UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

DESARROLLO DE UNA APLICACIÓN DE REALIDAD VIRTUAL

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Pablo Marcos Miñarro

Tutor: Eduardo Vendrell Vidal

Curso: 2015-2016

RESUMEN

Este proyecto trata sobre la creación de un entorno en 3D del campus de Vera de la Universidad Politécnica de Valencia en el que se pueda experimentar la Realidad Virtual mediante Oculus Rift, así como interactuar con el mismo mediante dispositivos de control utilizando el dispositivo Leap Motion, que permite una lectura de la posición en el espacio de las manos del usuario, proporcionando control sobre el entorno a través de gestos para desplazarse sobre él y realizar otras acciones.

También se permite la utilización de un joystick (mando de Playstation 3) para realizar dichas acciones en caso de no disponer de un Leap Motion.

Consta de dos partes:

Por un lado, se ha construido todo el campus pieza a pieza, gracias a los modelos en 3D proporcionados por la Universidad Politécnica de Valencia, pedido expresamente para desarrollar esta aplicación. Una vez creado el entorno, se ha aplicado un paquete de Unity 3D para Oculus Rift, cuyo funcionamiento se explicará más adelante.

Dicho entorno puede ser recorrido con una serie de gestos tales como desplazar las manos hacia delante para avanzar, hacia un lado para girarse, etc.

La segunda parte, consiste en un menú interactivo que permite la visualización de archivos tales como imágenes, videos, etc., el cual se puede manejar con las manos de una forma muy intuitiva, con una interfaz ágil y de fácil aprendizaje.

TABLA DE CONTENIDOS

1. INTRODUCCIÓN

La idea para este proyecto surgió al ver la aplicación del plano en 3D de la UPV, disponible en http://www.upv.es/plano/plano-3d-es.html.

Dado el gran potencial que tiene dicha aplicación, surgió la oportunidad de añadirle funcionalidades nuevas para darle una imagen innovadora, haciendo que resulte más atractiva y práctica. La herramienta elegida para dicha tarea fue el motor de videojuegos Unity 3D.

Las razones de utilizar este motor 3D fueron, principalmente, su gran comunidad, la cual permite obtener mucha información a la hora de abordar ciertos problemas, su compatibilidad con múltiples plataformas , o el hecho de que dispone de una versión gratis, sin necesidad de obtener una licencia, con lo que se ahorra en costes. Además, es un motor con el que estoy familiarizado desde hace tiempo y esto me ha facilitado la labor en el desarrollo de la aplicación.

Oculus Rift, sin embargo, es un dispositivo cuyo alto precio es un obstáculo para muchos usuarios. Existe una alternativa más económica, Cardboard, pero no se adapta al desarrollo de la aplicación, ya que este periférico utiliza el sistema operativo para móviles Android, con lo que el rendimiento sería menor, y más costoso de desarrollar en función de tiempo.

El siguiente problema con el que lidiar fue la interacción con el entorno. Se optaron por dos soluciones: Leap Motion y Joystick. Había una tercera opción, la cual consistía en una plataforma de movimiento, pero aún no hay versión comercial, así que se descartó al final.

La Realidad Virtual es un campo en expansión actualmente, donde aún puede innovarse, y este es el principal motivo por el que surge este proyecto. Así mismo, muchas de la ideas tuvieron que ser descartadas dada la enorme complejidad que suponían y el tiempo necesario para llevarlas a cabo. Estas ideas suponen un trabajo futuro de investigación con el ánimo de conseguir una aplicación aún más funcional y que se explicarán en los últimos apartados de este documento.

El campo que más aprovecha esta tecnología es el de contenido multimedia, cuya finalidad principal es la de ofrecer entretenimiento. No obstante, existen muchas otras áreas de conocimiento que también se benefician de las posibilidades que ofrecen estos dispositivos: arqueología, medicina, arquitectura, o incluso el ejército.

La sociedad cada vez tiene un mayor acceso a estos avances, ya que existen propuestas enfocadas directamente al usuario medio. Así, la idea es que cualquier persona pueda disfrutar de estos contenidos desde sus casas. Una de las opciones más

comunes y económicas permite visualizar contenido de Realidad Virtual a través de un Smartphone y unas gafas de cartón.

El sistema es simple: basta con introducir el teléfono en la estructura y se coloca a modo de gafas. La técnica para simular la visión tridimensional se consigue dividiendo la pantalla en dos para visualizar dos imágenes. Oculus Rift (el dispositivo elegido para este proyecto, actúa de esta manera, ya que en el interior del mismo se utiliza una pantalla de móvil). A pesar de ser el mismo concepto, esta opción no es viable para cualquier persona, dado su elevado precio de mercado.

En este documento entraremos en profundidad en los siguientes temas:

- Historia y evolución de la Realidad Virtual.
- Mercado actual y diferentes propuestas.
- Funcionamiento de los dispositivos escogidos para el desarrollo.
- Motor y herramientas auxiliares.
- Diseño y estructura.
- Fase de desarrollo.
- Simulación de ejecución.
- Conclusiones, trayectoria recorrida y propuestas de mejora.

2. OBJETIVOS

El objetivo principal de este proyecto es desarrollar un producto que aporte una experiencia totalmente nueva e innovadora al usuario, mediante la inmersión total en un entorno virtual. Dada esta premisa, se aborda el siguiente problema: ¿qué acciones puede llevar a cabo el usuario? Lo principal es que pueda desplazarse (avanzar, retroceder, girarse, saltar, etc.), también surgió la idea de que se pudiera interactuar con objetos, lo que dio lugar a la segunda parte del proyecto, que consiste en implementar una serie de "terminales" a los que el usuario puede acceder con la palma de la mano y que se encuentran ubicados en cada edificio del campus.

Estos terminales son independientes unos de otros y que, dependiendo de cuál se trate, albergará información relevante al edificio en cuestión al que pertenezca. Por ejemplo, si se accede al terminal situado en la Escuela Técnica Superior de Informática, aparecerá un menú interactivo con información exclusiva de dicha facultad, como videos promocionales de cada titulación, calendarios de exámenes, profesorado,....

Esta nueva funcionalidad permitiría que los usuarios dispongan de una gran cantidad de información acerca de toda la Universidad dentro de la misma aplicación. También sería posible utilizarse para anunciar eventos, menús de cafeterías al día, etc.

En definitiva, crear una aplicación que resulte útil y que atraiga a futuros usuarios y sea capaz de innovar progresivamente a raíz de nuevas ideas, dando total libertad a otros desarrolladores que quieran continuar con esta labor, siempre adoptando las bases de la innovación, con el fin de otorgar un servicio al que cualquiera sea capaz de acceder y beneficiarse de su contenido.

3. REALIDAD VIRTUAL Y DISPOSITIVOS INTERACTIVOS

3.1 Historia:

La realidad virtual es un concepto que se vincula directamente a la informática hoy en día, pero la realidad es que es incluso anterior a ella. Para entender el concepto, hay que retroceder a la época del estereoscopio (Charlse Wheatstone, 1844), el cual consistía en unas gafas a través de las cuales se ven dos fotografías iguales, solo que se diferencian en el punto de toma de la imagen. Esto hace que el cerebro cree un efecto tridimensional al mezclarlas.

Imagen 1, Comparativa estereoscopio – Oculus Rift, burgospedia1.wordpress.com, www1.oculus.com

Esta es la base de las gafas de realidad virtual de hoy en día. En términos de informática, desde una aplicación se envía la misma imagen desde dos puntos de vista diferentes a Oculus Rift, sobre una pantalla partida, para que cada ojo reciba una información diferente.

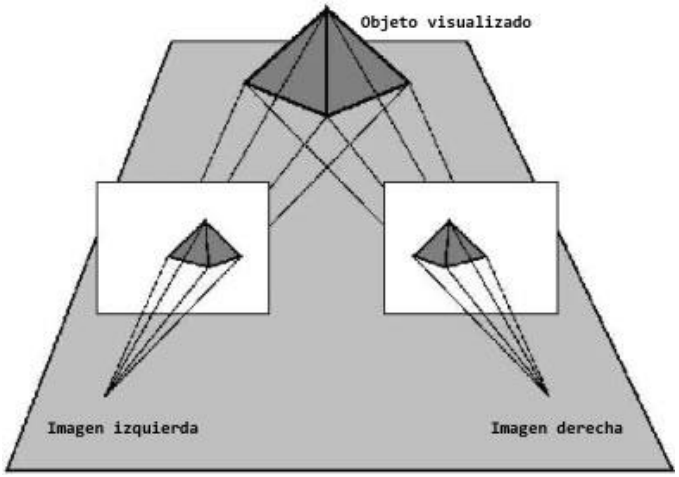


Imagen 2, Proyección estereoscópica, www.teknoplof.com

En la historia, este concepto se ha utilizado para numerosos ámbitos, tales como el cine, eliminando el color rojo para un ojo, y el verde y el azul para el otro (Louis Ducos du Hauron, anaglifo, 1891), fines militares (simulador de aviación con realidad virtual, 1971), o incluso para programas espaciales con la finalidad de entrenar astronautas.

El principal problema es el concepto de "inmersión", esto es, conseguir que el usuario se sienta partícipe del entorno y no se considere un elemento extraño. Para conseguir esto, no sólo se ha de tener en cuenta la vista, sino que se debe profundizar también en el resto de los sentidos. Una solución a la interacción con el entorno fue la aparición de uno de los primeros dispositivos hápticos (Tom Defanti y Daniel Sandin, 1977). Se trata de un guante que tenía en cada dedo un tubo flexible de fibra óptica con un emisor de luz en un extremo y un receptor en el otro. En función de la cantidad de luz que llegaba al receptor se podía calcular la flexión de los dedos.

Imagen 3, Sayre Glove, vw.fed.wiki.org

3.2 Tipos de Realidad Virtual:

Gafas LCD (Liquid Crystal Display):

Son unas lentes de visualización sobre cristal líquido. Utilizan un fotosensor que alterna el paso de la luz por cada cristal a una frecuencia lo suficientemente alta para que el usuario no se percate de que la imagen no se proyecta simultáneamente en ambos. Dicha frecuencia es de 60 Hz. Se proyecta primero por ejemplo en el cristal izquierdo la imagen, y acto seguido, se proyecta la misma imagen en el derecho pero desplazada hacia la derecha.

El problema de las LCD, es que no ofrecen una inmersión total, ya que el usuario es capaz de percibir el mundo real y debe concentrar la vista en el centro del cristal.

Imagen 4, Gafas LCD, www.mundomanuales.com

Actualmente, se está llevando a cabo un prototipo de lentes de contacto que utilizan tecnología LCD, con lo que no se necesita montura, y se elimina el principal problema de usar unas gafas de estas características, puesto que las lentes abarcan la totalidad del campo de visión de los ojos.

Un ejemplo de dispositivo que utiliza esta tecnología comercializado recientemente (Junio de 2014), sería el llamado "Google Glass". Desafortunadamente, no ha tenido el éxito que se esperaba de él, y cada vez se desarrollan menos aplicaciones enfocadas a esta tecnología. Se espera una segunda versión por parte de sus creadores que intensifique las cualidades de la primera.

Su alto coste, de alrededor de 1000€, supuso una barrera para que muchos desarrolladores se lanzaran a experimentar con ellas. Además, sufre problemas de sobrecalentamiento, lo cual para el usuario resulta bastante incómodo, ya que como consecuencia de un uso prolongado, su temperatura es tal que comienza a afectar en la cabeza del que las lleva puestas.

Como componente adicional, cuenta con una cámara que es capaz de recoger lo que el usuario tiene delante, permitiendo así la función de actuar como unas gafas de realidad aumentada, que muestran el entorno modificado, generalmente para aportar información.

Surgieron muchas quejas en cuanto a seguridad, puesto que se puso de moda entre los ciclistas, que las utilizaban para visualizar mapas de rutas mientras circulaban, provocando distracciones y accidentes.

BOOM (Binocular Omni-Orientation Monitor):

Uno de los primeros visualizadores digitales de realidad virtual y que fue sustituído por los cascos HMD.

Se compone de un brazo mecánico, unos sensores de posicionamiento y un monitor. El usuario puede girar el brazo, que es el que soporta su elevado peso, para cambiar su posición y orientación en el espacio, pero tiene los movimientos son muy limitados, y esto entorpece la experiencia.

Se pusieron de moda en los salones recreativos, dando lugar a los primeros videojuegos de realidad virtual, durante los años 90, y parte de la primera década del siglo XXI.

Imagen 5, BOOM, itdi12ct3.weebly.com

CAVE (Cave Automatic Virtual Environment):

Es sin duda el dispositivo más costoso de todos en términos monetarios, ya que se trata de una habitación en la cual se utilizan proyectores y espejos para reflejar imágenes en las paredes, techo y suelo (según la finalidad de la aplicación, se pueden utilizar diferente número de pantallas, siendo a veces algunas de las caras de un cubo al descubierto), alrededor del usuario. Éste ha de situarse en el centro y usar unas gafas 3D para ver el efecto tridimensional. Es necesario también utilizar una serie de sensores en la cabeza del mismo para calcular la orientación.

Los proyectores se colocan en las esquinas de la habitación y emiten imágenes sobre las pantallas situadas formando un cubo, al igual que los altavoces que emiten sonido para enfatizar la sensación de “realidad”. Los objetos aparecen en la escena “flotando” en el espacio y el usuario es capaz de atravesarlos e interactuar con ellos.

Las gafas están sincronizadas con un monitor que muestra las imágenes en consecuencia de la información recibida por los sensores de rastreo de posición acoplados en ellas.

El espacio se ve afectado por un único usuario, aunque se permite la presencia de más personas en el interior, con el único rol de observadores. También es posible interactuar con el entorno a través de otros periféricos, como guantes hápticos.

Su principal problema es el espacio, puesto que si el que se encuentra en el interior se desplaza, podría colisionar con alguna de las paredes y causar un accidente.

El origen de CAVE, en 1992, tenía la finalidad de ser una herramienta capaz de visualizar datos. El primer prototipo, constaba de tres pantallas en las paredes y dos proyectores, situados uno detrás, en la pared destapada, y otro en el techo, apuntando al suelo. Se utilizaban gafas de tecnología LCD, que proyectaban una imagen diferente a cada ojo, a una frecuencia de 96Hz, siendo de 48Hz para cada ojo.

Algunas ventajas y desventajas de este sistema son:

Ventajas	Desventajas
• Alta resolución y nitidez en la imagen.	• Alta complejidad en el desarrollo de aplicaciones.
• Realidad y virtualización son compatibles y no se anulan como ocurría con las LCD.	• Personal altamente cualificado para su mantenimiento.
• El usuario sólo necesita unas gafas, sin necesidad de cascos u otros componentes, por lo que el usuario no se siente pesado o molesto.	• Difícil aprendizaje en cuanto a movimiento e interactuación con el entorno virtual.

Su uso, principalmente, está enfocado a la simulación, sin que existan riesgos de ningún tipo, ya sean económicos o humanos. También para tareas de entrenamiento, como formación de pilotos aéreos, medicina, etc.

La Universidad Politécnica de Valencia cuenta con este sistema de visualización de realidad virtual desde hace unos años y se compone de:

- **La sala de máquinas:**
  En la que se encuentra el generador de imagen (SGI Prism), el procesador del sistema de control (Crestron), la matriz de video más video splitters (Extron) y el SAI (PowerWare).
- **La sala de control:**
  En la que se encuentran los monitores de control CRT, el touch-panel del Crestron (GUI del sistema de control de los proyectores) y el PC donde está instalado el software de tracking. La instalación funciona a 112 Hz, 56 Hz para cada ojo, con una resolución de 1024x1024.
- **La sala de proyección:**
  Donde encontramos el sistema de proyección, en la que destacan los proyectores Barco 909, las pantallas Stewart y los espejos. También se ubican en esta sala otros elementos auxiliares como las matrices de láser para el alineado inicial, los emisores de la señal de estéreo activo, los emisores/cámaras del sistema de tracking, o las cámaras del sistema de auto-alineado.

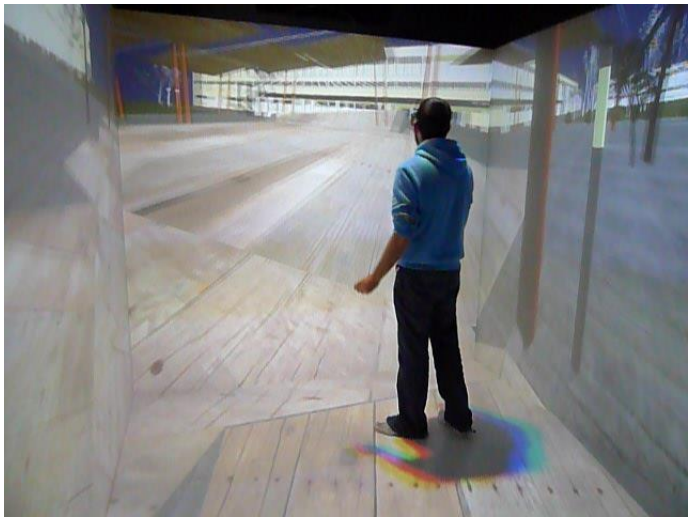


Imagen 6, CAVE de la UPV, www.victorbonilla.es

Plataformas de movimiento:

Son unas estructuras en las que el usuario se introduce y que se mueve mediante un sistema hidráulico en consecuencia de las imágenes que se van proyectando. Su finalidad es la de que aquellos que ven las imágenes proyectadas en la pantalla sientan que el mundo que ven les afecta a otros sentidos a parte de la vista.

Algunas están equipadas con otros elementos que aumentan la realidad de la experiencia, como el uso del agua para escenificar escenas con dicho elemento, o incluso olores. Todo esto, acompañado de un buen sistema de sonido, forma parte del truco. Normalmente están enfocadas para el uso en un grupo de personas, en lugar de una persona única. Esta una gran ventaja respecto a los demás sistemas de realidad virtual.

Aun así, no son muy utilizadas en la actualidad por su poco atractivo y por no conseguir una experiencia creíble, además de por su alto coste, siendo necesarias unas grandes instalaciones para su mantenimiento.

Un ejemplo puede ser una atracción de feria en la que se proyecta la imagen de una montaña rusa y unos asientos hidráulicos que realizan movimientos siguiendo las vías. A ésta práctica se le ha llegado a denominar como “4D”.

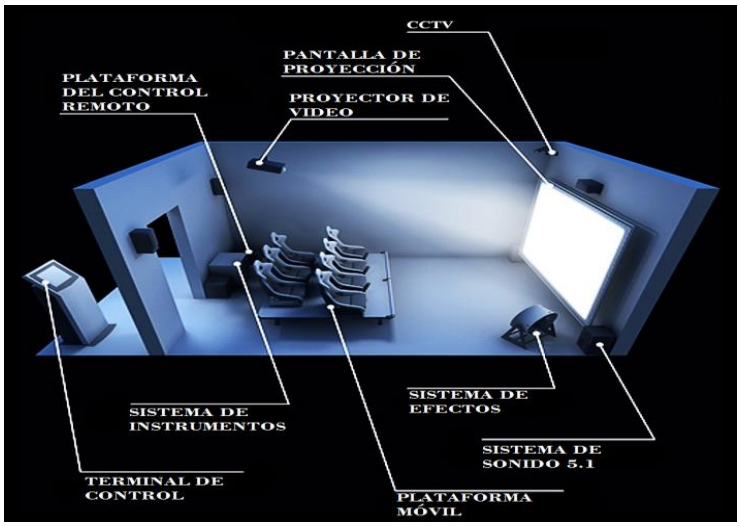


Imagen 7, Diagrama de un cine 4D, es.wikipedia.org

Dispositivos hápticos:

Son aquellos que permiten estimular el sentido del tacto. Se utilizan sobre todo para capturar movimientos del usuario para influir en el entorno, pero también es posible generar una respuesta al usuario a través de ellos. Hay guantes que ejercen presión en las manos para simular fuerzas, resistencias, o incluso el peso de un objeto al cogerlo.

Aunque ya se utilizaban hace unos años, el gran progreso realizado con este tipo de dispositivos, hace que sean de gran atractivo para los desarrolladores de aplicaciones, dado su gran potencial y resultados favorables. Funcionan muy bien con cualquier tipo de elemento visualizador de realidad virtual y aumenta exponencialmente la sensación de inmersión.

Además de guantes, existen otros tipos de dispositivos hápticos, tales como los denominados “rastreadores”, que son capaces de detectar las manos del usuario para reproducir su exacta posición en el espacio y recrear una copia virtual de las mismas. También existen otros dispositivos que son capaces de interactuar con otras partes del cuerpo, pero son menos habituales, siendo presentes en otros campos tales como la medicina

Así, las cualidades que presentan permiten a una persona tocar, sentir, manipular, crear y cambiar objetos tridimensionales en un espacio virtual.

Existen varias categorías dentro de la háptica:

- Propioceptivo: Relacionado con la información sensorial acerca del estado corporal.
- Cutánea: Perteneciente a la piel en sí misma o como órgano sensorial (presión, temperatura y dolor).
- Kinésica: Sensación de movimiento originadas en músculos, tendones y uniones en tensión-distensión.

- Vestibular: Relativo a la posición de la cabeza y su aceleración. Afecta al sentido del equilibrio.
- Realimentación táctil: Aporta información de la presión de los objetos, así como de su contorno y características de su superficie.
- Realimentación de fuerza: Referido a fuerzas y obstáculos que afectan al cuerpo.

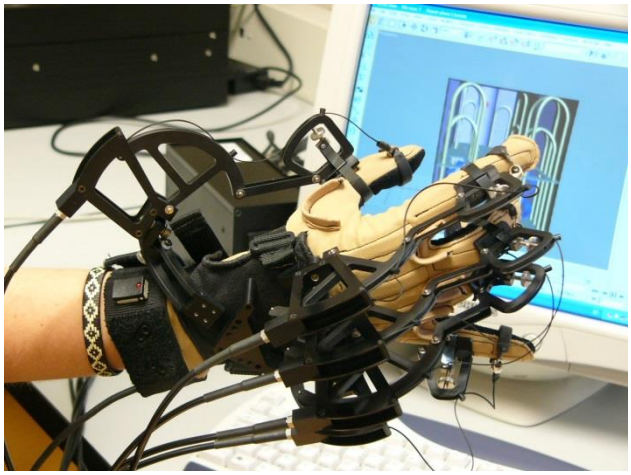


Imagen 8, Dispositivo Háptico, dac.escet.urjc.es

Casco HMD (Head-Mounted Display):

Este es el dispositivo más conocido de todos, porque es el más actual y con mayor número de prototipos en el mercado. Tiene el equilibrio perfecto entre todos los tipos de visores de realidad virtual y mejor éxito comercial.

Existen tres tipos de tecnología para la imagen en HMD:

- Pantallas LCD: Imagen más clara, pero con baja resolución y contraste. No se identifica bien la posición exacta de los objetos. Es la opción más barata.
- Fibra óptica: El fósforo de la pantalla se ilumina a través de la fibra, proporcionando mejor resolución y contraste, pero el coste es mucho mayor.
- CRT: La pantalla es iluminada mediante haces de electrones y un tubo de rayos catódicos. Como contrapartida, este tipo de HMD se calienta demasiado y resulta incómodo.

A todo lo anterior hay que añadir que es necesario utilizar un cable en la mayoría de los prototipos (aunque hay alguno inalámbrico), lo que no permite la movilidad deseada.

Imagen 9, HMD, escience.anu.edu.au

Hay que mencionar que para utilizar un HMD, se necesita un equipo muy potente, en especial una tarjeta gráfica capaz de enviar la imagen a buena velocidad para evitar retardos o paradas. Es bastante habitual que el equipo se sobrecaliente por este motivo.

Una ventaja que ofrecen es la posibilidad de emitir simultáneamente la imagen al casco y a una o varias pantallas, lo que habilita que varias personas sean capaces de ver lo mismo que el usuario que lo utiliza.

Otro aspecto a tener en cuenta es el campo de visión o "field of view". Hay cascos con mayor "FOV" que otros. Esto afecta a la amplitud del campo visual del usuario que es ocupada por la imagen virtual. Cuanto mayor sea, mayor será la sensación de inmersión. Suele estar entre los 100º.

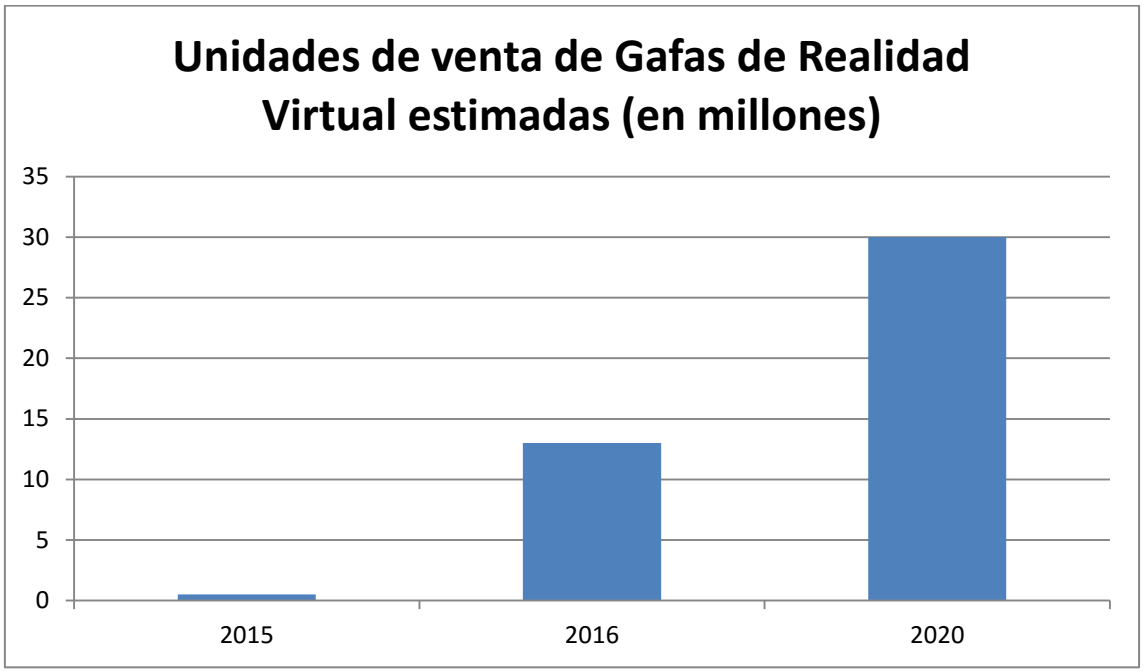
3.3 Mercado y Competencia:

2016 se considera como el año de la realidad virtual, ya que cada vez hay más propuestas de grandes compañías y su precio es más competitivo, haciendo que más personas puedan acceder a esta nueva tecnología. Su integración en el mundo de los videojuegos atrae a muchos aficionados que desean experimentarlo, aumentando potencialmente el crecimiento del sector.

Empresas como HTC, Samsung, Sony o Microsoft han entrado muy fuerte en el mercado de la RV. Así, cada marca ofrece diferentes prestaciones en relación con su precio, que puede variar desde los 400€ (Playstation VR) hasta los 1000€ (Oculus Rift). No obstante, existen alternativas para aquellos con menos recursos, ya que gracias a Google y sus gafas de cartón, se puede utilizar un Smartphone para el mismo fin. Obviamente, el resultado no es el mismo que con un buen dispositivo dedicado a ello.

Según datos de la consultora Gartner, en 2015 se vendieron alrededor de 140.000 dispositivos de RV, mientras que se espera que este año lleguen a ser 13 millones de unidades vendidas.

Otras consultoras, estiman que en el año 2020, serán 30 millones los dispositivos vendidos, al igual que también se espera un decremento en el precio de los mismos con el paso del tiempo, hasta estabilizarse entre los 100€ para los más baratos, y en torno a los 400€ para aquellos con mayores prestaciones.



Las propuestas de los grandes competidores en el mercado de la realidad virtual son:

Google Cardboard:

El modelo base está hecho de cartón, pero existen compuestos de plástico y otros materiales que resultan más cómodos para el usuario. Su precio, que ronda los euros, es su principal ventaja. Además, al utilizar un Smartphone cualquiera, existe una amplia variedad de aplicaciones compatibles, gracias al sistema operativo Android y a la plataforma Google Play, desde donde se pueden descargar.

Estas gafas son ajustables, de modo que casi cualquier modelo de Smartphone es compatible con ellas.

Por el contrario, la calidad de la experiencia se ve muy afectada por el rendimiento y la latencia. Se consideran una herramienta para ver contenidos de realidad virtual pero no para sumergirse en ella. Tal es así, que al cabo de unos minutos pueden sufrirse mareos o náuseas.

Imagen 10, Google Cardboard, http://i.blogs.es/22160a/cardboard/1366_2000.jpg

Samsung Gear VR:

Siguiendo la filosofía de las gafas de Google, surgió un proyecto con mejor proyección por parte de Samsung, que elaboró un prototipo de Cardboard evolucionado, con mejor estructura y prestaciones.

Son más cómodas de utilizar e incluye mejoras como el campo de visión más amplio (96º frente a los 90º de Cardboard), baja persistencia y latencia, sensores para mejorar el seguimiento de la cabeza, y la inclusión de un sistema de interacción por botones, situados a la derecha de las gafas, para desplazarse por menús e interfaces.

Sus principal baza es su precio, alrededor de 100€, si no tenemos en cuenta el inconveniente de que los únicos terminales compatibles son los modelos más caros de Samsung (S6, S6 Edge, S7, S7 Edge y Note5), cuyos precios oscilan entre los 400 y los 800 euros. De esta forma, únicamente resulta una opción barata si ya se dispone de uno de esos terminales.

Imagen 11, Samsung Gear VR, http://i.blogs.es/a7db00/oculus/1366_2000.jpg

PlayStation VR:

Sony tiene previsto lanzar su producto en octubre de este año, a un precio de 399 euros. Únicamente serán compatibles con su consola PlayStation 4, por que se ha de desembolsar otros 350€ para su adquisición, además de un complemento extra que es la cámara, que complementa las funciones de las gafas, y que cuesta alrededor de 60 euros más, con lo que experimentar la realidad virtual mediante la opción de Sony cuesta en torno a 800€.

Al margen de ser cómodas, no son inalámbricas como las dos opciones anteriores, aunque utilizarán un cable muy largo para no entorpecer demasiado al usuario.

Sus características son:

- Pantalla de 5,7 pulgadas.
- Resolución de 1920x1080 píxeles.
- 120Hz de frecuencia de refresco (más alta que la de sus competidores).
- 90º de campo de visión (inferior a las otras propuestas).
- No necesitan configuración, ya que son exclusivas para la consola de Sony, al contrario que las opciones que funcionan en PC.

Imagen 12, Playstation VR, http://i.blogs.es/f2696d/psvr/1366_2000.jpg

3.4 Oculus Rift y Leap Motion:

Oculus Rift:

Para el desarrollo de esta aplicación, en cuanto al sentido de la vista, se optó por utilizar un casco HMD, ya que sus cualidades permiten conseguir la experiencia marcada como objetivo. En concreto, Oculus Rift, dado a que existen muchos recursos del mismo para Unity 3D, lo que facilita en gran medida el trabajo.

Oculus Rift es un dispositivo en fase de desarrollo por la compañía Oculus VR. Actualmente, cuenta con dos versiones no destinadas para consumidores, sino para desarrolladores. Es por esto que ambos modelos llevan la etiqueta “Development Kit”, versión 1 y 2. La diferencia entre ambas versiones sin duda es la inclusión de una cámara que detecta la posición y el movimiento en la versión 2, además de mejorar la resolución de la anterior. Se espera que a lo largo de 2016 se lance una nueva versión enfocada al consumo. Para este proyecto se emplea la versión “Development Kit 2”.

Este dispositivo se compone principalmente de:

- Una pantalla y un par de lentes que permiten una resolución en cada ojo de 960 x 1080 pixeles, con un refresco de 75Hz.
- 3 sensores,
    - Un acelerómetro. Mide velocidades, aceleraciones, etc.
    - Un giroscopio. Calcula rotaciones en el espacio y cambios de orientación.

  - Un magnetómetro. Sirve para cuantificar señales magnéticas en fuerzas y direcciones.
- Una cámara que detecta la posición en el espacio.

El principal problema es su alto coste, lo que aleja bastante al público a la hora de poder utilizar la aplicación. No está por tanto dirigida a público en general que pueda disponer de este HMD en su casa, sino más bien a lugares públicos donde exponerse.

Si bien es cierto que actualmente su precio es desorbitado, se estima que su versión comercial será más asequible, dada la fuerte competencia del sector y que está en auge.

Algunas mejoras previstas son:

- Aumento de la resolución de la pantalla: Pantalla 4K, 2K en cada ojo, con lo que supera a cualquiera de sus competidores.
- Frecuencia de refresco de 90Hz.
- Inclusión de botones para interactuar con el entorno.
- Estabilizador de imagen (la versión actual provoca mareos si se prolonga la exposición a la pantalla).
- Audio 3D integrado. Ningunas gafas actuales poseen esta faceta.

Leap Motion:

Se trata de un rastreador o "tracker" el cual se conecta a través de un puerto USB y que permite conocer la posición de la estructura de las manos en el espacio.

Funciona a través de unos sensores infrarrojos, compuestos por 3 LEDs cuya longitud de onda es de 850nm, que colisionan con distintos puntos de la mano, se crea una malla de puntos, y se envían las coordenadas de cada uno de ellos en vectores X, Y, Z, para construir un modelo 3D en la aplicación en base a ellos. La estructura de la mano captura la palma, los dedos y las falanges, permitiendo capturar movimientos tan complejos como cerrar la mano, abrirla, mover un dedo,….

Imagen 13, Leap Motion Controller

Cuando las manos son iluminadas por los sensores, se produce una reflexión de luz que recogen las 2 cámaras integradas y se almacena una imagen digitalizada. Dicha

imagen se examina pixel a pixel para comprobar la luminosidad y se cuantifica en valores de 8 bits (un valor entre 0 y 255), para obtener una imagen RAW en escala de grises. Es decir, si un pixel tiene valor 0, quiere decir que el color en escala de grises es el negro, y si es 255, es blanco. Los pixeles que se acercan a negro no se tratan, puesto que la cámara no ha detectado nada en esos puntos.

Imagen 14, Captura Leap Motion

Gracias al uso de dos cámaras, podemos saber la distancia a la que se encuentran las manos del dispositivo, es decir, profundidad. Es el mismo concepto visto antes cuando hablábamos del estereoscopio para la percepción de las 3D.

Tiene un límite de distancia, luego no se debe acercar ni alejar demasiado las manos del sensor o no se captará toda la información necesaria. El rango de detección efectivo se encuentra entre los 25 y los 600 milímetros sobre el controlador.

Todas estas características descritas hacen que Leap Motion encaje con las pretensiones deseadas para ser el enlace entre la aplicación y el usuario. Además, es la opción más viable dado que es una de las propuestas más económicas de todas las disponibles.

Integrando ambos dispositivos:

Ambos tienen sus limitaciones por separado, pero al ser totalmente compatibles, hacen que la experiencia resulte verdaderamente creíble, ya que el usuario percibe el entorno con la vista, y es capaz de interactuar con él usando sus propias manos. Aprovechando sus cualidades individuales, se consigue el dispositivo perfecto.

Uno de los problemas que tiene Leap Motion es su movilidad. El periférico debe estar sobre una superficie plana para poder funcionar, con lo que si el usuario se desplaza del lugar, no podrá seguir utilizando el controlador. En este aspecto es donde realmente se beneficia de la versatilidad de Oculus. La idea es simple, se coloca el rastreador en la parte frontal del casco mediante cinta de doble cara, con lo que los dos quedan totalmente unidos, y el usuario siempre va a poder utilizar sus manos sin

preocuparse de la localización del primero, aunque sí debe tener en cuenta que se encuentra en su cabeza.

De igual modo, solamente con el casco, el usuario lo único que puede hacer es ver. No puede realizar ninguna acción más que girar en el sitio sin moverse.

Imagen 15, Oculus + Leap

Son estas razones las que hacen posible el objetivo principal de la aplicación, que es que el usuario sea capaz de explorar algo que no es “real”, sino virtual.

4. MOTORES GRÁFICOS

Aunque surgieron con motivo de facilitar la labor en el desarrollo de software enfocado a los videojuegos, actualmente se utilizan en todo tipo de proyectos que requieran un cierto nivel de programación gráfica, ya que disponen de un potente diseñador de escenas tanto en 2D como en 3D.

Como funcionalidades que proveen, entre otras muchas, se pueden incluir:

- Motor de renderizado para gráficos.
- Motor de físicas.
- Detección de colisiones.
- Gestión de scripts.
- Animaciones.
- Inteligencia Artificial.

Existen motores “open-source” y comerciales o de pago, desarrollados por empresas para luego ser vendidos a otros estudios para que desarrollen sus videojuegos. Estos últimos suelen ser más potentes y prácticos, mientras que los primeros suelen servir de ayuda para aplicaciones implementadas en lenguajes de programación tales como C++ o JAVA.

Términos asociados a motores gráficos:

Assets:

Los assets pueden ser traducidos como elementos que serán introducidos en el videojuego. Estos elementos incluyen Modelos 3D, personajes, texturas, materiales, animaciones, scripts, sonidos, y algunos elementos específicos de cada motor. Cada motor trabaja de una manera distinta a otros lo cual puede aceptar "Assets" que otros motores no pueden manejar, sin embargo los ejemplos mencionados antes, son elementos que todos los motores de hoy en día usan.

Application Programming Interface (Interfaz de Programación de Aplicaciones):

Es un sistema de rutinas, de protocolos y de herramientas para desarrollar programas de aplicación. Un buen API hace más fácil desarrollar un programa proporcionando todos los bloques del desarrollo del programa. El programador pone los bloques juntos. Los más importantes son el DirectX (de Microsoft) y el OpenGL (que trabaja con la mayoría de los sistemas operativos).

Render (Renderización):

El render o renderización, es el proceso de la computadora en mostrar en pantalla el aspecto visual de nuestro juego. Se encarga de mostrar al jugador todo el poder gráfico que el desarrollador haya configurado en el motor (el terreno o BSP, modelos, animaciones, texturas y materiales). Contribuye todo el aspecto visual del juego.

Objetos 3D:

Existen tres categorías en función a su nivel de detalle:

Low-Poly (Polígonos bajos): Son modelos que su composición de polígonos es baja, lo cual es probable que el modelo tenga un muy mal detalle y no se obtenga un resultado favorable. Estos modelos se usan para optimizar el rendimiento del videojuego y es efectivo en modelos que no requieren mucho detalle, como objetos pequeños y poco visibles.

Mid-Poly (Polígonos medios): Son modelos que su composición de polígonos es media y logran dar mejor detalle que los "Low-Poly" aunque su velocidad de procesamiento es más lenta. Estos modelos son más usados para modelos que requieren un poco más de detalle, como armas o casas.

High Poly (Polígonos altos): Son modelos que su composición de polígonos es alta y llegan a dar un detalle magnífico pero su procesamiento es más complejo y tiende a ralentizar el ordenador, dependiendo de la potencia que el Hardware de la computadora o consola cuente. Estos modelos son usados para modelos que precisan de un buen grado de detalle. Se utilizan sobre todo para el diseño de personajes.

Higher-order surfaces (superficies de alto orden):

Es una forma de renderización especialmente para terrenos y paisajes en un videojuego. Esta técnica se puede utilizar para otros modelos pero es exclusivo para modelos High-Poly, puesto que se especializa en deformar con curvas.

Se utiliza para conseguir un entorno foto realista de máxima calidad y suele consumir muchos recursos. Actualmente es una técnica muy utilizada en producciones de alto presupuesto, siendo muy inaccesible para estudios pequeños.

Árbol BSP (Binary Space Partioning):

Es el modelo o terreno base que el motor siempre va a renderizar en todo momento, el BSP se diferencía por ser lo que conforma el ecosistema y estructura de la escena. Un BSP se puede crear de muchas maneras pero hoy en día las técnicas más utilizadas son mediante Brushes y Heightmaps.

Brushes: Es una herramienta que se utiliza para aplicar deformaciones y texturas a un terreno (hierba, tierra, piedras, etc.).

Heightmaps: Se trata de una imagen codificada en escala de grises que determina la altura del terreno en diferentes coordenadas del mismo. El color de cada píxel representa la altura en ese punto del espacio, siendo el negro el valor mínimo y el blanco el máximo.

Culling:

Codificado que logra que los objetos que no se ven en determinado cuadro de la escena por causa de objetos que los obstaculizan (como una pared) no tomen tiempo de renderizado. Así se reduce la cantidad de trabajo del motor. El Culling es más fácil de implementar en juegos en donde la visión es controlada. Un método de Culling puede ser utilizando "Árboles BSP".

Una práctica habitual en el Culling es crear una cortina de niebla a una distancia de la cámara para que el observador no note el efecto de que el escenario se va renderizando a medida que avanza.

Iluminación:

La iluminación es un proceso de renderización en la que el motor ilumina todo lo que sea 3D ya sea por pixel o por vértice. Varía dependiendo de la configuración que haya establecido el usuario al motor, y es una parte muy compleja del desarrollo. Una buena iluminación puede dar un aspecto visual espectacular.

Por lo general la iluminación es influenciada por APIs, como DirectX y OpenGL.. El sombreado es otro factor sumamente importante y que reacciona mediante la luz, si el mundo obtiene buena iluminación también tendrá un buen sombreado. Cuando se ejecuta una aplicación, si se experimenta una bajada de FPS, lo recomendable es reducir la calidad de iluminación y sombreado, ya que suele ser lo que más recursos consume.

Flat Shading Lighting (Iluminación de Sombreado Plano):

Consiste en que cada polígono represente un valor leve que se pase al polígono completo que genere una imagen plana del mismo, a esta imagen también se le asigna un color determinado.

Vertex Shading (Sombreado de Vértice, Gouraud shading): solicita al motor de renderizado un color para cada vértice, luego por medio de interpolación se renderiza cada píxel por la distancia en relación con su respectivo vértice.

Phong Shading: es similar al Gouraud Shading, trabajan con la textura, solo que el Phong Shading usa a los píxeles en lugar de los vértices. Necesita más tiempo de procesamiento que el Vertex Shading pero su resultados son mucho mejores en cuestión de suavizado de texturas.

Light Map Generation (Generación del mapa de luz): se usa una segunda capa de textura (mapa de luz) que dará el efecto de iluminación a los modelos. Es un efecto excelente pero debe tomarse antes del renderizado pero si se tienen Luces Dinámicas

(o sea luces que se mueven, encienden o apagan sin intervención de programa). Se debe estar regenerando los mapas en cada Frame de animación, lo que toma mucha cantidad de memoria.

Textura: es esencial para que las escenas 3D se vean reales, en sí las texturas son imágenes que se rompen en los distintos polígonos del modelo, muchas imágenes tomarán mucho espacio en la memoria por eso se debe usar técnicas de compresión.

Mapeo MIP: consiste en preprocesar las texturas creando múltiples copias del mismo, cada una la mitad de la anterior.

Texturas Múltiples: requiere múltiples renderizados por lo que para obtener buen resultado se necesita una tarjeta con Acelerador de Gráficos. Se puede colocar una imagen sobre otra (más transparente) para dar el sentido de movimiento pulso o hasta sombra.

Bump Mapping: técnica vieja de texturas que tratan de mostrar como la luz se refleja en el objeto. Solo hasta hace poco se vuelto a retomar.

Antialiasing: El anti-aliasing revisa los polígonos y difuminará las bordes y vértices, para que los bordes no se vean como dentados. Esta técnica se puede hacer de dos maneras. La primera se realiza de modo individual, entremezclando polígonos para sobreponerlos unos delante de otros. La segunda se hace por medio de tomar todo el marco y quitarle los bordes dentados, pero esto requiere de mucha memoria.

Vertex and Pixel Shaders (Vértices y Sombreo de Pixeles): Con este método se pueden extraer y utilizar directamente las características y facilidades de la tarjeta de video, sin tener que utilizar mucho la API. Pero no es utilizable en todas las tarjetas.

Stencil Shadowing (Plantilla de Sombreado): la idea es renderizar una vista de un modelo desde la perspectiva de la fuente de luz y después utilizar esto para crear o para generar un polígono con la forma de esta textura sobre las superficies afectadas por el modelo. Así se obtiene una iluminación que parece real.

LOD (level of detail, nivel de detalle): el sistema de nivel de detalle está relacionada con la complejidad geométrica de los modelos. Algunos sistemas necesitan que se hagan múltiples versiones del modelo, para que dependiendo de cuan cerca esté el modelo, así será su cantidad de polígonos. Otros sistemas ajustan dinámicamente esta característica pero en este caso da más carga a la CPU.

Depth Testing (prueba de profundidad): Con esto se empieza a eliminar los píxeles ocluidos y se pone en práctica el concepto de sobre dibujado. La prueba de profundidad es una técnica utilizada para determinar que objetos están delante de otros en la misma localización del píxel.

Sobre Dibujado: es la cantidad de veces que se ha dibujado un píxel en un frame. Se basa en la cantidad de elementos existentes en la tercera dimensión (profundidad).

Scripting Systems (Sistemas de scripting):

Pre-scripted Cinematics: usada normalmente en una situación que necesita la explicación en una manera controlada. Para presentar las escenas en las que no se debe permitir al usuario interactuar, y, por medio de transiciones, se pasa a las gráficas reales del juego.

Visual Scripting Systems: permite manejar el script en un ambiente gráfico en lugar de un código escrito. Se maneja un carácter real en un ambiente de la escena real.

Inteligencia Artificial (IA):

Es la característica más importante que se le atribuye a un motor al lado de la representación de modelos o Render.

La inteligencia artificial de determinado juego puede tornarse muy compleja, primero se debe definir la línea base del comportamiento de los NPC (Non Player Characters - Personajes no Jugables), primero debe definirse qué hace el NPC, luego se delimita su "visión del mundo", que es lo es el NPC puede ver del mundo del juego; se debe tomar en cuenta que el personaje no sólo estará en medio del mundo del juego sino que también se interactuará con él, después vienen las rutinas de "Toma de Decisión", que son las acciones que debe realizar en función de lo que percibe del entorno.

4.1 Unity 3D:

Unity 3D es una plataforma orientada al desarrollo de videojuegos. Concretamente, se trata de un motor multiplataforma, es decir, permite crear videojuegos para soportes tales como PlayStation, Xbox, Wii, Pc, dispositivos móviles, e incluso para navegadores web.

Dispone de un potente diseñador de niveles, que, gracias al enorme soporte del que se nutre, tanto oficial como por usuarios casuales, cuenta con un gran número de recursos que pueden utilizarse libremente para proyectos propios.

En la store oficial de Unity, podemos encontrar todo tipo de elementos, tales como modelados de personajes, entornos, animaciones, texturas, archivos de audio, scripts que realicen determinadas acciones, etc. Incluso, es posible descargar ejemplos de proyectos para verlos en el editor y realizar pruebas con ellos. Esto es muy útil para que los nuevos usuarios aprendan viendo la estructura de un proyecto real. Las descargas oficiales pueden realizarse desde este enlace.

Además, se encuentran disponibles una serie de tutoriales (en inglés), aparte de los miles de tutoriales de gente anónima que circulan por internet. La comunidad tan inmensa que tiene, hace a Unity 3D una opción muy interesante a la hora de decantarse por el desarrollo, no sólo de videojuegos, sino de cualquier tipo de aplicación, puesto que permite ponerse en contacto con desarrolladores más expertos puede ser una buena solución a un problema que no sepamos resolver.

Este proyecto se ha realizado con la versión 4.6, aunque hay versiones más actualizadas. Es posible actualizar un proyecto a una versión posterior sin demasiadas dificultades, siendo necesario modificar en algún caso una pequeña parte del código fuente de algún script, siempre marcado por el detector de errores incorporado en la plataforma. De esta manera, tu proyecto siempre podrá estar en continuo desarrollo, al mismo tiempo que se desarrolla Unity.

Hoy en día, es uno de los motores más utilizados, tanto a nivel independiente como por grandes compañías. Algunos ejemplos de aplicaciones realizadas:

- Oddworld: New 'n' Tasty
  https://madewith.unity.com/games/oddworld-new-n-tasty
- Pillars of Eternity (Nominado a mejor videojuego RPG del año)
  https://madewith.unity.com/games/pillars-eternity
- The Coherentrx Suite (Aplicación móvil para diagnósticos clínicos)
  https://www.coherentrx.com/

4.1.1 Entorno Gráfico:

El editor gráfico se utiliza, sobre todo, para diseñar los niveles o "escenas" de nuestra aplicación. Se compone de 4 partes esenciales que pueden verse en la siguiente imagen:

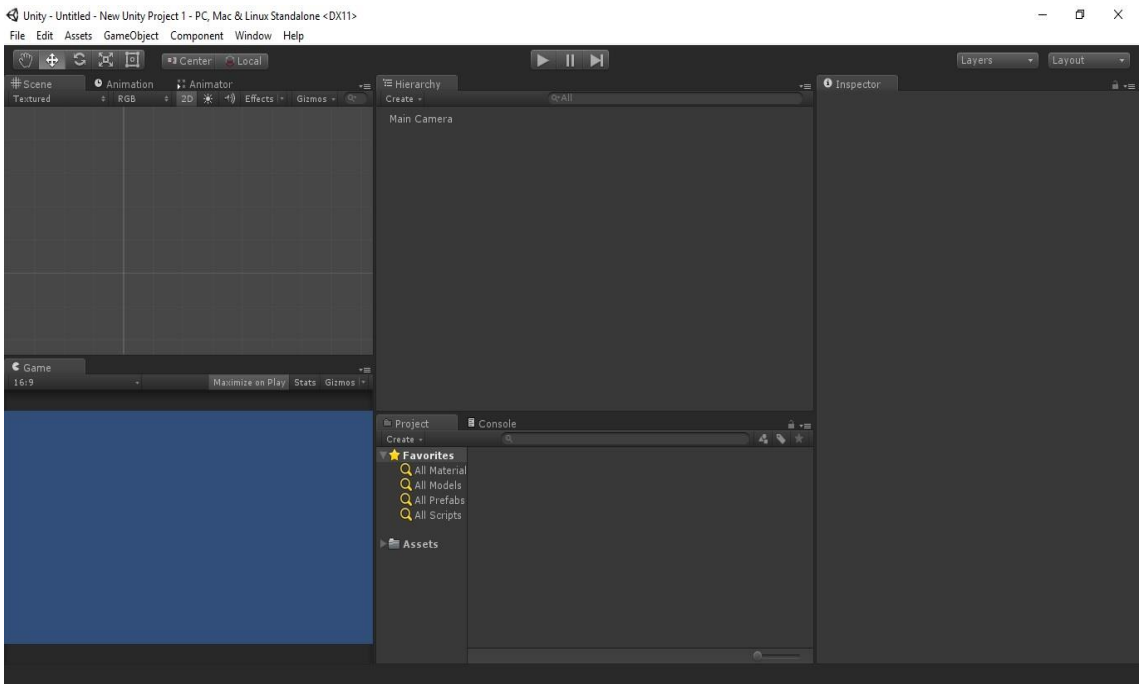


Imagen 16, Entorno gráfico

En la parte de la izquierda se muestran dos modos de vista, en el modo “Scene” se muestra el diseño de la escena actual y permite realizar cambios en el mismo, mostrando en todo momento cómo va cambiando la escena. Por el contrario, en el modo “Game”, se refleja la imagen de la cámara principal, es decir, donde está enfocando actualmente, y no permite modificación alguna. Se trata de un “preview” de nuestra aplicación que nos representa lo primero que se mostrará al ejecutar la escena actual. Los botones de ejecución, pausa y detención se encuentran en la barra de herramientas y se utilizan para probar la escena en tiempo real.

Continuando hacia la parte central, tenemos el apartado “Hierarchy” o “Jerarquía” en español. Su función es la de presentar los objetos que se encuentran en la escena y mostrar su estructura interna, si es que la tienen. Por ejemplo, un personaje controlable se compone de varias partes que pueden ser: Cabeza, brazos, piernas, etc. Cuando se crea un proyecto nuevo, el único objeto en escena es la cámara principal. Mediante la pestaña GameObject del menú, podremos insertar nuevos objetos en la escena, simples como cubos, esferas,…, o complejos (focos de iluminación, sistemas de partículas,…)

Justo debajo, en la pestaña “Project”, disponemos de todos los recursos del proyecto actual, y que pueden introducirse en la escena. Son los llamados “Assets” o “Recursos” (texturas, archivos de audio, scripts, materiales, etc.). En la otra pestaña situada en la misma ventana, hayamos la consola, que nos mostrará los errores de compilación y mensajes de salida de carácter “debug” que el programador estime oportunos.

En la siguiente zona, que se encuentra más a la derecha de la pantalla, encontraremos un visor de propiedades, que aporta información del objeto seleccionado. Algunos datos que muestra son su posición (x,y,z), orientación, textura, solidez,… y un amplio abanico de posibilidades que varían en función del objeto en cuestión.

Crearemos un cubo para ver algunas propiedades importantes de los objetos. El apartado Transform, hace referencia a la posición, orientación y escala en cada eje. Basándonos en este sistema tan simple, podremos situar cada objeto en el lugar que consideremos idóneo para formar el entorno. Para que lo que creemos tenga un comportamiento “real”, podemos añadirle propiedades tales como un cuerpo rígido, que hará que actúen fuerzas sobre él como la gravedad, las colisiones con otros objetos, o la propia luz. También se nos permite aplicarle texturas para cambiar su apariencia, ya sea un color básico o incluso una imagen.

El material también se ve afectado por el efecto de la luz, el cual puede ser reflectante, difuso, transparente,….

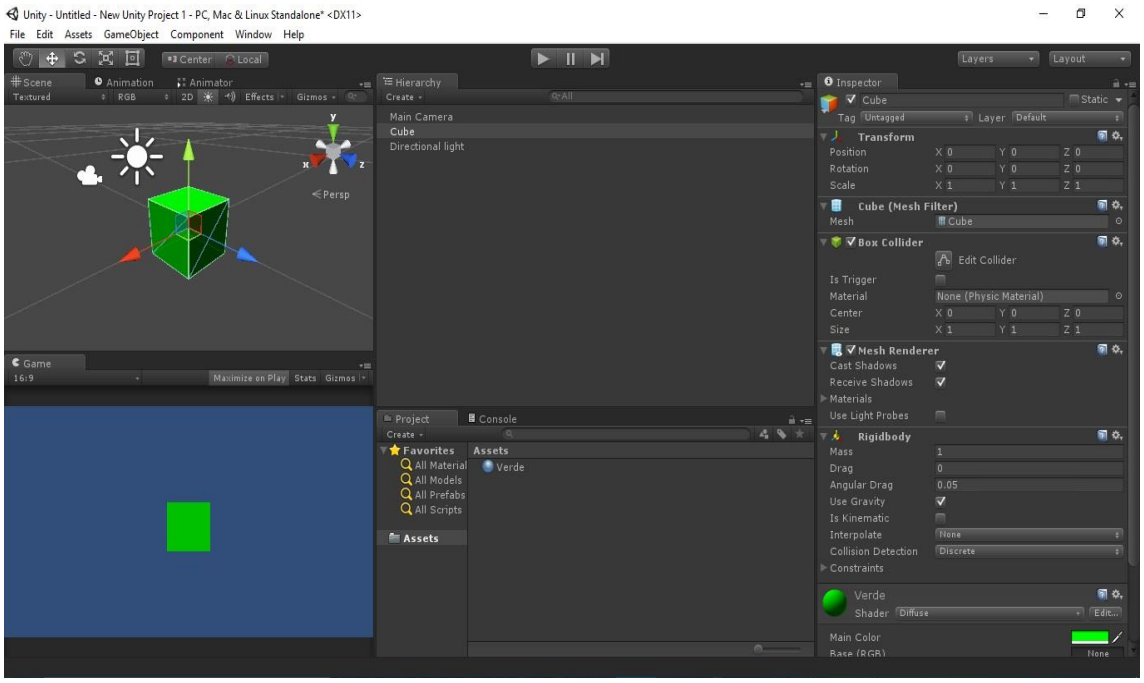
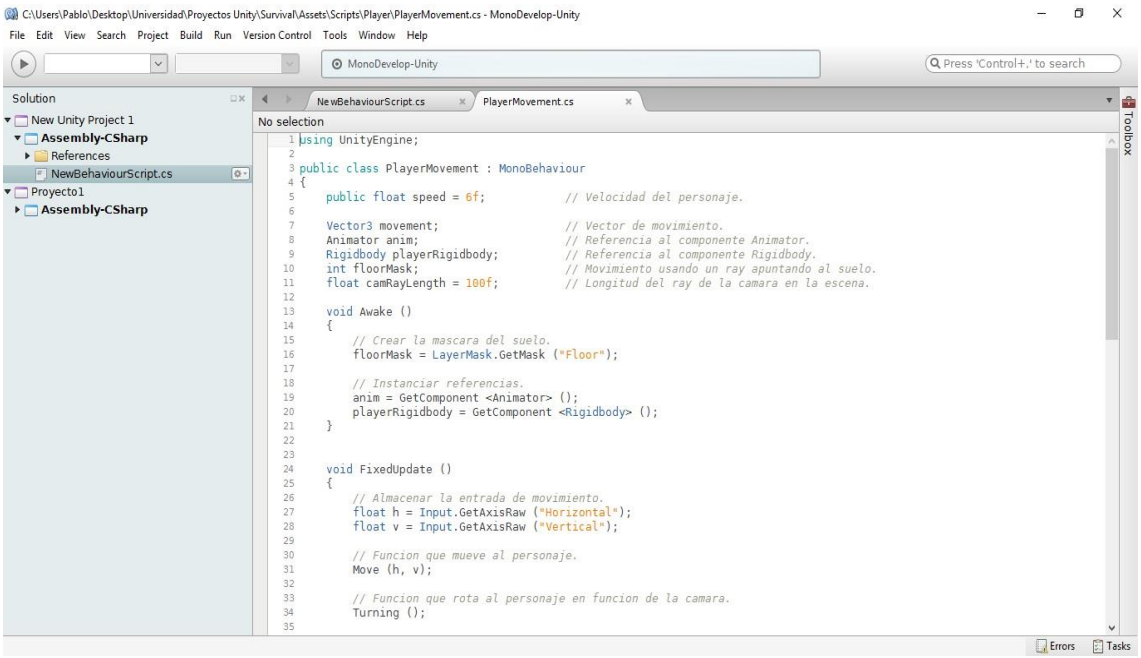


Imagen 17, Propiedades básicas de un GameObject

Como vemos, en la escena aparecen los mismos objetos que en la jerarquía, en este caso, la cámara, el cubo, y una luz direccional. La cámara está situada detrás del cubo apuntando a este, por lo que en la ventana inferior, aparece el cubo de frente, justo por la cara que tiene delante. La cámara también dispone de libre movimiento, por lo que podemos colocarla en otro lugar para obtener una perspectiva diferente. La parte azul sería un “background” predefinido que podemos cambiar a nuestro parecer, pudiendo colocar por ejemplo una imagen de un cielo y darle un toque de realidad.

4.1.2 Editor de Código:

Por otro lado, Unity3D soporta 3 lenguajes de programación: C#, JavaScript y Boo. Para realizar los scripts que modelarán el comportamiento de los objetos, viene incorporado el editor de código “MonoDeveloper”. Es un entorno muy cómodo para trabajar, ya que ofrece ayuda a la hora de implementar el código. Por ejemplo, dispone de motor de “matching” de instrucciones que recomienda como completar el código mediante una búsqueda, es decir, muestra una lista de variables, funciones, etc., que pueden utilizarse en base a lo que se está programando en ese momento.




```
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    public float speed = 6f;            // Velocidad del personaje.

    Vector3 movement;                   // Vector de movimiento.
    Animator anim;                      // Referencia al componente Animator.
    Rigidbody playerRigidbody;          // Referencia al componente Rigidbody.
    int floorMask;                      // Movimiento usando un ray apuntando al suelo.
    float camRayLength = 100f;          // Longitud del ray de la camara en la escena.

    void Awake ()
    {
        // Crear la mascara del suelo.
        floorMask = LayerMask.GetMask ("Floor");

        // Instanciar referencias.
        anim = GetComponent <Animator> ();
        playerRigidbody = GetComponent <Rigidbody> ();
    }


    void FixedUpdate ()
    {
        // Almacenar la entrada de movimiento.
        float h = Input.GetAxisRaw ("Horizontal");
        float v = Input.GetAxisRaw ("Vertical");

        // Funcion que mueve al personaje.
        Move (h, v);

        // Funcion que rota al personaje en funcion de la camara.
        Turning ();

```

Imagen 18, Editor de código

Gracias a los scripts, las propiedades vistas en el apartado anterior pueden ser modificadas, buena práctica para los usuarios más avanzados. Para hacer esto, hay que instanciar un objeto como un GameObject, al cual se le asocia el objeto en cuestión mediante la interfaz gráfica. Una vez enlazada la variable con la instancia, podemos acceder a la propiedad deseada mediante <nombredelavariable>.<propiedad>. Un ejemplo para modificar la posición del cubo anterior sería:

GameObject objeto; => Declaración de la variable.

objeto.transform.position = new Vector3(1, 1, 1);

Se busca el componente transform del objeto y se modifica la propiedad posición, utilizando un vector de tres coordenadas (una por cada eje), siendo el punto de destino X = 1, Y = 1, Z = 1.

La potencia de los scripts permite incluso la no necesidad de usar el editor gráfico, aunque no es recomendable, ya que no permite la visualización en el momento de los cambios en la escena y dificulta mucho el desarrollo. Es fundamental encontrar el equilibrio entre lo que se puede hacer mediante código y a través de la interfaz visual.

Hay que tener en cuenta que los errores en el código no aparecen en Mono Developer, sino que únicamente se muestran en la consola de la interfaz, siendo necesario alternar entre las dos herramientas.

4.1.3 Modelado 3D:

Unity solo permite la creación de tipos básico de objetos, como cubos y esferas, por lo tanto, es necesario utilizar programas externos si se desean estructuras más complejas. Hace falta aclarar, que no todos los formatos son aceptados.

Los realmente compatibles son .fbx y .obj, pero también es posible importar modelos de otros software de diseño tales como Blender, Maya o 3DMax, aunque existen inconvenientes (se necesita tener instalado el programa en cuestión con una licencia válida, el proceso de validación y dibujado del modelo es más lento, y ocupan más espacio en memoria).

Para introducir un modelo en el proyecto actual desde el editor, debemos buscar en la barra de herramientas el desplegable “Assets”, el cual nos dará la opción de importar un nuevo recurso (Import New Asset). Una vez aparezca el cuadro de diálogo, buscamos el directorio que contiene dicho modelo y le damos a Importar. Para que dicho elemento aparezca en la escena, se ha de arrastrar al visor de la escena y después, colocarlo en posición, bien introduciendo las coordenadas en el componente Transform, o bien moviendo el objeto con el ratón.

El modelo es importado como un objeto prefabricado, el cual puede contener a su vez otros elementos dependiendo de su estructura. Esto es muy útil para tareas de animación, puesto que es posible desplazar cada uno de estos elementos de forma independiente, como si fuera un esqueleto.

En esta aplicación se utilizarán edificios, suelos y decoraciones, por lo que no será necesario utilizar animaciones. Todos ellos tendrán formato .fbx, dado el gran número de modelos y la necesidad de que la aplicación sea ágil y tenga una gran distancia de dibujado sin verse afectado el rendimiento.

Un modelo es únicamente una figura, una forma sin colores que diferencien unas partes de otras. De ahí la importancia de las texturas. Por ejemplo, un edificio con forma de cubo es simplemente eso, un cubo, en cambio, si le aplicamos ciertas texturas en lugares concretos, podremos obtener un edificio con ventanas, puertas, pilares,…. He aquí una comparativa de un modelo con y sin texturas:

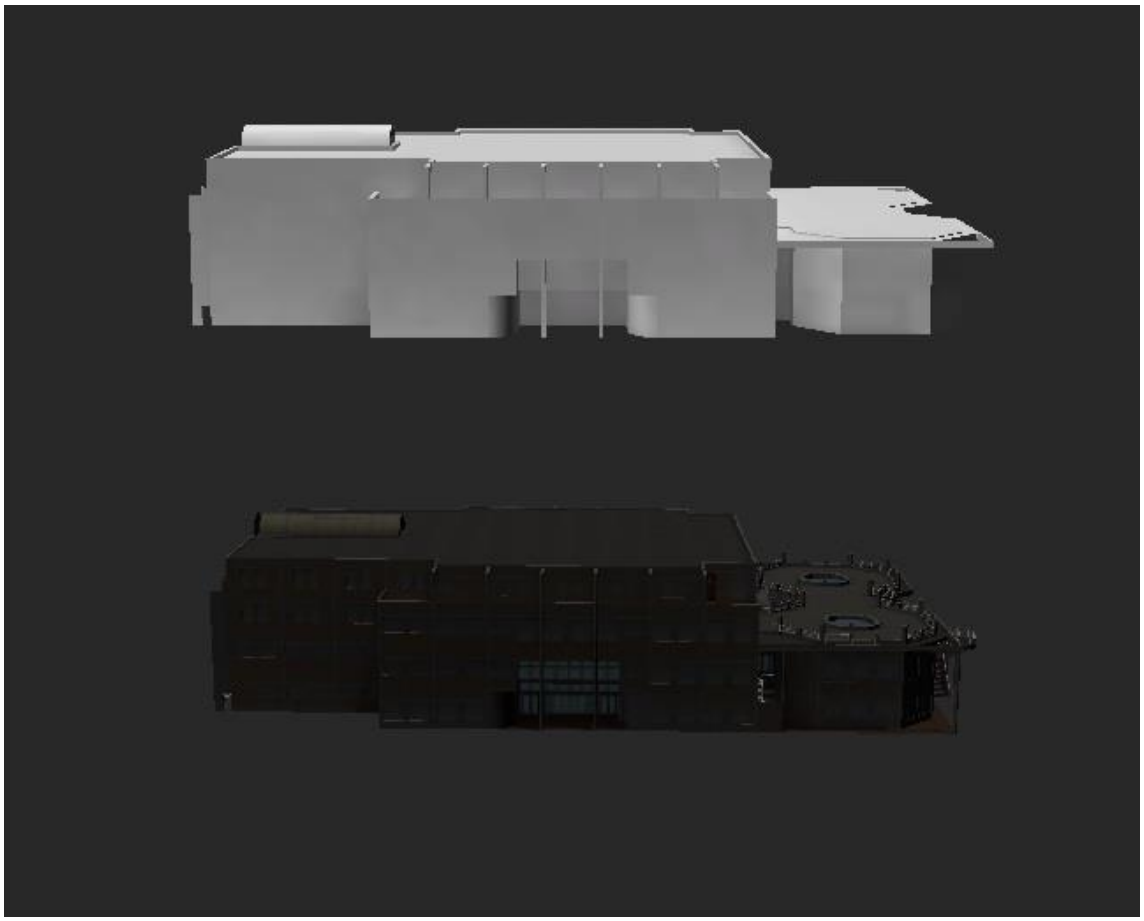


Imagen 19, Texturas en modelos 3D

Es importante una vez introducidos en la escena, determinar el comportamiento o las propiedades del objeto al que representan. Un edificio es un cuerpo rígido al que le afectan las fuerzas físicas y que no se puede atravesar, por tanto, es necesario indicar todas estas características en el editor. También es obvio que un edificio debe estar situado en un lugar concreto, esto es, sobre el suelo y en un lugar que no esté ocupado ya por otro objeto.

4.1.4 Compilación:

Al ser un motor multiplataforma, es posible compilar una aplicación para distintos dispositivos, por lo que se ha de seleccionar el deseado antes de empezar el proceso.

La opción para compilar un proyecto se encuentra en el apartado File – Build Settings… En la nueva ventana que aparece, se deben seleccionar aquellas escenas que formen parte del producto final, entre todas las que pertenecen al proyecto actual en cuestión, así como la plataforma a la que está dirigida la aplicación, y pulsar el botón “Build”.

Aparecerá el explorador de archivos para seleccionar el destino donde guardar el ejecutable y un directorio “Data” que contendrá todos los recursos necesarios para que funcione correctamente (generalmente archivos metadata que contienen la información referente a todo el contenido multimedia tales como imágenes, texturas, materiales, modelos, etc.).

Dicho ejecutable tendrá un tipo de extensión que dependerá del sistema operativo para el que se ha compilado, siendo .exe en el caso de Windows, .html para navegadores web, apk para android, etc.

Es importante saber qué tipo de archivos soporta cada plataforma, ya que por ejemplo, las texturas no funcionan igual en todas ellas, siendo más restrictivo en el caso de los sistemas equipados en Smartphone, al no ser tan potentes como el PC. Es por esto que nada más empezar el desarrollo de un proyecto, se seleccione la plataforma de antemano, para evitar un posible mal funcionamiento.

Del mismo modo, también es importante decidir si se enfocará a 2D o 3D al inicio, puesto que las librerías que se manejan son distintas y si se realizara un cambio en mitad del desarrollo, se deberán incorporar tanto las de un modo como las del otro, y podrían existir conflictos.

Mención especial para el caso de dispositivos pertenecientes a Apple (Mac OS e IPhone), ya que es necesario disponer de un Mac para poder compilar proyectos destinados a ellos. Es decir, desde un PC con sistema operativo Windows o Linux, no se permite compilar para plataformas relacionadas con Apple. Además existen librerías propias que deberán utilizarse para funciones tales como reproducir un video o realizar una llamada, que no son compatibles con las librerías comunes que se utilizan para el resto de sistemas.

Hay que tener en cuenta que si se utiliza la versión gratuita, compilar, publicar y distribuir aplicaciones está permitido, siempre y cuando las ganancias obtenidas por la misma no superen los cien mil dólares. En tal caso, se deberá abonar una cantidad en relación a los beneficios. Además, siempre aparecerá la marca de Unity 3D al ejecutarla, al no disponer de una licencia pro.

5. APLICACIÓN DESARROLLADA

5.1 Diseño y Estructura de la Aplicación:

Como se ha explicado al comienzo de este documento, existen dos proyectos, cuyo diseño y desarrollo es completamente aislado. Por un lado, toda la parte del modelado y creación del entorno del campus de Vera de la Universidad Politécnica de Valencia y la interacción con el mismo, y por otro, un sistema interactivo que consiste en la visualización de archivos, mediante un menú circular.

Para la primera parte, detallaremos los pasos a seguir mediante el siguiente diagrama:

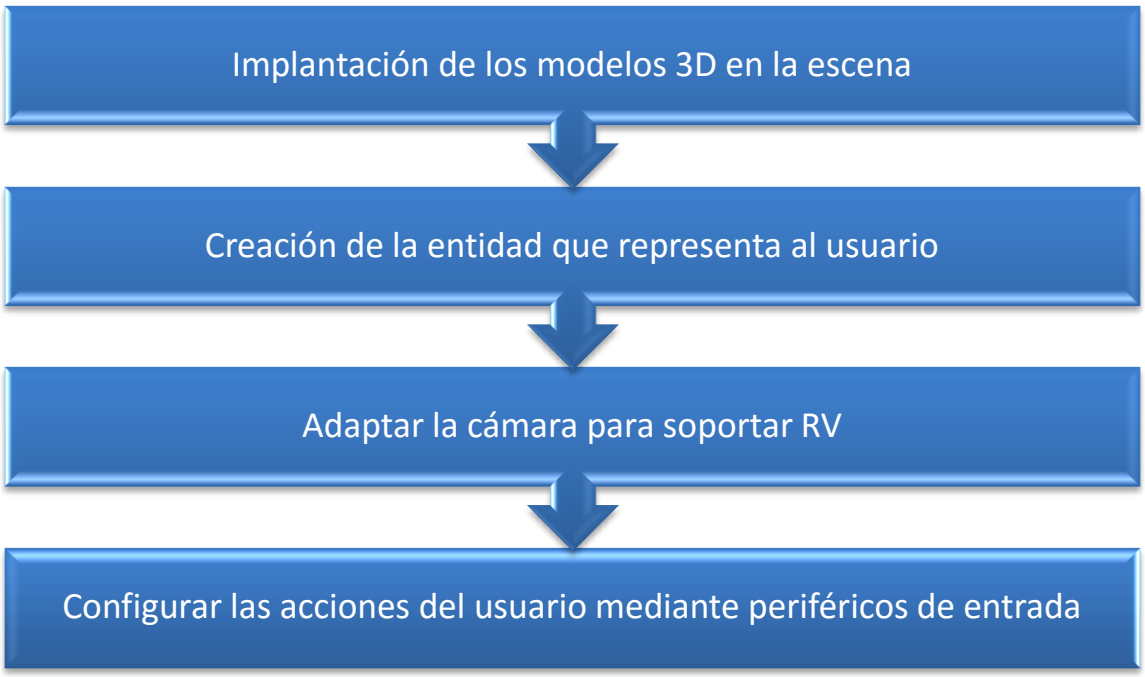


Figura 1, Esquema de diseño parte 1

Construir el campus virtual será el primer objetivo. Debe recrear fielmente el campus real, por lo que la ubicación de los edificios, carretera, vegetación y demás modelos, deben coincidir perfectamente en el plano. El componente transform de cada modelo será la clave para conseguir esto. Deben calcularse perfectamente las coordenadas de cada elemento con el fin de que ninguno se superponga con otro o que queden espacios vacíos, lo que supondría un efecto fatal a la vista del usuario.

También tener en cuenta que el espacio debe ser cerrado, esto es, delimitar la superficie explorable, porque de lo contrario, al salirse de dichos límites y acabarse el terreno, el usuario caería infinitamente hacia abajo debido al efecto de la gravedad.

Una vez montados los modelos 3D, deben aplicarse las texturas correspondientes. De lo contrario, el mapeado no tendrá tonalidades y únicamente se verán estructuras de color blanco. Se ha de iluminar la escena para apreciar bien los colores y matices de las texturas.

Además de las texturas, deberemos otorgar propiedades a cada elemento, tales como rigidez, solidez, cajas de colisión, o la capacidad de refractar la luz que inciden en ellos, para crear sombras.

El usuario será un objeto al cual se le añadirá una cámara que acompañará a este en todo momento, situada a una altura concreta para hacer la función de “ojos”. Se definirá el movimiento del usuario mediante un script, en primera instancia, utilizando como método input el teclado y el ratón.

Recordando el concepto de realidad virtual y el estereoscopio, para que se dé el fenómeno de tridimensionalidad, es necesario utilizar dos puntos de vista con diferente enfoque sobre la escena. Por lo tanto, utilizaremos una cámara diferente para cada ojo, que enviarán imágenes simultáneamente a su ojo asociado, pero separadas por una distancia en el eje X.

Aprovechando los sensores de Oculus Rift, conseguiremos que el usuario sea capaz de mirar alrededor de sí mismo al mover la cabeza. Esto se consigue utilizando la información recogida por el acelerómetro, el giroscopio y la cámara de posicionamiento.

Además de utilizar el teclado para desplazarse, se podrán utilizar tanto el controlador de Leap Motion, como el mando de Playstation 3 con el mismo fin. Los gestos a realizar con las manos y sus funciones serán las siguientes:

- Avanzar: Desplazar una mano hacia delante.
- Retroceder: Mover la mano hacia atrás.
- Girar: Llevar la mano hacia derecha/izquierda, dependiendo de la orientación deseada.
- Saltar: Cerrar el puño y efectuar un movimiento ágil hacia arriba.
- Interactuar con terminales: Llevar la palma de la mano hacia el terminal en cuestión.

Las mismas acciones podrán ser llevadas a cabo con el mando, utilizando tanto las flechas del D-Pad (arriba, abajo, izquierda y derecha), como con el stick analógico inclinándolo hacia la dirección correspondiente. Para saltar, utilizaremos el botón triángulo. Los paneles de información se activarán pulsando el botón equis al situarse delante de los mismos.

El planteamiento de la segunda parte es totalmente diferente. Es un proyecto nuevo en el que se plantean los siguientes problemas:

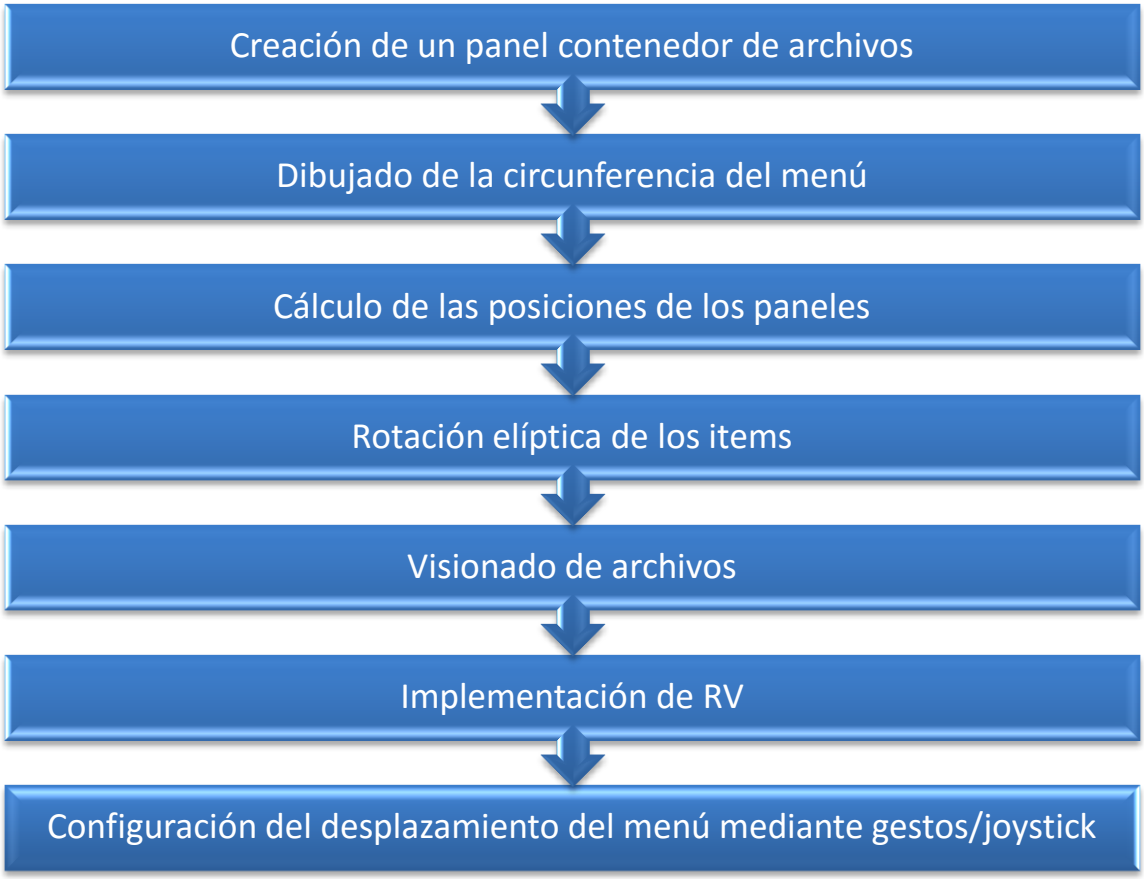


Figura 2, Esquema de diseño parte 2

Un panel será un objeto de tipo Cubo, con una longitud y una altura determinadas por la resolución de la textura de la imagen o vídeo en cuestión que se quiera visionar. Esto se hace para no perder la proporción de las imágenes y evitar así deformaciones en las mismas. La profundidad del mismo será ínfima, para dar la sensación de que es un panel, y no un cubo. En cambio, si el objeto es un modelo 3D, será una miniatura del mismo (o una esfera para texturas de carácter 360º).

No obstante, será necesario aplicar una escala de la imagen para evitar que los paneles sean demasiado grandes. Un ejemplo:

Tenemos una imagen de 1028x768 y queremos incrustarla en un panel. Sin aplicar una escala, tendríamos un panel que ocuparía una gran parte de la pantalla (si no toda, dependiendo de la resolución que disponga nuestro equipo). La forma de escalar una imagen es definir por ejemplo una altura fija, y en base a dicha altura, calcular la longitud de la imagen sin que pierda su relación de aspecto X/Y. Si consideramos que la altura de los paneles es de 50 pixeles, podemos aplicar una regla de tres para saber el tamaño de la longitud de la imagen.

$$x = \frac{1028 * 50}{768}$$

Por otra parte, tenemos la interfaz del menú, que seguirá el siguiente esquema:

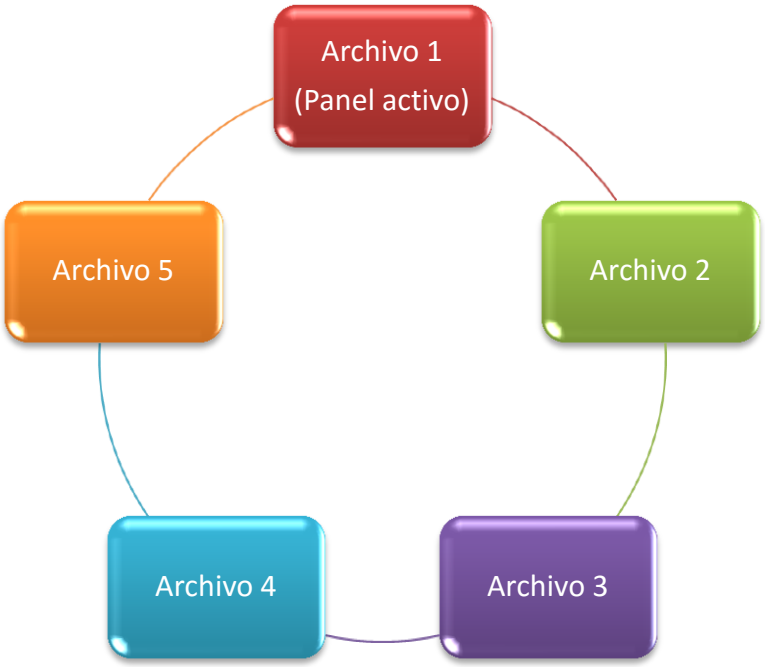


Figura 3, Diseño del menú

Dependiendo del número de archivos que queramos implantar en el menú, tanto la disposición de los mismos como su posición en el espacio variarán, es decir, la circunferencia contará con mayor o menor número de puntos. Dichos puntos serán los paneles a los cuales se enlazará cada archivo. El archivo que se encuentre frente al usuario será el "panel activo", sobre el cual se podrá interactuar.

Se podrá desplazar el menú hacia derecha o izquierda para cambiar dicho panel aplicando una rotación de unos grados determinados por la longitud de los ítems listados. La rotación se efectuará en el eje Y, dado que el menú se desplegará a lo largo de los ejes X y Z (longitud y profundidad). La altura o eje Y de todos los elementos será la misma y se ubicará con respecto a la altura de la cámara.

Se dibujará una circunferencia de color que determinará el perímetro del menú, sobre la que se situarán los paneles. Como detalle estético, se colocará una esfera en el centro para evitar la sensación de vacío, así como un logotipo en el fondo perteneciente a la facultad propietaria del terminal accedido.

La distribución de los archivos sobre el menú consistirá en partir la circunferencia en N partes iguales, siendo N el número estos, dando lugar a N puntos en el espacio sobre los que se instanciará cada panel. El principal problema es que el espacio es limitado, por lo que se debe acotar el tamaño de N, para que los objetos no se superpongan.

Si queremos hacer girar el menú para seleccionar otro panel, debe haber una rotación de todos los elementos que lo componen. En función de su número, el ángulo entre la posición actual y la posición objetivo con respecto al centro debe ser aquel

que permita que siempre haya un “panel activo”, es decir, siempre debe mostrarse una imagen delante del observador.

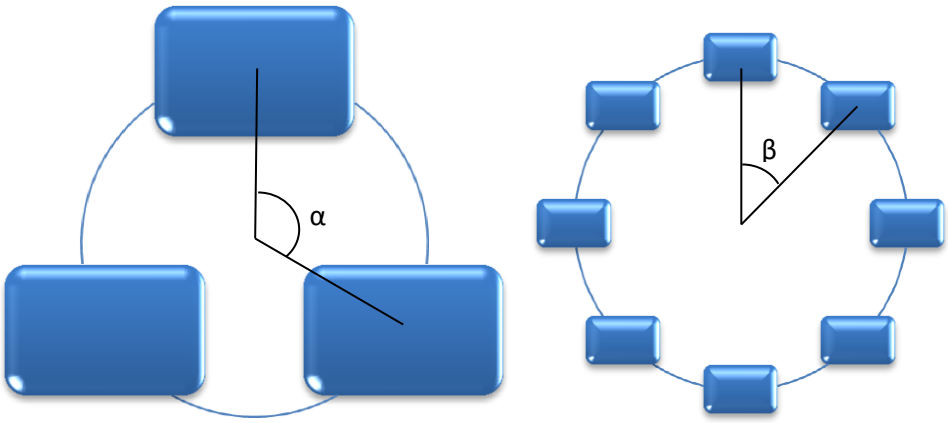


Figura 4, Diferenciación en grados de rotación del menú

La proporción en los grados de rotación es inversamente proporcional al número de elementos del menú. Estos grados son respecto al centro de la circunferencia pero, también deben rotar respecto a su propio centro, de lo contrario, no se mostrarían de cara al usuario, por lo que es necesario calcular otra rotación en sentido contrario para compensar la anterior, sobre cada panel.

Si el panel activo contiene una imagen, esta podrá ampliarse para verse con más detalle. En cambio, si se trata de un vídeo, se ampliará y reproducirá al seleccionarse. Además, se podrá pausar, detener, reanudar, o visualizar desde el principio. En caso de ser un modelo 3D, se abrirá una nueva escena en la cual aparecerá el objeto en cuestión para su libre exploración, pudiéndose rotar en todos los ejes o ampliar y reducir.

Para las texturas en 360º, se situará una cámara dentro de una esfera, cuya cara interior será un material creado a partir de la textura (imagen o vídeo). Se permitirá el giro de cámara para observar toda la esfera. Este truco permite crear la sensación de estar dentro de la imagen que se está viendo en pantalla.

Una vez creado el funcionamiento del menú, se aplicará la misma técnica de la primera parte del proyecto para implementar la realidad virtual.

Por último, diseñar el conjunto de gestos (o mapeado del joystick) que permiten realizar todas las acciones descritas anteriormente:

- Girar menú: Realizar un movimiento con la mano en un sentido u otro, para que avance en el mismo (o fechas izquierda/derecha).
- Seleccionar elemento: Elevar la mano para visualizar el contenido del panel activo (o botón equis). Si se desea volver al menú, se deberá efectuar un movimiento hacia abajo con la mano (o botón círculo).
- Reproducir vídeo: Después de seleccionar un vídeo, llevar la mano hacia la parte derecha de la pantalla (o botón equis).
- Pausar vídeo: Después de seleccionar un vídeo, llevar la mano hacia la parte izquierda de la pantalla (o botón cuadrado).
- Detener vídeo: Después de seleccionar un vídeo, llevar la mano hacia la parte de arriba de la pantalla (o botón círculo).
- Girar modelo 3D sobre eje Y: Llevar la mano hacia la parte derecha/izquierda (o mover el stick analógico horizontalmente).
- Girar modelo 3D sobre eje X: Llevar la mano hacia arriba/abajo (o mover el stick analógico verticalmente).
- Girar modelo 3D sobre eje Z: Extender un dedo y realizar un giro circular hacia la derecha/izquierda (o botón R1/L1).
- Zoom: Cerrar el puño y acercar/alejar el puño para ampliar/reducir (o botón R2/L2).
- Mover la cámara en contenido 360º: Llevar la mano hacia donde quiera desplazarse, siendo el centro el punto neutro que hará la cámara se quede en el punto donde se encuentra (o utilizando el stick analógico).

Estas son, a gran escala, las tareas a realizar. Veremos en profundidad el trabajo a realizar en la parte de desarrollo.

5.2 Desarrollo de la Aplicación:

Escena del campus:

Para empezar, abriremos el editor de Unity 3D y crearemos un nuevo proyecto. Nos encontraremos con una escena vacía y una cámara principal. Importaremos los modelos 3D y los buscaremos dentro del editor. Si seleccionamos la opción “All Models” de la ventana “Project”, nos aparecerán todos los recursos con extensión .fbx o .obj que tenemos asociados al proyecto.

Creamos un GameObject Empty o vacío, que no tiene ninguna propiedad, pero que nos servirá de contenedor para tener organizados todos los modelos (podemos ponerle el nombre que queramos, pero es una buena práctica utilizar nombres que identifiquen una finalidad o descripción).

Necesitaremos apoyarnos en un plano para tomarlo como referencia a la hora de construir el campus, así que nos basaremos en el de la aplicación alojada en esta web: http://www.upv.es/plano/plano-3d-es.html

Buscamos aquellos que contengan en su nombre la palabra “SUELO” y los vamos arrastrando encima del objeto que acabamos de crear en la ventana Hierarchy (nota: de todos los modelos hay dos versiones; se deben utilizar aquellas en las que no aparezca la etiqueta LOW en el nombre).

Es importante que a medida que coloquemos cada parte del suelo, ajustemos las coordenadas para que queden correctamente. De lo contrario, si se incluyen todas a la vez, nos quedarán unas encima de otras y nos dificultará el progreso. Una vez colocados, la escena debe quedar de la siguiente manera:

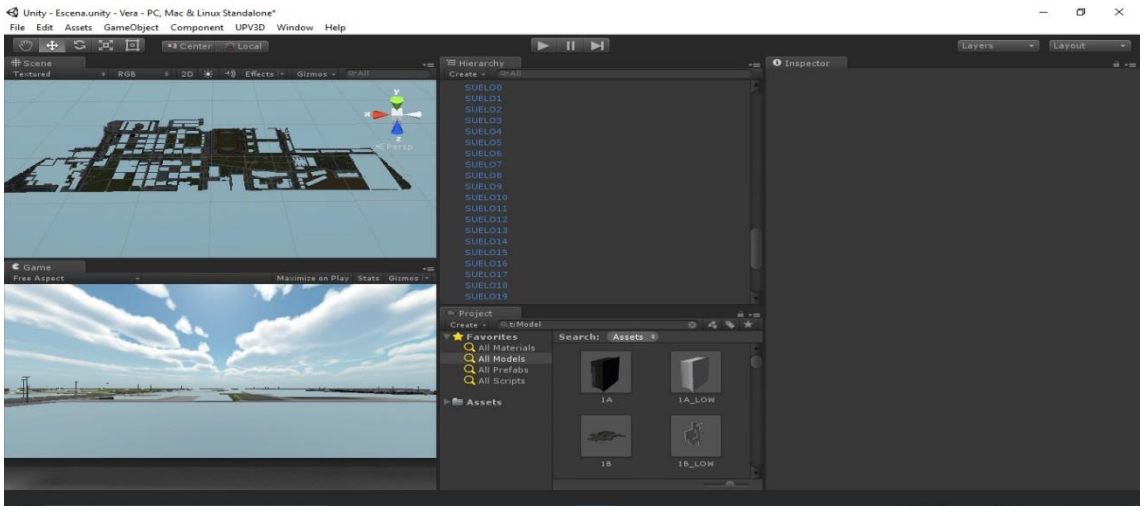


Imagen 20, Colocación del suelo

Para el cielo, se ha de crear un Skybox a partir de un material llamado MNubes (Material de Nubes). Dicho material se crea a partir de seis texturas que se pueden importar desde packetes por defecto de Unity y que representan un cielo con nubes. Para crear un material, primero creamos una carpeta de nombre Material, y después pulsamos el botón derecho del ratón estando situados en el interior de la carpeta. Le damos a la opción Create y después a Material.

Seleccionamos el material que acabamos de crear, y miramos sus propiedades en el inspector. Aparecerá un desplegable con una serie de opciones, de entre las que elegiremos “RenderFX/Skybox”. Nos aparecerán seis huecos a los que deberemos adjuntar las texturas que conforman el cielo, en forma de cubo, con sus caras Back, Front, Left, Right, Up y Down. Una vez colocadas correctamente, ya tendremos el material preparado. Ahora en la barra de herramientas buscamos la opción Edit y seleccionamos “Render Settings”. En el inspector de propiedades a la derecha, buscaremos “Skybox Material” y seleccionaremos el material que acabamos de crear.

Acto seguido, procederemos a introducir en la escena los modelos correspondientes a los edificios de igual forma que con el suelo, teniendo en cuenta el lugar donde debe ir cada uno, y haciendo que coincidan en el plano.

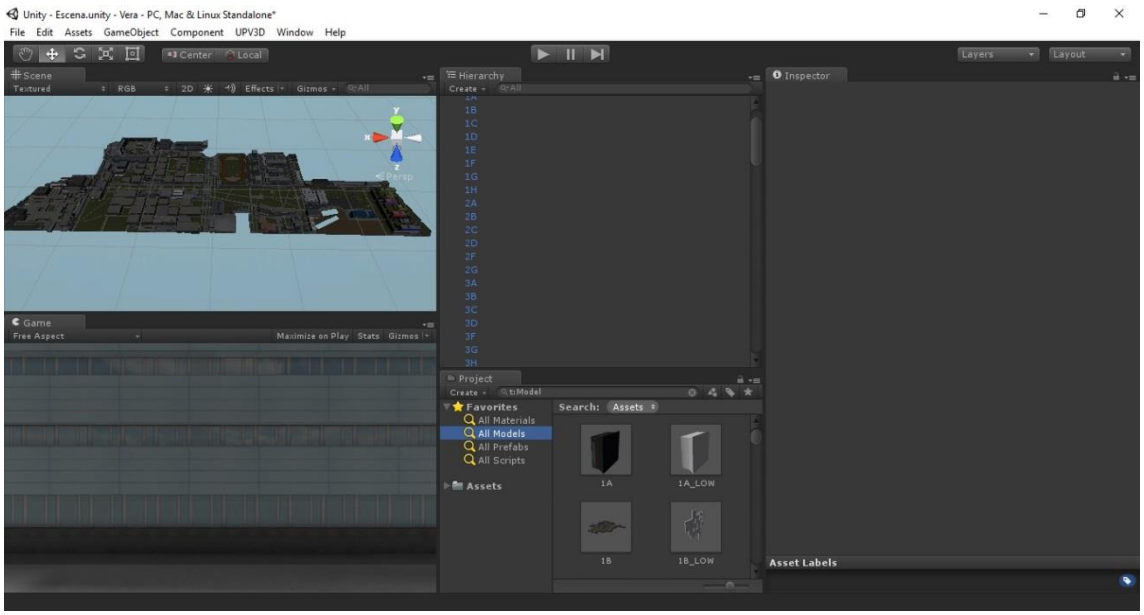


Imagen 21, Escena Montada

Lo siguiente será generar una entidad que se pueda controlar durante la ejecución y que representará al usuario. Para ello, utilizaremos un GameObject Empty y lo llamaremos “Player”. Haciendo uso del objeto “Main Camera” y moviéndolo en el panel de jerarquía dentro de “Player”, conseguiremos una “cámara móvil, ya que al desplazarse el objeto que acabamos de crear, la cámara se moverá con él.

Ahora vamos a escribir unas líneas de código que harán que este objeto se pueda mover utilizando como método input el teclado.

Necesitaremos una carpeta nueva a la que llamaremos “Scripts”, dentro de la cual pulsaremos con el botón derecho, después en Create, y por último en C# Script. La funcionalidad será la de mover al personaje, por lo que le pondremos un nombre acorde a su función. Se lo añadimos al objeto “Player” arrastrándolo desde la carpeta hasta él. Una vez hecho esto, adquiere el comportamiento que especifiquemos en su código.

Abrimos el script con Mono Developer, haciendo doble click sobre el mismo y escribimos lo siguiente:

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class Movimiento : MonoBehaviour
5  {
6      public float velocidad = 1f; //Velocidad de movimiento
7
8      void FixedUpdate()
9      {
10         //Vector de coordenadas de movimiento en X y Z
11         Vector3 vectorDireccion = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
12         this.transform.position += vectorDireccion * velocidad;
13     }
14 }
15
16
```

Imagen 22, Iniciación al movimiento

La función Input.GetAxis() recoge la entrada por teclado de aquellas teclas que implican movimiento, siendo estas las flechas de dirección, o aquellas que quieran definirse en el editor. Dicha entrada se codifica en 1 o -1, según si representa un valor positivo o negativo tanto en horizontal como en vertical. Como el desplazamiento puede ser en los ejes X y Z, se construye un vector (x,y,z), donde el valor x vendrá determinado por la dirección horizontal, el valor de y siempre será 0, y el eje z indicará si el usuario avanza o retrocede.

Como el input siempre tendrá valor equivalente a una unidad, podemos variar la velocidad de desplazamiento utilizando una variable auxiliar que habrá de multiplicarse por el valor de la entrada input. Una vez hechos los cálculos, cambiamos la posición del objeto en la escena mediante su componente transform.position, que será calculada cada en cada frame, pero sólo variará si el usuario indica que debe moverse.

Con esto, ya está construida la escena y se permite el movimiento a través de la misma. Lo siguiente es adaptarla a realidad virtual y a otros sistemas de input además del teclado, importando las librerías de Oculus Rift y Leap Motion, las cuales se pueden descargar desde el paquete Unity Integration alojado en la web:

https://developer.oculus.com/downloads/. Una vez descargado, se instalan haciendo doble-click en el archivo .exe obtenido.

Acto seguido, se oculta el "Player" creado anteriormente y se importa a la escena el asset "OVRPlayerController", situado en el directorio Assets/OVR/Prefabs/. Este se ha convertido ahora en el objeto que representará al personaje que se moverá por la escena. La diferencia con el anterior, es que ahora en lugar de tener una cámara, se compone de dos, una para cada ojo, que apuntan a la escena en el mismo sentido desde un punto de vista diferente, definido por la separación entre ambos ojos.

Aplicando el código anterior al nuevo personaje, arrastrando el script al objeto, obtendremos una cámara de realidad virtual que es capaz de desplazarse mediante input por teclado.

El siguiente paso es utilizar nuevas formas de input para aumentar la sensación de realidad. Para esto, disponemos del controlador Leap Motion y un joystick de PlayStation 3. El primero no necesita configurarse, ya que al integrar su paquete de librerías, queda totalmente adaptado a la plataforma, sin embargo, en el caso del mando, deberemos asignar los botones a las diferentes acciones que representen su activación mediante tags.

El mapeado input de Unity se encuentra en la opción Edit - Project Settings - Input, de la barra de herramientas del editor. Deben ajustarse el tamaño de acciones para ser capaces de otorgarle una acción a cada botón.

Cada nuevo tag o acción debe tener un nombre identificativo de lo que representa (por ejemplo, botón X, Stick Izquierdo, etc.), y se ha de relacionar con el botón en cuestión. Los botones del mando son numerados por Unity a partir del 0, y cada botón representa un número distinto. El mapa de botones y su numeración es la que puede verse en la siguiente imagen:

Imagen 23, Mapeado Mando PS3, http://forum.unity3d.com/threads/ps3-button-map.89288/

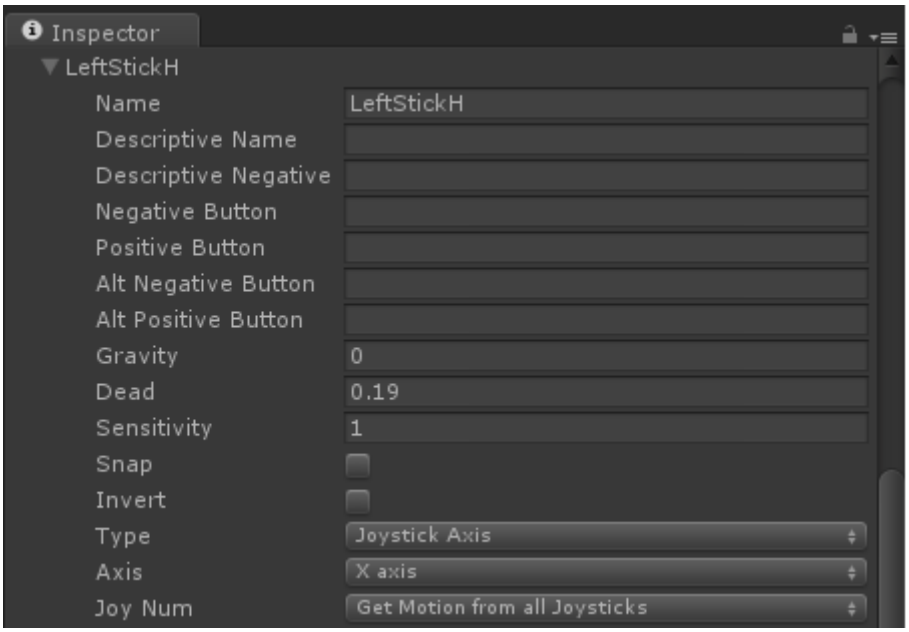


Imagen 24, Ejemplo Input Stick Izquierdo, Mando PS3

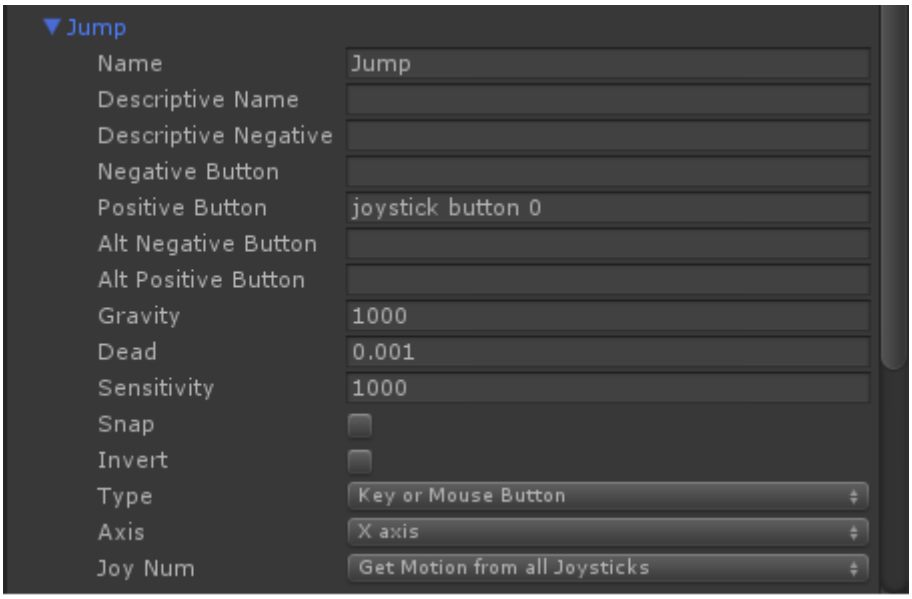


Imagen 25, Ejemplo Input Botón X, Mando PS3

Una vez declarados los tags de los botones, se podrá hacer referencia a ellos mediante código, para darles una funcionalidad concreta, mediante el nombre otorgado a cada uno de ellos.

Ahora que ya están configurados ambos controladores, diseñamos un script que implique movimiento, haciendo uso de ellos en lugar del teclado.

```
23      void Update()
24      {
25          // Get the input vector from kayboard or analog stick
26          Vector3 directionVector = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
27          Frame frame = controlador.Frame ();
28          HandList manos = frame.Hands;
29          FingerList dedos = manos [0].Fingers.Extended();
30          int extendidos = dedos.Count;
31
32          if (manos [0].PalmPosition.x > 50 || Input.GetAxis ("RightStickH") > 0)
33              transform.Rotate (0, 1, 0);
34          if (manos [0].PalmPosition.x < -50 || Input.GetAxis ("RightStickH") < 0)
35              transform.Rotate (0, -1, 0);
36          if (manos [0].PalmPosition.z < -50 || Input.GetAxis ("LeftStickV") < 0)
37              directionVector.z = 1;
38          if (manos [0].PalmPosition.z > 50 || Input.GetAxis ("LeftStickV") > 0)
39              directionVector.z = -1;
40          if (directionVector != Vector3.zero)
41          {
42              // Get the length of the directon vector and then normalize it
43              // Dividing by the length is cheaper than normalizing when we already have the length anyway
44              float directionLength = directionVector.magnitude;
45              directionVector = directionVector / directionLength;
46
47              // Make sure the length is no bigger than 1
48              directionLength = Mathf.Min(1.0f, directionLength);
49
50              // Make the input vector more sensitive towards the extremes and less sensitive in the middle
51              // This makes it easier to control slow speeds when using analog sticks
52              directionLength = directionLength * directionLength;
53
54              // Multiply the normalized direction vector by the modified length
55              directionVector = directionVector * directionLength;
56          }
57
58          // Apply the direction to the CharacterMotor
59          motor.inputMoveDirection = transform.rotation * directionVector;
60          if (manos [0].PalmVelocity.y > 400 && extendidos == 0)
61              motor.inputJump = true;
62          else
63              motor.inputJump = Input.GetButton("Jump");
64      }
65  }
```

Imagen 26, Implementación del movimiento

La información recogida por Leap Motion se almacena en la clase Controller, por lo que hay que declarar una variable de dicho tipo e inicializarla llamando al constructor de la clase. Esto se hace sólo una vez, al comienzo de la aplicación, gracias a la función Awake(), que se activa justo en el momento en el que se abre la escena.

El objeto controlador que se acaba de crear, contiene un método Frame(), que contiene la información codificada de la imagen capturada con las cámaras del dispositivo en cada frame, esto es, una serie de valores que permiten reconocer en todo momento si el usuario ha colocado las manos sobre el mismo. El atributo Hands del objeto frame es un array que permite conocer los parámetros de cada mano, siendo manos[0], la primera mano captada, ya sea la izquierda o la derecha. A su vez, cada elemento de este array tiene otro array con la información de cada uno de los dedos de la mano, llamado Fingers.

Haciendo uso de todos estos datos, es posible saber si por ejemplo el usuario ha colocado una o dos manos, si los dedos están extendidos o no, (o cuántos de ellos), etc.

Una acción a realizar, por ejemplo avanzar, se realizaría llevando una de las manos hacia delante, y se puede gracias al atributo PalmPosition, que se encuentra dentro del primer elemento del array manos, y que detecta la posición en el espacio de la palma de la misma. Si su posición es negativa con respecto al eje Z, quiere decir que el usuario ha alejado la mano con respecto a su cuerpo, con lo que la dirección del desplazamiento será positiva en Z (avanzar). Del mismo modo es posible controlar la posición en otros ejes, como en el X, para determinar que se ha de girar hacia la izquierda, si el valor es negativo, o hacia la derecha si es positivo.

Por último, como acción adicional, se permite el salto comprobando que no hay ningún dedo extendido, y la velocidad del puño es positiva en el eje Y (se ha realizado un movimiento ágil hacia arriba con la mano cerrada).

Adicionalmente, se han aplicado estas funcionalidades al joystick haciendo referencia a los sticks, con la función Input.GetAxis(), para el desplazamiento, e Input.GetButton(), para el salto al pulsar el botón X.

Una vez constituidos el entorno y la interacción con el mismo, se da por concluida la primera parte del proyecto y se comienza a desarrollar la segunda, partiendo de una escena nueva vacía.

Menú Interactivo:

El primer paso es crear una “habitación cerrada” construida a partir de seis planos (objeto Quad) a modo de cubo. Dentro del mismo no se permitirá movimiento alguno, por lo que el usuario será capaz de ver a través de una cámara fija, que

apuntará hacia el centro de la estancia, en el cual se situará el origen del menú circular.

El menú se compone de "paneles", los cuales contendrán cada uno una imagen o un vídeo, formados por cubos con una profundidad ínfima para que parezcan planos, ya que a los objetos de tipo plano no se le pueden aplicar texturas a partir de imágenes o vídeos. También se podrán visualizar en formato de 360º, las cuales se presentarán al usuario mediante esferas en lugar de cubos para diferenciarlas. En el caso de los modelos 3D, se utilizarán miniaturas a escala de los mismos.

Podrá contener tantos elementos como se desee, aunque no es aconsejable incorporar demasiados, ya que llegado un punto, los elementos podrían colisionar entre ellos y, debido al motor de físicas, ejercer fuerzas y hacer que se desplacen y no constituyan una forma circular, que es lo que se pretende. Así, los elementos del menú se contendrán en un array de elementos como parámetro del objeto que representa al mismo. A su vez, se dispondrá de varios arrays que definen el número de elementos de cada tipo de textura.

Para adornar la escena, se situará un objeto en el centro, y que servirá más adelante para identificar cuál es el "panel activo" en todo momento. El concepto de "panel activo" es aquel que determina cuál es el objeto que el usuario tiene delante y con el que puede interactuar. Para cambiar de panel, se deberá girar el menú hacia la derecha o izquierda de uno en uno. Dicho panel tendrá diferentes funciones dependiendo del tipo de textura que contenga, pudiendo ampliarse si se trata de una imagen, reproducirse si es un video, etc.

Como los paneles representan miniaturas o *"thumbnails",* es necesario calcular una escala, de tal modo que las texturas no pierdan su relación de aspecto. La altura de todos ellos será la misma, para dar sensación de simetría a lo largo de todo el menú, con lo que en función de su resolución, el largo del panel puede variar. Se emplearán las funciones x' = x / y, y' = 1, expresado en unidades de medida del motor de Unity. Obtendremos de esta forma los paneles con medidas (x',y'), o lo que es lo mismo, (x',1).

Dependiendo del número de elementos que forman el menú, la separación entre ellos también varía, por lo que se ha de calcular su posición en el espacio de tal forma que queden estéticamente coherentes, así como los grados de rotación que se deben aplicar cada vez que se cambie de elemento. Los paneles deben ser presentados de cara al usuario, siempre mostrados al frente de la cámara, por lo que hay que aplicar dos rotaciones, una con respecto al eje central, y otra respecto al objeto mismo en cuestión para cada uno de ellos.

Al no conocer de antemano su número, los paneles se crearán dinámicamente al inicio de la ejecución y deberán estar contenidos dentro del objeto menú, con el fin

de actuar como una única entidad en la que actúan todos por igual. Para esto, se crearán algunos paneles desde el editor, para dejar constancia de la estructura, aunque estos no se vayan a utilizar en la propia ejecución.

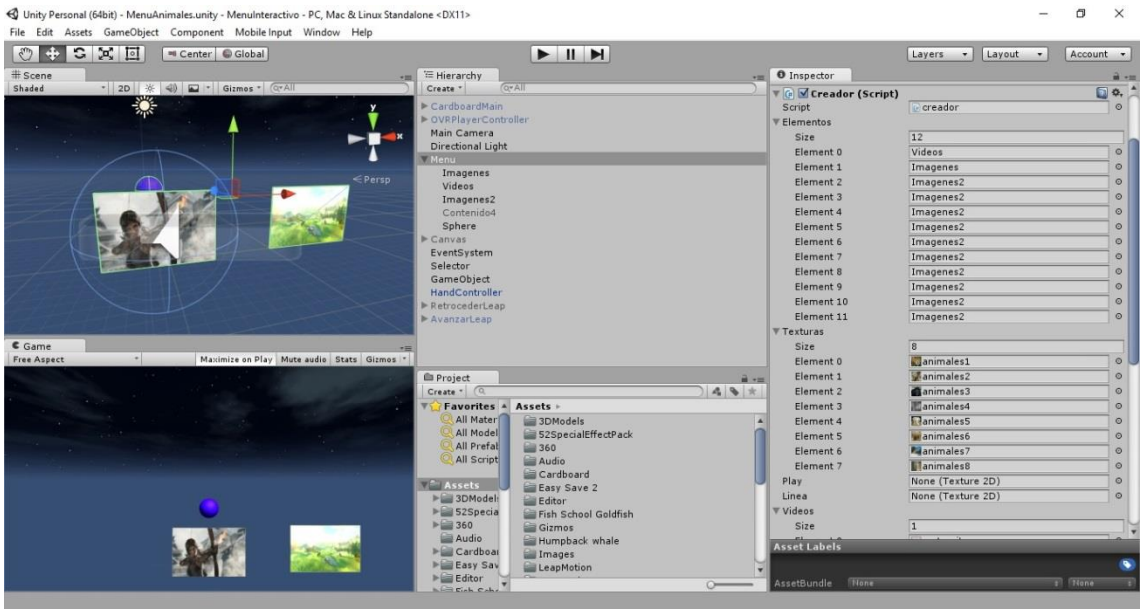


Imagen 27, Creación del menú y de algunos paneles

```
6     public GameObject[] elementos;
7     public Texture2D[] texturas;
8     public Texture2D play;
9     public Texture2D linea;
10    public MovieTexture[] videos;
11    public MovieTexture[] videos360;
12    public Texture2D[] imagenes360;
13    public AudioClip[] audios;
14    public AudioClip[] audios360;
15    public Texture2D[] thumbnails;
16    public GameObject[] prefabs;
17    private float distanciaEntreElementos;
18    private const float radianes = 2 * Mathf.PI;
19    private const float radio = 6.0f;
20    public float gradosRotacion;
21    public float turnSpeed = 50.0f;
22    public float rojo, rojo2;
23    public float verde, verde2;
24    public float azul, azul2;
25    private int tiempoDibujado;
```

Imagen 28, Estructura del menú, script "creador.cs"

Una vez definida la estructura, el siguiente paso es invocar a la función Start(), que generará el menú nada más entrar en la escena. Hay que tener en cuenta que no es necesario inicializar variables o arrays que estén definidos como public, ya que son visibles desde el editor y se permite alterar sus valores desde el mismo.

Lo primero que se debe tener en cuenta es el número de elementos que va a constituir al menú. Esto será útil para conocer los grados de la rotación que se aplicarán en cada momento y se calculan de la siguiente forma:

```
gradosRotacion = radianes / elementos.Length;
```

Un GameObject se puede crear desde código mediante la función Instantiate(), la cual necesita como parámetros: el objeto que se desea instanciar, un vector de posición (x,y,z), y un Quaternion, que es un vector de cuatro componentes y que se utiliza para fijar la rotación de los objetos.

El bucle principal que determina las posiciones de los paneles sobre el perímetro de la circunferencia se define de la siguiente forma:

```
j = 0;
for (i = Mathf.PI; i < radianes + radianes/2; i += gradosRotacion) {
    if (j < elementos.Length) {
        objeto = Instantiate (elementos [j], new Vector3 (radio * Mathf.Sin (i), 0, radio * Mathf.Cos (i)), new Quaternion (0, 180, 0, 0)) as GameObject;
        objeto.transform.parent = GameObject.Find ("Menu").transform;
```

Imagen 29, Colocar paneles como puntos de la circunferencia

De esta forma, calculando los radianes en la definición del bucle, y utilizando las funciones Math.Sin() y Math.Cos(), se obtiene la posición en el espacio de cada elemento del menú, que es un vector de 3 coordenadas donde el valor en el eje X se consigue con la función que calcula el seno, el valor del eje Y siempre es 0 para todos los objetos, y el valor de Z se calcula con el coseno.

El resultado de este pequeño fragmento de código es que se ha construido una circunferencia simulada de tantos puntos como número de elementos del menú, en forma de paneles blancos.

Como cada elemento puede ser de un tipo diferente (una foto, un vídeo,…), es necesario diferenciar unos de otros creando 5 objetos dentro del objeto menú desde el editor. A cada uno de ellos se le asociará tanto un nombre que identifique su tipo, como un tag. Esto se realiza modificando el tag del objeto en cuestión en la ventana de sus propiedades (Inspector). La opción de crear tags se encuentra en el apartado Edit – Proyect Settings – Tags and Layers.

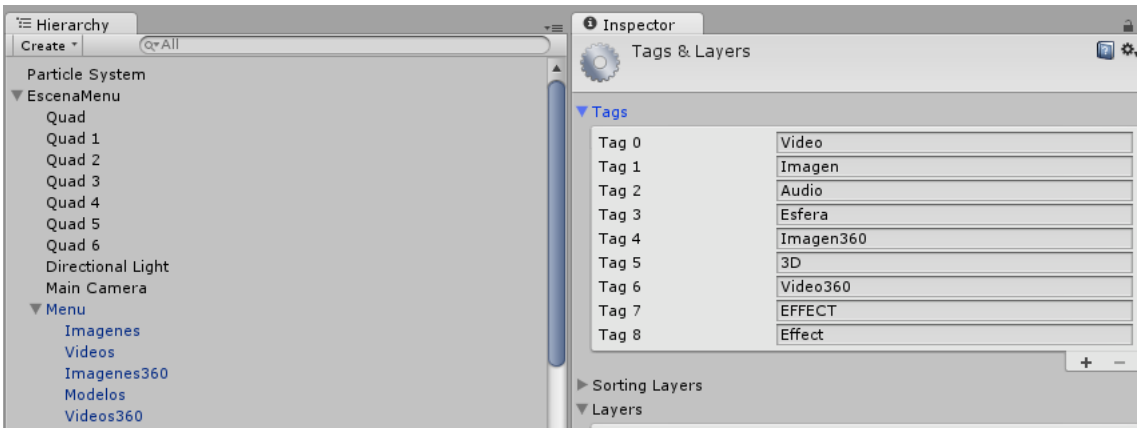


Imagen 30, Creación de objetos y tags asociados

Cada tipo de textura necesita diferente tratamiento, tanto a la hora de su creación, como las acciones que se pueden llevar a cabo con el panel que lo alberga.

- Imágenes:

```
if (objeto.CompareTag ("Imagen")) {
    objeto.GetComponent<Renderer> ().material.mainTexture = texturas [j % texturas.Length];
    //objeto.GetComponent<Renderer>().material.mainTextureScale.Set(1.5f, 1);
    objeto.transform.localScale = new Vector3 ((float)texturas [j % texturas.Length].width / (float)texturas [j % texturas.Length].height, 1, 0.01f);
}
```

Imagen 31, Crear objetos de tipo imagen

- Vídeos:

```
if (objeto.CompareTag ("Video")) {
    objeto.GetComponent<Renderer> ().material.mainTexture = videos [j % videos.Length];
    StartCoroutine (reproducir (objeto, j));
    objeto.transform.localScale = new Vector3 ((float)videos [j % videos.Length].width / (float)videos [j % videos.Length].height, 1, 0.01f);
    if (thumbnails.Length != 0) {
        thumbnail = GameObject.CreatePrimitive (PrimitiveType.Quad);
        thumbnail.transform.position = new Vector3 (objeto.transform.position.x, objeto.transform.position.y, objeto.transform.position.z - 0.01f);
        thumbnail.GetComponent<Renderer> ().material.mainTexture = thumbnails [j % thumbnails.Length];
        thumbnail.transform.localScale = new Vector3 ((float)thumbnails [j % thumbnails.Length].width / (float)thumbnails [j % thumbnails.Length].height, 1, 0.01f);
        thumbnail.transform.parent = objeto.transform;
    }
    botonPlay = GameObject.CreatePrimitive (PrimitiveType.Quad);
    botonPlay.transform.position = new Vector3 (objeto.transform.position.x, objeto.transform.position.y, objeto.transform.position.z - 0.05f);
    botonPlay.transform.localScale = new Vector3 (0.5f, 0.5f, 0.5f);
    botonPlay.GetComponent<Renderer> ().material = new Material (Shader.Find ("Legacy Shaders/Transparent/Bumped Diffuse"));
    botonPlay.GetComponent<Renderer> ().material.mainTexture = play;
    botonPlay.transform.parent = objeto.transform;
    botonPlay.transform.name = "Play";
}
```

Imagen 32, Crear objetos de tipo vídeo

- Modelos 3D:

```
if(cargarModelo.cargado == true)
{
    string informacion = PlayerPrefs.GetString("posicion"+(j+5),"(15,15,15)");
    Debug.Log (objeto.transform.name + "posicion" +j + informacion);
    string[] separador = informacion.Split(',','(',')');
    /*Debug.Log (objeto.name + j + "x" + separador[1]);
    Debug.Log (objeto.name + j + "y" + separador[2]);
    Debug.Log (objeto.name + j + "z" + separador[3]);*/
    objeto.transform.position = new Vector3(float.Parse (separador[1]),float.Parse (separador[2]),float.Parse (separador[3]));
}
```

Imagen 33, Crear objetos de tipo modelo 3D

- Imágenes 360º:

```
if (objeto.CompareTag ("Imagen360")) {
    objeto.GetComponent<Renderer> ().material.mainTexture = imagenes360 [j % imagenes360.Length];
    objeto.transform.localScale = new Vector3 ((float)imagenes360 [j % imagenes360.Length].width / (float)imagenes360 [j % imagenes360.Length].height, 1, 0.01f);
}
```

Imagen 34, Crear objetos de tipo imagen 360º

- Vídeos 360º:

```
if (objeto.CompareTag ("Video360")) {
    objeto.GetComponent<Renderer> ().material.mainTexture = videos360 [j % videos360.Length];
    StartCoroutine (reproducir (objeto, j));
    objeto.transform.localScale = new Vector3 (1, 1, 1);
    botonPlay = GameObject.CreatePrimitive (PrimitiveType.Quad);
    botonPlay.transform.position = new Vector3 (objeto.transform.position.x, objeto.transform.position.y + 0.2f, objeto.transform.position.z - 0.5f);
    botonPlay.transform.localScale = new Vector3 (0.5f, 0.5f, 0.5f);
    botonPlay.GetComponent<Renderer> ().material = new Material (Shader.Find ("Legacy Shaders/Transparent/Bumped Diffuse"));
    botonPlay.GetComponent<Renderer> ().material.mainTexture = play;
    botonPlay.transform.parent = objeto.transform;

}
```

Imagen 35, Crear objetos de tipo vídeo 360º

Para finalizar la confección del menú, es necesario eliminar de la escena aquellos objetos que se han utilizado a modo de plantilla para crear los que de verdad muestran el contenido deseado, por lo que se hará uso de la función SetActive() que permite la activación o desactivación de cualquier objeto en la escena.

```
GameObject.Find ("Imagenes").SetActive (false);
GameObject.Find ("Videos").SetActive (false);
GameObject.Find ("Imagenes360").SetActive (false);
GameObject.Find ("Modelos").SetActive (false);
GameObject.Find ("Videos360").SetActive (false);
if (cargarModelo.cargado == true)
    GameObject.Find ("Canvas").SetActive (false);
```

Imagen 36, Eliminar objetos plantilla

Como toque estético, se ha optado por dibujar la circunferencia que une los paneles utilizando LineRenderer(), que dibuja una línea recta de un punto a otro. De esta forma, dibujando pequeñas líneas entre puntos muy próximos, se consigue el efecto de circunferencia al desplazarse los puntos en los ejes X y Z.

Dando paso a las acciones que el usuario puede realizar con el menú, la primera a implementar será la de cambiar el panel activo en curso. Esto se puede realizar hacia el objeto situado a continuación tanto por la izquierda como por la derecha, en intervalos de uno en uno.

Para dicho fin, se ha de crear dos nuevos scripts, cuyos nombres en este caso son “ControladorLeap.cs” y “selector.cs”. El primero manda una notificación de que se ha realizado un movimiento con la mano sobre Leap Motion (o pulsando la flecha correspondiente en el mando PS3). Si dicho movimiento tiene sentido negativo en el eje X, mandará la orden de avanzar al otro script, que actúa como observador. Si por el contrario, se ejerce un movimiento en sentido positivo, comunicará que la acción a realizar es la de retroceder.

Los observadores se pueden utilizar gracias a la existencia de una clase llamada NotificationCenter, y sus métodos AddObserver() y PostNotification(). Es una forma muy versátil de trabajar, ya que permite la comunicación entre scripts.

Tanto si se pretende girar en un sentido o en el otro, se activará una subrutina que efectuará la operación, con la salvedad de que el parámetro a utilizar en el cálculo de la rotación será diferente y que será obtenido antes de empezar. Este parámetro tomará el valor anterior más uno, en el caso de avanzar, y uno menos en el de retroceder. Como siempre que se entra en la subrutina el valor de este parámetro se incrementa en una unidad, sólo en el caso de retroceder se le restará dos unidades antes, para que al incrementarle una, en realidad se le reste una única unidad.

```
private IEnumerator girar()
{
    var rotacion = Quaternion.AngleAxis(gradosRotacion * pasada, Vector3.up);
    pasada++;
    /*while(true)
    menu.transform.rotation = Quaternion.Slerp(menu.transform.rotation, rotacion, .05f);*/
    float tiempo = 0f;
    float smooth = 100f;
    while (tiempo <= 1f)
    {
        tiempo += Time.deltaTime * smooth/100f;
        menu.transform.rotation = Quaternion.Lerp(menu.transform.rotation, rotacion, tiempo);
        yield return new WaitForEndOfFrame();
    }
    NotificationCenter.DefaultCenter().PostNotification (this, "permitir");
}
```

Imagen 37, Implementación del giro

Cuando el menú se encuentre en reposo, se podrá visualizar su contenido según el tipo al que pertenezca. De esta funcionalidad se encarga el script “visualizar.cs” y deberá estar acoplado a cada uno de los objetos que representan a los paneles. Este script comprueba que el propio objeto es un “panel activo”, para así habilitar su función, de lo contrario, se podrían manipular varios objetos a la vez.

La forma de controlar esto es la de que si el panel se encuentra delante del objeto central en el eje Z, se activa un boolean propio que determina que el objeto es seleccionable. Existe una función llamada Physics.Raycast(), que lanza un "rayo" hacia la dirección indicada en uno de sus parámetros (vector de tres coordenadas), desde un punto preciso (en este caso, la posición del panel). Si este rayo impacta contra el objeto central, haciendo uso de un vector de coordenadas (0, 0, 1), el objeto tendrá la propiedad seleccionable. Obviamente, sólo un objeto puede tener este status.

```
if (Physics.Raycast (this.transform.position, bck, out hit) && hit.collider.tag == "Esfera")
    seleccionable = true;

else
    seleccionable = false;
```

Imagen 38, Comprobación panel activo

Acciones según el tipo de textura:

- Imágenes:

  Se amplía el panel y se coloca progresivamente en el centro de la pantalla.

```
private IEnumerator previsualizar()
{
    timer = 90;
    float tiempo = 0f;
    float smooth = 100f;
    Vector3 nuevaEscala = new Vector3(3*this.transform.localScale.x, 3*this.transform.localScale.y, 0.01f+this.transform.localScale.z);
    if(CompareTag ("Video360"))
       nuevaEscala = new Vector3(2*this.transform.localScale.x, 2*this.transform.localScale.y, 2*this.transform.localScale.z);
    while (tiempo <= 1f)
    {
        tiempo += Time.deltaTime * smooth/100f;
        this.transform.position = Vector3.Lerp(this.transform.position, new Vector3(0, 2, -6.1f), tiempo);
        this.transform.localScale = Vector3.Lerp(this.transform.localScale, nuevaEscala, tiempo);
        yield return new WaitForEndOfFrame();
    }
}
```

Imagen 39, Visualizar Imágenes

- Vídeos:

  Siguen el mismo tratamiento que las imágenes, pero también comienza su reproducción al ampliarse, por lo que después de llamar a la subrutina previsualizar(), se ejecuta el siguiente código:

```
if (this.CompareTag("Video") || this.CompareTag("Video360"))
{
    MovieTexture video;
    video = this.GetComponent<Renderer>().material.mainTexture as MovieTexture;
    video.Stop ();
    video.Play ();
    if(GameObject.Find ("Main Camera"))
        GameObject.Find ("Main Camera").GetComponent<AudioSource>().Pause ();
    if(GameObject.Find ("OVRPlayerController"))
        GameObject.Find ("OVRPlayerController").GetComponent<AudioSource>().Pause ();
    GetComponent<AudioSource>().Play();
    if(transform.FindChild("Quad"))
        transform.FindChild ("Quad").gameObject.SetActive (false);
    transform.FindChild ("Play").gameObject.SetActive (false);
}
```

Imagen 40, Visualizar Vídeos

Además, se utiliza un botón transparente “Play” sobre el panel, para denotar que se trata de un vídeo, y no de una imagen, por lo que una vez se empieza a reproducir, éste debe ocultarse hasta que el vídeo se detenga.

Durante su reproducción, se pueden realizar estas acciones:

- Pause: Situar la mano en la parte izquierda de la pantalla.
- Stop: Mover la mano hacia arriba.
- Volver a reproducir: Desplazar la mano hacia la derecha.

```
if (CompareTag ("Video") && grande && (manos[0].PalmPosition.x < -100 || Input.GetButtonDown("C")))
{
    MovieTexture video;
    video = this.GetComponent<Renderer>().material.mainTexture as MovieTexture;
    if(video.isPlaying)
    {
        video.Pause ();
        GetComponent<AudioSource>().Pause();
    }
}
if (CompareTag ("Video") && grande && (manos[0].PalmPosition.x > 100 || Input.GetButtonDown("T")))
{
    MovieTexture video;
    video = this.GetComponent<Renderer>().material.mainTexture as MovieTexture;
    if(!video.isPlaying)
    {
        video.Play ();
        GetComponent<AudioSource>().Play();
    }
}
if (CompareTag ("Video") && grande && (manos[0].PalmPosition.y > 300))
{
    MovieTexture video;
    video = this.GetComponent<Renderer>().material.mainTexture as MovieTexture;
    video.Stop ();
    GetComponent<AudioSource>().Stop();
}
```

Imagen 41, Manipular Vídeos

- Modelos 3D:

  En este caso, será necesario crear una nueva escena, asegurándose de que posee la misma ambientación que la propia del menú, esto es, la misma estética, colores, etc., para no desentonar. En dicha escena aparecerá

únicamente el modelo seleccionado en cuestión, posicionado en el centro de la misma.

Para que la nueva escena sea capaz de abrir el modelo seleccionado y no cualquier otro, es necesario utilizar un script que actúe únicamente como almacenador de información. No tendrá ningún método ni función, sólo tres variables, dos de ellas con la información de su nombre y tipo, y la última para indicar si ya ha sido cargado o no en la nueva escena.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class cargarModelo : MonoBehaviour {
5
6      public static string nombre;
7      public static string tipo;
8
9      public static bool cargado = false;
10
11     void Start () {
12
13     }
14
15     void Update () {
16
17     }
18 }
19
```

Imagen 42, "cargarModelo.cs"

Ya en la nueva escena, se realiza una búsqueda por nombre entre todos los modelos que contenga el menú y se carga en la misma.

```
public class instanciarPrefab : MonoBehaviour {

    public GameObject[] prefabs;
    GameObject prefab;
    Controller controlador;
    private int escenaMenu;

    void Start () {
        controlador = new Controller ();
        controlador.EnableGesture(Gesture.GestureType.TYPE_CIRCLE);
        controlador.Config.SetFloat("Gesture.Circle.MinRadius", 10.0f);
        controlador.Config.SetFloat("Gesture.Circle.MinArc", .5f);
        controlador.Config.Save();
        escenaMenu = PlayerPrefs.GetInt ("menu");
        Debug.Log (cargarModelo.nombre);
        for (int i = 0; i < prefabs.Length; i++)
        {
            if(prefabs[i].name.Equals (cargarModelo.nombre))
            {
                prefab = Instantiate (prefabs[i]) as GameObject;
                prefab.name = prefabs[i].name;
                prefab.transform.position = new Vector3(0,0,-2);
            }
        }
    }
```

Imagen 43, Búsqueda e instanciación de modelos 3D

El objeto se podrá rotar en los tres ejes a partir de diversos gestos:

- Eje Y: desplazar la mano horizontalmente (mover stick izquierdo en horizontal).
- Eje X: desplazar la mano verticalmente (mover stick izquierdo en vertical).
- Eje Z: dibujar un círculo con el dedo índice (botones L1 y R1).

También se permitirá hacer zoom para ampliar o reducir el objeto, con el puño cerrado y alejándolo o acercándolo a la pantalla (botones L2 y R2).

Para volver a la escena del menú, se debe realizar un desplazamiento con ambas manos hacia abajo sobre el controlador (botón círculo).

```
GestureList gestos = frame.Gestures();
for (int i = 0; i < gestos.Count; i++)
{
    Gesture gesto = gestos[i];
    if(gesto.Type == Gesture.GestureType.TYPE_CIRCLE)
    {
        CircleGesture circleGesture = new CircleGesture(gesto);
        if (circleGesture.Pointable.Direction.AngleTo(circleGesture.Normal) <= Mathf.PI/2)
        {
            if(extendidos == 1)
                prefab.transform.Rotate (0,0,-1f,Space.World);
        }
        else
        {
            if(extendidos == 1)
                prefab.transform.Rotate (0,0,1f,Space.World);
        }
    }

}
if(Input.GetButton ("R1"))
    prefab.transform.Rotate (0,0,-1f,Space.World);
if(Input.GetButton ("L1"))
    prefab.transform.Rotate (0,0,1f,Space.World);

if ((extendidos > 2 && /*manos[0].IsRight && */manos [0].PalmVelocity.y > 100) || Input.GetAxis ("Vertical") < 0 || Input.GetAxis ("LeftStickV") < 0)
    prefab.transform.Rotate (1f, 0, 0, Space.World);
if ((extendidos > 2 && /*manos[0].IsRight && */manos [0].PalmVelocity.y < -100) || Input.GetAxis ("Vertical") > 0 || Input.GetAxis ("LeftStickV") > 0)
    prefab.transform.Rotate (-1f, 0, 0, Space.World);
if ((extendidos > 2 && /*manos[0].IsRight && */manos [0].PalmVelocity.x > 100) || Input.GetAxis ("Horizontal") > 0 || Input.GetAxis ("LeftStickH") > 0)
    prefab.transform.Rotate (0, -1f, 0, Space.World);
if ((extendidos > 2 && /*manos[0].IsRight && */manos [0].PalmVelocity.x < -100) || Input.GetAxis ("Horizontal") < 0 || Input.GetAxis ("LeftStickH") < 0)
    prefab.transform.Rotate (0, 1f, 0, Space.World);
```

Imagen 44, Fragmento de código - manipulación de modelos 3D

- Imágenes 360º:

  La técnica para visualizar elementos en 360º es crear una esfera hueca con dos orificios, uno en la parte superior y otro en la inferior de la misma, y situar una cámara en el interior. Esta esfera se crea en una nueva escena llamada “Visualizador360”.

```
if(CompareTag ("Imagen360"))
{
    cargarModelo.nombre = GetComponent<Renderer>().material.mainTexture.name;
    cargarModelo.tipo = "imagen";
    //Debug.Log (GetComponent<Renderer>().material.mainTexture.name);
    PlayerPrefs.SetInt ("menu", Application.loadedLevel);
    foreach(Transform child in menu.transform)
    {
        if(GameObject.Find (child.transform.name))
        {
            PlayerPrefs.SetString ("posicion"+i,child.transform.position.ToString());
        }
        //cargarModelo.rotaciones[i] = child.transform.localRotation;
        i++;
    }
    PlayerPrefs.Save();
    cargarModelo.cargado = true;
    Application.LoadLevel ("Visualizador360");
}
```

Imagen 45, Cargar Imagen 360º

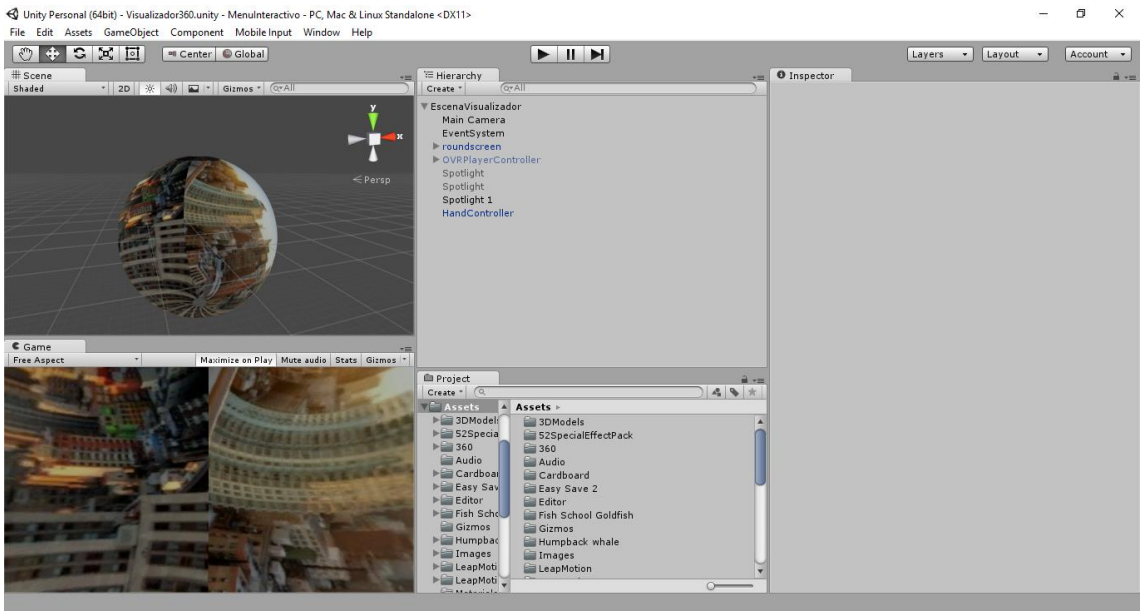


Imagen 46, Escena "Visualizador360"

El usuario toma el control de la cámara situada en el interior de la esfera, pudiendo girar en los ejes X e Y, desplazando la mano hacia el punto al que quiera dirigirse, es decir, la cámara sigue el movimiento de la mano. También es posible hacer zoom en un punto acercando o alejando de la pantalla el puño con los dedos contraídos.

Utilizando el joystick, el movimiento de la cámara se realiza con el stick derecho, y el zoom con los botones L2 y R2.

```
if (manos [0].PalmPosition.y > 200 || Input.GetAxis ("Vertical") < 0 || Input.GetAxis ("LeftStickV") < 0)
    if(this.transform.eulerAngles.x > 340 || this.transform.eulerAngles.x < 22)
        transform.Rotate (-0.2f, 0, 0);
if ((manos [0].PalmPosition.y < 150 && manos [0].PalmPosition.y > 0) || Input.GetAxis ("Vertical") > 0 || Input.GetAxis ("LeftStickV") > 0)
    if(this.transform.eulerAngles.x >= 338 || this.transform.eulerAngles.x < 20)
        transform.Rotate (0.2f, 0, 0);
if (manos [0].PalmPosition.x > 100 || Input.GetAxis ("Horizontal") > 0 || Input.GetAxis ("LeftStickH") > 0)
    transform.Rotate (0, 0.3f, 0, Space.World);
if (manos [0].PalmPosition.x < -100 || Input.GetAxis ("Horizontal") < 0 || Input.GetAxis ("LeftStickH") < 0)
    transform.Rotate (0, -0.3f, 0, Space.World);
if (Input.GetAxis ("RightStick") > 0)
    if(this.GetComponent<Camera>().fieldOfView < 80)
        this.GetComponent<Camera> ().fieldOfView += 1;
if (Input.GetAxis ("RightStick") < 0)
    if(this.GetComponent<Camera>().fieldOfView > 40)
        this.GetComponent<Camera> ().fieldOfView -= 1;
```

Imagen 47, Movimiento cámara 360º

- Vídeos 360º:

  La filosofía es compartida con las imágenes 360º, utilizando la misma escena y con los mismos movimientos de cámara. La única salvedad, es que, al ser un vídeo, se reproducirá nada más entrar en la escena.

  El vídeo no se podrá interrumpir, y si se desea salir de esta escena y volver al menú, se deberá ejercer un movimiento con ambas manos hacia abajo (botón círculo utilizando el mando PS3).

Unión:

Desde la escena del campus es posible acceder a la escena del menú interactuando con terminales situados delante de cada facultad. Cada menú corresponde con el edificio al que pertenece el terminal al que el usuario se ha conectado.

Para cambiar de escena, se utiliza la función Application.LoadLevel() cuyo parámetro es un String que contiene el nombre de la escena a la que se desea acceder.

Al no disponer de material suficiente para todo el campus, se ha optado por desarrollar únicamente el terminal perteneciente a la facultad de informática.

Compilación:

Se han de seleccionar todas las escenas que forman parte de la aplicación, estas son, la escena del campus, la del menú principal, la del visualizador de modelos 3D, y por último, el visualizador de texturas de 360 grados. La plataforma destino será Windows, como aplicación de escritorio.

El orden de selección importa, ya que la aplicación comenzará siempre en la escena cuyo índice sea 0.

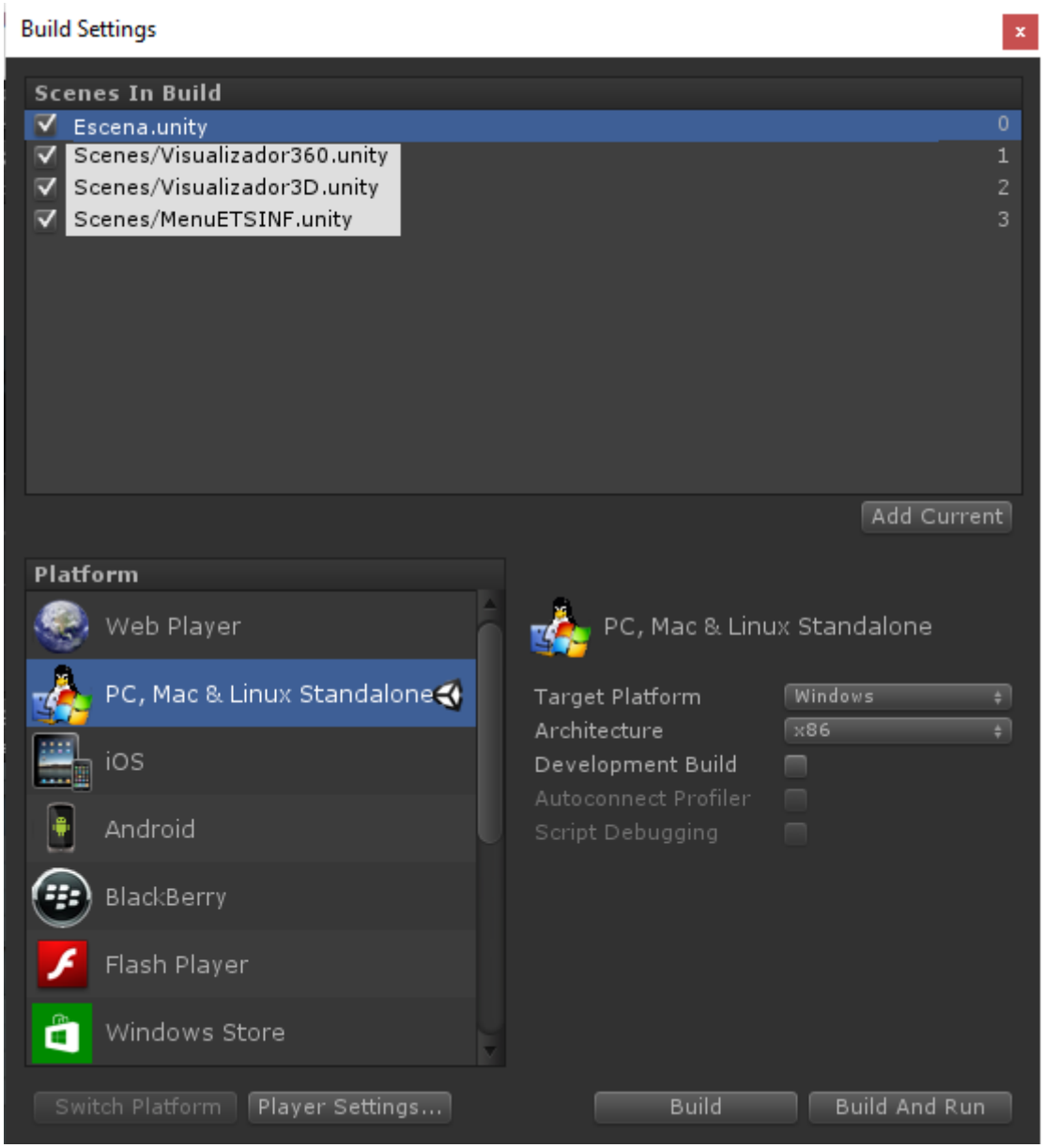


Imagen 48, Compilar

5.3 Ejemplo de Uso:

En este apartado se explican los pasos para lanzar la aplicación, así como su funcionamiento mediante una ejecución real.

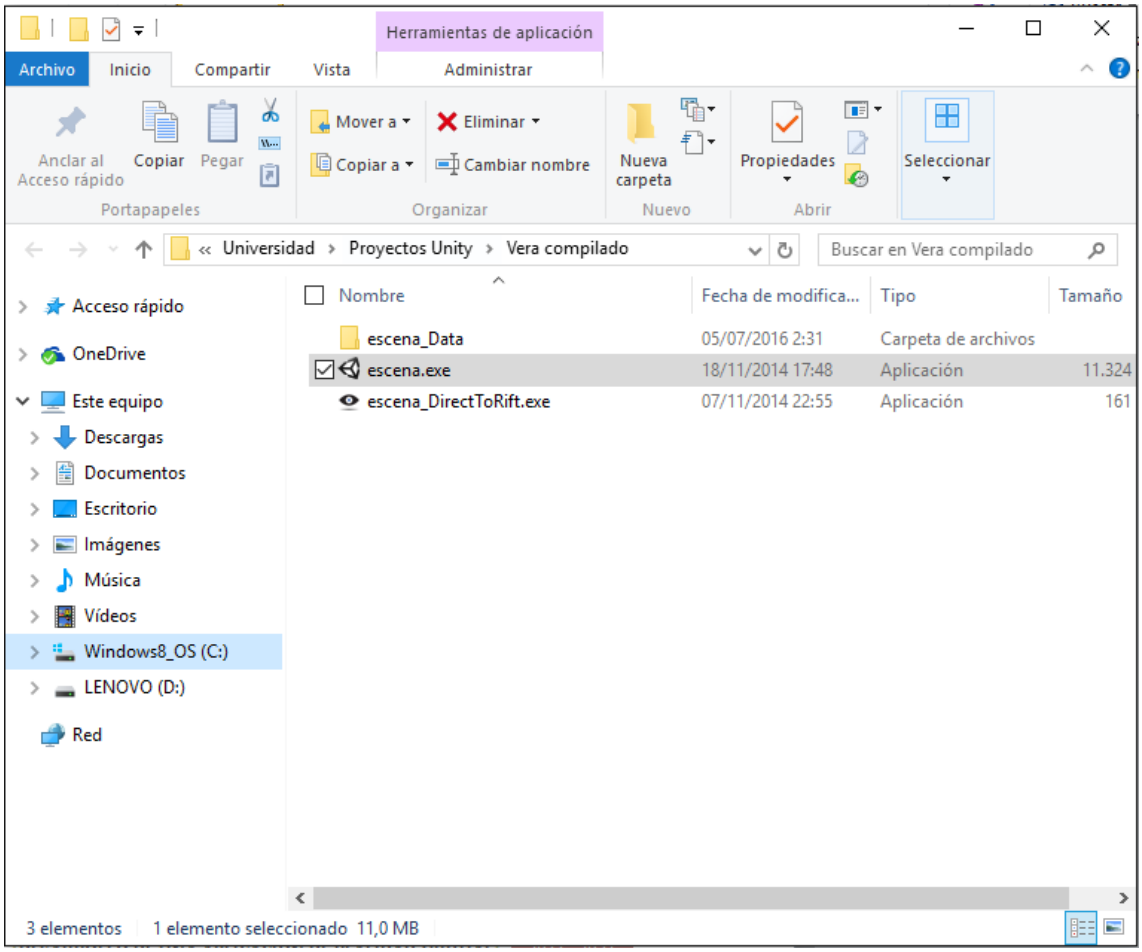


Imagen 49, Ejecutable

Al iniciarse la aplicación, el usuario se encontrará en una parte ya prefijada del campus, donde se le permitirá libre movimiento para explorar el entorno.

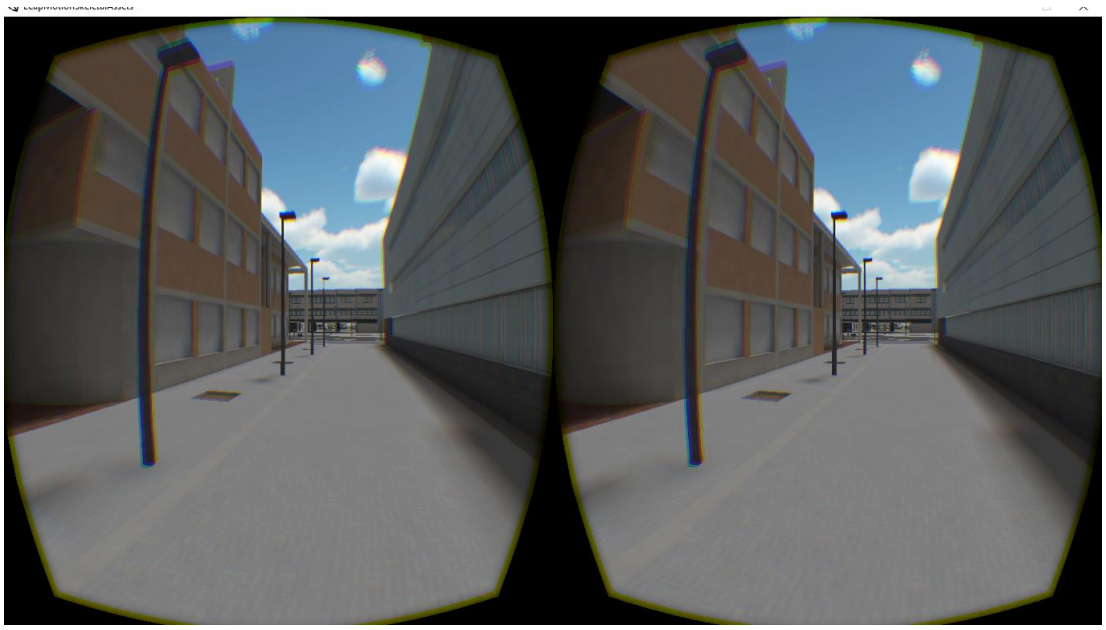


Imagen 50, Inicio

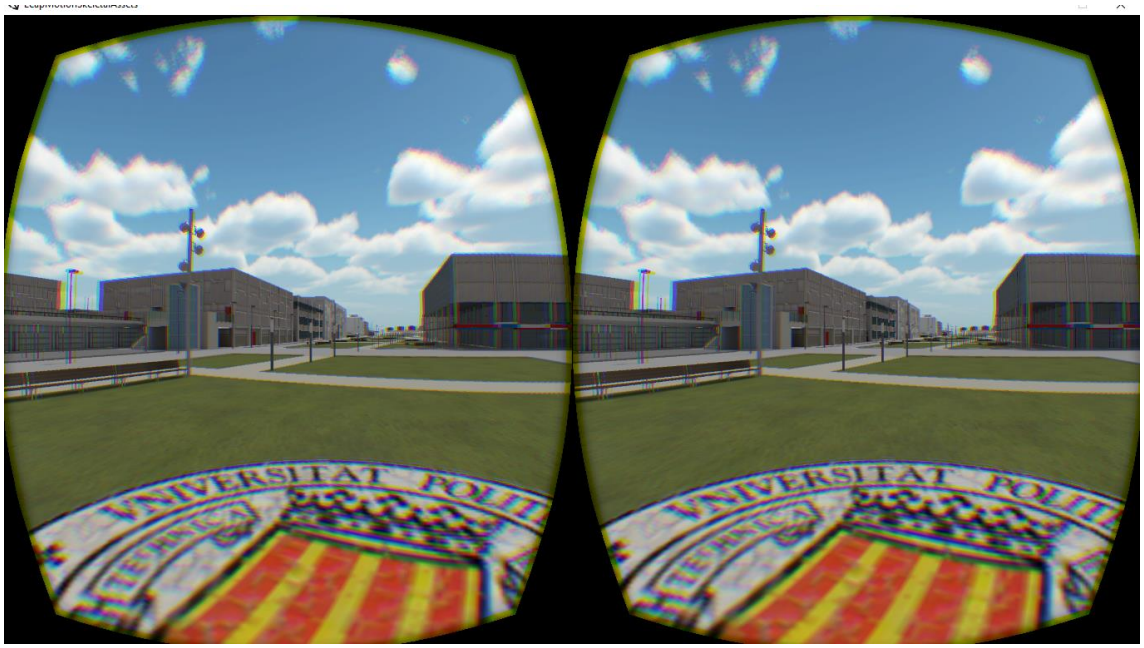


Imagen 51, Vista desde otro punto del campus

El usuario es capaz de buscar los terminales que hay repartidos por la escena y acceder a ellos pulsándolos con la palma de la mano.

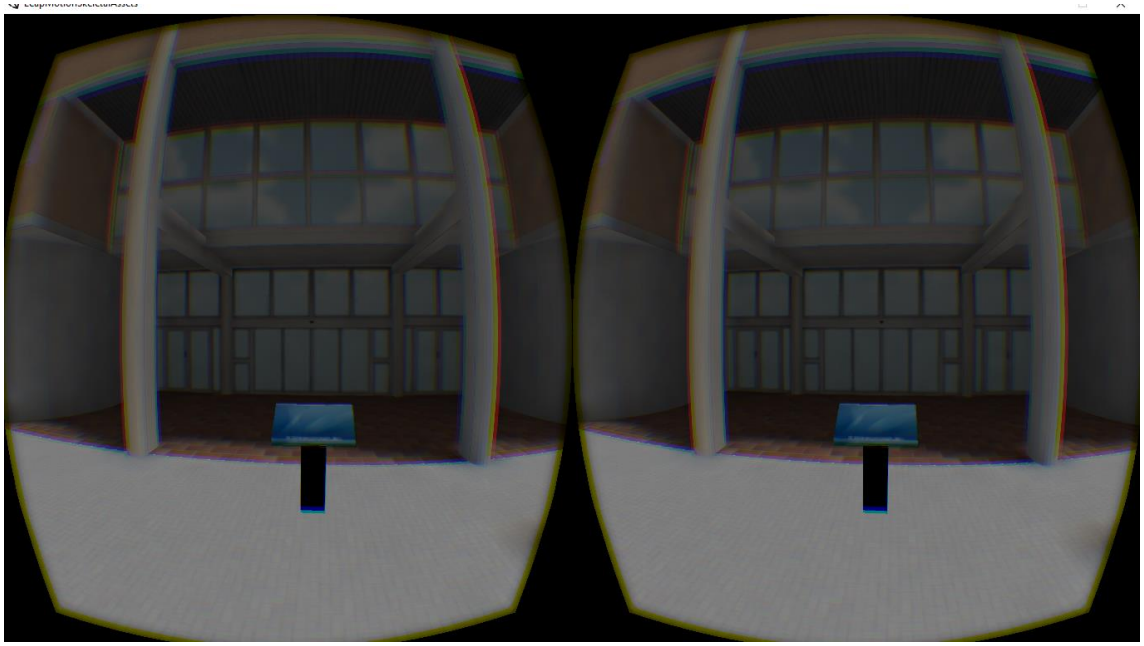


Imagen 52, Terminal de acceso ETSINF

Al interactuar con uno de ellos, se abre la escena del menú que corresponda a cada terminal, en este caso, el del edificio 1G.

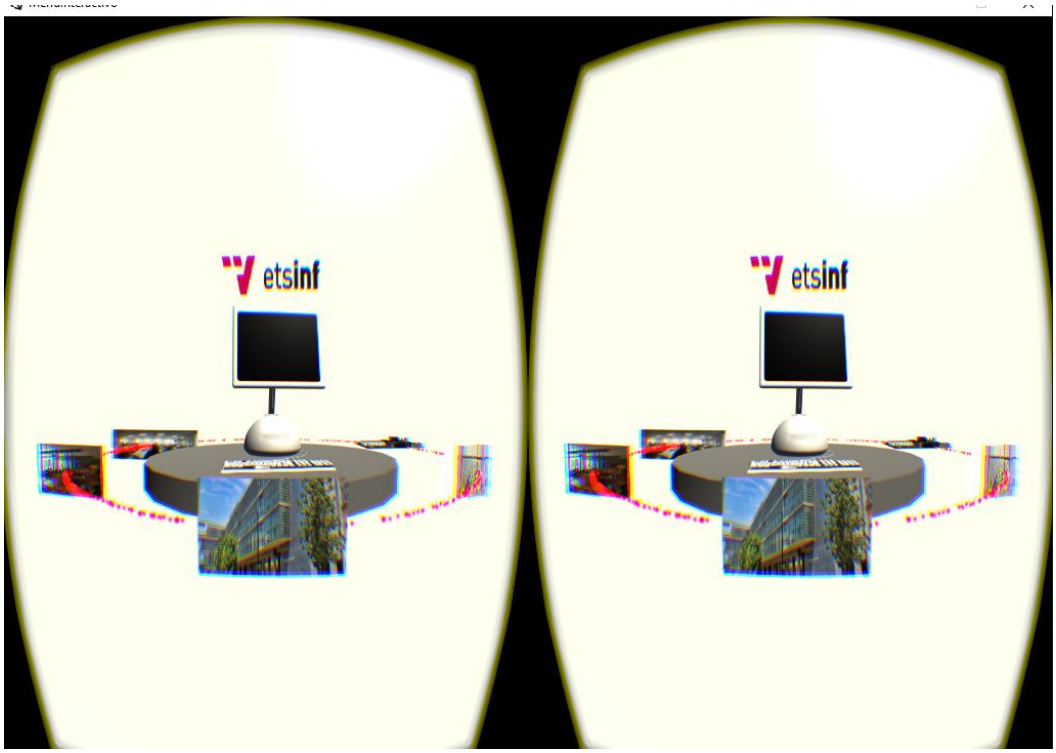


Imagen 53, Menú ETSINF

Este menú se compone de seis elementos (tres imágenes, dos vídeos y un modelo 3D). Todos ellos hacen referencia a información acerca de la facultad en cuestión. Las imágenes corresponden a sus instalaciones, los vídeos a presentaciones tanto del Grado en Ingeniería Informática, como del Máster en Gestión de la Información, y el modelo 3D es una placa base.

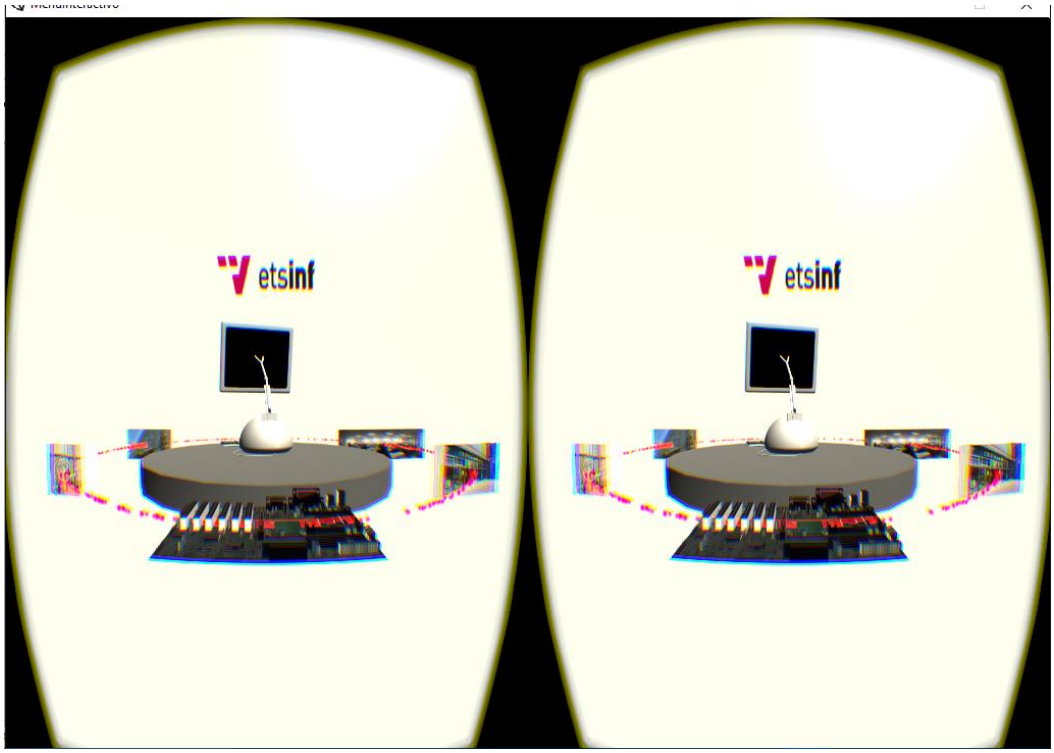


Imagen 54, Cambio de panel activo

Seleccionando como panel activo una de las imágenes, el resultado sería el siguiente:

Imagen 55, Ampliación de imágenes

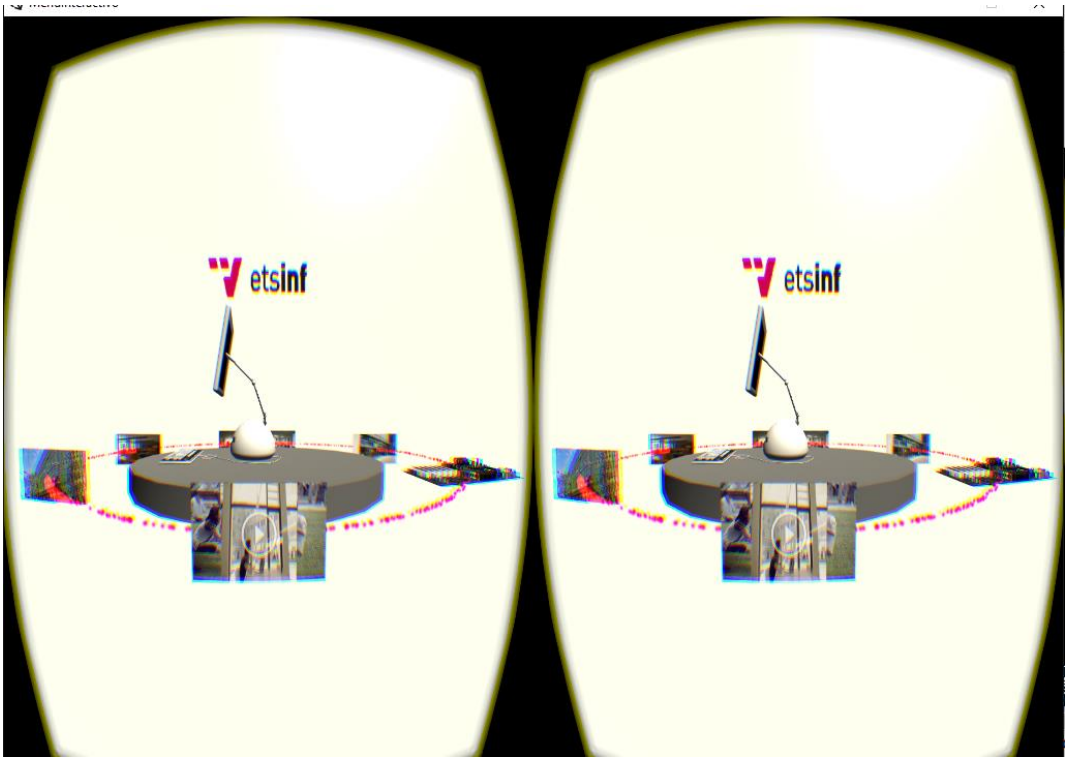


Imagen 56, Selección de vídeos

De igual forma, los vídeos (señalados con un botón de reproducción), se amplían y se reproducen. Una vez ampliados, dicho botón desaparece para no molestar al usuario que lo está viendo.

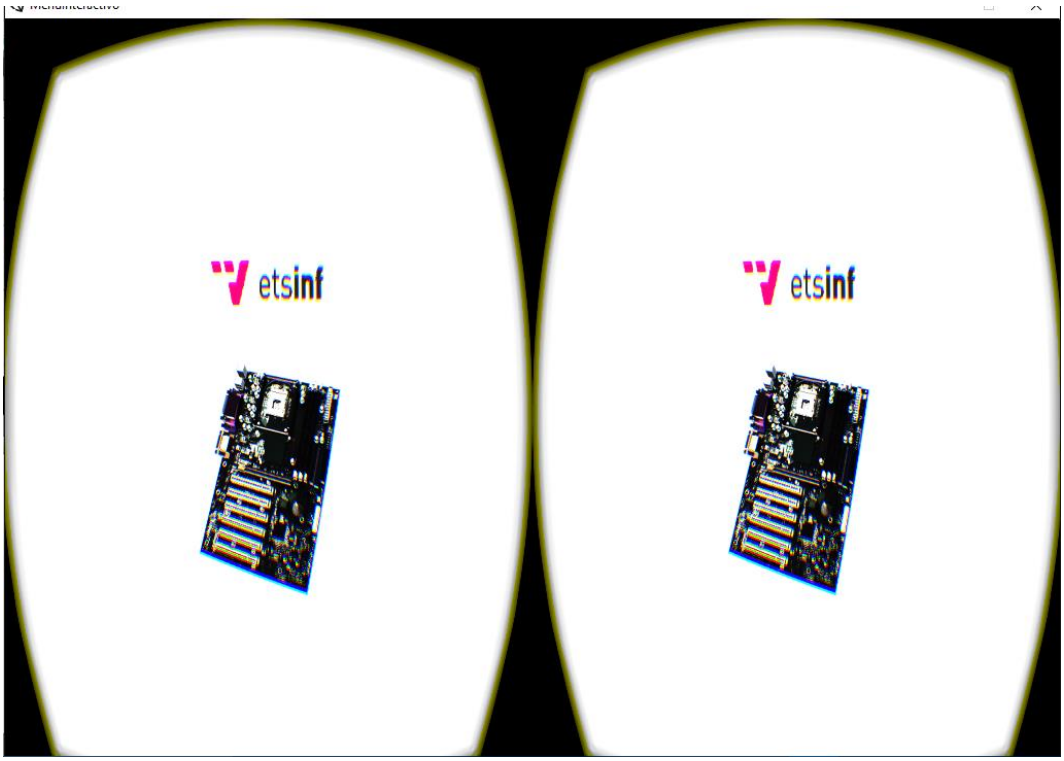


Imagen 57, Visor de modelos 3D

En esta nueva escena, se permite al usuario inspeccionar un modelo 3D desde cualquier ángulo, haciendo posible también acercarlo o alejarlo para ver ciertos detalles, o simplemente para tener una visión más general del objeto.

Por último, las texturas 360º, tanto imágenes como vídeos. Como se comentaba en la parte de desarrollo, se puede observar en la siguiente imagen un agujero en la parte superior de la imagen, ya que la esfera dentro de la que se encuentra la cámara debe estar hueca, por lo que necesita no ser un cuerpo cerrado, sino abierto.

Imagen 58, Imágenes y vídeos 360º

Al no disponer de ninguna imagen o vídeo de estas características sobre la universidad, se han realizado las pruebas de ejecución utilizando otro tipo de contenido, para así constatar su perfecto funcionamiento.

6. CONCLUSIONES

6.1 Trabajo Realizado:

Tras todo el proceso de investigación y desarrollo, se ha conseguido obtener un producto viable que cumple todos los requisitos establecidos al comienzo de este documento, esto es, una aplicación que aporte una experiencia innovadora e inversiva utilizando la Realidad Virtual como medio visual y dispositivos hápticos como medio sensorial. Además, como objetivo extra, se ha creado un visor de archivos que también hace uso de esta tecnología.

Al margen de haberse adaptado para RV, también se ha desarrollado una versión normal de escritorio, para que todo el mundo pueda utilizarla aunque no disponga de medios para adquirir unas gafas de realidad virtual o un controlador háptico, cuya salida de imagen utiliza la pantalla del propio PC, y que también aprovecha el teclado y el ratón.

Queda demostrado que este tipo de herramientas no sirven únicamente para diseñar videojuegos, sino que también se pueden usar en un amplio abanico de campos que no tienen nada que ver con el entretenimiento.

Bien es cierto que al no disponer de una licencia, algunas funcionalidades que ofrece Unity no han podido ser aprovechadas, como un sistema de control de versiones, aunque para este proyecto se ha utilizado GIT, combinado con un repositorio alojado en la web: https://bitbucket.org/.

El problema de este tipo de repositorios, es que no ofrecen una gran cantidad de almacenamiento, por lo que únicamente son útiles para controlar cambios en el código fuente. Es por esto que archivos grandes, como vídeos, imágenes o modelos 3D no se han almacenado en el repositorio.

Tanto la propia “Asset Store” de Unity, como la cantidad ingente de tutoriales que hay en la red, han sido de mucha utilidad para el avance del desarrollo. Es por esto que utilizar este motor ha sido todo un acierto, aunque también haya supuesto un problema en algunas ocasiones, sobre todo con Oculus Rift, que en algunas versiones no son del todo compatibles.

Como prueba de ello, este proyecto comenzó con la versión 4.6 de Unity, y se ha finalizado con la 5.1. Ambas versiones son completamente diferentes, no sólo por la interfaz, sino que también surgieron cambios en la propia API, con lo que se tuvo que reescribir buena parte del código con el desarrollo ya avanzado. El motivo del cambio de versión fue para conseguir una mejor experiencia con Oculus.

6.2 Propuestas de Mejora:

Aunque el producto final como resultado es, en general, muy satisfactorio, muchas de las ideas surgidas desde el inicio hasta el final, tuvieron que ser descartadas, bien por falta de medios, o bien por falta de tiempo.

Algunas de estas ideas son:

- Permitir conectividad online a varias personas utilizando su usuario de Intranet e interactuar entre ellos. Se abandonó la idea al tener que incorporar una base de datos muy grande, a la cual tampoco se tenía acceso, y, aunque podría haberse probado con otra más pequeña, habría complicado mucho el diseño de la aplicación y probablemente la parte del menú se habría visto afectada, o incluso se habría eliminado por completo.
- Plataforma de movimiento para desplazarse en lugar de utilizar Leap Motion o el mando de PlayStation 3. El problema de emplear esta tecnología es que aún no está en el mercado. Cuando se inició el proyecto, no había ninguna versión comercial o que se pudiera probar como desarrollador, por lo que no se pudo llevar a cabo.
- Incorporar al menú un lector de archivos PDF. Esta funcionalidad, combinada con la primera mencionada, supondría que cada alumno pudiera tener acceso a todos los apuntes de sus asignaturas desde la aplicación, siendo más atractivo de utilizar que la actual plataforma “Poliformat”. Obviamente, los profesores también tendrían que tener acceso para poder colgar todos los documentos pertinentes.
- Implementar paneles de información interactivos con el mapa del campus.
- Realizar pedidos online a las cafeterías del campus. Utilizando la misma filosofía que los terminales para acceder a cada menú, se podría colocar un terminal delante de las cafeterías en el cual aparezca una lista de los productos de cada una de ellas y realizar un pedido, o simplemente como medio informativo, para saber cuál es el menú del día.
- Reservar pistas de deporte, utilizando el servicio que ofrece la Intranet.
- Diseño del interior de los edificios, para no limitar el espacio únicamente al exterior.
- Sistema de calendarios para consultar exámenes o acontecimientos importantes que vayan a acontecer en cada facultad. Por ejemplo, conferencias.
- Mostrar titulaciones y planes de estudio de cada una por facultad.

7. BIBLIOGRAFÍA

[1] Historia de la Realidad Virtual:

http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/3D/Realidad%20Virtual/web/historia.html

[2] CAVE:

https://en.wikipedia.org/wiki/Cave_automatic_virtual_environment

[3] Dispositivos Hápticos:

http://dac.escet.urjc.es/rvmaster/rvmaster/asignaturas/mcdh/t2_dispositivos_hapticos.pdf

[4] Estudio de Mercado:

http://www.silicon.es/a-fondo-ventas-realidad-virtual-2016-2305838

[5] Competencia:

http://www.xataka.com/realidad-virtual-aumentada/la-querra-de-la-realidad-virtual-2016-ya-esta-aqui-comparativa-a-fondo-de-todas-las-opciones

[6] Funcionamiento Leap Motion:

http://blog.showleap.com/2015/05/leap-motion-ii-principio-de-funcionamiento/

[7] Motores Gráficos:

https://es.wikipedia.org/wiki/Motor_de_videojuego