View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by UPCommons. Portal del coneixement obert de la UPC

Programación de un entorno virtual interactivo para Oculus Rift

Alumno: Alberto Montemayor García

Tutor: Juan José Fábregas Ruesgas

Titulación: Grado en Multimedia

Fecha: 14/09/2015

Índice

Glosario

Input: Entrada de una tecla definida con tal de nombrarla y configurar su comportamiento al ser presionada o dejada de ser presionada.

RigidBody: Característica de los objetos de Unity que les permite ser afectados por las físicas.

Collider: Elemento sólido en Unity con el que se puede chocar. Los Colliders chocan con otros Colliders.

Trigger: Elemento que detecta cuándo un Collider o un RigidBody entra en contacto con él.

Layer: Capas de interacción de físicas. Permite definir qué elementos afectados por las físicas interaccionan entre sí.

Tags: Nombre que se le puede poner a cualquier objeto del escenario en Unity para definir un grupo específico. Este grupo puede estar formado por tan solo un elemento.

Script: Documento en el cual está escrito el código a ejecutar.

Función: Algoritmo que contiene un segmento del código o su totalidad que puede ser ejecutado en cualquier momento.

Corrutina: Función que se puede parar para luego iniciar desde el punto en el que se había pausado.

Variable: Contenedor de información dentro del script el cual puede almacenar distintos tipos de datos. El tipo de dato a almacenar está limitado al que se defina al declarar la variable y sólo puede almacenar un dato a la vez.

Array: Vector de variables. Capaz de contener distintas variables de un solo tipo.

Boleana: Tipo de variable que puede tener dos valores: "true" (veradero) o "false" (falso).

Entera: Tipo de variable que puede contener números enteros.

Flotante: Tipo de variable que puede contener números decimales.

AudioSource: Tipo de variable en el cual se almacena la salida del audio.

AudioClip: Tipo de variable en el cual se almacena el audio.

Introducción

Este proyecto se ha realizado con la intención principal de experimentar con las nuevas tecnologías que hay a nuestro alcance en el campo de los entornos virtuales y a su vez descubrir un nuevo mundo de posibilidades en la interacción virtual con la Realidad Virtual.

Por otra parte, hemos querido que nuestro proyecto se oriente hacia el ámbito de la e-salud y, más concretamente, hacia los tratamientos de algunos trastornos psicológicos y de la conducta, utilizando tecnologías de RV.

Objetivos

Qué queremos hacer

Conocer el estado del arte en relación con dispositivos de realidad virtual, tanto en lo referente a aspectos tecnológicos como a sus usos especialmente en el ámbito de la e-salud.

Aprender el funcionamiento de las Oculus Rift, y cómo se usan.

Aplicar los conocimientos de "programación informática" aprendidos en los estudios de Grado en Multimedia y ampliarlos en lo que sea necesario para generar una aplicación interactiva que funcione con Oculus Rift.

Diseñar y programar la interactividad de un entorno virtual con el objetivo de provocar emociones en los usuarios, pensando en el uso de estas aplicaciones de realidad virtual en el ámbito de la e-salud.

El objetivo final de este trabajo conjunto, es desarrollar el diseño de un entorno virtual con el objetivo de provocar emociones en los usuarios, pensando en el uso de estas aplicaciones de realidad virtual en el ámbito de la e-salud.

Objetivos personales

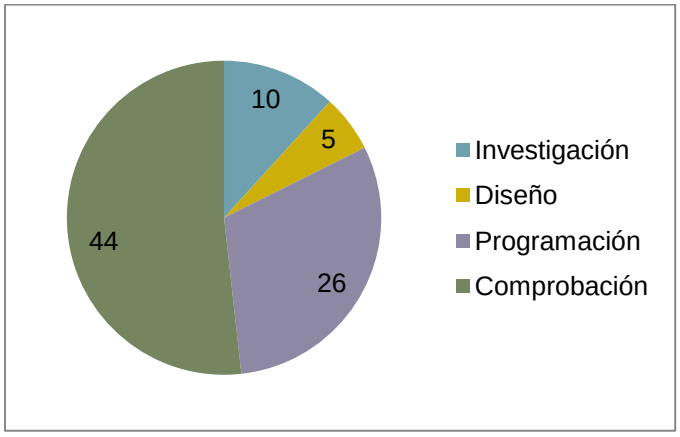
Una vez finalizado este proyecto, espero no solo haber aprendido cómo ha progresado la tecnología de realidad virtual y cómo poder programar para esta tecnología, concretamente Oculus Rift, sino también comprender la gestión y organización de los elementos del proyecto para poder darle un futuro.

A su vez, el poder comunicarme con compañeros (en este caso uno) para así sacar el máximo rendimiento al trabajo en equipo, facilitando tanto su trabajo como el mío, ya que dependo de la organización de sus modelos para luego poder trabajar con ellos de una u otra forma.

Planificación

Inicialmente se planteó un plan de trabajo intensivo bastante optimista en el que se acabaría el proyecto rápidamente siendo conscientes de que podría haber algún problema y que se prolongara un poco.

Este plan de trabajo está seccionado en cuatro etapas importantes que, a su vez se reparten en diversas tareas más pequeñas. Parte de estas etapas son conjuntas junto a mi compañero mientras que otras de ellas son individuales.



En el caso de la **investigación**, es la parte inicial que está dedicada a la obtención de información, tanto por búsqueda y lectura de documentos y páginas web, como haciendo pruebas con diversos usuarios. Una vez obtenida toda esta información se sintetiza para poder documentarla en el proyecto.

El **diseño** es la siguiente parte del proyecto, dedicada a crear los escenarios del entorno virtual por los que el usuario se moverá. Inicialmente haciendo unos bocetos, para luego acabar con el mapa completo.

La **programación** es una etapa que hago individualmente, sin la colaboración directa de mi compañero, en la que me encargo de escribir el código del entorno, tanto los controles, aplicados para Oculus Rift, como la interacción con el entorno, incluyendo la colocación de los elementos del entorno en la escena del proyecto.

La **comprobación** es la etapa final, aunque afecta directamente a la tercera etapa del proyecto ya que es la dedicada a las pruebas con usuarios del proyecto con tal de descubrir defectos y corregirlos.

Debido a que se prolongó más de lo previsto, se inició el 4 grado de carrera sin haberlo acabado, lo que afectó a nuestro ritmo de trabajo considerablemente haciendo que se prolongara más de lo estimado debido a que hubo que dedicar tiempo a los estudios parando, a veces por completo, el proyecto.

Investigación inicial

Previamente a hacer un entorno virtual interactivo para Oculus Rift, he de comprender, por un lado, lo que es la Realidad Virtual, y por otro, el qué son las Oculus Rift.

Además, al pretender darle un posible uso a esta aplicación, he tomado la decisión de enfocarlo a terapias de fobias. Por lo que también he de investigar sobre las fobias y los usos de la Realidad Virtual dentro de la E-salud y más concretamente en el uso del tratamiento las fobias.

Metodología

El desarrollo de proceso de software en sí mismo se puede definir como el trabajo en distintas fases o actividades con la intención de mejorar la planificación y la gestión.

Dentro del proceso de desarrollo de software existen muchas técnicas y sistemas de producción diferentes. En concreto en este trabajo me basaré en el sistema de prototipado con referencias al AUP (Agile Unified Process).

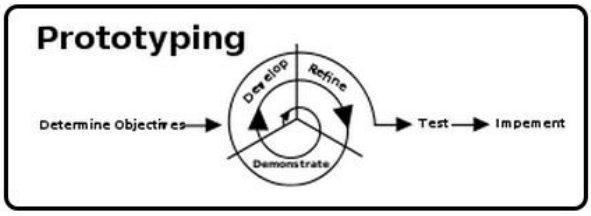


Fig. 1 - Metodología del prototipado

La creación de prototipos de software es el enfoque de las actividades durante el desarrollo de software, la creación de prototipos, es decir, de las versiones incompletas del programa de software que se están desarrollando. El elemento determinante de usar esta metodología es el poder estar constantemente comprobando si hay errores así como solucionándolos, con tal de ir alcanzando los objetivos del prototipo.

De esta forma nuestro proceso de prototipado seguirá la filosofía del sistema de desarrollo de AUP, un sistema consistente en 7 disciplinas (Fig. 2):

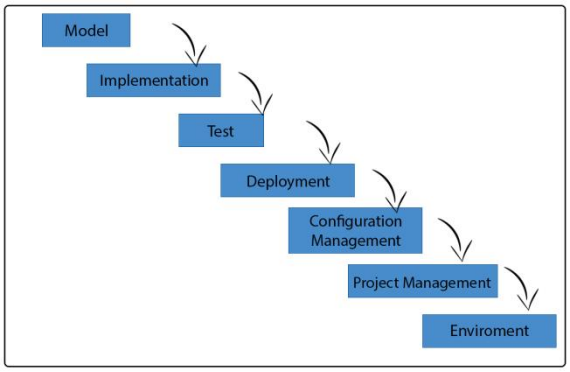


Fig. 2 - Disciplinas del prototipado

- **Modelo:** Entender el negocio de la organización, el dominio del problema que se aborda en el proyecto, e identificar una solución viable para resolver el problema.

- **Implementación:** Transformar el modelo en código ejecutable y realizar un nivel básico de las pruebas.
- **Test:** Llevar a cabo una evaluación objetiva para garantizar la calidad encontrando defectos, validando la funcionalidad del sistema y verificando el cumplimiento de los requisitos.
- **Despliegue:** Planificar la entrega del sistema y ejecutar el plan para que el sistema esté disponible para los usuarios finales.
- **Gestión de la configuración:** Mantener el control y la gestión de las versiones haciendo un seguimiento del proyecto.
- **Gestión de proyectos:** Dirigir las actividades que se realizan dentro del proyecto.
- **Entorno:** Garantizar que se dispone de las guías y herramientas necesarias.

Pese a que en nuestro trabajo conjunto omitiremos tanto el despliegue, la gestión de la configuración y la gestión de proyecto debido a que nos centraremos en desarrollar tan solo un prototipo y no un producto final.

Mundo virtual

Realidad virtual

La Realidad Virtual (RV) es una tecnología a medio camino entre la televisión y los ordenadores que nos permite ver, oír y sentir en un mundo creado gráficamente en tres dimensiones e interactuar con él.

Lo que aporta de nuevo la RV como tecnología es su capacidad de inmersión y de interacción.

- Inmersión porque a través de dispositivos especiales se consigue que la persona tenga la sensación de encontrarse físicamente presente en el mundo virtual.
- Interacción porque la RV no supone una visualización pasiva de la representación gráfica, sino que la persona puede interactuar con el mundo virtual en tiempo real.

Un área de aplicación de la RV en el campo de la salud es su uso como herramienta en el tratamiento de diversos problemas psicológicos. La idea de usar la RV para el tratamiento de estos problemas, se concibió por primera vez en Noviembre de 1992 en el Human-Computer Interaction Group de la Clark Atlanta University y, desde entonces, las aplicaciones se han ido desarrollado rápidamente.

RV frente a RA (Realidad Aumentada):

No confundir RV con RA, que consiste en introducir elementos virtuales en el mundo real. La persona puede observar al mismo tiempo una imagen compuesta por una visualización del mundo real y una serie de elementos virtuales que, están

superpuestos en el mundo real . Este proyecto consiste en inducir al espectador dentro de un entorno virtual.

Dispositivos de Realidad Virtual

Oculus Rift

Oculus Rift es un dispositivo de realidad virtual con un amplio campo de visión hecho con una pantalla que se coloca frente a los ojos como unas gafas. Está siendo desarrollado por Oculus VR.

La primera versión de desarrollador sufría de una baja resolución y de una latencia muy alta, por ello se hizo una segunda versión mejorando tanto la latencia como la resolución, además mejoraron el diseño estético y le añadieron el rastreo posicional. Aquí la diferencia de características:

	DK1	DK2
Resolución de pantalla	1280x800	1920x1080
OLED	NO	SÍ
Tamaño de pantalla	7"	5,7"
Latencia	50 - 60 ms	20 - 40 ms
Persistencia	~ 3 ms	2, 3 ms, full
Frecuencia de muestreo	60 Hz	60, 72, 75 Hz
Rastreo de orientación	SÍ	
Rastreo posicional	NO	SÍ
Sensores	Giroscopio, acelerómetro, magnetómetro	
FOV	110º	100º
3D	Estereoscópico	
Peso	380 gr	440 gr

Fig. 3 - Oculus DK1

Fig. 4 - Odulus DK2

Actualmente se ha anunciado una nueva versión: las CV1.

Estas nuevas Oculus tienen una resolución superior respecto a las anteriores de 2160x1200 lograda usando dos pantallas en lugar de una como se hacía anteriormente, la frecuencia de muestreo de ambas pantallas es de 90 Hz.

Fig. 5 - Oculus CV1 (Vista frontal)

Se ha reducido el peso a unos 200 - 300 gramos con un diseño mucho más ergonómico y se han mejorado las lentes para quitar las aberraciones cromáticas. También se ha reducido la latencia.

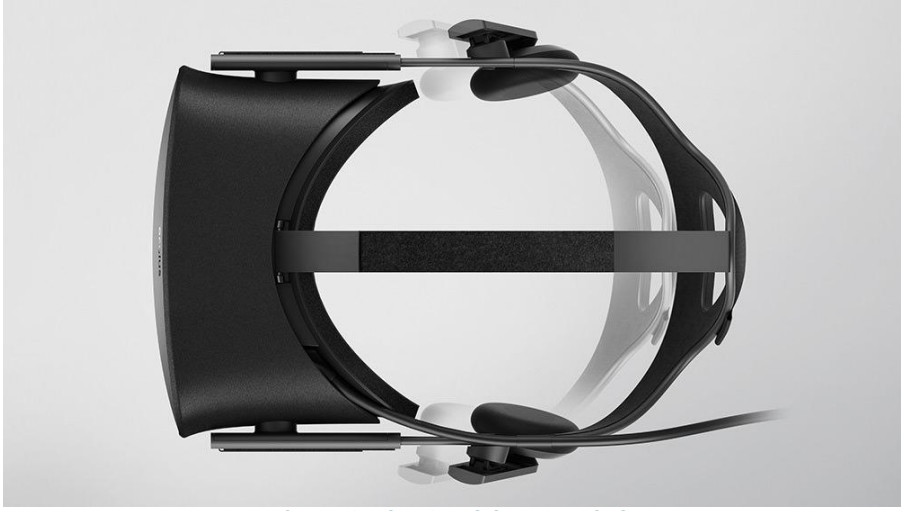


Fig. 6 - Oculus CV1 (Vista superior)

Se ha aumentado la precisión del rastreo de la cabeza utilizando un nuevo sensor que rastrea unos LEDs infrarrojos de las Oculus.

Fig. 7 - Sensor Oculus CV1

Por último, se han incorporado unos auriculares con un sistema de audio de realidad virtual permitiendo la sensación de espacio y profundidad.

Fig. 8 - Auriculares Oculus CV1

A todo esto, también han creado un dispositivo para utilizar tanto el Samsung GALAXY Note 4 como el S6 o el S6 edge como pantalla de Oculus, el Samsung Gear VR.

Fig. 9 - Gear VR

Los Oculus Touch son los mandos de control creados especialmente para el uso junto con las Oculus.

Estos mandos, aparte de tener los controles básicos que tienen actualmente todos los mandos, tienen una función de seguimiento de forma que, puedes mover los mandos y mover elementos dentro del entorno, aumentando así la capacidad interacción con el entorno y a su vez la sensación de integración dentro del mundo virtual.

Fig. 10 - Oculus Touch

Sensor Kinect

El sensor Kinect es un dispositivo físico con la tecnología de sensor de profundidad, una cámara integrada de color, un (IR) emisor de infrarrojos y un conjunto de micrófonos, lo que le permite detectar la ubicación y los movimientos de las personas, así como sus voces.

Fig. 11 - Sensor Kinect

Esto le permite ofrecer una mayor precisión en general, capacidad de respuesta, y las capacidades intuitivas para acelerar el desarrollo de aplicaciones que responden al movimiento, el gesto y la voz. La cámara de color del sensor tiene una resolución de 1080p que se pueden mostrar en la misma resolución que la pantalla. Además es capaz de realizar el seguimiento de hasta seis personas y 25 articulaciones del cuerpo por persona, las posiciones de orugas son más anatómicamente correcta y estable, y el rango de seguimiento es más amplio. La mayor fidelidad de la profundidad hace que sea mucho más fácil ver objetos más pequeños, para ver todos los objetos con mayor claridad, y para ver los objetos en tres dimensiones (3D).

Virtuix Omni

Virtuix Omni es un dispositivo de control, tal como sería un mando, con la diferencia que éste se controla moviendo el cuerpo.

Fig. 12 - Virtuix Omni

Está formado por unas placas de baja fricción con ranuras para aumentar la estabilidad al desplazarse que, junto a unos zapatos especiales, permiten desplazar los pies.

Fig. 13 - Zapatos Virtuix Omni

Mediante los sensores que hay colocados en los zapatos que detecten el desplazamiento para reproducirlo en la aplicación, ya sea caminar, correr, andar hacia atrás o incluso saltar.

Fig. 14 - Virtuix Omni Tracking PODs

Éste dispositivo está adaptado para combinarse junto con el Kinect para detectar gestos y así poder detectar y reproducir la máxima cantidad de movimientos posibles.

La combinación de estos tres dispositivos (Oculus, Kinect y Omni) permite generar una integración en el entorno virtual casi completa.

Project Morpheus

Playstation está creando también sus propias gafas de realidad virtual llamadas **Project Morpheus**.

Fig. 15 - Project Morpheus

Estas gafas de realidad virtual aún en están en proceso pero ya han mostrado algunas de sus características demostrando que también poseen un gran potencial.

Utiliza una pantalla OLED curva de 1920x1080 píxeles lo que hace que cada ojo tenga una resolución de 960x1080 píxeles con una velocidad de refresco de 120 Hz. La latencia de la pantalla es de 18 ms y gracias a 9 LEDs distribuidos por el dispositivo tiene un seguimiento posicional muy preciso.

Una característica a destacar es que se puede modificar la distancia focal de las lentes para poder ajustarlo a cualquier vista.

E-salut

Algunas definiciones sitúan a la e-salud (también denominada e-Health), en términos reciente como “una referencia para la práctica de la Salud, la cual, es apoyada por procesos electrónicos y de comunicación”. Otros sostienen que la e-Salud es “la aplicación de las Tecnologías de la información y Comunicaciones en el amplio rango de aspectos que afectan el cuidado de la salud, desde el diagnóstico hasta el seguimiento de pacientes, pasando por la gestión de las organizaciones implicadas en estas actividades”. En nuestro caso la relacionamos con el uso de nuevas tecnologías como la realidad virtual en el campo de la salud.

Fobias y tratamiento en RV

Fobias:

Las fobias se basan en el miedo excesivo e irracional a una situación determinada o a un objeto en concreto. Hay tres niveles de fobias:

- Fobia específica (miedo a las alturas, a los túneles, a volar a las arañas, a los perros…)
- Fobia Social
- Agorafobia

A pesar de que los pacientes de fobias son plenamente conscientes de que estos miedos son totalmente irracionales, a menudo se encuentran con que afrontar, o incluso solo pensar en tener que enfrentarse al objeto de sus miedos, lleva asociado un estado de ansiedad muy intensa e incluso la posibilidad de vivir un ataque de pánico. En la mayoría de ocasiones, las personas que sufren una fobia no buscan tratamiento para superarla, ya que lo que hacen es adaptar su vida evitando el contacto con los objetos, animales o situaciones que les puedo provocar dicha ansiedad. En general, no suelen querer realizar terapia porque piensan que el tratamiento les someterá a altos grados de ansiedad, sin embargo, las terapias actuales suelen estar resueltas en pocas sesiones y con unos niveles de estrés bajos.

Tratamientos en Realidad Virtual

Un área de aplicación de la RV en el campo de la e-salud es su uso como herramienta en el tratamiento de diversos problemas psicológicos.

La idea de usar la RV para el tratamiento de estos problemas, se concibió por primera vez en Noviembre de 1992 en el Human-Computer Interaction Group de la Clark Atlanta University y, desde entonces, las aplicaciones se han ido desarrollado rápidamente.

En 2002 se publicó en la revista IEEE Transactions on Information Technology in Biomedicine, Vol. 6, No. 3, el artículo titulado “The Development of Virtual Reality Therapy (VRT) System for the Treatment of Acrophobia and Therapeutic Case”, cuyos autores explicaban que la realidad virtual se estaba usando para el tratamiento del vértigo, pero planteaban algunas limitaciones relacionadas con el uso de esta tecnología, concretamente el coste elevado de los dispositivos

necesarios y el hecho de que los escenarios virtuales eran poco realistas. Su trabajo consistió en generar un entorno virtual más realista, consistente en un ascensor abierto pero con protecciones - barandas anti caída-, que subía a una torre. El escenario virtual incluye un ascensor abierto rodeado de apoyos al lado de una torre. Tal y como ellos dicen en su artículo, los resultados del tratamiento a una persona con acrofobia (miedo a las alturas), con este entorno de realidad virtual más realista, demostraron que este ambiente VR fue eficaz para la superación de la acrofobia de acuerdo no sólo a los resultados de la comparación de una serie de cuestionarios antes y después del tratamiento, sino también a los comentarios del sujeto que comento que el escenario virtual parecía evocar sentimientos más terribles que la situación real.

Desde 2002 hasta la actualidad la tecnología relacionada con la realidad virtual ha evolucionado mucho y los costes de los dispositivos han bajado lo suficiente como para que su uso pueda resultar incluso más económico que la exposición a entornos reales.

En 2011, apareció publicado en la revista Investigación y Ciencia, el artículo titulado “La realidad virtual en psicoterapia”, en cuya presentación se podía leer lo siguiente:

> El tratamiento con realidad virtual aprovecha la ilusión de presencia y la interacción para tratar a pacientes con trastornos psíquicos diversos, entre ellos, fobias, psicopatologías alimentarias, ansiedades, déficits de atención, adicciones e incluso el dolor crónico. Aunque las experiencias llevadas a cabo hasta ahora muestran la eficacia del método, su uso en clínica resulta todavía escaso. Aun así, según los expertos, falta poco para el cambio.

En junio de 2015, la revista Journal of Studies on Alcohol and Drugs, publicó el artículo titulado “Virtual Reality Therapy for the Treatment of Alcohol Dependence: A Preliminary Investigation with Positron Emission Tomography/Computerized Tomography”. Tal y como explican sus autores, en estos momentos las terapias realizadas actualmente con realidad virtual ya no se limitan a la estimulación visual y auditiva, sino que pueden incluir incluso la estimulación olfatoria y la gustativa, lo cual ha permitido plantearse el tratamiento de trastornos tipo adicción tales como el alcoholismo. Los autores del estudio evaluaron los resultados de la terapia comparando los cambios producidos en el metabolismo del cerebro de un grupo de pacientes tratados con tecnología de realidad virtual y otro grupo similar que no fue tratado. Como resultado de su trabajo y de los resultados positivos que obtuvieron, los autores recomiendan utilizar *tentativamente VRT para tratar el TDA a través de su efecto regulador en circuitos límbico*.

Aunque están documentados y publicados muchos otros casos que se pueden considerar como ejemplos del uso de la tecnología de Realidad Virtual para el tratamiento de trastornos psicológicos y de la conducta y, de su rápida evolución durante los últimos años, consideramos que con los tres casos expuestos se puede obtener una idea bastante precisa no sólo del importante salto producido en muy poco tiempo, sino de la clara tendencia al uso creciente de estas tecnologías en el ámbito de la salud psicológica.

Una demostración de lo afirmado en el párrafo anterior sobre la normalización y generalización del uso de tecnologías de RV en el tratamiento de trastornos psicológicos y de la conducta, la podemos buscando en google incluyendo las palabras realidad virtual psicoterapia (o terapia). Nos aparecerán, por ejemlo, vínculos a servicios como VirtualRet que ofrece “Terapia de Realidad Virtual” y explica en su página web cómo utilizan la RV en las terapias de diferentes trastornos, cuales son las ventajas, etc.

Por otra parte, la investigación científica respecto al uso de Entornos de Realidad Virtual en Terapias (VRET), es constante y se centra en mejorar los procesos o en revisar hipótesis, procedimientos, etc. En este sentido queremos citar, por ejemplo, el capítulo titulado “Virtual Reality Exposure Therapy for Anxiety and Specific Phobias”, del profesor Dr. Thomas D. Parsons, en el que se hace una revisión de investigaciones y avances desarrollados y se hacen afirmaciones como las siguientes que muestran el seguimiento, atención y seriedad científica implícita.

> Aunque los investigadores en Cyberpsychology sostienen que la VRET ha demostrado ser eficaz, un problema potencial en la interpretación y la conciliación de los resultados sobre la naturaleza y el alcance de los cambios afectivos resultantes de VRET es que la gran mayoría de los estudios han informado sobre muestras pequeñas y hecho uso de pruebas de significación de hipótesis nula inadecuadas. Hasta que se publiquen estudios a gran escala sobre los efectos afectivos de VRET, los meta-análisis estadísticos representan un remedio provisional.

Desde que surgió toda esta nueva tecnología ha habido múltiples casos en los que se ha intentado simular sensaciones como el vértigo o incluso hacer creer al usuario que realmente está cayendo desde miles de metros de altura con un paracaídas, tan solo con un dispositivo Oculus Rift, un captador de movimiento como Kinect y algunos segmentos de madera nos pueden hacer creer que realmente estamos en la cornisa de una pared, o en lo alto de un muro, sino unas Oculus, un ventilador un arnés y un árbol lo suficientemente alto y resistente nos pueden llevar a las alturas sin necesidad de paracaídas.

Todas estas aventuras pueden resultar emocionantes y muy entretenidas sin lugar a duda, pero al igual que desde que surgió esta tecnología, comenzaron a aparecer múltiples aplicaciones similares, desde un principio también se encaminó el uso de este dispositivo hacia el mundo de la e-salud. Un uso dentro de este cambio podría ser la superación de fobias.

En este caso, se ha enfocado en dos fobias en concreto: la nictofobia y la acrofobia.

Nictofobia:

La nictofobia (proveniente del griegos *nyx*: noche y *fobos*: miedo) es una fobia caracterizada por un miedo irracional a la noche o a la oscuridad. El nombre de la enfermedad hace alusión a Fobos el dios del miedo y a Nix, la diosa de la noche.

Es generada por una percepción distorsionada del cerebro de lo que podría pasar en medio de la oscuridad. También se le conoce como escotofobia, acluofobia, ligofobia, mictofobia o sencillamente miedo a la oscuridad.

Aunque es muy común en todo el mundo que los individuos puedan desarrollar un miedo excesivo a la oscuridad también es cierto que se ha investigado poco sobre esta patología. La nictofobia además es un mal solamente relacionado con niños, pero según explica J. Adrian Williams en su artículo *"Indirect Hypnotic Therapy of Nyctophobia: A Case Reports"*, es muy probable que los niños que hayan sufrido de un miedo a la oscuridad excesivo también puedan desarrollarla cuando adultos. Además en el mismo artículo Williams expone también que puede ser muy perjudicial la nictofobia tanto en adultos, como en personas discapacitadas.

Según especialistas el miedo a la oscuridad es común y normal entre los niños de 2 a 7 años. Aunque esto podría variar en un rango de un par de años.

Acrofobia:

Se denomina acrofobia (del griego ἄκρος alto, elevado y φόβος miedo) al miedo a las alturas. Por ejemplo no atreverse a practicar deportes extremos o de alturas, como lo serían la tirolesa, el paracaídas o el parapente. Al igual que otras fobias, la acrofobia genera fuertes niveles de ansiedad en los individuos que la presentan, lo que induce una conducta de evitación de la situación temida. En este caso, las situaciones con una altura notable, como asomarse a un balcón, encontrarse al borde de un precipicio o estar en un mirador elevado, son típicas de este tipo de fobia.

Ventajas de los tratamientos en RV:

En comparación con los tratamientos "tradicionales", la RV presenta interesantes ventajas.

Frente a la exposición en imaginación:

- La RV es más inmersiva, ya que se estimulan varias modalidades sensoriales (auditivas, visuales y vestibulares), lo cual resulta muy conveniente para aquellas personas que tienen problemas para imaginar.
- La RV permite al terapeuta saber en cada momento lo que el paciente está viendo, y por tanto puede saber con mayor precisión qué estímulo está provocando la respuesta de miedo.

Frente a la exposición in vivo:

- La RV permite ofrecer exposición a aquellas personas que se niegan a someterse a este tipo de técnica porque les resulta demasiado difícil o amenazador.
- Ofrece un mayor grado de confidencialidad, en el sentido de que el tratamiento se hace en consulta, por lo que la persona no tiene porqué temer que si la exposición se realiza en un ambiente público, los demás puedan conocer su problema.
- Permite diseñar a medida la jerarquía de exposición, con lo que la persona puede exponerse a prácticamente todas las situaciones posibles.
- Es segura, en tanto que la persona (y el terapeuta) controlan en todo momento lo que ocurre en el ambiente virtual, cosa que en muchas ocasiones depende del azar en las exposiciones en vivo.
- Se puede repetir la exposición a una situación todas las veces que sea necesario hasta conseguir que la ansiedad baje.
- Puede resultar bastante más barata, ya que la exposición se hace en la propia consulta, lo que la convierte en más rentable en términos de tiempo y dinero (piénsese, por ejemplo en la fobia a volar).

La RV permite al terapeuta contar con un ambiente protegido que permite al paciente:

- Conocer una situación que siempre ha considerado como amenazadora
- Hacerlo en la medida que él quiera y a su ritmo
- Reexperimentar muchas veces las implicaciones y consecuencias de su interacción con el ambiente que teme
- Ir más allá de la realidad, puesto que se puede ofrecer al paciente un entorno incluso más amenazador de lo que nos podemos encontrar en la realidad.

Por último, la RV permite ir más allá de la realidad:

- Posibilita que el contexto temido cambie a nuestra conveniencia.
- Se pueden diseñar una serie de contextos en los que el paciente puede afrontar virtualmente, no sólo lo que teme, sino distintos aspectos mucho más amenazadores (p.ej., hacer que la pared de una habitación se desplace haciendo que sea mucho más pequeña).
- La meta de la RV no tiene por qué ser "re-crear" la "realidad". Lo esencial es delimitar contextos que resulten significativos para el paciente, esto es, crear condiciones o situaciones a los que la persona, por el momento, o no tiene acceso o lo ha perdido.

- La RV se convierte así en un paso intermedio muy útil entre la consulta y el mundo real. Y no hará falta esperar a que se produzcan los acontecimientos en el mundo real, ampliándose las posibilidades de auto-entrenamiento.

Proyectos en común

Previamente a nuestro proyecto, hay otras personas que se les han ocurridos ideas de usar la realidad virtual semejante a la nuestra.

Phobos

Phobos es un proyecto indie realizado por un grupo pequeño de personas que, al igual que nosotros, pretendía generar un entorno virtual que permitiera superar diversas fobias, entre ellas la acrofobia.

El proyecto en cuestión se inició tratando la acrofobia (miedo a la altura), la aerofobia (miedo a volar), claustrofobia, distintas fobias a los animales y la agorafobia con la intención de aumentar el número de fobias a tratar, pero debido a la falta de de dinero no pudieron progresar.

Aún así, era un proyecto muy prometedor que, al haber podido continuar, hubiera resultado de mucha ayuda para realizar terapias.

Psious

Psious es una herramienta en la que se está trabajando para realizar el tratamiento de fobias específicas mediante unas gafas de realidad virtual que, combinadas a una aplicación, pueden monitorizar al paciente, comunicarse con él y tomar apuntes, además te permite controlar en todo momento lo que sucede dentro del entorno y cuantificar el nivel de ansiedad del usuario. Actualmente han realizado pruebas con encefalogramas para obtener los datos proporcionados por el electroencefalograma durante el periodo de del tratamiento, aunque aún no han perfeccionado este punto ya que hay algún problema si el usuario se mueve rápidamente.

Fig. 16 - Cómo funciona Psious

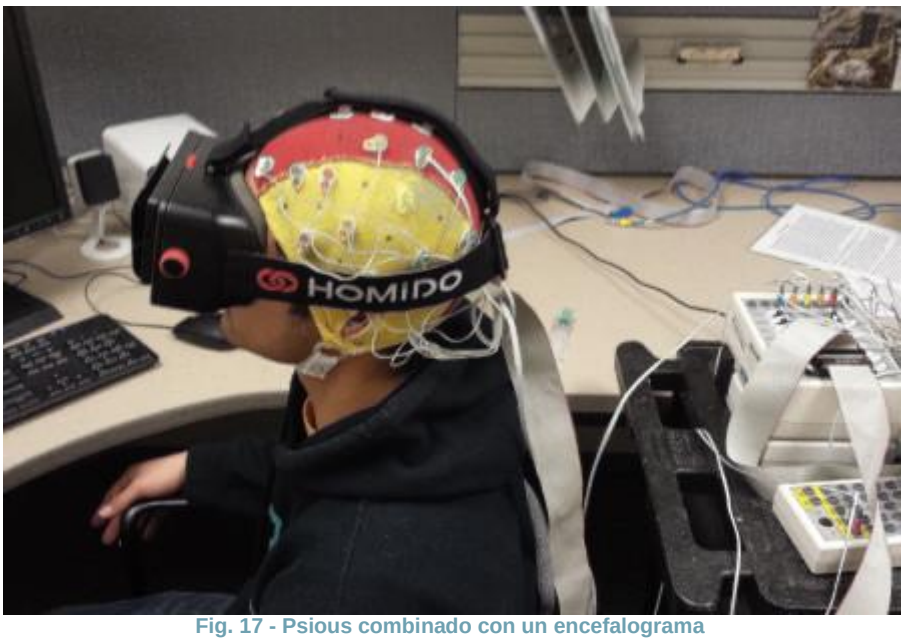


Fig. 17 - Psious combinado con un encefalograma

El usuario puede desplazarse por una gran variedad de escenarios que le permiten tratar con diversas terapias como son el miedo a volar, a las agujas, a hablar en público, a conducir o el miedo a los insectos. También ofrece tratamientos para la acrofobia, la claustrofobia o la agorafobia e incluso una sala de relajación.

Pruebas con usuarios

Previamente a hacer la aplicación se hizo un test con diferentes usuarios y diferentes aplicaciones para ver cuál era la reacción ante el uso de las Oculus.

Hay que tener en cuenta también que las pruebas se hicieron con las Oculus DK1.

A la pregunta "¿Te has mareado?", todos los usuarios respondieron que sí en mayor o menor medida. Se les pidió cuantificarlo y la mayoría afirmó marearse en una intensidad igual o inferior a 5.

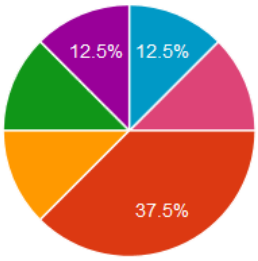


Fig. 18 - Gráfico de la intensidad de mareo

Al preguntar sobre la causa de porqué se habían mareado, un 62,5% respondió que la causa fue la sensación causada al moverse la imagen. Mientras que un 25% respondió que fue a causa de que no tenía la sensación de girar junto a la cámara y el resto fue causado por el malestar generado por cargar con las gafas.

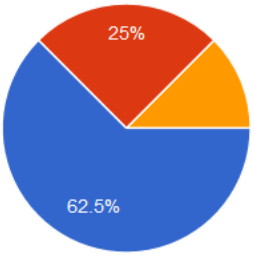


Fig. 19 - Gráfico de los motivos del mareo

Respecto a la tardanza del inicio del mareo hubo un gran varianza, entre 1 y 5 minutos, dependiendo de la aplicación ejecutada, en caso de haber más dinamismo, menor tiempo pasaba hasta empezar las primeras molestias.

También hicimos un cuestionario sobre qué mejorarían, todos concluyeron en que los puntos a mejorar eran la resolución de imagen, la calidad de los escenarios y la incomodidad causada por cargar con las gafas y el contacto con las lentes.

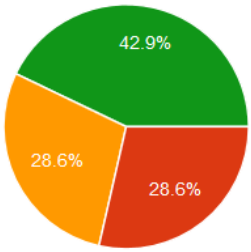


Fig. 20 - Gráfico de cosas a cambiar

Aún con estos defectos, la mayoría de usuarios se sintieron integrados gracias a la combinación de las gafas y el sonido ambiente.

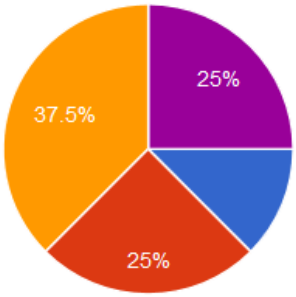


Fig. 21 - Gráfico de las causas de sensación de integración

Aún así a todos les pareció una experiencia satisfactoria.

Argumento e hilo conductor

Sinopsis

Después de muchos años encerrado y experimentando con él, logran soltar a un sujeto de pruebas, pero no ha salido del centro. Para ello tendrá que superar algunas salas de experimentación.

Referentes

Tomando como referencia principal el concepto de Portal, se han creado distintas salas de prueba en las que vas avanzando según completas cada una de ellas.

A su vez se ha cogido la estética de Portal para la sala central y la sala exterior.

Fig. 22 - Imagen Portal 2

Guión

SALA OSCURIDAD

El personaje se despierta en una mina junto a un farol.

VOZ EN OFF

¡¡Por fin te despiertas!! Llevas horas inconsciente. Llevan años experimentando con nosotros pero hemos logrado sacarte de allí. A partir de aquí tendrás que buscar una salida.

El personaje avanza en la oscuridad.

Encuentra una linterna.

VOZ EN OFF

- ¡¡Has encontrado una linterna!! Ahora podrás iluminar mejor la zona, pero… ¿estás seguro de que no va a fallar?, sería mejor que buscaras baterías, por si acaso.

Sale de la sala de la oscuridad.

SALA BLANCA

Aparece en una sala totalmente blanca totalmente vacía.

VOZ EN OFF

- ¿Realmente creías que te estaba ayudando de verdad? Estoy seguro de que no ves la salida que hay delante de ti.

Al avanzar por la sala aparece un edificio por el que puede salir.

SALA ALTURA

De repente se encuentra en una sala deteriorada con planchas metálicas.

VOZ EN OFF

- ¿¡Cómo!? Así que has encontrado la salida... pero aún no eres libre. Espero que a estas alturas no tengas miedo…

Al avanzar por la sala varias placas se caen, y al final del pasillo se encuentra una bifurcación.

VOZ EN OFF

¿Qué te crees, que las cosas aquí son fáciles? Escoge tu camino.

Después de seguir atravesando placas inestables y recorriendo vigas acaba encontrando la salida de ese lugar.

SALA BLANCA

Reaparece en la sala.

VOZ EN OFF

- Ooooh, que pena. Pues parece no era la salida.
Así no escaparás nunca. Esperaba mucho más de ti.

Pierde el conocimiento.

Diseño de la interactividad

Acciones

Controles

El usuario puede hacer que el personaje se desplace por los diversos escenarios usando tanto un mando de control o el teclado y el ratón.

Usando estas herramientas puede caminar hacia delante o hacia atrás, al igual que rotar el cuerpo para dirigir en qué dirección avanzará y retrocederá y usando las Oculus podrá mirar a su alrededor

Elementos a interaccionar

Organizando los elementos en las distintas salas, tenemos:

Sala oscuridad:

- Gotas que caen del techo y chocarán con el usuario simulando que le salpica.
- Una vela, que llevará al inicio del escenario y que le permitirá iluminar una escasa zona a su alrededor.
- Una linterna, que deberá encontrar, la cual irá parpadeando a medida que pase el tiempo y con cada parpadeo su intensidad se reducirá y con la que puede dirigir el cono de luz.
- Baterías repartidas por el escenario que recargarán la linterna para que vuelva a tener toda la intensidad.
- La salida de la sala, de la que aparecerá un conjunto de partículas al acercarse a cierta distancia y se iluminará, para que pueda vislumbrarla correctamente.

Sala blanca:

- El edificio que le permite avanzar al siguiente nivel. Este edificio sólo irá apareciendo mientras se aproxime a él, mientras que si se aleja, desaparecerá.
- Además, sin que el usuario sea consciente de ello, está rodeado por una esfera que, al tocar uno de sus extremos, será llevado al extremo opuesto con tal de simular un escenario infinito.

Sala altura:

- Durante el trayecto, varias plataformas y vigas reaccionarán a su paso con tal de sorprenderlo, ya sea cayéndose al vacío o desestabilizándose.
- Si el usuario se cae, antes de alcanzar el suelo, reaparecerá de nuevo arriba.

Desarrollo de la interacción

Preparación de la interacción

Para que los elementos interactúen correctamente entre ellos y para poder hacerles referencia en el código es necesario asignarles distintas características a los distintos elementos.

Triggers

Para accionar cada una de las interacciones ya mencionadas en el apartado de arriba, en cada uno de los escenarios hay repartidos distintos triggers que se activan al pasar el personaje.

- En el escenario de la oscuridad:

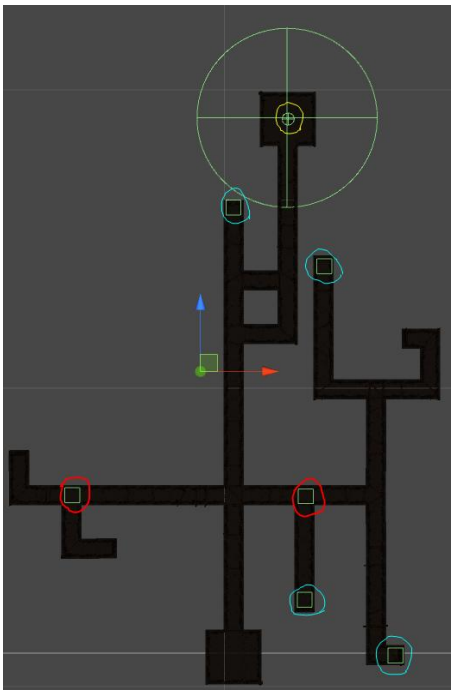


Fig. 24 - Mapa de "triggers" del mapa de oscuridad

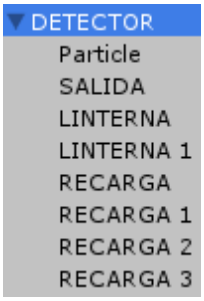


Fig. 23 - Grupo de "triggers"

En la imagen se pueden visualizar todos los triggers del escenario.

Los marcados en rojo son los triggers que activan la linterna, los azules son los que activan la recarga de la linterna, el marcado en amarillo es la salida, y por último, el trigger circular grande es el que activa la emisión de partículas que marca la salida.

Cada uno de los triggers es un objeto colocado en la posición que se puede visualizar en la imagen almacenados todos en un solo grupo para tenerlos localizados en todo momento.

- En el escenario de transición:

En este caso hay dos triggers, uno circular y otro cuadrado.

El circular detecta cuándo el personaje sale del trigger para colocarlo en el extremo opuesto.

El cuadrado detecta cuándo entra y ejecuta la siguiente escena.

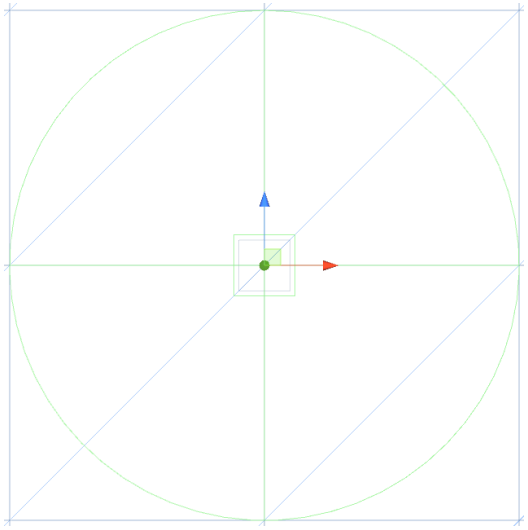
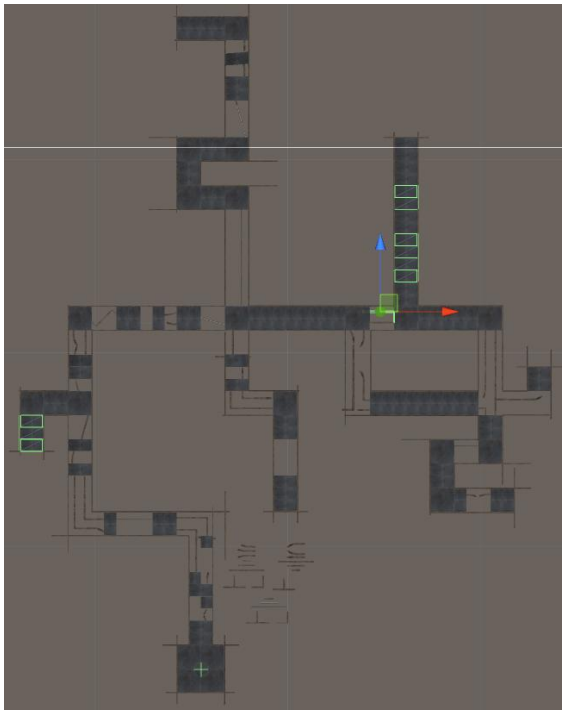


Fig. 25 - Mapa de "triggers" del mapa blanco

- En el escenario de la altura:



En este escenario, todos los triggers mostrados activan distintas animaciones en las que se hace caer algunas de las plataformas del escenario excepto por el circular que hay en la parte inferior de la imagen que es la salida del escenario.

A parte de estos, hay otro más que rodea todo el escenario a una altura inferior que detecta cuándo se cae el personaje.

Fig. 26 - Mapa de "triggers" del mapa de altura

Capas de físicas (Layers)

Para que todos estos triggers se activen adecuadamente al entrar en contacto con el elemento adecuado uso las layers:

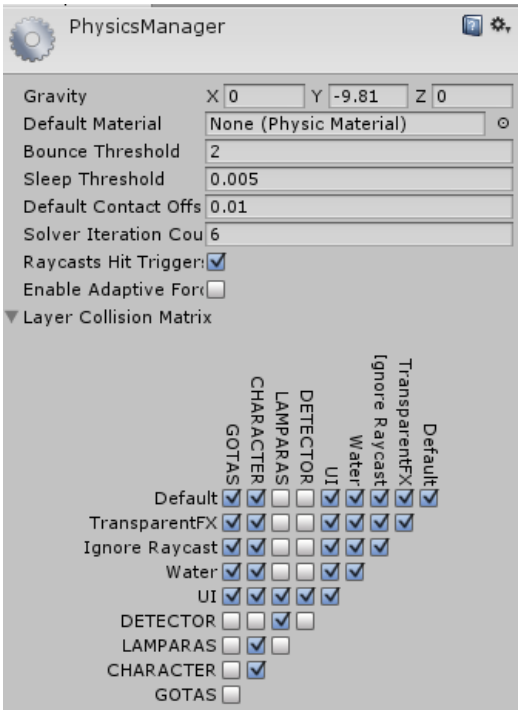


Fig. 28 - Lista de interacción entre "layers"

La layer de DETECTOR y la de LAMPARAS están creadas para que sólo puedan interactuar entre ellos.

Fig. 27 - Lista de "layers" creadas

Etiquetas (Tags)

Para identificar qué tiene que suceder cuando el personaje atraviesa uno de los triggers utilizo diversas etiquetas (tags).

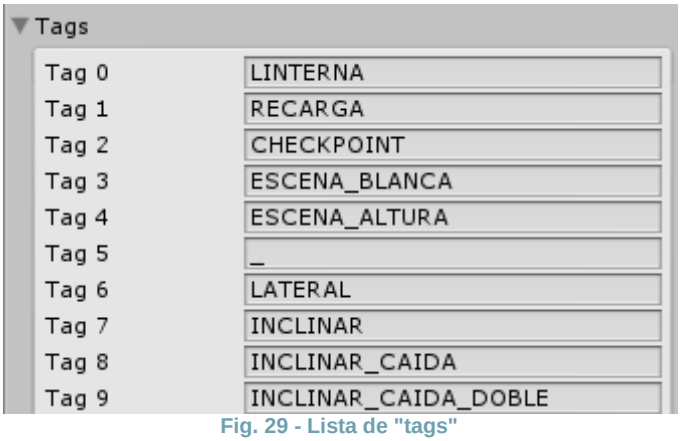


Fig. 29 - Lista de "tags"

Los "tags" de LINTERNA y RECARGA identifican acciones con la linterna que hay en el escenario de oscuridad, como son recoger la linterna y recargarla respectivamente. El de CHECKPOINT marca puntos de control dentro del escenario de la altura por si el jugador se cae.

Los dos siguientes "tags" corresponden a los triggers de salida, hacia el escenario blanco y el escenario de la altura.

Y los 4 "tags" restantes hacen referencia a las animaciones que se ejecutan al entrar en contacto con las plataformas del escenario de la altura.

Entradas (Inputs)

A la hora de controlar al personaje, Oculus define unos controles en relación a botones específicos, con tal de poder decir otros botones sin tener que modificar constantemente el código, he marcado unos "inputs" que me permita cambiarlos rápidamente.

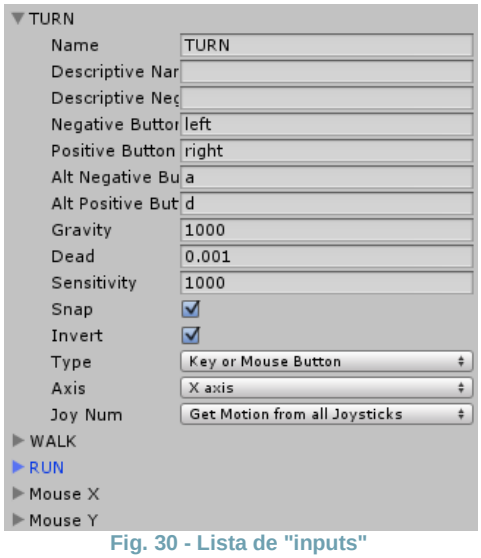


Fig. 30 - Lista de "inputs"

El de TURN indica los controles para girar hacia la derecha e izquierda, el de WALK es para avanzar y retroceder, RUN es para hacer que corra y los de "Mouse" sirven para el control de del ratón.

En el caso de TURN, WALK y RUN, he tenido que especificar una "Gravity" y una "Sensivity" muy alta para hacer que la acción por el botón funcionara y se parara al instante puesto que "Gravity" sirve para indicar la velocidad a la que se reduce la intensidad del botón y el "Sensitivity" indica la velocidad a la que aumenta.

Programación

La programación se puede seccionar en 5 partes bien definidas:

General

Este código se utiliza en todas las escenas.

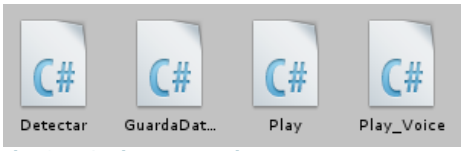


Fig. 31 - Scripts generales

Controla la reproducción de los efectos sonoros y de la voz, además de decir qué sucede en todo momento dependiendo de dónde esté el personaje. Todos los scripts están dentro del personaje ya que todo lo definido por estos scripts sucede en torno a él.

El script de *Play* (Fig. 63) se encarga de reproducir el audio de los pasos mientras camina el personaje evitando que se reproduzca el mismo audio dos veces seguidas. Para ello, tiene una animación que se reproduce al avanzar y retroceder con un evento que ejecuta el script.

Fig. 32 - Animación para la reproducción de pasos

En el script viene definida la salida de audio con una variable de *AudioSource* junto a un "array" de *AudioClips* para introducir los audios, que es variable para el caso de querer añadir más.

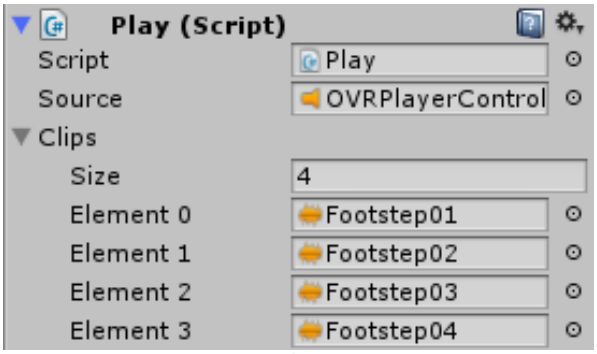


Fig. 33 - Variables públicas del script Play

Mediante dos variables de tipo entero se hace la comprobación de cuál ha sido el último audio reproducido. Para ello, a una se le asigna un número aleatorio que abarque una de las posiciones dentro del "array" y luego se comprueba si ese número ha sido el último en reproducirse utilizando la otra variable, en caso afirmativo, vuelve a repetir la asignación de valor hasta que no coincida. Una vez llegado a este punto, se reproduce el audio seleccionado y se guarda la posición del audio reproducido en la variable que indica la última reproducción para que se pueda comprobar en la siguiente ejecución. En la caso de la primera vez, está forzado que sea el primer audio del "array" el que no se reproduzca.

El script de *Guardadatos* crea una clase pública y estática para poder registrar cuándo ha entrado en la sala blanca. Al ser estática se podrá acceder en cualquier momento desde cualquier otro script, permitiendo conservar el cambio incluso al cambiar de escena.

Es el único de este conjunto que no está añadido al personaje puesto que no es necesario ponerlo para acceder a él.

```
1  public static class Guardadatos
2  {
3      public static bool HaEntrado = false;
4  }
```

Fig. 34 - Código del script *Guardadatos*

Play_Voice (Fig. 64) se encarga de reproducir la voz en off además de indicar cuándo se está reproduciendo para bloquear la ejecución de ciertas líneas que se verán más adelante.

Al entrar en una nueva sala, se ejecuta una función que reproduce el primer audio del "array" generado, excepto al entrar por segunda vez en la sala blanca, que se ejecuta el segundo.

Mientras el audio se esté reproduciendo, cambia los valores de las variables *isPlaying* de los distintos scripts que bloquearan las líneas.

Detectar es el script que permite saber en qué lugar está el personaje, activa ciertos cambios en el escenario dependiendo de por dónde pasa. Además controla el funcionamiento de la linterna usada en el caso de la sala de la oscuridad.

Para el funcionamiento del cambio de luz dentro de la sala de oscuridad se declaran cuatro variables que cogen, por un lado, el objeto contenedor (los **GameObject** *_lantern* y *_lamp*) y la luz dentro de estos objetos (los *Light* *_lanternLight* y *_lampLight*).

Para la primera vez que se encuentra una linterna en la sala de la oscuridad o la primera vez que se cae en la sala de la altura están las variables *_lampFirst* y *_first* respectivamente.

Las **booleanas** *Entrar* e *isPlaying*, se utilizan al entrar por segunda vez en la sala blanca para detectar que se ha entrado por primera vez y poder finalizar la aplicación.

```
7       private GameObject _lantern;
8       private Light _lanternLight;
9
10      private GameObject _lamp;
11      private Light _lampLight;
12
13      private bool _first = false;
14      private bool _lampFirst = false;
15
16      public bool Entrar = false;
17
18      public bool IsPlaying = false;
```

Fig. 35 - Variables del script *Detectar*

Al empezar una sala, busca la linterna, en caso de encontrarla, guarda tanto la linterna y el farol como sus respectivas luces (Fig. 65).

Al hacer click en la letra "R", se coge la dirección en la que está mirando el cuerpo del personaje y se ejecuta una función de Oculus Rift que reinicia tanto la orientación de cámara como la orientación del cuerpo a la inicial, para que luego se reoriente el cuerpo en la dirección previa antes de darle a la "R" y con ello la cámara.

Al entrar por segunda vez en la sala blanca, la variable *Entrar* queda como "true" y, al acabar de reproducirse la voz en off, se ejecuta la función *fade* que se encarga de cerrar la aplicación.

```
34      // Update is called once per frame
35      void Update()
36      {
37          if (Input.GetKeyDown(KeyCode.R))
38          {
39              Quaternion direction = transform.rotation;
40              OVRManager.display.RecenterPose();
41              transform.rotation = direction;
42          }
43          if (Entrar && !IsPlaying)
44          {
45              StartCoroutine(fade(3));
46          }
47      }
```

Fig. 36 - Función *Update* del script *Detectar*

Cuando el personaje pasa por un "trigger", se activa una función dependiendo del "tag" que tenga el objeto con ese "trigger":

- Si pasa por una linterna, se apagará la luz del farol y se encenderá el de la linterna, además se iniciará un corrutina que afecta a la intensidad de la luz, y si es la primera vez que se encuentra con una linterna, se ejecutará un audio de la voz en off.
- A la hora de pasar por una zona de recarga, se reinician todas las corrutinas y empieza de nuevo la corrutina ejecutada al coger la linterna.
- En la sala blanca, al pasar por ciertos puntos, se guarda un punto de reaparición, en caso de pasar por el primer punto de control, se reproduce la voz en off.

  Al llegar al final de cada sala, hay un punto de salida a la sala blanca o a la sala de altura y al saltar a una sala distinta se ejecuta una función que genera un fundido a negro (Fig. 66).

La corrutina iniciada al coger la linterna espera un cierto tiempo antes de iniciar la función que ejecuta el parpadeo. Esta función espera un tiempo aleatorio definido entre dos valores especificados al ejecutarla y ejecuta una última función que genera el parpadeo aumentando y reduciendo la intensidad de la linterna un número aleatorio de veces (Fig. 67).

Esta última función se ejecuta repetidamente reduciendo la intensidad cada vez que hay un conjunto de parpadeos (Fig. 67).

La función *fade* se encarga de generar un fundido a negro modificando el alpha (transparencia) de una textura negra, para luego enviar al personaje a la siguiente sala. En caso de ser la sala final, genera el fundido y cierra la aplicación.

```
136     public Image Fade;
137
138     IEnumerator fade(int scene)
139     {
140         for (float i = 0; i <= 1.5f; i += 0.1f)
141         {
142             Fade.color = new Color(0, 0, 0, i);
143             yield return new WaitForSeconds(0.01f);
144         }
145         if (scene == 3)
146         {
147             Application.Quit();
148         }
149         else
150         {
151             Application.LoadLevel(scene);
152         }
153     }
```

Fig. 37 - Corutina del script *Detectar* utilizada para hacer un fundido a negro y saltar a una nueva escena

Oscuridad

Estos son los utilizados en el escenario de la oscuridad.

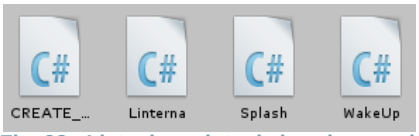


Fig. 38 - Lista de scripts de la sala oscuridad

El script *CREATE_PARTICLE* (Fig. 68) se utiliza para iniciar un conjunto de emisores de partículas y encender gradualmente una luz en la salida de la cueva cuando el usuario se aproxima a ésta.

Al ejecutar la función *GetComponent* se puede coger todos los componentes de un tipo especificado que haya en ese objeto, tanto del padre como de los hijos. Aprovechando esta característica, se almacena en una variable el padre, que también contiene la luz, para así manipular al padre y su contenido con una única variable.

Una vez tenemos el objeto padre, se ejecuta la función *Stop* del componente "ParticleSystem" nada más empezar para hacer que no haya ningún emisor de partículas emitiendo.

Cuando el personaje entre en el "trigger" correspondiente, se ejecutará por un lado el Play de los emisores y por otro una función del script *WakeUp* que contiene la luz que hace que la intensidad luz aumente gradualmente.

Al salir del "trigger", se ejecuta una función para parar la emisión de partículas y otra para reducir la intensidad de la luz gradualmente.

La luz de la prefab mencionada anteriormente contiene el código del script *WakeUp* (Fig. 69).

Este script coge la componente de la luz y ejecuta una corrutina en la que hace que la intensidad varíe gradualmente, dependiendo de qué función se ejecute la aumentará o la reducirá.

Para evitar que se ejecute indefinidamente tiene una comprobación que observa cuándo llega a los límites de intensidad máximos (0 y 8). Una vez hecha esa comprobación, le suma o resta un valor especificado con la función que se haya ejecutado previamente hasta llegar al límite. Puesto que en la comprobación no se consideran los extremos y la intensidad alcanza esos valores, la primera operación se ejecuta fuera de la comprobación para así hacer que el valor de la intensidad entre dentro de la condición.

Para evitar que se lleguen a ejecutar las dos funciones a la vez en caso de que se ejecute una cuando la otra no ha finalizado, antes de ejecutar cualquiera de las dos corrutinas se ejecuta una función que pausa todas las corrutinas del script.

El código de *Linterna* (Fig. 70) está adherido al objeto que representa la linterna. Con este código se rota la linterna al mover el ratón y se limita esta misma rotación de la linterna para que solo pueda apuntar en un cierto rango frente al personaje.

La rotación se ejerce detectando el desplazamiento vertical y horizontal del ratón individualmente y sumando a un número que empieza en 0 una cantidad específica a cada una de esas direcciones según se va moviendo el ratón. En caso de superar un cierto número marcado, la función *Clamp* hará que se quede en el límite especificado haciendo que a la hora de darle el ángulo a la linterna no se pueda pasar del límite.

El último script de la zona de oscuridad es el de *Splash*, está en el emisor de partículas de las gotas. Se encarga de detectar cuándo la gota colisiona con algún objeto para emitir el sonido de la gota al chocar. Además, para que no sea monótono, se le cambia el pitch de forma aleatoria para que suene más o menos agudo en cada colisión.

```
4  public class Splash : MonoBehaviour {
5
6      void OnParticleCollision(GameObject other) {
7          AudioSource source = GetComponentInChildren<AudioSource> ();
8          source.pitch = Random.Range (1.0f, 1.5f);
9          source.Play();
10     }
11 }
```

Fig. 39 - Código del script *Splash*

Sala blanca (Sala de transición)

En la sala blanca se utiliza un solo script junto con los ya mencionados que se utilizan en todas las salas.

Fig. 40 - Script de la sala blanca

El script *APARECER* (Fig. 71) se encarga de detectar la proximidad al edificio central, donde está la salida.

Cogiendo la posición tanto del personaje como del edificio, las resta para y luego transforma el punto resultante en una magnitud para saber qué distancia y muestra u oculta el edificio en consecuencia a esa distancia.

Para hacer que el edificio se muestre del todo antes de llegar frente a él, una vez se tiene la distancia, se le resta a la suma de la máxima distancia entre un punto más la distancia a la que se quiere que aparezca por completo, el desvío.

Además, utiliza un "trigger" en el que está el personaje y, cuando éste sale del "trigger", lo coloca en el extremo opuesto, generando la sensación de un camino sin fin.

Altura

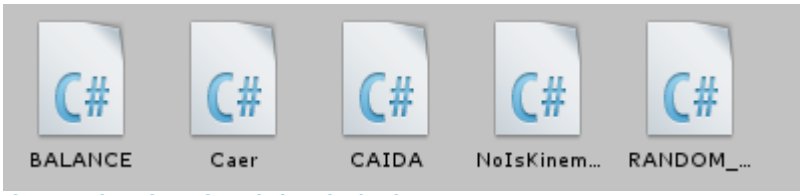


Fig. 41 - Lista de scripts de la sala de altura

El *RANDOM_PLAY* (Fig. 72) se encarga de reproducir aleatoriamente diversos efectos sonoros de forma aleatoria mediante una corrutina.

BALANCE (Fig. 73) es un script que ejecuta diversas animaciones de balanceo de las lámparas. Ejecuta una animación al azar y, mientras se ejecuta en bucle, cambiará a otra al azar creando una transición entre ambas.

Alrededor del personaje hay un "trigger" que se encarga de detectar las farolas cercanas haciendo que se muevan, evitan así una cantidad de procesos excesiva.

El script de *Caer* (Fig. 74) se encarga de detectar cuándo el personaje cae de alguna plataforma o viga mediante un "trigger" situado en la zona inferior de la sala que, al atravesarlo, crea un fundido a negro cogiendo un plano negro situado frente a la cámara y modifica el alpha para que se vuelva opaco, recoloca al personaje en uno de los puntos de control además de reiniciar la dirección del cuerpo y luego reduce el alpha a 0 para que se pueda ver de nuevo. Para acabar, reproduce un audio.

El script *CAIDA* (Fig. 75) detecta mediante "tags" qué animación tiene que ejecutar cuando el personaje entra en el "trigger" correspondiente. También está preparado para que en algún caso ejecute un par de animaciones seguidas, pudiendo controlar el retraso entre ambas.

NoIsKinematic (Fig. 76) es un script que detecta cuándo el usuario se aproxima a ciertos elementos para desactivar la característica de "Is Kinematic", haciendo que el objeto actúe siguiendo las físicas y caiga. Además, va reproduciendo varios sonidos para simular el choque con los distintos elementos del escenario utilizando dos reproductores, uno para el primer sonido y otro para el segundo.

OVR

Este es el código que viene definido en el "package" de Oculus Rift. Es un conjunto de scripts que, combinados, forman toda la funcionalidad del control, tanto del personaje como de la cámara de Oculus.

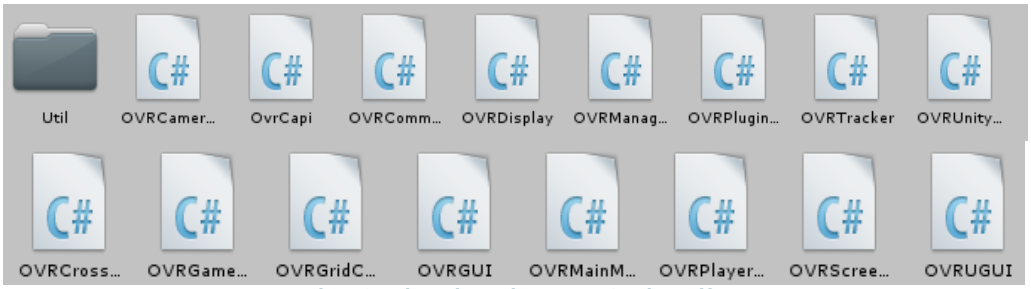


Fig. 42 - Lista de scripts para Oculus Rift

Inicialmente este código no es necesario tocarlo, pero yo lo he modificado para que utilice los "inputs" que he definido anteriormente además de hacer que se ejecuten audios de pasos al avanzar y retroceder y para bloquear estos mismos "inputs" cuando se reproduce la voz en off.

Los scripts principales son:

Colocado en el OVRCameraRig, necesario para el movimiento de la cámara con las Oculus.

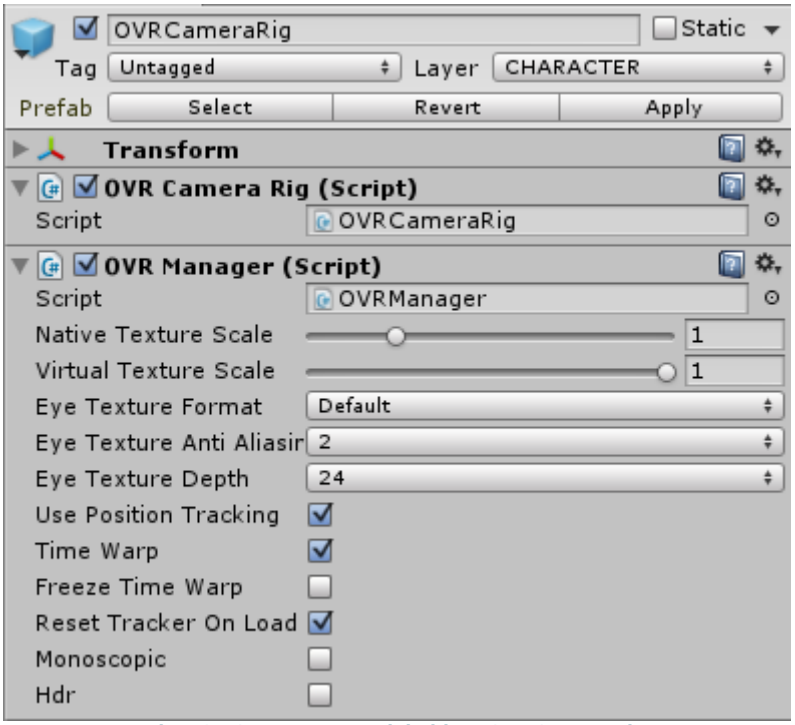


Fig. 43 - Componentes del objeto OVRCameraRig

El **OVRCameraRig** se encarga de colocar ambas cámaras en la posición correcta al igual que hacer que roten al mover las Oculus desde un mismo punto para simular correctamente el giro de la cabeza dentro del entorno virtual. Para poder obtener la configuración en la que tiene que trabajar, coge la información que le proporciona otro script, el **OVRManager**.

El *OVRManager* , proporciona todos los datos sobre la configuración definida previamente de las Oculus mediante el plugin. Además proporciona algunas opciones para la visualización de las imagenes:

- **Native Texture Scale:** Especifica la calidad de las texturas de cada ojo. Por debajo de 1, la calidad se reduce perdiendo imagen, por encima de uno, la calidad de imagen aumenta, pero también aumenta el procesamiento.

- **Virtual Texture Scale:** Indica el tamaño de la textura al renderizarlo en cada uno de los ojos. Al reducir el número, la imagen aumenta de tamaño.

- **Eye Texture Format:** Define el formato de la textura, qué espacio de color usa.

- **Eye Texture Anti Aliasing:** Marca la cantidad de antialiasing utilizada para el render, reduciendo la visualización del pixelado.

- **Eye Texture Depth:** La profundidad de la textura en bits.

- **Use Position Tracking:** Sirve en caso de usar las Oculus DK2, detectar el desplazamiento de las gafas.

- **Time Warp:** Si se activa se utiliza el efecto Time Warp que sirve para corregir el efecto causado por la latencia.

- F**reeze Time Warp:** Si se activa, se para el seguimiento de las Oculus y el efecto TimeWarp sólo se actua al desplazamiento de la cabeza.

- **Reset Tracker On Load:** Al cargar una nueva escena, hace reinicia la cámara y la pone en su posición inicial.

- **Monoscopic:** Activar esta opción hace que en los dos ojos se vea exactamente lo mismo, que será lo renderizado por la cámara del ojo izquierdo.

- **Hdr:** Sirve para obtener imágenes en HDR, pudiendo aprovechar la información extra de estas imágenes para aplicar ciertos efectos de cámara.

Y en el OVRPlayerController, necesario para mover al personaje por el escenario.

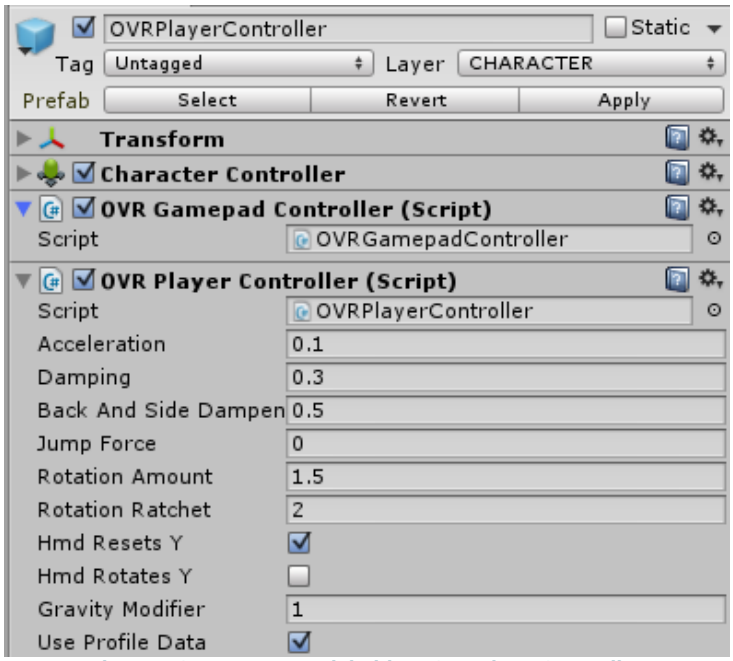


Fig. 44 - Componentes del objeto OVRPlayerController

El **OVRPlayerController** se encarga de hacer que el personaje se mueva recibiendo los inputs.

Para poder controlarlo, tiene varias características que puedes definir:

- **Acceleration:** Indica cuánto aumenta en un período de tiempo, por lo que, a mayor sea el número, mayor será la velocidad.

- **Damping:** Es la resistencia con el suelo. Si se pone a 0, nunca frenará, además, la velocidad aumentará de desplazamiento aumentará puesto que habrá un aumento de velocidad mayor.

- **Back And Side Danpen:** La velocidad de desplazamiento tanto lateral como hacia atrás.

- **Jump Force:** La intensidad de salto, a mayor intensidad, más alto es el salto.

- **Rotation Amount:** La intensidad de giro del personaje sobre sí mismo. En este caso sirve para girar con el joystick del mando y es para girar fluidamente.

- **Rotation Ratchet:** Al accionar ciertos botones, tanto de teclado como de mando, hace un giro de los grados especificados en este campo.

- **Hmd Resets Y:** Reinicia la orientación en el eje Y de la cámara cuando se reorienta el personaje.

- **Hmd Rotates Y:** Hace que el ratón pueda rotar la cámara.

- **Gravity Modifier:** La gravedad ejercida sobre el personaje, a mayor gravedad, más rápido descenderá.

- **Use Profile Data:** Obtener los datos de la altura especificada en el perfil de Oculus para usarla.

Para poder usar los controles de los mandos este script se complementa con el script *OVRGamepadController* que reconoce la entrada de los botones y los nombra para que el *OVRPlayerController* pueda identificarlos y usarlos.

En el caso del *OVRPlayerController*, como ya he mencionado anteriormente, lo he modificado para usar unos controles distintos.

En la función *Update* de este script se ejecuta la función *UpdateMovement* que está centrada en las funcionalidades del desplazamiento.

Inicialmente esta función declara las variables *boleanas* que detectan los distintos inputs haciendo que den valor "true" al accionar los botones especificados para cada una de las variables.

```
bool moveForward = Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow);
bool moveLeft = Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow);
bool moveRight = Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow);
bool moveBack = Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.DownArrow);
```

Fig. 45 - Variables del script *OVRPlayerController* para detectar qué input ha sido activado en el teclado

Por otro lado, define cuándo se acciona el eje vertical del joystick del mando. Para ello, declara una variable **boleana** que tendrá el valor "true" en el caso de accionarse el botón hacia arriba o hacia abajo del mando que serán reconocidos por el script ya mencionado, el *OVRGamepadController*.

```
bool dpad_move = false;

if (OVRGamepadController.GPC_GetButton(OVRGamepadController.Button.Up))
{
    dpad_move    = true;

}
if (OVRGamepadController.GPC_GetButton(OVRGamepadController.Button.Down))
{
    dpad_move = true;
}
```

Fig. 46 - Variables del script *OVRPlayerController* para detectar qué input ha sido activado en el teclado

Una vez se tienen los inputs definidos, se indica qué velocidad de desplazamiento tendrá (Fig. 77).

Teniendo en cuenta como velocidad base el andar, se especifica con una escala de velocidad, poniendo una escala menor al desplazarse en diagonal para evitar el aumento de velocidad ya que tendría un desplazamiento de escala 1 en ambas direcciones.

Después de haber especificado la escala, se aplica un cálculo para simular la aceleración y luego se multiplica a una variable de tipo flotante que especifica cuánto aumenta en cada unidad de tiempo. En caso de darle al botón de correr duplicará este último resultado haciendo así que el desplazamiento sea mayor.

Si por algún motivo el personaje no toca el suelo, no podrá desplazarse.

Finalmente se crea el vector de dirección sumando el valor específico para cada dirección generando un desplazamiento fluido.

```
if (moveForward)
    MoveThrottle += ort * (transform.lossyScale.z * moveInfluence * Vector3.forward);
if (moveBack)
    MoveThrottle += ort * (transform.lossyScale.z * moveInfluence * BackAndSideDampen * Vector3.back);
if (moveLeft)
    MoveThrottle += ort * (transform.lossyScale.x * moveInfluence * BackAndSideDampen * Vector3.left);
if (moveRight)
    MoveThrottle += ort * (transform.lossyScale.x * moveInfluence * BackAndSideDampen * Vector3.right);
```

Fig. 47 - Código del script *OVRPlayerController* en el que se efectua el desplazamiento

Además del desplazamiento, también se encarga de rotar el cuerpo del personaje.

En el caso del mando detecta dos de los gatillos del mando, uno izquierdo y uno derecho, para ejecutar el giro (Fig. 78).

Para evitar que este salto se haga constantemente al dejar el botón apretado, se hace una comprobación de si en el fotograma anterior el botón ha sido accionado, si no es así, se ejecutará el cambio, en caso contrario, no hará nada.

Para indicar cuándo se ha accionado se crea una variable **boleana** que será "true" al accionar el botón y "false" al soltarlo, luego se hace la comprobación y finalmente se le asigna a otra variable **boleana** el estado del botón (si está o no accionado). Al ejecutarse por segunda vez la función, si el botón sigue accionado, no se ejecutará porque la última variable mencionada tendrá guardado que el botón lo estaba anteriormente, y si el botón ya ha dejado de ser accionado, en la variable se guardará como "false" y se podrá volver a girar clicando de nuevo el botón.

En el caso del teclado usa como referencia los botones "Q" y "E".

```
//Use keys to ratchet rotation
if (Input.GetKeyDown(KeyCode.Q))
    euler.y -= RotationRatchet;

if (Input.GetKeyDown(KeyCode.E))
    euler.y += RotationRatchet;
```

Fig. 48 - Código del script *OVRPlayerController* utilizado para rotar al personaje con teclado

A la hora de rotar en una dirección u otra, lo hace dependiendo del ángulo especificado en el campo "RotationRatchet" explicado al inicio de este punto.

En caso de tener activada la opción de "Hmd Rotates Y", se podrá rotar al personaje usando el ratón.

```
if (!SkipMouseRotation)
    euler.y += Input.GetAxis("Mouse X") * rotateInfluence * 3.25f;
```

Fig. 49 - Código del script OVRPlayerController utilizado para rotar al personaje con el ratón

Si se está utilizando el mando, la base del desplazamiento es la misma pero varía en que, al utilizar el joystick izquierdo del mando, la intensidad del desplazamiento viene definida por la inclinación del eje (Fig. 79).

Además tiene el añadido de poder girar con el joystick derecho permitiendo girar al personaje a mayor o menor velocidad según su inclinación en el eje horizontal.

```
float rightAxisX = OVRGamepadController.GPC_GetAxis(OVRGamepadController.Axis.RightXAxis);

euler.y += rightAxisX * rotateInfluence;

transform.rotation = Quaternion.Euler(euler);
```

Fig. 50 - Código del script *OVRPlayerController* utilizado para rotar al personaje con el stick derecho del mando de control

También tiene una función para hacer saltar al personaje. En caso de estar tocando el suelo se le añadirá al vector de dirección una dirección vertical ascendente permitiendo saltar en cualquier dirección.

```
public bool Jump()
{
    if (!Controller.isGrounded)
        return false;

    MoveThrottle += new Vector3(0, JumpForce, 0);

    return true;
}
```

Fig. 51 - Código del script *OVRPlayerController* utilizado para hacer saltar al personaje

Y otra función para hacer que el personaje quede totalmente estático.

```
public void Stop()
{
    Controller.Move(Vector3.zero);
    MoveThrottle = Vector3.zero;
    FallSpeed = 0.0f;
}
```

Fig. 52 - Código del script *OVRPlayerController* utilizado para parar al personaje

Aprovechando éste código, le he hecho algunos **cambios**, al igual que le he **añadido algunas variables** para que tuviera la funcionalidad deseada para este proyecto.

Para empezar, le he añadido dos variables más para poder ejecutar animaciones y poder bloquear los controles cuando se reproduce la voz en off.

La primera variable, state, es de tipo entero e indica a qué animación tiene que ir. De esta forma hay 3 animaciones:

- La del personaje estático.

- La del personaje andando.

- La del personaje corriendo.

Excepto en el primer caso, todas contienen el script *Play* que reproduce los pasos, permitiendo no reproducir pasos o reproducirlos a mayor o menor velocidad.

La segunda variable, *isPlaying*, es **boleana** y cambia a "true" cuando se reproduce la voz en off ejecutada en el script *Play_Voice*.

```
203     //AÑADIR ANIMATOR
204     private int state = 0;
205     public bool isPlaying = false;
```

Fig. 53 - Variables añadidas al script *OVRPlayerController*

En caso de que la variable **boleana** *isPlaying* tenga valor "true", se ejecutará la animación de posición estática y saldrá de la función.

```
212         if (isPlaying)
213         {
214             GetComponent<Animator>().SetInteger("Walk", 0);
215             return;
216         }
```

Fig. 54 - Condición para forzar al personaje quedarse estático mientras se reproduce la voz en off

```
219         bool moveForward = false;
220         bool moveBack = false;
221
222         if (Input.GetAxis("WALK") > 0)
223         {
224             moveForward = true;
225         }
226
227         if (Input.GetAxis("WALK") < 0)
228         {
229             moveBack = true;
230         }
```

Fig. 55 - Modificación de la entrada de inputs en el teclado

He eliminado los controles de derecha e izquierda y he hecho que delante y atrás dependa de un input creado por mí. De esta forma el usuario no podrá desplazarse lateralmente, a cambio genero un control para que pueda rotar como si usara el mando.

En el caso para detectar el botón de correr he cambiado primero, para utilizar los inputs creados, y luego para poder especificar qué animación se tiene ejecutar, si la de correr (*state* = 2) o andar (*state* = 1).

```
266         if (dpad_move || Input.GetButton("RUN"))
267         {
268             moveInfluence *= 2.0f;
269             state = 2;
270         }
271         else {
272             state = 1;
273         }
```

Fig. 56 - Modificación de la entrada de inputs para correr

La variación respecto al original en el caso del desplazamiento según el input es el "if - else" exterior para cambiar a la animación de estático cuando no anda hacia delante o hacia atrás, es decir, está estático o gira sobre sí mismo.

```
282         if (moveForward || moveBack)
283         {
284             if (moveForward)
285                 MoveThrottle += ort * (transform.lossyScale.z * moveInfluence * Vector3.forward);
286 
287             if (moveBack)
288                 MoveThrottle += ort * (transform.lossyScale.z * moveInfluence * BackAndSideDampen * Vector3.back);
289         }
290         else {
291             state = 0;
292         }
```

Fig. 57 - Modificación para la asignación de velocidad de desplazamiento

Como ya he mencionado antes, éste es el control del giro del personaje, mientras se esté presionando el botón correspondiente girará constantemente.

```
if (Input.GetAxis("TURN") > 0)
    euler.y -= RotationAmount;

if (Input.GetAxis("TURN") < 0)
    euler.y += RotationAmount;
```

Fig. 58 - Modificación del giro del personaje

Variación sin Oculus

Para poder probar la aplicación sin el uso de las Oculus he añadido un personaje en primera persona utilizando el que viene predefinido y modificándolo para que sea compatible con el resto del código.

```
public bool isPlaying = false;

private void FixedUpdate()
{
    if (isPlaying) return;
```

Fig. 59 - Modificación del script *FirstPersonController* para bloquear el control al reproducirse la voz en off

Al igual que en el caso de Oculus Rift se ha creado la variable *isPlaying* para identificar cuándo se está reproduciendo la voz.

Efectos de cámara

Puesto que hay que simular la vista humana con tal de causar la mayor integración, se han añadido varios efecto a la cámara ya que sino se ve toda la imagen nítida constantemente. Además, estos scripts dan la oportunidad de hacer postproducción de la imagen embelleciendo el resultado final.

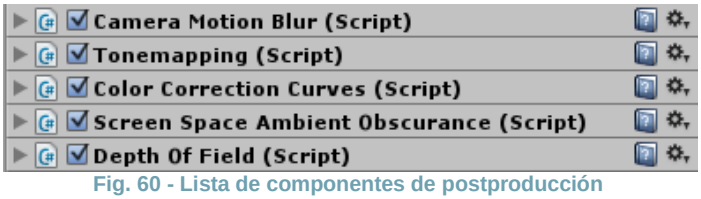


Fig. 60 - Lista de componentes de postproducción

Tonemapping

Permite obtener los datos de color de la imagen en 16 bits y pasarlo a una de 8 bits (cuantificado en valores del 0 al 1), pudiendo seleccionar qué franja de colores elegir y así ajustar el brillo de la imagen.

Fig. 61 - Comparativa "Tonemapping" sin efecto (izquierda) - con efecto (derecha)

Incluye distintos algoritmos de cálculo del tonemapping, entre ellos, hay algunos que son adaptativos, lo que permite que los colores varíen lentamente y con cierto retraso tal como lo haría el ojo humano.

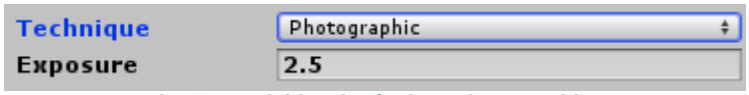


Fig. 62 - Variables de técnica "Photographic"

Este es un algoritmo que simula la modificación manual de la exposición de una cámara.

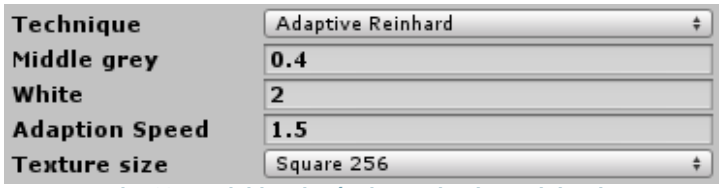


Fig. 63 - Variables de técnica "Adaptive Reinhard"

Y este es un método basado en el efecto adaptativo del ojo humano.

El **Middle grey** especifica qué valor de gris se tomará como medio marcando el conjunto de valores dentro de los 16 bits que se tomará.

El **White** indica qué valor dentro de la franja de 0 a 1 que se halla cuantificado será el que se considere blanco absoluto. En caso de poner un número superior a 1 se estará indicando que nunca habrá blanco absoluto.

La **Adaption Speed** es la velocidad de variación de un tono a otro después de un cambio de intensidad.

Y la **Texture size** es la textura que se utiliza para el muestre de los valores. A mayor sea el valor de la textura, más detalles calculará.

Depth of Field

Simula la profundidad de campo de la cámara para que solo se vea enfocado una parte de la imagen.

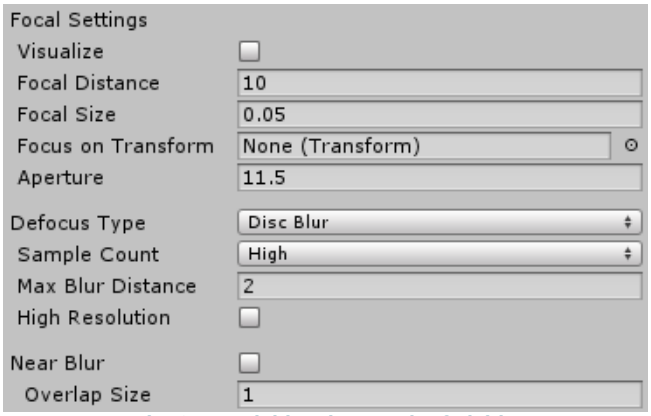


Fig. 64 - Variables de "Depth of Field"

Visualize muestra el desenfoque en blanco y negro.

Fig. 65 - Comparativa de "Visualize" desactivado (izquierda) y desactivado (derecha)

La zona en negro es la zona enfocada.

Focal Distance representa la distancia de la lente al objeto y, por lo tanto, modifica la distancia del área enfocada.

Focal Size es el tamaño del área enfocada.

Focus on Transform permite modificar la distancia focal en base la posición de ese objeto.

Aperture es el degradado de transición entre la zona enfocada y la desenfocada.

Defocus Type es el algoritmo que calcula el desenfoque.

Sample Count indica la cantidad de puntos que forman el desenfoque. Afecta considerablemente al rendimiento.

Max Blur Distance marca el tamaño de esos puntos.

High Resolution hace el desenfoque en alta resolución permitiendo que los puntos sean más definidos, esto sirve en caso de generar el efecto bokeh.

Near Blur crea un ligero desenfoque en las capas más cercanas a la cámara.

Overlap Size aumenta el efecto del Near Blur.

Screen Space Ambient Obscure

Oscurece pliegues, huecos y superficies que están cerca una de la otra para simular las sombras que no aparecen al usar la luz ambiental.

Fig. 66 - Comparativa "Screen Space Ambient Obscure" sin efecto (izquierda) - con efecto (derecha)

Fig. 67 - Variables de "Screen Space Ambient Obscure"

Intensity es el grado de oscuridad generado por el efecto.

Radius es el tamaño de las sombras.

Blur Iterations es el degradado de las sombras.

Blur Filter Distance se encarga de indicar a qué distancia empieza el degradado.

Downsample aumenta la resolución de los cálculos.

Rand y **Ao Shader** se encargan de la textura aplicada a la sombra.

Color Correction Curves

Te permite modificar el color de la escena mediante las curvas como se puede hacer con cualquier imagen o video.

Camera Motion Blur

Simula el desenfoque causado al mover la cámara.

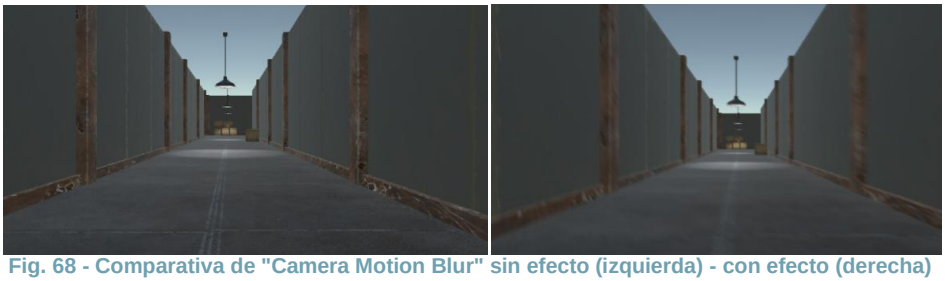


Fig. 68 - Comparativa de "Camera Motion Blur" sin efecto (izquierda) - con efecto (derecha)

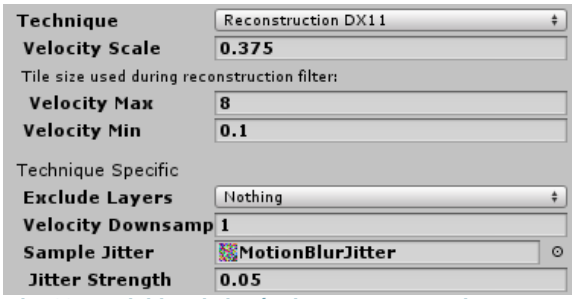


Fig. 69 - Variables de la técnica "Reconstruction DX11"

Technique es el algoritmo utilizado para el desenfoque.

Velocity Scale hace que haya más desenfoque a mayor es el número.

Velocity Max y **Velocity Min** limitan la prolongación del desenfoque en pixeles marcando el máximo de pixeles que se alargará y a partir de qué cantidad de pixeles se verá y desaparecerá respectivamente.

Estas son las variables en común, cada algoritmo, después, tiene ciertas variables que son características de cada uno. En este caso hablaré del usado en este proyecto.

Exclude Layers permite decidir qué objetos a los cuales se les haya asignado alguna de las capas (layers) creadas no les afecta el efecto creado por este script.

Velocity Downsample especifica la resolución del desenfoque.

Sample Jitter se utiliza para poner ruido al desenfoque evitando así que se vea la imagen por duplicado.

Jitter Strength es la intensidad del ruido.

Por último está la opción **Preview** que te permite visualizar cómo se vería el desenfoque con un desplazamiento teórico especificado.

Fig. 70 - Variables de la opción "Preview"

Conclusiones

La realidad virtual tiene un gran potencial en la e-salud pero los dispositivos de realidad virtual aun están en desarrollo por lo que de momento no se puede tampoco se han dado muchos usos en este campo.

Las Oculus son un dispositivo de realidad virtual muy capaz de integrar a una persona pero le faltan ciertas mejoras para evitar el mareo y la incomodidad que causa en algunas personas. Por otra parte, las Oculus solas no causan una integración total ya que se depende de un control, ya sea mando, teclado y ratón, etc., pero que combinándolas junto con otros dispositivos es posible seguir el movimiento del cuerpo (Kinect, Virtuix Omni) y causar una sensación casi absoluta de integración.

Es posible que con el último modelo de Oculus mostrado durante el transcurso de este proyecto y mencionado en él se haya reducido y/o evitado estos problemas ya que han mostrado una gran cantidad de mejoras pero puesto que no las hemos podido probar ya que aún no son accesibles no podemos asegurar nada.

En lo que se refiere a la organización del trabajo, es importante definir qué se va a hacer al completo para poder hacer el código según a todo el proyecto y no ir programando sobre la marcha ya que me he visto en la situación de repetir varias líneas de código innecesariamente, lo que me ha obligado que, al cambiar una de esas líneas, haya tenido que hacer cambio en el resto de scripts o ir creando parches en base al código creado mientras que podría haber hecho un script que me permitiera poner en común esas líneas con el resto del proyecto.

Por otra parte, probamos a utilizar físicas para calcular el movimiento de algunos de los objetos pero descubrimos que no era posible pues tratábamos de aplicarlo a una gran cantidad de objetos y requería demasiados procesos, por lo que lo simulamos mediante animaciones.

A parte de aprender de mis errores, he aprendido a darle uso a las variables estáticas y a su vez he mejorado mi comprensión sobre las corrutinas que, aún habiéndolas usado anteriormente, hasta ahora no había aprovechado potencial.

Por último, he sido capaz de manejarme con el código de Oculus ya que, aún siendo muy amplio, no resulta de una alta complejidad, por otra parte, aún me queda mucho por aprender de Oculus ya que, como he dicho, es muy amplio y no he podido verlo todo.

Agradecimientos

Para empezar quisiera agradecer a Sergio Martín Millán, mi compañero en este proyecto, ya que sin él no hubiera sido posible realizarlo pues me hubiera resultado imposible realizar el modelado, mapeado y texturizado de todos los elementos.

A su vez, a Juan José Fábregas Ruesgas por proporcionarme la oportunidad de realizar este proyecto al igual que guiarme durante el transcurso de éste y a Josep María Duque Ros por colaborar con sus valoraciones y aportación de diversas fuentes de información.

Por otra parte, quiero agradecer a todos aquellos que han colaborado en la producción y mejora del proyecto ofreciéndose como voluntarios para las diferentes pruebas que fuimos realizando y a David Catalán Sastriques por darle voz a nuestro observador dentro del entorno.

Y también a mi familia y a la de Sergio Martín Millán por apoyarnos durante todo el proyecto.

Bibliografía

E-Salut

A., H. C. (21 de febrero de 2011). *hcglobalgroup.* Obtenido de http://www.slideshare.net/hugoces/esalud-ehealth-tecnologas-y-la-nueva-realidad-de-los-sistemas-de-salud-a-nivel-global

Fobias

Wikipedia. (3 de agosto de 2014). *Wikipedia.* Obtenido de http://es.wikipedia.org/wiki/Nictofobia

Wikipedia. (12 de septiembre de 2014). *Wikipedia.* Obtenido de http://es.wikipedia.org/wiki/Acrofobia

Institut Barcelona Psicologia. (s.f.). *http://www.psicologosbarcelona.net/.* Obtenido de http://www.psicologosbarcelona.net/fobias.html?gclid=Cj0KEQjwyMafBRCU7OCRyc2vitsBEiQAKV4H9BcyxGqsb6xHYcW9b8I5T12DrH6f7fKH5_XWNRetQYEaAifG8P8HAQ

Virtualret. (s.f.). Obtenido de http://www.virtualret.com/, http://www.virtualret.com/entornos-virtuales/

Psychology. Obtenido de http://psychology.unt.edu/~tparsons/pdf/Parsons_Virtual%20Reality%20Exposure%20Therapy.pdf

Investigación y ciencia. (s.f.). Obtenido de http://www.investigacionyciencia.es/revistas/mente-y-cerebro/numero/49/la-realidad-virtual-en-psicoterapia-9002

Jsad. (s.f.). Obtenido de http://www.jsad.com/doi/abs/10.15288/jsad.2015.76.620

IEEE. (s.f.). Obtenido de http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1033949

Realidad Virtual/Aumentada

Previ. (s.f.). *http://www.previsl.com/*. Recuperado el agosto de 2014, de http://www.previsl.com/es/ntecno/raumentada.asp, http://www.previsl.com/es/ntecno/rv.asp

Psious. (s.f.). Obtenido de http://www.psious.com/site/offers, http://www.psious.com/site/tech

UploadVr. (s.f.). Obtenido de http://uploadvr.com/river-snapshot-psious-is-immersing-people-in-their-fears-and-improving-lives-along-the-way/

Psychiatrictimes. (s.f.). Obtenido de http://www.psychiatrictimes.com/anxiety/exposure-therapy-anxiety-disorders

Oculus. (s.f.). Obtenido de https://share.oculus.com/app/phobos---phobia-and-anxiety-mgmt---dk2-tech-demo

Indiegogo. (s.f.). Obtenido de https://www.indiegogo.com/projects/phobos-anxiety-management-vr-platform#/story

Oculus

Oculus Rift. (s.f.). Obtenido de https://www.oculus.com/en-us/gear-vr/

Oculus Rift. (s.f.). Obtenido de https://www.oculus.com/en-us/rift/

Reddit. (s.f.). Obtenido de https://www.reddit.com/r/oculus/comments/365e1s/oculus_cv1_is_so_much_more_then_just_resolution/

Riftinfo. (s.f.). Obtenido de http://riftinfo.com/oculus-rift-specs-dk1-vs-dk2-comparison

in2gpu. (s.f.). Obtenido de http://in2gpu.com/2014/08/10/oculus-rift-dk1-vs-dk2/

Sistematología de creación de una aplicación

Wikipedia. (31 de octubre de 2014). *Wikipedia.* Obtenido de http://en.wikipedia.org/wiki/Software_development_process

Wikipedia. (26 de junio de 2014). *Wikipedia.* Obtenido de http://en.wikipedia.org/wiki/Agile_Unified_Process

Otros dispositivos de RV

Virtuix Omni. (s.f.). Obtenido de http://www.virtuix.com/

Windows. (s.f.). Obtenido de https://dev.windows.com/en-us/kinect/hardware?navV3Index=1

Xataka. (s.f.). Obtenido de http://www.xataka.com/realidad-virtual-aumentada/virtuix-omni-el-complemento-para-movernos-y-sudar-en-realidad-virtual

IGN. (s.f.). Obtenido de http://es.ign.com/project-morpheus/94532/feature/project-morpheus-la-realidad-virtual-de-ps4

Playstation Blog. (s.f.). Obtenido de http://blog.es.playstation.com/2015/03/04/gdc-2015-novedades-sobre-project-morpheus/

Programación

Developer Oculus. (s.f.). Obtenido de https://developer.oculus.com/doc/0.4.4-unity/index.html

Unity Docs. (s.f.). Obtenido de http://docs.unity3d.com/Manual/index.html

Annexos

```
4  public class Play : MonoBehaviour {
5      public AudioSource source;
6
7      public AudioClip[] clips;
8
9      private int _lastPos = 0;
10     int pos = 0;
11
12     void Execute()
13     {
14         if (clips.Length > 1)
15         {
16             while (_lastPos == pos)
17             {
18                 pos = Random.Range(0, clips.Length);
19             }
20         }
21         Debug.Log(pos + " " + _lastPos);
22         _lastPos = pos;
23         source.clip = clips[pos];
24         source.Play();
25     }
26 }
```

Fig. 71 - Código del script *Play*

```
public class Play_Voice : MonoBehaviour {

    public AudioSource source;

    public AudioClip[] Voice = new AudioClip[2];

    // Use this for initialization
    void Start () {
        if (Guardadatos.HaEntrado && Application.loadedLevel == 1)
        {
            ExecuteVoice(1);
        }
        else
        {
            ExecuteVoice(0);
        }
    }

    void Update()
    {
        if (source.isPlaying)
        {
            if (name == "OVRPlayerController") GetComponent<OVRPlayerController>().isPlaying = true;
            else GetComponent<FirstPersonController>().isPlaying = true;
            GetComponent<Detectar>().IsPlaying = true;
            if (Guardadatos.HaEntrado && Application.loadedLevel == 1) GetComponent<Detectar>().Entrar = true;
        }
        else
        {
            if (name == "OVRPlayerController") GetComponent<OVRPlayerController>().isPlaying = false;
            else GetComponent<FirstPersonController>().isPlaying = false;
            GetComponent<Detectar>().IsPlaying = false;
        }
    }

    public void ExecuteVoice(int pos)
    {

        source.clip = Voice[pos];
        source.Play();
    }

}
```

Fig. 72 - Código del script *Play_Voice*

```
    // Use this for initialization
    void Start()
    {
        if (GameObject.Find("Linterna"))
        {
            _lantern = GameObject.Find("Linterna");
            _lanternLight = GameObject.Find("Foco").GetComponent<Light>();

            _lamp = GameObject.Find("Farol");
            _lampLight = GameObject.Find("Haz").GetComponent<Light>();
        }
    }
```

Fig. 73 - Función *Start* del script *Detectar*

```
50      void OnTriggerEnter(Collider other)
51      {
52          switch (other.tag)
53          {
54              case "LINTERNA":
55                  _lanternLight.enabled = true;
56                  StartCoroutine(Energy());
57
58                  _lamp.GetComponent<Renderer>().enabled = false;
59                  _lampLight.enabled = false;
60
61                  if (!_lampFirst)
62                  {
63                      GetComponent<Play_Voice>().ExecuteVoice(1);
64                      _lampFirst = !_lampFirst;
65                  }
66
67                  _lantern.GetComponent<Linterna>().enabled = true;
68                  break;
69              case "RECARGA":
70                  StopAllCoroutines();
71                  _lanternLight.intensity = 8;
72                  StartCoroutine(Energy());
73                  break;
74              case "CHECKPOINT":
75                  GameObject.Find("ESC-ALT-FIN").GetComponent<Caer>().Position = other.gameObject.transform.position;
76                  if (!_first)
77                  {
78                      GetComponent<Play_Voice>().ExecuteVoice(1);
79                      _first = !_first;
80                  }
81                  break;
82              case "ESCENA_BLANCA":
83                  StartCoroutine(fade(1));
84                  break;
85              case "ESCENA_ALTURA":
86                  Guardadatos.HaEntrado = true;
87                  StartCoroutine(fade(2));
88                  break;
89          }
90      }
```

Fig. 74 - Función *OnTriggerEnter* del script *Detectar*

```
92      IEnumerator Energy()
93      {
94          yield return new WaitForSeconds(60f);
95          StartCoroutine(TimeBlink(15f, 20f));
96          yield return new WaitForSeconds(30f);
97          StopCoroutine("TimeBlink");
98          StartCoroutine(TimeBlink(8f, 12f));
99      }
100
101     IEnumerator TimeBlink(float min, float max)
102     {
103         while (true)
104         {
105             yield return new WaitForSeconds(Random.Range(min, max));
106             StartCoroutine(Blink());
107             _lanternLight.intensity -= 0.2f;
108         }
109     }
110
111     IEnumerator Blink()
112     {
113         for (var i = 0; i < Random.Range(1f, 2f); i++)
114         {
115             _lanternLight.intensity -= 3;
116             yield return new WaitForSeconds(Random.Range(0.05f, 0.2f));
117             _lanternLight.intensity += 3;
118             yield return new WaitForSeconds(Random.Range(0.05f, 0.2f));
119         }
120     }
```

Fig. 75 - Corrutinas utilizadas para el parpadeo de la linterna

```
4   public class CREATE_PARTICLE : MonoBehaviour
5   {
6
7       public GameObject Particle;
8
9       void Start()
10      {
11          Particle.GetComponent<ParticleSystem>().Stop();
12      }
13
14      void OnTriggerEnter()
15      {
16          Particle.GetComponentInChildren<WakeUp>().WakeStart();
17          Particle.GetComponent<ParticleSystem>().Play();
18
19      }
20
21      void OnTriggerExit()
22      {
23          Particle.GetComponentInChildren<WakeUp>().WakeDown();
24          Particle.GetComponent<ParticleSystem>().Stop();
25      }
26  }
```

Fig. 76 - Código del script *CREATE_PARTICLE*

```
4  public class WakeUp : MonoBehaviour
5  {
6      private Light _light;
7
8      // Use this for initialization
9      void Start ()
10     {
11         _light = GetComponent<Light>();
12     }
13
14     public void WakeStart()
15     {
16         StopAllCoroutines();
17         StartCoroutine(Wake(0.1f));
18     }
19
20     public void WakeDown()
21     {
22         StopAllCoroutines();
23         StartCoroutine(Wake(-0.1f));
24     }
25
26     IEnumerator Wake(float i)
27     {
28         _light.intensity += i;
29         yield return new WaitForSeconds(0.05f);
30         while (_light.intensity > 0 && _light.intensity < 8.0f)
31         {
32             _light.intensity += i;
33             Debug.Log(_light.intensity);
34             yield return new WaitForSeconds(0.05f);
35         }
36     }
37 }
```

Fig. 77 - Código del script *WakeUp*

```
public class Linterna : MonoBehaviour {
    float rotateSpeed = 5.0f;
    float rotateLimit = 45;

    float v;
    float h;

    GameObject personaje;

    // Use this for initialization
    void Start () {
        personaje = GameObject.Find ("OVRPlayerController");
    }

    // Update is called once per frame
    void Update () {
        if(personaje){
            v += -rotateSpeed * Input.GetAxis("Mouse Y");
            h += rotateSpeed * Input.GetAxis("Mouse X");

            v = Mathf.Clamp (v, -rotateLimit, rotateLimit);
            h = Mathf.Clamp (h, -rotateLimit, rotateLimit);

            transform.localEulerAngles = new Vector3(v, h, transform.localEulerAngles.z);
        }
    }
}
```

Fig. 78 - Código del script *Linterna*

```
public class APARECER : MonoBehaviour
{
    private Transform _building;
    private Transform _char;

    // Use this for initialization
    void Start () {
        _building = GameObject.Find("Edificio").transform;
        if (name == "OVRPlayerController") _char = GameObject.Find("OVRPlayerController").transform;
        else _char = GameObject.Find("FPSController").transform;
    }

    void Update()
    {
        var distance = 75-((_building.position - _char.position).magnitude);
        //Debug.Log(distance);
        if (distance <= 25 && distance >= -25)
        {
            _building.position = new Vector3(0, distance, 0);
        }
    }

    void OnTriggerExit()
    {
        _char.position = new Vector3(-_char.position.x, _char.position.y, -_char.position.z);
    }

}
```

Fig. 79 - Código del script *APARECER*

```
4  public class RANDOM_PLAY : MonoBehaviour {
5
6      // Use this for initialization
7      void Start ()
8      {
9          StartCoroutine(PlayAudio());
10     }
11
12     IEnumerator PlayAudio()
13     {
14         while (true)
15         {
16             yield return new WaitForSeconds(Random.Range(40.0f, 60.0f));
17             GetComponent<AudioSource>().Play();
18         }
19     }
20 }
```

Fig. 80 - Código del script *RANDOM_PLAY*

```
4  public class BALANCE : MonoBehaviour {
5
6      void OnTriggerEnter(Collider other)
7      {
8          StartCoroutine(Change(other));
9      }
10
11     void OnTriggerExit(Collider other)
12     {
13         other.GetComponent<Animator>().SetInteger("Balanceo", 0);
14         StopAllCoroutines();
15     }
16
17     IEnumerator Change(Collider other)
18     {
19         while (true)
20         {
21             int rand = Random.Range(1, 4);
22             Debug.Log("Animación: "+ rand);
23             other.GetComponent<Animator>().SetInteger("Balanceo", rand);
24             yield return new WaitForSeconds(5.0f);
25         }
26     }
27 }
```

Fig. 81 - Código del script *BALANCE*

```
public class Caer : MonoBehaviour
{

    private GameObject _character;
    public Vector3 Position;
    public Quaternion Rotation;

    private Image _fade;

    void Start()
    {
        if (GameObject.Find("OVRPlayerController")) _character = GameObject.Find("OVRPlayerController");
        else _character = GameObject.Find("FPSController");
        Position = _character.transform.position;
        Rotation = _character.transform.rotation;
        _fade = GameObject.Find("Panel").GetComponent<Image>();
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.name == _character.name) StartCoroutine(fade());
    }

    IEnumerator fade()
    {
        for (float i = 0; i <= 1.5f; i += 0.1f)
        {
            _fade.color = new Color(0, 0, 0, i);
            yield return new WaitForSeconds(0.01f);
        }

        _character.transform.position = Position;
        _character.transform.rotation = Rotation;

        for (float i = 1; i >= -0.5f; i -= 0.1f)
        {
            _fade.color = new Color(0, 0, 0, i);
            yield return new WaitForSeconds(0.1f);
        }

        _character.GetComponent<Play_Voice>().ExecuteVoice(2);
    }
}
```

Fig. 82 - Código del script Caer

```
4  public class CAIDA : MonoBehaviour
5  {
6      private string _tag;
7      private Collider _collider;
8
9      public GameObject NextGameObject;
10     public float WaitTime;
11
12     void OnTriggerEnter(Collider other)
13     {
14         Debug.Log (other);
15         LoadAnimation(gameObject);
16     }
17
18     void LoadAnimation(GameObject it)
19     {
20         _tag = it.tag;
21
22         switch (_tag)
23         {
24             case "LATERAL":
25                 it.GetComponent<Animator>().SetInteger("Fall", 1);
26                 break;
27             case "INCLINAR":
28                 it.GetComponent<Animator>().SetInteger("Fall", 2);
29                 break;
30             case "INCLINAR_CAIDA":
31                 it.GetComponent<Animator>().SetInteger("Fall", 3);
32                 break;
33             case "INCLINAR_CAIDA_DOBLE":
34                 it.GetComponent<Animator>().SetInteger("Fall", 4);
35                 break;
36         }
37
38         Debug.Log(NextGameObject);
39
40         if (NextGameObject == null) return;
41         StartCoroutine(Wait(NextGameObject));
42         NextGameObject = null;
43     }
44
45     IEnumerator Wait(GameObject other)
46     {
47         yield return new WaitForSeconds(WaitTime);
48         LoadAnimation(other);
49     }
50 }
```

Fig. 83 - Código del script *CAIDA*

```
4  public class NoIsKinematic : MonoBehaviour {
5
6      public AudioSource[] source;
7      public AudioClip[] Sound;
8      private int enter = 0;
9
10     void OnTriggerEnter(Collider other)
11     {
12         GetComponent<Rigidbody> ().isKinematic = false;
13         if (enter == 0)
14         {
15             enter = 1;
16             StartCoroutine (ExecuteSound());
17         }
18     }
19
20     IEnumerator ExecuteSound()
21     {
22         int FRandomNum = Random.Range (0, Sound.Length);
23         int RandomNum;
24         Debug.Log(FRandomNum);
25         source[0].clip = Sound[FRandomNum];
26         source[0].Play();
27         yield return new WaitForSeconds (1.0f);
28         do{
29             RandomNum = Random.Range (0, Sound.Length);
30         }while(FRandomNum == RandomNum);
31         Debug.Log(RandomNum);
32         source[1].clip = Sound[Random.Range(0, Sound.Length)];
33         source[1].Play();
34         yield return new WaitForSeconds(1.0f);
35         enter = 0;
36     }
37 }
```

Fig. 84 - Código del script *NoIsKinematic*

```
MoveScale = 1.0f;

if ( (moveForward && moveLeft) || (moveForward && moveRight) ||
     (moveBack && moveLeft)    || (moveBack && moveRight) )
    MoveScale = 0.70710678f;

// No positional movement if we are in the air
if (!Controller.isGrounded)
    MoveScale = 0.0f;

MoveScale *= SimulationRate * Time.deltaTime;

// Compute this for key movement
float moveInfluence = Acceleration * 0.1f * MoveScale * MoveScaleMultiplier;

// Run!
if (dpad_move || Input.GetKey(KeyCode.LeftShift) || Input.GetKey(KeyCode.RightShift))
    moveInfluence *= 2.0f;
```

Fig. 85 - Código del script *OVRPlayerController* usado para indicar la velocidad de desplazamiento con teclado

```
bool curHatLeft = OVRGamepadController.GPC_GetButton(OVRGamepadController.Button.LeftShoulder);

Vector3 euler = transform.rotation.eulerAngles;

if (curHatLeft && !prevHatLeft)
    euler.y -= RotationRatchet;

prevHatLeft = curHatLeft;

bool curHatRight = OVRGamepadController.GPC_GetButton(OVRGamepadController.Button.RightShoulder);

if(curHatRight && !prevHatRight)
    euler.y += RotationRatchet;

prevHatRight = curHatRight;
```

Fig. 86 - Código del script *OVRPlayerController* usado para rotar el cuerpo con los gatillos del mando de control

```
float leftAxisX = OVRGamepadController.GPC_GetAxis(OVRGamepadController.Axis.LeftXAxis);
float leftAxisY = OVRGamepadController.GPC_GetAxis(OVRGamepadController.Axis.LeftYAxis);

if(leftAxisY > 0.0f)
    MoveThrottle += ort * (leftAxisY * moveInfluence * Vector3.forward);

if(leftAxisY < 0.0f)
    MoveThrottle += ort * (Mathf.Abs(leftAxisY) * moveInfluence * BackAndSideDampen * Vector3.back);

if(leftAxisX < 0.0f)
    MoveThrottle += ort * (Mathf.Abs(leftAxisX) * moveInfluence * BackAndSideDampen * Vector3.left);

if(leftAxisX > 0.0f)
    MoveThrottle += ort * (leftAxisX * moveInfluence * BackAndSideDampen * Vector3.right);
```

Fig. 87 - Código del script OVRPlayerController usado para indicar la velocidad de desplazamiento con el mando de control