Escuela Técnica Superior de Ingenieros de Caminos, Canales y Puertos.
UNIVERSIDAD DE CANTABRIA

SIMULACIÓN VIRTUAL DE UNA OBRA

TRABAJO FINAL DE MASTER

Trabajo realizado por:
Juan Luis López Montoya

Dirigido:
Miguel Cuartas Hernández
Valentín Arroyo Fernández

Titulación:
Máster Universitario en Ingeniería de Caminos, Canales y Puertos

Santander, Junio de 2017

ÍNDICE

1. RESUMEN

1.1. RESUMEN.

La Realidad Virtual es la representación de escenas o imágenes de objetos producida por un sistema informático, que da la sensación de su existencia real. Aunque no es un concepto nuevo, ha tenido un gran desarrollo en los últimos años gracias al mundo de los videojuegos. Sin embargo, tiene aplicación en muchos ámbitos más, como la educación, la ingeniería, el marketing, o incluso la medicina.

Recientemente, también se ha comenzado a promover el uso de la gestión de proyectos mediante BIM (Building Information Modeling) en el ámbito de la ingeniería civil. Esta metodología de trabajo consiste en la gestión de proyectos de edificación u obra civil a través de una maqueta digital, que permite gestionar los elementos que forman parte de la infraestructura durante todo el ciclo de vida de la misma.

Teniendo en cuenta que la Realidad Virtual permite simular entornos tridimensionales y que la metodología BIM permite obtener modelos tridimensionales, parece razonable combinar ambas tecnologías para conseguir visualizar y manipular estos modelos, proporcionando una total sensación de inmersión al usuario. La visión de los modelos BIM en Realidad Virtual, potencia todas las posibilidades que tiene un modelo BIM y optimiza su percepción al incluir importantes aspectos (profundidad, sensación espacial, etc.) imposibles de alcanzar trabajando con dicho modelo en una pantalla de ordenador.

Siguiendo esta línea, se ha desarrollado el presente Trabajo Fin de Master, con el fin de mostrar alguna de las aplicaciones de la Realidad Virtual en la ingeniería civil mediante la virtualización de un entorno con varias infraestructuras. Para la creación de este entorno se ha utilizado el motor de videojuegos multiplataforma Unity, ya que es compatible con archivos exportados de varios programas de modelado 3D y permite crear aplicaciones en múltiples plataformas, incluidas aquellas que soportan dispositivos de Realidad Virtual.

La virtualización del entorno se ha realizado a modo de ejemplo práctico para facilitar a cualquier profesional del sector, inexperto en el uso de esta tecnología, un acercamiento a la creación de aplicaciones destinadas al uso de dispositivos de realidad virtual. En el ejemplo se explica cómo modelizar un terreno real, importar modelos 3D de infraestructuras o cómo interactuar con el entorno y los objetos que lo componen mediante la programación de códigos y el uso de herramientas que proporciona Unity, para terminar creando una aplicación de PC compatible con dispositivos de realidad virtual, en la que podamos desplazarnos en primera persona por un entorno virtual en el que interactuar con los objetos y modelos que lo componen.

1.2. ABSTRACT.

Virtual Reality is the representation of scenes or images of objects produced by a computer system, which gives the impression of their real existence. Although it´s not a new concept, it has had a great development in recent years thanks to the world of videogames. However, it has application in many more areas, such as education, engineering, marketing, or even medicine.

Recently, the use of project management through BIM (Building Information Modeling) has also begun to promote in the field of civil engineering. This work methodology consists of the management of building projects or civil works through a digital model, that allows to manage the elements that form part of the infrastructure throughout its life cycle.

Considering that Virtual Reality allows to simulate three-dimensional environments and that the BIM methodology allows to obtain three-dimensional models, it seems reasonable to combine both technologies to be able to visualize and manipulate these models, providing a total immersion sensation to the user. The vision of the BIM models in Virtual Reality, upgrade all the possibilities that a BIM model has and optimizes its perception by including important aspects (depth, space sensation, etc.) impossible to reach by working with this model on a computer screen.

In this line, the present End of Master's Degree Project has been developed, in order to show some of the applications of Virtual Reality in civil engineering through the virtualization of an environment with several infrastructures. The multi-platform video game engine Unity has been used for the creation of this environment, as it is compatible with exported files of several 3D modeling programs and allows to create applications in multiple platforms, even those that support Virtual Reality devices.

The virtualization of an environment has been done as a practical example to supply an approach to the applications creations for virtual reality devices to any professional in the sector, inexperienced in the use of this technology. The example explains how to model a real terrain, import 3D models of infrastructures, or how to interact with the environment and the objects that compose it by programming codes and using tools provided by Unity,finally, it has been created a PC application compatible with virtual reality devices, in which we can move in first person through a virtual environment in which to interact with the objects and models that compose it.

2. INTRODUCCIÓN

2.1. CONTEXTO.

Recientemente, se ha comenzado a implementar la gestión de proyectos mediante BIM (Building Information Modeling) en el ámbito de la ingeniería civil. Esta metodología de trabajo consiste en la gestión de proyectos de edificación u obra civil a través de una maqueta digital, que conforma una gran base de datos con la que gestionar los elementos que forman parte de la infraestructura durante todo el ciclo de vida de la misma. Dicha metodología está suponiendo una revolución tecnológica para la cadena de producción y gestión de la edificación y las infraestructuras, ya que permite construir de una manera más eficiente, reduciendo costes al tiempo que permite a proyectistas, constructores y demás agentes implicados trabajar de forma colaborativa.

A su vez, se ha producido un gran desarrollo en el campo de la Realidad Virtual con la aparición de nuevos dispositivos y plataformas relacionados con esta tecnología. Por tanto, teniendo en cuenta que los modelos generados mediante BIM destacan por ser tridimensionales, parece lógico combinar ambas herramientas para visualizar e interactuar con estos modelos mediante el uso de dispositivos de realidad virtual.

La integración de modelos BIM con Realidad Virtual genera una total sensación de presencia en el modelo. Esto facilita la comprensión espacial del proyecto, al dar la sensación de vivir aspectos como la percepción de la amplitud, la proporción entre componentes o la funcionalidad de la propuesta. Por eso, no hablamos sólo de un valor estético sino también funcional, ya que tenemos la posibilidad de crear un ambiente en tres dimensiones en el que sumergirnos de una forma virtual y con el que poder trabajar a escala 1:1.

2.2. LA REALIDAD VIRTUAL.

La Realidad Virtual consiste en la simulación generada por ordenador de un ambiente tridimensional, cuyo objetivo principal es lograr una gran inmersión en dicho entono, es decir, conseguir que el usuario tenga la sensación de estar presente en él.

Actualmente, esta tecnología implica utilizar una pantalla montada en la cabeza, como unas gafas, para tener una visión estereoscópica mientras nos movemos gracias a controles manuales o sensores de movimiento.

Aunque la realidad virtual no es nueva, ha estado presente en laboratorios de investigación, instalaciones industriales o en el ámbito militar, hasta el año 2012 no

comenzó su auge con la aparición de un kit de desarrollo llamado Oculus Rift Development Kit 1. A partir de entonces, se han ido desarrollando nuevos y mejores dispositivos gracias a dos grandes avances producidos en el mundo de los teléfonos móviles: el desarrollo de pantallas ligeras, baratas y de alta resolución, así como el de una nueva generación de sensores de movimiento de gran precisión.

El impulso de la Realidad Virtual se está produciendo en gran parte gracias al éxito que ha tenido el mundo de los videojuegos. Sin embargo, tiene numerosas aplicaciones, pueden valerse de la visualización de modelos 3D mediante dispositivos de realidad virtual disciplinas relacionadas con el mundo de la educación, la ingeniería, el marketing, o incluso la medicina.

2.3. DISPOSITIVOS DE REALIDAD VIRTUAL.

Los cascos o gafas de realidad virtual, también llamados HMD (Head-Mounted Display), son dispositivos que permiten visualizar imágenes sobre una pantalla situada cerca de los ojos, englobando incluso todo el campo de visión del usuario. Además, gracias a que están sujetos a la cabeza del usuario pueden seguir sus movimientos, consiguiendo una mejor inmersión.

Podemos distinguir dos tipos: los que llevan pantalla incorporada y los que son una carcasa destinada a que el usuario introduzca un smartphone. También pueden incluir en ocasiones guantes con sensores, mandos con control de movimiento o cámaras de posicionamiento que permiten hacer cosas como andar dentro del escenario o tocar los objetos virtuales.

Para este proyecto, utilizaremos unas gafas del primer tipo, en concreto se ha utilizado el set de Oculus Rift, cuyas gafas cuentan con dos lentes y unos auriculares, un mando inalámbrico, un mando de Xbox inalámbrico y un rastreador de posición. Pero para disfruta de la Realidad Virtual necesitaremos también un dispositivo que genere el entorno virtual, como un ordenador, ya que las gafas de Realidad Virtual no incorporan procesador.

Figura 1. Set de Oculus Rift utilizado.

2.3.1. Funcionamiento del dispositivo.

El funcionamiento es sencillo, las gafas disponen de una pantalla y unas lentes. Por una parte, la pantalla genera dos imágenes ligeramente diferentes, una para cada ojo, para conseguir una visión en tres dimensiones. Esto es posible gracias a la visión estereoscópica, que permite solapar dos imágenes ligeramente diferentes, una para cada ojo, como una sola imagen. Por otra parte, las lentes amplían el ángulo de visión, generando la sensación de que la pantalla abarca por completo el campo de visión del usuario.

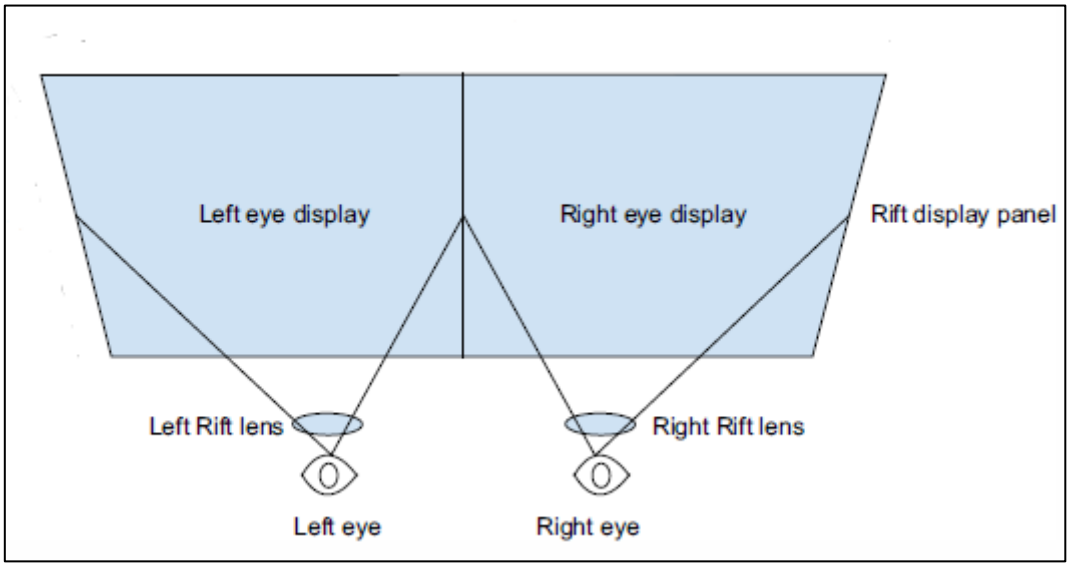

Figura 2. Funcionamiento de la pantalla y las lentes.[1]

Figura 3. Ejemplo de imágenes diferentes para visión estereoscópica.[2]

[1] Bradley Austin Davis, Karen Bryla, Phillips Alexander Benton. 2015.

[2] Ibíd.

2.4. INGENIERÍA CIVIL Y REALIDAD VIRTUAL.

Los modelos relacionados con la construcción necesitan poder generar cambios en la geometría del proyecto. La integración de dicha geometría junto con el plan del proceso constructivo son la base de los modelos en 4D (3D + tiempo) que junto a la Realidad Virtual permiten una representación muy realista de la obra y su construcción.

Actualmente, las empresas y profesionales españoles que quieran participar en proyectos de construcción, reforma, instalación, y explotación en lugares como Reino Unido, EE.UU., Emiratos Árabes, China o Australia tienen que implantar el modelo BIM de manera obligatoria para acceder a licitaciones, contratos y colaboraciones. Además, el Ministerio de Fomento ha marcado para el año 2018 como fecha límite para implantar, en las fases de diseño y construcción, el uso de un modelo BIM para las licitaciones de equipamientos e infraestructuras públicas de presupuesto superior a 2 millones de euros.

Por tanto, en un futuro próximo toda obra de envergadura requerirá de un modelo BIM. Lo que abre las puertas a un mercado creciente en todo lo relacionado con los modelos en tres dimensiones. Asimismo, como ya hemos indicado anteriormente, considerando que los modelos generados mediante BIM destacan por ser tridimensionales, parece adecuado combinar la Realidad Virtual y los modelos BIM.

Entre las principales ventajas del uso de la Realidad Virtual y el entorno BIM podemos destacar:

- Sumergirse en el modelo BIM, casi como si recorriesemos la infraestructura real.
- Facilitar la detección de fallos en el diseño y permitir cambiarlos en tiempo real.
- Simular procesos: evacuación, tránsito de persona, etc.
- Aplicar al mantenimiento y conservación de la infraestructura.
- Estudiar el comportamiento estructural de la infraestructura sometida a distintos esfuerzos.
- Facilitar la colaboración entre arquitectos, diseñadores, ingenieros, constructores, etc.
- Hacer más intuitivo y fácil de entender un modelo de información BIM.
- Aumentar la productividad y la calidad de los diseños
- Reducir costes en comercialización y marketing, el cliente podrá “pasear” por el modelo antes de ser construido.

3. UNITY 3D

3.1. INTRODUCCIÓN.

La creación de un entorno virtual con modelos tridimensionales para el uso de la Realidad Virtual era un trabajo complejo, que requería tener amplios conocimientos de programación. No obstante, gracias a programas como Unity se ha simplificado esta tarea, que junto a la aparición de nuevos y mejores dispositivos de realidad virtual han provocado el gran auge que está teniendo actualmente.

En concreto, Unity se trata de un motor de videojuegos multiplataforma, que permite su diseño, creación y representación. Sin embargo, nosotros aprovecharemos esta herramienta para crear una simulación virtual de una obra e interactuar con diferentes objetos. Por ejemplo, podríamos utilizar la simulación para mostrar diferentes fases de la construcción del proyecto mientras estamos “presentes” en la obra o enseñar una nueva infraestructura tras finalizar su construcción.

En el caso del proyecto llevado a cabo, nos centraremos en mostrar infraestructuras ya finalizadas. Puesto que, para representar las diferentes fases de construcción de una infraestructura necesitaríamos los modelos en sus diferentes fases, los cuales tendríamos que crear con algún programa de modelado 3D.

Unity dispone de varios tipos de licencia, desde una versión de pago, paga profesionales que busquen obtener ganancias y requieran de una personalización avanzada, hasta una versión gratuita para principiantes o aficionados. Esta versión gratuita dispone de suficientes prestaciones para desarrollar nuestro proyecto.

Entre sus características principales podemos destacar:

- Compatible con varios programas de modelado 3D como Blender, AutoCAD o SketchUp y modelado BIM como Revit.
- Tiene soporte para mapeado de relieve, texturas 3D y sombras dinámicas.
- Dispone del editor para códigos MonoDevelop, aunque ofrece integración nativa con el editor Visual Studio. Admite como lenguaje de programación C# y JavaScript.
- Desplegable en casi todas las plataformas móviles, de Realidad Virtual, de escritorio, consola, televisión o Web.
- Dispone de una tienda con una amplia gama de recursos gratuitos y de pago, como modelos 3D, materiales, scripts, animaciones o sonidos.

3.2. COMENZANDO A UTILIZAR UNITY.

A continuación, incluimos una pequeña introducción a Unity para explicar sus componentes básicos.

3.2.1. Ventanas principales.

Una vez abierto el programa y creado un proyecto nuevo, nos aparecerán las siguientes ventanas:

1- Scene: permite visualizar el proyecto. Permite entre otras acciones mover, escalar o rotar objetos.
2- Game: muestra la escena desde el punto de vista de la cámara mientras construimos la escena o la ejecutamos, para ejecutarla tendremos que pulsar el botón de reproducir situado en la parte superior.
3- Hierarchy: muestra los objetos que aparecen en la escena. Si hacemos doble click sobre el nombre del objeto, se enfocará la vista sobre él en Scene, apareciendo marcado y mostrará sus componentes en Inspector.
4- Project: muestra los recursos de los que disponemos en nuestro proyecto
5- Console: muestra mensajes del editor, da advertencias e informa de los errores que ocurren durante la creación del proyecto.
6- Inspector: muestra los componentes del objeto seleccionado y permite cambiar sus propiedades

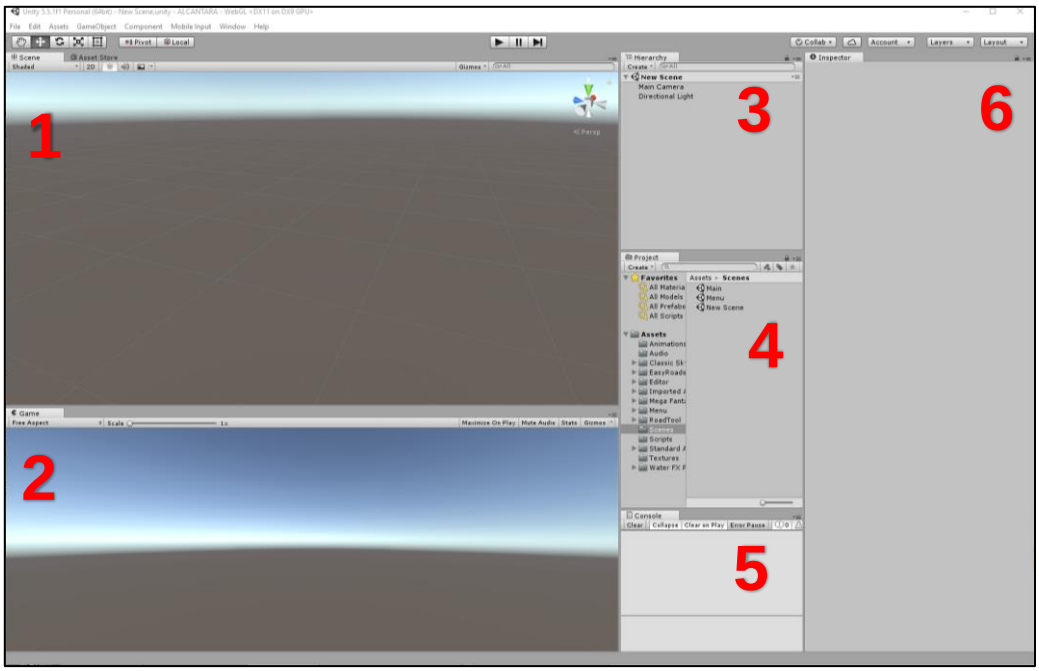

Figura 4. Interfaz de Unity.

Estas son las principales ventanas, aunque la interfaz del editor de Unity es personalizable, podemos distribuir las ventanas a gusto del usuario y añadir las que se necesitemos desde la pestaña Window.

3.2.2. Navegación.

Unity dispone de los siguientes botones de navegación, herramientas básicas para moverse por la escena y editar los objetos:

- Mano: sirve para desplazarse por la escena.
- Mover: sirve para mover un objeto sobre alguno de los ejes.
- Rotar: sirve para rotar un objeto sobre alguno de los ejes.
- Escalar: sirve para escalar un objeto sobre alguno de los ejes.

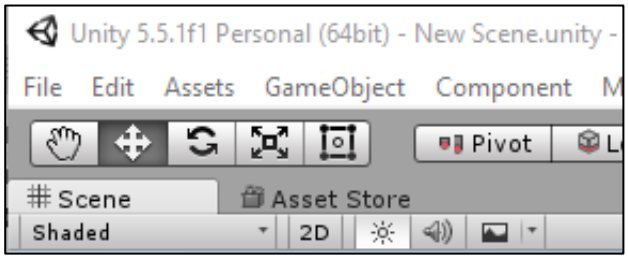

Figura 5. Botones de navegación.

3.2.3. Recursos.

Para la creación de un proyecto disponemos de diferentes recursos o asset, entre los que podemos destacar:

- Escenas: es el recurso principal que contiene los objetos que componen el proyecto.
- Materiales: definen el aspecto que se le puede dar a una malla (textura, color, sombreado, etc.).
- Scripts: permiten ejecutar diferentes eventos, como crear efectos gráficos o controlar el comportamiento de objetos.
- Clips de audio.
- Clips de animación.

En la carpeta Assets se recogen todos los recursos del proyecto, esta se crea automáticamente cada vez que creamos un nuevo proyecto. Podemos acceder a estos recursos desde la carpeta Assets que aparece dentro de la ventana Project.

Es muy importante mantener ordenados los diferentes recursos por carpetas para poder disponer fácilmente de ellos, ya que, según avancemos en un proyecto se añadirán nuevos recursos. Para crear una carpeta pulsaremos el botón derecho en la ventana Assets y seleccionaremos Create - Folder.

3.2.4. Paquetes de recursos.

Unity dispone una serie de recursos predefinidos que podemos importar desde la pestaña Assets seleccionado Import Package. Estos incluyen diferentes elementos como vegetación, agua, cielos, personajes o incluso vehículos.

Además, en la tienda online de Unity podemos encontrar otros recursos creados por desarrolladores, para descargar e importar a la escena. Para acceder a esta tienda seleccionaremos Asset Store desde la pestaña Window,

Valiéndonos de estos recursos podemos generar fácilmente un entorno virtual, ya que la creación de objetos como árboles o personajes, que no son objeto principal de la simulación virtual de una obra, requerirían de más trabajo.

3.2.5. Objetos.

Entre los elementos más importantes de Unity destacan los objetos predefinidos que podemos encontrar en la pestaña GameObject. Entre estos podemos resaltar los siguientes:

- 3D Object: permiten crear nodos con formas geométricas básicas, como planos, cubos, esferas, cilindros, etc. Disponen de componentes básicos como posición, orientación, material, malla geométrica y geometría de colisión.
- Empty GameObject: sólo incluye las componentes de posición y orientación. Son muy útiles para agrupar objetos.
- Camera: incluye un componente cámara que sirve para ver la escena.
- Light: incluye un componente Light que se comportará como una fuente de luz en nuestra escena.

- UI (User Interface): permiten crear interfaces de usuario, que permiten al usuario de la aplicación comunicarse, obtener información o modificar el estado de los objetos. Un ejemplo podría ser un texto, una imagen o un botón.

3.2.6. Importar Modelos 3D.

Asimismo, Unity permite importar modelos de muchos programas de modelado lo que facilita considerablemente la tarea de realizar una simulación virtualizar de una obra civil.

De hecho, podemos importar directamente a Unity archivos .fbx, .dae, .3DS, .dxf y .obj.

Ventajas:

- Solo se exportan los datos que uno necesita.
- Generalmente son archivos más pequeños.
- Fomenta un enfoque modular (diferentes componentes para tipos de colisión o interactividad)
- Soporta otros paquetes 3D para cuyo formato Unity no tiene soporte directo.

Desventajas:

- El flujo de trabajo puede ser más lento para la creación de prototipos e iteraciones.
- Es más fácil perder la pista de las versiones entre la fuente (archivo de trabajo) y los datos del juego.

Aunque también admite mediante conversión archivos, como: Max, Maya, Blender, Cinema4D, Modo, Lightwave &Cheetah3D, etc.

Ventajas:

- Proceso rápido de iteración.
- Simple de usar al principio.

Desventajas:

- Los archivos pueden llenarse de datos innecesarios.
- Los archivos grandes pueden ralentizar las actualizaciones de Unity.
- Menos validación, por lo tanto, es más difícil solucionar problemas.

3.2.7. Reproducción de la escena.

En la parte superior de la pantalla del editor hay tres botones para ejecutar, pausar o avanzar en la escena. Esto permite observar cómo se desarrolla la aplicación que se está editando.

3.3. REALIDAD VIRTUAL EN UNITY 3D.

Antes de comenzar a utilizar la Realidad Virtual en Unity debemos conocer las formas de poder integrarla en nuestro proyecto. Generalmente, necesitaríamos disponer de una cámara que renderice una visión estereoscópica, una imagen para cada lente del dispositivo de realidad virtual, pero Unity permite utilizar una cámara estándar siempre que activemos la opción Virtual Reality Supported en la configuración de Player Settings. Más adelante explicaremos como hacerlo utilizando un ejemplo práctico.

En concreto, al activar esta opción se llevan a cabo los siguientes procesos:

- Las matrices de vista y proyección se ajustan para tener en cuenta el campo de visión, de manera que, solo necesitamos una cámara para tener una visión estereoscópica.
- Las optimizaciones ocurren automáticamente para que sea menos costoso dibujar frames dos veces (uno para cada ojo).
- Se aplica automáticamente el seguimiento de la cabeza (Head Tracking) y se adapta el campo de visión.

También debemos saber si nuestro dispositivo de realidad virtual es soportado en Unity, de no ser así, el fabricante deberá de proporcionar un paquete de plugins compatibles con Unity. En nuestro caso, disponemos de unas Oculus Rift que son compatibles con Unity.

Cabe destacar que en 2015 se comenzó a desarrollar un proyecto de código abierto para plataformas de realidad virtual (Open Source Virtual Reality). Para dar acceso a todo el mundo a esta tecnología, permitiendo programar en una sola interfaz sin la necesidad de conocer el dispositivo final que utilizará el usuario. Esto nos permite crear nuestra aplicación en Unity para un sistema operativo específico (como Windows o Mac), pero deja al usuario configurar la aplicación para cualquier dispositivo que vaya a utilizar.

4. EJEMPLO DE VIRTUALIZACIÓN DE INFRAESTRUCTURAS.

4.1. INTRODUCCIÓN.

Con este ejemplo se pretende mostrar como importar modelos tridimensionales de un proyecto a una escena que represente virtualmente un lugar físico, y como introducir diferentes elementos para interactuar con el entorno y los modelos mediante la Realidad Virtual.

Principalmente, se trata de enseñar que gracias al programa Unity, este proceso es relativamente sencillo de realizar una vez se disponga de los modelos tridimensionales utilizados en el proyecto. De manera que, en este ejemplo nos centraremos en los pasos siguientes a la creación de los modelos, que hemos obtenido de bibliotecas gratuitas de la red.

4.2. INFRAESTRUCTURAS.

En este ejemplo se ha elegido el entorno del municipio de Alcántara, situado en la provincia de Cáceres. Para mostrar el puente romano de Alcántara y la presa del embalse de José María de Oriol-Alcántara II.

El puente romano fue construido entre el año 104 y 106 por Cayo Julio Lacer para salvar el río Tajo en la vía que comunicaba Norba(Cáceres) con Conimbriga (Condeixa-a-Velha). El puente está compuesto por seis arcos de altura desigual, que salvan una distancia de 214 m. Siendo su altura máxima de 48 m en los dos arcos centrales para poder salvar un profundo cauce con crecidas de caudal importantes.

La presa fue construida en 1969 por Hidroeléctrica Española, actual Iberdrola. Se trata de una presa de contrafuertes, con 570 m de longitud de coronación, 130 m de altura desde cimientos y 223 m de cota de coronación. Utilizada para generar energía eléctrica.

Además, se añade la posibilidad de activar o desactivar un nuevo puente de canto variable, ubicado entre el puente romano y la presa, para representar el puente que se va a construir para liberar del tráfico al puente romano y poder mostrar cuál sería su ubicación o como quedaría una vez construido.

En enero de 2017 finalizó el plazo para la presentación de ofertas por parte de las empresas consultoras para la asistencia técnica de la redacción del proyecto, por lo que el proyecto podría estar redactado para mayo o junio de 2018.

4.2.1. Obtención de los modelos.

Los modelos se han obtenido de la página web “3D Warehouse3”, biblioteca de código abierto donde cualquiera con SketchUp puede cargar y descargar modelos tridimensionales para compartir.

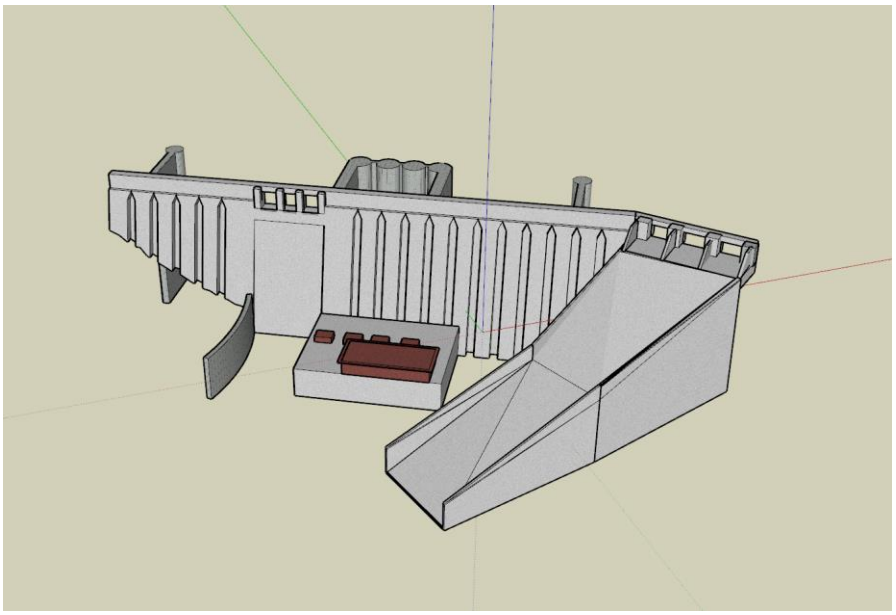

Figura 6. Presa de Alcántara en SketchUp

Figura 7. Puente Romano de Alcántara en SketchUp

[3] https://3dwarehouse.sketchup.com

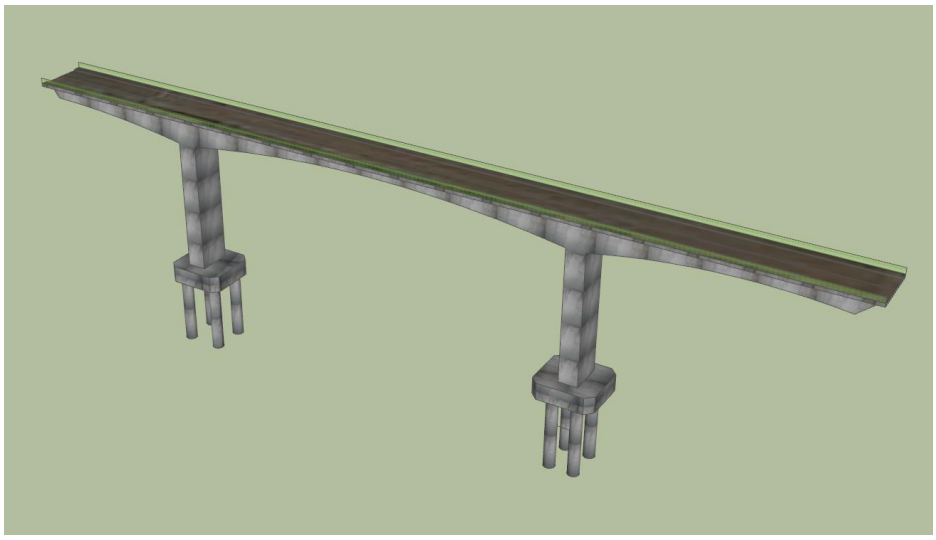

Figura 8. Nuevo Puente de Alcántara en SketchUp

Una vez descargados los modelos, tenemos varias opciones para importarlos a Unity:

1ºOpción: Abrir el modelo con Google Sketch Up y exportarlo en formato .dae (Default Settings). A continuación, abrir con Blender el archivo .dae generado y exportarlo a formato .fbx para poder importarlo a Unity.

2º Opción: Realizar el modelo con Revit o algún otro programa de BIM o si se dispone de Sketch Up Pro. De manera que podamos exportar directamente a un formato .fbx.

3º Opción: Insertar el modelo de Sketch Up directamente ya que Unity admite ficheros en formato .dae, pero se crearán varios objetos vacíos que dificultarán el trabajo

4.3. TERRENO.

Con Unity podemos generar un terreno a partir de un heightmap, fichero de tipo imagen en el que se representan las elevaciones del terreno, donde cada pixel representa la altura del terreno de ese punto mediante un color según una escala de grises. Un heightmap puede ser de 8-bits y representar altitudes de 0 a 256 o bien de 16-bits y representar alturas con los valores de 0 a 65536.

4.3.1. Obtención de datos.

Para obtener los datos que definan el terreno de la zona a representar utilizaremos la información de la base de datos MDT05/MDT05-LIDAR[4] del Instituto Geográfico Nacional (IGN). Esta base de datos es un modelo digital del terreno (MDT) que contiene la cota de todos los puntos en una cuadrícula de 5 metros de paso de malla. Las coordenadas usadas para referenciar dichos puntos son coordenadas ETRS89.

Dado que este MDT se distribuye en partes según la distribución oficial de hojas 1:25.000, cuando descarguemos el MDT es necesario localizar en que hoja se localizan las infraestructuras. Para localizar dicha hoja, se puede utilizar la Búsqueda en visor de la propia página de descargas del IGN que muestra un mapa de España con la distribución en hojas representada como rectángulos superpuestos sobre el mapa.

En nuestro caso, el municipio de Alcántara está ubicado en la hoja 0648. Dado que estas hojas proporcionan información de una gran superficie será necesario definir el área que queremos representar alrededor de la obra. Lo mejor para importar a Unity es escoger un área de tamaño cuadrado, en nuestro caso hemos elegido un cuadrado de 1200 m de lado.

Utilizando el programa Google Earth obtenemos las coordenadas de la esquina superior izquierda y de la esquina inferior derecha:

	COORDENADA ETRS89 (H29)	
ESQUINA	X (m)	Y (m)
SUPERIOR IZQUIERDA	680200	4400000
INFERIOR DERECHA	681400	4398800

Tabla 1 Puntos límite del terreno a modelizar

[4] Ministerio de Fomento – Centro Nacional de Información Geográfica

Figura 9. Vista aérea de la zona a modelizar

4.3.2. Generar el heightmap.

A la hora de generar el heightmap, utilizaremos el programa Heightmap Creator[5]. Este permite extraer los datos de la zona de interés (del fichero tipo .asc) que contiene el MDT05/MDT05-LIDAR y los codifica en un fichero de tipo heightmap que puede ser importado en Unity.

Dado que un archivo .raw no es fácilmente visualizable, se genera también otro archivo de tipo imagen convencional que contiene una información equivalente al raw. En la propia aplicación se realiza una vista preliminar. Cuanto más oscuro, menos altura y cuanto más claro, más altura.

El programa nos proporciona también la altura máxima del terreno creado dato que será importante en el momento de importar el heightmap en Unity 3D.

[5] Cuartas Hernández, Miguel (2014-2015)

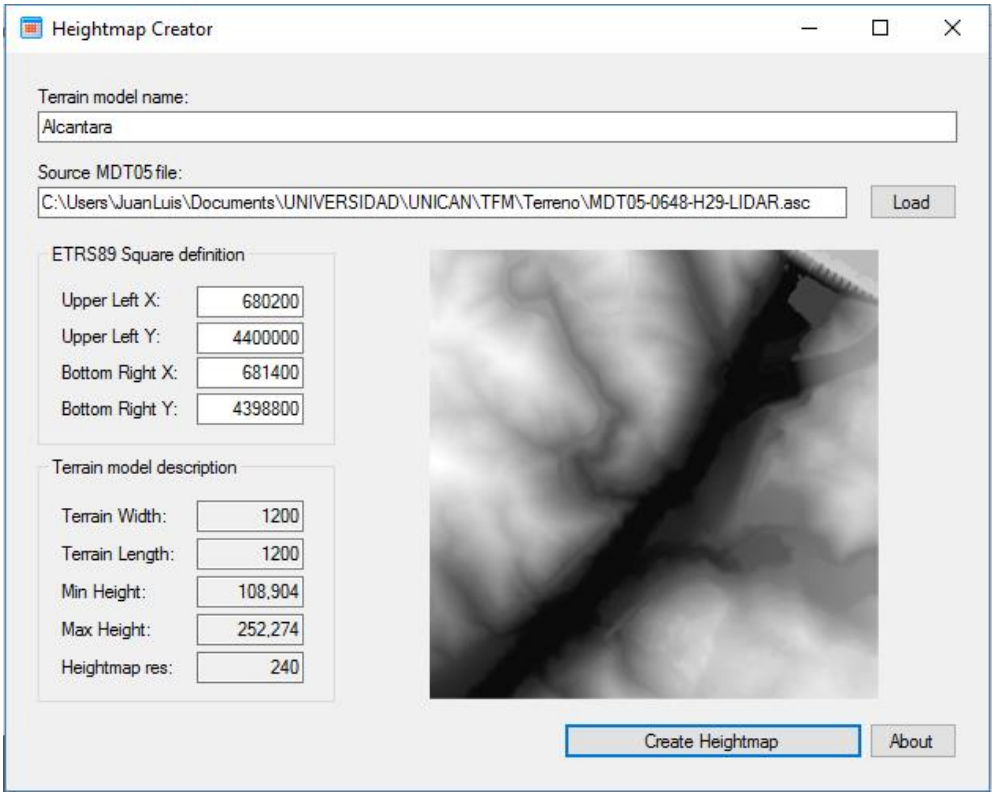

Figura 10. Captura de pantalla del programa para generar el heightmap

4.3.3. Importar el heightmap en Unity 3D.

Antes de importar el terreno, crearemos un nuevo proyecto. Tras abrir Unity, seleccionaremos la opción crear un nuevo proyecto y elegiremos una carpeta para su ubicación. Inicialmente guardaremos la escena que se crea automáticamente, en nuestro caso se ha nombrado como Main, y para ello crearemos una carpeta en Assets que se llame Escenas.

Tras crear el proyecto es necesario insertar un objeto tipo terreno a nuestra escena sobre el que importar el heightmap. Para ello, pulsando el botón derecho en la ventana Hierarchy seleccionamos Create - 3D Object - Terrain. También se puede acceder al menú a través de la pestaña GameObject. Una vez creado aparecerá en la escena y al seleccionarlo se mostrarán sus componentes en la ventana Inspector.

A continuación, es necesario acceder a la pestaña de Terrain Settings a través de un botón en forma de engranaje como se muestra en la siguiente figura. Las dimensiones deben ajustarse al tamaño que se le ha dado al heightmap, 1200x1200 y se debe de introducir la altura máxima del terreno, dato que obtenemos al crear el heightmap.

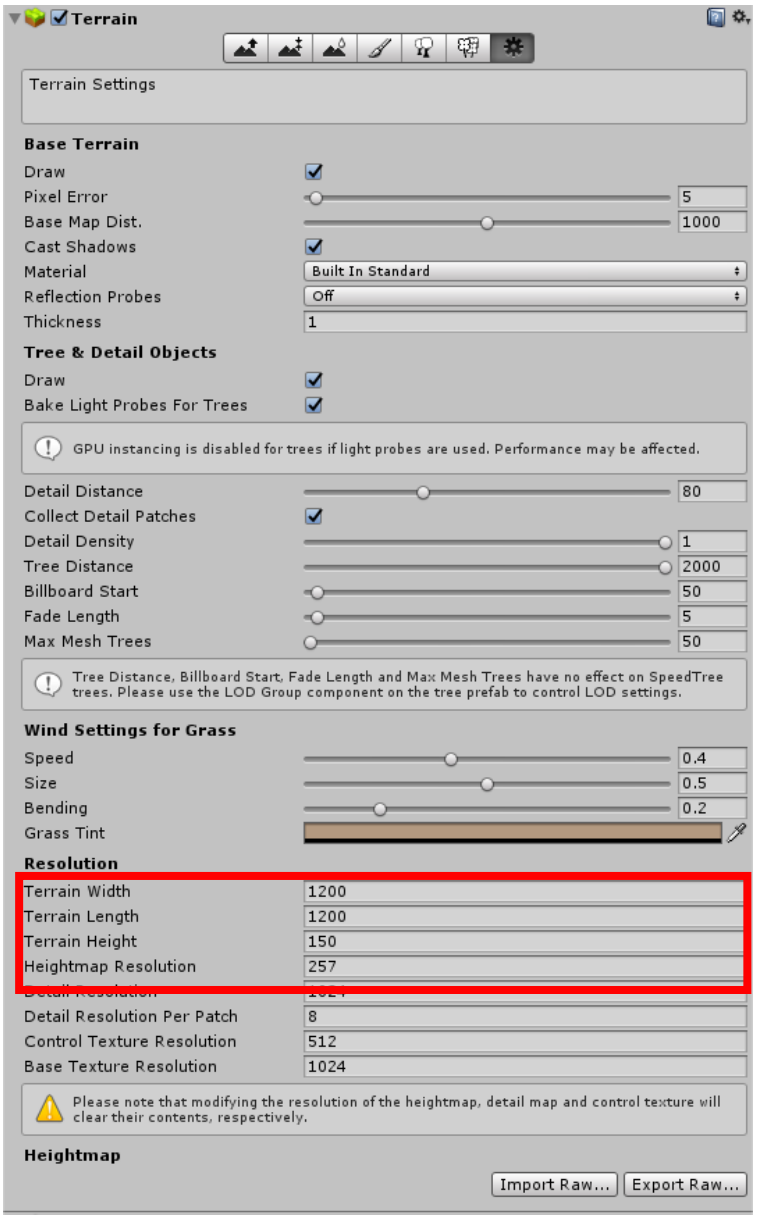

Figura 11. Inspector del objeto Terrain

Una vez introducidas los datos de las dimensiones en Resolution, podemos importar el heightmap a través del botón “Import Raw…” que aparece en la misma ventana. Tras pulsarlo será necesario acceder a la ruta donde se encuentra el heightmap y seleccionarlo. Aparecerá una ventana que muestra varios datos referentes al mismo y tendremos que comprobar que la profundidad está en Bit16, que las dimensiones (Width y Height) son las correcta y que el Byte Order es Windows.

4.4. DETALLES DEL TERRENO Y EL ENTORNO.

Desde la ventana Hierarchy, seleccionamos el terreno y se mostrará en el Inspector los componentes asociados y una serie de herramientas que nos permitirán editar el terreno.

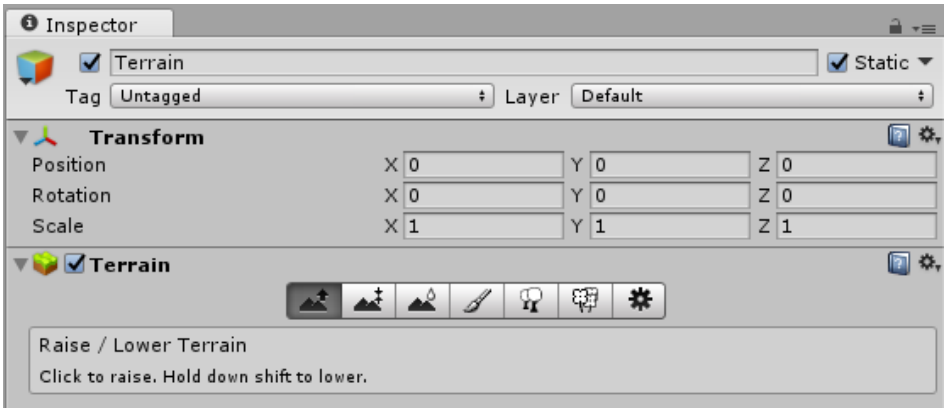

Figura 12.Herramientas del objeto Terrain.

Estas herramientas son, de izquierda a derecha:

- Raise/Lower Terrain: permite generar elevaciones o eliminarlas si se mantiene la tecla SHIFT pulsada.
- Paint Height: permite generar elevaciones con una altura determinada o establecer la altura por defecto del terreno.
- Smooth Height: permite suavizar la superficie del terreno.
- Paint Texture: permite definir y utilizar un conjunto de texturas en el terreno. Para utilizar una textura es necesario añadirla primero.
- Place Trees: permite incorporar árboles al terreno.
- Paint Details: permite definir un conjunto de detalles como césped o rocas sobre el terreno.
- Terrain Settings: permite definir los detalles principales del terreno. Es la pestaña que se ha utilizado para importar el Heightmap y definir la geometría del mismo.

Utilizando estas herramientas y el plano de referencia se pueden añadir texturas, vegetación y detalles al terreno. Para ello es necesario importar un paquete de recursos. De manera que, accederemos a la pestaña Assets - Import Package - Environment. Una vez importado, es necesario añadir las texturas y árboles en las pestañas de las herramientas correspondientes.

4.4.1. Plano de referencia.

Tras importar el terreno vamos a crear un plano con las dimensiones del terreno (1200x1200), al que añadiremos como textura la imagen de una fotografía aérea de la zona. Este plano lo utilizaremos como plano de referencia, el cual nos dará ciertas facilidades a la hora de ubicar los diferentes elementos que vayamos a añadir.

Para crear un plano pulsaremos el botón derecho en la ventana Hierarchy y seleccionaremos Create - 3D Object - Plane. Para añadir una textura al plano arrastramos la imagen dentro de la ventana Assets. A partir de ese momento la imagen formará parte de los recursos del proyecto y podrá ser utilizada como textura simplemente arrastrando la imagen dentro del elemento en el que queremos que se inserte. De esta manera tendremos un plano que podremos desplazar sobre el terreno para poder ubicar los diferentes elementos que vayamos a añadir.

Figura 13. Plano de referencia y terreno modelizado

4.4.2. Añadir texturas.

A la hora de añadir una textura es importante saber que la primera textura seleccionada será la que adquiera todo el terreno por defecto. En este caso se ha utilizado como base la textura “GrassRockAlbeldo”, de aspecto rocoso y con vegetación.

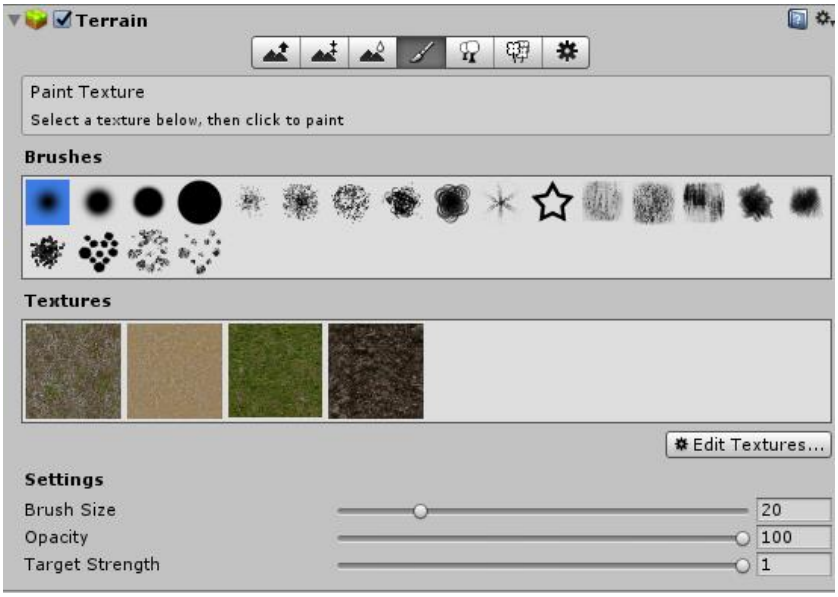

Figura 14. Herramienta Paint Textures con diferentes texturas añadidas.

Además, se han añadido otras texturas, como arena, césped o roca. Para añadir nuevas texturas, simplemente es necesario seleccionar el botón Edit textures - Add texture y elegirla. Si queremos incorporar nuevas texturas en el terreno, simplemente seleccionaremos la textura deseada y las características de la brocha como su tamaño u opacidad y pintaremos sobre el terreno.

4.4.3. Añadir árboles y vegetación.

El proceso para añadir árboles o vegetación es similar al utilizado en el caso de las texturas. Primero es necesario añadir los elementos seleccionando el botón Edit Trees, en el caso de los árboles, para después poder colocarlos sobre el terreno. De esta forma añadimos algunos modelos de árbol como Broadleaf_Desktop o Conifer_Desktop. Para incorporarlos al terreno los seleccionamos y elegimos las características de la brocha.

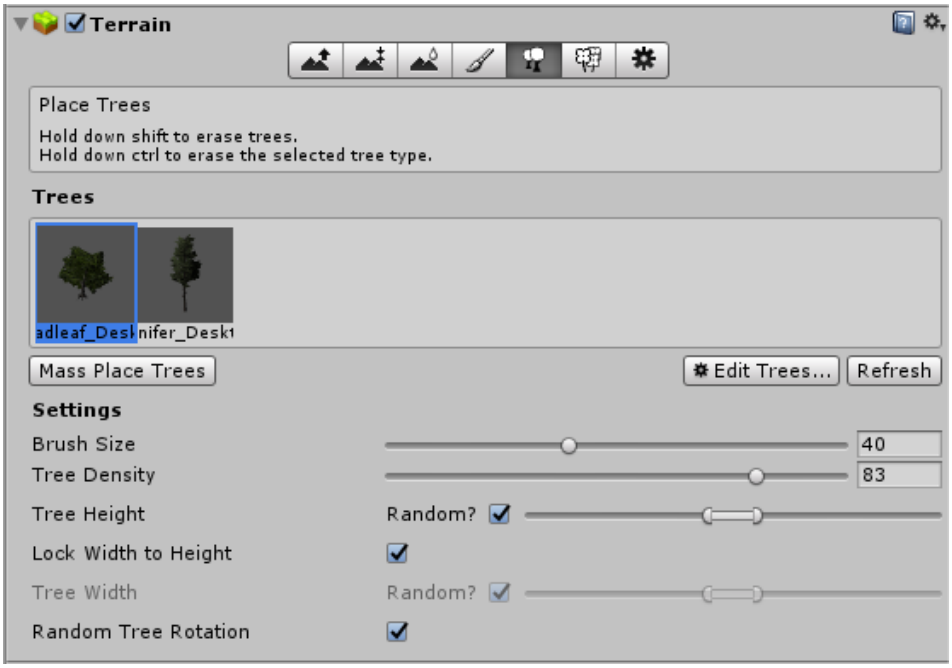

Figura 15. Herramienta Place Trees con diferentes tipos de árboles

Para añadir la vegetación, sería necesario seleccionar los detalles que se quieran incluir al terreno pulsando en el botón “Edit Details…” y luego “Add Grass Texture”. En este caso, no se ha utilizado ningún tipo de vegetación, ya que a la hora de utilizar las Oculus Rift, este elemento no se veía correctamente, al ser un elemento plano.

4.4.4. Añadir el cielo.

En un nuevo proyecto, Unity ofrece por defecto un material predefinido para el cielo (Skybox), pero para conseguir un aspecto más realista se puede utilizar diferentes recursos, como los que podemos descargar del Asset Store, para añadir un cielo de mejor calidad.

Para abrir el Asset Store seleccionamos la pestaña Window - Asset Store. En esta ventana se muestra un cuadro de búsqueda que nos permite introducir la descripción del recurso que buscamos, en este caso “skybox” y la opción de buscar recursos gratuitos. En este caso se ha utilizado el asset “Classic Skybox”. Una vez seleccionado lo importaremos a nuestra carpeta Assets.

Como hemos visto, el proceso para obtener recursos gratuitos es muy sencillo. Por lo que podemos utilizar muchos de los recursos que pone a nuestra disposición el Asset Store para mejorar la escena.

Para cambiar el Skybox seleccionaremos la pestaña Window - Lightning - Scene y añadiremos el skybox descargado. El paquete de materiales importado nos proporciona una amplia lista de cielos. Lo recomendable es utilizar el que muestre con mayor claridad e iluminación la escena.

Figura 16. Vista del skybox importado.

4.4.5. Añadir el río.

A la hora de añadir una masa de agua, en este caso queremos representar el río Tajo, necesitaremos importar ciertos recursos. Estos ya fueron añadidos junto con el paquete de Environment, utilizados para las texturas y árboles. Por ello, se encuentra en la carpeta Assets - Standard Assets - Environment - Water - Prefabs - WaterBasicDaytime. Para colocarlo en la escena, simplemente es necesario arrastrarlo sobre esta y ajustar su tamaño y posición para que coincida con la situación del río.

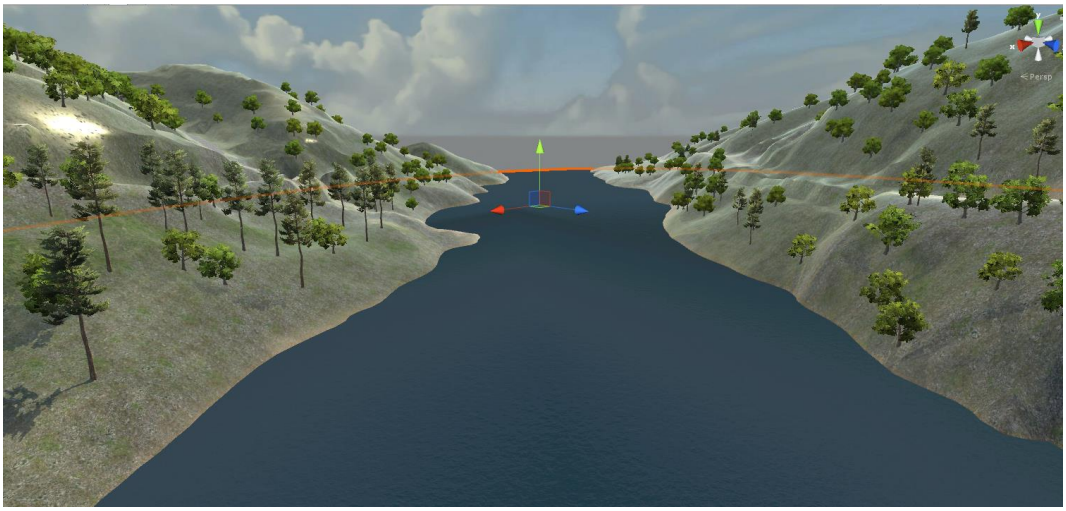

Figura 17. Vista del terreno con el río.

4.4.6. Añadir los modelos de las infraestructuras.

Para guardar cierto orden, antes de importar los modelos a Unity, crearemos una carpeta para cada modelo dentro de la carpeta Assets de nuestro proyecto. En estas carpetas, incluiremos una nueva llamada “Texturas” para importar en ella las texturas de cada modelo. Tras haber organizado todo arrastramos o importamos los modelos .fbx y sus texturas dentro de las carpetas correspondientes.

Una vez disponemos de los modelos en la carpeta Asset, podremos añadirlos a la escena arrastrando el modelo .fbx directamente sobre el terreno y con ayuda del plano de referencia ajustar su posición y escala.

Al añadir cada modelo tendremos que verificar que se ven correctamente las texturas. De no ser así, habrá que añadir a cada material la suya correspondiente. Para ello iremos seleccionando cada material y arrastrando sobre él su textura correspondiente.

Figura 18, Puente romano y nuevo puente de Alcántara.

Figura 19. Presa de Alcántara.

4.4.7. Añadir carreteras y caminos.

Para completar la escena, hemos añadido algunos de los caminos y carreteras que hay por la zona utilizando algunos de los paquetes gratuitos del Asset Store, que importaremos igual que en casos anteriores.

Dado que las alineaciones no están bien marcadas y presentan una superficie ligeramente irregular no es posible utilizar directamente objetos prefabricados ya que no

se ajustan bien al terreno. Por ello, primero hemos utilizado el paquete Easy Road que nos permite modificar la superficie del terreno gracias a un script (código) que incorpora.

Por ejemplo, para el caso de una carretera obtendremos una superficie plana y además nos regulará los contornos incorporando los desmontes o terraplenes necesarios, aunque tengamos que ajustar manualmente la posición de algunos puntos y suavizar los bordes que puedan aparecer al generar la nueva superficie del terreno afectado.

Para usar este paquete tendremos que utilizar las siguientes herramientas:

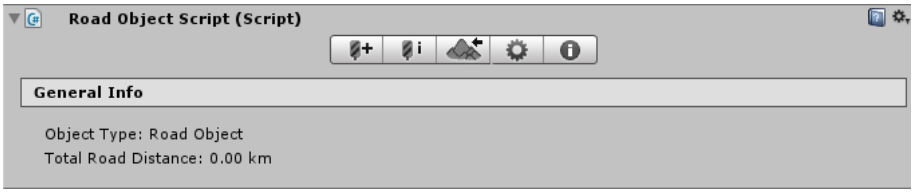

Figura 20. Cuadro de herramientas de Easy Road.

- El cuarto botón abre un menú que nos permite elegir el ancho de la carretera y la zona afectada por el desmonte o terraplén. Estos parámetros se deben elegir antes de colocar los puntos.
- El primer botón nos permite ir colocando los diferentes puntos que formarán la alineación.
- El tercer botón ejecutará el script que modifica el terreno.

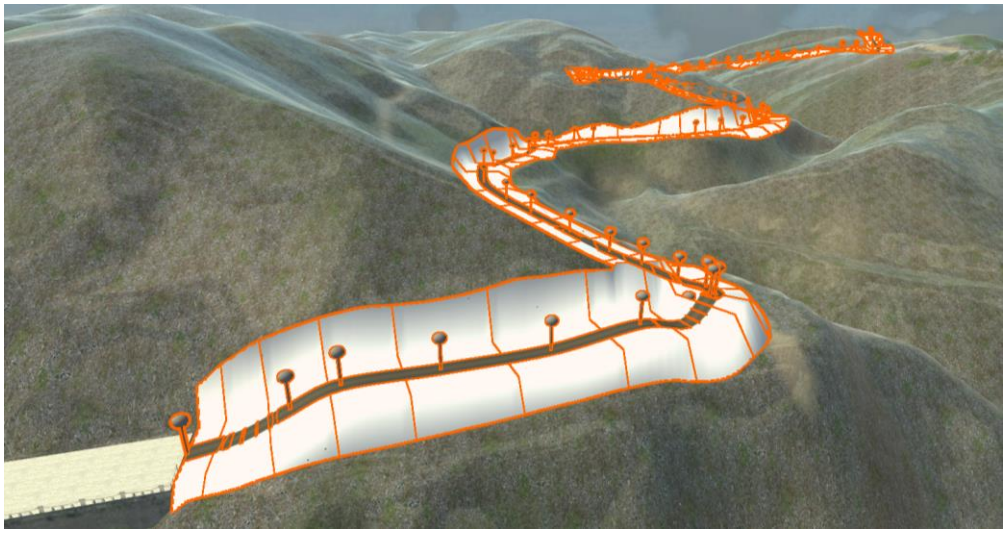

Figura 21.Puntos que forman un tramo de carretera creada con Easy Road y la superficie afectada.

Figura 22. Estado final de la carretera creada con Easy Road.

Una vez remodelado la zona del terreno en el que se sitúa la carretera, obteniendo una superficie relativamente plana, la creamos con el paquete Road Tool, introduciendo los puntos que la forman y eliminando la carretera creada anteriormente, ya que al utilizar la versión gratuita del paquete Easy Road solo nos permite colocar una carretera en la escena.

El paquete Road Tool nos permite elegir la anchura de la carretera, la altura a la que colocarla respecto al terreno y la textura que utilizar (material).

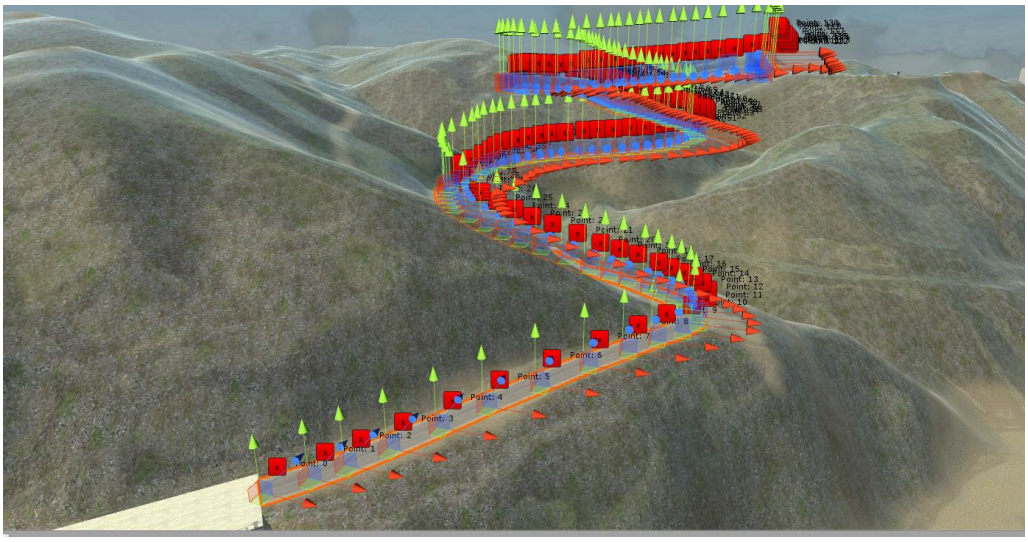

Figura 23. Puntos que forman un tramo de carretera creada con Road Tool.

Figura 24. Estado final de la carretera creada con Road Tool.

En el caso de los caminos simplemente hemos eliminado el camino creado con el paquete Easy Road y “pintado” la superficie con una textura de arena.

Figura 25. Planta de las carreteras y caminos representados.

4.4.8. Añadir otros objetos al entorno.

Hemos añadido más elementos a la escena, como el mirador o el pueblo, para representar el entorno con más detalle y conseguir una mejor representación de la zona.

Para crear el mirador, primero hemos utilizado la herramienta Paint Heigth del objeto Terrain para crear una superficie plana y hemos añadido una nueva textura para "pintar" la zona. También hemos utilizado cubos, añadiéndoles diferentes texturas para representar las aceras y la plataforma del mirador. Además, hemos importado un paquete del Asset Store, que incluía diferentes objetos, para añadir los bancos y las vallas

Figura 26. Mirador de la presa.

También hemos querido simular agua saliendo por el aliviadero central de la presa, para ello hemos utilizado el paquete Water FX Pack, del que solo importamos los componentes de la cascada.

Figura 27. Presa y aliviadero central soltando agua.

Igualmente hemos querido representar parte del municipio de Alcántara junto con el convento, situado entre el puente romano y Alcántara. Para ello hemos importado de nuevo algunos paquetes del Asset Store y dos modelos .fbx de la página web “3D Warehouse”, que representan el convento y la iglesia.

Figura 28.Convento y muralla.

Para la muralla que rodea el convento hemos utilizando uno los paquetes que incluía muros, torres y otros elementos, y para el pueblo uno que contenía diferentes modelos de casas que distribuimos intentando representar las principales calles de Alcántara. También se han representado algunas de las edificaciones que hay junto a la presa con los mismos modelos utilizados para el pueblo.

Figura 29.Plaza e Iglesia de Alcántara.

Figura 30.Vista en planta del convento y Alcántara.

4.4.9. Añadir audio de ambiente.

En Internet existen infinidad de bibliotecas con sonidos de todo tipo para descargar. Una vez descargado el clip lo importaremos a la carpeta Audio que crearemos previamente en la carpeta de Assets.

Para añadir un clip del audio seleccionaremos el elemento que emite el sonido, le añadiremos un componente de tipo Audio - Audio Source. Una vez añadido el componente es necesario arrastrar el clip a la ventana Audio Clip.

Dado que la mayor parte de la escena se desarrolla rodeada por la naturaleza utilizaremos como ruido de ambiente un clip de audio que simula el ruido de unos pájaros, este clip lo añadiremos al objeto que sea nuestro personaje para escucharlo constantemente, y un clip de audio de agua corriendo por un río que añadiremos al objeto que representa el río, pero con un volumen menor.

4.5. INTERACCIÓN CON EL ENTORNO.

Tras terminar con la creación del entorno vamos a trabajar en la interacción con objetos de la escena principal y en la creación de un menú de inicio.

Para ello Unity nos permite utilizar paneles, botones, animaciones, scripts (códigos), etc. Además de ofrecer diferentes paquetes de recursos para añadir personajes que podamos controlar para movernos por la escena.

Antes de mostrar con más detalle los nuevos elementos con los que interactuar, que hemos añadido, daremos una pequeña introducción a alguno sobre ellos.

4.5.1. Físicas en Unity 3D.

Para poder interactuar con los objetos que componen la escena es necesario un conocimiento básico en programación, que en nuestro caso se basará en el lenguaje de programación C#. Pero antes de avanzar es preciso explicar algunos componentes de los objetos en Unity.

Un objeto en Unity puede tener entre otros, los siguientes componentes:

- Rigidbody: permiten al objeto actuar bajo el control de la física. El Rigidbody puede recibir fuerzas o colisiones con otros objetos para hacer que el objeto asociado se mueva de una manera realista. Cualquier objeto debe contener un Rigidbody para ser influenciado por gravedad o por fuerzas externas añadidas mediante código.

- Collider: añaden una geometría de colisión al objeto, es decir, un área de interacción que permite al objeto al que se le añade reaccionar ante otros objetos que también tengan un componente Collider siempre que alguno de ellos tenga un componente Rigidbody. Como formas básicas dispone de una esfera, una cápsula o una caja, aunque puede adoptar formas más complejas usando un componente Mesh Collider. Cuando dos objetos con componentes Collider entran en contacto, sucede un evento por lo que la manera en que se comportan ambos objetos al interactuar puede ser programada mediante un script. Esta propiedad nos será muy útil más adelante, ya que el componente Collider tiene una opción para que el motor de física no tenga en cuenta los choques del colisionador y en su lugar inicie un evento que pueda tratarse por código (seleccionando la opción “Is Trigger” en el Inspector).

4.5.2. Interfaz de Usuario (UI).

Estos objetos se refieren a la interfaz de usuario que se muestra de manera bidimensional en una aplicación como textos, imágenes o botones. En Unity, estos objetos UI siempre están sobre un objeto canvas (lienzo) que representan un espacio abstracto. De manera que cuando se crea un objeto UI se creará automáticamente un canvas al que estará asociado.

Si la escena es en tres dimensiones, el objeto UI no pueden implementarse de manera tradicional, es decir, como un plano que forma parte de la pantalla sobre el que el usuario puede actuar. Por lo que para utilizar este tipo de elementos en aplicaciones en tres dimensiones será necesario ubicarlos como un objeto más en la escena. Además, para que el objeto UI sea visible al iniciar la escena es necesario elegir la opción world space en el Render Mode de su Inspector.

4.5.2.1. Canvas.

Para crear un canvas pulsaremos el botón derecho sobre la ventana Hierarchy y seleccionaremos UI – Canvas y elegiremos la opción world space.

El componente Rect Transform define la posición y dimensiones del plano sobre el que colocamos los objetos UI. Le daremos una escala y unas dimensiones acordes a su función. Una vez colocado podremos añadirle objeto UI como una imagen, un texto o botones y a su vez modificar el Anchor de su componente Rect Transform según su posición respecto al canvas.

4.5.2.2. Botones.

Para añadir un botón pulsaremos el botón derecho sobre la ventana Hierarchy y seleccionaremos UI – Button. El objeto botón en Unity está diseñado para iniciar una acción cuando el usuario lo pulse.

En el Inspector del botón aparecerán varias componentes que nos permiten ajustar su posición, forma, color o texto, pero uno de los componentes más importantes es el Button (Script), que permite definir la forma en la que responde el botón al ser pulsado.

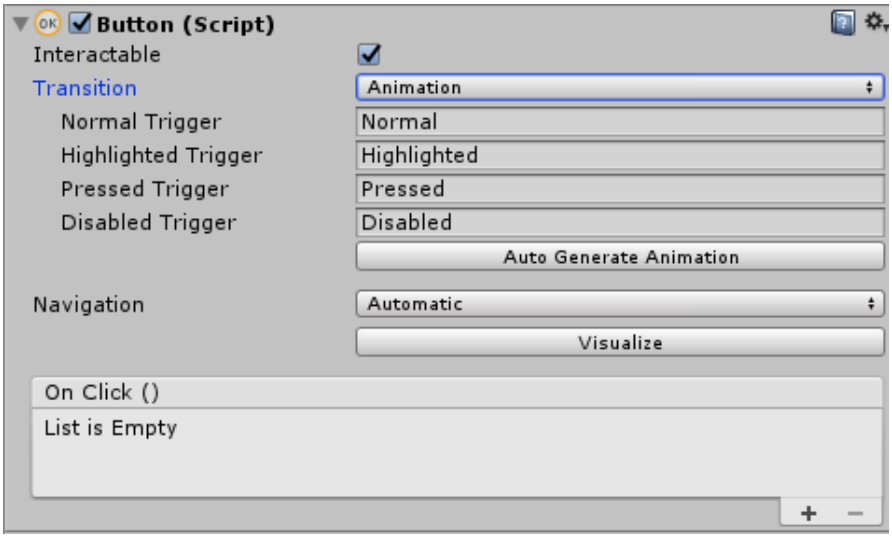

Figura 31. Componente Button (Script) de un botón.

Las propiedades que podemos modificar son:

- Interactable: sirve para definir si el botón va a ser interactivo o no.
- Transition: establece la manera en la que el control responde visualmente a las acciones del usuario. Hay varias opciones de transición y diferentes estados (normal, highlighted, pressed y disabled). Entre las opciones de transición tenemos:
    - None: no se realice ninguna transición entre los estados
    - Color Tint: cambia el color del botón dependiendo de en qué estado se encuentre. Es posible seleccionar el color para cada estado individual.
    - Sprite Swap: permite que diferentes sprites se muestren dependiendo de en qué estado se encuentre el botón.
    - Animation: permite crear animaciones dependiendo del estado del botón.
- Navigation: propiedades que determinan la secuencia de controles.

En este caso, hemos elegido la transición basada en una animación. Para crearla seleccionaremos la opción Animation y utilizaremos la opción de autogenerar una animación. Al seleccionar esta opción, el programa pide una ruta dónde se guardará la animación, por lo que crearemos una carpeta llamada Animations dentro de la carpeta Assets.

Tras crear la animación abriremos la ventana Animation desde la pestaña Window. Con el botón seleccionado podemos editar la animación para cada estado. El estado normal se suele dejar vacío y para el resto de estados se puede crear una animación, que modifique sus dimensiones o color, con un solo keyframe en el segundo 0 (inicio). La transición entre estados será controlada por el animation controller.

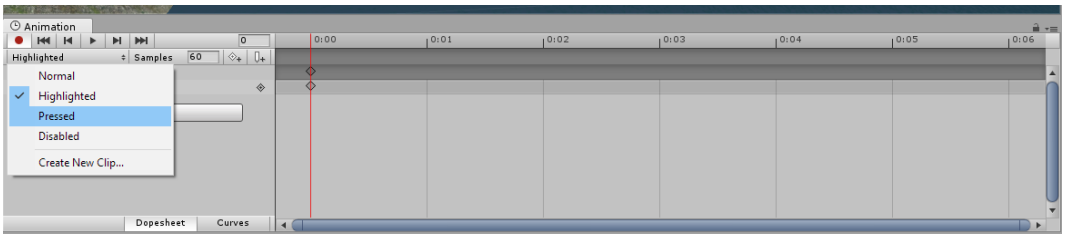

Figura 32. Ventana Animation de un botón.

Para definir qué pasará cuando el botón es presionado es necesario seleccionar el botón y en su Inspector buscar la ventana “On Click ()”, donde podemos añadir los eventos que sucederán cuando se presione el botón.

4.5.3. Introducción a la programación.

Esto es sólo un primer acercamiento a la programación en Unity, en los siguientes scripts creados se explicará un poco más a fondo su funcionamiento.

Unity es un programa por sí mismo, construido por código. Este código interno es accesible a través de la interfaz del editor (API) que se ha estado utilizando. En dicha Interfaz los scripts se muestran como componentes que se pueden configurar.

Unity nos es permite crear componentes utilizando scripts, los cuales controlan el comportamiento de los objetos, ya que nos permite activar o desactivar eventos que modifiquen las propiedades de estos objetos.

En concreto, Unity soporta dos lenguajes nativamente:

- C#, un lenguaje estándar de la industria similar a Java o C++.
- UnityScript, un lenguaje diseñado específicamente para su uso con Unity.

A la hora de utiliza un lenguaje de programación es muy importante obedecer la sintaxis del mismo. Si al crear un script Unity detecta algún error, lo mostrará a través de la ventana Console.

Para crear un script primero crearemos una carpeta en Assets, que llamaremos Scripts. Dentro de la carpeta pulsaremos el botón derecho y seleccionaremos Create - C# Script. Una vez creado podemos abrirlo con un editor para escribir el código que necesitemos. Unity incorpora un editor de código llamado MonoDevelop, aunque si se cuenta con algún otro editor, como Visual Studio de Microsoft, puede ser utilizado.

En un C# script de Unity algunos símbolos y palabras son parte del lenguaje C#, otros son parte de Microsoft .NET Framework y otros son proporcionados por la API de Unity. Un archivo C# script se compone inicialmente de las siguientes líneas:

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

public class NewBehaviourScript : MonoBehaviour {

            // Use this for initialization
            void Start () {

            }

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

            }
}
```

Figura 33. Líneas de un nuevo archivo C# script

Las primeras líneas indican que el script necesita otras librerías de código para ejecutarse. La línea using UnityEngine, permite usar la librería UnityEngine de la interfaz del programa y la línea using System.Collections, permite usar la librería de funciones relacionadas con el manejo de colecciones del .Net Framework.

Para facilitar la tarea de programar existen las clases que son como plantillas de código con sus propias propiedades (variables) y comportamiento (funciones). Las variables contienen datos de un tipo específico y las funciones permiten implementar la lógica, es decir, permiten definir las instrucciones paso a paso. Dichas funciones pueden recibir argumentos y pueden devolver nuevos valores cuando son ejecutadas.

Cuando se declara un miembro de una clase como público puede ser visto y usado por otro código en otro script, sin embargo, cuando es privado sólo puede ser Dentro de la misma clase donde se define.

Unity permite invocar funciones-mensaje especiales si se las define. Un ejemplo de ello son las funciones Start () y Update () incluidas en el código por defecto. Delante de las funciones se define el tipo de valor que devolverá.

La función Start () se llamará antes de que comience la escena por lo que se usa para dar instrucciones iniciales y la función Update () se llamará en la actualización de cada frame, es decir, a la vez que se ejecuta la escena.

4.5.4. Creación de un Menú de Inicio.

A modo de presentación, parece útil crear un menú de inicio que nos aparezca al ejecutar la aplicación. Para ello hemos duplicado nuestra escena principal (*Main*) y la hemos renombrado como *Menu*. A continuación, hemos eliminado alguno de los objetos, dejando solo aquellos elementos principales que podemos ver desde la posición elegida para colocar el menú.

Para pode ver el menú necesitamos crear una cámara que apunte hacía donde situaremos el mismo. La crearemos pulsando el botón derecho en la ventana Hierarchy y seleccionando Camera.

La idea es crear tres paneles: un panel inicial, con el título del trabajo, que dé paso a un segundo panel, menú principal, y este a su vez nos permita acceder a la escena principal, salir de la aplicación o acceder a un tercer panel, panel informativo.

El siguiente paso será crear un canvas donde situar el panel y los botones, ya que como hemos dicho, cualquier objeto UI que creemos tiene que estar asociado a un canvas. Activaremos la opción world space en el Render Mode del componente canvas para que aparezca como un objeto físico en la escena y modificaremos los parámetros del componente Rect Transform, como su posición, ancho, altura o escala.

Para mejorar la nitidez del texto que añadamos se recomienda modificar las propiedades del componente Canvas Scaler, cambiando el Dynamic Pixel Per Unit a 10.

Crearemos un panel inicial y desde el Inspector ajustaremos sus límites (Anchors), y cambiaremos a cero su posición en el Rect Transform para que quede centrado en el canvas. También podemos cambiar la imagen de los paneles y botones que creemos, para ello importaremos del Asset Store el paquete Samples UI, que incluye algunos sprites que podremos utilizar en el componente imagen de estos objetos

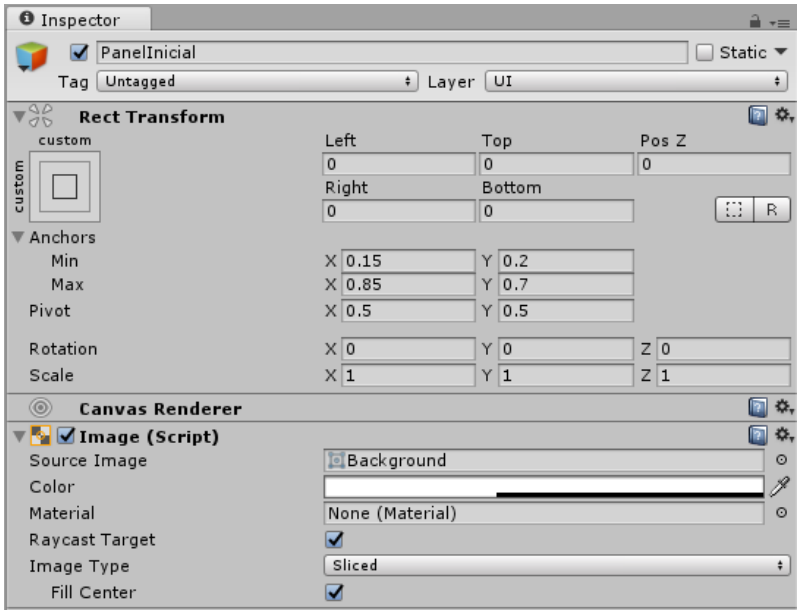

Figura 34. Inspector del panel inicial.

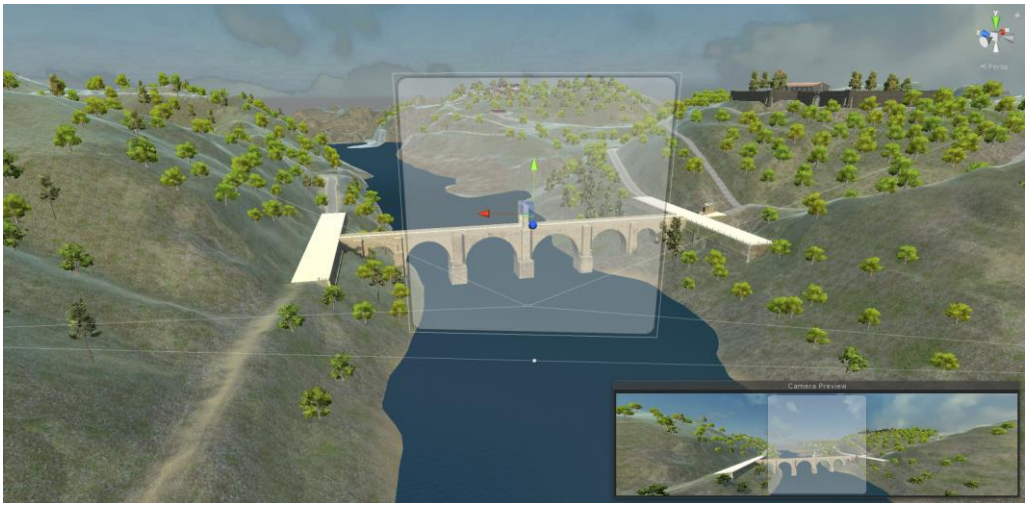

Figura 35. Canvas con un panel y vista desde la cámara.

De forma similar a la que hemos creado el panel, crearemos un objeto UI tipo Texto, con el mensaje que queramos añadir. Podremos elegir diferentes parámetros como tipo de fuente, tamaño o color.

A continuación, añadiremos los botones. Podemos colocar los botones uno a uno, ajustando su posición, o añadir al panel un componente Vertical Layout Group, que los apilará y espaciará según los creemos. En el panel inicial, simplemente añadiremos un botón (CONTINUAR) que nos de paso a un siguiente panel que será el menú principal.

Figura 36. Panel inicial.

Como ya hemos explicado anteriormente, al añadir un botón podremos elegir el tipo de transición entre sus estados, en nuestro caso utilizaremos una animación que modificará las dimensiones y el color del botón. Además, podemos elegir qué ocurrirá cuando el botón es presionado desde la ventana “On Click ()”.

Al igual que hemos creado el panel inicial, crearemos el panel del menú principal y del panel informativo, con los objetos UI que necesitemos.

Para el panel del menú principal hemos elegido y hemos añadido tres botones: si pulsamos EMPEZAR, se cargará la escena principal, si pulsamos INFORMACIÓN, se abrirá el panel informativo y si pulsamos SALIR, finaliza la aplicación.

Figura 37. Panel menú principal.

En cambio, el panel informativo cuenta con un texto y un botón para regresar al menú principal (VOLVER), por lo que para crearlos más fácilmente podemos copiar el panel inicial y modificarlo.

Figura 38. Panel informativo.

4.5.4.1. Configuración botones.

- CONTINUAR.

Crearemos dos eventos On Click, uno para desactivar el panel inicial (Function - GameObject.SetActive – False) y otro para activar el panel del menú principal (Function – GameObject.SetActive – True).

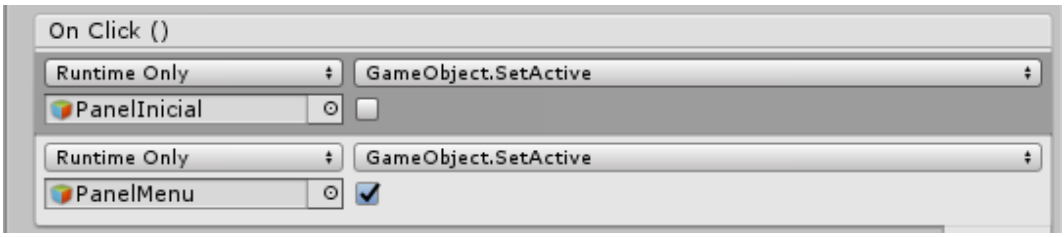

Figura 39. Ventana On Click del botón CONTINUAR.

- EMPEZAR.

Inicialmente, tendremos que añadir las escenas Menu y Main en Build Settings, abrimos la pestaña File y seleccionamos Build Settings. Primero añadiremos la escena Menu, con el índice 0 (primera escena que cargará al iniciar el juego) y después añadiremos la escena principal, con el índice 1.

También crearemos el siguiente script (*CargarEscena*):

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

public class CargarEscena : MonoBehaviour {

    public void LoadByIndex(int sceneIndex)
    {
        SceneManager.LoadScene(sceneIndex);
    }
}
```

Este script añade la función LoadByIndex que integrará un parámetro llamado SceneIndex. Cuando la función sea llamada, cargará la escena usando la clase SceneManager (UnityEngine).

Tras añadir el script al botón y crear el evento On Click, elegiremos la función LoadByIndex y el índice de la escena que queramos cargar, en este caso 1.

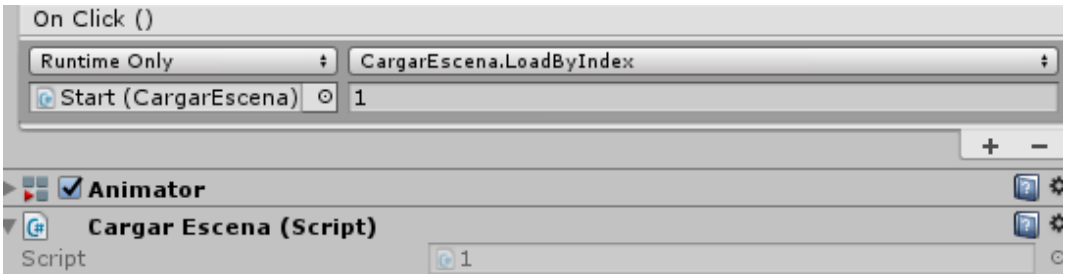

Figura 40.Ventana On Click del botón EMPEZAR y componentes Animator y C# Script.

- INFORMACIÓN

Crearemos los mismos eventos que para el botón CONTINUAR, pero desactivaremos el panel del menú principal y activaremos el panel informativo.

- VOLVER.

Crearemos los mismos eventos que para el botón INFORMACIÓN, pero desactivaremos el panel informativo y activaremos el panel del menú principal.

- SALIR.

Tendremos que crear el siguiente script (*Salir*), que añadiremos al botón.

```
using UnityEngine;
using System.Collections;

public class Salir : MonoBehaviour {

    public void Quit()
    {
#if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
#else
        Application.Quit ();
#endif
    }
}
```

Este script utiliza una compilación especifica según la plataforma, es decir, el script correrá de manera diferente si estamos en el editor de Unity o no. En el caso del editor, al pulsar el botón saldremos del play mode, y en el caso del ejecutable, cerrará la aplicación.

Tras añadir el script al botón y crear el evento On Click, elegiremos la función Quit.

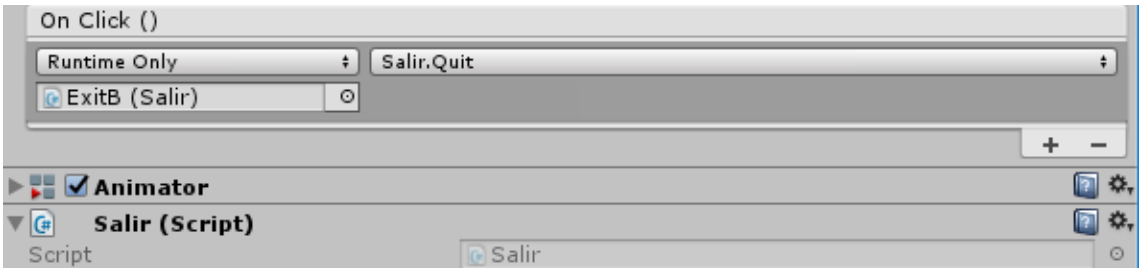

Figura 41. Ventana On Click del botón SALIR y componentes Animator y C# Script.

4.5.5. Pulsar botones fijando la mirada.

Una vez creados los botones, necesitamos una forma de pulsarlos sin la necesidad de utilizar un ratón. Debido a que mientras utilizamos las gafas de realidad virtual, nos interesa no depender de un ratón o un teclado para estar más cómodos a la hora de movernos por la escena. Para pulsar los botones al mirarlos podemos utilizar un componente Collider asociado a un objeto vacío (Empty GameObject) que superpondremos sobre el botón para que sea interceptados por el rayo coincidente con el centro de la visión.

Como los botones utilizados son rectangulares, se añadirán componente box Collider. Pero para que el rayo pueda distinguir estos objetos del resto en la escena, se creará una nueva capa donde se colocarán únicamente estos objetos. Para ello, seleccionado los objetos vacíos, iremos a su Inspector, seleccionaremos la pestaña Tag y añadiremos uno nuevo llamado *Botones* seleccionando la opción “Add Tag…”.

Figura 42. Botón CONTINUAR y en verde, malla del Collider asociado.

A continuación, necesitamos generar un script que marque los botones cuando el centro de visión se posa sobre ellos. Para ello se ha programado el siguiente script (*EjecutarBoton*), que tendremos que añadir a todos los botones que creemos:

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

public class EjecutarBoton : MonoBehaviour
{

    private GameObject currentButton;

    void Update()
    {
        Transform camera = Camera.main.transform;
        Ray ray = new Ray(camera.position, camera.rotation *
Vector3.forward);
        RaycastHit hit;
        GameObject hitButton = null;
        PointerEventData data = new
PointerEventData(EventSystem.current);

        if (Physics.Raycast(ray, out hit))
```

```
        {
            if (hit.transform.gameObject.tag == "Botones")
            {
                hitButton = hit.transform.parent.gameObject;
            }
        }
        if (currentButton != hitButton)
        {
            if (currentButton != null)
            {//unhighlight

ExecuteEvents.Execute<IPointerExitHandler>(currentButton, data,
ExecuteEvents.pointerExitHandler);
            }
            currentButton = hitButton;
            if (currentButton != null)
            {//highlight

ExecuteEvents.Execute<IPointerEnterHandler>(currentButton, data,
ExecuteEvents.pointerEnterHandler);

            }
        }
    }
}
```

El funcionamiento del script es el siguiente:

- Se declara privada la variable tipo objeto referente al botón.
- Se crean unas variables para referirse a la cámara, al rayo de visión central, y al objeto sobre el que colisionará. Tras lo que se define el rayo en función de la posición de la cámara y la colisión de éste con cualquier elemento.
- Se define el comportamiento del botón en caso de que el rayo colisione con un objeto ubicado en la capa Botones. En cada frame se capta el rayo que parte del centro de visión de la cámara y se comprueba si ha encontrado un objeto de la capa Botones. En caso de que lo haga se marca al padre del elemento (el botón en sí) como seleccionado (highlight).
- De forma parecida detecta si el rayo ha salido del objeto para deseleccionarlo (unhighlight).

De esta manera podemos seleccionar los botones al fijar la mirada sobre ellos, pero necesitamos que se pulsen los botones para que desencadene el evento definido en la ventana On Click. Por ello tenemos que completar el script de la siguiente forma:

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

public class EjecutarBoton : MonoBehaviour
{

    public float timeToSelect = 2.0f;
    private float countDown;
    private GameObject currentButton;

    void Update()
    {
        Transform camera = Camera.main.transform;
        Ray ray = new Ray(camera.position, camera.rotation *
Vector3.forward);
        RaycastHit hit;
        GameObject hitButton = null;
        PointerEventData data = new
PointerEventData(EventSystem.current);

        if (Physics.Raycast(ray, out hit))
        {
            if (hit.transform.gameObject.tag == "Botones")
            {
                hitButton = hit.transform.parent.gameObject;
            }
        }
        if (currentButton != hitButton)
        {
            if (currentButton != null)
            { //unhighlight

ExecuteEvents.Execute<IPointerExitHandler>(currentButton, data,
ExecuteEvents.pointerExitHandler);
            }
            currentButton = hitButton;
            if (currentButton != null)
```

```
            {//highlight

ExecuteEvents.Execute<IPointerEnterHandler>(currentButton, data,
ExecuteEvents.pointerEnterHandler);
                countDown = timeToSelect;
            }
        }
        if (currentButton != null)
        {
            countDown -= Time.deltaTime;
            if (countDown < 0.0f)
            {

ExecuteEvents.Execute<IPointerClickHandler>(currentButton, data,
ExecuteEvents.pointerClickHandler);
                countDown = timeToSelect;
            }
        }
    }
}
```

Esto añade un contador. De manera que, si se fija la mirada sobre el objeto durante un tiempo superior al establecido, que hemos definido como una variable pública para poder cambiar siempre que queramos desde el Inspector, se pulsará el botón. En concreto, hemos elegido que el tiempo para seleccionar el botón sea de dos segundos.

4.5.6. Añadir Puntero para seleccionar.

Para poder apuntar fácilmente y con exactitud a los botones vamos a utilizar un objeto UI que haremos dependiente de la cámara y hará la función de puntero.

Por tanto, crearemos un canvas, asociado a la cámara, que llamaremos *Puntero* y que situaremos en la posición (0,0,0). A este canvas le añadiremos una imagen centrada de tamaño reducido (0,04x0,04) y la situaremos en la posición (0,0,2), de manera que, quedará en el centro de la visión y alejado sobre el eje z.

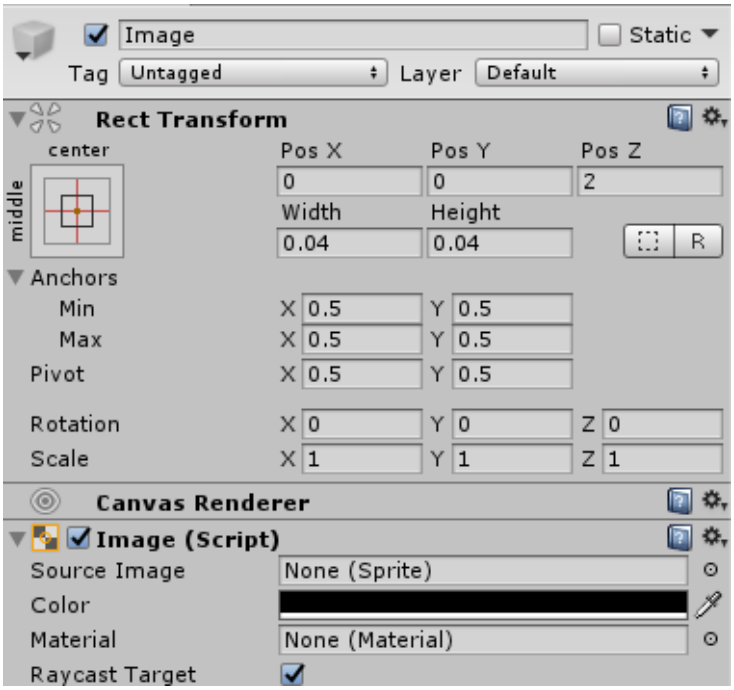

Figura 43. Inspector de la imagen asociada al puntero (canvas).

4.5.7. Personajes

Tras crear el menú, volveremos a la escena principal, donde crearemos el avatar o personaje que nos permita desplazarnos por la escena e interactuar con los elementos del entorno. Por ello, Unity ofrece un paquete llamado Characters, que importaremos desde Assets - Import Package, con dos tipos de personajes según el punto de vista que utilizan:

- Personaje en tercera persona: situando una cámara sobre el personaje y asociándola a este, participaremos en la escena como observador. Podremos desplazar el personaje por la escena gracias a un script de control de movimiento que incluye.
- Personaje en primera persona: simula la visión del personaje. Este tipo de personaje es el apropiado para la Realidad Virtual ya que nos permite dar la sensación de estar en la escena. También podremos desplazar el personaje por la escena gracias a un script de control de movimiento.

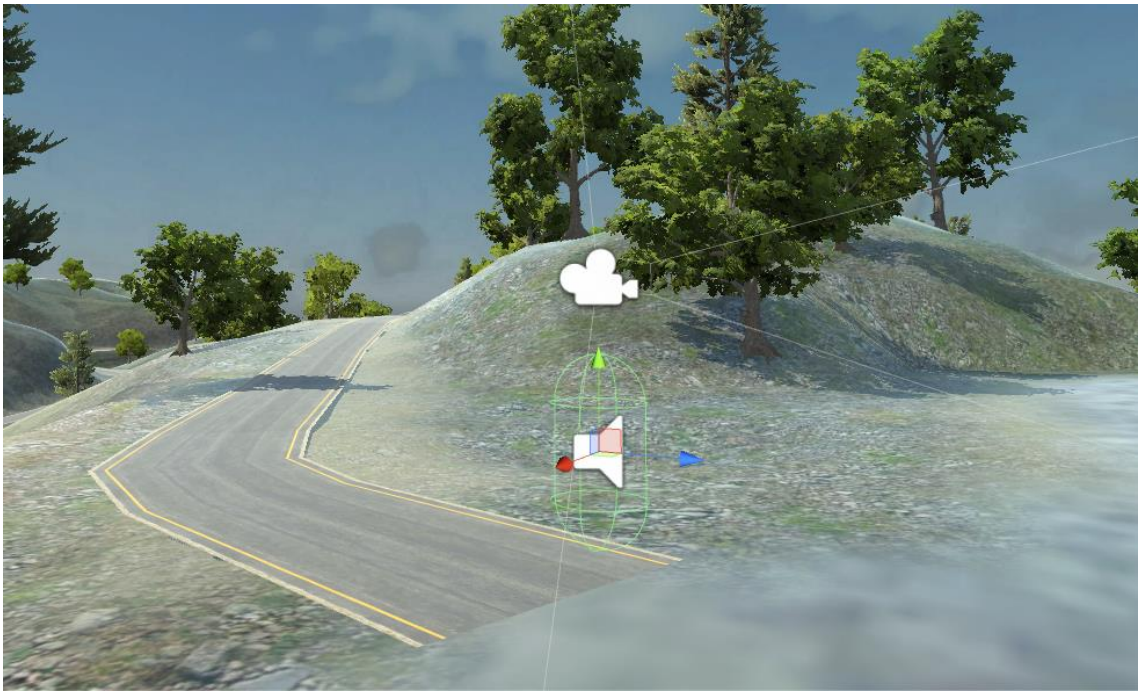

Figura 44. Escena con el personaje en primera persona y la cámara asociada

Para incorporar el personaje a la escena es necesario añadir el objeto prefabricado. Para añadir este elemento bastará con buscarlo en la carpeta Assets – StandardAssets - Characters – FirstPersonCharacter - Prefabs y arrastrarlo a la escena. Este prefabricado también incluye varios componentes como una cámara o un receptor de audio.

Entre los dos prefabricados de personajes en primera persona que ofrece Unity, para el uso de la Realidad Virtual se recomienda el Rigid Body FPC, ya que no simula el movimiento del personaje al dar pasos, lo que podría crear una sensación incómoda para el usuario, y además adapta los ejes de movimiento a la dirección a la que se enfoca lo que es esencial para el usuario de dispositivos de realidad virtual.

Por último, será necesario colocarlo en el punto en el que queramos que se encuentre el personaje al comienzo de la aplicación. Siempre que se ejecute el programa comenzará en dicho punto, a partir del cual el usuario podrá moverse libremente por la escena. Además, cambiaremos su nombre a *Player* ya que a la hora de programar los scripts que utilizaremos en los diferentes elementos será útil para poder diferenciarlo.

Antes de continuar, ejecutamos la aplicación la opción de Realidad Virtual activada desde la pestaña Edit – Project Settings - Player, y nos desplazaremos por la

escena para buscar una altura de cámara y una velocidad para desplazarnos que nos resulte cómoda. En nuestro caso, colocaremos la cámara a 1,8 m sobre el suelo y para evitar malestar ajustaremos la velocidad a tres. Para ello iremos al Inspector del personaje y modificaremos las velocidades en los ajustes de movimiento (Movement Settings).

4.5.8. Configuración de los controles del personaje.

Unity utiliza por defecto el teclado y el ratón para controlar el movimiento del personaje. Por tanto, para desplazar nuestro personaje por la escena utilizaremos las flechas del teclado y para mover la cámara el ratón o si activamos la opción de Realidad Virtual nos desplazaremos con las flechas y moveremos la cámara al mover la cabeza.

La posibilidad de estar de pie y alejado del equipo para moverte libremente nos permite conseguir un mayor realismo y una mejor inmersión en la escena. Por ello, hemos utilizado las siguientes dos opciones para desplazarnos por la escena mientras utilizamos las gafas de realidad virtual:

1- Usar un mando de Xbox One inalámbrico, como las flechas del teclado, para desplazarse por la escena. Para este modo, valdría con utilizar el script que ya incluye nuestro personaje por defecto, *RigidbodyFirstPersonController*. Aunque en caso de ser necesario configurar el mando, iremos a Edit - Project Settings – Input, donde podemos seleccionarse que botones controlan el movimiento del personaje.

2- Usar el mando a distancia que incluyen las Oculus Rift para iniciar o detener la marcha al pulsar un botón, y dirigir el movimiento del personaje enfocando hacia donde queramos movernos. De esta manera conseguimos una mayor inmersión y simplificamos los controles necesarios a la hora de moverse por la escena. En este caso, será necesario desactivar el script *RigidbodyFirstPersonController* para evitar temblores de la cámara e incorporar el siguiente script (*MoverMirada*) que hemos programado:

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

public class MoverMirada : MonoBehaviour {
    public float velocity = 1.0f;

      void Update () {
```

```
        Vector3 moveDirection = Camera.main.transform.forward;
        moveDirection = moveDirection * velocity * Time.deltaTime;
        transform.position = transform.position + moveDirection;
      }
}
```

El funcionamiento del script es el siguiente:

- Se publica la variable velocidad, para poder elegir desde el Inspector la velocidad que queramos.
- Conforme se ejecuta la aplicación se comprueba la dirección en la que apunta la cámara y mueve el objeto Player en esa dirección a la velocidad indicada.

Añadiendo este script como un nuevo componente en el objeto Player y eligiendo la velocidad que queramos, en nuestro caso tres, conseguiremos que nuestro personaje se mueva hacia dónde enfoquemos la mirada, pero queremos poder parar e iniciar la marcha pulsando el botón del mando por lo que lo completaremos de la siguiente forma:

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

public class MoverMirada : MonoBehaviour {
    public float velocity = 1.0f;
    public bool isWalking = false;
    private Clicker clicker = new Clicker();

      void Update () {
        if (clicker.clicked())
        {
            isWalking = !isWalking;
        }
        if (isWalking)
        {
            Vector3 moveDirection = Camera.main.transform.forward;
        moveDirection = moveDirection * velocity * Time.deltaTime;
        transform.position = transform.position + moveDirection;
        }

      }
}
```

El funcionamiento del script es parecido al anterior, pero incluye:

- Una nueva clase privada (*Clicker*), que permite comprobar si se pulsa algún botón. La crearemos previamente escribiendo el siguiente script:

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

public class Clicker {

      public bool clicked ()

      {
      return Input.anyKeyDown;
      }
}
```

- Una nueva clase pública de tipo booleano, representa valores binarios, que activa o desactiva el movimiento al pulsar un botón.

4.5.9. Creación de Carteles interactivos.

Conforme nos desplacemos por la escena iremos encontrando diferentes carteles. Estos tendrán diferentes funciones que explicaremos más adelante, como elegir el tipo de control que queremos usar para mover nuestro personaje o nos irán dando información sobre las infraestructuras que añadimos anteriormente.

4.5.9.1. Cartel para seleccionar los controles del personaje.

Antes de empezar a movernos por la escena queremos poder seleccionar los controles con los que mover nuestro personaje, para ello crearemos un cartel con dos botones con los que elegir el tipo de control que queremos utilizar. Este cartel lo colocaremos justo delante del personaje para que sea lo primero que veamos al llegar a la escena principal.

Para crear este cartel hemos utilizado exactamente los mismos conceptos explicados anteriormente para la creación de un Menú de Inicio.

En este caso, crearemos un canvas que llamaremos *CartelControles*, ajustaremos su escala y tamaño y le añadiremos varios objetos UI: un texto, una imagen, para añadir un fondo blanco, y dos botones.

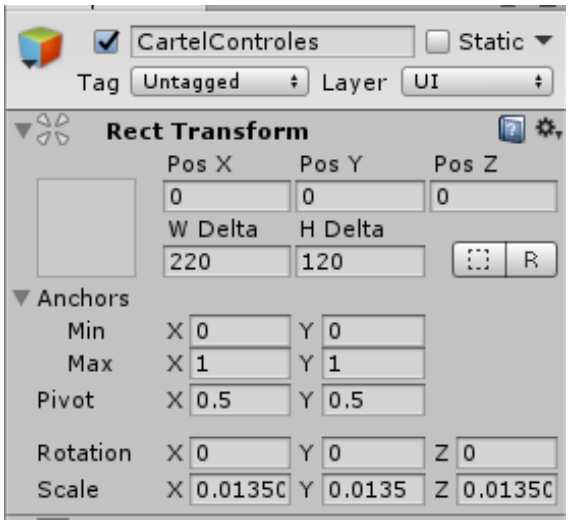

Figura 45. Componente Rect Transform del canvas CartelControles.

Importante seleccionar la opción world space y utilizar un valor de 10 para el atributo Dinamic Pixel Per Unit en el canvas. También deberemos cuidar de seleccionar el Tag Botones en los Collider de los objetos vacíos que coloquemos a los botones y de añadirles el script *EjecutarBotón* para poder pulsarlos fijando la mirada

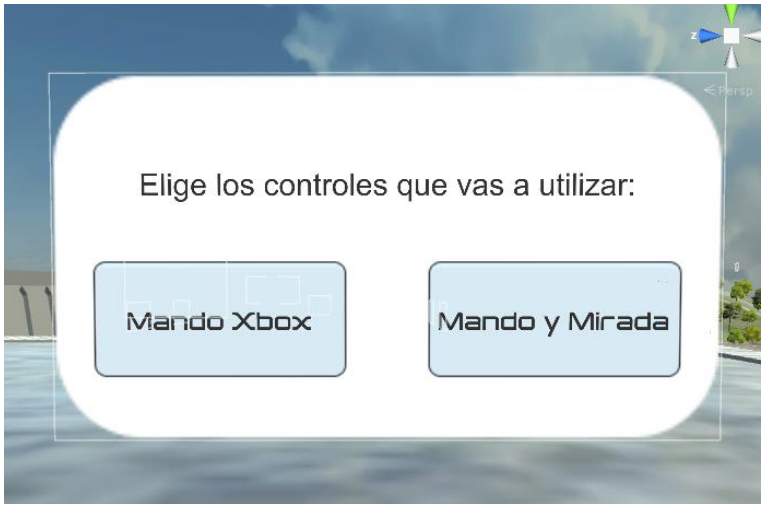

Figura 46. Canvas CartelControles.

Al iniciar la escena el personaje tendrá desactivado los dos scripts que controlan su movimiento, pero cuando pulse uno de los botones activará el script correspondiente

y desactivará el otro, permitiendo que podamos probar ambos tipos de control. El botón *Mando Xbox* activa el script *RigidbodyFirstPersonController* y mantiene desactivado el script *MoverMirada*, en cambio, el botón *Mando y Mirada* hace justo lo contrario.

Por tanto, crearemos dos eventos en la ventana On Click de cada botón asignándoles los scripts mencionados y eligiendo la función GameObject.SetActive (true o false) para activar o desactivar los scripts.

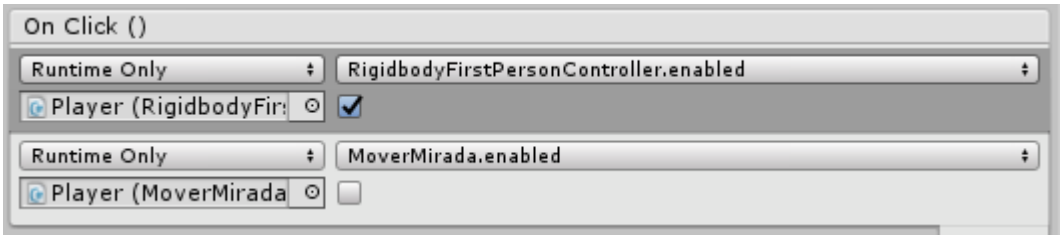

Figura 47. Ventana On Click del botón Mando Xbox.

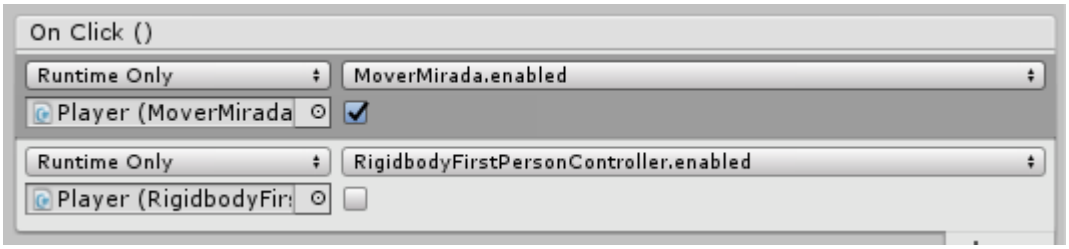

Figura 48. Ventana On Click del botón Mando y Mirada.

Como no queremos que los carteles aparezcan constantemente en la escena, utilizaremos un objeto que llamaremos A*ctivador*, para que solo aparezcan cuando estemos cerca. Esto lo crearemos mediante un objeto vacío con un componente Collider (con la opción is Trigger activada) y con el script *Activador* que hemos programado, cuyo evento se desencadenará al acerquemos al Collider.

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


public class Activador : MonoBehaviour
{
    public GameObject Objeto;
      // Use this for initialization
```

```
  void Start ()
{
    Objeto.SetActive(false);
  }

void OnTriggerEnter(Collider other)
{
    if (other.gameObject.name == "Player")
    { Objeto.SetActive(true); }
}
void OnTriggerExit(Collider other)
{
    if (other.gameObject.name == "Player")
    { Objeto.SetActive(false); }
}

  }
```

El funcionamiento del script es el siguiente:

- Se hace pública la variable *Objeto* que será aquel que aparecerá al acercarnos. De forma que sea visible en el Inspector y podamos asignarle un objeto.
- Antes de iniciar la escena se desactiva el objeto.
- Al iniciar la escena se utiliza una función que detecta cuando un objeto ha entrado en el área del Collider al que se añade el script y además si el objeto que entra en contacto se llama *Player* (nuestro personaje) se activa el objeto al que hemos llamado *Objeto*.
- Por último, se utiliza una función parecida a la anterior, pero que detecta cuando el objeto *Player* ha salido del área definida por el Collider para desactive el objeto al que hemos llamado *Objeto*.

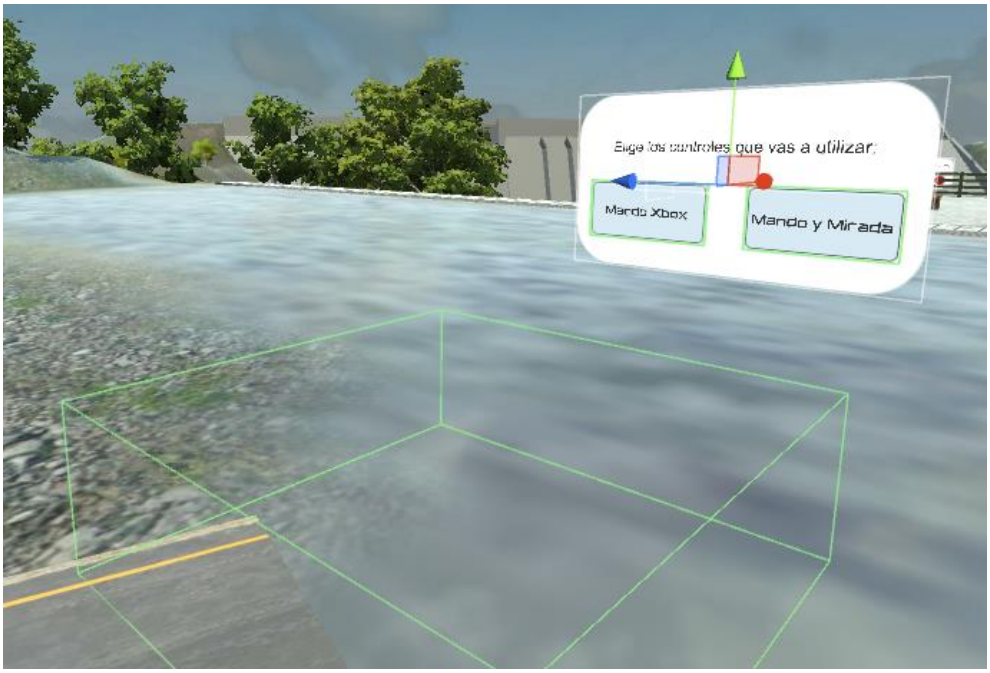

Figura 49. Canvas CartelControles con su activador (Collider),

A continuación, añadiremos el script al Activador como un nuevo componente. Una vez añadido nos aparecerá en el Inspector una casilla para añadir el objeto que queremos que sea activado, en este caso, el cartel. Por lo que se selecciona y se arrastra a la ventana.

Al igual que en la escena del menú, necesitaremos un puntero, que crearemos como explicamos en el apartado del menú y que solo será visible cuando necesitemos pulsar un botón de algún cartel, ya que en otro caso sería incomodo tener constantemente un punto en nuestro campo de visión. Por tanto, utilizaremos el mismo activador y script que activa el cartel.

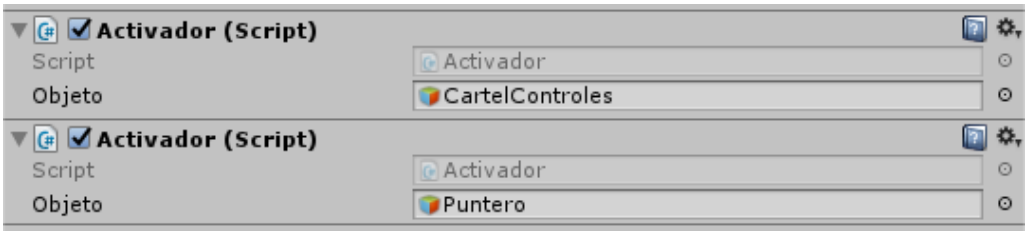

Figura 50.Componentes Activador (script) del Activador del CartelControles.

Además, para que los carteles sean cómodos de leer, necesitamos que estén siempre orientados hacia la cámara de nuestro personaje. Por ello, hemos programado el siguiente script *SeguirMirada,* que añadiremos a nuestro personaje principal.

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

public class SeguirMirada : MonoBehaviour {
    public GameObject Terreno;
    public Transform Cartel;

    void Start()
    {        }

    void Update()
    {
        Transform camera = Camera.main.transform;
        Ray ray;
        RaycastHit[] hits;
        GameObject hitObject;

        ray = new Ray(camera.position, camera.rotation *
Vector3.forward);
        hits = Physics.RaycastAll(ray);

        for (int i = 0; i < hits.Length; i++)
        {
            RaycastHit hit = hits[i];
            hitObject = hit.collider.gameObject;

            if (hitObject == Terreno)
            {
                Cartel.LookAt(camera.position);
                Cartel.Rotate(0.0f, 180.0f, 0.0f);
            }
        }
      }
     }
```

El funcionamiento del script es el siguiente:

- Se hacen públicas las variables terreno y el cartel para poder elegirlas desde el Inspector.

- Se crean unas variables para referirse a la cámara, al rayo de visión central, y al objeto sobre el que colisionará. Además, se define el rayo en función de la posición de la cámara y la colisión de éste con cualquier elemento.
- Para definir el comportamiento de los objetos, se utiliza un ciclo for que proyecta el rayo y registrar si colisiona con un objeto. Además, si el rayo colisiona con el terreno, es decir, no se está mirando al cartel, éste se orientará hacia la cámara a través del comando "LookAt". Este permitirá que el cartel quede enfocado hacia la dirección de la mirada cuando el usuario vuelva a mirarlo por lo que es necesario rotarlo 180 grados alrededor del eje y.

A continuación, añadiremos el script al personaje *Player* y arrastraremos en la casilla Terreno el objeto Terrain y a la casilla Cartel el cartel que queremos que se oriente en el Inspector. Si queremos que más de un objeto se oriente, por ejemplo, todos los carteles será necesario escribirlo en el script añadiendo nuevas variables, como la variable Cartel, copiando las líneas en las que aparezca.

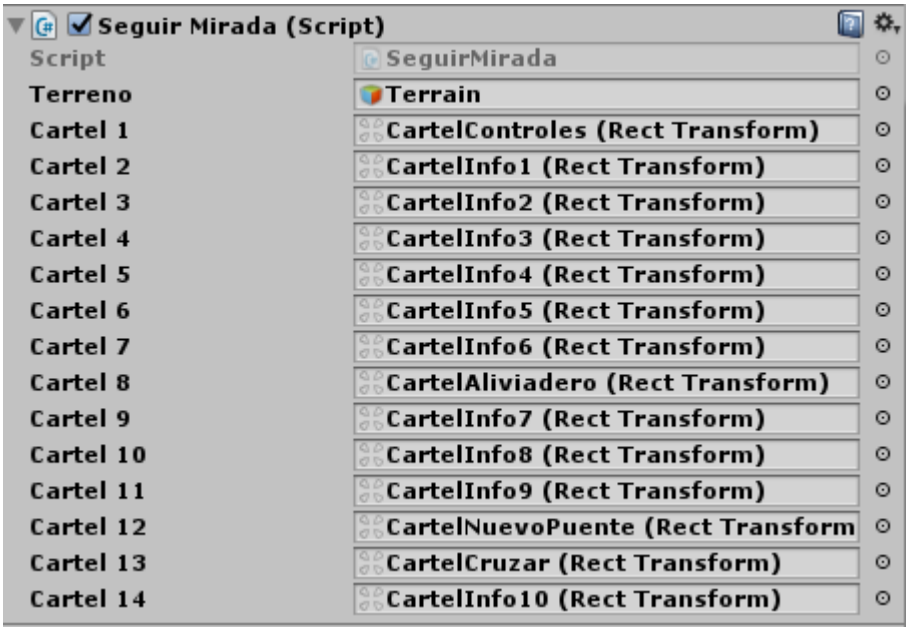

Figura 51.Componente SeguirMirada (script) con varios objetos añadidos.

Para no saltarnos ningún cartel o punto importante de la escena, podemos utilizar marcadores para indicar su posición. En nuestro caso añadiremos esferas naranjas y en algunos casos se utilizará un personaje secundario para hacer la escena más llamativa.

Crearemos un personaje secundario utilizando el prefabricado Third Person Controller, que encontraremos en la carpeta Characters, importada anteriormente. Pero

para utilizar un cuerpo más adecuado, descargaremos un modelo .fbx en cualquier biblioteca gratuita de Internet, que añadiremos a la escena y centramos en el del Third Person Controller. A continuación, eliminamos el cuerpo del Third Person Controller y sus scripts, y le asociamos nuestro nuevo personaje.

Así conseguiremos que nuestro personaje secundario mantenga la animación, para que no permanezca completamente estático, pero no el controlador para desplazarlo. También forzaremos su posición y rotación en todas las direcciones, en su componente Rigidbody, para evitar que al chocar con él se desplace o gire.

Figura 52. Prefabricado del Third Person Controller (izq.) y nuevo modelo (der.).

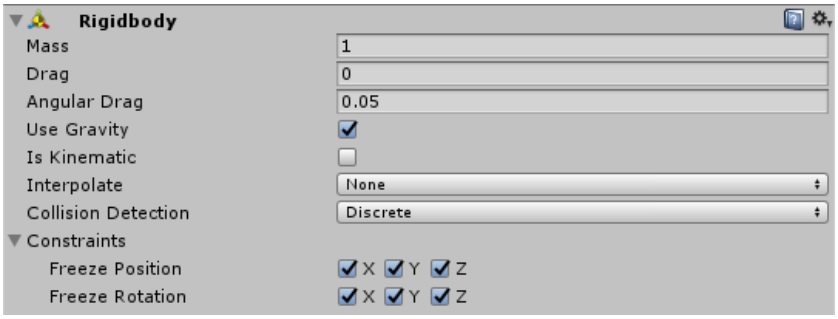

Figura 53.Componente Rigidbody del personaje secundario.

4.5.9.2. Carteles informativos.

Una vez hayamos seleccionado los controles de nuestro personaje podremos movernos por la escena para ir leyendo instrucciones o información sobre los elementos que hayamos añadido. Para ello utilizaremos carteles muy parecidos al creado anteriormente para seleccionar los controles, por lo que podemos duplicar el canvas *CartelControles* para crearlos más fácilmente utilizando los objetos UI y el activador que ya hemos añadido.

Una vez copiado y colocado en su nueva posición, cambiaremos su nombre (por ejemplo, a *CartelInfo1),* lo añadiremos al componente Activador (script) de su activador y al componente SeguirMirada (script) de nuestro personaje. Por tanto, solo aparecerán cuando nos acerquemos a su activador y estarán siempre orientados hacia nuestra cámara.

Además, eliminaremos los botones y ajustar las dimensiones del canvas y del texto según necesitemos.

Figura 54. Carteles informativos añadidos a la escena con su marcador.

4.5.9.3. Carteles para activar o desactivar objetos.

Volviendo a utilizar como plantilla el canvas de *CartelControles,* crearemos un cartel que nos permita activar o desactivar elementos de la escena. En nuestro caso, nos permitirá simular la apertura y cierre de un aliviadero de la presa o para activar el nuevo puente de Alcántara y mostrar cómo quedaría una vez se construya.

De manera que, copiaremos el canvas y seguiremos las mismas pautas que en el punto anterior, pero en este caso no eliminaremos los botones, sino que simplemente editaremos sus componentes. Podemos cambiar su forma, dimensiones, o incluso la animación utilizada en las transiciones de estado.

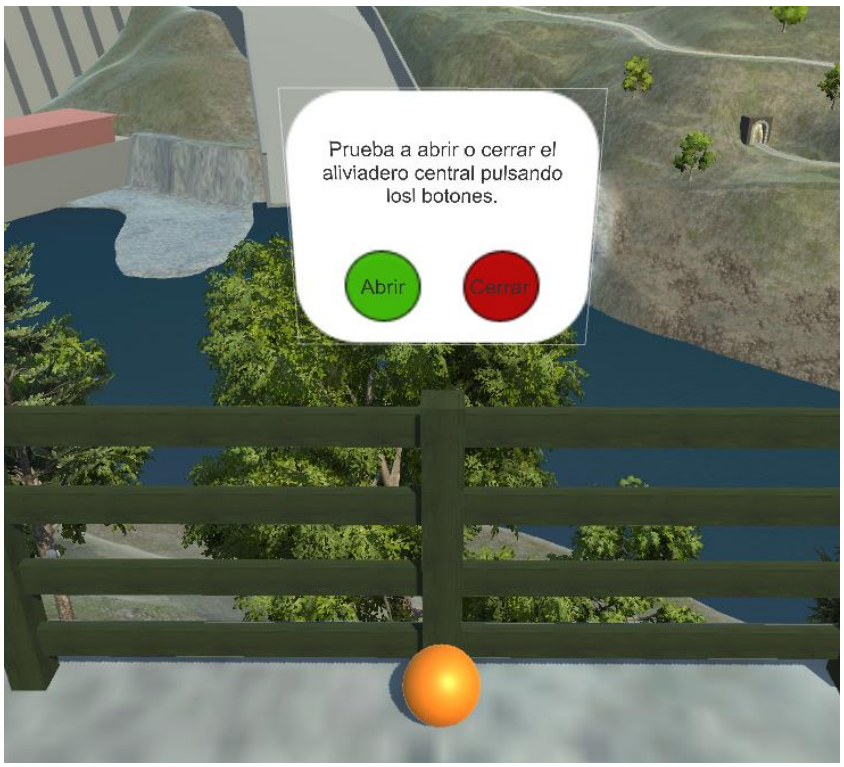

Figura 55. Cartel para abrir y cerrar el aliviadero

También tendremos que cambiar los eventos que se desencadenan al pulsarlos, por ejemplo, para el caso del aliviadero, uno de los botones activará el objeto que simula el agua saliendo por el aliviadero y otro lo desactivará.

Figura 56. Ventana On Click del botón Abrir (sup.) y ventana On Click del botón Cerrar (inf.).

4.5.10. Otras formas de moverse por la escena.

La forma más sencilla de desplazarnos por la escena es utilizar los controles que mueven a nuestro personaje, pero existen otras formas bastante interesantes que veremos a continuación, como crear una animación que mueva a nuestro personaje por un paseo predefinido o utilizar un script para trasladar de manera instantánea nuestro personaje hasta otro punto.

4.5.10.1. Animación de un recorrido predefinido.

Gracias a una animación, podemos crear un recorrido predefinido por la escena, que permita al usuario controlar la visión del personaje, pero no el movimiento.

Para crear la animación seleccionaremos el objeto que queramos que se mueva, desde la pestaña Windows abriremos la ventana Animation y pulsaremos el botón Create. Una vez creada la animación podemos elegir las propiedades del objeto que queremos modificar en el tiempo, en nuestro caso sería la posición.

Desde ventana Animation podemos crear eventos o keys sobre la línea temporal, que se mide en frames y en segundos, en los que cambiaremos las propiedades. La línea roja nos indica el frame que se está modificando.

Junto a la línea de tiempo tenemos los botones de grabar y de reproducir. Al pulsar el botón de grabar, los cambios hechos sobre el objeto serán agregados automáticamente a la animación, y al pulsar el botón reproducir visualizaremos la animación.

La propiedad Samples permite definir cuántos segundos equivalen a un frame, así que, la animación será más lenta si reducimos este número. Cuando utilicemos una animación sobre un objeto con cámara, utilizaremos valores de Samples bajos para no generar malestar en el usuario, en nuestro caso utilizaremos valores entre seis y cuatro.

En este caso queremos crear una animación simulando que nuestro personaje conduce un coche, por ello añadiremos un modelo .fbx de un coche a la escena, que hemos descargado he importaremos a Unity como hicimos con otros modelos. Una vez colocado el modelo, que llamaremos *Coche1*, en la posición deseada, duplicaremos nuestro personaje, al que llamaremos *Player2,* que colocaremos en el interior del coche y asociaremos a este, arrastrándolo en Hierarchy sobre él. También tendremos que restringir su posición en el eje vertical (y) para evitar sacudidas en la cámara y eliminar los scripts de control de movimiento para que cuando se inicie la animación se mueva el coche con nuestro personaje dentro, pero solo nos permita controlar la cámara.

Figura 57.Modelo de Coche1 con el objeto Player2 asociado.

Como ya hemos explicado, para añadir keys a la animación, seleccionaremos el objeto *Coche1* y modificaremos su posición y rotación en cada frame que vayamos seleccionando con la línea roja.

Figura 58.Ventana Scene creando la animación y ventana Animation.

Una vez creada la animación, al ejecutar la aplicación se iniciaría automáticamente, pero nos interesa iniciarla cuando nosotros queramos. Por tanto, basándonos en el activador utilizado para los carteles, crearemos un activador de la animación.

Para ello añadiremos un segundo coche, *Coche2*, que colocaremos en el punto desde donde queramos activar la animación y junto a él colocaremos un objeto vacío con un componente Collider, con la opción Is Trigger activada, al que añadiremos el script *MontarCoche* que hemos programado.

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

public class MontarCoche : MonoBehaviour {

    public GameObject Personaje1;
    public GameObject Personaje2;

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.name == "Player")
```

```
        {
            Personaje1.SetActive(false);
            Personaje2.SetActive(true);
        }
    }
}
```

Este script es parecido al utilizado para activar los carteles y su funcionamiento es el siguiente:

- Se hacen públicas las variables *Personaje1 y Personaje2* para que sean visibles en el Inspector y podamos asignarles un objeto.
- Al iniciar la escena se utiliza una función que detecta cuando un objeto ha entrado en el área del Collider al que se añade el script y además si el objeto que entra en contacto se llama *Player* (nuestro personaje) se activa el *Personaje2* y se desactiva el *Personaje1*.

En este caso, el *Personaje1* será el objeto *Player* y el *Personaje2* será *Coche1,* con el que hemos creado la animación, de manera que se desactivará *Player* y su cámara y se activará el *Coche1* y su animación, el cual contiene a su vez a *Player2* que nos permitirá ver desde el interior del coche como se desplaza.

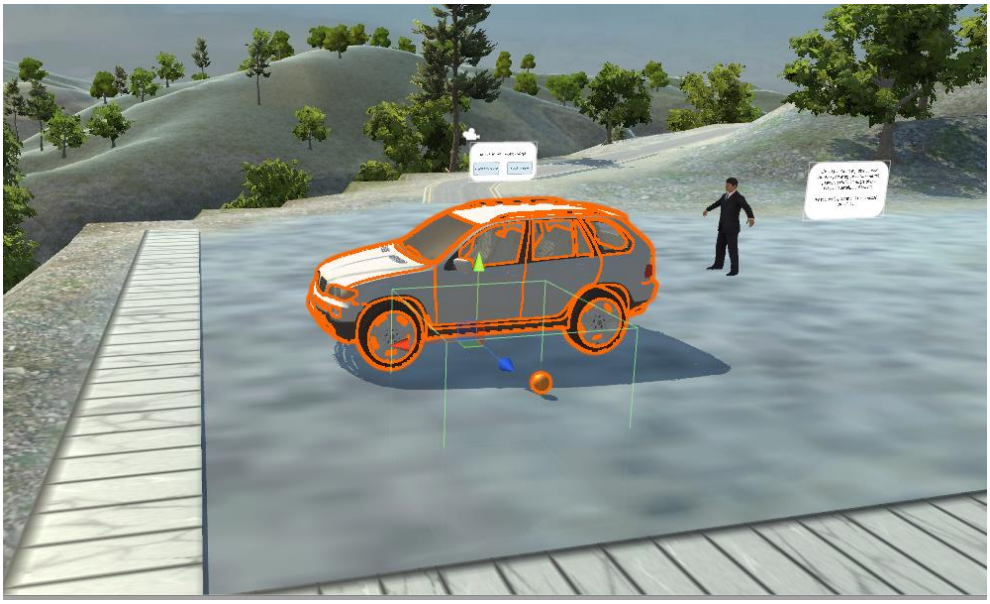

Figura 59. Coche2 desde el que activamos la animación con un collider.

Basándonos en el script anterior crearemos un script para ir desactivando los objetos que vayamos dejando atrás en la escena. Por ejemplo, el personaje secundario aparece al principio de la escena y tras reproducir la animación volveremos a encontrarlo, por lo que sería interesante ir desactivándolo para simular que solo hay uno. Para ello, usaremos el script *Desactivador*, que en este caso añadiremos al mismo objeto vacío que utilizamos para iniciar la animación.

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

public class Desactivador : MonoBehaviour {

    public GameObject Objeto;

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.name == "Player")
        {
            Objeto.SetActive(false);
        }
    }
}
```

Una vez se haya reproducido la animación, volveremos a hacer uso de un objeto vacío con un componente Collider, con la opción Is Trigger activada, que colocaremos en el punto en el que finalice la animación y al que añadiremos el siguiente script *PararAnim* que hemos programado:

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

public class PararAnim : MonoBehaviour
{

    public GameObject Objeto;

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.name == "Player2")
```

```
        {
            Animator myAnimator = Objeto.GetComponent<Animator>();
            myAnimator.enabled = false;
        }
    }

    }
```

Su funcionamiento es el siguiente:

- Se hace pública las variable *Objeto* para que sea visible en el Inspector y podamos asignarles un objeto, en este caso será *Coche1*.
- Al iniciar la escena se utiliza una función que detecta cuando un objeto ha entrado en el área del Collider al que se añade el script y además si el objeto que entra en contacto se llama *Player2*, personaje asociado al *Coche1*, buscará el componente Animator del *Objeto* y lo desactivará.

Tras haber parado la animación, necesitaremos volver a activar nuestro personaje *Player* y trasladarlo hasta la zona en la que hemos parado el coche. La manera de conseguir esto lo veremos en el siguiente apartado.

Basándonos en el script utilizado para parar la animación podríamos programar otro, que también habría que añadir a un collider, utilizando las mismas funciones para iniciar la animación:

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

public class IniciarAnim : MonoBehaviour {

    public GameObject Objeto;

    void Start()
    {
        Animator myAnimator = Objeto.GetComponent<Animator>();
        myAnimator.enabled = false;
    }

    void OnTriggerEnter(Collider other)
    {
            if (other.gameObject.name == "Player")
```

```
        {
            Animator myAnimator = Objeto.GetComponent<Animator>();
            myAnimator.enabled = true;
        }
    }
}
```

La diferencia entre el script *IniciarAnim* y el script *PararAnim* es que antes de iniciar la escena desactivamos la animación y una vez se inicia se utiliza una función que detecta cuando un objeto ha entrado en el área del Collider al que se añade el script y además si el objeto que entra en contacto se llama *Player*, en este caso, buscará el componente Animator del *Objeto* y lo activará.

4.5.10.2. Trasladar objetos de forma instantánea.

Como hemos dicho anteriormente, otra forma de desplazarse por la escena sería utilizar un script para trasladar de manera instantánea nuestro personaje hasta otro punto.

En nuestro caso, queremos volver a activar nuestro personaje *Player* y trasladarlo hasta la zona en la que hemos parado el coche tras haber finalizado la animación. Para ello, hemos programado el siguiente script *Teleport* que añadiremos al mismo objeto vacío que hemos utilizado para iniciar la animación del coche*:*

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

public class Teleport : MonoBehaviour {

    public GameObject Personaje;
    public Vector3 PosDes;
    public GameObject ObjDes;

    void Start()
    {
        PosDes = ObjDes.transform.position;
    }

    void OnTriggerEnter(Collider other)
```

```
    {
        if (other.gameObject.name == "Player")
        {
            Personaje.transform.position = PosDes;
        }
    }
}
```

Su funcionamiento es el siguiente:

- Se hace públicas las variable *Personaje* (objeto a trasladar), *PosDes* (posición del objeto de destino) *y ObjDes* (objeto de destino) para que sea visible en el Inspector y podamos asignarles un objeto
- Antes de iniciar la escena se almacena en la variable *PosDes* la posición del componente Transfrom del objeto de destino.
- Al iniciar la escena se utiliza una función que detecta cuando un objeto ha entrado en el área del Collider al que se añade el script y además si el objeto que entra en contacto se llama *Player*, cambiará la posición del componente Transfrom del objeto *Personaje* por la almacenada en *PosDes.*

Para que funcione correctamente el script tendremos que crear un objeto vacío para seleccionarlo como objeto de destino, que colocaremos junto al lugar donde finaliza la animación del coche, o bien, podemos usar el mismo objeto vacío que utilizamos para detener la animación.

A continuación, tendremos que volver a activar nuestro personaje principal y desactivar el objeto *Coche1* para poder continuar moviéndonos libremente por la escena. Por tanto, hemos programado el siguiente script *BajarCoche*, que añadiremos al mismo objeto vacío que hemos utilizado para detener la animación.

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

public class BajarCoche : MonoBehaviour {

    public GameObject Personaje1;
    public GameObject Personaje2;
    public GameObject Coche;

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.name == "Player2")
```

```
        {
            Personaje1.SetActive(false);
            Personaje2.SetActive(true);
            Coche.SetActive(true);
        }
    }
}
```

El funcionamiento del script es el siguiente:

- Se hace públicas las variable *Personaje1, Personaje2* y *Coche* para que sea visible en el Inspector y podamos asignarles un objeto
- Al iniciar la escena se utiliza una función que detecta cuando un objeto ha entrado en el área del Collider al que se añade el script y además si el objeto que entra en contacto se llama *Player2*, personaje asociado *a Coche1*, desactivará el objeto *Personaje1* y activará los objetos *Personaje2* y *Coche.*

En este caso asignaremos el objeto *Coche1* a la variable *Personaje1*, el objeto *Player* a la variable *Personaje2* y un tercer coche, *Coche3*, que colocaremos junto al objeto vacío que contiene el script, a la variable *Coche*. De manera que, volveremos a poder movernos con nuestro personaje principal, desactivaremos el objeto *Coche1,* que contiene la animación, y activaremos el objeto *Coche3* para simular que hemos parado el coche.

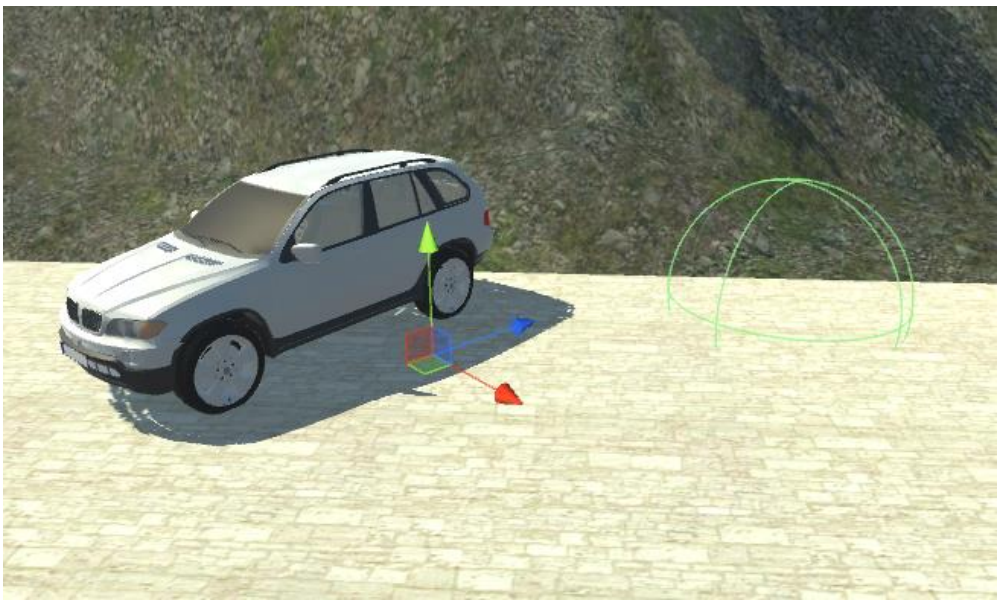

Figura 60. Coche3 junto al objeto vacío (objeto de destino) y el Collider que detiene la animación.

También podemos utilizar el script *Teleport* con un botón. Funcionará igual que en el caso anterior, pero trasladará al personaje cuando pulsemos el botón.

Para ello, utilizaremos lo explicado sobre carteles para activar objetos, ya que, usaremos un cartel con un botón, cuyo evento On Click activará un objeto vacío que tenga como componentes el script y un Collider, con la opción Is Trigger activada. Por tanto, cuando pulsemos el botón se activará el objeto y el script, pero para que funcione correctamente, antes de iniciar la escena el objeto vacío tiene que estar desactivado.

4.6. CONSTRUCCIÓN DE LA APLICACIÓN.

Cuando terminemos de editar todos los objetos de la escena y de añadir los diferentes modelos, podremos crear una aplicación para diferentes plataformas (Pc, iOs, Android, etc.). Para ello abriremos la pestaña File y seleccionaremos Build Settings, que nos abrirá una nueva ventana desde la que podremos elegir la plataforma en la que ejecutaremos la aplicación, sus parámetros e iniciar el proceso de construcción de la misma.

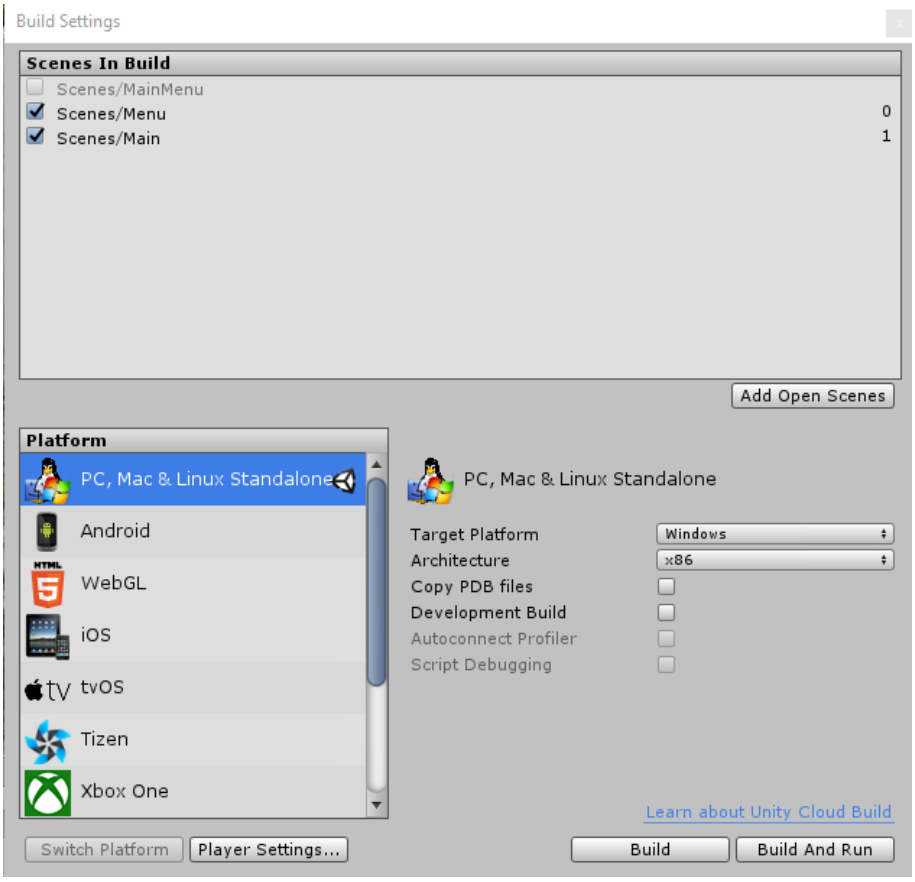

Figura 61.Ventana de Build Settings con la plataforma PC seleccionada.

Antes de iniciar la construcción de la aplicación revisaremos que están añadidas todas las escenas, en el orden correcto para evitar problemas al iniciar la aplicación o al cargar alguna escena. Una vez este todo listo pulsaremos el botón Build y elegiremos una ruta en la que guardar el archivo.

También podemos personalizar la pantalla de inicio de la aplicación. En este caso hemos escogido como nombre de la compañía UNIVERSIDAD DE CANTABRIA y como título de la aplicación PUENTE Y PRESA DE ALCÁNTARA, además se ha añadido como icono el logo de la Universidad de Cantabria.

4.6.1. Aplicación para PC.

Para crear una aplicación para PC seleccionaremos en la lista Platform la opción PC, Max & Linux tras lo que aparecerá Window x 86_64 en la pestaña Target Platform. Además, podremos seleccionar las siguientes opciones:

- *Development Build: que* permite al desarrollador trabajar sobre la aplicación mientras ésta se construye.
- Autoconnect Profiler: cuando la opción Development Build sea seleccionada, permite al profiler estar conectado a la construcción.
- Script Debugging: si la opción Development Build se selecciona, el código script puede ser depurado.

Antes de generar la aplicación no debemos olvidarnos de habilitar la opción de Realidad Virtual, para que la aplicación admita el uso de dispositivos de realidad virtual, en nuestro caso Oculus Rift. Para ello seleccionaremos el botón Player Settings en la ventana Build Settings. Al seleccionarlo, se mostrará en el inspector una serie de características entre las que se encuentra la opción Virtual Reality Supported. Una vez activada, el programa preguntará por el dispositivo de realidad virtual que será utilizado.

4.6.2. Aplicación para web.

La versión más reciente de Unity no permite generar una aplicación web que soporte el uso de dispositivos de realidad virtual, pero se puede montar una aplicación destinada a mostrar la escena a través de la pantalla del ordenador en un navegador Web. Para ello, es necesario escoger como plataforma de trabajo WebGL.

Aunque se espera que en próximas versiones de Unity se incluya la plataforma WebVR, interfaz de programación de aplicaciones (API) experimental basada en Javascript que proporciona soporte para dispositivos de realidad virtual. Las aplicaciones creadas con esta interfaz podrían utilizarse en cualquier plataforma que disponga de navegador, sin necesidad de conocer el sistema operativo del usuario, y mucho menos el dispositivo de realidad virtual que se esté utilizando.

4.7. ENFERMEDAD DEL SIMULADOR.

La enfermedad del simulador es una condición en la que una persona exhibe síntomas similares a la cinetosis, es decir, a la sensación de mareo producida por el movimiento. El uso de gafas realidad virtual puede llegar a producir este efecto por lo que es muy importante tenerlo en cuenta a la hora de diseñar aplicaciones que utilicen esta tecnología.

Estos síntomas se deben principalmente al uso videojuegos o aplicaciones similares que combinan la simulación de movimiento con la ausencia de movimiento detectada por el oído interno, ya que cuando una persona mueve la cabeza, su cerebro espera que el entorno cambie exactamente en sincronía. A esto tenemos que añadirle que pueda producirse algún retraso perceptible que llegue a crear una sensación de incomodidad en el usuario. Por tanto, para el uso de dispositivos de realidad virtual se recomienda una alta velocidad de frame por segundo, debido a que, la latencia es menor cuánto mejor sea la velocidad de renderizado por cada frame.

En general, este problema se debe tanto a la calidad de los procesadores como a la de los dispositivos de realidad virtual. Aunque está habiendo grandes avances y se esperan mejoras en el futuro.

A continuación, se indican algunos puntos a considerar para mejorar la experiencia del usuario y su seguridad:

- Utilizar una velocidad moderada al utilizar un personaje en primera persona para moverse por la escena.
- Intentar mirar al frente le mayor tiempo posible, ya que la sensación de mareo disminuye.
- No realizar giros de cabeza rápidos, ya que el problema de latencia aumenta cuánto más rápido sea el cambio de posición de la cámara.
- Utilizar una cámara en tercera persona si se realizan movimientos a alta velocidad.

- Añadir referencias visuales al entorno, como horizontes u objetos cercanos, para ayudar al usuario a orientarse.
- Proporcionar opciones para volver a centrar la vista, sobre todo para dispositivos móviles de realidad virtual.
- Intentar no utilizar cortes de escena, debido a que rompen la inmersión en la Realidad Virtual.
- Optimizar el rendimiento del renderizado. Algunos objetos, texturas o reflejos no están adaptados al uso de dispositivos de realidad virtual, por lo que conviene eliminarlos para ofrecer una buena experiencia al usuario.
- Recomendar a los usuarios tomar descansos o parar inmediatamente si se experimentan nauseas o mareos.
- Ajustar el dispositivo de realidad virtual a cada usuario y en las primeras ocasiones utilizarlo sentado.

5. CONCLUSIONES

El objeto de este trabajo ha sido estudiar formas de crear una aplicación en la que virtualizar una obra o infraestructura con la que interactuar gracias a la Realidad Virtual, analizando las posibilidades que puede ofrecer a la Ingeniería Civil.

La Realidad Virtual combinada con el uso de modelos tridimensionales proporciona importantes ventajas a la hora de acercarnos a una infraestructura entre las que podemos destacar:

- La inmersión en el modelo, permitiendo recorrer la infraestructura real.
- Facilitar el diseño y toma de decisiones en la fase de proyecto, construcción, mantenimiento y conservación de la infraestructura.
- Estudiar el comportamiento estructural de la infraestructura.
- Reducir costes en comercialización y marketing, ya que el cliente podrá “pasear” por el modelo antes de ser construido.
- Ofrecer un paseo interactivo por la infraestructura a modo de herramienta turística o didáctica.

También destacaremos la utilidad del motor de videojuegos Unity. Gracias al cual hemos podido importar fácilmente modelos tridimensionales a un entorno virtual y crear formas de interactuar con dicho entorno a través de la Realidad Virtual, aunque sí es cierto que para crear estos mecanismos de interacción necesitamos un conocimiento más profundo tanto de Unity como de programación.

Finalmente, añadir que los dispositivos de realidad virtual están teniendo una gran acogida no solo en el mundo de los videojuegos, de hecho, algunas empresas relacionadas con la ingeniería y la construcción están comenzando a utilizarlos. Por lo que deberemos acostumbrarnos a que cada día estarán más presentes en el ámbito de la ingeniería civil.

6. BIBLIOGRAFÍA

BIBLIOGRAFÍA

Bradley Austin Davis, Karen Bryla, Phillips Alexander Benton. 2015. *Oculus Rift in Action.* Shelter island : Manning Publications Co.

Biblioteca de código abierto SketchUp 3D Warehouse

https://3dwarehouse.sketchup.com/

Centro Nacional de Información Geográfica - Ministerio de Fomento. Centro de descargas.

http://centrodedescargas.cnig.es/CentroDescargas/

Cuartas Hernández, Miguel. 2014-2015. Unity 3D. *Crear un terreno real en Unity 3D.*

EspacioBIM. Centro de Formación Autodesk especializado en aprendizaje on-line en entorno BIM.

https://www.espaciobim.com/

Linowes, Jonathan. 2015. *Unity Virtual Reality Projects.* Birmingham : Packt Publishing Ldt., 2015.

SOUNDBIBLE Biblioteca gratuita de clips de audio.

http://soundbible.com/

TF3DM Biblioteca gratuita de modelos 3D

http://tf3dm.com/

Unity Technologies–Asset Store.

https://www.assetstore.unity3d.com/

Unity Technologies–Documentation. Unity Manual

https://docs.unity3d.com/es/current/Manual/

Unity Technologies–Tutoriales.

https://unity3d.com/es/learn/tutorials/

Visual Technology Lab.

https://www.vt-lab.com/