Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

MÁSTER UNIVERSITARIO EN DESARROLLO DE VIDEOJUEGOS

Trabajo Fin de Máster

# Explorando la aplicación de la realidad virtual en la enseñanza: Un enfoque en el desarrollo de videojuegos con Oculus Quest 2

*Exploring the application of Virtual Reality in education: An approach on Game Development with Oculus Quest 2*

Felipe Andrés Álvarez Avaria

D./Dña. **José Ignacio Estévez Damas**, con N.I.F. 43786097P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

**C E R T I F I C A**

Que la presente memoria titulada:

“Explorando la aplicación de la realidad virtual en la enseñanza: Un enfoque en el desarrollo de videojuegos con Oculus Quest 2”

Ha sido realizada bajo su dirección por D. Felipe Andrés Álvarez Avaria, con N.I.F. 55366894Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en San Cristobal de La Laguna, a 7 de julio de 2023.

**Resumen**

El presente trabajo de Fin de Máster, explora la aplicación de la tecnología de realidad virtual sobre la plataforma Oculus Quest 2 para la enseñanza del ciclo de estaciones del planeta Tierra en el ámbito educativo.

El dispositivo de realidad virtual escogido es de los más competitivos en el mercado, por su sencillez de uso y, asegurando una calidad decente a un costo moderadamente bajo, lo que la convierte en una plataforma óptima para centros educativos y para personas que se están iniciando en la realidad virtual.

En este trabajo se exploran conceptos de desarrollo de videojuegos, donde hay que diseñar, desarrollar y producir uno, aunque también se exploran conceptos técnicos en lo referido a la realidad virtual, así como algunos conceptos de astronomía.

**Palabras clave:** Realidad Virtual, Oculus Quest 2, Enseñanza, Desarrollo de Videojuegos.

**Abstract**

The present Master's thesis explores the application of virtual reality technology on the Oculus Quest 2 platform for teaching the Earth's seasons cycle in the educational field.

The chosen virtual reality device is one of the most competitive in the market due to its user-friendly interface and decent quality at a moderately low cost, making it an optimal target platform for educational institutions and individuals who are new to virtual reality.

This work explores game development concepts, including the design, development, and production of a game, while also delving into technical aspects of virtual reality and some astronomy concepts.

**Keywords:** Virtual Reality, Oculus Quest 2, Education, Game development.

# Índice

# Capítulo 1
# Introducción

La Realidad Virtual (RV) ha emergido como una tecnología revolucionaria que ha transformado la forma en que interactuamos con el mundo digital. Sus aplicaciones en diversos campos, como la educación y la simulación, han abierto un abanico de posibilidades para el aprendizaje inmersivo y la experimentación científica.

El objetivo principal de este Trabajo de Fin de Máster es desarrollar una maqueta del sistema solar utilizando la tecnología de Realidad Virtual, que permita a los usuarios experimentar y comprender de manera interactiva cómo ocurren las estaciones en nuestro planeta. Para lograr esto, se aprovecha el potencial del Oculus Quest 2, un dispositivo de RV autónomo y de alta calidad que brinda una experiencia inmersiva sin necesidad de cables ni equipos adicionales.

La elección de Oculus Quest 2 como plataforma se basa en sus capacidades técnicas, su comodidad de uso y su accesibilidad. Este dispositivo, equipado con potentes sensores y controladores de movimiento, permite a los usuarios explorar entornos virtuales de manera intuitiva y natural. Además, al tener la posibilidad de operar sin cables, junto a su diseño ergonómico y su sistema de seguimiento preciso ofrecen una experiencia cómoda y libre de obstáculos físicos, lo que garantiza una inmersión total en el mundo virtual.

Por otro lado, la simulación del ciclo de estaciones en la Tierra es un tema fundamental en la educación científica, ya que proporciona una comprensión profunda de los fenómenos astronómicos y climáticos que afectan nuestro planeta. Sin embargo, la implementación de experimentos prácticos puede ser limitada debido a las restricciones físicas y la dificultad para visualizar los movimientos celestes en tiempo real.

En las siguientes secciones, se describe el proceso de desarrollo de un videojuego serio, así como las metodologías utilizadas para crear una experiencia educativa inmersiva y precisa. Además, se presentarán los resultados y las conclusiones obtenidas, destacando el potencial de la Realidad Virtual en la enseñanza y el aprendizaje de conceptos científicos complejos, como el ciclo de estaciones en la Tierra.

## 1.1. Antecedentes

La comprensión del ciclo de estaciones en la Tierra ha sido un tema de interés en la astronomía y la ciencia durante siglos. Desde los primeros estudios de los astrónomos clásicos hasta las investigaciones modernas, se han realizado numerosos avances en nuestra comprensión de este fenómeno natural.

En el pasado, los investigadores dependían principalmente de observaciones directas y cálculos matemáticos para estudiar las estaciones. A medida que avanzaba la tecnología, se desarrollaron modelos y simulaciones por ordenador para visualizar y predecir los movimientos de los astros y las variaciones estacionales. Sin embargo, estas metodologías a menudo estaban limitadas por la falta de recursos y la dificultad para representar de manera precisa y realista los fenómenos astronómicos complejos.

La llegada de la Realidad Virtual (RV) ha abierto nuevas posibilidades para investigar y comprender los ciclos estacionales de una manera más inmersiva y experimental. La RV permite a los usuarios sumergirse en entornos virtuales tridimensionales y tener una experiencia más cercana a la realidad. La tecnología de seguimiento de movimiento y los controladores de RV permiten una interacción natural con estos entornos, lo que brinda una sensación de presencia y participación activa en el proceso de aprendizaje.

En particular, el dispositivo Oculus Quest 2 ha ganado popularidad en el campo de la RV debido a su facilidad de uso y a su capacidad para ofrecer experiencias de alta calidad sin necesidad de cables o equipos adicionales. De esta manera, los usuarios pueden disfrutar de una inmersión total en el mundo virtual, explorando entornos y objetos tridimensionales con una sensación de escala y realismo sin precedentes.

Teniendo en cuenta estos avances en la tecnología de la RV y el potencial educativo que ofrece, surge la necesidad de desarrollar herramientas interactivas que permitan a los estudiantes comprender de manera efectiva y experimental el ciclo de estaciones en la Tierra.

Teniendo en cuenta la tecnología que exploraba Cabo Gil, N. (2018), donde se hace un estudio usando un prototipo de realidad aumentada de los mismos conceptos del sistema solar, se concluye que la tecnología despierta interés entre los estudiantes y tiene un poder educativo muy elevado. Por otro lado, el mayor inconveniente es la escasa oferta de aplicaciones similares, por lo que esta tecnología resulta poco práctica para su uso ahora mismo.

Este trabajo intenta usar una tecnología superior para desarrollar un prototipo que puede ser objeto de un estudio de similares características en el campo de la investigación educativa. En este contexto, el presente trabajo de investigación se

propone aprovechar las capacidades de la RV y el Oculus Quest 2 para desarrollar una experiencia educativa inmersiva y precisa que permita a los estudiantes explorar y comprender el ciclo de estaciones en la Tierra de una manera interactiva y experimental. Mediante la simulación virtual del sistema solar, se espera proporcionar una herramienta efectiva para la enseñanza y el aprendizaje de conceptos astronómicos complejos, fomentando así un mayor interés y comprensión en el campo de la ciencia.

## 1.2. Objetivo General

El objetivo general de este trabajo es realizar una aplicación de realidad virtual que suponga una ventaja real en comparación a otros tipos de experiencias basada en sistemas multimedia interactivos. Con este fin, se explora el campo de la enseñanza de conceptos relacionados con la interacción de objetos en relación al espacio y el tiempo.

## 1.3. Objetivos Específicos

Los objetivos de este proyecto son:

- Explorar y comprender las capacidades tecnológicas del Oculus Quest SDK e Interaction SDK.
- Se pretende que el videojuego de realidad virtual sea desarrollado siguiendo las recomendaciones de uso establecidas para favorecer la experiencia en usuarios que tengan poca pericia en el ámbito de la realidad virtual.
- Lograr una visualización e interacción adecuadas para facilitar la enseñanza del ciclo de las estaciones en el planeta Tierra
- Lograr formalmente un correcto diseño y desarrollo del videojuego hasta llegar a una etapa de preproducción.

## 1.4. Estado Actual del Tema

Actualmente existen algunas herramientas y aplicaciones comerciales que se centran en el uso de la realidad virtual en el apoyo educativo.

- **Google Expeditions:** Es una herramienta de realidad virtual y aumentada que permite a docentes llevar a sus estudiantes de “viaje virtual” a lugares de interés histórico, cultural o científico. Los estudiantes pueden explorar diferentes destinos a través de dispositivos móviles y obtener una experiencia de aprendizaje enriquecedora.

- **Labster:** Es una plataforma de laboratorio virtual que brinda a los estudiantes la oportunidad de realizar experimentos científicos en un entorno seguro y virtual. Ofrece una amplia gama de laboratorios virtuales que cubren diversas disciplinas científicas y permite a los estudiantes practicar y adquirir habilidades prácticas.

- **zSpace:** Es una herramienta de realidad virtual que permite a los estudiantes interactuar con objetos virtuales en 3D utilizando un sistema de visualización y seguimiento especial. Los estudiantes pueden explorar objetos y conceptos complejos en una experiencia de aprendizaje práctica e inmersiva.

- **SimX:** SimX es una plataforma de simulación médica virtual que permite a los estudiantes de medicina y profesionales de la salud practicar procedimientos médicos en un entorno seguro y virtual. Ofrece una variedad de escenarios clínicos y permite a los usuarios mejorar sus habilidades prácticas y tomar decisiones clínicas.

Estas herramientas representan el estado del arte en el campo de la educación y la tecnología, brindando experiencias de aprendizaje inmersivas y prácticas para los estudiantes.

# Capítulo 2
# Documento de Diseño de Videojuegos

Para este trabajo, se ha realizado una aplicación que se puede considerar un juego serio. Se trata de un pequeño videojuego con el que una persona se puede iniciar en el mundo de la Realidad Virtual (VR) y al mismo tiempo que puede adquirir conocimientos sobre el sistema solar en su conjunto.

La aplicación está diseñada de tal manera que la curva de aprendizaje del dispositivo sea de poco peso, utilizando las funcionalidades que ofrece para reducir en todo lo posible el hecho de tener que aprender los controles. Esto consigue finalmente que se trate de un diseño intuitivo en su uso, incluso para alguien que no haya jugado videojuegos en realidad virtual.

En los siguientes apartados se presentarán los relativos al diseño del videojuego.

## 2.1. Título

Este videojuego serio se llama: “EarthVR”.

## 2.2. Plataforma

Este videojuego está pensado para que se ejecute en un dispositivo Oculus Quest 2 (Meta Quest 2) en modo Oculus. Esto significa que el dispositivo puede actuar de manera independiente con el videojuego instalado en su memoria interna.

## 2.3. Sinopsis de Jugabilidad y Contenido

En esta emocionante experiencia de realidad virtual, los usuarios tienen la oportunidad de explorar el sistema solar de una manera completamente inmersiva. Con el uso de tecnología de vanguardia, podrán sumergirse en un entorno tridimensional y observar de cerca cada uno de los planetas y otros cuerpos celestes.

Pero no solo se trata de una simple observación; los usuarios tienen la capacidad de interactuar con sus propias manos, sin necesidad de controladores, y así controlar, mover y rotar los planetas, así como modificar otros parámetros de la

simulación. Pueden agarrar y girar los planetas, ajustar su velocidad de movimiento e incluso cambiar su eje de rotación, permitiéndoles experimentar y comprender mejor los fenómenos astronómicos.

Además, el modo "Concurso" añade un elemento de desafío y aprendizaje interactivo. En este modo de juego, el sistema presenta preguntas que los usuarios pueden resolver de diferentes maneras. Algunas preguntas requieren seleccionar la respuesta correcta de una lista de opciones, mientras que otras implican modificar los parámetros de la simulación para crear escenarios ficticios basados en la ciencia.

## 2.4. Categoría

Este videojuego tiene cabida dentro de la categoría de juego serio, enfocado al aprendizaje del sistema solar y lo relativo a la interacción con elementos de realidad virtual.

## 2.5. Mecánica

La mecánica del videojuego se basa en el uso de un casco de realidad virtual (HMD) para permitir al usuario ver y moverse por el mundo en 3D. Además, se utiliza el seguimiento de las manos para controlar todas las interacciones lo que otorga al usuario la capacidad de interactuar con la interfaz de forma que es capaz de tocar y manipular sus elementos como si estuviera utilizando una pantalla táctil.

Una de las funciones principales es que el usuario pueda agarrar objetos virtuales y cambiar parámetros específicos utilizando únicamente sus manos. Por ejemplo, puede agarrar y girar un planeta, ajustar su velocidad de movimiento o incluso cambiar su eje de rotación.

Además, el usuario puede seleccionar diferentes astros en el espacio para realizar acciones adicionales. Por ejemplo, al apuntar y “pellizcar” un astro, puede acceder a información adicional sobre él o interactuar con él a distancia.

Del mismo modo, si el usuario está de pie, tiene la capacidad de moverse localmente dentro del entorno virtual, lo que brinda una mayor sensación de inmersión y libertad para explorar el espacio a su alrededor.

Por último, a través de la interfaz de usuario, el usuario también puede moverse por el sistema solar, lo que le permite explorar diferentes ubicaciones y astros de manera intuitiva y fácil de usar.

### 2.5.1 Agarrar

El jugador podrá acercar la mano y agarrar diversos objetos. Con esta interacción el usuario podrá mover o rotar ciertos objetos en el escenario. Para las interfaces cercanas es importante esta interacción ya que de esta manera el usuario las puede colocar donde más le interese en cada momento.

### 2.5.2 Apuntar y pellizcar

Esta interacción está pensada para interactuar con objetos lejanos. El usuario podrá apuntar con el dedo índice, y si el objeto lo soporta, sobre él se verá una representación del puntero. Cuando el usuario pellizque el dedo índice con el pulgar, hará una interacción análoga a hacer “click”. De esta manera se utilizará para interactuar con interfaces lejanas y astros fuera del alcance de las manos.

### 2.5.3 Tocar

Esta otra interacción se basa en que el usuario con el dedo índice puede tocar elementos acercando la mano al objeto y hacerlos reaccionar. Se suele utilizar sobre todo para interfaces cercanas, por lo que el usuario tendrá a su disposición una serie de elementos que podrá manipular como si usase una tableta.

## 2.6. Tecnología

El proyecto utiliza varias tecnologías de vanguardia para ofrecer una experiencia inmersiva e interactiva. Algunas de las tecnologías más relevantes utilizadas incluyen:

- **Unity**: Se emplea el motor de desarrollo de videojuegos Unity para crear y renderizar el entorno virtual. Unity ofrece herramientas poderosas y versátiles para la creación de mundos 3D, permitiendo una experiencia

visualmente impresionante y fluida. Por otro lado es un motor afianzado en la industria con salida comercial óptima para un proyecto pequeño.

- **Realidad Virtual (Oculus Quest 2)**: La experiencia se desarrolla específicamente para dispositivos de realidad virtual, en este caso, se utiliza el Oculus Quest 2. Estos dispositivos de VR permiten una inmersión total al colocar al usuario en un entorno virtual 3D, brindando una experiencia más realista y envolvente. Este dispositivo es de los más económicos del mercado en cuanto a una experiencia VR completa, ya que no es necesario el uso de un PC o dispositivo móvil adicional para ejecutar el videojuego.

- **API gráfica de Realidad Extendida (XR) (OpenXR)**: Se emplea la API OpenXR para garantizar la compatibilidad con diferentes dispositivos de realidad virtual en PC. OpenXR es un estándar de código abierto que facilita el desarrollo de aplicaciones de realidad virtual y aumentada multiplataforma, permitiendo que el proyecto sea accesible a través de diferentes dispositivos de PCVR.

- **Reconocimiento de gestos (Controles)**: Se utilizan controles con capacidad de reconocimiento de gestos para permitir que el usuario interactúe con el entorno virtual. Estos controles capturan los movimientos y gestos de las manos del usuario, lo que les permite manipular objetos virtuales, seleccionar opciones y realizar acciones en el entorno 3D sin necesidad de saber usar controles con anterioridad.

## 2.7. Escenario e Interfaces

El diseño de niveles de la aplicación se organiza de la siguiente manera: Por un lado, existe un único escenario donde ocurre toda la interacción. Por otro lado, hay presentes varias interfaces con las que el usuario tiene que interactuar para progresar o cambiar características de la simulación.

A continuación se entra en detalle en cada apartado.

### 2.7.1. Escenario

El escenario principal es el Sistema Solar, formado por los astros más importantes de los que lo conforman. Al tratarse de un escenario único, a medida que el

usuario explora los distintos modos de juego, los elementos que componen el entorno de este se van añadiendo, modificando o eliminando.

En todo caso, el usuario debe ser capaz de moverse en este escenario, ya sea manualmente, “a pie” o por medio de movimientos de cámara. Además teniendo en cuenta el enfoque educativo de este proyecto, se hace necesario introducir el “teletransporte” como herramienta que permite moverse entre astros de forma más ergonómica e inmediata.

### 2.7.2. Interfaces:

Hay varias interfaces que el usuario puede utilizar. Las más comunes son interfaces planas, donde el usuario puede pellizcar o tocar directamente.

Estas interfaces se integran en el juego para brindar una experiencia intuitiva y fácil de usar. El menú principal permite elegir entre los modos de juego disponibles, mientras que el menú in-game y los tooltips ofrecen opciones de control y acceso a información adicional. Mientras que el modo de juego Concurso proporciona una experiencia desafiante donde los usuarios pueden poner a prueba su conocimiento científico y habilidades de modificación del sistema solar.

Como una de las limitaciones más importantes del seguimiento de manos es la oclusión de éstas, el diseño de la interacción será de tal manera que ocurra siempre hacia delante del usuario, donde están las cámaras del dispositivo.

A continuación se explica en detalle cada una

- **Menú principal (Figura 2.1):** Esta interfaz aparece al iniciar el juego y permite a los usuarios seleccionar entre los diferentes modos de juego disponibles. También ofrece la opción de configurar parámetros de accesibilidad para adaptar la experiencia a las necesidades individuales.

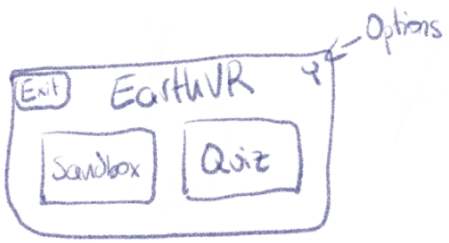


Figura 2.1: Boceto de menú principal.

- **Menú in-game (Figura 2.2):** Esta interfaz se activa al presionar el botón de pausa durante el juego. Permite a los usuarios controlar ciertos parámetros de la aplicación, cómo la cámara o la posición de la Tierra. También ofrece la opción de regresar al menú principal o reiniciar el modo de juego actual.

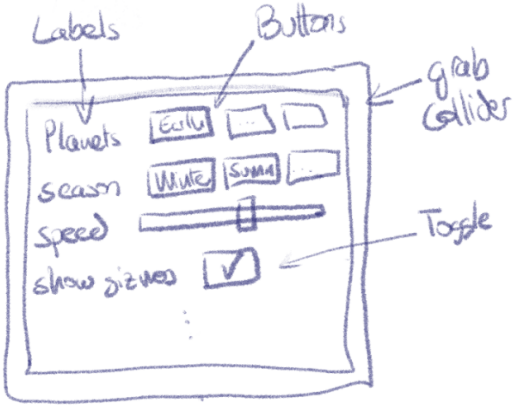


Figura 2.2: Boceto de menú in-game.

- **Tooltip (Figura 2.3):** Cuando los usuarios seleccionan un astro pellizcando, se muestra un desplegable cercano a su ubicación con información adicional. Este tooltip proporciona detalles relevantes sobre el planeta seleccionado.

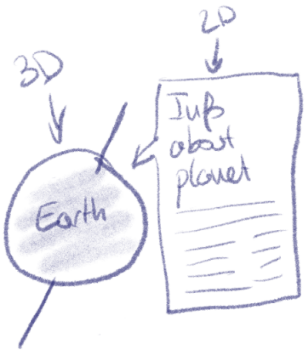


Figura 2.3: Boceto de tooltip.

- **Concurso (Figura 2.4):** Esta interfaz se presenta durante el modo de juego Concurso. En ella se presentan una serie de acertijos que el usuario deberá responder. No obstante, la solución no siempre será introducir una respuesta o seleccionarla entre una lista de opciones, sino que podrá ser necesario que explore la configuración de los elementos del escenario y modifique sus parámetros para lograr el entorno ficticio que responda a la pregunta planteada.

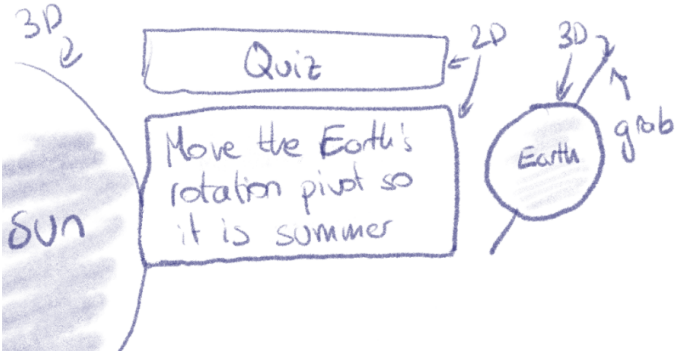


Figura 2.4: Boceto de modo quiz.

## 2.8. Público

El público objetivo de este videojuego son principalmente jóvenes y personas que no hayan tenido ninguna experiencia en Realidad Virtual. Por otro lado, también es de utilidad para docentes y estudiantes.

# Capítulo 3
# Desarrollo

Para el desarrollo de la aplicación se ha usado el motor gráfico Unity 2020 LTS. Para ello tenemos que iniciar un proyecto nuevo e integrar los SDK y plugins necesarios para soportar la plataforma de Oculus Quest 2. El desarrollo se ha basado principalmente en las guías de desarrolladores de Oculus y en la documentación de las SDK.

## 3.1. Proyecto de Unity

Antes de nada, creamos un proyecto 3D vacío de Unity. Luego tenemos que instalar los siguientes componentes:

- Unity XR Plugin Management package
- OpenXR/OculusXR Plugin
- OpenXR Oculus Interaction profile
- Meta's Integration SDK

En los siguientes apartados se entrará en detalle el uso que se le da a cada componente y cómo se instala. Una vez todos instalados, el entorno de desarrollo está listo para trabajar.

### 3.1.1. El paquete "XR Plugin Management"

El paquete "XR Plugin Management" es una herramienta esencial para la gestión y configuración de plugins relacionados con la realidad extendida (XR) en los proyectos de Unity. Proporciona una interfaz unificada y una serie de funcionalidades para administrar eficientemente los plugins XR utilizados en un proyecto.

Este paquete está disponible directamente desde el Package Manager (administrador de paquetes) en el registro de Unity.

### 3.1.2. XR Plugins

Los plugins de XR son componentes adicionales que se integran en Unity para proporcionar funcionalidades específicas relacionadas con XR, como soporte para plataformas de realidad virtual (VR), realidad aumentada (AR) o realidad mixta (MR). Algunos plugins XR comunes incluyen las funcionalidades correspondientes a las plataformas de realidad virtual, como Oculus, SteamVR, Windows Mixed Reality, entre otros.

Para este proyecto vamos a instalar dos plugins, OculusXR y OpenXR. OculusXR es el plugin del dispositivo, usa el backend de OpenXR y nos permite ejecutar la aplicación en modo Oculus standalone directamente en el dispositivo sin necesidad de un PC para ejecutar la aplicación. El plugin de OpenXR por otro lado también nos da acceso al backend de OpenXR, pero en PC, y es necesario para ejecutar la aplicación en modo PCVR, siendo de gran ayuda para el desarrollo.

Se instalan directamente desde el Package Manager de Unity o desde la ventana de XR Plug-in Management (Figura 3.1).

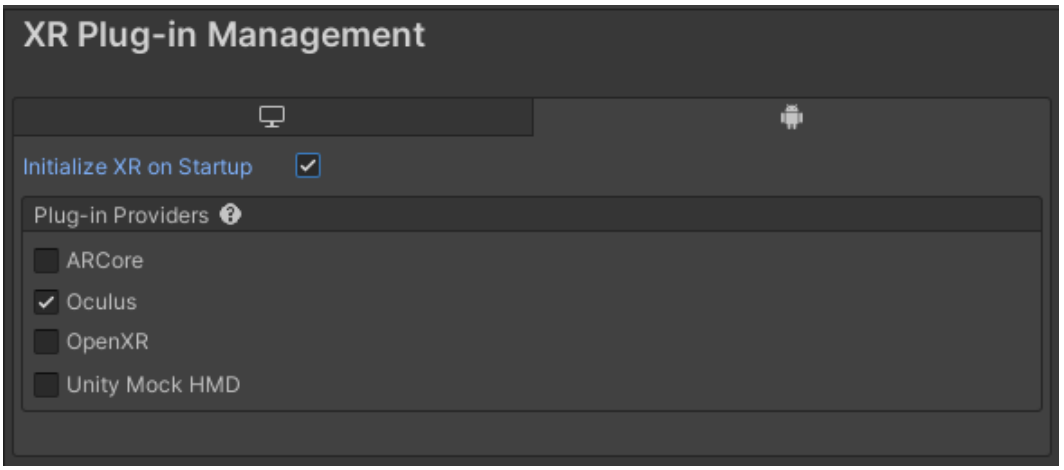


Figura 3.1: Ventana XR Plug-in Management en Unity.

### 3.1.3. OpenXR Oculus Interaction profile

En OpenXR, la lista de "interaction profiles" en la configuración de Unity se utiliza para definir los perfiles de interacción admitidos por un sistema de realidad virtual

(VR). Estos perfiles describen los dispositivos de entrada que pueden ser utilizados por el usuario para interactuar con el entorno virtual.

Cada dispositivo de entrada en realidad virtual tiene su propio conjunto de capacidades y características específicas. Por ejemplo, un controlador de mano puede tener botones, palancas analógicas y seguimiento de posición y orientación, mientras que un controlador de cabeza puede tener sensores de movimiento y un botón de activación.

Aunque para esta aplicación no se vayan a usar los controladores, internamente el sistema que ofrece la Oculus Integration SDK utiliza el mismo esquema de controles, por lo que, necesitamos añadir el "Oculus Touch Controller Profile" a la lista de perfiles (ver figura 3.2).

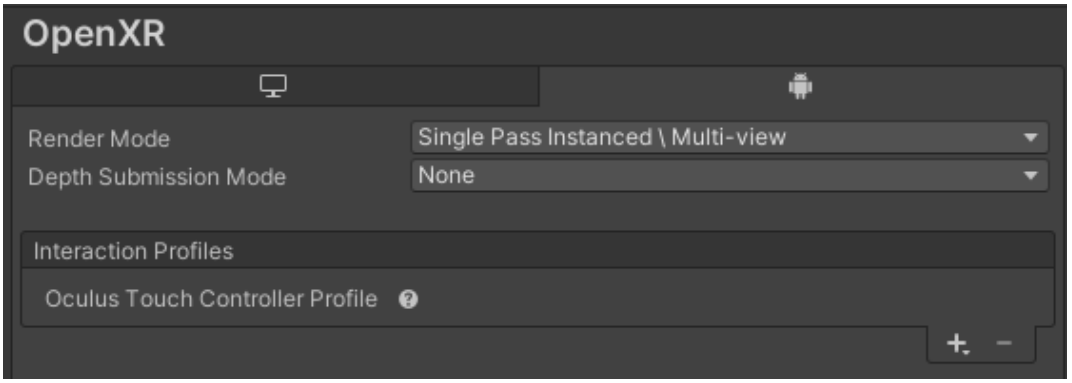


Figura 3.2: Ventana OpenXR Plugin en Unity.

## 3.1.4. Oculus Integration SDK

Esta SDK, desarrollada por Meta, se trata de un paquete de Unity, que proporciona una serie de APIs y herramientas que facilitan la integración de los dispositivos Meta con Unity. Estas herramientas incluyen prefabs, scripts y ejemplos de código que son útiles para empezar a desarrollar en dispositivos de Realidad Virtual.

En principio este SDK soporta las funcionalidades del Meta Quest 2, pero la mayoría de funciones al ser comunes con otros visores de RV, funcionan en otros visores si se configuran correctamente para multiplataforma.

En este proyecto se están usando tanto la funcionalidad de Oculus Quest en modo PCVR como en modo Oculus standalone. El modo PCVR es de vital

importancia para el desarrollo, ya que supone una ventaja en velocidad de desarrollo y prueba en el dispositivo, respecto al modo Oculus. El modo Oculus por otro lado se considerará como el modo de producción, ya que es la plataforma objetivo en la que se ejecutará el programa.

## 3.2. Oculus Interaction SDK

El "Interaction" SDK es una parte fundamental del Oculus Integration SDK, el cual incluye varios SDK como el Interaction SDK, Voice SDK y Platform SDK, entre otros componentes. Dentro del Interaction SDK, encontramos una variedad de prefabs, scripts y ejemplos de configuración relacionados con la interacción de controles y manos en realidad virtual.

Siguiendo los pasos a continuación, lograremos configurar adecuadamente la interacción con los controles y manos en el entorno de realidad virtual, brindando una experiencia más inmersiva y realista para los usuarios.

Para comenzar a utilizar esta herramienta, el primer paso es configurar el objeto "OVRManager" en la escena. Ajustamos el parámetro "Tracking Origin Type" a "Floor Level" y luego agregamos el prefab llamado "OVRInteraction" en la jerarquía del "OVRCameraRig" como hijo a la raíz (ver figura 3.3).

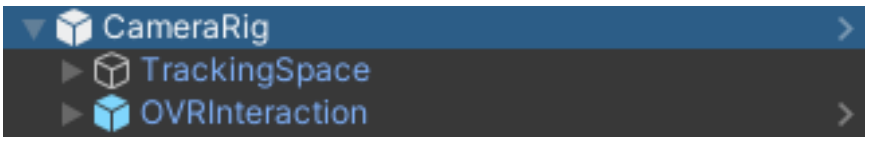


Figura 3.3: OVRCameraRig con el nuevo OVRInteraction añadido.

### 3.2.1 Manos como controladores

Una vez configurado el "OVRCameraRig", procedemos a establecer las manos como controles interactivos. Para hacer esto, habilitamos el soporte de manos en el componente "OVRManager" y agregamos el prefab "OVRHandPrefab" en cada pivote de mano correspondiente del "OVRCameraRig", llamados “RightHandAnchor” y “LeftHandAnchor” (ver figura 3.4). Por último, deshabilitamos la malla y los renderizadores de las manos que acabamos de añadir (ver figura 3.5).


Figura 3.4: OVRHandPrefab en cada pivote.

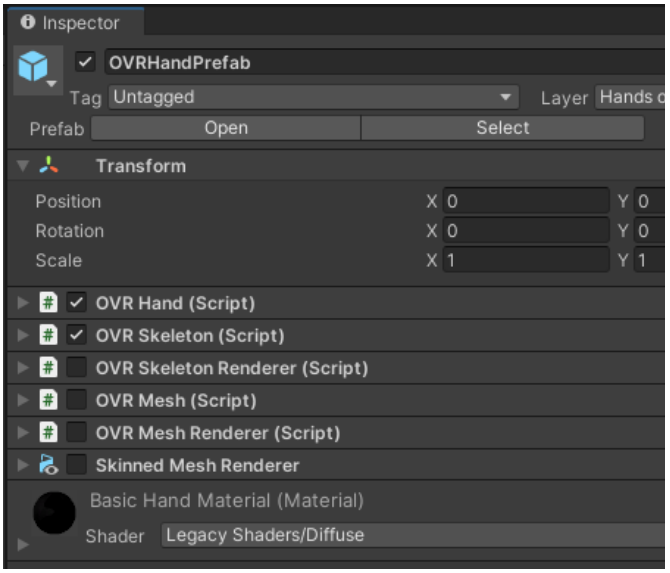


Figura 3.5: Mallas y renderizadores deshabilitados en los OVRHandPrefabs

Es importante destacar que hasta este punto, los prefabs que hemos agregado solo representan el modelo de interacción con las manos, pero aún no tienen apariencia visual ni interacción. Dado esto, añadimos el prefab llamado "OVRHands" en la jerarquía del objeto "OVRInteraction" (ver figura 3.6). De esta manera, podremos ver las manos en acción cuando tengamos puesto el casco de realidad virtual (HMD).

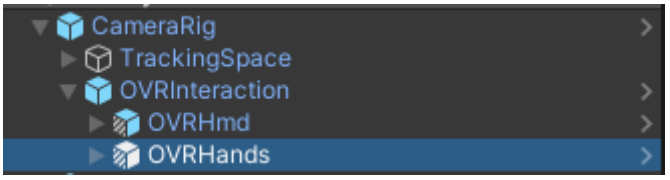


Figura 3.6: OVRHands como hijo en la jerarquía de OVRInteraction.

## 3.2.2 Interacción de las Manos

En el estado actual, tenemos una representación de las manos, pero éstas todavía no pueden interactuar con los objetos del mundo. Para solucionar esto, vamos a

darle la capacidad de agarrar, pellizcar y de dar un toque a los distintos objetos del mundo.

Para darle comportamiento a las manos, vamos al componente “OVRHands” que hemos añadido anteriormente y tanto en el componente de la mano izquierda como en la derecha, dentro existe otro par de componentes “HandInteractorLeft” y “HandInteractorRight”. A estos le tenemos que añadir implementaciones de IInteractor.

La IntegrationSDK nos incluye varias implementaciones, que vamos a usar en este paso. Para darle el comportamiento de agarrar, añadimos como hijo en la jerarquía a los componentes anteriormente mencionados un prefab llamado “HandGrabInteractor”(ver figura 3.7). Por último en el componente padre, añadimos una referencia del interactor que acabamos de añadir a la lista de “interactors” (ver figura 3.8). Repetimos el mismo procedimiento para añadir el comportamiento de tocar, esta vez, añadiendo el prefab “HandPokeInteractor” y para añadir el comportamiento de “Ray”, repetimos con “HandRayInteractor”.

Recapitulando, en la jerarquía de “HandInteractorLeft” y “HandInteractorRight” debemos de tener como hijos estos tres prefabs: “HandGrabInteractor”, “HandPokeInteractor” y “HandRayInteractor”, además de añadirlos a la lista de “interactors” en cada padre.


Figura 3.7: HandInteractorsLeft con los tres interactors en su jerarquía.

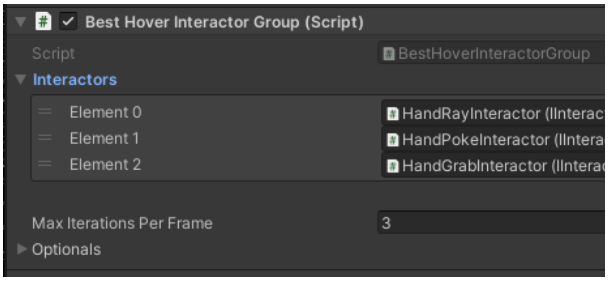


Figura 3.8: Inspector de HandInteractorsLeft con los interactors asignados en la lista.

Con esta configuración, las manos están listas para poder interactuar con los elementos que a continuación haremos responder a estos comportamientos.

## 3.3 Maqueta del sistema solar

Para la maqueta del sistema solar, se han creado varios prefabs y varios scripts para darle comportamiento personalizado. Hay un prefab por cada astro, uno para la cámara y otros para componentes reutilizables. En los siguientes apartados se entra en detalle sobre los scripts y prefabs más importantes:

### 3.3.1 Game Manager

Este script se encarga de enlazar todos los astros de la escena y los demás componentes que existen para controlarlos según el modo de juego. Por un lado se encarga de inicializar todos los componentes en cuanto empieza la aplicación (listing 3.1) y también es el que responde a los eventos de las interfaces.

```
private void InitializeAllStars(float scale) {
  foreach (var planet in planets) {
    planet.Initialize(new Planet.InitializationParameters {
      sun = sun,
      planetScale = scale
    });
  }

  moon.Initialize(new Planet.InitializationParameters {
    sun = earth.gameObject,
    planetScale = scale
  });
}
```

Listing 3.1: Función InitializeAllStars en GameManager.

### 3.3.2 Camera Rig

Este script se encarga de controlar la cámara, para poder desplazarse entre los diferentes pivotes y la gestión de transiciones. Esta clase se la asignaremos al objeto raíz de la “OVRCameraRig”, de esta manera podemos mover la cámara del HMD.

En cuanto a las transiciones, al hacer un cambio de cámara, el componente CameraFadeToBlack (ver listing 3.2) se encarga de cambiar la opacidad de OVRScreenFade, que intermitente gestiona un fundido a negro.

```
public class CameraFadeToBlack : MonoBehaviour {
  [SerializeField] private OVRScreenFade screenFade;
  [SerializeField] private float duration;

  public void Transition(
    Action midTransitionCallback,
    Action endAnimationCallback = null) {
      StartCoroutine(MakeTransition(
        midTransitionCallback,
        endAnimationCallback));
  }

  private IEnumerator MakeTransition(
    Action callback,
    Action endAnimationCallback) {
      yield return Fade(0f, 1f, duration / 2);
      callback();
      yield return Fade(1f, 0f, duration / 2);
      endAnimationCallback?.Invoke();
  }

  private IEnumerator Fade(float start, float end, float duration) {
    var value = start;
    var time = 0.0f;
    while (time < duration) {
      value = Mathf.Lerp(start, end, time / duration);
      screenFade.SetUIFade(value);

      yield return null;
      time += Time.deltaTime;
    }
  }
}
```

Listing 3.2: CameraFadeToBlack.

### 3.3.3 Planet

Este script se encarga de todo lo relativo a un planeta. Esta clase define a los planetas del escenario y es el intermediario entre GameManager y los parámetros de cada planeta. Por un lado gestiona la rotación sobre sí mismo y la traslación alrededor del sol. Para eso, al inicio primero computa el ángulo que tiene que girar y recorrer en cada ciclo de update (listing 3.3 y 3.4)

```
private void ComputeValues(
  float yearDurationInSeconds,
  float dayDurationInSeconds) {
    orbitAngle =  -1.0f / period * 360.0f / yearDurationInSeconds;
    ownRotationAngle = -1.0f /
      ownRotationPeriod * 360.0f / dayDurationInSeconds;
}
```

Listing 3.3: Cómputo de valores para el ciclo Update

```
private void Orbit() {
  translationPivot.RotateAround(sun.transform.position,
    Vector3.up, orbitAngle * Time.deltaTime);
  translationPivot.eulerAngles = Vector3.zero;

  if (updateSunPositionOnOrbit) {
    translationPivot.position =
      (translationPivot.position - sun.transform.position).normalized *
      distanceFromSun + sun.transform.position;
  }
}

private void Rotate() {
  rotationPivot.Rotate(rotationPivot.up,
    ownRotationAngle * Time.deltaTime, Space.World);
}
```

Listing 3.4: Funciones de rotación y traslación que se usan en el ciclo Update.

Por otro lado mantiene la representación del eje de rotación para poder modificarla con la interacción de agarrar. Para ello, tenemos una configuración donde la malla de la Tierra está en un pivote y la representación del eje de rotación en otro (ver figura 3.9), además de luego actualizar el de La Tierra en LateUpdate mediante el script de “FollowRotationTransform” (ver figura 3.10). De esta manera separamos los conceptos que gestiona cada transform en Unity.

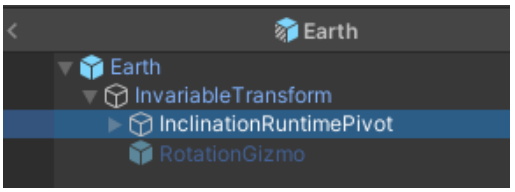


Figura 3.9: Jerarquía del Prefab Earth.

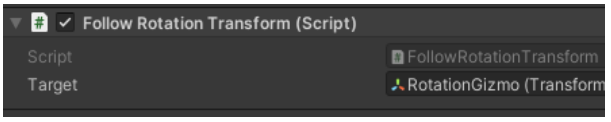


Figura 3.10: FollowRotationTransform en InclinationRuntimePivot.

Para configurar el comportamiento de agarrar en la representación del eje de rotación, primero hay que agregar un collider con “isTrigger” y un rigidbody sin uso de la gravedad. De esta manera el sistema de Interaction podrá mover este objeto con los siguientes componentes. A continuación agregamos el componente “Hand Grab Interactable” y “Grabbable” (ver figuras 3.11 y 3.12), donde podremos configurar cómo se realiza el agarre. Podemos configurar qué dedos inician y terminan un agarre, si es válido agarrar pellizcando, o si por el contrario tiene que agarrar con la palma de la mano, etc.

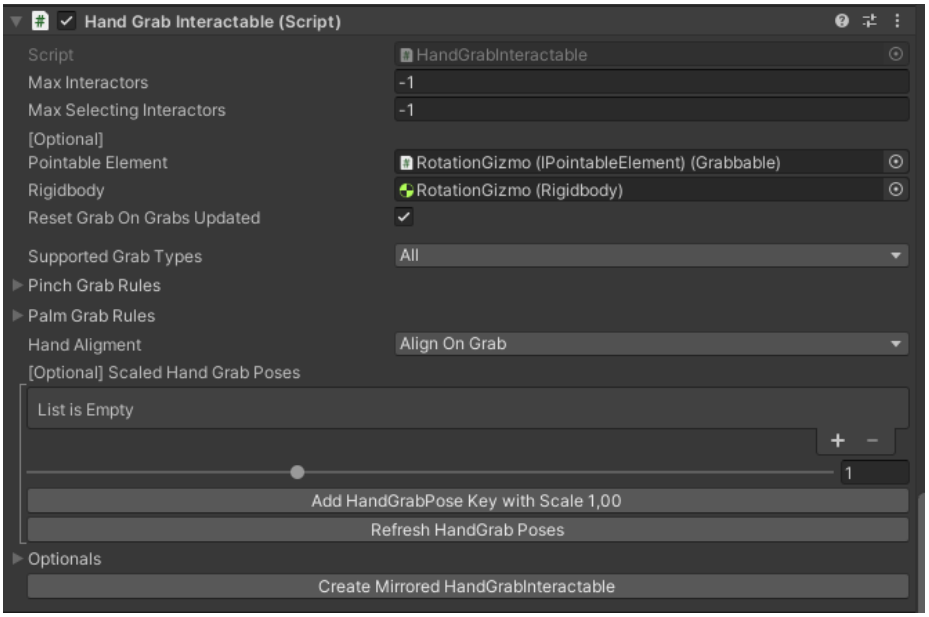


Figura 3.11: Componente HandGrabInteractable en la representación del eje de rotación.

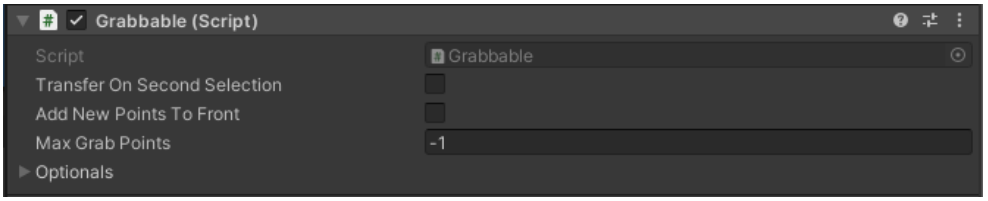


Figura 3.12: Componente Grabbable en la representación del eje de rotación

Para este objeto, se dejará el comportamiento por defecto y se agrega otro componente "One Grab Rotate Transformer" (ver figura 3.13) que nos dará la posibilidad de fijar o bloquear un eje de rotación cuando estemos agarrando el objeto. Esto es de particular importancia ya que no queremos que el eje de rotación se mueva en todos los ejes posibles, sino que únicamente pueda rotar en relación al eje "Right".

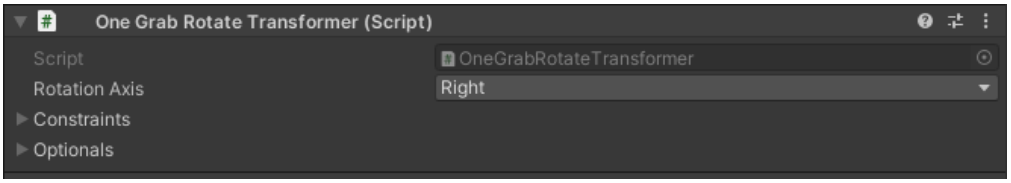


Figura 3.13: Componente OneGrabRotateTransformer en la representación del eje de rotación

En el caso especial de la Tierra, también gestiona otro componente llamado EarthPoints, el cual se encarga de mostrar unos puntos en la superficie de la Tierra (ver figura 3.14), que corresponden a algunas ciudades del planeta.

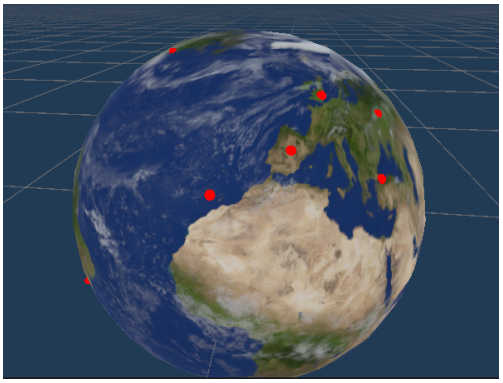

Figura 3.14: Puntos en la superficie del planeta Tierra.

### 3.3.4 Texturas y Modelos

Todas las texturas se han obtenido de Solar System Scope, el cual contiene texturas de varios astros del sistema solar en proyección cilíndrica equidistante en formato .jpg y .tiff de resoluciones 2k y 8k. Desafortunadamente no contiene texturas de todos los astros del sistema solar, pero con los más importantes es suficiente para este proyecto.

Para el skybox de la escena, se ha utilizado una de las texturas de fondo estrellado disponibles en la misma fuente, además se usa la misma fuente de luz del sol para el sistema de iluminación de Unity (ver figura 3.15).


Figura 3.15: Configuración de Skybox en el menú de Iluminación de Unity.

Por otro lado, todos los modelos 3D han sido generados en Unity automáticamente ya que se tratan de simples esferas, cubos y materiales simples.

## 3.4 Interfaces y sistemas de control

Hay dos tipos de interfaces con las que puede interactuar el jugador en la aplicación, las que están lejos y las que están cerca. Las lejanas se pueden controlar con el gesto de apuntar en la dirección de la interfaz y pellizcar. Las cercanas se pueden controlar pulsando directamente sobre un control, como si fuera un objeto físico (ver figura 3.16). Las interfaces lejanas son las del Menú Principal, y la interfaz del modo Concurso. La interfaz cercana es la del menú de Pausa.

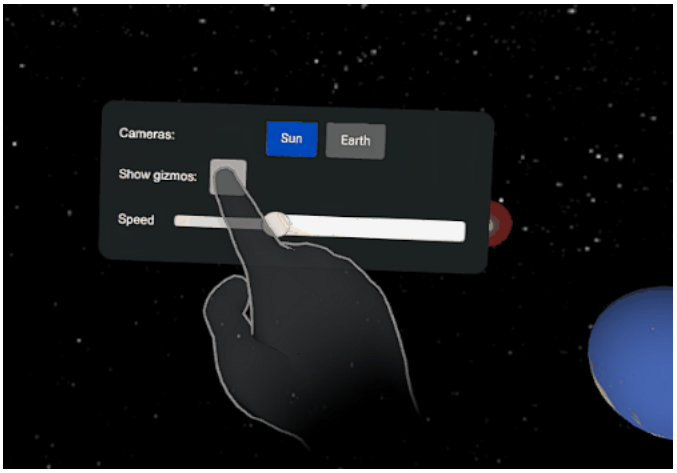


Figura 3.16: Interacción con una interfaz cercana

### 3.4.1 Interfaces

Para desarrollar estas interfaces, se ha usado el sistema de Canvas de Unity, y luego se han adaptado para poder usarlas con la Interaction SDK de Meta. De esta forma podemos realizar una interfaz estándar y darle el comportamiento esperado en este tipo de aplicación.

Para adaptarlas a que se puedan usar con las manos se ha seguido el siguiente procedimiento:

1. Crear interfaz con diseño adecuado a realidad virtual y uso con manos.
2. Para las interfaces cercanas, le damos comportamiento de tocar y agarrar, para las interfaces lejanas, le damos comportamiento de pellizcar.
3. Por último, añadimos un control de posicionamiento relativo a la posición del CameraRig

En cuanto al comportamiento de las interfaces, se hace de manera similar a lo realizado anteriormente. Para el comportamiento de agarre, hay que añadir colliders con “isTrigger” y rigidbody sin uso de gravedad. Luego añadimos los componentes “Grabbable” y “Hand Grab Interactable”.

Para el comportamiento de tocar, añadimos varios componentes; el primero es uno llamado “PointableCanvas” junto con “PointableCanvasUnityEventWrapper”.

Por último añadimos “PokeInteractable” (ver figura 3.17). Ahora podemos configurar el comportamiento, pero por suerte el comportamiento por defecto funciona muy bien con las interfaces de Unity, si hacemos las interfaces de manera estándar.

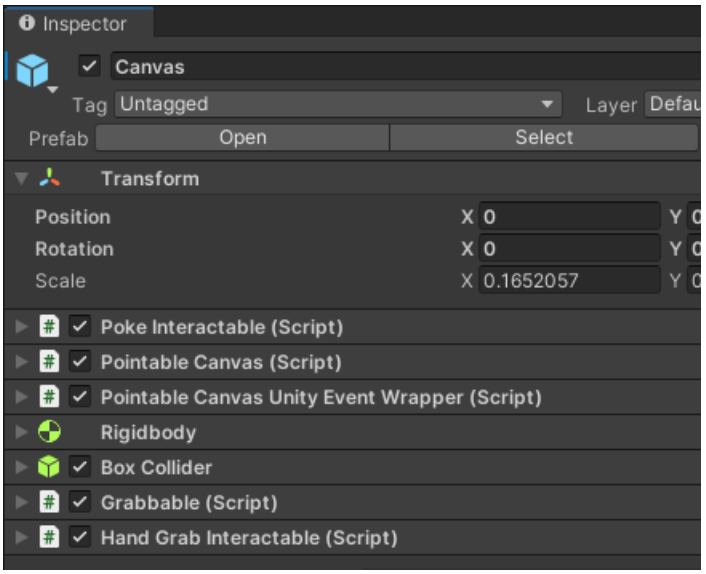


Figura 3.17: Configuración de una interfaz cercana

## 3.4.2 Sistemas de control

En cuanto al ciclo de vida de la aplicación, hay un componente llamado ApplicationLifecycle que al iniciarse ejecuta el menú principal con una pequeña animación y éste permite elegir el modo de juego o salir.

Al seleccionar un modo de juego, este componente cargará la configuración inicial del modo escogido mediante el GameManager.

Para el modo de juego Sandbox, el GameManager inicializará el escenario con todos los astros y pondrá al usuario en la posición del sol. Después de eso, empezará a ejecutar la simulación a razón de 3600 segundos para 1 vuelta de la Tierra alrededor del Sol. Este parámetro es configurable mediante el objeto

GameParameters (ver figura 3.18) o en tiempo de ejecución mediante el menú de pausa.

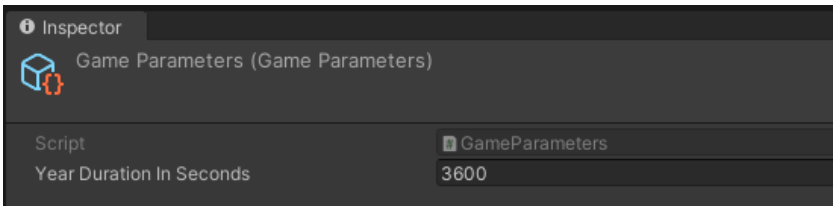


Figura 3.18: Game Parameters con su configuración inicial.

En este modo, el usuario tendrá todas las opciones habilitadas excepto las del modo Concurso, por lo que podrá desplazarse entre los planetas, modificar sus propiedades y cambiar la velocidad de la simulación (ver figura 3.19). Mientras el jugador esté en este modo, se puede desplazar y mover libremente y para salir, tendrá que usar el gesto "pausa" y salir al menú principal.

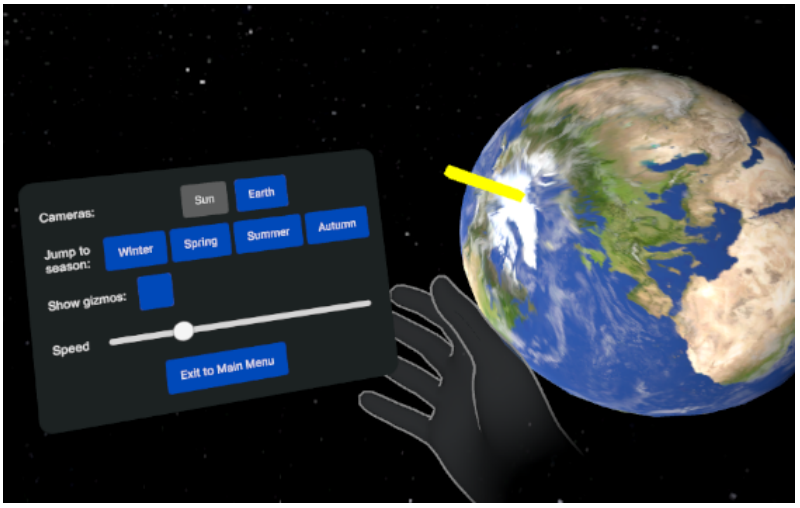


Figura 3.19: Modo sandbox, junto a La Tierra

En cuanto al modo de juego Concurso, el GameManager inicializará el escenario sólo con el Sol, La Tierra y La Luna. Después pondrá la cámara cerca de La Tierra y empezará a hacer preguntas. El usuario en este modo no tiene todas las opciones disponibles y sólo podrá salir al menú principal o contestar las preguntas (ver figura 3.20)


Figura 3.20: Modo Concurso, en mitad de una pregunta.

# Conclusiones y Líneas futuras

En conclusión, se puede afirmar que la realidad virtual ofrece una experiencia más inmersiva en comparación con otras tecnologías de realidad extendida. Esta característica confiere a la realidad virtual una clara ventaja educativa sobre los sistemas convencionales. Además, el dispositivo en concreto de realidad virtual analizado destaca por su asequibilidad, lo cual representa una ventaja adicional.

Aunque la maqueta actualmente no está lista para su uso educativo, se puede destacar que mediante funcionalidades adicionales es muy posible que funcione de manera excelente en ese ámbito. En concreto una mejora en cuanto a el número de modos de juego y otras funcionalidades como información adicional al respecto de lo que se está interactuando, pueden suponer una forma muy cómoda de explorar conceptos complejos de manera experimental.

Sin embargo, es importante mencionar como desventaja que algunas personas pueden experimentar mareos al utilizar este sistema, especialmente aquellos que no están acostumbrados a este tipo de tecnología. No obstante, se ha tratado de mitigar este efecto incorporando un diseño que permite una experiencia más tolerable y menos propensa a causar mareos.

En el transcurso de este trabajo de Fin de Máster, se ha adquirido un amplio conocimiento sobre la realidad virtual y sus aplicaciones comerciales. Este aprendizaje ha permitido comprender en mayor profundidad las ventajas y desafíos asociados con esta tecnología, así como su potencial educativo y su impacto en diversos sectores industriales.

# Summary and Conclusions

In conclusion, it can be stated that virtual reality offers a more immersive experience compared to other extended reality technologies. This immersive feature gives virtual reality a clear educational advantage over conventional systems. Furthermore, the specific virtual reality device analyzed stands out for its affordability, which represents an additional advantage.

Although the current prototype is not yet ready for educational use, it can be highlighted that, with additional functionalities, it is very likely to perform excellently in that field. Specifically, an improvement in the number of game modes and other features such as additional information about the interaction can provide a very convenient way to explore complex concepts experimentally.

However, it is important to mention as a disadvantage that some people may experience motion sickness when using this system, especially those who are not accustomed to this type of technology. Nevertheless, efforts have been made to mitigate this effect by incorporating a design that allows for a more tolerable and less dizziness-inducing experience.

Throughout this Master's thesis, a broad understanding of virtual reality and its commercial applications has been acquired. This learning has allowed for a deeper comprehension of the advantages and challenges associated with this technology, as well as its educational potential and its impact on various industrial sectors.

# Bibliografía

Cabo Gil, N. (2018). *La realidad aumentada como técnica para la mejora de la adquisición de conocimientos en educación infantil*.

Solar System Scope. (n.d.). https://www.solarsystemscope.com/textures/

Designing for hands | Oculus developers. (n.d.). https://developer.oculus.com/resources/hands-design-intro/

Oculus documentation | Oculus developers. (n.d.). https://developer.oculus.com/documentation/

# Apéndice

## Planet.cs

```
public class Planet : MonoBehaviour {

    [SerializeField] private bool updateSunPositionOnOrbit;

    [Header("Orbit")]
    [SerializeField] private float distanceFromSun;
    [SerializeField] [Tooltip("In years relative to Earth")] private
float period;
    [SerializeField] private Transform translationPivot;

    [Header("Rotation")]
    [SerializeField] [Tooltip("In degrees relative to Sun's Equator")]
private float inclination;
    [SerializeField] private float ownRotationPeriod;
    [SerializeField] private Transform rotationPivot;

    [Header("Others")]
    [SerializeField] private Transform invariableTransform;
    [SerializeField] private EarthPoints earthPoints;

    [Header("Gizmos")]
    [SerializeField] private GameObject rotationAxisGizmo;

    private Vector3 initialPosition; // this position is Winter (north
hemisphere)
    private bool transformInitialized;

    public bool GizmosEnabled {
        get => gizmosEnabled;
        set {
            gizmosEnabled = value;
            if (value) {
                EnableRuntimeGizmos();
            }
            else {
                DisableRuntimeGizmos();
```

```
            }
        }
    }

    private GameObject sun;

    private float ownRotationAngle;

    private float orbitAngle;

    private bool isRunning;

    private Vector3[] debugOrbitPoints;
    private bool gizmosEnabled = false;

    public void Initialize(InitializationParameters
initializationParameters) {
        this.sun = initializationParameters.sun;

        var size = initializationParameters.planetScale;
        transform.localScale = new Vector3(size, size, size);

        ComputeSunDependantValues();
        SetInitialTransform();
    }

    public void SetParameters(float yearDurationInSeconds, float
dayDurationInSeconds) {
        ComputeValues(yearDurationInSeconds, dayDurationInSeconds);
    }

    public void PauseMovement() {
        isRunning = false;
    }

    public void ResumeMovement() {
        isRunning = true;
    }

    public void StartMovement() {
        isRunning = true;
    }
```

```
    public void MoveToSeason(Season season) {
        translationPivot.position = initialPosition;
        switch (season) {
            case Season.Spring:
                translationPivot.RotateAround(sun.transform.position,
Vector3.up, 90);
                break;
            case Season.Summer:
                translationPivot.RotateAround(sun.transform.position,
Vector3.up, 180);
                break;
            case Season.Autumn:
                translationPivot.RotateAround(sun.transform.position,
Vector3.up, 270);
                break;
        }
        translationPivot.eulerAngles = Vector3.zero;
    }

    public void ShowRandomPoint() {
        var point = earthPoints.GetPoint();
    }

    private void Update() {

        var drawGizmosUnityEditor = false;

#if UNITY_EDITOR
        drawGizmosUnityEditor =
SceneView.lastActiveSceneView.drawGizmos;
#endif

        if (GizmosEnabled || drawGizmosUnityEditor) {
            if (updateSunPositionOnOrbit) {
                ComputeSunDependantValues();
            }
        }

        if (!isRunning) {
            return;
        }

        Orbit();
```

```
        Rotate();
    }

    private void SetInitialTransform() {
        if (transformInitialized) {
            translationPivot.position = initialPosition;
            return;
        }

        translationPivot.position = (translationPivot.position -
sun.transform.position).normalized * distanceFromSun +
sun.transform.position;

        var rotation = invariableTransform.eulerAngles;
        rotation.x = inclination;
        invariableTransform.eulerAngles = rotation;

        initialPosition = translationPivot.position;
        transformInitialized = true;
    }

    private void ComputeValues(float yearDurationInSeconds, float
dayDurationInSeconds) {
        orbitAngle =  -1.0f / period * 360.0f / yearDurationInSeconds;
        ownRotationAngle = -1.0f / ownRotationPeriod * 360.0f /
dayDurationInSeconds;
    }

    private void ComputeSunDependantValues() {
        debugOrbitPoints = new Vector3[360];
        for (var i = 0; i < 360; i++) {
            var angle2 = i * Math.PI / 180.0f;
            debugOrbitPoints[i] = new Vector3(distanceFromSun * (float)
Math.Cos(angle2), 0, distanceFromSun * (float) Math.Sin(angle2)) +
sun.transform.position;
        }
    }

    private void Orbit() {
        translationPivot.RotateAround(sun.transform.position,
Vector3.up, orbitAngle * Time.deltaTime);
        translationPivot.eulerAngles = Vector3.zero;
```

```
        if (updateSunPositionOnOrbit) {
            translationPivot.position = (translationPivot.position -
sun.transform.position).normalized * distanceFromSun +
sun.transform.position;
        }
    }

    private void Rotate() {
        rotationPivot.Rotate(rotationPivot.up, ownRotationAngle *
Time.deltaTime, Space.World);
    }

    private void OnDrawGizmos() {

        if (debugOrbitPoints == null) {
            return;
        }

        Gizmos.color = Color.white;

        for (var i = 0; i < debugOrbitPoints.Length; i++) {
            var nextPointIndex = i == debugOrbitPoints.Length - 1 ? 0 :
i + 1;

            var point = debugOrbitPoints[i];
            var point2 = debugOrbitPoints[nextPointIndex];
            Gizmos.DrawLine(point, point2);
        }
    }

    private void EnableRuntimeGizmos() {
        rotationAxisGizmo.gameObject.SetActive(true);
    }

    private void DisableRuntimeGizmos() {
        rotationAxisGizmo.gameObject.SetActive(false);
    }

    public struct InitializationParameters {
        public GameObject sun;
        public float planetScale;
    }
```

```
    public enum Season {
        Winter,
        Spring,
        Summer,
        Autumn
    }
}
```

## ApplicationLifecycle.cs

```
namespace SolarSystem {
      public class ApplicationLifecycle : MonoBehaviour {

            [SerializeField] private GameManager gameManager;
            [SerializeField] private CameraRig cameraRig;

            [Header("Menus")]
            [SerializeField] private GameObject mainMenuCanvas;
            [SerializeField] private AnimationCurve
mainMenuAnimationCurve;

            private void Start() {
                  ShowMainMenu();
            }

            public void StartSandboxMode() {
                  gameManager.Initialize();

                  cameraRig.SetCamera(CameraPivot.Sun, () => {
                        mainMenuCanvas.SetActive(false);
                        gameManager.InitializeSimulation();
                  });
            }

            public void StartQuizMode() {
                  gameManager.Initialize();

                  cameraRig.SetCamera(CameraPivot.Earth, () => {
                        mainMenuCanvas.SetActive(false);
                        gameManager.InitializeSimulation();
                  }, () => {
```

```
                        gameManager.StartQuiz();
                });
        }

        private void ShowMainMenu() {
                gameManager.Disable();
                mainMenuCanvas.SetActive(true);

                OVRManager.HMDMounted += HandleHMDMounted;
                StartCoroutine(MoveUIAnimation());
        }

        private void HandleHMDMounted() {
                StartCoroutine(MoveUIAnimation());
        }

        private IEnumerator MoveUIAnimation() {

                var targetPosition =
cameraRig.CenterEyeTransform.position +
cameraRig.CenterEyeTransform.forward * 1;
                var initialPosition =
mainMenuCanvas.transform.position;

                var initialYEulerAngles =
mainMenuCanvas.transform.eulerAngles.y;
                var targetYEulerAngles =
cameraRig.CenterEyeTransform.eulerAngles.y;

                var duration = 1f;
                var time = 0f;

                while (time < duration) {

                        var lerpIndex =
mainMenuAnimationCurve.Evaluate(time / duration);

                        var newPosition =
Vector3.Lerp(initialPosition, targetPosition, lerpIndex);
                        mainMenuCanvas.transform.position =
newPosition;

                        var newYEulerAngles =
```

```
Mathf.Lerp(initialYEulerAngles, targetYEulerAngles, lerpIndex);
                        var newEulerAngles =
mainMenuCanvas.transform.eulerAngles;
                        newEulerAngles.y = newYEulerAngles;
                        mainMenuCanvas.transform.eulerAngles =
newEulerAngles;

                        yield return null;
                        time += Time.deltaTime;
                    }
                }

        }
}
```

## GameManager.cs

```
namespace SolarSystem {
        [ExecuteAlways]
        public class GameManager : MonoBehaviour {

                [SerializeField] private GameParameters gameParameters;

                [Header("Stars")]
                [SerializeField] private GameObject sun;

                [SerializeField] private Planet venus;
                [SerializeField] private Planet earth;
                [SerializeField] private Planet mars;

                [SerializeField] private Planet moon;

                [Header("References")]
                [SerializeField] private CameraRig cameraRig;
                [SerializeField] private CurrentTimeHelper timeHelper;
                [SerializeField] private GameObject quizCameraVisor;

                private Planet[] planets;
                private Planet[] bodies;

                private void Awake() {
                        planets = new[] { venus, earth, mars };
```

```
                bodies = new [] { venus, earth, mars, moon};
        }

        public void Initialize() {
                InitializeAllStars(0.25f);


timeHelper.Initialize(gameParameters.DayDurationInSeconds);
                cameraRig.Initialize(new CameraRigInitializationData
{
                        sun = sun,
                        earth = earth
                });
        }

        public void InitializeSimulation() {
                Enable();

SetSpeedAllStars(gameParameters.YearDurationInSeconds,
gameParameters.DayDurationInSeconds);
                StartMovementAllStars();
        }

        public void Disable() {
                foreach (var body in bodies) {
                        body.gameObject.SetActive(false);
                }

                sun.SetActive(false);
                timeHelper.gameObject.SetActive(false);
        }

        public void Enable() {
                foreach (var body in bodies) {
                        body.gameObject.SetActive(true);
                }

                sun.SetActive(true);
                timeHelper.gameObject.SetActive(true);
        }

        public void StartQuiz() {
                earth.ShowRandomPoint();
```

```
                    quizCameraVisor.SetActive(true);

                    var newPosition =
cameraRig.CenterEyeTransform.position;
                    newPosition += cameraRig.CenterEyeTransform.forward
* 0.5f;

                    quizCameraVisor.transform.position = newPosition;

quizCameraVisor.transform.LookAt(cameraRig.CenterEyeTransform);
             }

             #region EventHandlers

             public void SetSunCamera(bool value) {
                    if (!value) {
                           return;
                    }

                    cameraRig.SetCamera(CameraPivot.Sun);
             }

             public void SetEarthCamera(bool value) {
                    if (!value) {
                           return;
                    }

                    cameraRig.SetCamera(CameraPivot.Earth);
             }

             public void SetSpeed(Single value) {
                    var dayDurationInSeconds = value /
gameParameters.DaysInYear;
                    SetSpeedAllStars(value, dayDurationInSeconds);
                    timeHelper.SetSpeed(dayDurationInSeconds);
             }

             public void SetGizmos(bool value) {
                    foreach (var body in bodies) {
                           body.GizmosEnabled = value;
                    }
             }
```

```
        public void SetWinterSeason() {
                SetSeason(Planet.Season.Winter);
        }

        public void SetSpringSeason() {
                SetSeason(Planet.Season.Spring);
        }

        public void SetSummerSeason() {
                SetSeason(Planet.Season.Summer);
        }

        public void SetAutumnSeason() {
                SetSeason(Planet.Season.Autumn);
        }

        public void Pause() {
                foreach (var body in bodies) {
                        body.PauseMovement();
                }

                timeHelper.Pause();
        }

        #endregion

        private void SetSeason(Planet.Season season) {
                earth.PauseMovement();
                earth.MoveToSeason(season);
                timeHelper.SetSeason(season);
                earth.ResumeMovement();
        }

        private void InitializeAllStars(float scale) {
                foreach (var planet in planets) {
                        planet.Initialize(new
Planet.InitializationParameters {
                                sun = sun,
                                planetScale = scale
                        });
                }
```

```
                    moon.Initialize(new Planet.InitializationParameters
{
                          sun = earth.gameObject,
                          planetScale = scale
                    });
             }

             private void SetSpeedAllStars(float yearDurationInSeconds,
float dayDurationInSeconds) {
                    foreach (var body in bodies) {
                          body.SetParameters(yearDurationInSeconds,
dayDurationInSeconds);
                    }
             }

             private void StartMovementAllStars() {
                    foreach (var body in bodies) {
                          body.StartMovement();
                    }
             }

      }
}
```