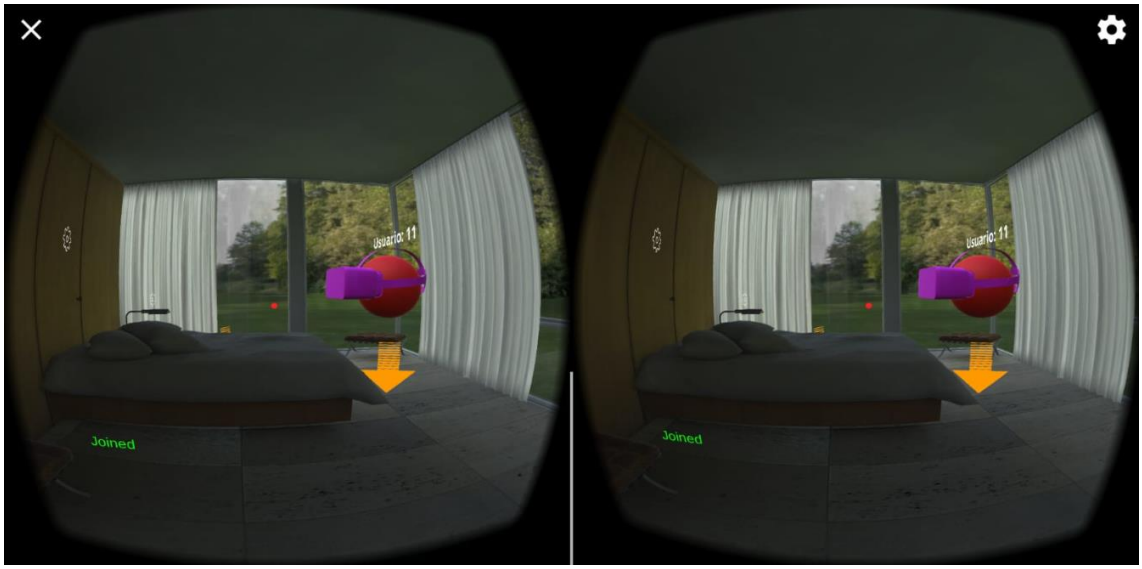


Figura 92: Cambio del color de la cama no sincronizado en red.

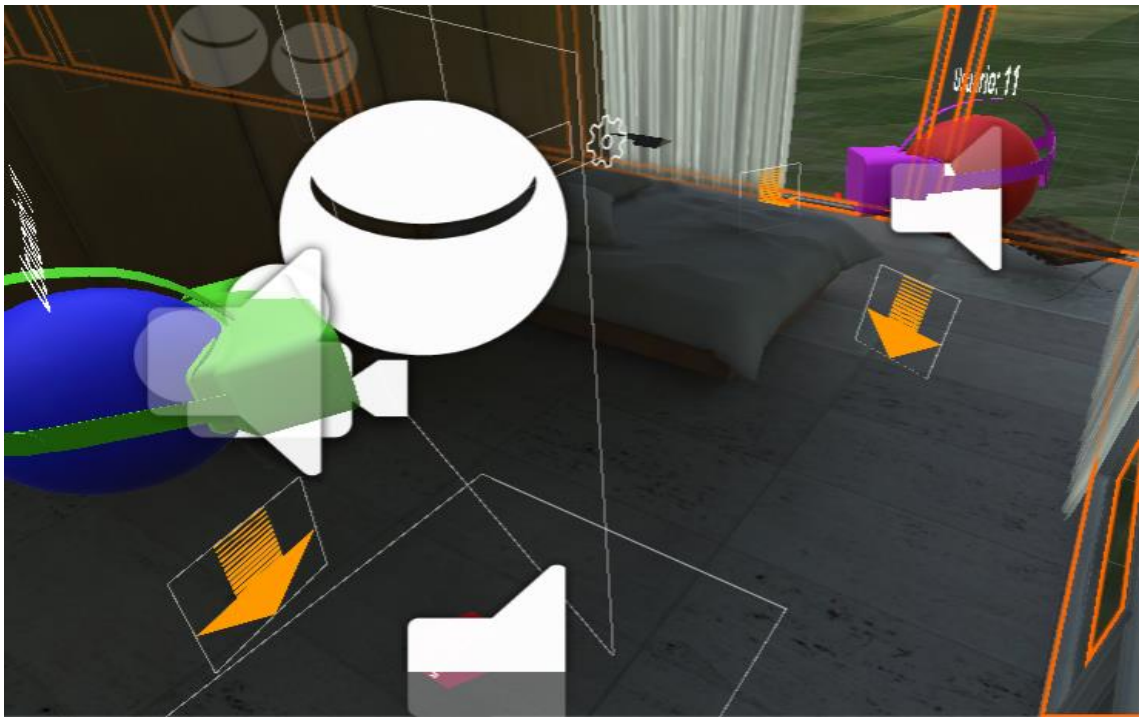


Figura 93: Cambio del color no sincronizado en red desde la ventana Ispector.

Sincronización del color en red:

Para que el cambio del color de los muebles sea sincronizado y percibido en tiempo real por cada uno de los usuarios conectados en la aplicación, es necesario, enviar a cada uno de los usuarios no locales la información con los parámetros del color escogido. Esta información ha sido almacenada en una variable de *Unity* del tipo *Vector3 (x, y, z)*, de modo que almacene los parámetros del color *RGB* escogido. La información es enviada al resto de usuarios conectados y la almacenan en la variable creada *_ColorSinc*. Finalmente, se le aplica el color almacenado en *_ColorSinc* al objeto.

Fragmento de código del *script ColorObjeto.cs*

```
public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
    {
        if (stream.isWriting)
        {
            //Envía la información del color.
            _InfoColor = new
Vector3(this.GetComponent<MeshRenderer>().material.color.r,
this.GetComponent<MeshRenderer>().material.color.g,
this.GetComponent<MeshRenderer>().material.color.b);

            stream.Serialize(ref _InfoColor);
        }
        else
        {
            //Obtiene la información del color.
            stream.Serialize(ref _InfoColor);

            _ColorSinc = new Color(_InfoColor.x, _InfoColor.y, _InfoColor.z,
1.0f);

        }
    }
```

Objetivo del código: sincronizar el cambio de color con el resto de usuarios conectados.

Tabla 24: Implementación de la sincronización con el resto de usuarios del cambio de color.

Las siguientes figuras muestran como la sincronización del color en red se realiza correctamente:

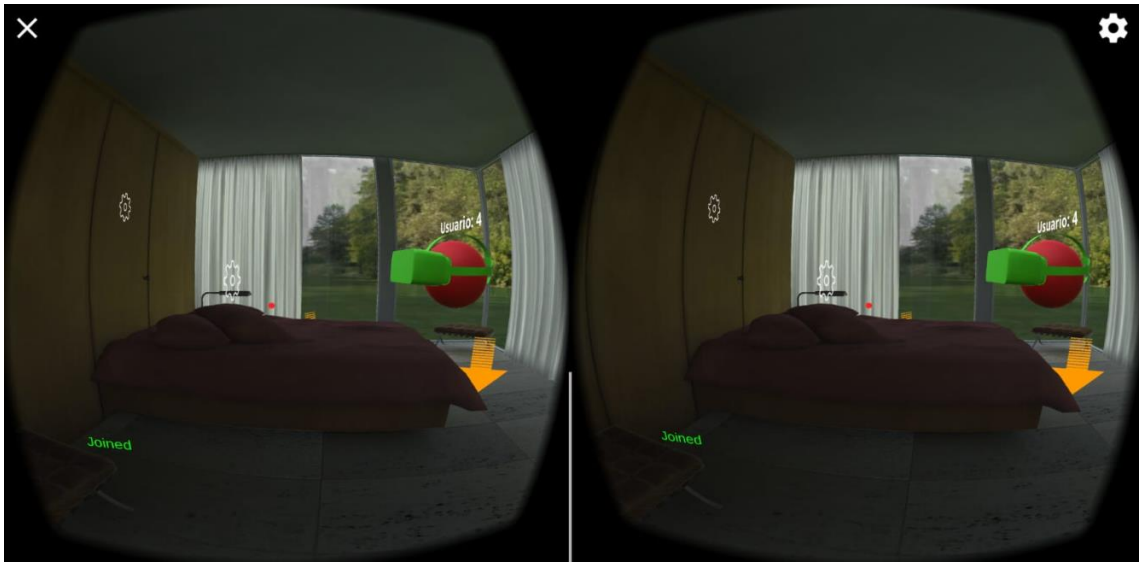


Figura 94: Cambio de color de la cama sincronizado en red.

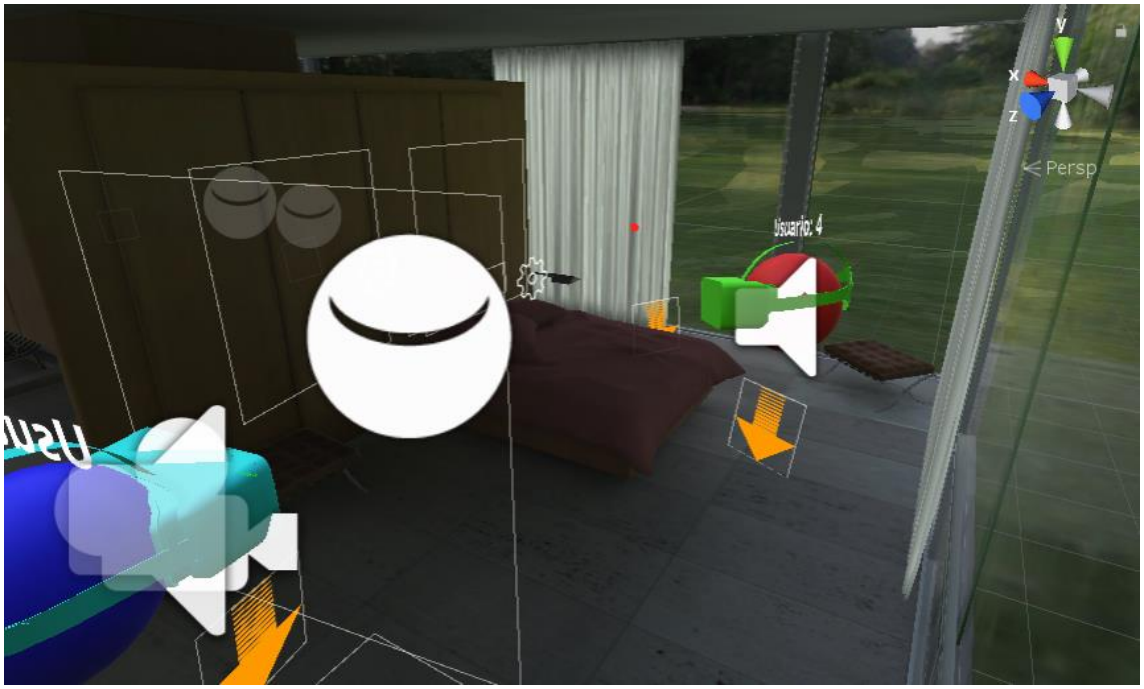


Figura 95: Cambio de color sincronizado en red desde la ventana Inspector.

3.7 Implementación de la base de datos

Con el objetivo de realizar un control del tiempo de conexión de cada usuario en la aplicación, se ha configurado un servidor *Apache* (para ejecutar *scripts* en *PHP* y tener acceso a la base de datos a través de *PhpMyAdmin*) y una base de datos *MySQL* por medio de *XAMMP* que almacene la información de fecha de entrada y la fecha de salida de cada usuario en la aplicación. Esta base de datos es accedida a través de *Unity* por medio del *script UsuarioConectado.cs*, utilizando para ello dos ficheros externos en *PHP, RegistrarEntradaUsuario.php y RegistrarSalidaUsuario.php,* los cuales son los encargados de obtener los datos del *script* asignado al personaje y almacenarlos mediante *SQL* en la base de datos. Inicialmente se planteó una solución que no necesitara de la utilización de ficheros intermediarios en *PHP* para la inserción de datos, utilizando para ello librerías *dll*. Sin embargo, esta solución no es válida para el desarrollo de aplicaciones con sistema operativo *Android*, ya que las librerías *dll* son propias del sistema operativo *Microsoft Windows*.

La base de datos creada se denomina *aplicacion_tfg* y es accesible a su gestión a través del servidor *Apache*. En ella se han creado la siguiente tabla llamada *usuarios*, de modo que esta es utilizada para almacenar la información proveniente de los usuarios de la aplicación:

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
usuarios	Examinar Estructura Buscar Insertar Vaciar Eliminar	8	InnoDB	latin1_spanish_ci	16 KB	-
1 tabla	Número de filas	8	InnoDB	latin1_spanish_ci	16 KB	0 B

Figura 96: Tabla utilizada para almacenar la información de los usuarios.

La tabla *usuarios* contiene los siguientes campos:

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	id	int(11)			No	*Ninguna*		AUTO_INCREMENT	Cambiar Eliminar Más
2	nombre_usuario	varchar(50)	latin1_spanish_ci		No	*Ninguna*			Cambiar Eliminar Más
3	fecha_entrada	datetime			No	*Ninguna*			Cambiar Eliminar Más
4	fecha_salida	datetime			No	*Ninguna*			Cambiar Eliminar Más

Figura 97: Campos creados para almacenar los datos de la tabla usuarios.

Donde la función de cada uno de los campos es la siguiente.

- ***Id***: utilizado como clave primaria para la identificación de cada uno de los usuarios a través de un entero.
- ***nombre_usuario***: campo utilizado para almacenar el nombre de los usuarios de la aplicación por medio del tipo *varchar*.
- ***fecha_entrada***: su función es la de almacenar la fecha de entrada del usuario en la aplicación obteniendo el valor en formato *datetime.*
- ***fecha_salida***: campo cuyo objetivo es el de almacenar la fecha de desconexión de los usuarios de la aplicación, obteniendo al igual que en el campo *fecha_entrada* el valor en formato *datetime.*

Los *scripts* utilizados para la inserción de datos en la base de datos son los siguientes:

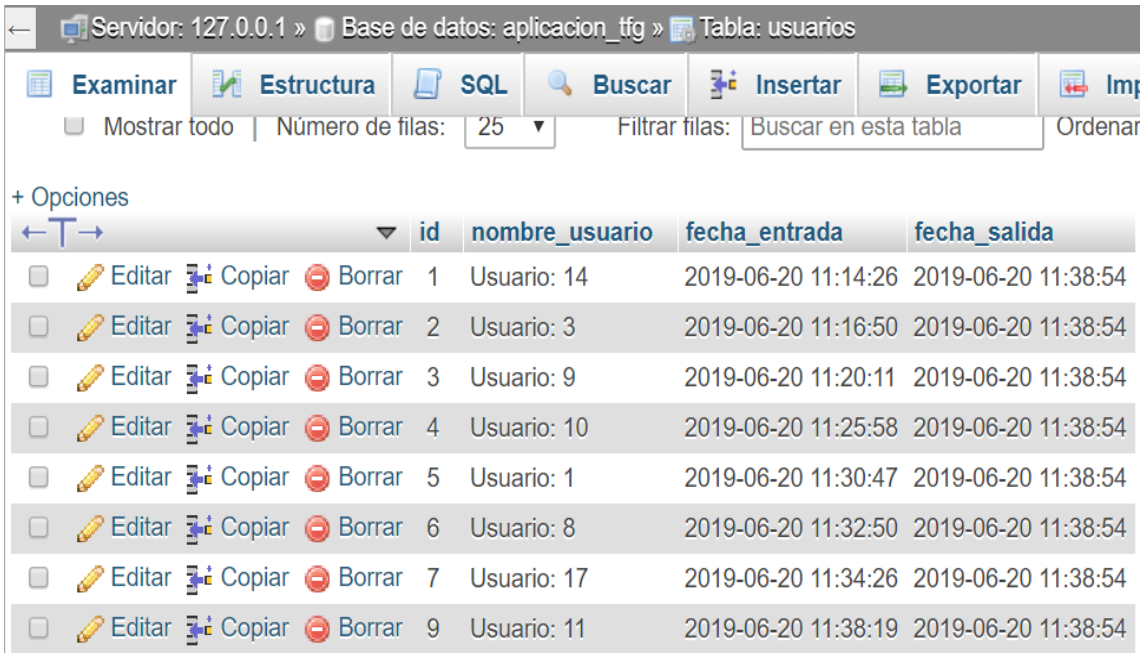
Script: RegistrarEntradaUsuario.php ver código en el [Anexo 10]
Objetivo del *script*: El objetivo del siguiente *script* es el de obtener el nombre de usuario proveniente del *script* asignado al personaje *UsuarioConectado.cs* e insertarlo en la base de datos *aplicación_tfg en el campo nombre_usuario* junto con la fecha de entrada en la aplicación en fecha_entrada. **Funcionalidades del *script*:** • Insertar la fecha de entrada de los usuarios en la aplicación cuando se conecten.

Tabla 25: Resumen del script RegistrarEntradaUsuario.php.

Script: RegistrarSalidaUsuario.php ver código en el [Anexo 11]
Objetivo del *script*: El objetivo del siguiente *script* es el de actualizar la fila del usuario creada en la entrada de la aplicación, introduciendo en el campo *fecha_salida* la fecha en la que se realiza la desconexión de cada usuario en la aplicación. **Funcionalidades del *script*:** • Insertar la fecha de salida de los usuarios en la aplicación cuando se desconecten.

Tabla 26: Resumen del script RegistrarSalidaUsuario.php.

La siguiente imagen permite observar cómo se realiza la inserción de datos en la base de datos creada para la aplicación, almacenando la fecha de entrada y la fecha de salida de diferentes usuarios:



	id	nombre_usuario	fecha_entrada	fecha_salida
Editar Copiar Borrar	1	Usuario: 14	2019-06-20 11:14:26	2019-06-20 11:38:54
Editar Copiar Borrar	2	Usuario: 3	2019-06-20 11:16:50	2019-06-20 11:38:54
Editar Copiar Borrar	3	Usuario: 9	2019-06-20 11:20:11	2019-06-20 11:38:54
Editar Copiar Borrar	4	Usuario: 10	2019-06-20 11:25:58	2019-06-20 11:38:54
Editar Copiar Borrar	5	Usuario: 1	2019-06-20 11:30:47	2019-06-20 11:38:54
Editar Copiar Borrar	6	Usuario: 8	2019-06-20 11:32:50	2019-06-20 11:38:54
Editar Copiar Borrar	7	Usuario: 17	2019-06-20 11:34:26	2019-06-20 11:38:54
Editar Copiar Borrar	9	Usuario: 11	2019-06-20 11:38:19	2019-06-20 11:38:54

Figura 98: Datos almacenados en la base datos por diferentes usuarios.

3.8 Implementación del chat de voz

En este capítulo se detallan los pasos realizados para la configuración y despliegue del chat de voz en la aplicación, con el objetivo de ofrecer un sistema de comunicación en tiempo real basado en la transmisión de voz *IP* con los usuarios conectados.

Para comenzar con la implementación ha sido necesario importar en el proyecto de la aplicación el paquete *Photon Voice Unity* o también conocido como *PUNVoice* ofrecido por *Photon Unity Networking (PUN)* disponible gratuitamente en Unity. Se trata de un complemento que permite la comunicación de los usuarios a través de un chat de voz, utilizando para ello un servidor dedicado únicamente para las transmisiones de audio.

Una vez importado el paquete en el proyecto de la aplicación se procede a la creación del servidor para la transmisión de audio en la cuenta personal de *PhotonEngine*, desde la cual se pueden crear, borrar y administrar cada uno de los servidores *(PUN).*

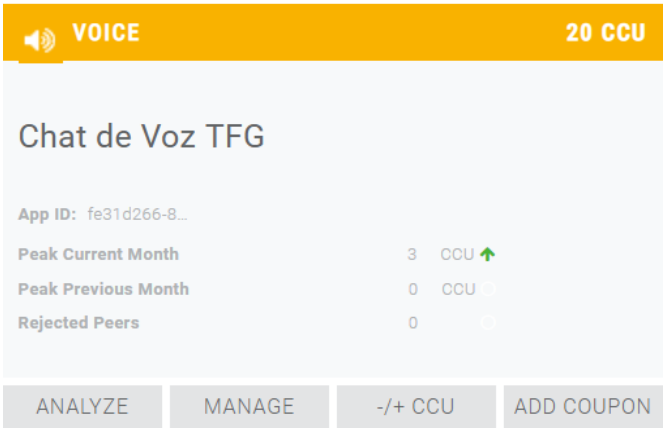


Figura 99: Servidor utilizado para el chat de voz.

Creado el servidor es necesario introducir su identificador en el fichero *PhotonServerSettings*, de esta manera el tráfico de voz enviado durante la transmisión de cada usuario será gestionado desde este.

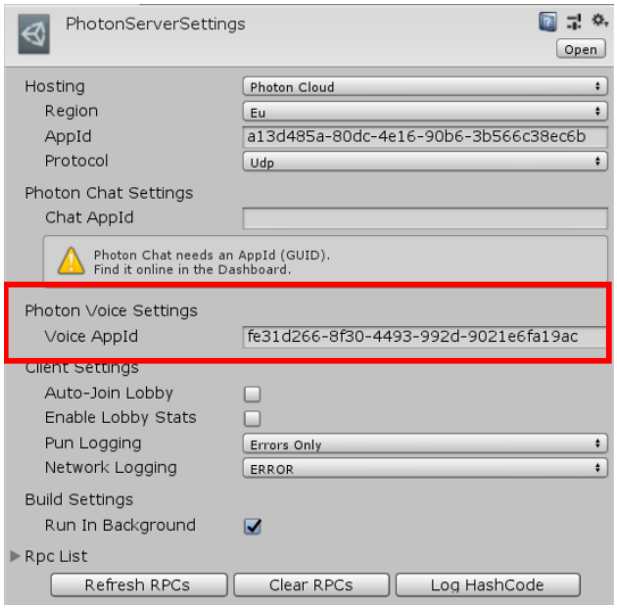


Figura 100: Inserción del ID del servidor de voz.

A continuación, se insertan los componentes proporcionados por *PUNVoice* en el personaje utilizado por el usuario, estos componentes utilizados sobre el personaje del usuario van a ser los encargados de gestionar la información recibida por el micrófono y por el altavoz.

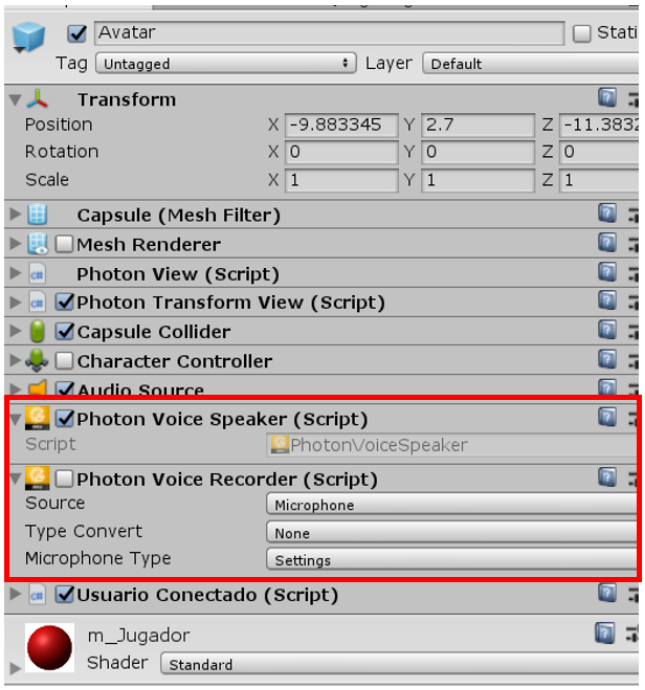


Figura 101: Componentes asignados al personaje para el chat de voz.

Es importante desactivar el micrófono por defecto ya que debe ser controlado a través del *script ConexionJugador.cs,* de modo que únicamente se pueda captar la voz recibida por el usuario local a través del micrófono, de no ser así, se captaría la voz por medio del micrófono de los usuarios no locales, lo que producirían ecos de sonido durante la ejecución hasta el punto de no escucharse nada debido al ruido.

Fragmento de código del *script UsuarioConectado.cs*

```
private void Start()
    {
        if (photonView.isMine)
        {
            ...
            //Activar el micrófono únicamente al usuario local.
            GetComponent<PhotonVoiceRecorder>().enabled = true;
            ...
        }
```

Objetivo del código: activar el componente *PhotonVoiceRecorder* del jugador en el caso de ser el usuario local, captando de esta manera la información recibida en el micrófono para ser transmitida en el chat de voz.

Tabla 27: Código que permite la activación del micrófono para escuchar al usuario local.

Por último, es necesaria la modificación del *script ConexionServidor.cs* para que en el momento que se realice la conexión al servidor en el arranque de la aplicación, se identifiquen cada uno de los usuarios clientes que se conecten al servidor de voz para poder ser gestionados.

Fragmento de código del *script ConexionServidor.cs*

```
   void Start ()
   {
        ...
        //Identificación del cliente conectado al servidor de voz.
        var _IdCliente = PhotonVoiceNetwork.Client;
    }
```

Objetivo del código: Identificación de los clientes en el servidor utilizado en el chat de voz conforme se conecten en el servidor.

Tabla 28: Código que permite la identificación de los clientes en el servidor utilizado en el chat de voz.

3.9 Menú de inicio

Como forma de ofrecer un entorno inicial en el momento de ejecución de la aplicación se ha desarrollado un menú de inicio, que sirva como la interfaz inicial mostrada al usuario para empezar con la navegación en la aplicación.

3.9.1 Panel principal

Esta es la ventana principal del menú y desde la cual es posible acceder resto de paneles habilitados, dispone de tres botones que dan accesibilidad a diferentes contenidos de la aplicación:

- ***Botón Salir***: Utilizado para habilitar el panel de salida y cierre de la aplicación.
- ***Botón Info***: Utilizado para acceder al panel de información de la aplicación.
- ***Botón Acceder***: Es el botón que da acceso a casa inmobiliaria y contenido de la aplicación.

Figura 102: Panel principal.

3.9.2 Panel de salida

Panel accesible a través del *Botón Salir* donde se ofrece la posibilidad de cerrar la aplicación o de volver al *Panel principal*:

Figura 103: Accediendo al panel de salida.

Figura 104: Panel de salida.

3.9.3 Panel de información

Panel accesible a través del botón *Info* donde se muestra información relacionada con la aplicación y el botón habilitado para regresar al *Panel principal*:

Figura 105: Accediendo al panel de información.

Figura 106: Panel de información.

4. CONCLUSIONES

Hoy en día, el usuario se ha vuelto más exigente debido en gran parte a la aparición de nuevas tecnologías. Esa exigencia viene acompañada de un cambio de rol del usuario final, el cual quiere participar en los contenidos y no conformarse únicamente con la mera observación de los mismos. Es por ello que se ha realizado la adaptación a estas nuevas tecnologías por medio de las herramientas que ofrece la aplicación multiusuario de realidad virtual, ofreciendo la capacidad de sumergir al usuario en el entorno inmobiliario y haciéndole participe en la interacción del contenido en un sistema sincronizado en red.

El principal reto encontrado durante el desarrollo ha sido conseguir la integración de las tecnologías utilizadas en la aplicación desarrollada junto con la superación de los diferentes hitos planteados como objetivo, ya que he tenido que llevar a cabo la superación de estos a base de prueba y error para la obtención de los resultados esperados. No obstante, me he sentido totalmente satisfecho con la labor desempeñada, debido en gran parte a que los objetivos estipulados han sido cumplidos con éxito ya que la aplicación desarrollada ha cumplido las expectativas iniciales. Por ello es importante resaltar y no dejar de lado la posibilidad de su explotación a nivel de negocio. Me gustaría dar unas pequeñas pinceladas de lo que se podría llevar a cabo como proyecto empresarial, por ejemplo, en el entorno interiorismo y arquitectura, pequeñas y medianas empresas que quieran hacer uso de la aplicación para un servicio de valor añadido a sus clientes. Otro sector donde la aplicación podría explotarse sería en los entornos educativos, en el que los usuarios construyesen puzles de manera cooperativa en una sala virtual. No obstante, el éxito empresarial de la aplicación va a depender de las demandas y necesidades del propio mercado.

5. PLANES FUTUROS

Completado el proyecto con el desarrollo de la aplicación, a continuación, se incluyen los trabajos futuros y ampliaciones que permitirían obtener mejoras en la aplicación:

- **Agregar mayores funcionalidades de edición en tiempo real.** Con el objetivo de ofrecer al usuario una mayor capacidad de edición en tiempo real, se podría agregar la funcionalidad de poder mover y eliminar cada uno de los objetos del hogar, de manera que en el momento que se realicen los cambios sobre el objeto en el entorno inmobiliario, los datos sean sincronizados con el resto de usuarios conectados. Así como la posibilidad de reemplazar los muebles mediante la creación de un inventario que los almacene por categorías. Finalmente, en este aspecto se podría incluir la posibilidad de cargar diferentes entornos inmobiliarios en tiempo real. Esta funcionalidad permitiría al usuario la posibilidad de poder escoger entre más de un entorno inmobiliario como sala de conexión, pudiendo escoger diferentes modelos de entre un repositorio y cargarlos en tiempo de ejecución.
- **Autenticación con *login* cifrado**. Dada la naturaleza y usabilidad de la aplicación no se ha realizado ningún cifrado de datos en el acceso a la base de datos externa, no obstante, la protección de la información es importante y un aspecto que en trabajos futuros debe tenerse en cuenta, sobre todo si se realiza un acceso con *login* en la aplicación, siendo necesario administrar sus niveles de seguridad para proteger los datos del usuario tratando de ofrecer una mayor confidencialidad, integridad y disponibilidad de los datos. Como posible opción de cifrado podría incluirse el sistema de encriptación *AES*.
- **Exportación de aplicación a otro tipo de plataformas**. Con la finalidad de no centrarse únicamente en un desarrollo focalizado en *Android* y aumentar la capacidad de usabilidad de la aplicación al usuario.
- **Explotación y estudio de la aplicación en un entorno empresarial.** Donde se estudiarían los diferentes casos de uso de la aplicación en función de las necesidades del propio mercado.

6. BIBLIOGRAFÍA

[1] Patiño, A. (2014). De la "Paradoja de la productividad" y la Ley de Moore al papel de las TIC en el aumento de la productividad de las empresas y de las naciones. *INGE CUC*, *10*(2), 51 - 59. Recuperado a partir de https://revistascientificas.cuc.edu.co/ingecuc/article/view/490

[2] Martínez, E. (2001). La evolución de la telefonía móvil. *Revista Red*, *1*, 1-6.

[3] Ortega, G. C. (2013). Implementación de un sistema de control para seguridad de un domicilio en la provincia de Esmeraldas barrio caliente sector centro (Tesis de pregrado). Universidad de las Américas, Quito. Recuperado a partir de http://dspace.udla.edu.ec/handle/33000/3514

[4] Martínez, R. y García-Beltrán, A. (2000). Breve historia de la informática. Universidad Politécnica de Madrid.

[5] Botella C., García-Palacios, A., Baños, R. M. y Quero S. (2007). Realidad Virtual y Tratamientos Psicológicos. *Cuadernos de Medicina Psicosomática*, *82*, 17-31. Recuperado a partir de http://www.terapiacognitiva.eu/cpc/dwl/VR/Cuad%20N82%20trabajo%202.pdf

[6] Miró, J., Nieto, R., y Huguet, A. (2007). Realidad virtual y manejo del dolor. *Cuadernos de Medicina Psicosomática*, *82*, 52-64.

[7] Vázquez-Mata, G. (2008). Realidad virtual y simulación en el entrenamiento de los estudiantes de medicina. *Educación Médica*, *11*(Supl. 1), 29-31. Recuperado a partir de http://scielo.isciii.es/scielo.php?script=sci_arttext&pid=S1575-18132008000500006&lng=es&tlng=es

[8] Flores, J. A., Camarena, P., y Avalos, E. (2014). La Realidad Virtual una Tecnología Innovadora Aplicable al Proceso de Enseñanza de los Estudiantes de Ingeniería. *Apertura: Revista de Innovación Educativa*, *6*(2), 86-99. Recuperado a partir de https://dialnet.unirioja.es/servlet/articulo?codigo=5547077

[9] Sidorenko, P., Calvo, L. M., y Cantero de Julián, J. I. (2018). Marketing y publicidad inmersiva: el formato 360º y la realidad virtual en estrategias transmedia. *Miguel Hernández Communication Journal*, *0*(9), 19-47. doi: http://dx.doi.org/10.21134/mhcj.v0i9.227

[10] Unity, "Products – Unity" [Online]. Available: https://unity3d.com/es/unity [Accessed: 10-Jun-2019]

[11] Unity, "Prefabs – Unity Manual" [Online]. Avaiable: https://docs.unity3d.com/es/current/Manual/Prefabs.html [Accessed: 10-Jun-2019].

[12] Unity, "Unity – Manual: La Vista del Juego (Game View)" [Online]. Avaiable: https://docs.unity3d.com/es/530/Manual/GameView.html [Accessed: 10-Jun-2019].

[13] Unity, "GameObjects – Unity Manual" [Online]. Available: https://docs.unity3d.com/es/current/Manual/GameObjects.html [Accessed: 10-Jun-2019].

[14] Unity, "Introducción a los componentes" [Online]. Available: https://docs.unity3d.com/es/current/Manual/Components.html [Accessed: 10-Jun-2019].

[15] Unity, "Creando y usando scripts – Unity Manual" [Online]. Available: https://docs.unity3d.com/es/current/Manual/CreatingAndUsingScripts.html [Accessed: 10-Jun-2019].

[16] PhotonEngine, "Introduction | Photon Engine" [Online]. Available: https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro [Accessed: 10-Jun-2019].

[17] Guerrero, B. y Valero, L. (2013). Efectos secundarios tras el uso de realidad virtual inmersiva en un videojuego. *International Journal of Psychology and Psychological Therapy*, *13*(2), 163-178. Recuperado a partir de https://dialnet.unirioja.es/servlet/articulo?codigo=4261790

7. ANEXOS

Anexo 1: *Script ConexionServidor.cs*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ConexionServidor : Photon.MonoBehaviour {
    public string _Version;
    public Transform _ZonaInicio;
    //Objeto del tipo TextMesh para obtener el estado de conexión al servidor
    [SerializeField] public TextMesh _TextoConexion;
    public TextMesh _NombreUsuario;

    //Utilizada en la inicialización.
    void Start ()
    {
        PhotonNetwork.ConnectUsingSettings(_Version);
        _ZonaInicio = GameObject.Find("Zona Inicio").transform;
        _TextoConexion = GameObject.Find("TextoConexion").GetComponent<TextMesh>();
        //Identificación del cliente conectado al servidor de voz.
        var _IdCliente = PhotonVoiceNetwork.Client;
    }

    private void OnConnectedToMaster()
    {
        //Accede a una sala en red o la crea en su ausencia.
        PhotonNetwork.JoinOrCreateRoom("Sala", new RoomOptions() { MaxPlayers = 20 }, null);
    }
    void OnJoinedRoom()
    {
        //En el momento se acceda a la sala instancia el personaje Avatar en ella.
        PhotonNetwork.Instantiate("Avatar", _ZonaInicio.transform.position,
_ZonaInicio.transform.rotation, 0);
        //Asignación del nombre de usuario.
        _NombreUsuario = GameObject.Find("Nombre").GetComponent<TextMesh>();
        _NombreUsuario.text = "Usuario: " + Random.Range(0, 20);
        PhotonNetwork.playerName = _NombreUsuario.text;
    }

    //Update se ejecuta por cada frame por segundo en la aplicación.
    void Update()
    {
        //Obtención del estado de conexión.
        _TextoConexion.text = PhotonNetwork.connectionStateDetailed.ToString();
        if (_TextoConexion.text == "Joined")
        {
            _TextoConexion.color = Color.Lerp(_TextoConexion.color, Color.green,
Time.deltaTime);
        }

    }
}
```

Anexo 2: *Script* UsuarioConectado.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;

public class UsuarioConectado : Photon.MonoBehaviour {
    public GameObject _Usuario;
    public Transform _Camara;
    public Transform _PunteroCamara;
    public Transform _CabezaVR;
    public float _ContadorSalida = 5.0f;
    public TextMesh _ObjetoContador;
    public bool _BotonDescPulsado = false;
    public TextMesh _EstadoConexionBBDD;

    //Variables utilizadas para mostrar los usuarios conforme se conectan.
    public TextMesh UsuarioGlobalConectado;
    public float _ContadorUsuarioGlogbalConectado;

    //Variable utilizada para el mennú CanvasBajoJugador.
    public GameObject _CanvasBajoJugador;

    //Utilizada en la inicialización.
    void Start () {
        _CabezaVR = GameObject.Find("Cabeza").transform;
        _Camara = GameObject.Find("Main Camera").transform;

        if (photonView.isMine)
        {
            //Si soy usuario local cambiar nombre a Usuario Local.
             gameObject.name = "Usuario Local";
            _Usuario.GetComponent<MeshRenderer>().enabled = false;
            _PunteroCamara = GameObject.Find("GvrReticlePointer").transform;
            _CabezaVR.transform.SetParent(_Camara);
            _CabezaVR.transform.localRotation = _PunteroCamara.localRotation;
            _Camara.transform.parent = this.transform;
            _Camara.localPosition = Vector3.zero;

            //Encuentra el objeto utilizado para el contador de la desconexión.
            _ObjetoContador = GameObject.Find("ContadorSalida").GetComponent<TextMesh>();
            //Llamada a la corrutina para registrar la entrada del usuario.
            StartCoroutine(RegistrarEntradaUsuario(PhotonNetwork.player.NickName));
            //Encuentra el objeto utilizado para mostrar los usuarios conforme se conectan
            UsuarioGlobalConectado =
GameObject.Find("usuarioconectado").GetComponent<TextMesh>();

            //El menú CanvasBajoJugador solo debe ser visto por el jugador local.
            _CanvasBajoJugador.SetActive(true);

            //Activar el micrófono únicamente al usuario local.
            GetComponent<PhotonVoiceRecorder>().enabled = true;
        }
        else
        {
            //El usuario no local identificado como global.
            gameObject.name = "Usuario Global";
            _CabezaVR.transform.rotation = _Camara.transform.rotation;
```

```
            //Si el jugador no es local, el menú CanvasBajoJugador no tiene que verse.
            _CanvasBajoJugador.SetActive(false);
        }

       }
    IEnumerator RegistrarEntradaUsuario(string _NombreUsuario)
    {
        WWWForm form = new WWWForm();
        //El primer parámetro es la variable del fichero PHP a enviar el dato.
        //El segundo parámetro es el dato que quiero enviar.
        form.AddField("UsuarioRegistrado", _NombreUsuario);

        using (UnityWebRequest www =
UnityWebRequest.Post("http://127.0.0.1/Unity/RegistrarEntradaUsuario.php", form))
        {
            yield return www.SendWebRequest();

            if (www.isNetworkError || www.isHttpError)
            {
                Debug.Log(www.error);
            }
            else
            {
                Debug.Log(www.downloadHandler.text);
            }
        }
    }

    public void Desconectar()
    {
        if (photonView.isMine)
        {
            _BotonDescPulsado = true;
        }
    }

    IEnumerator RegistrarSalidaUsuario(string _NombreUsuario)
    {
        WWWForm form = new WWWForm();
        //El primer parámetro es la variable del fichero PHP a enviar el dato.
        //El segundo parámetro es el dato que quiero enviar.
        form.AddField("UsuarioRegistrado", _NombreUsuario);

        using (UnityWebRequest www =
UnityWebRequest.Post("http://127.0.0.1/Unity/RegistrarSalidaUsuario.php", form))
        {
            yield return www.SendWebRequest();

            if (www.isNetworkError || www.isHttpError)
            {
                Debug.Log(www.error);
            }
            else
            {
                Debug.Log(www.downloadHandler.text);
            }
        }
    }
```

```
    //Update se ejecuta por cada frame por segundo en la aplicación.
    void Update ()
    {
        if (photonView.isMine)
        {
            if (_BotonDescPulsado)
            {
                if(_ContadorSalida > 0.0f)
                {
                _ContadorSalida -= Time.deltaTime;
                _ObjetoContador.text = _ContadorSalida.ToString("0");
                }
                else
                {
                _ObjetoContador.text = "0";
                StartCoroutine(RegistrarSalidaUsuario(PhotonNetwork.player.NickName));
                PhotonNetwork.Disconnect();
                Application.Quit();
                }
            }
            foreach(PhotonPlayer _Usuario in PhotonNetwork.otherPlayers)
             {
                _ContadorUsuarioGlogbalConectado += Time.deltaTime;
                if (_ContadorUsuarioGlogbalConectado > 3.5f)
                {
                UsuarioGlobalConectado.text = "";

                }
                else
                {
                UsuarioGlobalConectado.text = "Conectado con: " + _Usuario.NickName;
                UsuarioGlobalConectado.color = Color.Lerp(UsuarioGlobalConectado.color,
Color.green, Time.deltaTime);
                }

             }
         }
    }
}
```

Anexo 3: *Script* GiroCabeza.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
/*
 *El siguiente script sincroniza la rotación de la cabeza a través del servidor.
 * Para ello se almacena el valor de la rotación original en un vector y se envía la
información al resto de usuarios conectados en la aplicación.
 */
public class GiroCabeza : Photon.MonoBehaviour {
    private Quaternion _RotacionCabeza;
    private GameObject _GafasVR;
    public PhotonView _Nombre;
    public PhotonView _Material;

    //Utilizada en la inicialización.
    private void Start()
    {
        if (photonView.isMine)
        {
            _GafasVR = GameObject.Find("Gafas");
            //Si el jugador es local que la cabeza sea de color azul
            this.GetComponent<MeshRenderer>().material.color = Color.blue;
            _Nombre.RPC("ActualizarNombre", PhotonTargets.AllBuffered,
PhotonNetwork.playerName);
            _Material.RPC("SincronizarColor", PhotonTargets.AllBuffered);
        }
        else
        {
            return;
        }

    }

    //Update se ejecuta por cada frame por segundo en la aplicación.
    void Update ()
    {
        if (photonView.isMine)
        {
            return;
        }
        else
        {
            this.transform.localRotation = Quaternion.Lerp(this.transform.localRotation,
_RotacionCabeza, .1f);
        }
    }
    void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
    {
        if (stream.isWriting)
        {
            stream.SendNext(this.transform.rotation);

        }
        else
        {
            _RotacionCabeza = (Quaternion)stream.ReceiveNext();
        }
    }
}
```

Anexo 4: *Script* NombreUsuario.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class NombreUsuario : Photon.MonoBehaviour {
    public TextMesh _EtiquetaNombre;

    [PunRPC]
    public void ActualizarNombre(string nombre)
    {
        _EtiquetaNombre.text = nombre;
    }

}
```

Anexo 5: *Script* ColorGafas.cs

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

public class ColorGafas : Photon.MonoBehaviour
{
    public Material[] _MaterialesGafasVR;
    public Material materialactual;

    //Utilizada en la inicialización.
    void Start()
    {
        this.GetComponent<MeshRenderer>().material = materialactual;

    }
    [PunRPC]
    public void SincronizarColor()
    {
        this.materialactual = _MaterialesGafasVR[Random.Range(0,
_MaterialesGafasVR.Length)];
        if (photonView.isMine)
        {
            return;
        }
        else
        {

            this.GetComponent<MeshRenderer>().material = materialactual;
        }
    }
}
```

Anexo 6: *Script* ClickRadialPuntero.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public class ClickRadialPuntero : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler
{
    //Contador flotante.
    public float _Contador = 0f;
    //Barra de Progreso del puntero.
    public Transform _BarraProgreso;
    //Booleano para informar si se detecta un objeto con el puntero.
    public bool _ObjDetectado = false;
    //Tiempo de acción de la barra de progreso.
    private float _ClickAl = 3f;

    //Update se ejecuta por cada frame por segundo en la aplicación.
    void Update()
    {
        if (_ObjDetectado)
        {
            _Contador += Time.deltaTime;
            _BarraProgreso.GetComponent<Image>().fillAmount = _Contador / 3;
            if (_Contador >= _ClickAl)
            {
                ExecuteEvents.Execute(gameObject, new PointerEventData(EventSystem.current),
ExecuteEvents.pointerClickHandler);
                Debug.Log("CLICK");
                ResetearContador();
            }
        }
    }
    public void ResetearContador()
    {
        _Contador = 0f;
        _BarraProgreso.GetComponent<Image>().fillAmount = _Contador;
    }
    #region IPointerEnterHandler implementation
    public void OnPointerEnter(PointerEventData eventData)
    {
        _Contador = 0f;
        _ObjDetectado = true;
        _BarraProgreso.GetComponent<Image>().fillAmount = _Contador;
    }
    #endregion
    #region IPointerExitHandler implementation
    public void OnPointerExit(PointerEventData eventData)
    {
        _ObjDetectado = false;
        ResetearContador();

    }
    #endregion
}
```

Anexo 7: *Script* MostrarTexto.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class MostrarTexto : MonoBehaviour {
    public Text _TextoAMostrar;
    public string _ContenidoTexto;
    public float _Contador;
    public bool _MostrarInfo;

       //Utilizada en la inicialización.
       void Start () {
         _TextoAMostrar.color = Color.clear;
      }

       //Update se ejecuta por cada frame por segundo en la aplicación.
       void Update () {

        EfectoVisualizacion();
       }
    public void PointerEnter()
    {
        _MostrarInfo = true;
    }
    public void PointerExit()
    {
        _MostrarInfo = false;
    }
    void EfectoVisualizacion()
    {
        if (_MostrarInfo)
        {
            _TextoAMostrar.text = _ContenidoTexto;
            _TextoAMostrar.color = Color.Lerp(_TextoAMostrar.color, Color.white, _Contador *
Time.deltaTime);

        }
        else
        {
            _TextoAMostrar.color = Color.Lerp(_TextoAMostrar.color, Color.clear, _Contador *
Time.deltaTime);
        }
    }
}
```

Anexo 8: *Script* Teletranspote.cs

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

public class Teletransporte : MonoBehaviour {
    public Camera _Camara;
    public GameObject _Usuario;
    public Animation _ATeletransporte;
    private float _Contador;
    public bool _UsuarioEncontrado = false;

    //Update se ejecuta por cada frame por segundo en la aplicación.
    private void Update()
    {
        if (!_UsuarioEncontrado)
        {

            _Usuario = GameObject.Find("Usuario Local");

            if (_Usuario != null)
            {
                _UsuarioEncontrado = true;
            }
        }
    }
    public void DispararRaycast()
    {
        Ray _raycast = _Camara.ViewportPointToRay(new Vector3(0.5F, 0.5F, 0));
        RaycastHit _objetivo;
        if (Physics.Raycast(_raycast, out _objetivo))
        {
            if (_objetivo.transform != null)
            {

                Vector3 newLocation = new Vector3(_objetivo.point.x, 2.7f,
_objetivo.point.z);
                Debug.Log(newLocation);

                _Usuario.transform.localPosition = newLocation;

                _ATeletransporte.Play();

            }
        }
    }
}
```

Anexo 9: *Script* ColorObjeto.cs

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

public class ColorObjeto : Photon.MonoBehaviour
{
    public Color _Color1;
    public Color _Color2;
    public Color _Color3;
    public Vector3 _InfoColor;
    public Color _ColorSinc;

    void Update()
    {
        if (!photonView.isMine)
        {
            this.GetComponent<MeshRenderer>().material.color = _ColorSinc;
            return;
        }
    }

    public void CambiarColor(int i)
    {
        switch (i)
        {
            case 1:
                this.GetComponent<MeshRenderer>().material.color = _Color1;
                _ColorSinc = _Color1;
                break;
            case 2:
                this.GetComponent<MeshRenderer>().material.color = _Color2;
                _ColorSinc = _Color2;
                break;
            case 3:
                this.GetComponent<MeshRenderer>().material.color = _Color3;
                _ColorSinc = _Color3;
                break;
        }
    }
    public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
    {
        if (stream.isWriting)
        {
            //Envía la información del color.
            _InfoColor = new Vector3(this.GetComponent<MeshRenderer>().material.color.r,
this.GetComponent<MeshRenderer>().material.color.g,
this.GetComponent<MeshRenderer>().material.color.b);

            stream.Serialize(ref _InfoColor);
        }
        else
        {
            //Obtiene la información del color.
            stream.Serialize(ref _InfoColor);

            _ColorSinc = new Color(_InfoColor.x, _InfoColor.y, _InfoColor.z, 1.0f);

        }
    }
}
```

Anexo 10: *Script* RegistrarEntradaUsuario.php

```php
<?php
$servidor = "127.0.0.1";
$usuario = "root";
$contraseña = "";
$nombreBD = "aplicacion_tfg";

//Variables enviadas por el usuario.
$UsuarioRegistrado = $_POST['UsuarioRegistrado'];
$date = date("Y-m-d H:i:s");
//Creación de la conexión al servidor.
$conn = new mysqli($servidor, $usuario, $contraseña, $nombreBD);

//Comprobación de la conexión.
if ($conn->connect_error) {
    die("Conexion fallida: " . $conn->connect_error);
}
//Inserta el nombre de usuario y la fecha de entrada en la base de datos.
$sql = "INSERT INTO usuarios (nombre_usuario, fecha_entrada) VALUES ('" . $UsuarioRegistrado .
"', '". $date ."')";
 echo "Creando Usuario...";

if ($conn->query($sql) === TRUE) {
       echo "Usuario registrado en la BBBDD con exito: " .date("Y-m-d") ." a las "
.date("H:i:s");
} else {
       echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();
?>
```

Anexo 11: *Script* RegistrarSalidaUsuario.php

```php
<?php
$servidor = "127.0.0.1";
$usuario = "root";
$contraseña = "";
$nombreBD = "aplicacion_tfg";

//Variables enviadas por el usuario.
$UsuarioRegistrado = $_POST['UsuarioRegistrado'];
$date = date("Y-m-d H:i:s");
// Creación de la conexión al servidor.
$conn = new mysqli($servidor, $usuario, $contraseña, $nombreBD);

//Comprobación de la conexión.
if ($conn->connect_error) {
    die("Conexion fallida: " . $conn->connect_error);
}
//Actualiza la fecha de salida.
$sql = " UPDATE usuarios SET fecha_salida = ('". $date ."')";

if ($conn->query($sql) === TRUE) {
    echo "Fecha de salida guardada correctamente.";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();
?>
```