TRABAJO FINAL DE GRADO

APLICACIÓN DE REALIDAD MIXTA PARA CONFERENCIAS ONLINE, VISUALIZACIÓN E INTERACCIÓN CONJUNTA DE MAPAS, Y GESTIÓN DE RECURSOS.

Autor: Samuel Trujillo Santana
Tutor: José Daniel Hernández Sosa
Cotutor: Agustín Trujillo Pino
Cotutor de empresa: Antonio José Sánchez López

ÍNDICE

0. GLOSARIO

- **Buildear o generar**. Término que ser refiere a construir una aplicación. Básicamente juntar todos los scripts y objetos que se usan en la escena y generar un ejecutable, el cual se puede pasar a las gafas para usarlo.
- **Boolean**. Tipo de variable que representa dos valores, o verdadero o falso.
- **Callback**. Tipo de función que se llama automáticamente cuando otra función termina.
- **Canvas.** Panel de dos dimensiones que contiene los elementos tipo interfaz: textos, imágenes, campos de texto bidimensionales, menus, etc.
- **Collider**. una clase a la que le das unos límites, y envía una señal si otro objeto colisiona con esos límites.
- **Double**. Como el Float, solo que guarda número con muchos decimales.
- **Float**. Tipo de variable que sirve para guardar un número con decimales.
- **IputField**. Elemento de interfaz de Unity que consiste en un campo de texto pensado para que el usuario escriba en él.
- **Json**. Es un formato de texto que sirve para guardar datos. Funciona similar al HTML
- **NearinteractableTouchable**. Clase del paquete de MRTK que permite que un objeto pueda interactuar con el usuario al intentar tocarlo con las manos.
- **Prefab**. Objeto prefabricado. Un objeto de Unity que dentro puede tener clases componentes modelos. Se guardan en disco como prefab para poder reutilizarlos de una forma más fácil
- **String**. Tipo de variable que sirve para guardar una lista de caracteres, una palabra, frase o párrafo entero.
- **Quaternion**. Variable de Unity que sirve para representar una rotación tridimensional.

1. INTRODUCCIÓN

Desde siempre las personas han dependido de imágenes, dibujos, representaciones visuales en general, para poder recibir información. Empezaron plasmando sus ideas en papel, creando cuadros, mapas, luego pasaron a las fotografías y vídeos, para conservar momentos concretos, y la realidad mixta será el próximo paso. Actualmente lo común es poder ver imágenes, vídeos e información en una pantalla plana, y diversas formas limitadas de interactuar con ellas, ya sea un teclado, mandos remotos, etc. Pero con la realidad mixta se podrá ver información más detallada, inmersiva y realista, ya sea transmitiendo a tu alrededor para hacer creer que se está en otro sitio, como solapando información al mundo que te rodea, ampliando los datos que se reciben, y mejorando la forma de interactuar con ella, usando las mismas manos.

La realidad mixta es la combinación de realidad virtual y realidad aumentada. La realidad virtual crea entornos virtuales en los cuales el usuario, mediante unas gafas especiales, puede sentirse inmerso en estos entornos, como si estuviera físicamente en ellos, y la realidad aumentada, es la otra cara de la moneda, es la posibilidad de plasmar objetos virtuales en un entorno físico, los cuales se pueden ver mediante teléfonos móviles, entre otros dispositivos, para simular que están ahí. Pues bien, la realidad mixta combina las dos, permitiendo crear nuevos espacios en los que interactúan tanto objetos y/o personas reales como virtuales.

La aplicación que se desarrolla en el proyecto utiliza esta tecnología para aportar funcionalidades innovadoras y proponer nuevas formas de utilizar una aplicación informática.

La aplicación consta, básicamente, de una aplicación de realidad mixta, que, en el entorno físico donde el usuario se encuentra, muestra un mapa donde puede instanciar y manipular recursos, y ver dichos recursos en multiusuario en tiempo real con otras personas en otras partes del mundo.

El trabajo ha sido desarrollado en conjunto con el trabajo titulado “Kit de herramientas para el tratado de mapas en conferencias de Realidad Mixta” del compañero Alberto Mejías Márquez. A lo largo de esta memoria se explica la distribución de las partes realizadas por cada uno, en mi caso, la integración en el dispositivo de realidad mixta.

1.1. Motivación y Objetivos del proyecto

El proyecto surge a partir de la realización de unas prácticas curriculares en la empresa **The Singular Factory SL**, centradas en el trabajo con Unity como motor de creación de videojuegos, algo en lo que siempre estuve personalmente interesado. La empresa quería retomar el campo de la realidad mixta así que decidieron rescatar un antiguo proyecto suyo que utilizaba el software de mapas, y permitir generar mapas tridimensionales holográficos para poder trabajar con ellos en remoto y de una manera innovadora. Para trabajar en ello me dediqué a investigar sobre Unity, sobre cómo desarrollar aplicaciones y videojuegos y también empecé a investigar sobre la realidad virtual, aumentada y mixta.

El desarrollo de este proyecto sirvió para investigar y aprender el tema de la programación de videojuegos y la realidad virtual y, como producto final, la creación de una aplicación funcional que sirva para realidad mixta y que permita la manipulación de mapas y reuniones remotas.

En lo que respecta a la distribución de tareas con el trabajo del compañero, el presente trabajo se ha centrado en la investigación de la realidad mixta y su programación, mientras que el otro proyecto coordinado está enfocado hacia el ámbito multiusuario.

1.2. Competencias académicas

Este TFG se realizó teniendo en mente cumplir con las siguientes competencias:

- **CII016: Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería de software.** Se han utilizado diversos recursos y técnicas de ingeniería del software, así como algunas metodologías vistas en clase. Concretamente podemos hablar de la metodología scrum, que es la que he se ha utilizado junto al compañero de proyecto para favorecer la coordinación en su desarrollo.

- **CII017: Capacidad para diseñar y evaluar interfaces persona computador que garanticen la accesibilidad y usabilidad a los sistemas, servicios y aplicaciones informáticas.** En este caso se ha trabajado en la interacción persona dispositivo, ya que se he trabajado en la realización de una interfaz para el dispositivo de realidad mixta, una interfaz en la que el usuario interactúa directamente con las manos sobre la aplicación.

- **CP03: Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.** En esta aplicación se ha tenido que realizar trabajo de investigación que en algunas ocasiones no ha dado resultados y se ha visto la necesidad de generar maneras propias de resolver algún asunto. En otras ocasiones incluso se ha llegado a desarrollar código sin la búsqueda de información, solo desarrollando la metodología por cuenta propia (el caso del RoomState, que se hablará más adelante, fue idea propia).

- **CP06: Capacidad para desarrollar y evaluar sistemas interactivos y de presentación de información compleja y su aplicación a la resolución de problemas de diseño de interacción persona computadora.** El hecho de que la aplicación se desarrolle para realidad mixta supone una nueva forma de interacción persona computadora, por lo que insta alejarse del común teclado y ratón para optar por un enfoque distinto, ampliando las posibilidades de interacción: coger una cosa literalmente con la mano en vez de clicar y arrastrar con el ratón, pulsar botones imaginarios en el aire...

1.3. Concepto de Realidad Mixta

Una parte importante de este trabajo era investigar sobre la realidad mixta y el dispositivo prestado por la empresa que utiliza esta tecnología, las HoloLens 2 de Microsoft.

Todo el mundo se ha visto muchas escenas y películas de ciencia ficción donde se ven hologramas, ya sean pantallas en el aire donde se puede trabajar, como en Iron Man [Ilustración 1].

Ilustración 1 Escena de la película Iron Man. Fuente:Google

U hologramas por la ciudad a modo de carteles y anuncios [Ilustración 2]

Ilustración 2 Escena de la película Ghost in the Shell. Fuente: Google

Todo esto, la tecnología de hologramas y las pantallas flotantes en el aire son un concepto que se basa en añadir objetos inexistentes a un escenario real, para poder percibir mayor cantidad de información, bien este concepto se llama realidad mixta.

Como se ha explicado anteriormente, la realidad mixta es la combinación de la realidad virtual y la realidad aumentada.

1.3.1 Realidad Virtual

A la realidad virtual se le puede llamar un conjunto de técnicas informáticas, que permiten, gracias a dispositivos especiales, como gafas [Ilustración 3], guantes o trajes, generar imágenes y sensaciones que hagan que el usuario sienta que se encuentra en una realidad diseñada por ordenador. Permite ver cosas que no están ahí, o ser capaz de verse a uno mismo en algún sitio lejano, ya sea real o ficticio, e interactuar con ese entorno u objetos que estén en el mismo [Ilustración 4]

Ilustración 3 Dispositivo de realidad virtual. Fuente: Google

Ilustración 4 Ilustración de imagen virtual. Fuente: Google

A continuación, se mencionan algunos campos donde la realidad virtual se ha estado utilizando:

- **Educación.** Como es una tecnología que permite al usuario sentirse que está en otro lugar, se pueden crear aplicaciones con fines didácticos, generando entornos virtuales con objetos que representen objetos reales, para poder aprender e interactuar con ellos, como recrear una cirugía, simulador de vuelo para los pilotos [Ilustración 5], o mostrar distintas localizaciones en una clase de geografía.

Ilustración 5 Simulador de vuelo. Fuente: Google

- **Psicoterapia.** En este ámbito se utiliza a veces para tratar fobias, simulando situaciones que afecten a los pacientes, como simular que están en un sitio muy alto, o encontrase rodeado de algo que les produzca terror (como la fobia a las arañas o a las serpientes).

- **Medicina.** Principalmente se utiliza para el caso mencionado en el apartado de enseñanza, un simulador de cirugía donde los estudiantes pueden practicar con pacientes virtuales, inexistentes [Ilustración 6].

Ilustración 6 Simulador de cirugía "Surgeon Simulator". Fuente: Google

- **Entretenimiento.** Gracias a las gafas y los dispositivos de realidad virtual podemos mostrar vídeos e imágenes en 360 grados, permitiendo, por ejemplo, mostrar una película en la cual el usuario la vea toda a su alrededor, para añadirle una mayor sensación de inmersión.

- **Videojuegos.** El ámbito de los videojuegos es el que más ha avanzado con respecto a la realidad virtual. Actualmente existen videojuegos inmersivos de casi cualquier tipo: simuladores de carreras, simuladores de disparos, juegos de puzles, peleas de espadas... [Ilustración 7]

Ilustración 7 Videojuego de realidad virtual "Beat Saber". Fuente: Google

- **Software.** Algunos softwares de modelado 3D han creado sus versiones de realidad virtual para que el usuario pueda verse en un lienzo o universo donde pueda trabajar con objetos y verlos en tres dimensiones como se verían en la realidad [Ilustración 8]

Ilustración 8 Aplicación de modelado y dibujo en 3D "MasterpieceVR". Fuente: Google

1.3.2. Realidad Aumentada

La realidad aumentada también es un conjunto de técnicas informáticas, pero esta vez se centran en hacer lo opuesto a la realidad virtual, traer datos virtuales a la vida real.

Gracias al sentido de la vista, a lo que vemos por los ojos, las personas pueden recibir información del entorno, ver un sitio, una persona, un dibujo, un texto, un peligro, gran cantidad de información que les permite ser consientes y comportarse de una manera. Pues bien, la realidad aumentada, añade objetos virtuales que realmente no están en un sitio para que la vista capte aún más información de la que existe realmente. Por ejemplo, cuando una persona anda por la calle, es capaz ver la calle por donde transita, los edificios alrededor, otras personas que se cruzan, pues con la realidad aumentada se puede ver lo mismo y además añadir más información, como indicar que el edificio a la izquierda es una pizzería, hay un cajero al girar

en el siguiente cruce, el edificio al final de la calle es un hotel... y así añadir más información a lo que se ve [Ilustración 9].

Ilustración 9 Ejemplo de cómo mostraría información una aplicación de realidad aumentada. Fuente: Google

El termino de realidad aumentada se debe a que, superponiendo esos objetos virtuales a la realidad, se puede "aumentar" la información de la realidad que se percibe.

A continuación, se mencionan algunos campos donde la realidad virtual se ha estado utilizando:

- **Educación.** Como se menciona al principio del apartado de la realidad aumentada, el concepto principalmente es añadir más información a lo que se ve. Con este concepto en mente se han creado aplicaciones didácticas que, por ejemplo, permitan añadir información a textos escritos, a imágenes u objetos expuestos en un museo, añadir contenido audiovisual a libros de texto, y más.

- **Televisión.** La realidad aumentada se está empezando a usar mucho en los programas de deporte, aportando información adicional, como mostrar las estrategias que se están siguiendo en el campo, aportar datos sobre los jugadores, el equipo o el encuentro en general, resaltar cosas importantes como la línea de fuera de juego en el fútbol [Ilustración 10], o la línea del Down en el fútbol americano. También se usa en los noticiarios para hacer recreaciones de situaciones reales, como inundaciones o derrumbes.

Ilustración 10 Realidad aumentada en un partido de fútbol. Fuente: Google

- **Información y Turismo.** Un gran ejemplo en este ámbito es que Google permite que un usuario apunte con la cámara de su smartphone a un cartel y que en la pantalla se muestre el cartel traducido, lo que hace más fácil a los turistas que no entiendan el idioma escrito del lugar a donde viajen [Ilustración 11].

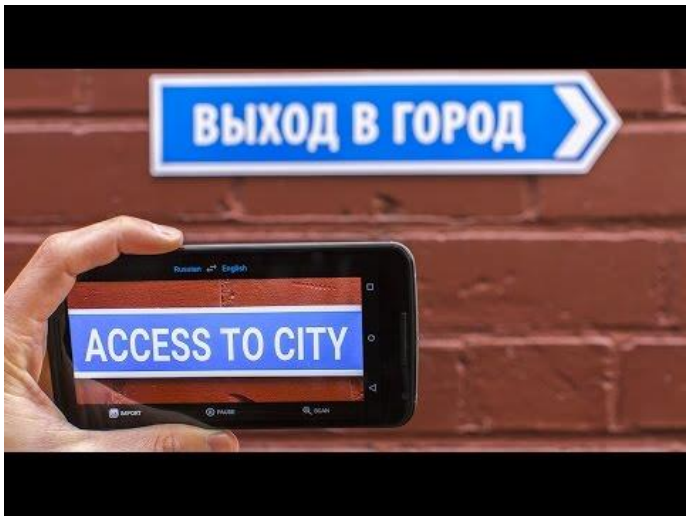

Ilustración 11 Aplicación de realidad aumentada para traducir carteles. Fuente: Google

1.3.3. Realidad Mixta

La realidad mixta es la mezcla de la realidad virtual y la realidad aumentada. Un conjunto de técnicas informáticas, que permite transportar objetos virtuales a la realidad del usuario, interactuar con esos objetos, y además el aporte de que los objetos virtuales interactúen con la realidad. Esto último, hace que la experiencia de usarla sea más realista que la realidad aumentada. Mientras que la realidad aumentada superpone el objeto virtual a la imagen que está viendo una cámara, la realidad mixta es consciente del escenario y permite que un objeto, si se sitúa en una mesa, sea consciente de que está en una mesa, y, por tanto, si el usuario intenta

verlo desde debajo de la mesa no lo hará, porque la mesa se interpone [Ilustración 12 e Ilustración 13]. Además de eso los dispositivos que trabajan con la realidad mixta pueden detectar las manos del usuario y simular que interactúa con el objeto, como agarrarlo y moverlo, o golpearlo y hacer que se desplace, utilizando el entorno real como escenario base [Ilustración 14].

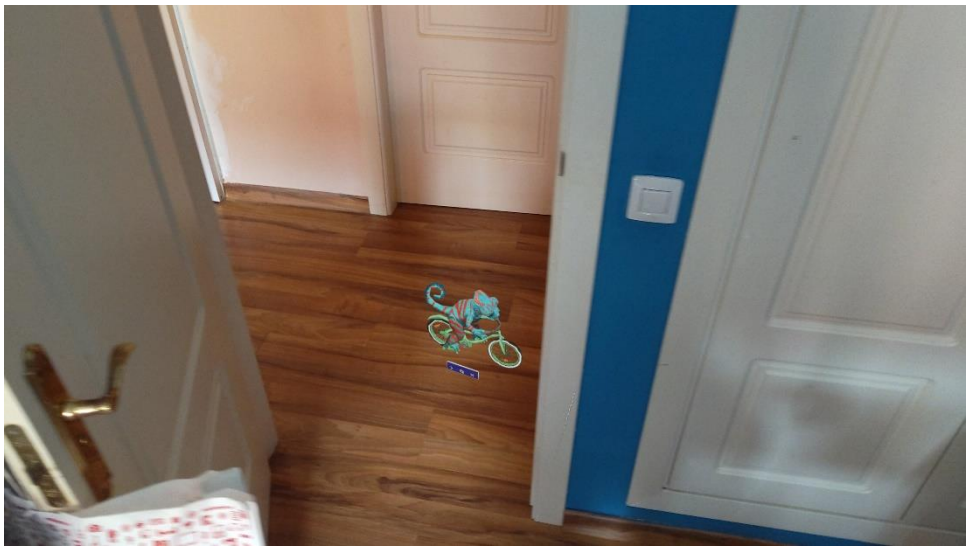

Ilustración 12 En esta imagen se muestra el holograma de un camaleón en bicicleta en el suelo. Fuente: Imagen capturada por las gafas cedidas por la empresa

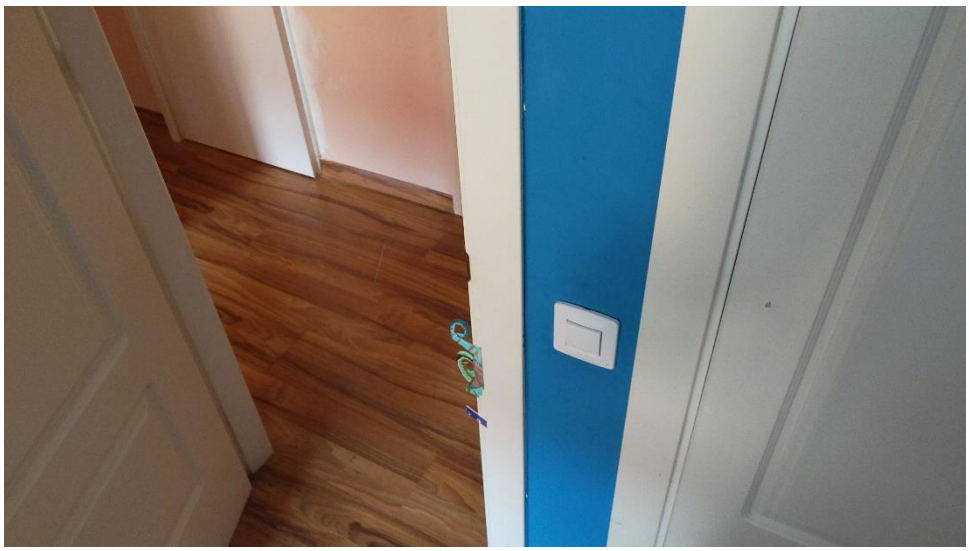

Ilustración 13 En esta imagen se muestra que como el camaleón está en el suelo detrás de la puerta, al mirarlo desde otro ángulo, como se encuentra detrás de la pared, se oculta parcialmente. Fuente: Imagen capturada con las gafas cedidas por la empresa

Ilustración 14 Videojuego de realidad mixta. Fuente: Google

La realidad mixta cada vez tiene más aplicaciones para mejorar experiencias en diversos trabajos o situaciones:

- **Ejército**. En el ejército se está utilizando como entrenamiento militar, y se está investigando su uso en el terreno para que los soldados puedan tener visores con información que reacciona al entorno, como telemetría, vista de un mapa, etc.

- **Trabajo remoto.** permite que usuarios de distintas partes del mundo se conecten en reuniones virtuales para poder llevar a cabo reuniones a distancia, de una forma más dinámica y cercana que por videollamada.

- **Diseño industrial o arquitectura**. Poder visualizar modelos de viviendas o diseños de prototipos a escala real y ver su impacto en la vida real.

- **Publicidad y comercio**. Poder superponer datos virtuales al mundo real permite que la publicidad de marcas de ropa, o de muebles y diseños de interiores, pueda simularse sobre un entorno o persona real, permitiendo a los clientes una mejor manera de ver los productos, que en una imagen plana.

- **Videojuegos**. Esta tecnología permite crear videojuegos que interactúen con el entorno del usuario, creando experiencias nuevas, como que los enemigos se escondan detrás de tus muebles, o el mapa se adapte al entorno de tu salón.

2. RECURSOS UTILIZADOS

En este capítulo se mencionan todos los recursos utilizados para desarrollar el proyecto, tanto a nivel de hardware como de software.

2.1. HoloLens 2

El trabajo se ha desarrollado con el dispositivo **HoloLens 2** de Microsoft [Ilustración 15], prestado por la empresa colaboradora.

Ilustración 15 Gafas HoloLens 2. Fuente: Google

Este dispositivo son unas gafas especiales que trabajan con realidad mixta. Poseen unas pantallas individuales para cada ojo, como si fueran los cristales de unas gafas normales, las cuales son transparentes y proyectan dos imágenes separadas. Esas imágenes son ligeramente distintas entre sí, son representaciones de la misma escena vista desde dos puntos distintos, en los lugares donde estarían los ojos de una persona, y al mostrárselas al ojo humano real, el cerebro del usuario utiliza las diferencias entre las imágenes para obtener una sensación de profundidad, como funcionan en la vida real [Ilustración 16].

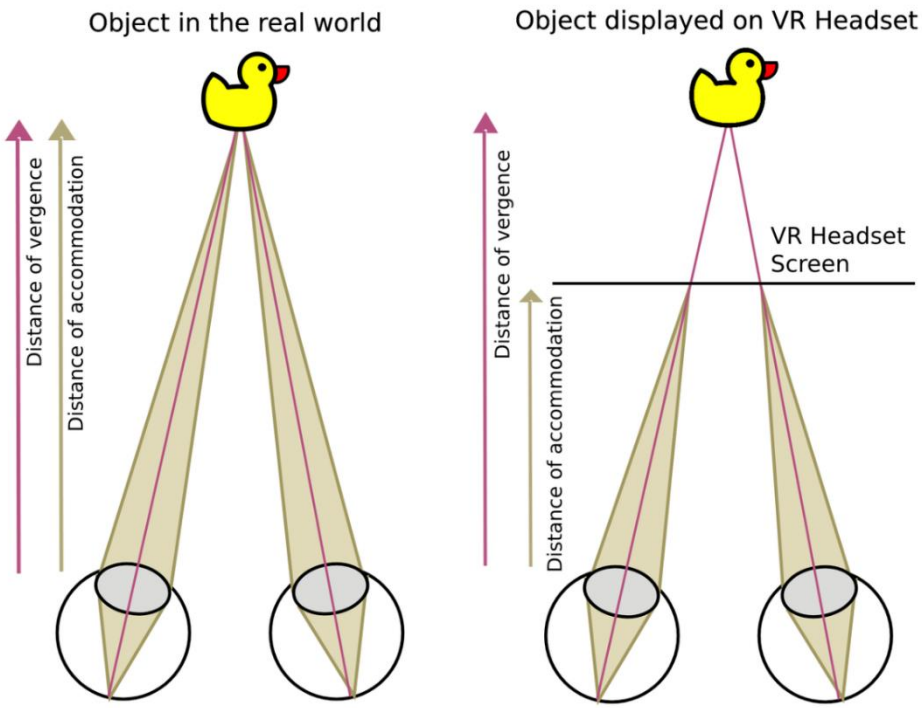

Ilustración 16 Modelo de cómo funciona un dispositivo VR para mostrar imágenes con profundidad.
Fuente: https://en.wikipedia.org/wiki/Vergence-accommodation_conflict

Las gafas poseen en su interior un acelerómetro y un giroscopio, por lo que son capaces de detectar la orientación que tienen con respecto al suelo. Poseen cuatro cámaras para poder detectar profundidad como hace el ojo humano o reconocer formas, como el caso de las manos, para saber los movimientos que hace el usuario.

2.2. Unity

Unity es un motor de videojuego multiplataforma muy utilizado en la actualidad por muchas empresas, y es el motor que se utiliza para crear la aplicación. La razón por la que se ha escogido este motor principalmente es porque la empresa colaboradora hace uso a diario del mismo, así que se podía contar con asesoramiento y ayuda a la hora de aprender a utilizar esta herramienta. Además, es compatible con un gran número de librerías y plugin que ya estén escritos en C# (es el lenguaje en el que se programa en Unity), o en C++. Entre esos plugin está el de MRTK o MixedRealityToolkit, que permite crear aplicaciones para las gafas de realidad mixta.

Unity es uno de los motores de videojuego multiplataforma más populares debido a su gran versatilidad a la hora de integrar módulos y trabajar en una gran cantidad de plataformas diferentes, y a una gran usabilidad, tal así que no solo se usa en el ámbito de los videojuegos, sino también en el ámbito de la arquitectura, la industria automotriz, y el cine, entre otros. Además, posee una facilidad de aprendizaje la cual permite que cualquier persona pueda descargárselo y generar su propio videojuego después de seguir algunos tutoriales.

Debido a todas esas buenas cualidades se eligió el motor de Unity para realizar este proyecto.

2.3. Mixed Reality Toolkit

Unity tiene acceso a **Mixed Reality Toolkit**, un conjunto de herramientas y procesos desarrollados por Microsoft que permiten trabajar y crear proyectos de Realidad Mixta en Unity.

Siguiendo un conjunto de tutoriales creados por Microsoft, un usuario puede crear y preparar una escena en un proyecto de Unity que pueda usarse en un dispositivo de Realidad Mixta, como es el caso de las HoloLens 2. Esta preparación incluye:

- La integración e interacción del entorno físico real con la escena, permitiendo que los objetos reaccionen con el entorno de la manera que se desee, como apoyarse en el suelo o mesas del entorno, reaccionar al color, detectar la profundidad... [Ilustración 17].

Ilustración 17 El kit de MRTK. Fuente: imagen de la web de Mixed Reality Toolkit

- La posibilidad de reconocer las manos y permitir interactuar con objetos virtuales [Ilustración 18].

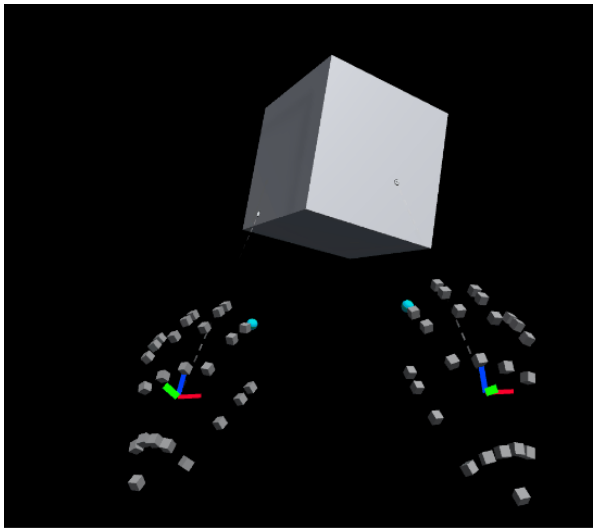

Ilustración 18 El usuario puede interactuar con objetos virtuales. Fuente: imagen de la web de Mixed Reality Toolkit

- El reconocimiento de posibles gestos de la mano para poder ejecutar comandos (como mirarse la palma de la mano para abrir el menú, o simular el pulsar un botón, entre otras cosas) [Ilustración 19].

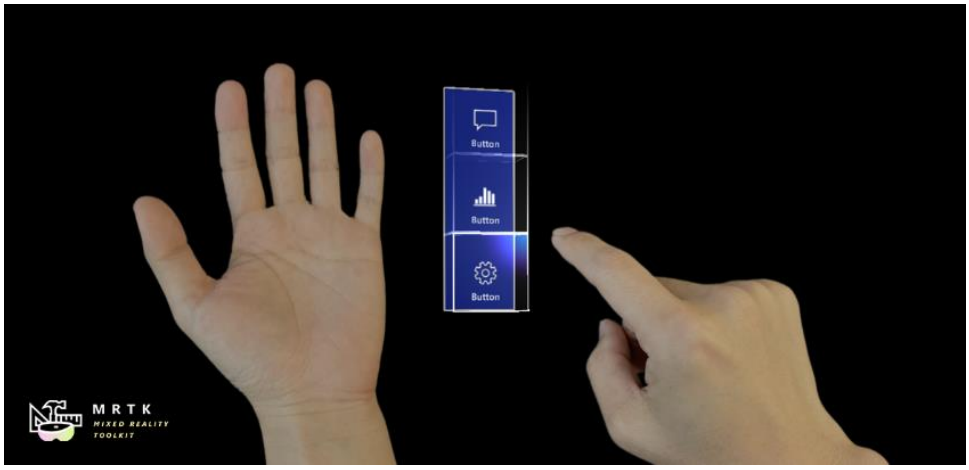

Ilustración 19 El usuario puede hacer gestos como pulsar un botón con el dedo. Fuente: imagen de la web de Mixed Reality Toolkit

Además, se ha comprobado durante el dearrollo que el MRTK no es solo exclusivo para las gafas de Realidad Mixta propias de Microsoft, sino que también permite operar con otras gafas de Realidad Virtual o Mixta, como las Quest 2 de Meta.

2.4. PUN (PhotonEngine)

Photon (PUN) [Ilustración 20] es un framework que sirve para alojar servidores multijugador para juegos y aplicaciones. Incluye clases y métodos que permiten conectar una aplicación a un servidor multijugador, el cual puede ser gratuito si el número de usuarios es bajo.

Ilustración 20 Logo Photon. Fuente: Google

2.5. OnlineMaps

OnlineMaps es un software multiplataforma diseñado para el desarrollo de mapas. Posee una solución instalable en Unity que permite su uso para crear aplicaciones y videojuegos que utilicen ya sea 2D, 3D, como Realidad Virtual o Realidad Aumentada.

La solución permite una gran personalización, poder adaptarse a muchas formas tamaños y maneras de expresar la información [Ilustración 21]

Ilustración 21 Aplicación de OnlineMaps. Fuente: Google

2.6. Visual Studio

Visual Studio Es el entorno de desarrollo que viene por defecto con Unity [Ilustración 22]. Es cómodo y completo, permite acceso a gran cantidad de librerías y poder depurar en conjunto con la ejecución de la aplicación en Unity.

Ilustración 22 Logo VisualStudio. Fuente: Google

2.7. Trello

Trello es un software de administración de proyectos el cual se utiliza para poder gestionar las tareas y el tiempo de desarrollo del trabajo [Ilustración 23]. Es una herramienta útil que permite una gestión y visualización de las tareas en formato de tarjetas y tableros cómos de usar.

Ilustración 23 Logo de Trello

La disposición inicial constará una columna con tareas a hacer, otra con trabajos en proceso, una columna donde depositar las tareas pendientes de testeo, y una columna para las tareas ya terminadas, pero a lo largo del trabajo cabe la posibilidad de añadir más.

2.8. Git

Git:

es el software que se utiliza para el control de versiones [Ilustración 24]. Además del proyecto se aloja en un repositorio remoto en una localización propia de la empresa que funciona a modo de GitHub, donde se suben los cambios y se coordina el proyecto de forma cooperativa.

Ilustración 24 Logo Git. Fuente: Google

Gracias al repositorio remoto cedido por la empresa es posible desarrollar cómodamente en paralelo, llevar un control de versiones y guardar el estado del proyecto para poder editarlo desde cualquier parte en todo momento. Además, posee la ventaja que permite subir archivos grandes (en GitHub el límite está en que los archivos no sobrepasen los 100M, mientras que, en el repositorio de la empresa, nadie ha llegado al límite).

3. ANÁLISIS

3.1. Concepto

La aplicación final será una aplicación visible en las gafas HoloLens, unas gafas de realidad mixta, que permitirán desplegar un mapa virtual en cualquier superficie real o incluso suspendido en el aire mismo, con el que se podrá interactuar para navegar. Sumado al mapa, se podrá disponer de diversos recursos, ya sean modelos 3d de edificios, vehículos o personas, hasta vídeos o fotos, que puedan situarse en el mapa para simular objetos reales que pueden estar situados o no en la vida real en la localización que se muestra en el mapa. Toda esta experiencia contiene también un sistema de reuniones para poder visualizar e interactuar con el mismo contenido en el mismo tiempo desde distintos sitios.

El diseño de la aplicación podría decirse que está basado las salas de control que se muestran en muchas películas de acción y de ciencia ficción: Una mesa (u holograma) central a cuyo alrededor se reúnen varias personas para discutir o elaborar planes, situaciones y estrategias sobre un área de terreno. Esta mesa central posee un mapa del lugar en cuestión y también accesorios, objetos u dispositivos que permiten aportar un mayor grado de información y manipulación.

Un claro ejemplo de este concepto es el que se muestra en la [Ilustración 25]. A esto se le conoce como “caja de arena”: Consiste en imitar el nivel del terreno correspondiente a una operación con arena encima de una mesa. En este terreno se disponen figuras simulando edificaciones, vehículos, tropas aliadas o enemigas, y en ella un oficial puede explicar al resto de personas presentes cómo se desarrollará la operación, moviendo las figuras simulando desplazamientos reales.

Ilustración 25 Cajón de arena militar para mostrar información del terreno y planear estrategias. Fuente: Google

Este concepto se puede ver mucho en películas de acción y de ciencia ficción, donde se suele aprovechar la situación ficticia para llevar el tema de mapas a un nuevo nivel, con hologramas o entornos virtuales que permitan ver esta información de manera mucho más precisa y detallada. Un ejemplo puede ser una imagen sacada de la película de ciencia ficción “Avatar” [Ilustración 26], donde se muestra una escena en la que varias personas se reúnen en torno a una representación holográfica del terreno, donde se muestra varias capas de información,

como la topografía, la huella de calor, y probablemente muchos más datos, todo ello visible en el aire.

Ilustración 26 Fotograma de la película de Avatar, en el cual se muestra un mapa holográfico del terreno. Fuente: Google

Bien, pues este trabajo de fin de carrera consiste en acercar este concepto de mapa holográfico que se muestra en las películas, una forma de poder operar sobre un terreno en tres dimensiones con objetos y funciones virtuales en un entorno inmersivo, permitiendo además la opción de conferencia remota para poder ver exactamente lo mismo al mismo tiempo en distintas partes del mundo.

3.2. Uso de la aplicación en la vida real

Una vez dejado claro el concepto se puede hablar de los posibles usuarios y usos de la aplicación:

- Como se menciona en el apartado de concepto, el ejército puede utilizar esta aplicación como una caja de arena. Varios usuarios se conectarían desde distintos sitios a una misma sala donde pueden planear una operación. Podrían disponer el mapa de una región en la que se vaya a realizar la operación, crear objetos 3d que representen soldados o vehículos y situarlos donde se desee, dibujar sobre el mapa las órdenes de movimientos, calcular distancias y planear bien una operación. Además de eso estas gafas las puede llevar tanto un alto cargo desde un centro de operaciones como un soldado de campo en el terreno de la operación.

- Dejando de lado el tema bélico, también se pueden coordinar operaciones de emergencia, como planes de actuación ante incidentes como incendios, accidentes..., utilizando el mismo procedimiento que en el ámbito militar. Se puede dibujar el perímetro del fuego y organizar la disposición de los vehículos de emergencias, como los bomberos o delimitar una zona para las ambulancias.

- También es de utilidad para las empresas de transporte, para poder situar sus vehículos, almacenes, fábricas de producción... En las reuniones que hacen internamente para ver la situación o pensar nuevas rutas o distribuciones, se situará un mapa en una mesa y se representarán sus activos (almacenes, vehículos...) sobre este.

- Además de las empresas de transporte, también cualquier empresa que tenga activos que deban estar situados en puntos estratégicos, como cadenas de centros comerciales, empresas eléctricas, ferroviarias, gasolineras... utilizarán un mapa para distribuir esos activos, esta aplicación ofrece una forma de poder organizar esto de una manera más completa.

3.3. Casos de uso

A continuación, se muestra un diagrama de casos de uso donde explica las acciones que puede realizar un usuario con esta aplicación [Ilustración 27].

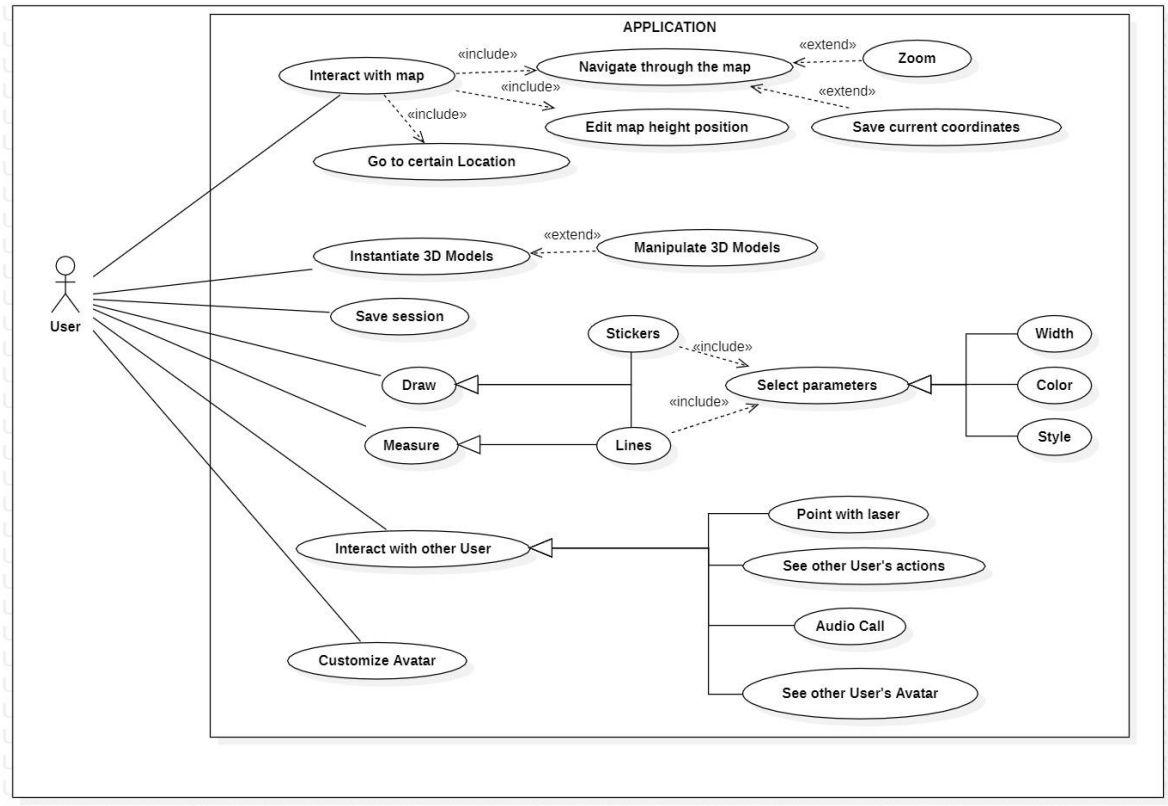

Ilustración 27 Diagrama de casos de uso de la aplicación. Fuente: diagrama desarrollado en conjunto con el compañero

- **Interactuar con el mapa**. El usuario podrá **navegar** por el mapa. Será capaz de desplazar el mapa por los cuatro ejes cardinales, e incluso dispondrá de la opción de **hacer zoom** sobre el mapa, acercando o alejando la imagen, para poder ver un mayor o menor nivel de detalle. En vez de desplazarse por una coordenada a nivel constante, el usuario puede **ir a una localización concreta** entre varias localizaciones mostradas

por la aplicación. El mapa posee información del relieve y podrá simular la elevación de las montañas y valles a medida que el usuario se acerque a ellos con el zoom del mapa. El usuario podrá **guardar** la posición actual en la que se encuentra como una de las localizaciones concretas antes mencionadas. El usuario podrá también **editar la altura a la que se muestra el mapa**, dejándolo en el suelo o subiéndolo a la altura de alguna mesa. (Este caso de uso se realiza en conjunto, en el presente proyecto consiste en integrar el mapa y el sistema de localizaciones y en el proyecto coordinado la movilidad del mapa)

- **Instanciar objetos 3D**. Se dispondrá de un catálogo de modelos 3D para poder situar sobre al mapa, desplazarlos, y hacer que mantengan su posición acorde a las coordenadas relativas del mapa, permitiendo situar un vehículo o edificio en unas coordenadas del mapa concretas y que este mantenga su posición. Estos objetos instanciados podrán ser **manipulados** en cuanto a su rotación, tamaño, posición relativa, y ser recortados, clonados o alineados. (Este caso de uso lo abarco yo en su totalidad)

- **Guardar la sesión**. El usuario puede guardar la sesión con las coordenadas actuales, los objetos y los dibujos que ha hecho o puesto durante la sesión. Para poder retomarla en otro momento. (Este caso de uso pertenece a este proyecto).

- **Dibujar**. El usuario tendrá acceso a una herramienta que le permitirá dibujar sobre el mapa y los objetos instanciados en él líneas o pegatinas, con distintos **parámetros**, entre ellos elegir **color**, **grosor** o **estilo**, y la capacidad de que esos dibujos se mantengan en las coordenadas relativas del mapa o no, según se desee. (Este caso de uso forma parte del proyecto coordinado)

- **Medir**. Entre otras herramientas, también existe la posibilidad de utilizar otra herramienta para realizar mediciones en el terreno o en el aire, con una escala relativa al mapa para poder hacerse una idea de las dimensiones del lugar, y la posible relación entre este y los objetos que se instancien. (Este caso de uso forma parte del proyecto coordinado).

- **Interactuar con otros usuarios**. La aplicación principalmente es una herramienta para llevar a cabo reuniones logísticas telepresenciales, por lo que los usuarios deben poder comunicarse e interactuar entre ellos. Para ello, la aplicación posee varias herramientas: los usuarios **tendrán avatares** para poder situarse y reconocer alguien o algo a lo que dirigirse; dispondrán de un **puntero** para poder señalar sitios o cosas del mapa; todo lo que pasa en la aplicación se **sincroniza entre todos los usuarios**, por lo que verán el mismo punto del mapa, los mismos objetos en él y los mismos dibujos. Además de los puntos anteriores, la aplicación dispone de **chat de voz**. (En el proyecto coordinado se menciona el apartado de avatares, el puntero y el chat de voz, y en este proyecto la sincronización de los usuarios)

- **Personalizar avatar**. El usuario podrá elegir distintos modelos de avatares, piezas sueltas, o distintos colores, para personalizar el suyo y hacerlo distinto del modelo base. (Este caso de uso se lleva a cabo por el proyecto coordinado)

3.4. Requisitos de usuario

REQUISITOS FUNCIONALES	
1	El usuario debe poder elegir entre dos salas diferentes.
2	El usuario debe poder ir a cualquier coordenada geográfica en el mapa.
3	El usuario debe poder crear un acceso rápido a una coordenada geográfica.
4	El usuario debe poder aumentar o disminuir el zoom del mapa.
5	El usuario debe poder ver los avatares del resto de usuarios conectados en su sala.
6	El usuario debe poder utilizar un menú para manejar toda la aplicación.
7	El usuario debe poder instanciar una serie de modelos 3D de un conjunto predefinido por el sistema.
8	El usuario debe poder activar y desactivar un puntero láser que sale de su mano.
9	El usuario debe poder dibujar líneas de diferentes colores con sus manos.
10	El usuario debe poder medir entre dos puntos que señale con sus manos.
11	El usuario debe poder colocar pegatinas con sus manos.
12	El usuario debe poder elegir si el dibujo, la medición o las pegatinas se instancian en sus dedos o a ras del mapa.
13	Los usuarios que se encuentren en la misma sala deben poder comunicarse entre ellos.
14	El usuario debe poder guardar la escena en disco.

Ilustración 28 Tabla de requisitos Funcionales. Fuente: tabla desarrollada

REQUISITOS NO FUNCIONALES	
1	El sistema debe ser intuitivo y fácil de usar para usuarios sin experiencia previa en realidad virtual.
2	El sistema debe tener un rendimiento fluido y una respuesta rápida.
3	El sistema debe poder ser escalable. Que permita añadir mejoras con las posteriores versiones.
4	El sistema debe mantener la sincronización entre usuarios en todo momento.
5	El sistema guardará las teselas del mapa en caché.
6	El sistema debe comunicarse con el servidor multiusuario en un tiempo de respuesta menor a 2 segundos.

Ilustración 29 Tabla de requisitos no funcionales. Fuente: tabla desarrollada

4. DISEÑO

4.1. Diagrama de clases

Para analizar el funcionamiento interno de la aplicación realizamos el siguiente diagrama de clases, que muestra las principales clases en las que se compone la aplicación [Ilustración 30].

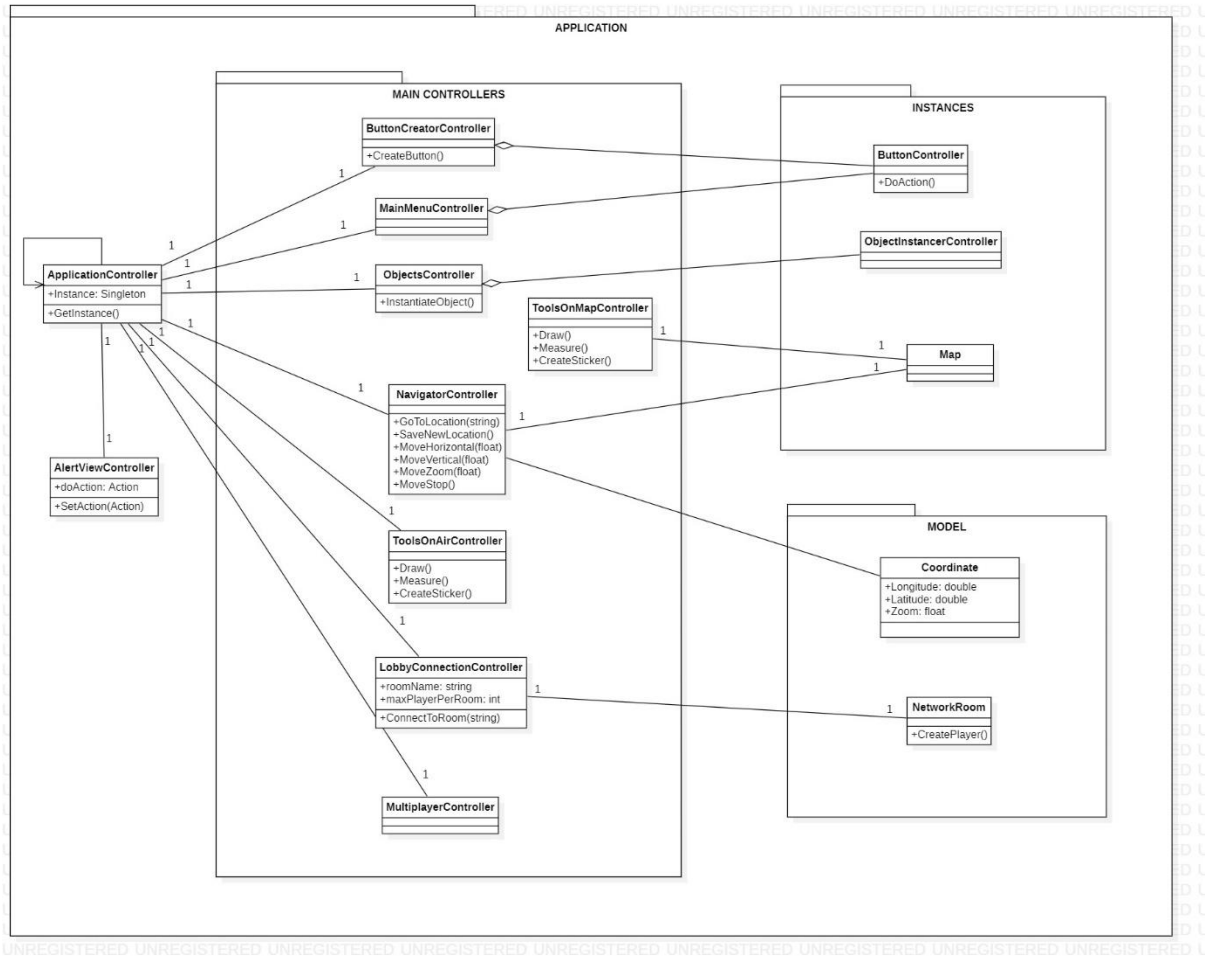

Ilustración 30 Diagrama de clases de la aplicación. Fuente: diagrama desarrollado en conjunto con el compañero

- **ApplicationController**: Es la clase central. Contiene las referencias al resto de clases importantes.

- **ButtonCreatorController**: Clase encargada de crear los botones del menú basándose en el contenido local que tengamos disponibles (objetos 3D, localizaciones guardadas, bookmarks, etc.). Permite el dinamismo a la hora de añadir contenido nuevo, ya que evita tener que crear manualmente un botón por cada contenido nuevo que se agregue a la aplicación.

- **MainMenuController**: Esta clase activa y gestiona todos los submenús y botones que posee el menú principal de la aplicación.

- **ButonController**: Un componente que poseen tanto los botones como los submenús del menú principal. Esta clase permite gestionar los cambios de selección de submenús, permite activar o desactivar los botones a la hora de mantener coherencias, y guarda las funciones a las que llaman estos botones. Permite además controlar el color que se muestra y el estado de encendido o de apagado, y coordinarse con otros botones.

- **ObjectsController**. Clase encargada de crear los objetos 3d que se muestran en el mapa y de almacenarlos.

- **ObjectInstanceController**. Los objetos 3d que se crean tienen esta clase la cual se encarga de habilitar las clases que permiten manipular el objeto, así como las que lo anclan al mapa.

- **ToolsOnMapController**: esta clase controla las herramientas de dibujar y medir en el mapa, esto forma parte de la sección de mi compañero.

- **NavigatorController**: Gestiona principalmente las localizaciones, permite saltar entre ellas y disponer de nuevas cuando se desea.

- **Map**. El mapa es un objeto que contiene diversas clases del plugin de OnlineMaps. Estas clases se encargan de crear y mostrar el mapa, y poseen las herramientas para poder navegar o añadirle datos.

- **Coordinates**. Clase que almacena un modelo de coordenadas latitud longitud y zoom.

- **ToolsOnAirController**. esta clase controla las herramientas de dibujar y medir en el aire, esto forma parte de la sección de mi compañero.

- **LobbyConnectionController**. Clase encargada de realizar la conexión multiusuario, forma parte de la sección de mi compañero.

- **NetworkRoom**. Clase de datos que almacena la información de la sala multiusuario, forma parte de la sección de mi compañero.

- **AlertViewController**. Clase utilizada para crear un panel con información para el usuario. Es el principal elemento de comunicación con el usuario, permitiéndole recibir mensajes de error cuando corresponda, o eventualidades tipo tutoriales.

4.2. Diseño del menú

La aplicación posee un menú principal el cual está diseñado para aparecer en la palma de la mano del usuario, es decir, cuando el usuario quiere abrir el menú solo tiene que mirarse la palma de la mano y el menú aparecerá delante suya, y se quedará ahí hasta que el usuario lo cierre o lo abra en otra posición.

El menú consta de una sección principal de botones agrupados en una coluna [Ilustración 31], y cada vez que se selecciona uno se abre un submenú con otros botones. Una vez se pulsa un botón este queda resaltado y el submenú aparece a la derecha.

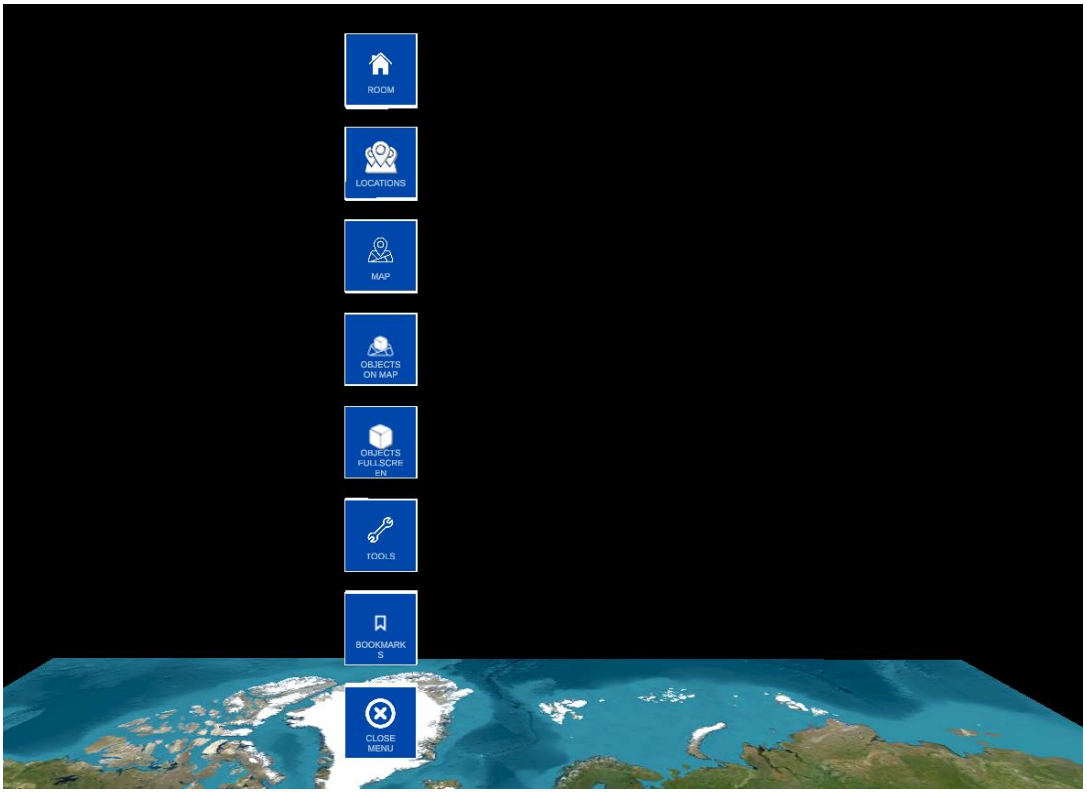

Ilustración 31 Menú principal. Fuente: aplicación vista desde el editor

A continuación, se muestran los submenús que tiene la aplicación:

- **Room**. Tiene botones para configurar el chat de voz, silenciar la aplicación o silenciar el micrófono propio [Ilustración 32].

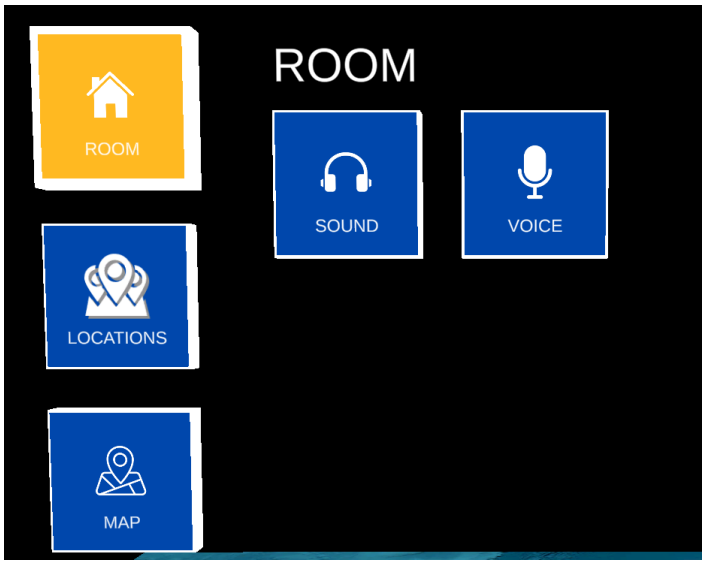

Ilustración 32 Submenú Room. Fuente: aplicación vista desde el editor

- **Locations**. Muestra localizaciones a las que uno se puede desplazar directamente. Cuando se pulsa un botón de una de ellas el mapa se mueve a ese destino. También a la derecha hay un panel en el cual se puede escribir para darle nombre y guardar las coordenadas actuales del mapa [Ilustración 33]. Cuando el usuario presiona sobre el campo de texto, aparece un teclado flotante alfanumérico para ponerle nombre a la localización.

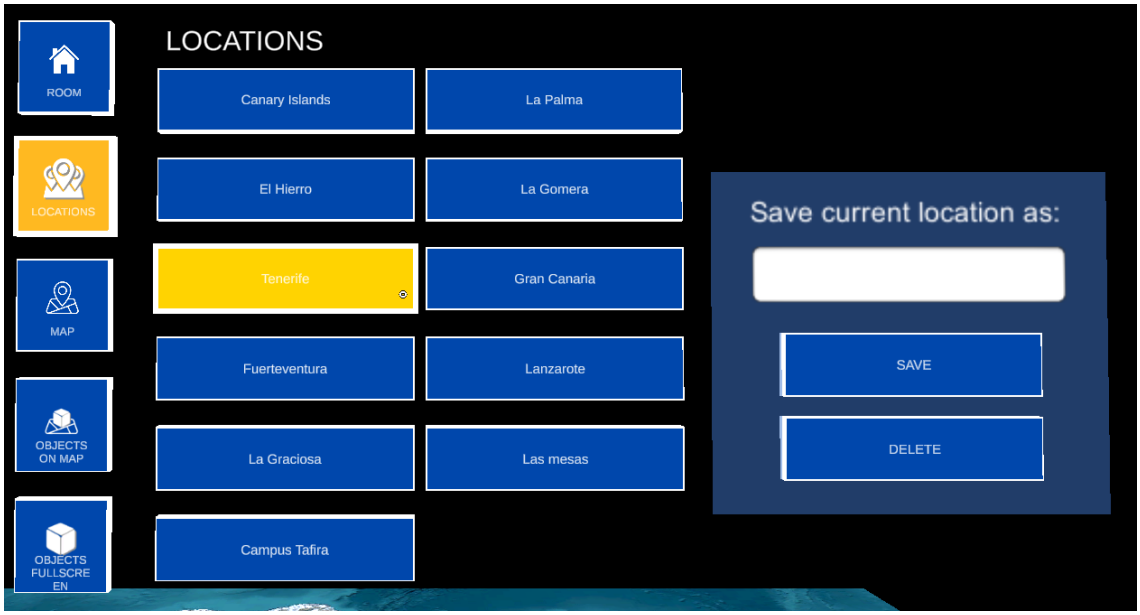

Ilustración 33 Submenú Locations. Fuente: aplicación vista desde el editor

- **Map**. Este submenú consiste en varios botones para operar con el mapa, desplazar las coordenadas que muestra viajando al norte, sur, este y oeste; ampliando o alejando el zoom; subir o bajar el mapa físicamente para que esté a la altura del suelo, o a la altura del pecho [Ilustración 34].

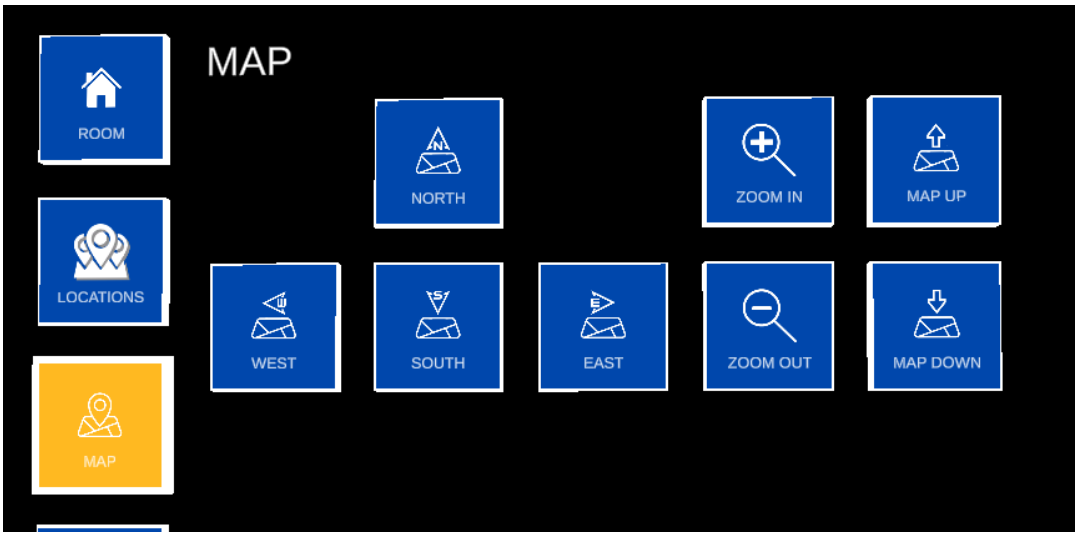

Ilustración 34 Submenú Map. Fuente: aplicación vista desde el editor

- **Objects on map**. Sirve para crear los objetos que mostramos en el mapa. Pulsar en uno de los botones crea el objeto 3d en el centro del mapa, para luego poder manipularlo con las manos[Ilustración 35].

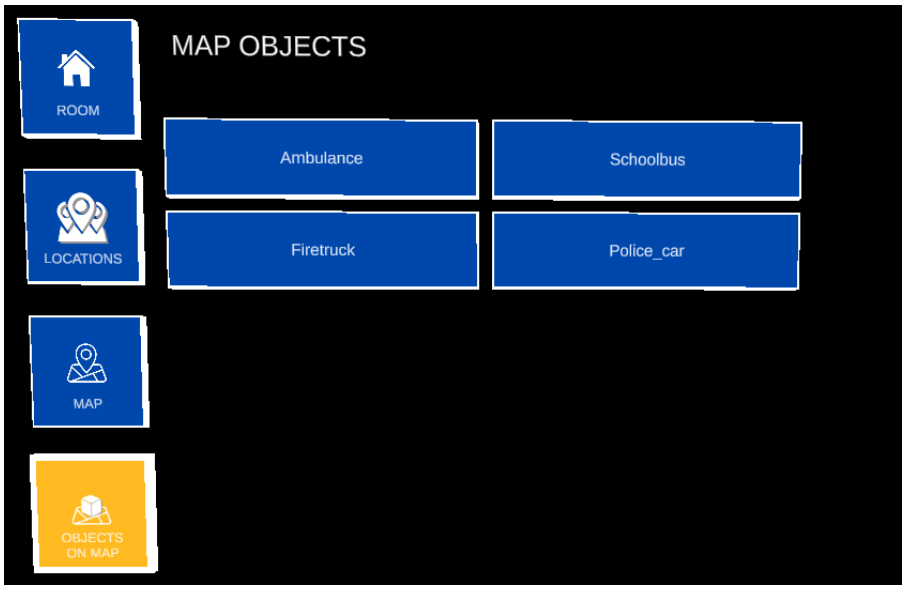

Ilustración 35 Submenú Map Objects. Fuente: aplicación vista desde el editor

- **Objects Fullscreen**. El menú es igual que el anterior pero estos botones muestran una versión grande del objeto que desea el usuario, para poder centrarse en examinar el modelo [Ilustración 36].

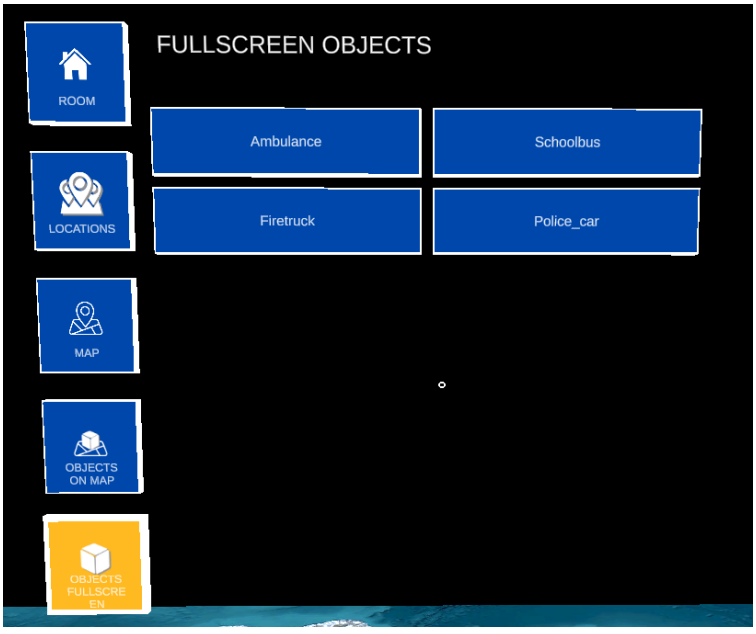

Ilustración 36 Submenú Objects Fullscreen. Fuente: aplicación vista desde el editor

- **Tools**. Este submenú contiene los botones para poder dibujar y medir [Ilustración 37]. Cuando el usuario pulsa para dibujar o medir o elegir una sticker, el submenú se expande con más opciones [Ilustración 38].

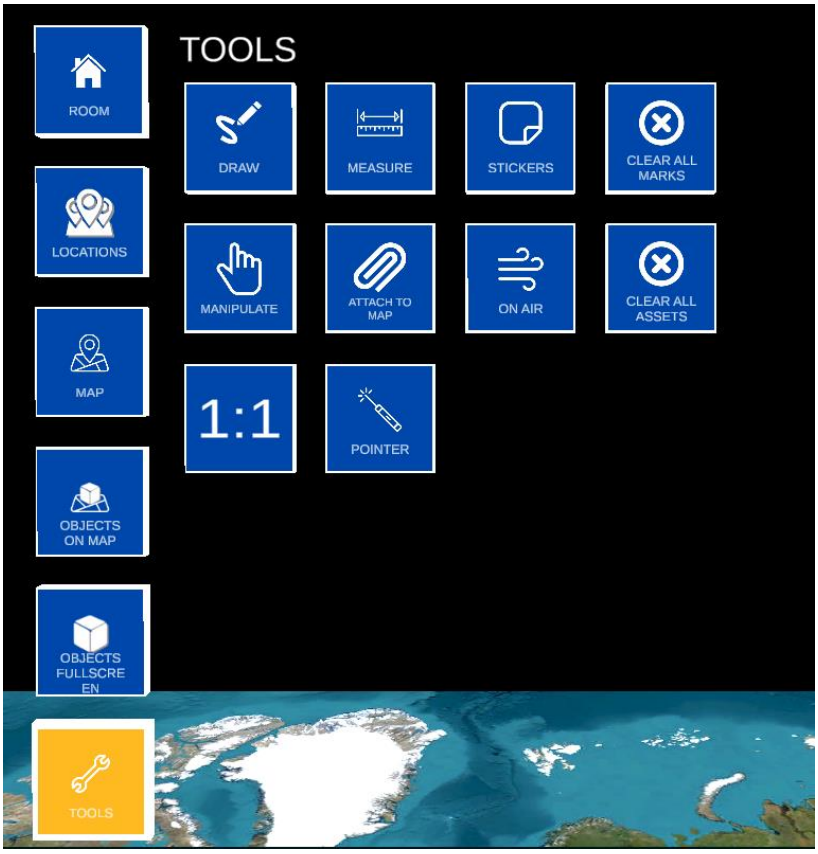

Ilustración 37 Submenú Tools. Fuente: aplicación vista desde el editor

Ilustración 38 submenú herramientas con las opciones de dibujar y stickers desplegadas. Fuente: aplicación vista desde el editor

- **Bookmarks**. El último submenú contiene un botón para guardar el estado actual de la sala, y en la lista aparecen anteriores escenas guardadas con el día y la hora [Ilustración 39].

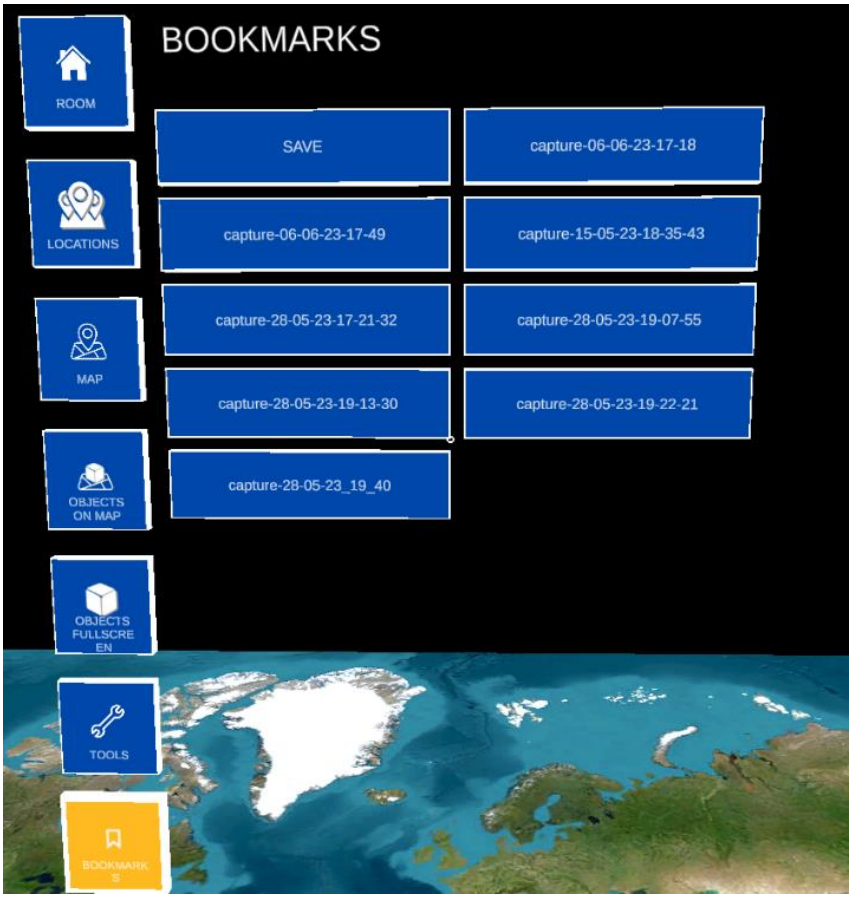

Ilustración 39 submenú Bookmarks

- Finalmente, el menú tiene un botón para cerrarlo, pudiéndose abrir de nuevo mirándose la palma de la mano [Ilustración 40].

Ilustración 40 Botón Close. Fuente: aplicación vista desde el editor

5. PLANIFICACIÓN

Para poder desarrollar el trabajo en conjunto se planteó un modelo de desarrollo iterativo e incremental, basado en la metodología scrum, en la cual se intentaría tener esprints de dos tres semanas, donde se añadirían nuevas funcionalidades y arreglar posibles fallos o problemas. Estos sprints terminarían en una reunión en conjunto, para poder unificar los avances de cada uno, y evitar posibles conflictos, y cada 4 semanas, una reunión con alguno de los tutores académicos para mostrar avances y resolver dudas. El uso de la expresión "basado" es debida a que no llega a cumplir el método scrum del todo. Debido a que ambos compañeros trabajamos en una empresa, la falta de tiempo causaba que los sprints fueran de duraciones variables, alternando entre 2 y 4 semanas. Luego, cada 4 semanas se tenía la interacción con alguno de los tutores (que hacían el papel del Product Owner) para mostrar resultados y definir los requisitos para el siguiente sprint, pero además de estas reuniones, mi compañero y yo realizaríamos reuniones entre sprints, algunas en intervalos regulares, y alguna en días concretos, para poder fusionar funciones, asegurando de que lo que hiciera cada uno no interfiera con lo que hiciera el otro, y prestar ayuda mutua en algún caso.

La hoja de ruta inicial consistía en 7 módulos principales, uno en conjunto, y luego 6 módulos repartidos en 3 para cada uno, desarrollándose en paralelo [Ilustración 41].

MÓDULOS DEL PROYECTO	
En el primer módulo se haría el análisis y el diseño de la aplicación, y la creación de un primer prototipo. Se crea el proyecto con una escena, se añade el mapa y se trabaja en sus principales controladores. Aunque la fase era conjunta se dividió el trabajo en pequeñas funciones para dinamizar (por ejemplo, mientras que yo realizaba la integración del mapa en la escena mi compañero trabajaba la base del menú principal).	
Módulos asignados a mí	**Módulos asignados a mi compañero**
Esta fase consistiría en la investigación del funcionamiento de la realidad virtual y mixta, del aprendizaje para utilizar e integrar de la herramienta para poder trabajar con las gafas, y crear una aplicación que pudiera verse correctamente en las gafas.	Esta fase la lleva a cabo el compañero en paralelo a la fase dos, dedicándose a investigar Photon para poder tener una aplicación multiusuario.
Programar la función de crear objetos en el mapa y manipularlos. Trabajar en el funcionamiento online de estas funciones.	Programar las herramientas de dibujo y medición, adaptarlas al funcionamiento online.
Hacer un sistema de guardado de escena para poder guardar sesiones y mejorar el sistema de sincronización	Mejoras de comunicación e interacción, introducción de avatares y chat de voz

Ilustración 41 Distribución del trabajo en módulos. Fuente: tabla desarrollada

Teniendo en cuenta este sistema, se fueron realizando sprints con incrementos para el prototipo:

- Primer sprint, en conjunto se deja el mapa creado en la escena, y se crean los principales controladores.
- Segundo sprint, se dispondría de una versión multiusuario, funcional en las gafas de realidad mixta, con algunos fallos. En este sprint mi compañero se encargaría de la parte multiusuario, y yo realizaría una primera integración de la parte de realidad mixta.
- Tercer sprint, se crean las primeras funciones y herramientas en el mapa para poder interactuar con él, en mi caso la función de instanciar objetos y utilizar localizaciones grabadas para el mapa.
- Cuarto sprint, se continuaría desarrollando las herramientas de cada uno y corrigiendo fallos. En mi caso se desarrolló el sistema de guardado de escena.
- Quinto sprint, Corrección de errores y terminando el desarrollo de las herramientas.
- Sexto sprint, correcciones de bugs, comprobaciones de la correcta fusión de todas las herramientas en conjunto, finalizar los asuntos de multiusuario.

6. DESARROLLO

Como se menciona en el apartado de planificación, el proyecto se fue realizando de manera incremental e iterativa, empezamos creando un proyecto de Unity al cual se le irían añadiendo paquetes, clases y funciones, creando un prototipo de aplicación nuevo cada vez que se añadía una función.

En este apartado se narra el desarrollo de la aplicación, explicando las clases y funciones que se fueron haciendo en los sprints. Las partes hechas por mi compañero se mencionarán vagamente, este trabajo está centrado en las que partes hechas por mi o en conjunto.

Durante al apartado de desarrollo pasa a utilizarse la primera persona en vez de utilizar un texto impersonal, para poder distinguir lo que hago yo frente a lo que hace mi compañero o hacemos en conjunto.

6.1. Primer sprint, creación de un proyecto mínimamente funcional

En este sprint se empezó a crear el proyecto de Unity.

6.1.1. Instalación básica de los paquetes

Creamos una escena, instalamos los paquetes de **OnlineMaps**, para poder utilizar el mapa, y el paquete de **Mixed Reality Toolkit**, para poder integrar las gafas (aunque la integración se dejó para el siguiente sprint), y algunos de los controladores principales [Ilustración 42].

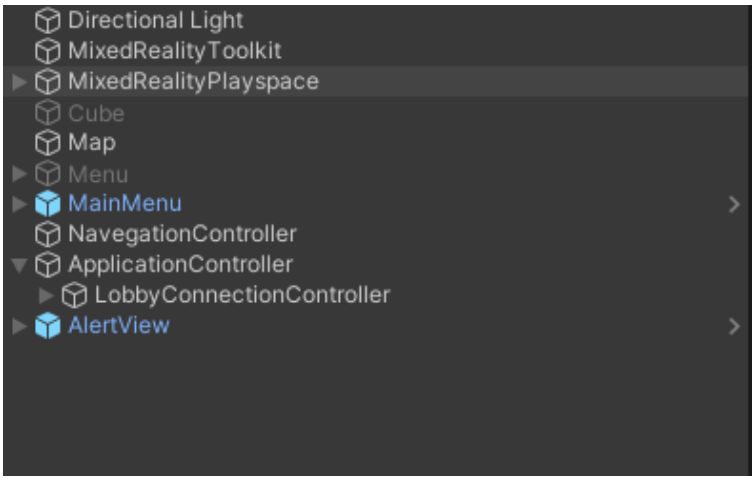

Ilustración 42 Escena del primer sprint. Fuente: aplicación vista desde el editor

Al instalar los paquetes del MRTK y del OnlineMaps, añadimos lo más básico a la escena:

Con el MRTK pusimos la configuración base (añadir dos componentes a la escena, se habla más en detalle de ello en el siguiente sprint)

Con el OnlineMaps utilizamos el paquete para generar un mapa (es el objeto Map de la escena), aunque tuvimos que estar probando diferentes parámetros y modos para poder adaptarlo a la forma que queríamos, ya que por defecto el mapa se instancia con un tamaño de 1024 metros cuadrados, tan grande que el proyecto no lo mostraba entero (para no gastar memoria), y con un script incómodo que hacía que la cámara siempre mirase hacia el centro del mapa [Ilustración 43].

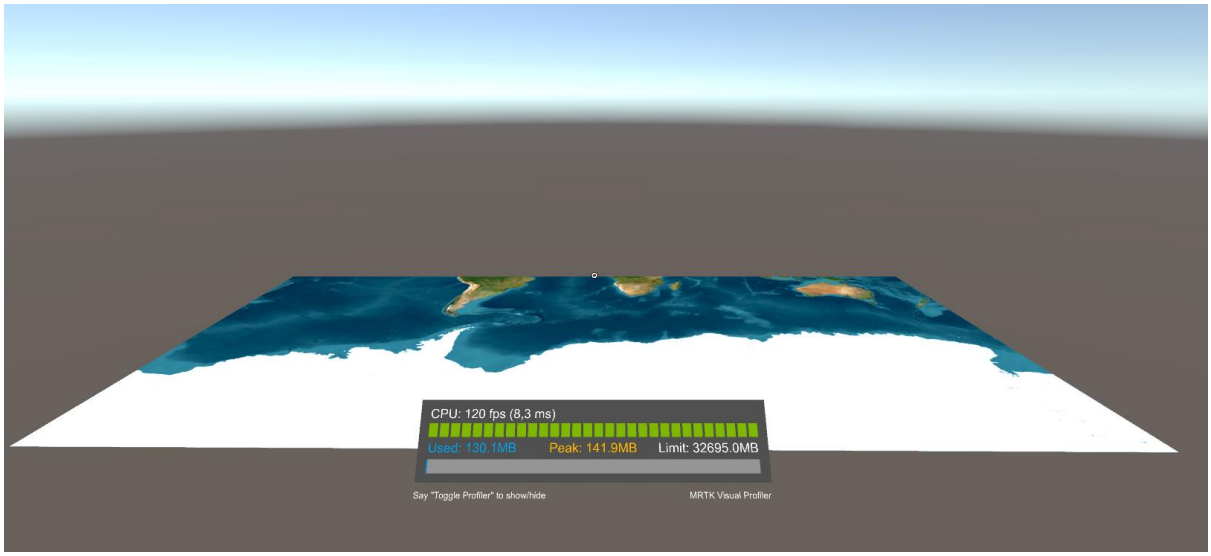

Ilustración 43 primera vista del mapa. Fuente: aplicación vista desde el editor

6.1.2. Primeros Controladores

En la escena creamos un objeto con un script que será el **ApplicationController**. Este script será el nexo que permita a todos los controladores de la aplicación interactuar entre sí, y para ello tendrá una versión de patrón Singleton o instancia única, solo existirá una única instancia de este script en la escena, y este tendrá referencias a las instancias del resto de controladores, todos ellos instancias únicas.

Aparte de la función principal de hacer de nexo, ApplicationController se hará cargo de invocar la llamada inicial al menú de la aplicación, posee la llamada para apagar la aplicación y también realizará el cambio de cultura al iniciar todo [Ilustración 44].

```
Mensaje de Unity | 0 referencias
private void Awake()
{
    CultureInfo.CurrentCulture = CultureInfo.InvariantCulture;
    GameObject.Find("Directional Light").GetComponent<Light>().enabled = true;
    if (instance == null)
    {
        instance = this;
        DontDestroyOnLoad(this.gameObject);
    }
    else
    {
        Destroy(this.gameObject);
    }
}
```

Ilustración 44 función inicial del ApplicationController. Fuente: aplicación vista desde el editor

El cambio de cultura es importante debido a que, como hemos descubierto en nuestro tiempo trabajando en la empresa, si en algún momento intentamos pedirle a un usuario o a algún servidor datos que contengan números decimales, la conversión de String a Double o Float podría dar errores ya que hay culturas que representan el decimal con el símbolo del punto '.', y otras culturas con el símbolo de la coma ','.

En este sprint se crea también el **NavigatorController**, el script encargado de proporcionarle al mapa las coordenadas a las que el usuario desea ir. Estas coordenadas se almacenan en un diccionario tipo <String, Coordinates> [Ilustración 46], donde la String es un nombre asociado al objeto Coordinates, que tiene un Double para la latitud, otro para la longitud, y un Float para el zoom [Ilustración 45].

```
16 referencias
public struct Coordinates
{
    11 referencias
    public Coordinates(double longitude, double latitude, int zoom)
    {
        Longitude = longitude;
        Latitude = latitude;
        Zoom = zoom;
    }

    2 referencias
    public double Longitude { get; set; }
    2 referencias
    public double Latitude { get; set; }
    2 referencias
    public int Zoom { get; set; }
}
```

Ilustración 45 Código del objeto Coordinates. Fuente: aplicación vista desde el editor

```
locations = new Dictionary<string, Coordinates>();

//default locations;
locations.Add("Canary Islands", new Coordinates(-15.8119423114983, 28.502016673201, 8));
locations.Add("La Palma", new Coordinates(-17.868, 28.65, 10));
locations.Add("El Hierro", new Coordinates(-18.0233580435249, 27.7478482628998, 10));
locations.Add("La Gomera", new Coordinates(-17.232997310054, 28.1186752786148, 10));
locations.Add("Tenerife", new Coordinates(-16.5104157687774, 28.262289943536, 10));
locations.Add("Gran Canaria", new Coordinates(-15.5909935336019, 27.9513721928408, 10));
locations.Add("Fuerteventura", new Coordinates(-14.0469908873591, 28.3908768588089, 10));
locations.Add("Lanzarote", new Coordinates(-13.6229081365003, 29.0780849026683, 10));
locations.Add("La Graciosa", new Coordinates(-13.5104862910405, 29.2483466207175, 10));
locations.Add("Las Palmas", new Coordinates(-15.4425174318652, 28.1525397636108, 16));
```

Ilustración 46 Diccionario que almacena las localizaciones y el formato en el que se guardan (de ejemplo están las Islas Canarias). Fuente: aplicación vista desde el editor

Mi compañero realizo una versión del menú principal utilizando un modelo de diseño que aprendimos ambos en la empresa, crear un script que manejara el menú principal y controlara las interacciones entre los botones y los submenús, y otro script para los botones, encargados de mostrar los iconos el texto y almacenar las funciones que realizarían.

Luego en ese menú yo creo un submenú de localizaciones en el cual el NavigatorController crea botones dinámicamente con las localizaciones que tenía guardadas. Este método crea los botones y les asigna la función de viajar a las coordenadas que estén guardadas en la lista de localizaciones y lleven el nombre que se muestra en el botón [Ilustración 47].

```
public void CreateNewLocationButton(string name)
{
    GameObject button = Instantiate(textButtonPrefab, LocationsButtonPosition.transform);
    button.name = name;

    ButtonController buttonController = button.GetComponent<ButtonController>();
    Debug.Log("Buttons in Presentations: " + buttonController.GetItemId());
    buttonController.SetItemId("Locations:" + name);
    buttonController.SetLabel(name);

    buttonController.ChangeInteractionType(ButtonController.InteractionType.CLICK);

    buttonController.invokeMethodClick.AddListener(delegate { ApplicationController.GetInstance().navigatorController.GoToLocation(name); });

    LocationsButtons.Add(button);
}
```

Ilustración 47 Método para crear un botón de localización. Fuente: aplicación vista desde el editor

6.2. Segundo sprint, integrar las funciones para que funcione la aplicación en las gafas y multiusuario

6.2.1. MRTK

En el primer sprint solamente se integró la herramienta de MRTK en el proyecto y en la escena. Cuando haces eso, si generas directamente una versión podrías ver la escena en las gafas, y como tenemos ya el mapa, ver el mapa en la escena, pero no podrías interactuar con nada ni hacer nada además te sale una consola de rendimiento que molesta (Ilustración 43 primera vista del mapa).

Luego la herramienta principal para usar las gafas, la clase **MixedrealityTookit** permite editar numerosos parámetros para poder ajustar el proyecto, como elegir la configuración para el modelo que se vaya usar (puedes tener una configuración para HoloLens, otra que funcione en PC, o una básica que sirva para otros dispositivos VR, como las meta Quest); habilitar un sistema de teletransporte dentro de la escena; habilitar el seguimiento de las manos y la representación de estas; mostrar una maya generada con la información espacial que recibe de las cámaras; y otras funciones varias [Ilustración 48].

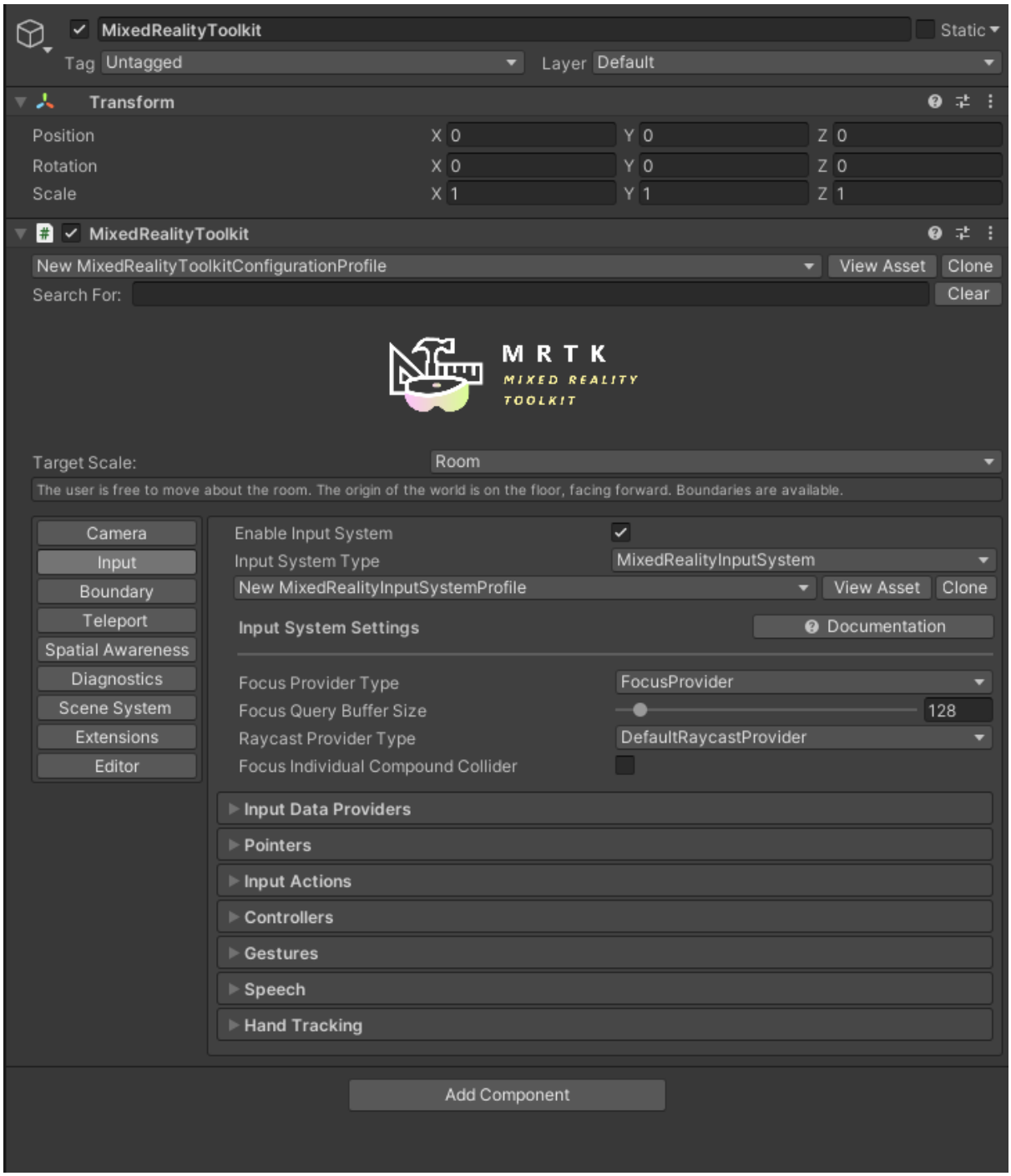

Ilustración 48 MixedRealityToolkit. Fuente: aplicación vista desde el editor

6.2.2. Posición del menú

El menú originalmente aparece en medio de la escena, pero lo configuro de forma que aparece cuando el usuario se mira la palma de la mano durante más o menos medio segundo. El menú aparece y se queda en una posición fija para facilitar el uso y permite cambiarlo de sitio volviendo a mirarse la palma de la mano en otra posición. Para facilitar eso creo un objeto vacío que llamo **MenuHolder**, al que le añadimos un script del MRTK llamado **HandConstraintPalmUp**, y este a su vez añade el **HandBounds** y el **SolverHandler**. El

HandConstraintPalmUp es un script que permite asignarle una función al gesto de enseñar la palma a la cámara principal, o a el objeto que posee el script, como por ejemplo que cuando apuntes con la palma de la mano a un objeto este desaparezca, o, en nuestro caso, que cuando apuntes con la palma de la mano hacia ti mismo abras el menú. Y para que esto sea posible al objeto que posee este script se le asocian el script HandBounds, que permite detectar lo viene siendo la mano, y el SolverHandler, un script general que le otorga al objeto que lo posee la posición de lo que se trakee, en este caso son las palmas de ambas manos, pero se puede especificar una de las manos como la cabeza, un puntero al que apunte la mano, o incluso a uno de los esqueletos de una mano en concreto, tipo el nudillo del dedo anular, si así se desea [Ilustración 49].

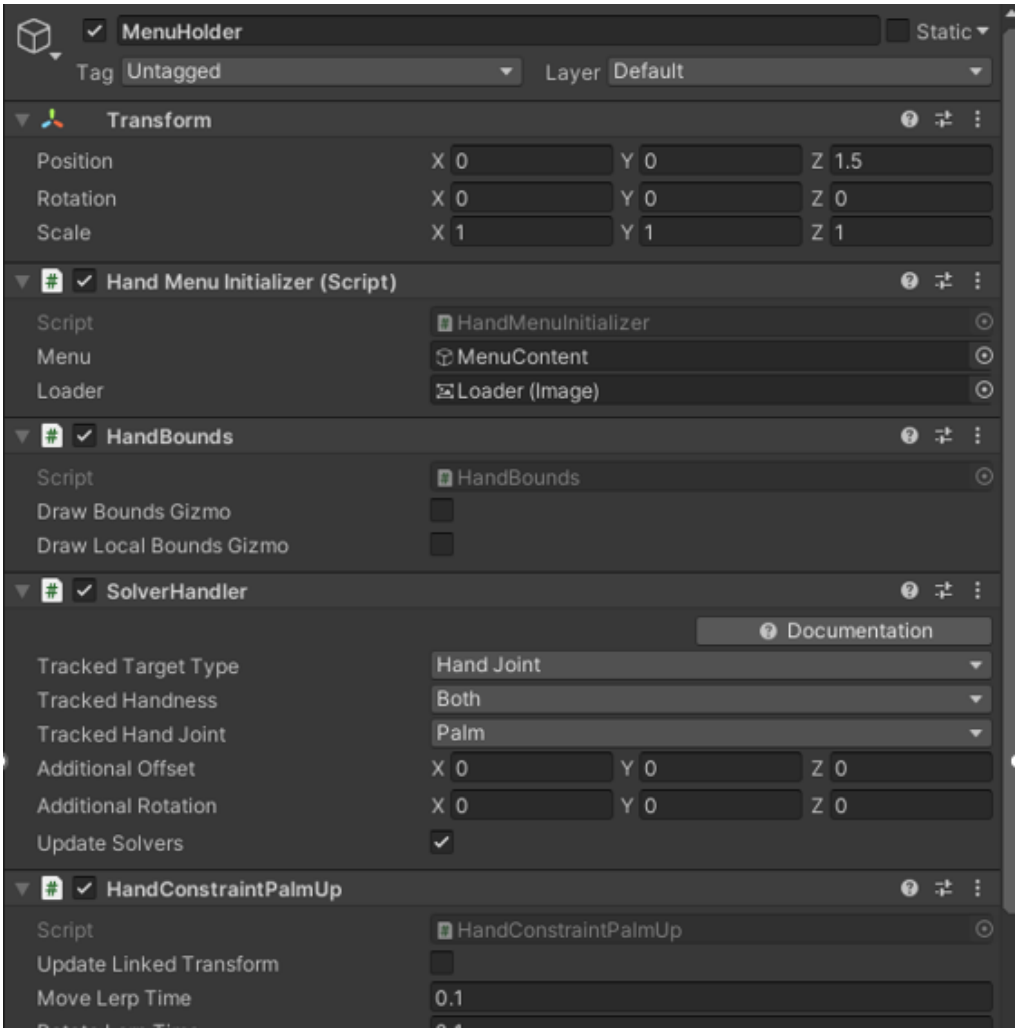

Ilustración 49 Scripts puestos en el objeto MenuHolder. Fuente: aplicación vista desde el editor

Originalmente estos scripts estaban en el objeto que contiene el menú entero, pero al hacer eso el menú se instanciaba siempre que el usuario mostrara la palma, lo cual podía suceder sin querer, causando que el menú se moviese sin que el usuario quiera. Por eso, él se creó el script **HandMenuInitialicer**, y este script posee una función que espera medio segundo mientras se esté viendo la palma de la mano y después de eso mueve el menú hasta la posición de la mano

6.3. Tercer sprint, desarrollo de las primeras herramientas, instanciar objetos en el mapa y guardar coordenadas

6.3.1. ObjectsController

Para crear los objetos en el mapa, creo un nuevo controlador anexado al ApplicationController, el **ObjectsController**, cuya función es gestionar los contenidos 3d. Se encarga de instanciar los objetos, tanto en la versión del mapa como en la versión de pantalla completa, y gestiones del tipo almacenarlos o borrarlos.

El procedimiento para instanciar los objetos consiste en un método que crea un prefab llamado **Holder** (hablaré sobre él en el siguiente párrafo), en el cual añado como hijo una copia del modelo 3d que se quiere crear, y configura los parámetros del Holder de la forma correcta para el caso de uso de un objeto en el mapa o en formato fullscreen (además de poner los objetos sobre el mapa hay otra función que sirve para visualizar solamente el objeto, para poder examinarlo) [Ilustración 50].

Ilustración 50 Objeto instanciado en el mapa. Fuente: aplicación vista desde el editor

El objeto **Holder** es un objeto que posee, como se menciona en el párrafo anterior, diversos scripts útiles para lo que consiste la instancia de un objeto3D [Ilustración 51]. Estos scripts son:

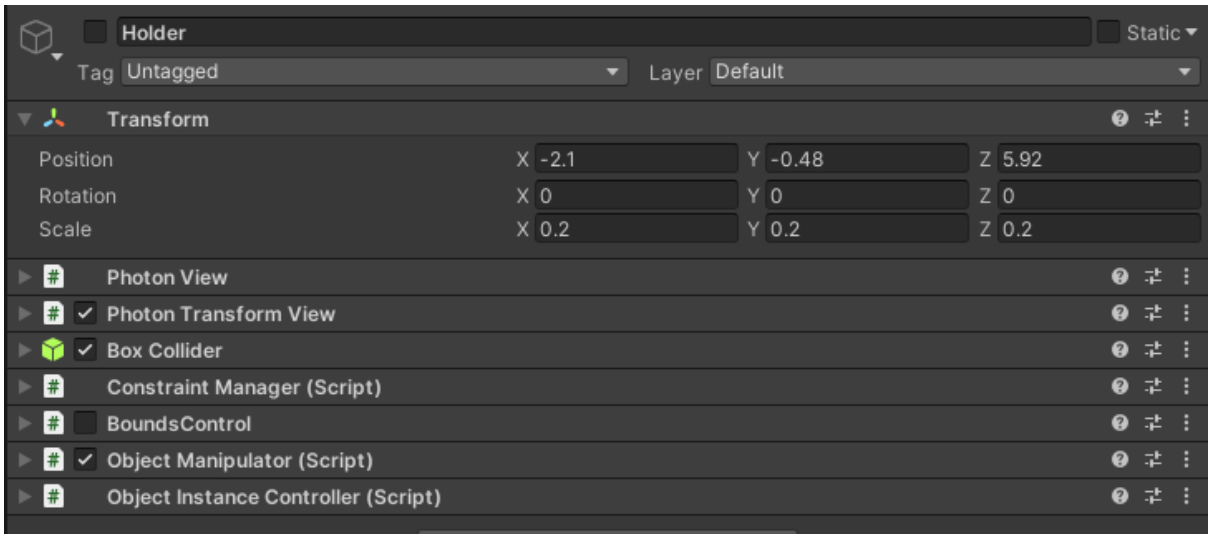

Ilustración 51 Objeto Holder. Fuente: aplicación vista desde el editor

- Un **PhothonView** y un **TransformView** que sirven para sincronizar la posición y rotación del objeto en tema de multiusuario.
- Un **BoxCollider** para que pueda interactuar con él.
- El script **BoudsControl** que permite manipular la rotación y el tamaño de un objeto utilizando su **BoxCollider**.
- El script **ObjectManipulator**, un script muy completo que habilita que un usuario pueda interactuar con el objeto poseedor de este script usando sus propias manos. Este script por sí solo aporta también un cursor que parte de las manos del usuario y le permiten seleccionar objetos o pulsar botones.
- El script **ObjectInstanceController** se encarga de asociar e integrar un objeto cualquiera creado en la escena al mapa, creando así un marcador, un objeto con unas coordenadas del mapa asociadas para que aparezcan en ellas (en la imagen siguiente se muestran los parámetros que podemos configurar para poner el objeto 3d donde queremos: la latitud, longitud, la escala, la rotación en el eje Y, etc.).

6.3.2. Guardar coordenadas como localizaciones

La otra función es guardar coordenadas que se muestran en el mapa, por si el usuario se ha desplazado por él y desea guardar el punto exacto que se está mostrando actualmente para tenerlo como localización, como un acceso rápido. Para esta función añado un panel al menú y un conjunto de scripts y métodos. El panel está compuesto de [Ilustración 52]:

- Un campo de texto en el que se introduce el nombre para guardar las coordenadas
- Un botón que llama a un método que crea una entrada nueva en el diccionario con las localizaciones con el nombre que haya puesto el usuario, y las coordenadas actuales del mapa, y posteriormente llama a crear un nuevo botón para esa entrada en tiempo de ejecución.

Ilustración 52 Panel para guardar coordenadas actuales. Fuente: aplicación vista desde el editor

Estas localizaciones creadas se guardarán en disco para que persistan en otras sesiones. Para ello creo un fichero de texto en disco donde escribo las coordenadas actuales y el nombre con el que se guardan. Este fichero se guarda en con una dirección especial, llamada PersistentDataPath, que es la única carpeta que podemos editar y acceder dentro del disco de las gafas desde la aplicación. Una vez guardadas en el fichero de disco, cuando el usuario entra a la aplicación, esta lee el fichero de texto y añade las coordenadas que están ahí escritas a las coordenadas disponibles. Con este método podemos también añadir coordenadas a la aplicación editando directamente ese fichero de texto, así podríamos poner varios puntos a la vez, y no tener que ir en el mapa para grabarlos.

Como curiosidad, con solo poner el campo de texto en la escena, este no funcionaba cuando intentas pueda interactuar con la mano del usuario, o con el puntero que sale de la mano en las gafas, así que cabe decir que se le han añadido componentes al campo de texto para que pueda ser compatible con el input de MRTK. Microsoft había creado un componente llamado **TMP_KeyboardInputField** afirmando que con solo añadirlo al campo de texto se podría interactuar perfectamente él, pero resulta que solo funciona en la versión de Unity 2018.4. Como no se podía usar este componente básicamente añadimos un Collider, al campo de texto y un NearinteractableTouchable, para que reaccionara a la mano del usuario, y después añadir en algún script (el NavigatorController ya que es el que controla la navegación) que abriera el teclado de las HoloLens al notar que la mano colisiona con el Collider del Iputfield.

6.3.3. Desarrollo del AlertView

En este sprint también desarrollo un prefab llamado **AlertView**, otra cosa aprendida de la empresa, que consiste en un panel con un texto editable y un botón de OK y otro de cancelar. Este panel sirve como herramienta para comunicarnos con el usuario, para confirmaciones, y mencionar errores (un ejemplo de uso es cuando el usuario quiere guardar una localización o

borrarla, la aplicación muestra este panel para pedirle confirmación de la acción) [Ilustración 53].

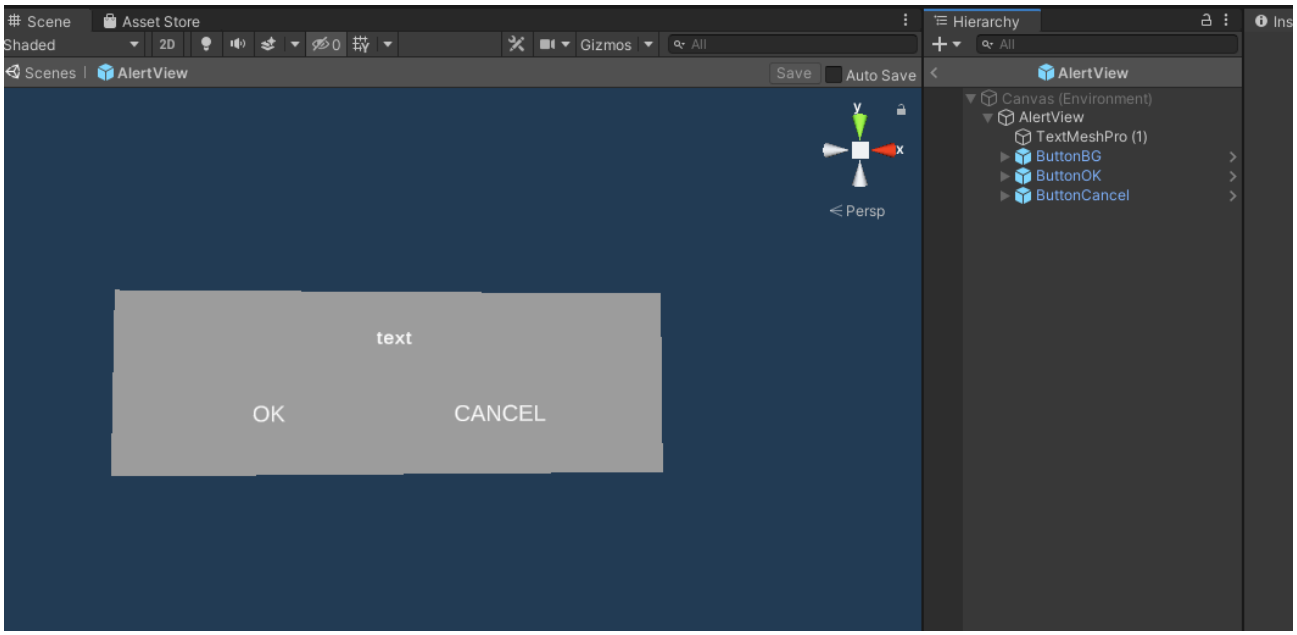

Ilustración 53 Prefab AlertView. Fuente: aplicación vista desde el editor

El prefab en sí para funcionar utiliza principalmente tres scripts: el **SolverHandler**, mencionado anteriormente, para poder ser intratable; Un script llamado **RadialView** cuya función es hacer que el objeto con ese script se mantenga en el grado de visión de la cámara; y el script **AlertViewController**. Este último sirve para poder mostrar una pestaña con un mensaje al usuario, es la principal forma de comunicación entre la aplicación y el usuario. Permite avisar de errores y también como confirmación a la hora de pulsar botones del tipo borrar un contenido o reiniciar el estado de una sala. Para usarlo, creamos un array de acciones de Unity, en el cual le ponemos todas las funciones que queramos ejecutar al pulsar okey en el AlertView, posteriormente le asignamos ese array de acciones, introducimos el texto que queremos que muestre, y lo habilitamos (véase la imagen de abajo) [Ilustración 54]. También se puede usar sin el apartado de crear el array de acciones, si solo queremos que el panel avise de algo, como un error, o que se está intentando guardar una localización con un nombre ya existente, etc.

```
System.Action[] actions = new System.Action[3];
actions[0] = () => SaveNewLocation();
ApplicationController.GetInstance().alertView.SetActions(actions);
ApplicationController.GetInstance().alertView.SetText("Do you want to save current coordinates
ApplicationController.GetInstance().alertView.EnableAlertView();
```

Ilustración 54 Ejemplo de código de uso del AlertView. Fuente: aplicación vista desde el editor

6.4. Cuarto sprint, desarrollo del sistema de guardado de escena

En este momento ya teníamos un prototipo de aplicación que permitía al usuario ir a las localizaciones disponible del mapa, moverse, crear objetos para poner sobre el mapa, y pintar diversas líneas, pero en el momento de multiusuario tenía unos ligeros problemas. La aplicación funciona de manera multiusuario porque, gracias a una función de Photon que permite avisar a otros usuarios para ejecutar una función, cuando un usuario pulsa una localización, o crea un objeto, la aplicación avise al resto de usuarios para realizar las mismas acciones, para poder sincronizar lo que están viendo. Esto funcionaba bien, pero tenía el problema de que los usuarios tenían que estar conectados para recibir esos avisos, por lo que, si un usuario entraba más tarde a la sesión, no vería los cambios realizados antes de que entrara, por lo que habría un fallo de sincronización. Photon ofrecía una solución que es meter esos avisos en un buffer para que al entrar un usuario nuevo los viese y los siguiera, como unas instrucciones, pero estas se traspapelaban y causaban conflictos entre ellas. Para solucionar este problema decidí crear una clase que guardase el estado actual de la sala, indicando las coordenadas que muestra el mapa en ese momento, los objetos creados y sus posiciones, igual que los dibujos y mediciones para que, cuando un usuario entrara tarde, pudiera saber el estado actual de la sesión, y recrearla. Además, esa clase podría pasarse a formato json (una restra de caracteres que describen un objeto, como un html), y escribirse en un fichero de texto para poder guardarse en disco y recrear la sesión en otro momento. En adición a la clase RoomState se crea una clase llamada SceneContentController para controlar la clase de datos que es RoomState [Ilustración 55].

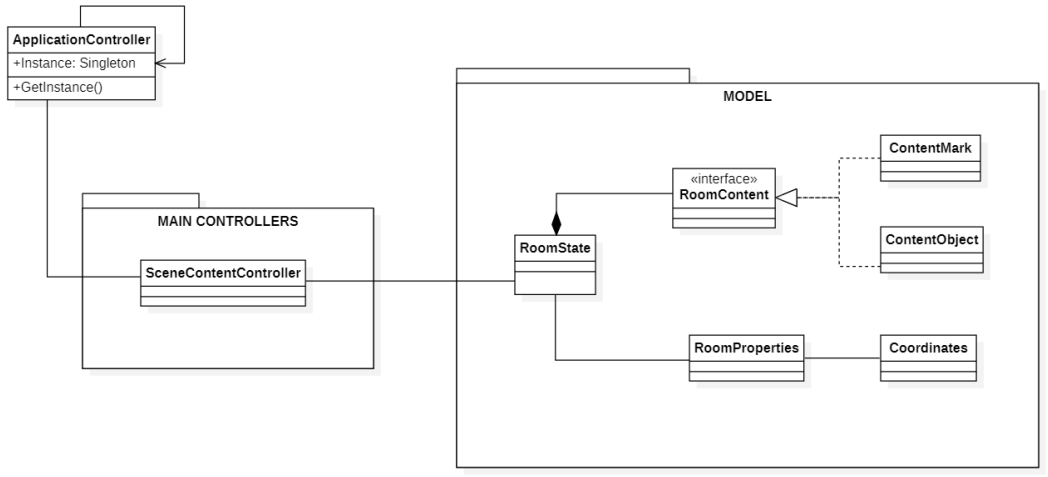

Ilustración 55 Adicción del RoomState y SceneContentController al diagrama de clases. Fuente: diagrama desarrollado

6.4.1. RoomState

El **RoomState** es una clase serializable cuyo objetivo principal es guardar valores que indican la situación actual de la escena en la aplicación, para luego serializarse en un Json que se guarda, y posteriormente, deserializarlo de nuevo y poder obtener el estado que indica.

Esta clase se divide en dos partes: **RoomProperties** y **RoomContent**.

RoomProperties es una clase interna a Roomstate, y contiene los datos principales del mapa: el estado (si está visible o no), las coordenadas actuales, las dimensiones del mapa, la posición del mapa en la escena y la posición de su padre (un asunto relacionado con subir y bajar el map) [Ilustración 56].

```
[Serializable]
2 referencias
public class RoomProperties
{
    public State state = State.MAP;
    public Coordinates currentCoordinatesZoom = new Coordinates(0, 0, 0);
    public Vector2 sizeInScene;
    public int width;
    public int height;
    public Vector3 MapPos;
    public Vector3 MapParentPos;
}
```

Ilustración 56 Clase RoomProperties. Fuente: aplicación vista desde el editor

RoomContents no es una clase, sino una lista que contiene todo el contenido actual de la escena (los objetos instanciados, las líneas pintadas, los stickers, etc.) [Ilustración 57]. La lista es de objetos tipo **RoomContent**, una clase abstracta de la que parten otras clases que guardan la información de los contenidos.

La clase RoomContent por si sola solo tiene un parámetro para identificar el contenido, pero de esta clase parten otras dos que si guardan más información: **ContentObject** y **ContentMark**

```
[Serializable]
11 referencias
public class RoomContent
{
    public string contentResourceId;
};
```

Ilustración 57 Clase RoomContent. Fuente: aplicación vista desde el editor

ContentObject sirve para guardar la información de los objetos 3d, posee diversos campos [Ilustración 58]:

- Un número llamado viewId, que guarda el photon viewid, un identificador que permite a la aplicación al usar multiusuario para poder identificar cuando dos objetos son el mismo. Lo guardamos en el RoomState para el caso de entrar un usuario a la aplicación después de que alguien haya manipulado cosas ya dentro. Este id permite que el usuario recién llegado asocie la información del ContentObject a un objeto que ya se haya creado en la escena por el usuario que entró antes.

- Un boolean isOnMap indicando si el objeto se instancia sobre el mapa o en el modo fullScreen.

- Un parámetro de la clase Coordinates, mencionada anteriormente, que guarda la latitud, longitud y zoom en la que se encuentra el objeto.

- Un Vector3 con la posición en la escena de Unity en la que se encuentra el objeto. Gracias a esto, cuando se una escena guardada, el usuario sabe dónde poner el objeto que ha visto que existe en la escena. También ayuda en la parte de sincronizar al entrar en la sala a posteriori. Originalmente gracias a Photon, con el PhotonTransformView (se menciona en el primer sprint, en el Holder), si un usuario mueve un objeto, esta acción se sincroniza para el resto y todos los usuarios ponen el objeto en el mismo sitio, aunque tiene una limitación y es que no sincroniza al crearse así que la primera vez hay que ponerlos por código en el sitio que se sabe dónde está (por el RoomState), y ya, una vez cualquiera de los usuarios mueva el objeto, el resto de usuarios lo pondrán en el mismo sitio.

- Un Quaternion para indicar la rotación, misma finalidad que el vector posición que mencionamos en el punto anterior.

- Un Vector 3 para la escala, para saber la escala del objeto.

```
[Serializable]
25 referencias
public class ContentObject : RoomContent
{
    public int viewId;
    public bool isOnMap;
    public Coordinates coordinates;
    public Vector3 position;
    public Quaternion rotation;
    public Vector3 scale;
}
```

Ilustración 58 Clase ContentObject. Fuente: aplicación vista desde el editor

ContentMark es una clase con la misma finalidad que ContentObject, pero sirve para guardar la información de las líneas y stickers [Ilustración 59]. Contiene también diversos parámetros:

- Una string lineId, un identificador para poder identificar la línea si ya está creada (caso de entrar en la sala a posteriori), para poder ponerle los mismos atributos (puntos, tamaño, color, grosor...).

- Un parámetro de la clase Coordinates, para cuando la línea está anclada al mapa saber las coordenadas a las que está anclada (las líneas pueden estar ancladas o no al mapa, eso hace que los trazos mantengan una posición con respecto a un punto real del mapa).

- Un Vector3 para la posición en la escena del objeto que representa en sí la línea, el contenedor de esta.

- Un Quaternion que guarda la rotación del contenedor de la línea (normalmente siempre es cero, pero si en algún momento se modifica se guardaría en este parámetro).

- Un Vector3 que es la escala (igual que el Quaternion, suele ser siempre la unidad, pero si se cambia se guardaría en este parámetro).

- Un float thickness que contiene el valor del grosor de la línea.

- Un entero color para el color de la línea (es el índice de una paleta de colores).

- Un entero para la textura (es el índice en un conjunto de stikers predefinidos).

- Un array de Vector3 llamado pointPositions que, como su nombre indica, contiene las posiciones de todos los puntos de la línea. Esto permite poder reconstruir la línea tal cual, en cualquier momento, cargando de la escena, o entrando posteriormente en la sala. El funcionamiento normal de la línea permite, si los usuarios están conectados en la escena, que la línea se dibuje a la vez en el mismo sitio para todos los usuarios en sus respectivos visores, pero claro eso solo pasa si estás conectado en el momento de crear la línea, de ahí lo de guardarla en el RoomState para luego. Además, se probó usar el método de dejar en el buffer las acciones a llevar a cabo para crear la línea, pero la aplicación al ejecutar esos pasos del buffer se confundía y cambiaba el orden de los puntos o incluso si hacías más de una línea, mezclarlas entre sí, así que se descartó.

```
[Serializable]
7 referencias
public class ContentMark : RoomContent
{
    public string lineId;
    public Coordinates coordinates;
    public Vector3 position;
    public Quaternion rotation;
    public Vector3 scale;
    public float thickness;
    public int color;
    public int texture;
    //public LineDrawController.Mode markType;
    public Vector3[] pointPositions;
}
```

Ilustración 59 Clase ContentMark. Fuente: aplicación vista desde el editor

La clase RoomState es una clase que almacena datos. Para interactuar con ella creo otra clase con una instancia de RoomState y métodos para manipularla, para así tener una separación modelo controlador.

6.4.2. SceneContentController

Esta clase tiene métodos para cambiar los datos que guarda el RoomState, métodos para guardar un RoomState en disco (escenas guardadas) y otro método para disponer la aplicación según indique un RoomState guardado, ahora entro en detalle.

- Tiene el método **UpdateState**, uno de los más importantes [Ilustración 60]. Este método se llama cada vez que se añade, edita, o elimina algo del RoomState. Lo que hace es convertir la clase RoomState en un Json, y subirlo como String a las propiedades de la escena o RoomProperties. Esto último es un sistema que utiliza Photon, el servidor multiusuario, que es asociar una serie de variables como propiedades de la sala a la que estén conectados los usuarios. Digamos que la sala de Photon es un aula y las RoomProperties son cosas que están escritas en la pizarra, así todo el usuario que entre las puede ver. Pues bien, este método lo que hace es escribir en esa "pizarra" el estado actual de la sala, para que los usuarios nuevos lo lean al entrar, vean cómo está la escena, y la organicen, usando otro método de la clase actual, que se mencionará a posteriori.

```
private void UpdateState()
{
    string json = JsonUtility.ToJson(roomState);
    Debug.Log("SAVING " + json);
    //Debug.Log("mmm " + roomState.roomContent.Count);

    ExitGames.Client.Photon.Hashtable properties;
    if (PhotonNetwork.CurrentRoom != null && PhotonNetwork.CurrentRoom.CustomProperties != null)
    {
        properties = PhotonNetwork.CurrentRoom.CustomProperties;
        properties.Clear();
    }
    else
    {
        properties = new ExitGames.Client.Photon.Hashtable();
    }

    properties.Add("roomState", json);

    try
    {
        if (!RoomStateFull) PhotonNetwork.CurrentRoom?.SetCustomProperties(properties);

    }
    catch (NotSupportedException)
    {
        Debug.Log("RoomState too long, canceling last operation");
        //roomState = lastRoomState;
        RoomStateFull = true;
        DeleteRoomContent();
    }

}
```

Ilustración 60 Método UpdateState, para guardar en Photon el estado de la sesión. Fuente: aplicación vista desde el editor

- Se tienen varios métodos que se encargan de añadir o editar el contenido de RoomState:

    - **SetState** indica si ahora mismo en la escena se está mostrando el mapa (modo mapa), o el mapa está escondido (modo objetos fullscreen).

    - El método **UpdateMapPosition** actualiza la posición del mapa en la escena (cuando un usuario sube o baja el mapa, lo hace para el resto de los usuarios también, para mantener la sincronización de los objetos sobre este, coincidir en donde apunta el puntero del usuario, etc.). La posición resultante se escribe en las propiedades del RoomState.

    - **UpdateCoordState** guarda las coordenadas actuales que se ven en el mapa, se llama cada vez que se cambian, ya sea moviéndonos a una localización o moviendo el mapa con los botones. Esto es importante porque es la manera que tienen los usuarios que entran tarde a una escena para saber las coordenadas actuales a las que apunta el mapa, que por defecto se enciende en las coordenadas cero. También se guarda el ancho y largo del mapa, por si este se modificara.

    - **SetNewAsset** guarda indicadores de los contenidos que se muestran en la escena en ese momento. No podemos serializar un objeto 3d, pero como todos los

usuarios tienen la misma aplicación, ya tienen los modelos guardados en a la guna parte, lo que guardamos es un índice de referencia para que sepan que objeto 3d corresponde a esta instancia (ese es el parámetro contentResourceId). Después de eso tenemos una variable booleana para saber si el objeto va sobre el mapa, o está en modo fullscreen, más unos vectores que indican la posición, la rotación y el tamaño del objeto. Estos tres últimos vectores empiezan valiendo cero o 1 en el caso del tamaño, la unidad más básica, pero cuando un usuario guarda la escena, actualiza los vectores de todos los objetos poniendo su posición, rotación y tamaño actual, para que cuando carguen la escena guardada, puedan ponerlos correctamente. Es cierto que si solo se actualizan cuando un usuario guarda, si es el caso de que un usuario entre tarde a la sala no verá los vectores correctos, pero esa información la saca gracias a Photon y los Photon View (hay objetos que instancia Photon en la sala, que cuando un usuario nuevo entra, también los instancia, y comparten posición, rotación y tamaño.

- El método **SetNewMarker** se encarga de guardar las líneas que dibuja mi compañero tanto en el aire como en el mapa, y también los stickers, para poder dibujarlas después de nuevo, al cargar una escena o entrar en la sala tarde. Este método crea un recurso en el momento que se empieza a crear la línea, guarda su id para identificarla en la escena, si está sobre el mapa o no, su color, grosor, su patrón... y se marca como línea actual para que luego, utilizando el método **UpdateMarker**, se cree una lista de puntos con coordenadas en la escena que indican los puntos de la línea, para poder redibujarla igual.

- Existen también los métodos **DeleteAllAssets** y **DeleteAsset** para borrar assets del RoomState.

He de añadir que todos los métodos llaman siempre al UpdateState, para que este actualice el estado de la “pizarra”.

- Otro método importante es el método **SaveScene** [Ilustración 61]. Este método guarda la escena en disco. Lo que hace es igual que con las propiedades de Photon, convertir la clase RoomState a un Json y escribirlo en un fichero de texto que se guarda en disco. El fichero se guarda en con una dirección especial, llamada PersistentDataPath, que es la única carpeta que se puede editar y acceder dentro del disco de las gafas desde la

aplicación. Ahí se guarda en una carpeta llamada bookmarks, con el día y la hora como nombre.

```
private void SaveScene()
{
    Debug.Log("Saving scene");
    SetPositions();
    string json = JsonUtility.ToJson(roomState);
    Debug.Log("actual scene: " + json);

    ApplicationController.GetInstance().multiplayerController.SetProperties(json, "currentRoom");
    DateTime time = DateTime.UtcNow;

    string bookmaksDir = Application.persistentDataPath + "/bookmarks";

    if (!Directory.Exists(bookmaksDir))
    {
        Directory.CreateDirectory(bookmaksDir);
    }

    bookmaksDir += "/" + ApplicationController.GetInstance().lobbyConnectionController.GetRoomName();

    if (!Directory.Exists(bookmaksDir))
    {
        Directory.CreateDirectory(bookmaksDir);
    }

    string path = bookmaksDir + "/capture-" + time.ToString("dd-MM-yy-HH-mm");

    try
    {
        if (!File.Exists(path))
        {
            File.WriteAllText(path, json);
        }
    }
    catch (Exception e)
    {
        Debug.Log("Exception: " + e.Message);
    }
}
```

Ilustración 61 Método SaveScene. Fuente: aplicación vista desde el editor

- Por último, se encuentra entre los métodos más importantes el **SetCurrentRoomState**, el método responsable de generar la escena a imagen de un roomState guardado en disco, o el que se encuentra escrito en la "pizarra" al entrar en la sala de Pothon [Ilustración 62]. Este método recibe un RoomState, ya sea leyendo del fichero en disco, o en las propiedades de la sala, sobrescribe su actual RoomState, lo recorre, y aplica los cambios que se describen en él: enciende o apaga el mapa, pone las coordenadas que dice, e instancia los objetos 3d y las líneas donde corresponde (esto último ayudándose del método **ReconstructMark**).

```
private void SetCurrentRoomState()
{
    Debug.Log("smap state e " + roomState.roomProperties.state);
    switch (roomState.roomProperties.state)
    {
        //map is visible
        case RoomState.State.MAP:
            ApplicationController.GetInstance().map.transform.parent.position = roomState.roomProperties.MapParentPos;
            ApplicationController.GetInstance().map.transform.position = roomState.roomProperties.MapPos;

            ApplicationController.GetInstance().mapController.ApplyHideMap(false);
            ApplicationController.GetInstance().map.GetComponent<OnlineMaps>().SetPositionAndZoom(roomState.roomProperties.currentCoordinatesZoom.lon, roomSta
            break;
        //a fullscreen model is shown
        case RoomState.State.FULLSCREEN:
            ApplicationController.GetInstance().mapController.ApplyHideMap(true);
            break;
    }

    foreach (RoomState.RoomContent content in roomState.roomContent)
    {
        if (content is RoomState.ContentObject)
        {
            RoomState.ContentObject ca = content as RoomState.ContentObject;

            if (fromSavedScene)
            {
                ApplicationController.GetInstance().objectsController.CallToInstantiateObject(ca.contentResourceId, ca.isOnMap, ca.viewId);
                //ApplicationController.GetInstance().mainMapController.SetTransformFromRoomState(ca);
            }
            else
            {
                ApplicationController.GetInstance().objectsController.SetObject(ca.contentResourceId, ca.viewId, ca.isOnMap);
            }
        }

        if (content is RoomState.ContentMark)
        {
            ReconstructMark(content as RoomState.ContentMark);
        }
    }
}
```

Ilustración 62 Método SetCurrentRoomState. Se ve que lo que hace es leer el RoomState y llamar a todos los procedimientos de uno en uno. Fuente: aplicación vista desde el editor

6.5. Quinto sprint, Corrección de errores, mejoras en las herramientas

En este sprint se desarrollaron algunos cambios en el funcionamiento de los objetos del mapa.

6.5.1. Cambios en el funcionamiento de los objetos

Como se mencionan en el sprint dos, cuando instanciamos un objeto en el mapa lo que hago es crear un objeto llamado Holder el cual contiene unos scripts que permiten manipular su posición, rotación y escala interactuando con el Collider, y además utilizo un script de plugin de OnlineMaps, el OnlinemapsMarker3DManager, que crea un objeto llamado marcador, básicamente un objeto 3d asociado a unas coordenadas del mapa.

Originalmente al crear un objeto se crea ese marcador en las coordenadas del centro del mapa, así, el objeto se muestra de acorde a las coordenadas que tiene [Ilustración 63]. Además, la aplicación permite interactuar con los objetos mediante las manos y cambiarles la posición, rotación, y escala, pudiendo situarlos donde se deseara [Ilustración 64], pero había un fallo.

Ilustración 63 Al instanciar un objeto en el mapa se instancia en las coordenadas del centro del mapa. Fuente: aplicación vista desde el editor

Ilustración 64 El objeto se puede mover, rotar y escalar como se quiera. Fuente: aplicación vista desde el editor

Mover el objeto en el espacio no cambiaba las coordenadas del marcador, lo que significaba que el objeto, aunque se pusiera en cualquier otro sitio, estaba anclado al centro del mapa, y si este desaparecía porque el usuario se desplazaba en alguna dirección, el objeto desaparece [Ilustración 65].

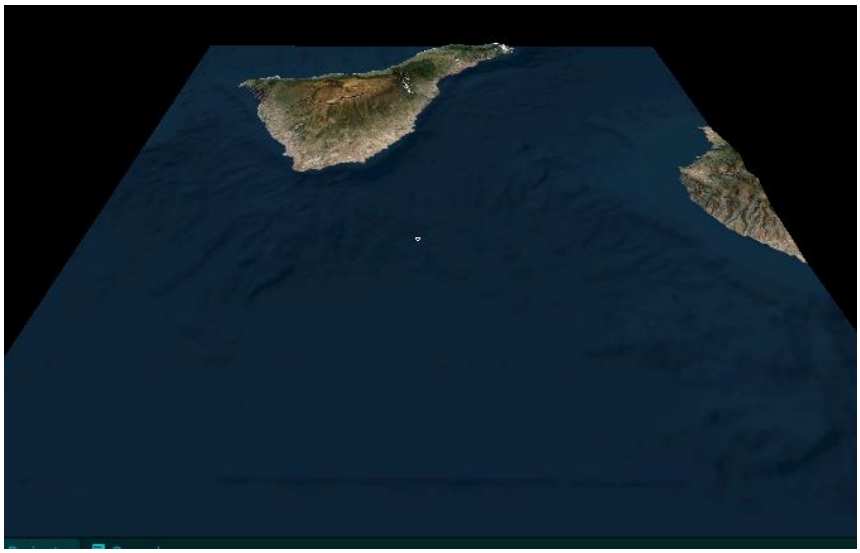

Ilustración 65 Si las coordenadas donde está anclado el objeto, en este caso el centro de Gran Canaria, dejan de estar a la vista, el objeto desaparece. Fuente: aplicación vista desde el editor

Este fallo lo he logrado arreglar unos meses después, al descubrir que la clase que permite manipular el Holder poseía un callback para avisar cuando se dejase de manipular. Esto me dio la idea de crear un método que reasigne las coordenadas al marcador, y luego reinicie la posición local del objeto, permitiendo así que el objeto se ponga donde se desee y siempre tenga las coordenadas a las que se ancla debajo de él[Ilustración 66].

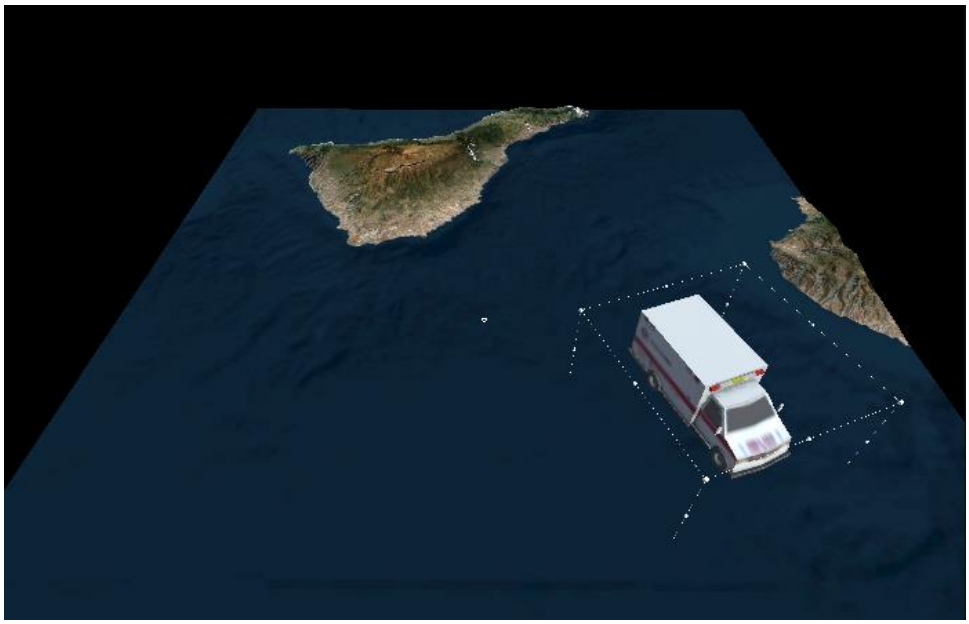

Ilustración 66 Ahora las coordenadas donde está anclado el objeto se actualizan a la posición de este, permitiendo que sea visible en todo momento. Fuente: aplicación vista desde el editor

Otra de las actualizaciones fue que, originalmente, los modelos se creaban con un tamaño en función del zoom que mostraba el mapa, pero se está mostrando una vista en la que se ve toda la ciudad, un vehículo podía verse del tamaño de 4 manzanas [Ilustración 67].

Ilustración 67 El camión media demasiado. Fuente: aplicación vista desde el editor

Así que añadí un cambio. Los modelos 3d ahora se instancian con un tamaño realista [Ilustración 68], y se les añade un script llamado **ObjectZoomChanger** el cual, junto a un prefab que he hecho de una chincheta, produce que cuando pongamos objetos en el mapa, y se tenga un zoom demasiado pequeño, se muestre en su lugar una chincheta indicando donde están.

Es un script muy simple, solamente consiste que en el método Update comprueba el nivel de zoom del mapa y cuando llegue a un umbral, pasa de mostrar el objeto 3d original a una chincheta [Ilustración 69].

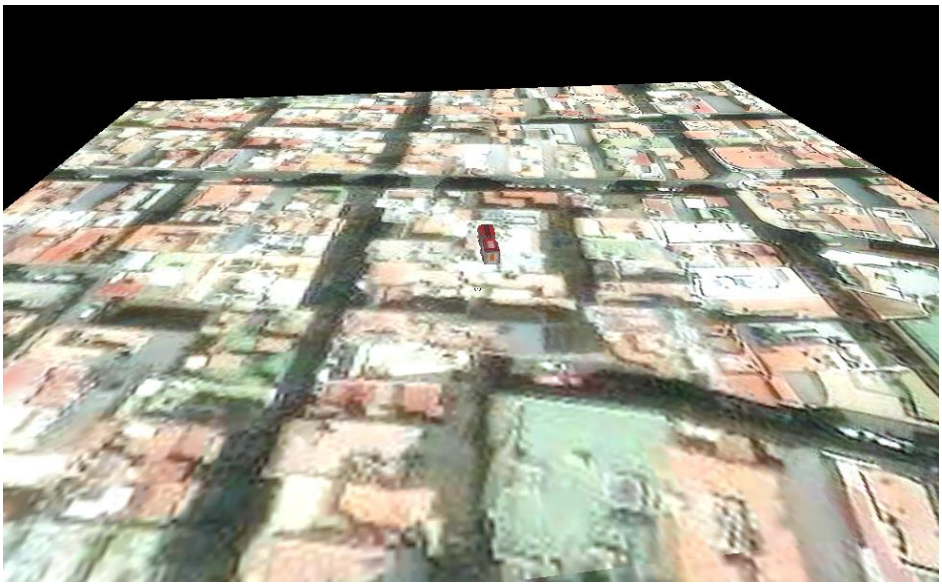

Ilustración 68 Ahora el camión tiene un tamaño decente. Fuente: aplicación vista desde el editor

Ilustración 69 Cuando nos alejamos se ve un icono. Fuente: aplicación vista desde el editor

6.5.2. Exclusividad de la sala

Hasta aquí la aplicación había dispuesto de una única sala de Photon con todo el contenido por lo que hice los ajustes para mostrar distintas salas con contenido exclusivo, como mencionamos que la aplicación haría.

Una vez hechos los cambios, al entrar en la aplicación el usuario puede elegir entrar en salas distintas, que tienen distinto tema, como por ejemplo una para gestión de emergencias y otra para gestión de operaciones militares. Con esas dos opciones se causa que los usuarios entren a salas de Photon distintas, para que los que entren a la de emergencias se vean entre sí, pero no vean a los que entren en el ejército. Luego a su vez, junto con esa separación, se puede

separar todo el contenido para cada sala, por ejemplo, poner unas localizaciones concretas para una sala y otras para la otra sala, y poner contenido exclusivo, como que el contenido militar solo se puede usar en la sala militar.

En el caso de las localizaciones, para las predeterminadas se tiene un diccionario que indica a que sala pertenecen [Ilustración 70], y, además, cuando se guardan localizaciones personalizadas estando dentro de la aplicación, estas se escriben en un fichero en disco exclusivo para cada sala, así, cuando se vuelve a entrar a la sala, se lee de la carpeta correspondiente las localizaciones que se han guardado, para mostrarlas; y lo mismo pasa con los bookmarks. Cuando se quiere guardar el estado de la sala, este se escribe en un fichero en disco en la carpeta de la sala, así esta escena solo se recupera si se ha entrado a la sala correspondiente, si no es el caso, es como si no existiera [Ilustración 71].

```
roomLocations = new Dictionary<string, string>();
roomLocations.Add("Canary Islands", "any");
roomLocations.Add("La Palma", "any");
roomLocations.Add("El Hierro", "any");
roomLocations.Add("La Gomera", "any");
roomLocations.Add("Tenerife", "any");
roomLocations.Add("Gran Canaria", "any");
roomLocations.Add("Fuerteventura", "any");
roomLocations.Add("Lanzarote", "any");
roomLocations.Add("La Graciosa", "any");
roomLocations.Add("Las Palmas", "ArmyRoom");
roomLocations.Add("rotonda", "EmergencyRoom");
```

Ilustración 70 Diccionario que indica la exclusividad de una localización. Fuente: aplicación vista desde el editor

```
private void SetLocationsFromDisk(string roomName)
{
    string LocationsFile = Application.persistentDataPath + "/locations/" + roomName;

    if (!File.Exists(LocationsFile)) return;

    string[] locations = File.ReadAllLines(LocationsFile);

    foreach (string location in locations)
    {

        string[] locationDivided = location.Split(' ');
        currentRoomLocations.Add(locationDivided[0], new Coordinates(double.Parse(locationDivided[1]),
            double.Parse(locationDivided[2]), int.Parse(locationDivided[3])));
    }
}
```

Ilustración 71 Método para cargar las localizaciones de una sala concreta. Fuente: aplicación vista desde el editor

El método del diccionario de las localizaciones también lo se usa para los objetos 3d, así se indica si un objeto puede usarse en todas las salas, o una en concreto.

6.6. Sexto sprint, fusión final, solución de problemas

A final del sprint, como en los anteriores, se fusiona el trabajo mío y de mi compañero, y se estuvo probando la aplicación, conectándose en multiusuario y usando el editor de Unity, y continuar dedicándose a corregir problemas que aparecían a la hora de usar la aplicación en las gafas. Para ello se utiliza el prefab de **DebugWindow**.

6.6.1. DebugWindow

A la hora de generar una versión de la aplicación para las HoloLens, muchas veces se daban errores que solo aparecían en las gafas, o errores producidos por algún bug, y en las gafas no había forma de saber qué estaba ocurriendo. Para poder trabajar bien se utiliza un prefab que se encuentra en el plugin de MRTK que llamado **DebugWindow** un prefab que consistía de un canvas con los scripts de BoundsControll, SolverHandler y FollowMe, para poder manipularlo, moverlo y que mantuviese una posición de acuerdo con la posición de la cabeza del usuario, y un script llamado DebugWindow que mostraba los mensajes de la consola en el canvas, en el cual también uno se puede desplazar deslizando arriba o abajo para ver el total de mensajes de la consola [Ilustración 72].

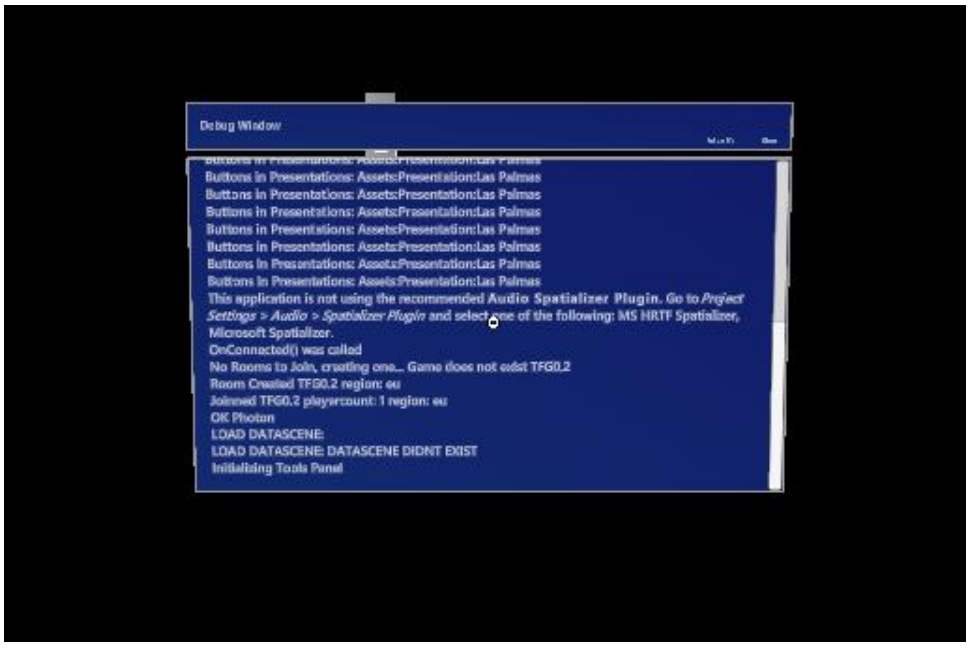

Ilustración 72 DebugWindow. Fuente: aplicación vista desde el editor

Por desgracia, al usar esta ventana en las HoloLens, se mostraba continuamente un mensaje que parecía estar en un bucle que repetía todo el rato un mensaje indicando que no se detectaban las manos, y al imprimir el mensaje ese continuamente en la pantalla hacía que la aplicación fuera muy lenta, hasta el punto de colgarse. Finalmente se arregló esto si en el script de la ventana especificamos el tipo de mensajes que queremos ver, como mostrar solo errores.

6.6.2. TransferOwnership

Como se menciona en el apartado de desarrollo, Los objetos 3d que se instancian en el mapa se instancian dentro de un objeto que llamamos **Holder**, el cual tiene un **PhotonView** y un **PhotonTransformView**, y esto sirve para que en una sala multiusuario de Photon, todos los usuarios vean el mismo objeto en el mismo sitio, y cuando este se mueva, se mueva de la misma manera para todos los usuarios. Pero a la hora de usarlo en la aplicación, había una restricción que afectaba, y es que un objeto con PhotonView y TransformView, para que se transmita y sincronice su posición, debe moverlo el usuario que lo haya creado. Esto causaba que cuando alguien que no fuera el máster de la sala (el que haya creado el objeto) lo movía localmente, es decir, solo en su aplicación, no se sincronizaba, pero si era el máster el que lo mueve, ese movimiento se sincroniza para todos los usuarios.

Para poder arreglar ese asunto lo recurrí a un método de PhotonView que es el TransferOwnership. Este método hace lo que dice, transfiere la propiedad del objeto con el PhotonView al usuario que le indiquemos. Este método se llama cada vez que un usuario pulsa sobre el objeto para intentar moverlo. Al pulsarlo, se le transfiere la propiedad del objeto, y, como ahora es suyo, si lo mueve sincronizará el movimiento a los demás usuarios [Ilustración 73]. Luego, si otro usuario intenta mover el objeto, se le pasará la propiedad a él, y si varios usuarios lo tocan a la vez, la propiedad será del último que lo haya hecho, así que el objeto irá donde diga este.

```
0 referencias
public void OnPointerDown(MixedRealityPointerEventData eventData)
{
    TransferOwnership();
}
```

```
2 referencias
public void TransferOwnership()
{
    if (GetComponent<PhotonView>().Owner != PhotonNetwork.LocalPlayer)
    {
        GetComponent<PhotonView>().TransferOwnership(PhotonNetwork.LocalPlayer);
    }
}
```

Ilustración 73 Método TransferOwnership. Fuente: aplicación vista desde el editor

Este asunto de transferir propiedades obliga a cambiar otra cosa, cómo borrar los recursos. Originalmente si se quería borrar los objetos se le indicaba al master, ya que era el propietario de todos los objetos, pero ahora, si se están tocando, acaban con distintos propietarios y Photon solo permite borrar un objeto al propietario de este. Así que lo que hice fue mandar un RPC a todos los usuarios para que borrasen todos los objetos, así, como solo pueden borrar los que tienen en propiedad, cada uno borrará los suyos, y al final se borran todos (cuando un usuario borra su objeto de Photon, lo borra de la sala, se lo borra a todo el mundo) [Ilustración 74].

```
public void ApplyDeleteAssets()
{
    foreach(GameObject item in ApplicationController.GetInstance().photonInstantiatedGameObjects)
    {
        if (item.GetComponent<PhotonView>().Owner.Equals(PhotonNetwork.LocalPlayer))
        {
            PhotonNetwork.Destroy(item);
        }
    }
    ApplicationController.GetInstance().photonInstantiatedGameObjects.Clear();
}
```

Ilustración 74 Método DeleteAssets. Fuente: aplicación vista desde el edito

6.7. Diseño Final

Al final entre mi compañero y yo se decidió documentar un nuevo diagrama de clases para mostrar el aspecto final del proyecto [Ilustración 75].

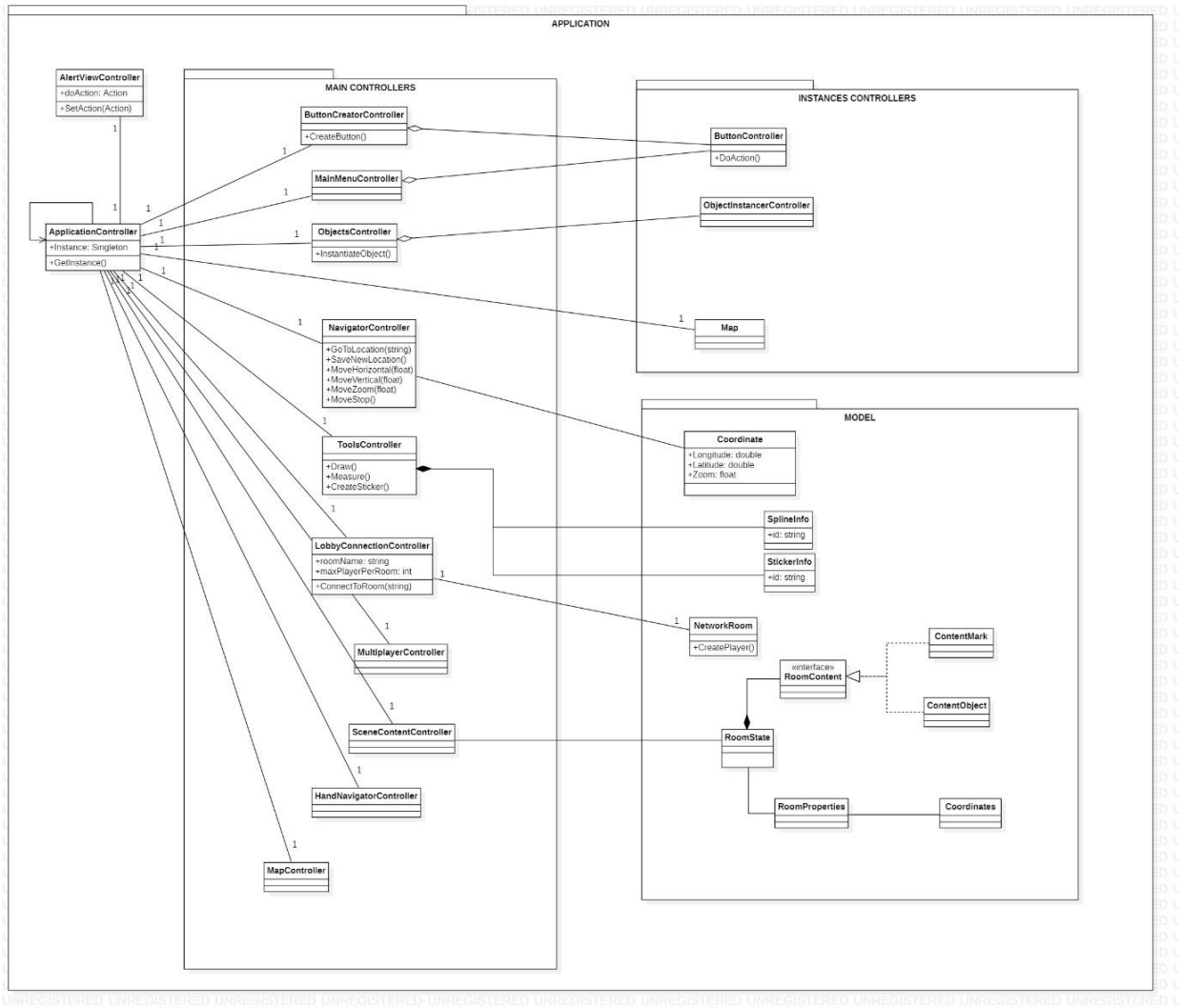

Ilustración 75 Diagrama de clases final

7. INTEGRACIÓN DE LA VERSIÓN COMPLETA

La aplicación que se ha desarrollado es la base de una aplicación que se está desarrollando en la empresa en la que trabajamos. Esta aplicación, de nombre XRSandbox, además de hacer lo que hace nuestra aplicación, permite además conectarse a un portal web desarrollado por la empresa también, para poder añadir contenido, crear salas, invitar a usuarios, y gestionar la aplicación de forma mucho más dinámica.

Mi compañero y yo estuvimos desde el comienzo de la aplicación, y se ha extraído parte de lo que hemos hecho en esa aplicación para el TFG, crear el mapa añadirle contenido, dibujar... todo lo hacemos en la aplicación del TFG. Pero en el trabajo hemos desarrollado más cosas para la aplicación que no se han incluido en el TFG porque dependían del portal web, que eso es algo en lo que no hemos tomado parte, lo han hecho otros trabajadores de la empresa, por lo tanto, no se quería incluir en el TFG por falta de conocimiento y porque el trabajo no es propio.

En lo que si hemos trabajado es que los desarrolladores del portal también crearon un api para poder comunicarnos con ella, y hemos trabajado en esa comunicación.

La aplicación del trabajo empieza con un menú con unos campos para poner usuario y contraseña. Del portal se obtiene la información del usuario que ha iniciado sesión, y una lista de salas a las que tiene acceso. Estas salas sirven para separar el contenido que mostramos y permitir que varios usuarios se conecten a la vez a la misma aplicación, pero en distintas salas, por si desean tratar con contenidos diferentes o usuarios diferentes.

Una vez el usuario elige una sala, se le muestra todo el contenido que esté asociado en el portal a esa sala.

7.1. Contenido remoto

En la aplicación final se tienen varios tipos de contenido:

- Las **Localizaciones** se pueden crear en el portal pinchando en un pequeño mapa que mostramos para registrar las coordenadas, y luego en las gafas se crea un botón por cada entrada de localización que haya en la sala, que si se pulsa lleve el mapa a esas coordenadas registradas.

- En el portal el usuario puede introducir las direcciones a unos MapServers (capa de datos que se puede plasmar sobre una aplicación de visualización de mapas) para poder mostrarlas en el mapa de la aplicación. A ese recurso se le llama capa o **Layer**.

- Se tiene un recurso llamado **WebPanel** en el cual un usuario en el portal introduce una dirección de una página web y en la aplicación se crea una pantalla interactiva que muestra esa página web. Al igual que WebPanels se tiene otro recurso que sirve para mostrar imágenes o vídeos que se suban al portal.

- El usuario puede indicar en el portal unos **Marcadores de mapa**, unas chinchetas con texto en las coordenadas que deseen.

- Los **objetos 3D** son modelos fbx (Filmbox) que los usuarios suben al portal. Originalmente con los objetos era que básicamente el usuario subía un fichero con extensión fbx a la web, el api de la web almacena el fbx en una base de datos, y cuando la aplicación pregunta por este recurso, el api le devuelve una dirección web para descargar el fbx.

- Contenido de **Nube de puntos**. Un usuario podía incluir un fichero LAS (un fichero de nube de puntos) para poder visualizar esa nube en la aplicación.

El método que se usa para el contenido de objetos 3D se tuvo que actualizar porque en la aplicación usuarios pedían utilizar unos modelos demasiado pesados para almacenarlos en las HoloLens (modelos de más de un gigabyte), e intentar descargar uno o varios de esos modelos hacían que la aplicación se colgara y cerrase automáticamente, porque no podían descargarlo. Para solventar ese problema, se utiliza una tecnología de Unity que se llama sistema de addresables. El sistema de addresables (direcciones), permite que un usuario utilice un asset de Unity mediante la dirección en la que se encuentra almacenado, pudiendo ser esta en remoto. Esto significa que se puede guardar un asset de un objeto 3D en un ordenador que no tiene problema en almacenar modelos de un gigabyte o más, y que las gafas lo puedan utilizar solo teniendo la dirección en la que se guarda. Gracias a esto las gafas no almacenan esos modelos por lo que pueden trabajar con ellos sin problemas. Cabe decir que otra ventaja frente a utilizar los ficheros fbx directamente es que los addresables cargan los assets de forma asíncrona, por lo que no retienen el hilo principal de la aplicación, permitiendo hacer otras funciones mientras se cargan.

El contenido de objetos 3D también se dividía a su vez en varios tipos distintos:

- Modelo 3d. Este recurso consiste en un objeto 3D el cual se puede crear en cualquier parte del mapa, y desplazarlo, rotarlo, o escalarlo con las manos.

- Modelo 3d posicionado. Esta versión es un objeto 3D, pero con unas coordenadas asociadas las cuales se introducen en el portal. Este modelo cuando se instancia traslada el mapa a la posición donde se encuentra y su posición o tamaño no puede ser manipulada por el usuario en la aplicación.

- Contenido GPS. A este contenido se le asocia el modelo 3D y un id el cual pertenece a una aplicación móvil desarrollada por la empresa que transmite la posición GPS del móvil. Así el contenido GPS con el id de una aplicación móvil muestra en el mapa la posición de ese teléfono en tiempo real. Esto es una gran función ya que permite que a la hora de gestionar emergencias o de utilizar la aplicación, se pueda ver en tiempo real las posiciones de los vehículos y unidades para poder organizarlas y reubicarlas.

Además, aparte del contenido para mostrar, se puede añadir y crear salas distintas con sus contenidos exclusivos. En este trabajo separamos el contenido en dos salas de ejemplo, creadas directamente, así cuando un usuario entra a una sala, no verá a usuarios que entren en la otra,

y tendrá unos contenidos que no se podrán ver en la otra sala. La versión de la empresa te permite crear las salas que quieras en el portal, y añadirles contenido exclusivo. Incluso se puede añadir y controlar los usuarios que tienen acceso a esas salas: puede haber un usuario que pueda acceder a cinco salas distintas, y tratar en ellas con usuarios diferentes que solo tengan accesos a una de ellas, con el contenido exclusivo de esa sala.

7.2. Desarrollo Futuro

Para el futuro, junto con la empresa, se plantea añadir muchas más funciones a la aplicación para que sea mucho más completa todavía. Hace unos meses se hizo unas pruebas con un dron de otra empresa, el cual transmitía su posición en tiempo real. Para ello estoy en proceso de investigar en una forma de hacer streaming del vídeo del dron dentro de la aplicación, e incluso utilizar una api propia del dron para poder levantarlo desde las gafas, con el gesto de la mano. A esto aún le falta tiempo, pero cuando esté completo permitirá dar órdenes de reubicación a unidades sobre el terreno, ya manipuladas por humanos, o robots, y ver en tiempo real lo que hacen.

La aplicación también está pensada para usarse sobre el terreno, por lo que también se espera añadir funciones que aporten información al usuario que use las gafas in situ, como ver las líneas dibujadas representadas en el mundo real (participé en una demo para desarrollar esto que implicó utilizar las gafas en un barco navegando).

Otra función en la que también estoy trabajando es en la visualización de nubes de puntos dentro de las gafas. Si el cliente poseía nubes de puntos escaneadas de las regiones que desea mostrar, ofrecía una forma más detallada de representar que el mapa. Esto también requiere seguir investigando y trabajando: representar punto de manera tridimensional en VR, usando las dos pantallas de los ojos; modificar la forma de renderizado para reducir el coste computacional a realizar en las gafas, etc.

Por otro lado, se investiga también el uso de una herramienta externa para crear avatares personales de los usuarios, para poder representarlos de una forma más realista, y mejorar el concepto de reunión en un entorno virtual.

8. CONCLUSIÓN

Trabajar en este proyecto me ha permitido aprender más sobre el motor Unity y sobre la realidad virtual y mixta, unos campos que siempre me habían interesado. Gracias al trabajo en la empresa y a este proyecto de fin de grado he tenido una excusa más para poder dedicar tiempo a algo que me interesa y en lo que me gusta trabajar.

El estar trabajando a jornada completa y realizar el proyecto de fin de grado es bastante difícil, te obliga a buscar hueco por las tardes y a pasar más tiempo de lo normal en el ordenador, pero realmente gracias al trabajo en la empresa es como he aprendido las cosas necesarias para desarrollar el proyecto, y viceversa.

Desarrollar ideas, encontrarse problemas y buscar soluciones creativas a ellos a veces puede ser muy tedioso, pero otras veces es entretenido y, aunque pases horas y horas parado en el mismo problema, todos los documentos que ojees, sitios web que visites, foros en los que preguntes, aunque no te resuelvan ese problema en el que te encuentras en concreto, servirán como aprendizaje.

Y trabajar en el desarrollo de la aplicación con la empresa me ha aportado muchas experiencias enriquecedoras además de aprender Unity: he podido viajar a varios sitios para asistir a reuniones con potenciales clientes, participar en muchas demos en muchos sitios distintos (una de estas demos implicó pasar 24 horas en el mar), trabar en conjunto con otras empresas, y tener muchas anécdotas que contar.

Dejando de lado mi desarrollo personal, este proyecto culmina en una aplicación que posee una utilidad real y una manera innovadora de trabajar. Con el aporte significativo y novedoso que implica la realidad mixta, la aplicación que desarrollamos, con más tiempo y mejoras, aportará una nueva manera de gestión de recursos para las empresas de logística y más todavía para los organismos que realicen operaciones en un terreno. Ya no tendrán que reunirse todos los participantes junto a una misma mesa, sino que podrán reunirse desde cualquier parte del mundo para ver y manipular los recursos de una manera mucho más cómoda y realista que arrastrando con el ratón sobre un plano en una pantalla.

No solo en este ámbito, la realidad mixta seguirá introduciéndose en el día a día en muchos lugares, y formar parte en esa integración es de los más interesante.

9. REFERENCIAS

- Qué es la realidad Mixta: https://learn.microsoft.com/es-es/windows/mixed-reality/discover/mixed-reality
- Unity: https://unity.com/es
- Manual de Unity: https://docs.unity3d.com/Manual/index.html
- Photon: https://www.photonengine.com/pun
- Web HoloLens: https://www.microsoft.com/en-us/hololens
- Mixed Reality Toolkit: https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05
- Online Maps: https://infinity-code.com/assets/online-maps
- Visual Studio: https://visualstudio.microsoft.com/es/
- Trello: https://trello.com/es
- Git: https://gitforwindows.org