Universidad Politécnica de Cartagena

ETS Ingeniería de Telecomunicación

Universidad Politécnica de Cartagena

USO DE OCLUSIÓN EN EL DESARROLLO DE UNA APLICACIÓN DE REALIDAD AUMENTADA

Jose Cavas Alcaraz

Dirigido por:

Fernando Losilla López
María Francisca Rosique Contreras

2 de diciembre de 2020

A mis padres por hacer de mi lucha la suya propia.

A Marina por ser el punto de apoyo desde el que mover cualquier mundo.

"Sin duda hay que perderse para hallar destinos inalcanzables

o de lo contrario todo el mundo sabría dónde están."

Índice general

Índice de tablas

Índice de ilustraciones

Resumen

El proyecto que se presenta a continuación tiene como objetivo el desarrollo de una aplicación de Realidad Aumentada haciendo uso de la oclusión del mundo virtual con respecto al mundo real, siendo este uno de los retos actuales a los que se enfrenta esta tecnología entre otros. Con este fin se van a utilizar herramientas de desarrollo las cuales no sólo nos van a permitir utilizar la oclusión, sino también crear una aplicación orientada al uso de niños teniendo la vista puesta en la enseñanza multisensorial para la estimulación temprana de estos. Para fortalecer este propósito se aplicará la técnica de sonido 3D en la aplicación y así podremos crear una búsqueda de objetos mediante el oído.

En la aplicación se podrá elegir entre dos modalidades entre las que varían los objetos que se presentarán en escena. Se va a necesitar de la colaboración de un supervisor para esconder los objetos en el espacio real y, cuando su tarea haya concluido, deberá dejarle el dispositivo móvil al niño para que comience su búsqueda.

Para el desarrollo se van a utilizar diferentes tecnologías como el motor de videojuegos Unity, el entorno de desarrollo Visual Studio 2019, las herramientas proporcionadas por AR Foundation y un dispositivo móvil Android para las pruebas.

Palabras clave: aplicación, Realidad Aumentada, Unity, multisensorial, oclusión, sonido 3D.

Abstract

The project presented below aims at developing an Augmented Reality application making use of occlusion between virtual and real objects. This is one of the current challenges being faced by this technology, among others. To do this, several development tools will be used not only to allow us to use occlusion but also to create an application oriented to the use of children with an eye on multisensory education for their early stimulation. To reinforce this purpose, the 3D sound technique will be applied in the app so you can create an object search through your ear.

In the application you can choose between two modes where the objects, that will be presented in the scene, vary. Collaboration from supervisors will be necessary to hide objects in real space, and when their task is completed, they will have to give the child the mobile device to begin their search.

Different technologies such as the Unity game engine, the Visual Studio 2019 development environment, the tools provided by AR Foundation, and an Android mobile device for testing will be used for development.

Keywords: application, Augmented Reality, Unity, multisensory, occlusion, 3D sound.

Capítulo 1: Introducción.

1.1. Conceptos generales.

Para poder hablar del desarrollo de una aplicación de Realidad Aumentada (AR, por sus siglas en inglés) debemos conocer una serie de conceptos que tienden a confundirse habitualmente[1]. La Realidad Extendida (*Extended Reality*, XR) abarca distintas tecnologías que se diferencian por la forma de fusionarse o interactuar entre el mundo real y el mundo virtual[2]. A continuación, vamos a explicar las tecnologías que engloba la XR:

1.1.1. Realidad Virtual

La Realidad Virtual (*Virtual Reality*, VR) es el término que se utiliza de forma más común para referir a aquel contenido que puede reproducirse mediante dispositivos tales como smartphones o gafas de realidad virtual. De esta forma, el usuario queda "inmerso" en una realidad virtual en la que, según la aplicación que se le haya dado a esta tecnología o si se ha incluido uno o varios controladores, podrá interactuar libremente. Esta realidad virtual se denomina realidad virtual interactiva[3].

Otra forma de definir o sintetizar este concepto sería denominar esta realidad como un entorno 3D multisensorial que imita al mundo real y es generado a través de un ordenador[4]. Podemos enumerar sus características principales como las siguientes:

- Interactivo: mediante controladores o movimiento se podrá interactuar con el entorno virtual.

- Inmersivo: se debe creer que se está en el mundo virtual, enganchándonos a la experiencia y olvidándonos del mundo real.

- Multisensorial: activa los sentidos más allá de la vista.

- Sintético: emula un entorno virtual completo. Existen opiniones encontradas sobre si un vídeo 360º se puede considerar VR.

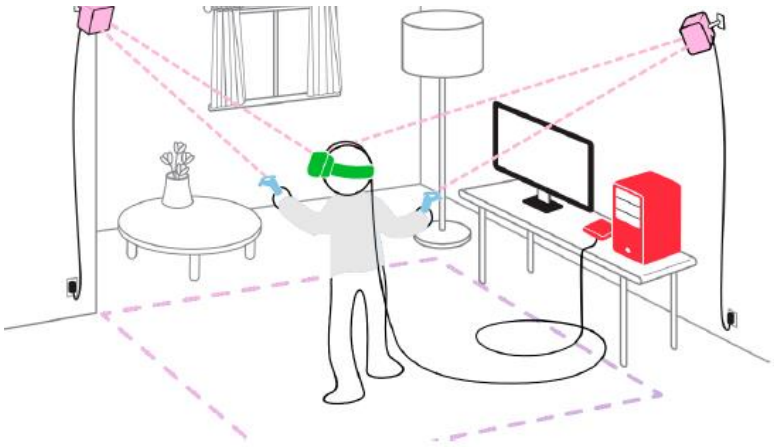

Ilustración 1. Elementos básicos de un sistema de realidad virtual: tracking (en rosa, visualización (en verde), interacción (en azul) y entorno virtual 3D generado por ordenador (en rojo). (Fuente: página web de HTC Vive).

1.1.2. Realidad Aumentada

La Realidad Aumentada (*Augmented Reality*, AR) incorpora contenido virtual al mundo real[5]. Por mundo real también se concibe un vídeo grabado en el propio mundo real.

Podemos clasificar la AR de dos formas distintas: por los dispositivos utilizados para su uso o por si se utilizan marcadores o no[5]. En el caso de los distintos dispositivos electrónicos que podemos utilizar, la AR se divide en:

- Video Pass-through.
- Optical see-through.
- AR móvil.
- HMD.
- Basadas en proyección.

Y, por otro lado, podemos clasificar esta tecnología según si se utilizan marcadores o no (*marker* o *markerless*).

Más adelante profundizaremos en esta tecnología, ya que es nuestro objetivo de desarrollo.

Ilustración 2. Ejemplo de aplicación de AR. Pokemon Go (Niantic).

1.1.3. Realidad Mixta

La Realidad Mixta (*Mixed Reality*, MR) es el concepto más abstracto, ya que más que una tecnología en sí misma, este concepto refiere a la unión o combinación de las dos tecnologías definidas anteriormente, VR y AR[6]. Entre sus definiciones más usadas se encuentra la del uso del continuo realidad-virtualidad, en el que la interacción entre el mundo completamente real y el mundo completamente virtual es constante y uno condiciona al otro. La idea es la de generar un modelo 3D de la realidad y superponer elementos virtuales en él, de forma que se consiga una interacción completa[7].

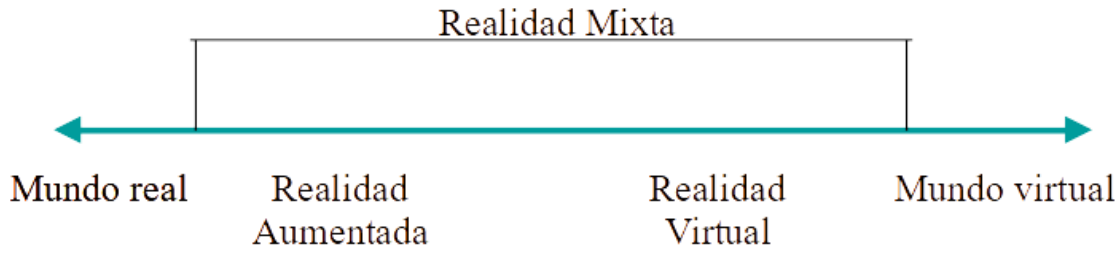

Ilustración 3. Continuo realidad-virtualidad.

Las aplicaciones potenciales de la MR son muy variadas y alimentan a gran parte de la industria, algunos ejemplos son:

- Sustitución de las pantallas fijas por pantallas virtuales.
- Prototipado.
- Formación técnica on-site.

1.2. Objetivo del proyecto.

En los últimos años, el avance de las TIC (Tecnologías de la Información y la Comunicación) ha dejado patente el carácter disruptivo de algunas tecnologías. Éste es el caso de la Realidad Aumentada, la cual ha irrumpido en numerosos ámbitos de la sociedad[8][9]. Al igual que muchos avances tecnológicos, éste también cuenta con una variedad de retos a los que enfrentarse. Aunque más adelante explicaremos cada uno de ellos en más detalle, nuestro proyecto se centra en la oclusión del mundo virtual con el real.

Actualmente el reto de la oclusión se encuentra en desarrollo, incluso las principales plataformas de desarrollo AR han implementado recientemente las herramientas para poder hacer que el mundo real oculte al mundo virtual. En este proyecto vamos a poner a prueba estas herramientas e intentar aplicar la oclusión de elementos virtuales a una aplicación para dotarla de mayor realismo. Mediante la ocultación de elementos virtuales y con el añadido de un efecto de sonido en 3 dimensiones desarrollaremos una aplicación infantil que implique una enseñanza multisensorial para la estimulación temprana de los niños[10].

Capítulo 2: Estado del arte y tecnologías utilizadas.

En este capítulo vamos a ver qué se entiende hasta la actualidad por AR y qué conocemos de esta tecnología, además vamos a definir las herramientas que se van a utilizar en el desarrollo de la aplicación.

2.1. Realidad Aumentada.

La Realidad Aumentada, tal y como comentamos en el apartado *1.1.2. Realidad Aumentada*, es la tecnología que introduce objetos virtuales en el mundo real.

2.1.1. Tipos de Realidad Aumentada.

Su clasificación según el dispositivo electrónico que utilicemos es la siguiente:

● Video pass-through. La cámara de nuestro dispositivo electrónico graba el mundo real y la imagen se aumenta, es decir, se añaden los elementos virtuales[5][11]. Por ejemplo, las gafas Lynx R1 utilizan este tipo de tecnología.

Ilustración 4.Lynx R1 como ejemplo de video pass-through. (Fuente: Página web de Lynx-r).

● Optical see-through. En este tipo de AR el mundo real atraviesa la lente como lo harían unas gafas normales y los objetos virtuales que se posicionan en el mundo real se reflejan hacia el ojo. Ésto hace que este tipo de dispositivo sea más adecuado para la manipulación de objetos, movimientos y desplazamientos del usuario en su entorno real[12]. Ejemplos de este tipo las Hololens de Microsoft o las Magic Leap One entre otros dispositivos.

Ilustración 5. Microsoft HoloLens como ejemplo de optical see-through. (Fuente: Página web de Microsoft).

● AR móvil. La Realidad Aumentada en dispositivos móviles es la que se utiliza en móviles o tablets de forma que a través de sus cámaras podamos capturar el mundo real y mediante el mismo dispositivo como controlador podamos interactuar con los objetos virtuales. Un ejemplo sería el de la *Ilustración 2* referente al juego Pokemon Go o los también conocidos filtros de Instagram y su utilización del rastreo de caras (*fase tracking*).

● HMD. De sus siglas en inglés *Head Mounted Display*, esta tecnología podría englobar a las dos primeras ya que se refiere al concepto genérico de un dispositivo que se coloca en la cabeza.

● Basadas en proyección. Esta tecnología se basa en la proyección de luz artificial sobre superficies. En algunos casos también se permite la interacción del usuario a través de la diferenciación entre la proyección esperada y la cambiante que ha provocado el usuario.[13].

Ilustración 6. Tecnología HUD (Head Up Display) basada en proyección. (Fuente: portal web Xataka).

Y, por otro lado, podemos clasificar esta tecnología según su uso de marcadores:

● Basadas en marcadores (*marker*). En AR se pueden utilizar marcadores que son elementos que hacen que nuestro sistema de ejecute una acción o transmita una respuesta[9]. Un marcador puede ser cualquier imagen con las suficientes características (*features*) para que su detección sea sencilla. Un caso conocido sería la herramienta Vuforia[5]. Estas herramientas trabajan de la siguiente forma:

- Se introducen las imágenes a utilizar como marcadores en la base de datos de la herramienta.

- La imagen es procesada en busca de características.

- Se asocia un objeto virtual a cada imagen o marcador.

- La aplicación que utilicemos va a comparar las características de la imagen de la cámara con las de las imágenes de la base de datos.

● Sin marcadores (*markerless*). A diferencia de las tecnologías basadas en marcadores, ésta tiene la dificultad de reconocer las características de las imágenes en tres dimensiones del mundo real, además de poder reconocer planos. Para ello se utilizan algoritmos de ejecución en tiempo real que detectan las *features* de estas imágenes y determinan un punto de vista[9]. Además, También pueden basarse en geolocalización si el dispositivo cuenta con sistema de posicionamiento.

Ilustración 8. AR basada en marcadores. (Fuente: informaikta.blogspot.com).

Ilustración 7. AR sin marcadores. (Fuente: Página web intheloop).

2.1.2. Plataformas de Realidad Aumentada.

Existen tres plataformas principales en el diseño de aplicaciones de AR cuyas diferencias principales residen en los dispositivos destino de dichas aplicaciones[14]. Son las siguientes:

● ARCore. Se trata del conjunto de tecnologías de Google para el desarrollo sobre Android.

● ARKit. En este caso hablamos de las tecnologías de Apple cuyo software destino a iOS.

● AR Foundation. Es una tecnología multiplataforma de Unity (herramienta que definiremos más adelante). Es compatible con ARKit, ARCore, HoloLens y Maigc Leap. Con otros frameworks puede tener problemas de compatibilidad o ser compatible con retraso.

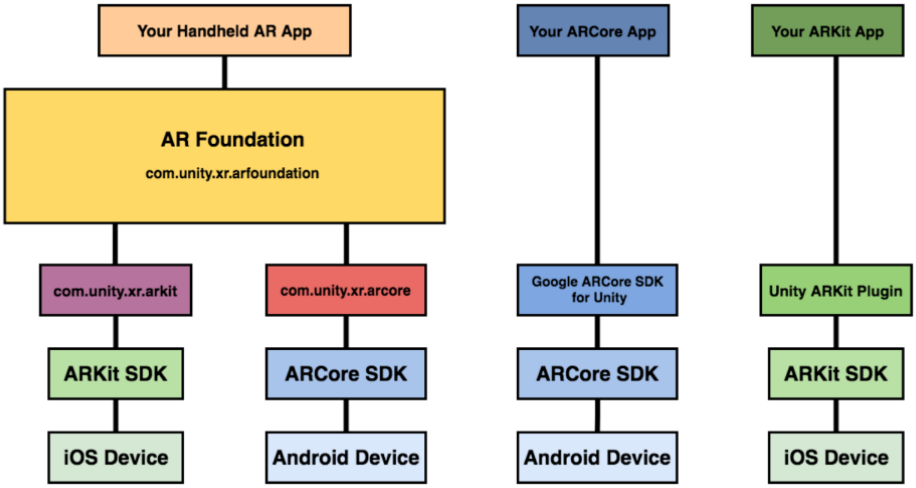

Ilustración 9. Resumen conceptual de tecnologías para AR. (Fuente: https://blogs.unity3d.com/es/2018/12/18/unitys-handheld-ar-ecosystem-ar-foundation-arcore-and-arkit/).

2.1.3. Retos de la Realidad Aumentada.

La Realidad Aumentada presenta todavía grandes retos a los que enfrentarse, algunos son funciones por implementar y otros, aspectos por perfeccionar[5]. A continuación, vamos a ver los más significativos:

● Detección de movimiento (*Simultaneous Localization and Mapping*, *SLAM*). Su objetivo es el de mapear el entorno y determinar su posición en el mismo en tiempo real. Para ello, esta tecnología se nutre de la Unidad de Medida Inercial (*Inertial Measurement Unit*, *IMU*) compuesta por acelerómetros, giroscopios y, a veces, magnetómetros para medir movimiento, rotación y orientación, y de la visión por ordenador (*Visual Inertial Odometry*, *VIO*) para determinar la orientación y posición mediante la IMU e imágenes del mundo real. Estas imágenes del mundo real ayudan a corregir los errores de estimación acumulables de la IMU.

● Reconocimiento del entorno. Engloba el SLAM, la detección de planos y de caras. Para esta acción se requiere de condiciones estrictas: una iluminación adecuada, planos estáticos con rasgos distintivos y movimientos lentos del usuario. En cuanto al rastreo de caras, se obtiene una malla (*mesh*) de la misma y una pose (posición + orientación) del centro del *mesh* para trabajar con ellas.

● Anclaje de elementos al mundo real (*anchors*). Sirven para posicionar un objeto virtual en el mundo real de forma que no se desplacen de donde se colocaron inicialmente, sino que guarden su posición.

● Estimación de luz. Para dar realismo a los elementos virtuales es conveniente que se encuentren bajo las mismas condiciones de luz que el mundo real. Se puede estimar la temperatura del color o la intensidad de luz mediante frameworks, incluso las últimas versiones pueden advertir la dirección de la luz y las sombras que provocarían en el objeto virtual.

● Experiencias multi-usuario. Para que varios usuarios puedan compartir la misma experiencia, las plataformas ARKit y ARCore comparten los *anchors*. En el caso de ARKit, utilizan ARWorldMap para compartir mapas locales entre los distintos dispositivos, y ARCore usa Cloud Anchors para compartir mediante la nube los *anchors* de cada objeto.

● Detección de profundidad. La detección de profundidad se puede realizar mediante escáneres de luz estructurada o cámaras ToF (*Time of Flight*), pero la apuesta por el futuro de esta característica se encuentra en las redes neuronales[15] sobre las que ya han trabajado ARCore con su ARCore Depth API capaz de crear un mapa de profundidad y ARKit con su People Occlussion que permite segmentar la imagen de una cámara y separar los píxeles pertenecientes a una persona.

● Oclusión de objetos virtuales. Hasta hace poco era complicado ocultar un objeto virtual detrás de uno real, ya que los procedimientos se consideraban rudimentarios o computacionalmente caros. Se necesitaría un mapa de profundidad el mundo real, tal y como se ha implementado en las versiones más recientes de ARCore.

2.2. Tecnologías utilizadas.

2.2.1. Unity Hub y Unity.

Unity Hub se utiliza como herramienta para administrar los distintos proyectos de Unity y sus versiones e instalaciones de componentes[16].

El motor de videojuegos de Unity se desarrolló por Unity Technologies en Dinamarca e integra un motor de renderizado personalizado junto con el motor de físicas Nvidia PhysX y la implementación de código abierto (*open source*) de las librerías .NET de Microsoft, Mono[17]. Unity posee una serie de características que lo hacen muy competente respecto a otros motores de simulación:

● Multiplataforma. Unity puede compilar en OSX, Windows o Web-player con algún complemento de navegador similar a Adobe Flash.

● Gratuito. A pesar de que existe versiones Unity Pro y Unity Enterprise de pago, la plataforma básica de Unity es gratuita para uso personal.

● Documentación. Unity cuenta con una extensa documentación que facilita en gran medida su uso, contando con explicaciones de todas sus API y funciones.

● Comunidad de desarrolladores. Además de la documentación, Unity también tiene una comunidad de desarrolladores muy activa que no sólo ayuda a nuevos usuarios, sino también a resolver dudas de cualquier aspecto.

● Facilidad de uso. Las funciones de arrastrar y soltar, la jerarquía organizada de objetos en escena y la organización de sus *assets* hacen que el uso de Unity sea intuitivo y fluido.

Ilustración 10. Logo del motor de videojuegos Unity. (Fuente: Página web de Unity).

2.2.1.1. Conceptos básicos de Unity.

Seguidamente vamos a definir una serie de conceptos que se van a repetir a lo largo del proyecto y que son cruciales para entender el uso del programa Unity.

● Proyecto. Se trata del conjunto de archivos necesarios para el diseño de un juego. Estos archivos se denominan *assets*.

● Escena. Contiene los objetos de un juego.

● GameObject. Objetos fundamentales en Unity presentes en cada escena considerados contenedores. No hacen nada por sí mismos y necesitan de propiedades o componentes.

● Componente. Se asignan a los GameObject con el objetivo de proporcionarles una característica o comportamiento. Por ejemplo, el componente *transform* le asigna al objeto una posición y orientación y el *renderer* le confiere propiedades de visualización.

● Paternidad de objetos. Se pueden encontrar objetos padre e hijo. El *transform* del padre afecta al hijo.

● Textura. Imagen aplicable sobre un objeto.

● Shader. Se trata del código encargado de efectuar los cálculos para renderizar cada píxel.

● Material. Proporciona el valor a los parámetros que ha definido el shader.

● Script. Archivos programados en C# necesarios para responder a ciertas entradas, ejecutar eventos o controlar un comportamiento entre otras funciones.

● Prefab. Tipo de *asset* que permite almacenar un GameObject junto a la configuración de sus propiedades y componentes.

● Canvas. Se trata del GameObject donde deben estar todos los elementos de la interfaz de usuario (*User Interface, UI*).

● Collider. Componentes con el objetivo de proporcionar forma a un objeto para las colisiones físicas que le ocurran.

● Raycasting. Técnica por la cual podemos saber si existen *colliders* en una cierta dirección lanzando un rayo invisible.

● Asset Store. Tienda de Unity que contiene *assets* para descargar y utilizar. Los hay de pago y gratuitos.

2.2.1.2. Elementos del editor de Unity.

Dentro del editor de Unity encontraremos algunos elementos que nombraremos con regularidad durante el desarrollo del proyecto.

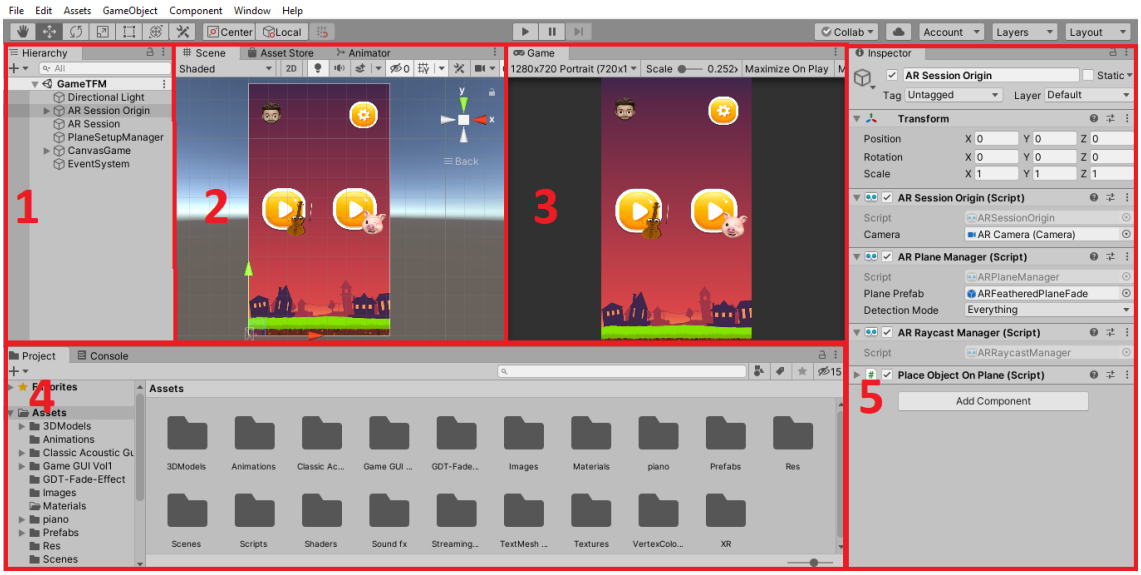

Ilustración 11. Elementos del editor de Unity.

1. Jerarquía. Contiene todos los GameObject de una escena de forma ordenada.

2. Escena. Permite editar una escena.

3. Vista de juego. Muestra lo que la cámara del juego ve de nuestro juego ya finalizado.

4. Contenido del proyecto. Panel donde se puede consultar las carpetas que componen el proyecto y el contenido de las mismas.

5. Inspector. Se utiliza para visualizar y editar los componentes y propiedades de los objetos y también otros ajustes de Unity.

Durante la realización del juego en Unity se va a mencionar la funcionalidad de arrastrar cualquier *asset* hacia un elemento del editor. Esto se debe a que Unity permite esta función en su entorno, pudiendo realizar esta acción en cualquier momento a excepción de la vista de juego.

2.2.2. AR Foundation.

En el apartado *2.1.2. Plataforma de Realidad Aumentada* se habló de AR Foundation como una tecnología multiplataforma de Unity con una amplia compatibilidad.

AR Foundation se basa en unas interfaces independientes de la plataforma para utilizar diferentes tipos de información, los subsistemas. Nosotros interactuaremos con el paquete relacionado con AR que se encuentra en AR Subsystems cuando sea necesario, como ejemplo, la interfaz para detectar planos la provee el paquete ARPlaneSubsystem[18].

Característica soportada	ARCore	ARKit	Magic Leap	HoloLens
Detección dispositivo	✓	✓	✓	✓
Detección de planos	✓	✓	✓	
Nubes de puntos	✓	✓		
Anclajes (*anchors*)	✓	✓	✓	✓
Estimación de luz	✓	✓		
Pruebas de entorno	✓	✓		
Detección facial	✓	✓		
Detección de imagen 2D	✓	✓	✓	
Detección de objeto 3D		✓		
Mallado		✓	✓	✓
Detección de cuerpo en 2D y 3D		✓		
Experiencias multi-usuario		✓		
Segmentación humana		✓		
Raycast	✓	✓	✓	
Pass-through video	✓	✓		
Administración de sesión	✓	✓	✓	✓
Oclusión	✓	✓		

Tabla 1.Características soportadas por las plataformas de AR. (Fuente: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.1/manual/index.html).

2.2.2.1. Configuración básica AR Foundation.

En una escena de AR Foundation deben estar dos GameObjects obligatoriamente, *ARSession* y *ARSessionOrigin.*

● *ARSession* tiene tres objetivos sobre el ciclo de vida de una sesión AR:

- Activar y desactivar AR.

- Comprobar si el dispositivo que se está utilizando soporta AR.

- Comprobar el estado del *tracking* o rastreo del mundo real.

● *AR Session Origin*. El propósito de este componente es el de transformar características rastreables, como superficies planas, en su posición, orientación y escala final dentro de la escena. Este GameObject es padre del objeto *AR Camera* cuyos componentes son los siguientes:

- AR Pose Driver. Añade la pose de un dispositivo a un GameObject.

- AR Camera Manager. Tiene como objetivos transformar el vídeo de la cámara a texturas y controlar la iluminación y el enfoque de la cámara del dispositivo.

- AR Camera Background. Permite utilizar la textura de la cámara del dispositivo como fondo.

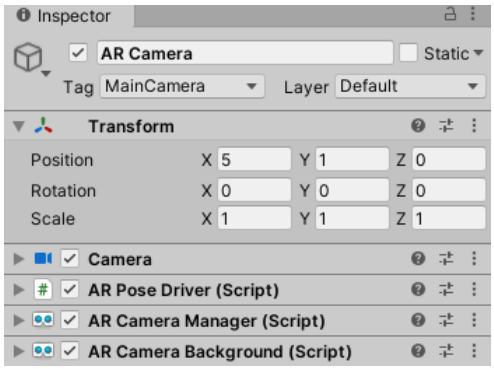

Ilustración 12. Contenido GameObject ARCamera.

Cuando se utiliza AR Foundation debemos recordar eliminar el GameObject *Main Camera* de nuestra jerarquía, ya que se va a utilizar la cámara del GameObject que hemos visto.

2.2.3. Visual Studio 2019.

Para el desarrollo de *scripts*, Unity te redirige a la descarga de este entorno de desarrollo integrado (*Integrated Development Environment, IDE*) gratuito y compatible con gran cantidad de lenguajes de programación como .NET, Java o C# en nuestro caso[19].

Ilustración 13. Logo Visual Studio 2019. (Fuente: Página web de Visual Studio).

2.2.4. Android Studio.

Dado que nosotros vamos a desarrollar específicamente para nuestro único dispositivo Android, vamos a necesitar el IDE de desarrollo Android Studio.

2.2.5. Android USB Driver para Windows.

El ordenador que vamos a utilizar para desarrollar la aplicación utiliza el sistema operativo Windows, esto hace que necesitemos un controlador (*driver*) USB para poder ejecutar pruebas en el dispositivo móvil.

2.2.6. Teléfono móvil compatible con ARCore o ARKit.

Para este proyecto necesitamos un dispositivo móvil compatible con ARCore, ya que no se va a desarrollar para iOS por la necesidad de un sistema Mac OS del que no se dispone. En nuestro caso se ha seleccionado un móvil Samsung Galaxy S8 cuyas especificaciones técnicas se detallan en la *Tabla 2*:

	Samsung Galaxy S8
Dimensiones	148.9 x 68.1 x 8mm.(155gr.)
Pantalla	Super AMOLED curva 5.8”
Resolución	1440 x 2960 píxeles
Procesador	Exynos 8895 o Qualcomm Snapdragon 835 (según región)

Núcleos	Octacore (2,3 Ghz + 1,7 Ghz) 64 Bit de 10 nanómetros
RAM	4 GB
Memoria	64 GB (UFS 2.1) ampliables vía microSD (hasta 256 GB)
Software original	Android 7.0 con TouchWiz
Software actual	Android 9.0
Conectividad	LTE Cat.16, Wi-Fi 802.11 a/b/g/n/ac (2.4/5GHz), VHT80 MU-MIMO, 1024QAM Bluetooth® v 5.0 (LE up to 2Mbps), ANT+, USB Type-C, NFC, Localización (GPS, Galileo, Glonass, BeiDou)
Cámaras	Trasera de 12 megapíxeles con una lente con OIS y f/1,7 Frontal de 8 megapíxeles con f/1,7, autofocus
Batería	3.000 mAh
Otros	Protección IP68, carga rápida y carga inalámbrica

Tabla 2. Especificaciones técnicas Samsung Galaxy S8.

Ilustración 14. Samsung Galaxy S8. (Fuente: Página web de la tienda Samsung).

2.2.7. Ordenador portátil.

El ordenador portátil que se ha utilizado para el desarrollo y donde se han instalado los programas vistos con anterioridad es un Asus A53SD SX375V. Se trata de un PC con especificaciones técnicas medias que cumple con los requisitos mínimos establecidos por Unity. Estos requisitos responden a la ejecución del programa y no a la velocidad de compilación, que siempre es mejorable cuanto mejor sea el dispositivo del que hagamos uso. Al igual que con el dispositivo móvil mostramos sus especificaciones en la siguiente tabla:

	Asus A53SD SX375V
Procesador	Intel Core i7-2670QM (4x2,20 Ghz / 6 Mb caché)
Pantalla	LED de 15.6” Glare HD
RAM (modificada)	8 GB de memoria RAM DDR3
Memoria (modificada)	500 GB 5400 rpm(SATA) + 250 GB SSD
Tarjeta gráfica	NVIDIA® GeForce® 610 de 2GB DDR3 dedicada
Dimensiones	378x253x28,3 (frontal) a 34,9 mm / 2,60 kg
Interfaces	1 HDMI, lector de tarjetas, 2 x USB 2.0, 1 x USB 3.0
Sistema operativo original	Windows® 7 Home (64bits)
Sistema operativo actual	Windows® 10 Pro (64bits)

Tabla 3. Especificaciones técnicas de ordenador portátil Asus A53SD SX375V.

Ilustración 15. Ordenador portátil Asus A53SD SX375V. (Fuente: Página web de Asus-shop).

Capítulo 3: Desarrollo.

Este proyecto se ha desarrollado en Unity y está destinado a un dispositivo Android, aunque gracias a AR Foundation podría ser compatible con iOS. En una primera parte veremos cómo se prepararon los dispositivos con los que se va a trabajar y en la segunda parte explicaremos los pasos que se han seguido para desarrollar la aplicación de AR de forma que pueda reproducirse en caso de ser necesario. Nuestro objetivo es el de hacer una aplicación que sirva a los niños para buscar una serie de objetos virtuales escondidos en el mundo real a través del sonido que éstos emiten.

3.1. Instalación de herramientas.

3.1.1. Preparación de ordenador.

El primer paso que se tomó fue el de instalar la última versión de Java existente (*version 8 update 261* en el momento en el que se escribió este informe). Para descargarlo podemos hacerlo directamente desde la página oficial de Java.

Para la instalación de Unity y las herramientas que se necesitan se ha seguido el manual de usuario de Unity[16] y el manual de AR Foundation[18]. Empezamos descargando e instalando El administrador Unity Hub de la página oficial, cuyo enlace se encuentra en el manual en el apartado *Installing Unity*. Una vez hemos seguido los pasos del instalador nos registramos e iniciamos sesión. Como mencionamos en el apartado *2.2.1. Unity Hub y Unity*, Unity es gratis para su uso personal y deberemos especificarlo en Unity Hub. Para ello, en el apartado *License Management* activamos una nueva licencia y marcamos la opción adecuada.

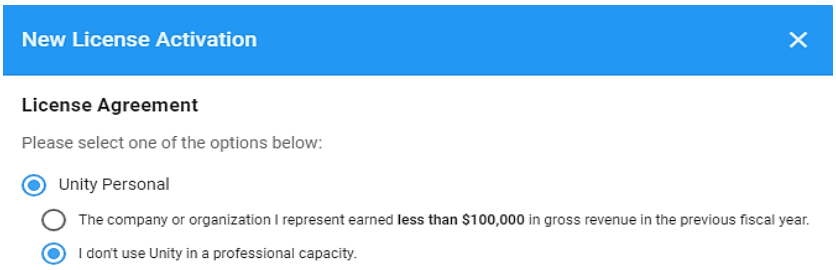

Ilustración 16. Activación de licencia gratuita de Unity Hub.

Ahora vamos a instalar Unity. Desde la pestaña *Installs* de Unity Hub (versión 2.3.2) seleccionamos la versión que nos convenga y los módulos necesarios.

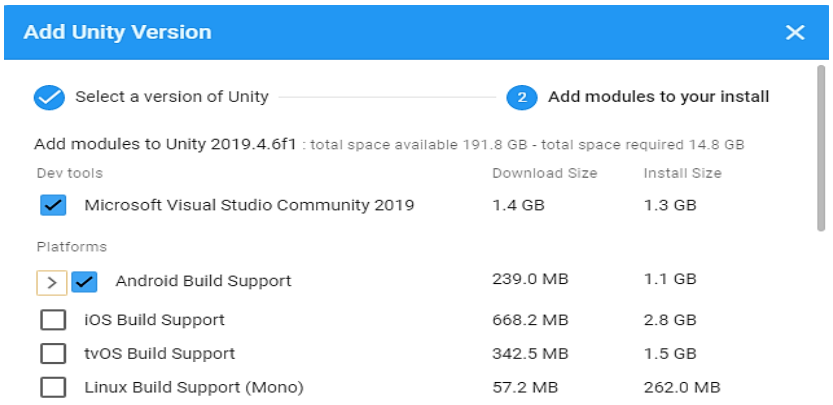

Ilustración 17. Selección de módulos de la versión Unity 2019.4.6.f1.

Después instalamos el IDE Visual Studio 2019, que también podemos descargar desde su página oficial.

Para terminar, vamos a hacer que nuestro ordenador sea compatible para trabajar con Android, por lo que tenemos que instalar Android Studio y un controlador para nuestro móvil. Al igual que el resto de software, vamos a descargar ambos elementos desde su página oficial en este caso, de los desarrolladores para Android: developer.android.com. En esta página podemos encontrar el software de Android Studio y su guía de usuario[20] donde encontraremos en el apartado *Cómo instalar controladores USB de OEM* las instrucciones para conseguir el *driver* para que el ordenador y el móvil puedan comunicarse de forma adecuada. En nuestro caso, siguiendo los pasos del manual, acabamos descargando el controlador desde la página de desarrolladores de Samsung: developer.samsung.com/mobile/android-usb-driver.html.

Una vez hemos instalado todo el software necesario debemos entrar en Unity y configurar ciertos parámetros necesarios para trabajar. Estos cambios se tienen que producir dentro del proyecto que queramos, por tanto, vamos a crear un proyecto desde Unity Hub y dentro de él seguiremos los siguientes pasos:

● Accedemos a la pestaña *Window* > *Pakcage Manager*.

● En el buscador de la ventana emergente buscamos AR Foundation e instalamos la versión 3.1.3.

● Buscamos ARCore XR Plugin e instalamos la versión 3.1.3.

● Buscamos ARKit XR Plugin e instalamos la versión 3.1.3.

● Accedemos a la pestaña *File* > *Build Settings…*.

● Seleccionamos el sistema para el que vamos a desarrollar y pulsamos el botón *Switch platform.*

● En la esquina inferior izquierda de la misma ventana seleccionamos *Player Settings*.

● Dentro del apartado *Player* y en el desplegable *Other Settings* vamos a realizar una serie de cambios:

- Deseleccionamos *Multithreaded Rendering**.

- Eliminamos Vulkan de la lista de *Graphics API*.

- Configuramos *Minimum API Level* con el valor *Android 7.0 'Nougat' (API level 24)*.

● Por último, al mismo nivel que el apartado de *Player* tenemos que acceder a *XR Plug-in Management* y seleccionamos la casilla de *ARCore* en *Plug-in Providers*.

En nuestro proyecto sólo hay una escena y también podremos dejarla cargada en Unity para cuando hagamos las pruebas en el móvil, para ello, en la ventana de *Build Settings…* hacemos click en el botón *Add Open* Scenes.

Una vez terminamos estas configuraciones nuestro ordenador queda configurado totalmente para desarrollar la aplicación y podemos configurar el dispositivo móvil que conectaremos mediante USB.

3.1.2. Preparación de dispositivo móvil.

Con el objetivo de utilizar el teléfono móvil como destino de nuestra aplicación, debemos habilitar la opción de depuración mediante USB del mismo y para ello volveremos a la guía de usuario de desarrollo en Android y accederemos al apartado *Cómo ejecutar tu app*:

- Abrimos el apartado de configuración o ajustes de nuestro dispositivo.
- Hacemos click en *Acerca del teléfono*.
- Seleccionamos *Información de sofware.*
- Dentro de este apartado pulsamos siete veces en *Número de compilación* y aparecerá un aviso advirtiendo de que se ha activado el modo desarrollador.
- Ahora regresamos a la ventana de *Ajustes* y dentro de *Opciones de desarrollador* habilitamos la depuración por USB.

Es importante tener la última versión de Android instalada en nuestro dispositivo móvil con el fin de evitar problemas de compatibilidad.

3.2. Desarrollo.

Dividiremos este apartado en base a los objetivos a lograr comenzando por conseguir ocultar un objeto virtual detrás de uno real, después implementaremos los objetos que se van a utilizar en la aplicación y las funciones principales de la aplicación y, por último, diseñaremos una interfaz útil y fácil de usar para el usuario.

3.2.1. Desarrollo de oclusión.

Empezamos dirigiéndonos a la jerarquía para eliminar la *Main Camera* ya que utilizaremos la cámara destinada a AR y después añadimos los Game Objects mínimos que deben estar presentes en la escena:

- AR Session. Activa y desactiva AR y controla su ciclo de vida.
- AR Session Origin. Sirve para controlar la posición y rotación de la cámara AR del dispositivo que se utiliza de forma indirecta ya que no se puede posicionar arbitrariamente en la escena de Unity.
- AR Camera. Se encuentra dentro del Game Object anterior como hijo, su función es la de permitir que el video de la cámara se transforme en texturas y controla la iluminación y el enfoque de la misma.

Una vez tenemos estos Game Objects en escena, vamos a crear una serie de carpetas que nos ayudarán a organizar el proyecto:

- 3DModels
- Images
- Materials
- Prefabs
- Res
- Scripts
- Shaders
- Textures

A partir de aquí se van a enumerar una serie de directrices llevadas a cabo para crear el manager del subsistema destinado al reconocimiento de planos, *AR Plane Manager*. Cabe destacar que gracias al manager podremos acceder a los datos del subsistema[5].

Para este *AR Plane Manager* vamos a necesitar crear un plano que nos proporcione oclusión. Su propósito es el de en un principio permitirnos visualizar los planos detectados a través de un material visible que le aplicaremos. Después le añadiremos una funcionalidad que nos aplique otro material, en este caso invisible, pero ocluyendo a los objetos virtuales. Por último, cuando este plano que creamos esté completo, lo guardaremos como *prefab* con el objetivo de usarlo en repetidas ocasiones.

- En la carpeta *Materials* creamos un material con el nombre *PlaneMat* y otro denominado *OcclussionMat*.
- En *OcclusionMat* cambiamos su shader a *AR/Occlussion*.
- En *PlaneMat* seleccionamos un shader *Unlit/FeatheredPlaneShader*.
- En la carpeta *Res* vamos a añadir un patrón de puntos y lo arrastraremos a la textura de nuestro *PlaneMat*, llamaremos a este patrón *PlanePatternDot*. En el inspector de Unity deberíamos ver que queda de la siguiente forma:

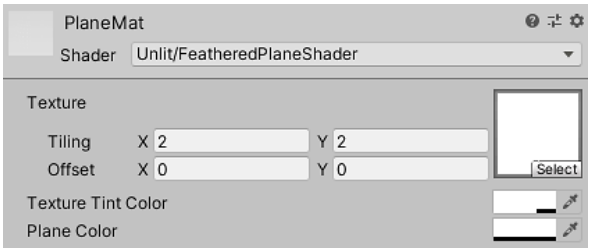

Ilustración 18. Características PlaneMat.

- Creamos en la jerarquía un objeto vacío y lo llamamos *ARFeatheredPlaneFade*.
- Añadiremos a este objeto los componentes *ARPlane (Script)*, *AR Plane Mesh Visualizer (Script)*, *Mesh Renderer* y *Animator*.
- Añadimos en *Element 0* de *Mesh Renderer* el material *PlaneMat*.
- En la carpeta *Animations* creamos una animación *AR Feathered Plane Fade* con dos estados que corresponden a la aparición (*PlaneFadeOn*) y a la desaparición (*PlaneFadeOff*) del patrón del plano y una transición en cada sentido.
- Añadimos esta animación al controlador del componente *Animator* del Game Object *ARFeatheredPlaneFade* en el inspector.
- Creamos dos scripts en la carpeta *Scripts*, los nombramos *AR Feathered Plane Mesh Visualizer*, para extender la malla por el plano, y *Fade Plane On Boundary Change*, para manejar la animación, y los añadimos a *ARFeatheredPlaneFade.*
- Incorporamos un *Mesh Collider* para proporcionar una malla al motor físico para colisiones y un *Mesh Filter* para asignar una malla 3D al objeto[21].
- Una vez terminamos nuestro *ARFeatheredPlaneFade* vamos a arrastrarlo y guardarlo a nuestra carpeta *Prefabs* y lo podemos borrar de nuestra jerarquía.

El siguiente paso será añadir el *prefab* que acabamos de crear, para ello vamos al Game Object *AR Session Origin* e incorporamos los componentes *AR Plane Manager* y *AR Raycast Manager*, éste último para la comprobación de *raycasts*. Entonces arrastramos al apartado de *Plane Prefab* nuestro Game Object *ARFeatheredPlaneFade*.

Existe otra alternativa a esta técnica para ocultación de objetos virtuales. En la versión más reciente de AR Foundation, versión 4.1, se puede utilizar la *Depth API* de Google. Permitiría

una oclusión más realista que utiliza un mapa de profundidad en vez de planos con este fin, pero existe la desventaja de necesitar un dispositivo que sea compatible con la *API* y esto no ocurre con todos ellos.

Ahora vamos a comprobar que todo funciona correctamente para que exista oclusión, pero necesitamos algún objeto en escena para ocultar por lo que vamos a crear un script para posicionar objetos. Añadiremos nuestro primer objeto y también crearemos dos botones y un script para darles funcionalidad:

- Creamos un script en nuestra carpeta y lo llamamos *PlaceObjectOnPlane*.
- Añadimos las librerías para el uso de AR Foundation (*UnityEngine.XR.ARFoundation*), el manejo del subsistema de AR (*UnityEngine.XR.ARSubsystems*) y la recepción de eventos del sistema (*UnityEngine.EventSystems*).
- Aunque ahora sólo necesitamos un objeto, en previsión de la aplicación que estamos creando, vamos a declarar una lista de Game Objects y otra serie de elementos necesarios. Deberíamos conseguir algo así:

```
private ARPlaneManager planeManager;
private ARRaycastManager raycastManager;
//Nuestro contenedor de toques
private List<ARRaycastHit> s_Hits = new List<ARRaycastHit>();
//Índice para recorrer nuestra lista de objetos
private int _objPlaceIndex = 0;

private List<GameObject> visualElements; //Lista con todos los objetos
```

Ilustración 19. Declaración inicial en PlaceObjectOnPlane.cs.

- Ahora necesitaremos un método *awake()* para obtener los componentes de *ARRaycastManager* de *ARPlaneManager* y un método *update()* que registre cuándo tocamos la pantalla.
- Después programamos el método para colocar objetos, *PlaceObject()*, y añadimos otro método para asegurarnos de que al tocar un botón no estamos posicionando un objeto, es decir, que no exista confusión.

```
private void PlaceObject()
{
    Touch touch = Input.GetTouch(0); //Toques en la pantalla

    if(touch.phase == TouchPhase.Began && !IsPointerOverUIObject()) //Se ha tocado la pantalla fuera de un objeto UI
    {
        if (raycastManager.Raycast(touch.position, s_Hits, TrackableType.PlaneWithinPolygon)) //Condicional de toque
        {
            Pose hitPose = s_Hits[0].pose; //hitPose contiene la pose (posición + orientación)

            if (_objPlaceIndex < visualElements.Count) //Quedan más objetos?
            {
                //Instanciamos cada objeto de la lista con la pose del toque en pantalla
                GameObject obj = Instantiate(visualElements[_objPlaceIndex], hitPose.position, hitPose.rotation);
            }
            _objPlaceIndex++;
        }
    }
}
private bool IsPointerOverUIObject() //Método para no confundir tocar un botón con tocar la pantalla
{
    PointerEventData eventDataCurrentPosition = new PointerEventData(EventSystem.current);
    eventDataCurrentPosition.position = new Vector2(Input.GetTouch(0).position.x, Input.GetTouch(0).position.y);
    List<RaycastResult> results = new List<RaycastResult>();
    EventSystem.current.RaycastAll(eventDataCurrentPosition, results);

    return results.Count > 0; //mayor que cero significa que hemos tocado un objeto UI
}
```

Ilustración 20. Método PlaceObject() junto a IsPointerUIObject().

- Añadimos este script a *AR Session Origin* mediante el inspector.
- Incorporamos el objeto a utilizar para comprobar oclusión creando un cubo y simplemente añadiéndolo a la lista de *visualElements* en el inspector de *AR Session Origin*. El objeto que creemos como prueba lo añadiremos a la jerarquía para darle el tamaño adecuado y lo guardaremos como prefab, eliminándolo de la jerarquía tal y como hicimos con *ARFeatheredPlaneFade*.
- El siguiente paso será crear un script denominado *PlaneSetupManager* para dar la funcionalidad a dos botones, uno aplicará el patrón para ver los planos detectados y el otro fijará el material de oclusión.
- Este script contendrá dos métodos: uno para aplicar el material de puntos y ver los planos que detectamos, *SetPlaneMaterial()*, y otro para el material de oclusión, *SetOcclussionMaterial()*. En la *Ilustración 19* enseñamos uno de los dos métodos ya que sólo varía el material que se aplica:

```
public void SetPlaneMaterial()
{
    planePrefab.GetComponent<Renderer>().material = planeMat;

    foreach (var plane in planeManager.trackables)
    {
        plane.GetComponent<Renderer>().material = planeMat;
    }
}
```

Ilustración 21. Método SetPlaneMaterial().

- En la jerarquía añadimos un objeto vacío nominado *PlaneSetupManager* y arrastramos hacia el inspector el script creado.
- También en el inspector deberemos añadir los elementos de la siguiente forma:

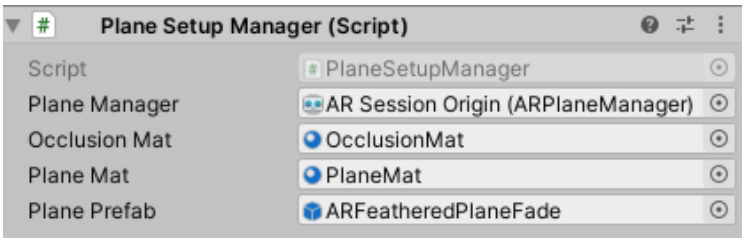

Ilustración 22. Variables de PlaneSetupManager.cs en Unity.

- Creamos dos botones y los llamaremos *SetOcclusionButton y SetPlaneButton* añadiendo la textura o texto que prefiramos.
- En el inspector de cada botón, en su método *OnClick()* debemos añadir una función. Arrastramos desde la jerarquía el objeto *PlaneSetupManager* y seleccionamos la función del script que corresponde a cada botón.

En este momento podemos comprobar en nuestro dispositivo móvil que la oclusión existe, para ello vamos a ir a la pestaña *File > Build Settings…*, seleccionamos nuestro dispositivo conectado por USB en *Run Device* y hacemos click en *Build and Run*. Después de un tiempo de compilación y transferencia al dispositivo podremos utilizar la prueba en nuestro móvil.

Ilustración 23. Primera prueba de oclusión con objeto arbitrario.

3.2.2. Funcionalidad principal de la aplicación.

Una vez conseguido el objetivo de oclusión vamos a centrarnos en el juego en sí. Va a constar de dos modalidades, una con instrumentos y otra con animales. Cada objeto emitirá un sonido y éste debe ser en 3D para que el volumen descienda conforme nos alejamos y viceversa.

Primero escogemos los modelos 3D que vamos a utilizar para el juego. Hemos escogido cinco objetos de cada modalidad. Podemos descargar distintos modelos desde páginas gratuitas de internet como *free3d* o *turbosquid*, o desde la tienda de Unity dentro del propio programa (*Asset Store*).

Vamos a listar una serie de pasos a seguir para conferir al objeto de un sonido en 3D:

- Posicionamos el objeto en la escena.
- Añadimos un componente *Audio Source* al objeto.
- Descargamos de internet o de la *Asset Store* el sonido que queramos y lo añadimos al proyecto en la carpeta *Res*.
- Arrastramos el archivo de sonido al componente en el inspector.
- Vamos a hacer que se reproduzca en bucle seleccionando la opción *Loop*.
- Subimos el nivel de *Spacial Blend* a 1 y cambiamos su *Volume Rolloff* a modo lineal.
- Por último, configuramos los valores de máxima y mínima distancia como nos parezca conveniente. En nuestro caso fueron los valores 1 y 2.

No debemos olvidar añadir un componente *Audio listener* al Game Object *AR Camera* para que nuestro dispositivo pueda escuchar el sonido que emite el objeto.

El siguiente paso será el de dimensionar y orientar todos los objetos. Vamos a posicionarlos todos en escena y variaremos su tamaño entre ellos para que guarden una medida proporcionada. Dado que en el script *PlaceObjecOnPlane* los objetos se instancian con la orientación adquirida de nuestro toque en pantalla, los objetos aparecen en su forma original aún cambiando los valores en el transform, lo que muchas veces no es conveniente ya que pueden aparecer con una pose no adecuada. Para solucionar este problema hemos creado un objeto vacío que será padre de cada uno de los objetos que queremos mostrar por pantalla y será a este objeto vacío al que cambiaremos los valores de rotación.

Esta modificación también soluciona un problema en la altura de los objetos ya que en algunos de ellos los ejes se encuentran en el centro del objeto y al instanciarlos parte de éste queda por debajo del plano, provocando que al aplicar oclusión parte del objeto desaparezca como podemos ver en la *Ilustración 22*.

Ilustración 24. Objeto oculto en el suelo por error.

No debemos olvidar guardar cada objeto creado en nuestra carpeta de prefabs, añadirlos a la lista de *visualAnimals* o *visualMusic* según corresponda y borrarlos de la jerarquía.

Vamos a añadir una función para que cuando ya hayamos posicionado todos los objetos en escena aparezca un cartel indicándolo al jugador. A la misma vez, utilizaremos la condición de lista de objetos vacía para hacer un cambio de botones, es decir, al entrar a la aplicación tendremos la opción de escanear los planos y cuando el último objeto esté ya en escena, dejaremos que aparezca el botón para aplicar la oclusión y empezar el juego:

- Creamos un texto tipo UI llamado *WarningText* y escribiremos: Ya no quedan más objetos.
- Lo guardaremos como prefab y lo eliminaremos de jerarquía.
- En *PlaceObjecOnPlane* vamos a crear un Game Object para el texto y otros dos para los botones.
- Añadimo un método *ShowWarningText()* que instancie el texto *WarningText* y lo destruya en un segundo haciendo que desaparezca de escena.

```
private void ShowWarningText()
{
    Pose hitPose = s_Hits[0].pose;
    GameObject text = Instantiate(WarningText, hitPose.position, hitPose.rotation);
    Destroy(text, 1);
}
```

Ilustración 25. Método para mostrar el texto de advertencia.

- Por último, en el método *PlaceObject()*, en nuestra condición anterior que comprueba si quedan elementos en nuestra lista, añadimos un *else* para llamar al método que muestra el texto y mediante *SetActive()* habilitamos el botón *SetOcclussionButton* y deshabilitamos *SetPlaneButton*.

3.2.3. Interfaz y funciones secundarias.

Ahora que todas las funcionalidades principales están implementadas, podemos crear una interfaz amigable para el usuario tanto adulto, que será el encargado de esconder los objetos, como el infantil, destinado a explorar y buscar a través del oído y la vista.

Se va a crear una pantalla principal que será el inicio del juego y ofrecerá las dos modalidades de éste. También diseñaremos una pantalla de ajustes u opciones a la que se accederá desde la pantalla de inicio y ofrecerá las funciones de activar o desactivar volumen, acceder a las redes sociales del desarrollador o abrir un texto informativo. Por último, modificaremos la escena de juego (*Canvas*) para añadir otras opciones.

Empezamos descargando de la *Asset Store* un conjunto de botones, en nuestro caso utilizamos el *Game GUI Buttons*. Para simplificar el proceso dividiremos la explicación por distintas pantallas:

- Pantalla principal (*Main Background*). Pantalla inicial del juego.
- Pantalla de juego (*CanvasGame*). Donde se ejecuta la funcionalidad principal.
- Pantalla de configuración (*SettingsBackground*). Funciones secundarias de volumen e información.

Para el fondo de todas las pantallas, a excepción del *CanvasGame*, se ha utilizado una imagen obtenida de internet. Cuando queremos añadir una imagen al canvas necesitamos cambiar su tipo de textura en el inspector a *Sprite (2D and UI*) y ya podremos añadirla a la escena. Además, también se han usado imágenes para las texturas de los botones, los carteles informativos y para la interfaz de juego que evitará encontrar los objetos de forma demasiado sencilla.

Existe un gran número de botones en nuestra aplicación, por lo que vamos a listarlos en una tabla:

Nombre	Localización	Funcionalidad
SetPlaneButton	CanvasGame	Escanear planos
SetOcclussionButton	Canvas Game	Aplicar oclusión
QuestionButton	Canvas Game	Abrir ventana de instrucciones de juego
HomeInGameButton	Canvas Game	Volver a pantalla principal
SettingsButton	Main Background	Ir a pantalla de configuración
MusicPlayButton	Main Background	Jugar a modalidad instrumentos
AnimalPlayButton	Main Background	Jugar a modalidad animales
ExitButton	Ventana emergente de QuestionButton	Salir de ventana de instrucciones de juego
TwitterButton	SettingsBackground	Ir a la cuenta de Twitter del desarrollador
FacebookButton	SettingsBackground	Ir a la cuenta de Facebook del desarrollador
SoundOnButton	SettingsBackground	Encender volumen
SoundOffButton	SettingsBackground	Apagar volumen
HomeButton	SettingsBackground	Volver a pantalla principal
InfoButton	SettingsBackground	Abrir ventana de información
InfoExitButton	Ventana emergente de InfoButton	Salir de ventana de información

Tabla 4. Distribución y funcionalidad de botones.

Gran parte de estos botones sólo tienen la función de activar o desactivar ciertos Game Object (*SetActive()*) o pausar la música añadida a los menús del juego, en cambio otros botones utilizan métodos que vamos a crear en el script *PlaceObjectOnPlane*. Estos métodos son:

● MusicPlayButton(). A la lista de elementos que vamos a utilizar en el juego le pasamos el contenido de la lista que contiene todos los objetos de instrumentos. Se aplica al botón *MusicPlayButton*.

● AnimalPlayButton(). Tiene la misma funcionalidad que el método anterior con la diferencia de que la lista contiene ahora todos los objetos de animales. Se aplica al botón *AnimalPlayButton.*

● BackToHome(). Mediante la librería *SceneManager* cargamos la escena, por lo que volvemos a la pantalla principal. Se aplica a los botones *HomeInGameButton* y *HomeButton.*

● SetInterface(). Se habilita la imagen de la interfaz de juego mediante el método *SetActive()*. Se aplica cuando ya no quedan más elementos por posicionar en la escena.

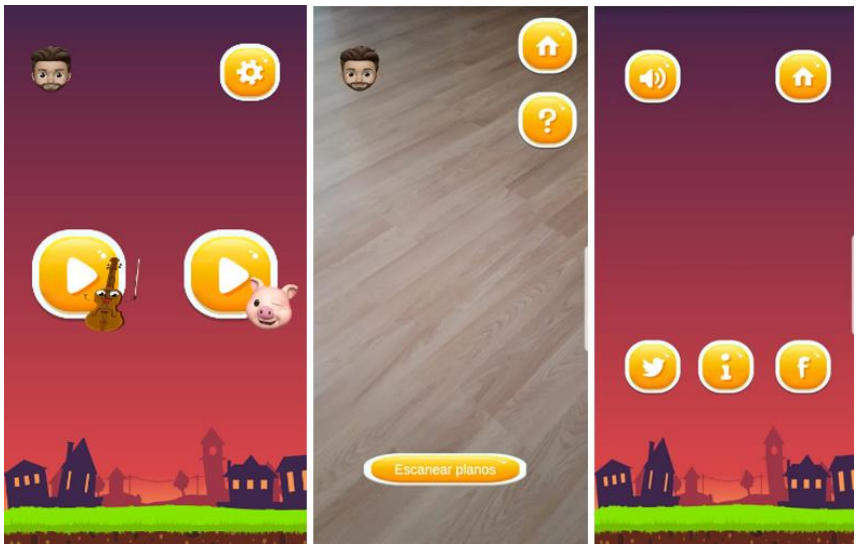

Ilustración 26. Pantallas de inicio, juego y configuraciones.

Capítulo 4: Ejemplo de uso.

El sistema ha sido diseñado para el uso de dos personas, un supervisor que escanee los planos, esconda los objetos y aplique la oclusión, y un jugador que explore la estancia buscándolos a través del sonido y con la dificultad añadida que le confiere la interfaz de juego.

4.1. Preparación del escenario de juego.

En esta parte será el usuario supervisor quien actúe. Como vemos en la *Ilustración 24*, en la pantalla principal podremos elegir entre las dos modalidades de juego o acceder a la pantalla de configuración. El supervisor puede elegir una de las dos modalidades de juego y al hacerlo se activará la cámara de nuestro dispositivo.

En la pantalla de juego el usuario puede volver a la pantalla de inicio o pulsar el botón de *Escanear planos*. En este segundo caso deberá escanear lentamente cada plano donde quiera posicionar o esconder un objeto.

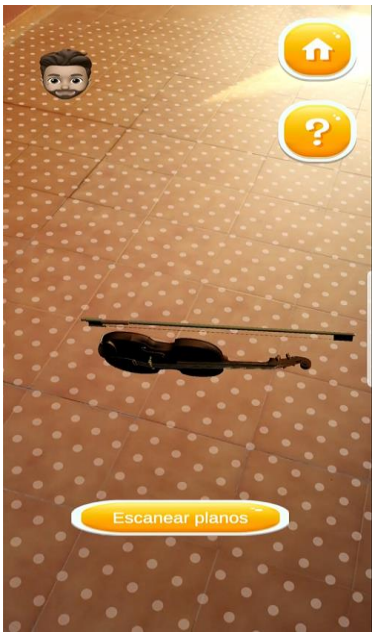

Ilustración 27. Escaneo de planos y posicionamiento de objeto.

Una vez haya posicionado todos los objetos en escena aparecerá un cartel indicándolo y se activará el botón *Aplicar oclusión y jugar*. Al pulsarlo se aplica la oclusión y aparece una interfaz para dificultar la búsqueda. Ante cualquier duda se ha dotado de un botón en la pantalla de juego con instrucciones a seguir.

Ilustración 28. Cartel con texto de advertencia.

4.2. Comienzo del juego.

Ahora entra en juego el segundo usuario, la persona que va a jugar. Ahora el niño a través del oído y la vista deberá empezar a buscar los objetos colocados anteriormente.

En cualquier momento se podrá cambiar de modalidad de juego pulsando el botón de volver al inicio (arriba a la derecha), reiniciándose así el juego y volviendo a iniciar el proceso.

Ilustración 29. Capturas del entorno de juego.

Capítulo 5: Conclusión y líneas futuras.

5.2. Conclusiones.

Este proyecto me ha permitido profundizar en el mundo de la Realidad Aumentada, adquiriendo habilidades nuevas y desarrollando otras con las que ya contaba desde mi paso por la asignatura del máster dedicada a ello.

Se ha conseguido combatir uno de los principales retos de esta tecnología y crear una aplicación sencilla orientada al descubrimiento de los niños. A través de herramientas como Unity y AR Foundation se ha conseguido un desarrollo sencillo y multiplataforma.

También se ha implementado la característica de sonido 3D para dotar a la aplicación de un fin más allá de su desarrollo, la educación multisensorial, fomentando la exploración a través del movimiento y la manipulación, y ofreciéndole al niño la capacidad de crear aprendizajes significativos[22].

La forma de redacción de este informe se ha orientado para que sea posible la reproducción del proyecto y su aplicación asociada, dejando así la opción de ampliación del mismo.

5.3. Líneas futuras y mejoras.

Como se comentó en el apartado anterior este informe está redactado de forma que sea posible ampliarlo o cambiarlo, añadiendo cualquier característica como vibración por cercanía al objeto o convertir la búsqueda de objetos en una caza (aumentaría el rango de edad para el uso de la aplicación).

El desarrollo de AR es constante y durante la realización de este proyecto Unity lanzó la herramienta Unity Mars. Este avance cuenta con una prueba gratuita de 45 días y posibilita la creación de aplicaciones inteligentes de AR que permiten la interacción con el espacio físico y funciona en cualquier lugar y con cualquier tipo de datos[23]. Con esta herramienta se podría subir de nivel en nuestra aplicación, ya que seríamos capaces de escanear todos los objetos de una habitación y que lo objetos se escondan automáticamente sin la intervención directa de un supervisor.

Como se menciona en este informe, la AR no sólo se enfrenta al reto de la oclusión. Hoy en día se sigue avanzando en técnicas de reconocimiento, detección y seguimiento que se incorporan poco a poco a AR Foundation. Esto hace posible mejorar ampliamente en la interacción entre objetos del mundo real y el virtual.

Bibliografía.

[1] Https://www.ionos.es/digitalguide/online-marketing/vender-en-internet/realidad-extendida/, "XR : ¿ qué es la realidad extendida ?," pp. 1–11, 2020.

[2] M. J. Abásolo *et al.*, "Aplicaciones de Realidad Extendida y Aplicaciones Móviles," pp. 327–333.

[3] N. Bockholt, "Realidad virtual, realidad aumentada, realidad mixta. y ¿qué significa 'inmersión' realmente?," 2020. https://www.thinkwithgoogle.com/intl/es-es/futuro-del-marketing/nuevas-tecnologias/realidad-virtual-aumentada-mixta-que-significa-inmersion-realmente/?gclid=Cj0KCQiAqdP9BRDVARIsAGSZ8Am2l695WSOZCRBcYvS2IAOTnlFXbwaZ91IV0WhYyM1lGekm1lqqYtUaAls8EALw_wcB.

[4] F. Losilla, "Realidad Virtual y Aumentada. Apuntes de clase del Máster de Ingeniería telemática de la Universidad Politécnica de Cartagena.," 2017. [Online]. Available: http://books.google.com/books?id=XJhHMwEACAAJ&pgis=1.

[5] F. Losilla, "Realidad aumentada. Apuntes de clase del Máster de Ingeniería telemática de la Universidad Politécnica de Cartagena.," pp. 70–72, 2007.

[6] A. Merino, "Realidad mixta » Realidad virtual," *Univ. Católica Nuestra Señora la Asunción*, 2018.

[7] "Realidad mixta - ¿Qué es y qué oportunidades nos ofrecerá? | Editeca." https://editeca.com/realidad-mixta/ (accessed Nov. 18, 2020).

[8] F. Montecé-Mosquera, A. Verdesoto-Arguello, C. Montecé-Mosquera, and C. Caicedo-Camposano, "Impacto De La Realidad Aumentada En La Educación Del Siglo XXI," *Eur. Sci. Journal, ESJ*, vol. 13, no. 25, p. 129, 2017, doi: 10.19044/esj.2017.v13n25p129.

[9] I. Ugr-raing *et al.*, "La Ingeniería como facilitador de los ODS : Inteligencia Artificial y Tecnologías Digitales Disruptivas," 2020.

[10] A. Méndez, M. Rodrigo, and J. A. Ferreyra, "El uso de las TIC en la Educación Especial: descripción de un sistema informático para niños discapacitados visuales en etapa preescolar," *TE*, vol. no. 3, Aug. 2009, Accessed: Nov. 19, 2020. [Online]. Available: http://repositoriocdpd.net:8080/handle/123456789/351.

[11] H. Fuchs, "Optical Versus Video See-Through Head-Mounted Displays in Medical Visualization Article in Presence Teleoperators & Virtual Environments," 2000, doi: 10.1162/105474600566808.

[12] D. Mill, D. M. Carmen, and J. Lizandra, "Posibilidades de la Realidad Aumentada en entornos laborables . Estudio comparativo entre dispositivos video see-through y optical see-through Declaration of Authorship."

[13] "Tipos de realidad aumentada." https://www.raykarimi.com/es/types-of-augmented-reality (accessed Nov. 19, 2020).

[14] Z. Oufqir, A. El Abderrahmani, and K. Satori, "ARKit and ARCore in serve to augmented reality," Jun. 2020, doi: 10.1109/ISCV49265.2020.9204243.

[15] R. Zatarain-Cabada, M. L. Barrón-Estrada, M. B. Ibañez-Espiga, and A. Uriarte-Portillo, "Cuerpos y planos geométricos usando realidad aumentada y computación afectiva," *Res. Comput. Sci.*, vol. 147, no. 8, pp. 203–213, 2018, doi: 10.13053/rcs-147-8-15.

[16] Unity, "Unity - Manual: Instalación de Unity Hub."

https://docs.unity3d.com/Manual/GettingStartedInstallingHub.html (accessed Nov. 20, 2020).

[17] J. D. Craighead, J. Burke, R. R. Murphy, J. Craighead, J. Burke, and R. Murphy, "Using the Unity Game Engine to Develop SARGE: A Case Study Wandering Pattern Detection for Person with Dementia View project Using the Unity Game Engine to Develop SARGE: A Case Study." Accessed: Nov. 20, 2020. [Online]. Available: https://www.researchgate.net/publication/265284198.

[18] Unity, "About AR Foundation," 2020. https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@2.2/manual/index.html (accessed Nov. 20, 2020).

[19] B. Johnson, "Essential Visual Studio 2019," *Essential Visual Studio 2019*, 2020. https://visualstudio.microsoft.com/es/vs/.

[20] AndroidStudio, "Introducción a Android Studio | Desarrolladores de Android," *Obtenido de httpsdeveloperandroidcomstudiointrohles419*. 2015, Accessed: Nov. 20, 2020. [Online]. Available: https://developer.android.com/studio/intro.

[21] F. Losilla, "Unity. Apuntes de clase del Máster de Ingeniería telemática de la Universidad Politécnica de Cartagena.," pp. 1–34.

[22] E. D. E. Padres, "Enseñanza multisensorial. Ventajas del método de enseñanza multisensorial," pp. 1–8, 2020.

[23] "Unity MARS: la herramienta más nueva para los desarrolladores de AR | Aplicación de desarrollo de realidad aumentada con menos programación | MARS | Unity." https://unity.com/es/products/unity-mars (accessed Nov. 23, 2020).

Anexo.

A. ARFeatheredPlaneMeshVisualizer.cs

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

/// <summary>
/// This plane visualizer demonstrates the use of a feathering effect
/// at the edge of the detected plane, which reduces the visual impression
/// of a hard edge.
/// </summary>
[RequireComponent(typeof(ARPlaneMeshVisualizer), typeof(MeshRenderer),
typeof(ARPlane))]
public class ARFeatheredPlaneMeshVisualizer : MonoBehaviour
{
    [Tooltip("The width of the texture feathering (in world units).")]
    [SerializeField]
    float m_FeatheringWidth = 0.2f;

    /// <summary>
    /// The width of the texture feathering (in world units).
    /// </summary>
    public float featheringWidth
    {
        get { return m_FeatheringWidth; }
        set { m_FeatheringWidth = value; }
    }

    void Awake()
    {
        m_PlaneMeshVisualizer = GetComponent<ARPlaneMeshVisualizer>();
        m_FeatheredPlaneMaterial = GetComponent<MeshRenderer>().material;
        m_Plane = GetComponent<ARPlane>();
    }

    void OnEnable()
    {
        m_Plane.boundaryChanged += ARPlane_boundaryUpdated;
    }

    void OnDisable()
    {
        m_Plane.boundaryChanged -= ARPlane_boundaryUpdated;
    }

    void ARPlane_boundaryUpdated(ARPlaneBoundaryChangedEventArgs eventArgs)
    {
        GenerateBoundaryUVs(m_PlaneMeshVisualizer.mesh);
    }

    /// <summary>
    /// Generate UV2s to mark the boundary vertices and feathering UV coords.
    /// </summary>
    /// <remarks>
    /// The <c>ARPlaneMeshVisualizer</c> has a <c>meshUpdated</c> event that can
be used to modify the generated
    /// mesh. In this case we'll add UV2s to mark the boundary vertices.
```

```
    /// This technique avoids having to generate extra vertices for the boundary.
It works best when the plane is
    /// is fairly uniform.
    /// </remarks>
    /// <param name="mesh">The <c>Mesh</c> generated by
<c>ARPlaneMeshVisualizer</c></param>
    void GenerateBoundaryUVs(Mesh mesh)
    {
        int vertexCount = mesh.vertexCount;

        // Reuse the list of UVs
        s_FeatheringUVs.Clear();
        if (s_FeatheringUVs.Capacity < vertexCount) { s_FeatheringUVs.Capacity =
vertexCount; }

        mesh.GetVertices(s_Vertices);

        Vector3 centerInPlaneSpace = s_Vertices[s_Vertices.Count - 1];
        Vector3 uv = new Vector3(0, 0, 0);
        float shortestUVMapping = float.MaxValue;

        // Assume the last vertex is the center vertex.
        for (int i = 0; i < vertexCount - 1; i++)
        {
            float vertexDist = Vector3.Distance(s_Vertices[i],
centerInPlaneSpace);

            // Remap the UV so that a UV of "1" marks the feathering boudary.
            // The ratio of featherBoundaryDistance/edgeDistance is the same as
featherUV/edgeUV.
            // Rearrange to get the edge UV.
            float uvMapping = vertexDist / Mathf.Max(vertexDist -
featheringWidth, 0.001f);
            uv.x = uvMapping;

            // All the UV mappings will be different. In the shader we need to
know the UV value we need to fade out by.
            // Choose the shortest UV to guarentee we fade out before the border.
            // This means the feathering widths will be slightly different, we
again rely on a fairly uniform plane.
            if (shortestUVMapping > uvMapping) { shortestUVMapping = uvMapping; }

            s_FeatheringUVs.Add(uv);
        }

        m_FeatheredPlaneMaterial.SetFloat("_ShortestUVMapping",
shortestUVMapping);

        // Add the center vertex UV
        uv.Set(0, 0, 0);
        s_FeatheringUVs.Add(uv);

        mesh.SetUVs(1, s_FeatheringUVs);
        mesh.UploadMeshData(false);
    }

    static List<Vector3> s_FeatheringUVs = new List<Vector3>();

    static List<Vector3> s_Vertices = new List<Vector3>();

    ARPlaneMeshVisualizer m_PlaneMeshVisualizer;
```

```
    ARPlane m_Plane;

    Material m_FeatheredPlaneMaterial;
}
```

B. FadePlaneOnBoundaryChange.cs

```
using UnityEngine;
using UnityEngine.XR.ARFoundation;

[RequireComponent(typeof(ARPlane))]
[RequireComponent(typeof(Animator))]
public class FadePlaneOnBoundaryChange : MonoBehaviour
{
    const string k_FadeOffAnim = "FadeOff";
    const string k_FadeOnAnim = "FadeOn";
    const float k_TimeOut = 2.0f;

    Animator m_Animator;
    ARPlane m_Plane;

    float m_ShowTime = 0;
    bool m_UpdatingPlane = false;

    void OnEnable()
    {
        m_Plane = GetComponent<ARPlane>();
        m_Animator = GetComponent<Animator>();

        m_Plane.boundaryChanged += PlaneOnBoundaryChanged;
    }

    void OnDisable()
    {
        m_Plane.boundaryChanged -= PlaneOnBoundaryChanged;
    }

    void Update()
    {
        if (m_UpdatingPlane)
        {
            m_ShowTime -= Time.deltaTime;

            if (m_ShowTime <= 0)
            {
                m_UpdatingPlane = false;
                m_Animator.SetBool(k_FadeOffAnim, true);
                m_Animator.SetBool(k_FadeOnAnim, false);
            }
        }
    }

    void PlaneOnBoundaryChanged(ARPlaneBoundaryChangedEventArgs obj)
    {
        m_Animator.SetBool(k_FadeOffAnim, false);
        m_Animator.SetBool(k_FadeOnAnim, true);
        m_UpdatingPlane = true;
        m_ShowTime = k_TimeOut;
    }
}
```

C. PlaceObjectOnPlane.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
using UnityEngine.EventSystems;
using UnityEngine.SceneManagement;


public class PlaceObjectOnPlane : MonoBehaviour
{
    private ARPlaneManager planeManager;
    private ARRaycastManager raycastManager;
    //Nuestro contenedor de toques
    private List<ARRaycastHit> s_Hits = new List<ARRaycastHit>();
    //Índice para recorrer nuestra lista de objetos
    private int _objPlaceIndex = 0;

    private List<GameObject> visualElements; //Lista con todos los objetos
    public List<GameObject> visualMusic; //Lista juego de instrumentos
    public List<GameObject> visualAnimals; //Lista juego de animales
    //Texto de advertencia
    public GameObject WarningText;

    public GameObject KidInterface;
    public GameObject setPlaneButton;
    public GameObject setOcclusionButton;

    private void Awake() //Este método es llamado cuando se carga la instancia
del script o se activa un Game Object que contiene el script
    {
        raycastManager = GetComponent<ARRaycastManager>();
        planeManager = GetComponent<ARPlaneManager>();
    }

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

        if(Input.touchCount == 1)//hay que truncar el condicional para que si se
pulsa más de uno no aparezca el siguiente
        {
            PlaceObject();
        }
    }

    private void PlaceObject()
    {
        Touch touch = Input.GetTouch(0); //Toques en la pantalla

        if(touch.phase == TouchPhase.Began && !IsPointerOverUIObject()) //Se ha
tocado la pantalla fuera de un objeto UI
        {
            if (raycastManager.Raycast(touch.position, s_Hits,
TrackableType.PlaneWithinPolygon)) //Condicional de toque
            {
                Pose hitPose = s_Hits[0].pose; //hitPose contiene la pose
(posición + orientación)
```

```
                if (_objPlaceIndex < visualElements.Count) //Quedan más objetos?
                {
                    //Instanciamos cada objeto de la lista con la pose del toque
en pantalla
                    GameObject obj = Instantiate(visualElements[_objPlaceIndex],
hitPose.position, hitPose.rotation);
                }
                else
                {
                    ShowWarningText();
                    setOcclusionButton.SetActive(true);
                    setPlaneButton.SetActive(false);
                }
                _objPlaceIndex++;
            }
        }
    }

    private bool IsPointerOverUIObject() //Método para no confundir tocar un
botón con tocar la pantalla
    {
        PointerEventData eventDataCurrentPosition = new
PointerEventData(EventSystem.current);
        eventDataCurrentPosition.position = new
Vector2(Input.GetTouch(0).position.x, Input.GetTouch(0).position.y);
        List<RaycastResult> results = new List<RaycastResult>();
        EventSystem.current.RaycastAll(eventDataCurrentPosition, results);

        return results.Count > 0; //mayor que cero significa que hemos tocado un
objeto UI
    }

    private void ShowWarningText()
    {
        Pose hitPose = s_Hits[0].pose;
        GameObject text = Instantiate(WarningText, hitPose.position,
hitPose.rotation);
        Destroy(text, 1);
    }

    public void BackToHome()
    {
        SceneManager.LoadScene("GameTFM");
    }

    public void MusicPlayButton()
    {
        foreach(GameObject obj in visualMusic)
        {
            visualElements = visualMusic;
        }
    }

    public void AnimalPlayButton()
    {
        foreach (GameObject obj in visualAnimals)
        {
            visualElements = visualAnimals;
        }
    }
    public void setInterface()
    {
```

```
            KidInterface.SetActive(true);
        }
}
```

D. PlaneSetupManager.cs

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

public class PlaneSetupManager : MonoBehaviour
{

    public ARPlaneManager planeManager;
    public Material occlusionMat, planeMat;
    public GameObject planePrefab;

    public void SetOcclusionMaterial()
    {
        planePrefab.GetComponent<Renderer>().material = occlusionMat;

        foreach (var plane in planeManager.trackables)
        {
            plane.GetComponent<Renderer>().material = occlusionMat;
        }
    }

    public void SetPlaneMaterial()
    {
        planePrefab.GetComponent<Renderer>().material = planeMat;

        foreach (var plane in planeManager.trackables)
        {
            plane.GetComponent<Renderer>().material = planeMat;
        }
    }
}
```