



TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

Embodiment: Percepción del cuerpo real del usuario en entornos de Realidad Virtual

Autor

Andrés Molina López

Directores

Pedro Cano Olivares

Miguel Ángel Muñoz García



Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación

—
Granada, 5 septiembre de 2019



ugr

Universidad
de Granada

Embodiment: Percepción del cuerpo real del usuario en entornos de Realidad Virtual

Autor

Andrés Molina López

Directores

Pedro Cano Olivares

Miguel Ángel Muñoz García

Embodiment: Percepción del cuerpo real del usuario en entornos de Realidad Virtual

Andrés Molina López

Palabras clave: ilusión de la mano de goma, realidad virtual inmersivo, *embodiment*, Unity, sensaciones táctiles

Resumen

El objetivo principal de este Trabajo de Fin de Grado (TFG) es el desarrollo de un software que permita al grupo de investigación de Psicofisiología Humana y Salud, perteneciente al Centro de Investigación Mente, Cerebro y Comportamiento de la Universidad de Granada, realizar una simulación del experimento de la ilusión de la mano de goma en un entorno de realidad virtual inmersivo. El objetivo último es constatar si en este tipo de entornos se produce un *embodiment* de la persona con el avatar que maneja.

Para la consecución de este objetivo, se ha creado un software en Unity apoyado en la tecnología de inmersión Oculus Rift y el hardware de virtualización de manos dentro del entorno virtual, Leap Motion. El programa se compone de dos escenas. Una primera escena de adaptación, en la que el usuario participa en un pequeño juego diseñado para que haga uso de las manos, y se acostumbre a la interacción con los objetos. La segunda escena de simulación, consiste en recrear las condiciones experimentales de la “ilusión de la mano de goma”, provocando sensaciones táctiles en la mano real y virtual. Como parte del TFG se evaluó el grado de asimilación de la mano virtual comparado con los resultados obtenidos en una pasación de la “ilusión de la mano de goma”. Los resultados señalaron la eficacia de la mano virtual para generar la sensación de *embodiment*.

Embodiment: Perception of the real body of the user in Virtual Reality environments

Andrés Molina López

Keywords: rubber hand illusion, immersive virtual reality, embodiment, Unity, tactile sensations

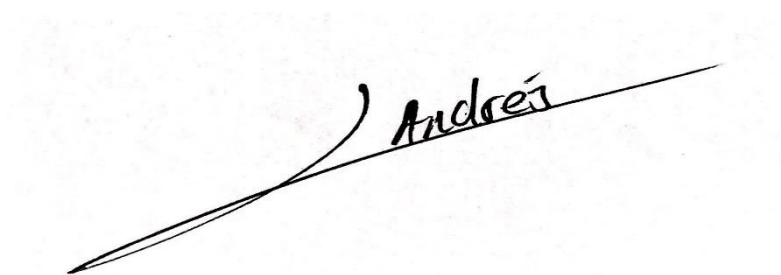
Abstract

The main goal of this “Final Degree Project” (TFG) is the development of a software application that allows the research group of Human Psychophysiology and Health, belonging to the Mind, Brain and Behaviour Research Centre of the University of Granada, to carry out a simulation of the experiment of the rubber hand illusion in an immersive virtual reality environment. The ultimate goal is to verify whether in this type of environments there is an embodiment of the person with the avatar he/she drives.

In order to achieve this goal, a software application has been created in Unity supported by Oculus Rift immersion technology and hand virtualization hardware in virtual environment, Leap Motion. The program consists of two scenes. A first adaptation scene, in which the user participates in a small game designed to make use of the hands, and get used to the interaction with objects. The second simulation scene consists in recreating the experimental conditions of the “rubber hand illusion”, causing tactile sensations in the real and virtual hand. As part of the TFG, the degree of assimilation of the virtual hand was evaluated compared to the results obtained in a “rubber hand illusion” pass. The results indicated the effectiveness of the virtual hand to generate the sensation of embodiment.

Yo, **Andrés Molina López**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 74741577H, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Andrés Molina López

A handwritten signature in black ink, reading 'Andrés', with a long, sweeping horizontal stroke extending to the left.

Granada a 5 de septiembre de 2019.

D. **Pedro Cano Olivares**, Profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

D. **Miguel Ángel Muñoz García**, Profesor Titular del Departamento de Personalidad, Evaluación y tratamiento Psicológico de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Embodiment: Percepción del cuerpo real del usuario en entornos de Realidad Virtual***, ha sido realizado bajo su supervisión por **Andrés Molina López**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 4 de septiembre de 2019.

Los directores:



Pedro Cano Olivares



Miguel Ángel Muñoz García

Agradecimientos

A mi familia por apoyarme en todo momento, y animarme a llegar hasta aquí.

A Pedro y Miguel Ángel, sin los cuales este proyecto no habría sido posible, y a los compañeros del laboratorio, Cristina, Mariola, Guzmán y Antonio, que siempre han estado dispuestos a ayudarme con todas mis dudas y problemas.

Y a todos los compañeros y amigos que he tenido la suerte de conocer a lo largo de estos años, y que han hecho que sean mucho más entretenidos y agradables. Todo lo que me habéis aportado no se puede describir con palabras.

No me habría sido posible llegar hasta aquí sin todos vosotros.

Índice

RESUMEN	6
ABSTRACT	8
1. INTRODUCCIÓN	20
1.1. Motivación	20
1.2. Objetivos	21
1.3. Estructura del trabajo	22
2. PLANIFICACIÓN	23
2.1. Fases del proyecto	23
2.2. Metodología	25
2.3. Presupuesto	25
3. REALIDAD VIRTUAL	27
3.1. ¿Qué es la Realidad Virtual?	27
3.1.1. Requisitos de la realidad virtual	28
3.1.2. Elementos esenciales de la realidad virtual	29
3.1.3. Tipos de realidad virtual	29
3.2. Historia de la realidad virtual	30
3.3. Motores gráficos para desarrollar RV	31
3.3.1. Unity 3D	32
3.3.2. Unreal Engine 4	32
3.3.3. CryEngine 5	33
3.4. Hardware para VR	33
3.4.1. Análisis de HMDs	34
3.4.1.1. Oculus Rift S	34
3.4.1.2. HTC Vive	34
3.4.1.3. PlayStation VR	35
3.4.2. Leap Motion	35
4. EMBODIMENT. ILUSIÓN DE LA MANO DE GOMA	37
4.1. ¿Qué es el <i>embodiment</i> ?	37
4.1.1. Sentido de autolocalización	38
4.1.2. Sentido de voluntad	38
4.1.3. Sentido de propiedad del cuerpo	38
4.2. Ilusión de la mano de goma	38
4.2.1. Procedimiento clásico del paradigma de la RHI	40
4.2.2. Virtualización del paradigma de la RHI	41
4.2.2.1. Estudio piloto realizado	41
5. ANÁLISIS Y DISEÑO DEL SISTEMA	42
5.1. Especificaciones de requerimientos	42
5.1.1. Requerimientos de información	43
5.1.1.1. Fase de entrenamiento	43
5.1.1.2. Fase de simulación	43
5.1.2. Requerimientos funcionales	44
5.1.2.1. Fase de entrenamiento	44
5.1.2.2. Fase de simulación	45
5.1.3. Requerimientos no funcionales	46
5.1.3.1. Requerimientos no funcionales generales	46
5.1.3.2. Fase de entrenamiento	47
5.1.3.3. Fase de simulación	47

5.2. Modelado de casos de uso	47
5.2.1. Descripción de los actores	48
5.2.2. Diagramas de casos de uso	48
5.2.2.1. Subsistema de fase de entrenamiento	48
5.2.2.2. Subsistema de fase de simulación	56
5.2.3. Diagramas de secuencia	62
5.2.3.1. CU-6: Iniciar juego	63
5.2.3.2. CU-18: Leer fichero de órdenes	64
5.2.3.3. CU-19: Ejecutar órdenes	65
5.3. Diseño del fichero de órdenes	66
5.3.1. Parámetros de las órdenes	66
5.4. Diagramas de clase	69
5.4.1. Fase de entrenamiento	70
5.4.2. Fase de simulación	71
6. IMPLEMENTACIÓN	72
6.1. Herramientas utilizadas	72
6.2. Bibliotecas externas	73
6.3. Implementación de funciones	73
6.3.1. Leer duración del juego	74
6.3.2. Leer fichero de órdenes	75
6.3.3. Voltar modelo de la mano	76
7. CONCLUSIÓN	78
7.1. Revisión de los objetivos del proyecto	78
7.2. Resultados obtenidos	79
7.3. Habilidades adquiridas	80
7.4. Vías futuras	80
BIBLIOGRAFÍA	82
APÉNDICE I	86
APÉNDICE II	98

Índice de figuras

1. Diagrama de Gantt con las fases del proyecto	24
2. Espectro en el que se encuentra la realidad mixta	28
3. Logo de Unity 3D	32
4. Logo de Unreal Engine 4	32
5. Logo de CryEngine	33
6. Modelo de Oculus Rift S	34
7. Modelo HTC Vive	34
8. Modelo PlayStation VR	35
9. Leap Motion por dentro	35
10. Dispositivo Leap Motion	35
11. Ilusión de la mano de goma	39
12. Representación del circuito que participaría en la RHI	40
13. Diagrama de casos de uso de la fase de entrenamiento	49
14. Diagrama de casos de uso de la fase de simulación	56
15. Diagrama de secuencia del CU-6	63
16. Diagrama de secuencia del CU-18	64
17. Diagrama de secuencia del CU-19	65
18. Posiciones donde se puede colocar el lápiz con sus correspondientes valores	68
19. Ejemplo de fichero de órdenes	69
20. Diagrama de clases – clase común para ambas fases	69
21. Diagrama de clases de la fase de entrenamiento	70
22. Diagrama de clases de la fase de simulación	71
23. Función para leer la duración del juego	74
24. Función para leer el fichero de órdenes	75
25. Función para crear un sprite a partir de la dirección de una imagen	76
26. Función para voltear el modelo de la mano	77

Índice de tablas

1. Presupuesto del proyecto según los recursos necesarios	26
2. Rendimiento cualitativo de los distintos tipos de VR para determinadas características	30
3. Propositiones para el SoE hacia un cuerpo B	37
4. Definición del actor 1 – Participante	48
5. Definición del actor 2 – Experimentador	48
6. Definición del actor 3 – Temporizador	48
7. CU-1 Desplegar botón mano	49
8. CU-2 Crear proyectil	50
9. CU-3 Lanzar proyectil	50
10. CU-4 Avanzar en la información	51
11. CU-5 Retroceder en la información	51
12. CU de extensión	52
13. CU-6 Iniciar juego	52
14. CU-7 Desplegar menú	53
15. CU-8 Introducir duración del juego	53
16. CU-9 Crear enemigo de ejemplo	54
17. CU-10 Generar enemigo	54
18. CU-11 Quitar enemigo	55
19. CU-12 Finalizar juego	55
20. CU-13 Aumentar tamaño mano derecha	57
21. CU-14 Reducir tamaño mano derecha	57
22. CU-15 Aumentar tamaño mano izquierda	58
23. CU-16 Reducir tamaño mano izquierda	58
24. CU-17 Voltar mano izquierda	59
25. CU-18 Leer fichero de órdenes	60
26. CU-19 Ejecutar órdenes	61
27. CU-20 Mostar menú	61
28. CU-21 Crear lápiz vibrador	62
29. CU-22 Destruir lápiz vibrador	62
30. Consecución de los objetivos del proyecto	78

Capítulo 1

“What is real? How do you define ‘real’? If you’re talking about what you can feel, what you can smell, what you can taste and see, then ‘real’ is simply electrical signals interpreted by your brain.”

– Morpheus, The Matrix

Introducción

La realidad virtual, un mundo de infinitas posibilidades, citando a Ivan E. Sutherland *“con la adecuada programación [...] podría ser el Mundo de las Maravillas en el que Alicia entró”* [1]. Medio siglo después de que su proyecto conocido como *“La Espada de Damocles”* viese la luz, siendo este el primer casco de realidad virtual (HMD) de la historia, esta visión que planteó, no se ha perdido a día de hoy. Es más, en la última década, sectores de todo tipo están apostando por esta tecnología, ya que permite explorar campos que de otra manera no sería posible o tendría costes mucho más elevados.

En la actualidad, la RAE define el término realidad virtual como la *“representación de escenas o imágenes de objetos producida por un sistema informático, que da la sensación de su existencia real”* [2]. Aunque ya es posible representar la realidad con un alto nivel de detalle y mostrarlo a través de un HMD incluso sin cables, aún queda mucho por investigar en lo referente a las sensaciones que tienen las personas que usan estos dispositivos y como de integradas se sienten en el entorno que las rodea virtualmente. Recientemente, gracias a la comercialización de gafas de realidad virtual por compañías como Oculus Rift o HTC, que hacen que esta tecnología quede al alcance de la mano de los usuarios, y a la incorporación de los dispositivos hápticos, que hacen posible añadir el sentido del tacto a los entornos de realidad virtual inmersiva, se puede ver que aunque aún quede camino para llegar a los niveles de inmersión total que nos presentan películas como *The Matrix* o *Ready Player One*, una cosa sí que se puede afirmar, y es que esta vez, la realidad virtual ha vuelto para quedarse y afianzarse como un nuevo paradigma en la manera de interactuar con los ordenadores.

1.1 Motivación

La mejora de los gráficos por ordenador acompañada de las sucesivas mejoras en los componentes que conforman los ordenadores, aumentando su rendimiento y disminuyendo su tamaño, ha hecho posible que la realidad virtual vuelva a ser un tema de interés y al orden del día, lo que ha ocasionado que gran cantidad de sectores comiencen a utilizarla para diversas tareas. Para algunos de estos sectores como el ocio, estas mejoras en la parte hardware y software son suficientes para ofrecerle al usuario una sensación de inmersión que le permita entretenerse en mayor o menor medida. Pero para otros sectores como la salud y en especial, en el campo de la neurorrehabilitación, estas mejoras son indispensables para hacer que el usuario se sienta integrado dentro del mundo virtual, es decir, que le produzca la sensación de que el avatar que maneja en la realidad virtual es en realidad su propio cuerpo.

Esta sensación de estar dentro de la realidad virtual, lo que podría considerarse la fusión del cuerpo con el avatar, es lo que se conoce como *embodiment*¹. La definición de este término es compleja, pero para los objetivos de este trabajo se va a utilizar la definición de *Sense of Embodiment (SoE)* dada por Kiltner, Groten y Slater: “*SoE toward a body B is the sense that emerges when B’s properties are processed as if they were the properties of one’s own biological body.*”² [3]. El interés en estudiar si este fenómeno se produce dentro de un entorno de realidad virtual no es nuevo. Para su estudio se ha utilizado el paradigma de la “ilusión de la mano de goma” (en posteriores capítulos se verá en que consiste en detalle), pero hasta la fecha no hay ningún software que permita replicar este paradigma ni ampliar el conocimiento que se tiene sobre el tema en un entorno virtual. Es por esto, que el equipo de investigación de Psicofisiología Humana y Salud perteneciente al Centro de Investigación Mente, Cerebro y Comportamiento (CIMCYC) de la Universidad de Granada, requirió de un programa capaz de emular el paradigma de la ilusión de la mano de goma lo más fielmente posible, con el propósito de poder comparar los resultados con el experimento tradicional. De este modo se podría comprender los cambios psicológicos asociados a la sensación de *embodiment*, y cuyos resultados permitirían avanzar en la neurorrehabilitación de problemas cognitivos, motores y dolorosos, causados por patologías como el síndrome del miembro fantasma o la fibromialgia.

1.2 Objetivos

Como ya se ha mencionado anteriormente, el desarrollo de este proyecto nace de la necesidad de utilizar un software capaz de recrear el paradigma de la mano de goma en un entorno de realidad virtual. Por lo tanto, el objetivo principal es la implementación de un programa capaz de emular dicho paradigma en un entorno virtual, que sea flexible para ser usado en diferentes estudios y que cubra las necesidades de los protocolos experimentales del grupo de investigación.

Los objetivos secundarios son:

- Integración de Oculus Rift en el programa como visor del entorno de realidad virtual.
- Integración de Leap Motion para virtualizar las manos dentro del programa y así conseguir una mayor sensación de *embodiment*.
- Diseño e implementación de un pequeño juego que se use a modo de escena de entrenamiento para que el usuario se adapte a la realidad virtual.
 - Introducción del tiempo de duración del juego por teclado para controlar la duración de la fase de adaptación.
 - Aparición de los enemigos de manera aleatoria por el espacio disponible pero dentro del campo de visión y avisando de su lugar de aparición.
 - Asignación de un modelo aleatorio a los enemigos entre una lista de modelos.
 - Adición de un sonido aleatorio a la aparición del enemigo entre una lista de sonidos preseleccionados.
- Diseño e implementación de la escena en la cual se desarrolla el experimento de la mano de goma.

¹ El diccionario de Cambridge define el término *embodiment* como “*algo o alguien que representa una cualidad o idea exactamente*” aunque la traducción al español más correcta en el contexto del proyecto sería pertenecía, en el sentido de estar contenido en algo.

² Traducción de la definición citada: “El sentido de *embodiment* (pertenencia) hacia un cuerpo B es el sentido que surge cuando las propiedades de B se procesan como si fueran las propiedades del propio cuerpo biológico”.

- Capacidad para leer las instrucciones desde un fichero localizado en una carpeta específica del ordenador.
- Flexibilidad para decidir las imágenes que se quieran enseñar al usuario.
- Capacidad para enviar señales por el puerto de serie a los controladores de Arduino.
- Posibilidad de agrandar o reducir el tamaño de los modelos de las manos.
- Posibilidad de hacer que la mano izquierda se vea de manera invertida a como se tiene en el mundo real
- Permitir cambiar de una escena a otra a decisión del investigador que controla el programa desde fuera de la realidad virtual.

1.3 Estructura del trabajo

El documento aquí presentado, está estructurado en una serie de capítulos que van a ir inicialmente dando información sobre los distintos temas que abarca este proyecto, desde la realidad virtual hasta cuestiones de neuropsicología referentes al *embodiment*. Posteriormente se tratará el diseño e implementación de la aplicación desarrollada, y por último se presentan los resultados de un estudio piloto sobre la efectividad del software para inducir la sensación de la mano de goma, comparado con el paradigma tradicional. Además, también se tratará brevemente perspectivas futuras en el uso del programa creado y sus aplicaciones. Así, el presente trabajo se articula en 7 capítulos que cubren los objetivos señalados y que son:

En el capítulo 2 se expone la planificación y metodología seguida para realizar el proyecto y se hace una estimación del presupuesto que habría sido necesario para llevarlo a cabo.

El capítulo 3 muestra algunas de las diferentes alternativas actuales para desarrollar en realidad virtual. Se incluyen algunos de los diferentes HMDs disponibles en el mercado, como los motores gráficos más usados para el desarrollo de los entornos. Además, se hace una pequeña introducción sobre Leap Motion.

El capítulo 4 explica el concepto de *embodiment*, y su relación el paradigma de la mano de goma. Se expondrán los aspectos sensoriales y los resultados neurológicos de los que se dispone sobre el sentido de pertenencia.

En el capítulo 5 se hace un análisis detallado de los requisitos que tiene que cumplir el programa que se va a desarrollar, además de hablar sobre otras particularidades del diseño desde el punto de vista de la ingeniería del software.

En el capítulo 6 se especifican las herramientas y bibliotecas externas que ha sido necesario utilizar para poder llevar a cabo la implementación del programa, y se explica el proceso de implementación de aquellas funciones cuyo código es complejo, comentando sus particularidades.

Finalmente, en el capítulo 7 se repasa la consecución de los objetivos señalados en el proyecto. A su vez se exponen los resultados obtenidos en el estudio piloto y se comentan posibles vías que se podrían seguir a partir de este trabajo, aprovechando el software realizado.

Capítulo 2

“To succeed, planning alone is insufficient. One must improvise as well.”

— Isaac Asimov, *Foundation*

Planificación

En este capítulo se van a exponer las fases en las cuales se ha dividido el proyecto, así como la metodología de desarrollo utilizada para la implementación del software. Finalmente, se hará una estimación del coste total del trabajo, en base a los recursos necesarios para su desarrollo.

2.1 Fases del proyecto

El desarrollo del proyecto se divide en 5 fases principales (Figura 1):

1. **Reuniones iniciales:** para comenzar se han tenido una serie de reuniones con ambos tutores, de cara a plantear el proyecto, entender las necesidades que se requieren, y conocer a los miembros del equipo de investigación.
2. **Recopilación de información:** una vez conocido el tipo de software solicitado, se ha realizado una revisión en busca de proyectos similares, para conocer el estado actual del tema. Además, se ha recopilado información acerca de los aspectos psicológicos, para comprender mejor el proyecto, con el objetivo de anticiparse o adaptarse a las necesidades que pudieran surgir a lo largo del desarrollo.
3. **Desarrollo del proyecto:** esta es la fase principal y a la que más tiempo se le ha dedicado. Siguiendo el orden cronológico en el que se ha desarrollado, se puede dividir en dos fases independientes, correspondientes a las escenas del programa:
 - 3.1. **Desarrollo del juego**
 - 3.2. **Desarrollo de la simulación**Aunque podemos agruparlas en una sola fase porque ambas escenas se enlazan entre ellas, ya que funcionan en una sola aplicación. No es posible establecer una fase clara de análisis, diseño, desarrollo y prueba dentro de estas fases, ya que no se ha seguido una metodología tradicional de cascada. En la siguiente sección se explicará el marco de desarrollo seguido para trabajar en ellas.
4. **Obtención de resultados:** en esta fase es en la que se ha realizado el estudio piloto del software comprobando así su capacidad de *embodiment*. A su vez, se han analizado los datos obtenidos, comparándolos con los que ya se disponían del experimento tradicional.
5. **Redacción de documentos:** finalmente, se ha confeccionado este documento con diversas anotaciones que se han ido tomando a lo largo de las fases anteriores. Adicionalmente, se ha elaborado un manual de usuario para que quede a disposición del equipo de investigación en todo momento.

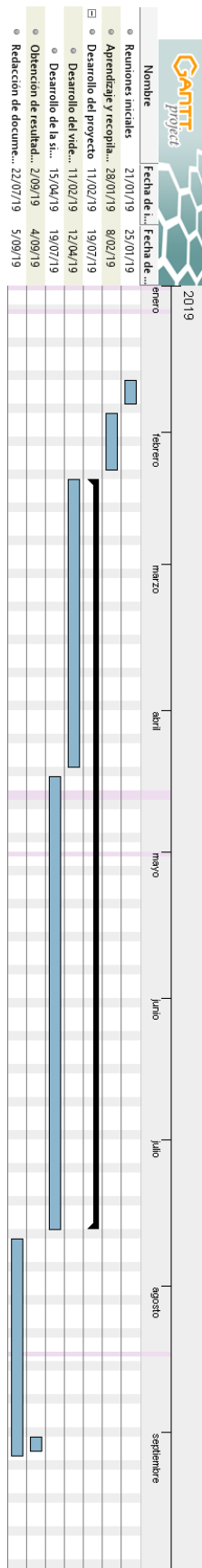


Figura 1: Diagrama de Gantt con las fases del proyecto

2.2 Metodología

Como se ha mencionado, la metodología usada para el desarrollo del proyecto no ha sido en cascada. Esto se debe a que durante las reuniones iniciales se vio que el equipo, aunque sabía cuál era el objetivo final, no tenía claro que requisitos y funciones querían para ello. Por lo tanto, se optó por la metodología de desarrollo ágil Scrum, que permite integrar al cliente (el equipo en este caso), dentro del proyecto e ir haciendo entregas parciales del mismo.

Scrum se basa en un modelo de desarrollo incremental, en el cual los ciclos de vida son cortos (sprints), para que así se puedan presentar los resultados al cliente y se puedan tomar las decisiones a partir de ellos [4]. Esto hace que la flexibilidad y la adaptación del proyecto sean mayores, ya que permite al cliente redirigir el proyecto en función de los requisitos completados, que le ayudan a comprender mejor el producto. A su vez, permite aumentar la productividad y la calidad ya que después de cada sprint se analiza lo que en él ha ocurrido, de cara a planificar mejor el siguiente [5].

La duración de los sprints para el desarrollo del proyecto, inicialmente eran de una semana. Pero tras analizar un par de ellos, y además debido a restricciones temporales para hacer las reuniones de revisión y planificación de estos, se decidió reajustar la duración a dos semanas. Como herramienta de apoyo para establecer el sprint backlog (tareas que se tienen que realizar durante un sprint) de cada iteración, se ha utilizado el material proporcionado por el profesor Luis Castillo Vidal para la asignatura Desarrollo Basado en Agentes. Este material es una hoja de cálculo en la cual además de establecerse la duración del sprint y su correspondiente sprint backlog, se genera un gráfico sprint burndown que ayuda a organizar el esfuerzo restante del sprint entre las distintas tareas.

2.3 Presupuesto

Para calcular el presupuesto del proyecto (Tabla 1) se va tener en cuenta el material que ha sido necesario para el desarrollo del mismo, tanto hardware como software, así como el coste del personal, es decir, el empleo de un ingeniero del software / desarrollador / programador.

En lo referente al hardware utilizado, el equipo en el que se ha trabajado está equipado con un i7-7700, 32 GBs de memoria RAM, y una CPU NVIDIA GTX 1080 con 8 GBs de memoria GDDR 5X. Estos requisitos son necesarios para el correcto funcionamiento del casco de realidad virtual Oculus Rift que ha sido utilizado. Adicionalmente, el controlador del guante que estimula la mano del usuario, como es un diseño específico para poder realizar el experimento, se ha hecho a partir de una placa de Arduino UNO, pero ha sido necesario el trabajo de un ingeniero en electrónica para hacer el diseño y crear el producto final. El conversor de señales para el polígrafo también es una placa de Arduino Uno. Por último, también hay que contar con un dispositivo Leap Motion para virtualizar las manos.

En cuanto al software, se han intentado reducir todos los gastos posibles, haciendo uso de las licencias gratuitas de los programas que se han requerido, y buscando librerías gratuitas de Unity siempre que fuese necesario añadir alguna funcionalidad o modelo al programa. Sin embargo, debido al requisito de utilizar un modelo de manos lo más realista posible para Leap Motion, y la escasez de estos en la tienda de Unity, ha sido inevitable tener que comprar el paquete Leap Motion Realistic Hands disponible en Unity Asset Store.

Finalmente, en la parte que atañe al coste de personal, solo hay que considerar el empleo de un ingeniero de software, debido al carácter individual de proyecto. Para estimar este gasto se han extraído los datos de payscale³, página en la que se puede consultar el salario medio anual de un ingeniero del software / desarrollador / programador en España a fecha de mayo de 2019. Como la duración del proyecto ha sido de 7 meses y el periodo de trabajo diario ha sido el equivalente a media jornada, se tendrá solo en cuenta la parte porcentual $((\text{salario} * 0,58) / 2)$.

Tabla 1: Presupuesto del proyecto según los recursos necesarios

Recurso	Coste
Ordenador con i7-7700 y NVIDIA GTX 1080	1300€ aprox.
Arduino UNO	26,78€
Guante estimulador + mano de obra (ingeniero electrónico) ⁴	200€ + 35577€
Oculus Rift	500€
Leap Motion	95,59€
Leap Motion Realistic Hands asset	26,79€
Ingeniero de software / desarrollador / programador	7626,5€
TOTAL	45352,66€

³ https://www.payscale.com/research/ES/Job=Software_Engineer_%2F_Developer_%2F_Programmer/Salary

⁴ https://www.payscale.com/research/ES/Job=Electrical_Engineer/Salary

Capítulo 3

“Everything’s science fiction until someone makes it science fact.”

— Marie Lu, Warcross

Realidad Virtual

En anteriores capítulos ya se han introducido algunos aspectos de la realidad virtual, ya que es uno de los pilares fundamentales de este proyecto. En este capítulo se va a dar una definición más técnica, remarcando sus diferencias con otros tipos de realidades, y viendo que tipos de realidades virtuales se distinguen. Además, también se hará una revisión de su historia y estado actual, así como de los medios que hay para trabajar con ella. Finalmente se introducirá qué es el dispositivo Leap Motion en detalle, por qué es interesante para este trabajo, y qué proyectos se han hecho con él.

3.1 ¿Qué es la Realidad Virtual?

En el primer capítulo se dio la definición que ofrece la RAE acerca de la realidad virtual (VR), pero a un nivel más técnico es una definición falta de precisión. Así, cuando se intenta definir este término, se tiende a separar las palabras que lo componen y dar la definición de ambas. Y, aunque, dar la definición de “virtual” es sencillo, dar la definición de “realidad” no lo es en ningún sentido. Por lo tanto, para el propósito de este proyecto, y cogiendo la definición desde un punto de vista más orientado a la materia, se puede utilizar la definición que dan Sherman y Craig: *“A medium composed of interactive computer simulations that sense the participant’s position and actions, providing synthetic feedback to one or more senses, giving the feeling of being immersed or being present in the simulation.”*⁵ [6].

A parte de definir la VR, también es importante, dar una definición de la realidad aumentada (AR), ya que, en la actualidad debido al mal uso de los términos en los medios de comunicación, estos suelen mezclarse. Aunque esto tampoco termina de ser incorrecto porque gracias a los avances en la tecnología se ha llegado a desarrollar una mezcla entre ambas, conocida como realidad mixta (MR). Por lo tanto, la AR se define como una “combinación de lo real con objetos virtuales en un entorno real; alinea objetos reales y virtuales unos con otros y se ejecuta de forma interactiva, en tres dimensiones, en tiempo real.” [10]. Por otro lado, para definir la MR tenemos que coger el espectro de posibilidades que hay entre la VR y la AR (Figura 2), entonces en ese rango es donde se encuentra este tipo de realidad. Así, la MR representa los límites del mundo real dentro de un entorno virtual y, además, añade elementos como a esa representación, que realmente no existen en el mundo real. [11]

⁵ Traducción de la definición citada: “Un medio compuesto de simulaciones interactivas por ordenador que detectan la posición y las acciones del participante, proporcionando retroalimentación sintética a uno o más sentidos, dando la sensación de estar inmerso o presente en la simulación”.

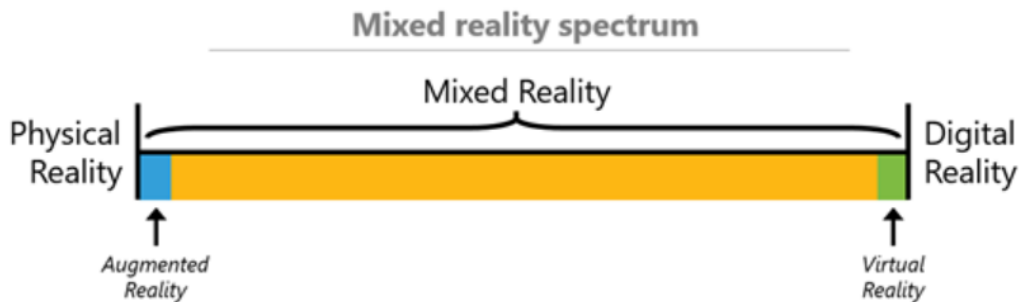


Figura 2: Espectro en el que se encuentra la realidad mixta (MR) [11]

3.1.1 Requisitos de la realidad virtual

Una vez hecha la distinción entre los distintos tipos de realidades (VR/AR/MR), se va a profundizar en los requisitos que tiene que cumplir la VR. Esto no se refiere a los requisitos técnicos que tiene que tener un ordenador para poder hacer que funcionen entornos de VR de una manera fluida, si no a los requisitos que tendría que tener una VR para que se comportase como el mundo real [12]. La respuesta a esta pregunta fue: *“se debe asumir el **constante proceso de información**: ese proceso de información funciona de la misma manera en todos los mundos”* [12]. Además, también existen los siguientes requisitos:

- **Asignaciones de procesamiento finitas:** *“El procesamiento que crea una VR que se comporta como nuestro mundo asigna su procesamiento en cantidades finitas.”*. Esta definición hace referencia al hecho de que se cree que cada cantidad de materia, energía, tiempo y espacio tiene una capacidad de información finita. [12]
- **Autonomía:** *“Una vez comenzado, una VR que se comporta como nuestro mundo tiene que funcionar por su cuenta sin ninguna información de entrada más.”* [12]
- **Auto registro consistente:** *“Una VR que se comporta como nuestro mundo debe “registrarse” consistentemente con los “observadores” internos”*. Esto significa a que cuando se ve la realidad, esta interacciona con nosotros en el mismo mundo, no con un observador externo como hacen las simulaciones por ordenador. Por esto para a una VR “registrarse” significa que sus interacciones internas deben ser consistentes con respecto a cada “observador” local. [12]
- **Calculabilidad:** *“Una VR que se comporta como nuestro mundo debe ser calculable en todo momento”*. Una fuente de procesamiento finito debe asegurar que ningún cálculo tiende a infinito. [12]

Además de estos requisitos, Bierbaum y Just señalaron los 3 requisitos primarios para desarrollar un sistema de VR:

- **Rendimiento:** *“La efectividad de los entornos inmersivos necesita un alto ratio de refresco y baja latencia. Un bajo rendimiento no solo es un inconveniente para el usuario si no que además puede causar efectos desagradables como desorientación y mareo. Por lo tanto, un sistema de VR debería poder aprovechar todos los recursos disponibles del sistema.”* [13]
- **Flexibilidad:** el entorno debe poder adaptarse a distintas configuraciones hardware y software de manera que se puedan usar diferentes aplicaciones y no esté solo limitado a su ámbito de utilidad. [13]
- **Facilidad de uso:** *“El sistema desarrollado debería ser fácil de configurar y de aprender.”* [13]

3.1.2 Elementos esenciales de la realidad virtual

Además de requisitos para la VR, también existen una serie de elementos que son esenciales para experimentar la VR. Estos han sido descritos por Sherman y Craig [8] y son los siguientes:

- **Participante/s:** es el elemento más importante, ya que es la persona que va a vivir la experiencia de VR. Ya que es una experiencia en parte subjetiva, distintos participantes pueden tener distintas experiencias incluso dentro del mismo entorno de VR. [8]
- **Creador:** es la persona o grupo de personas que implementan la aplicación para que sea experimentada por los participantes. Es esencial no solo porque, literalmente, hace la aplicación, si no porque el esfuerzo conjunto entre este y los participantes es lo que realmente va generar la experiencia. [8]
- **Mundo virtual:** *“Una descripción de una colección de objetos en un espacio y las reglas y relaciones que rigen dichos objetos.”*. Es como el montaje de una película, el guion describe como son los actores, escenarios y demás elementos, y establece las reglas y relaciones entre ellos. Así, ver una película se puede traducir como experimentar el mundo virtual de esta [8].
- **Inmersión:** *“Sensación de estar en un entorno; puede ser puramente un estado mental o puede lograrse mediante medios físicos: la inmersión física es una característica definitoria de la VR; la inmersión mental es un objetivo principal en la mayoría de creadores de contenido”*. El hecho de definir este término lleva directamente a la necesidad de hablar sobre la *retroalimentación sensorial*, ya que es crítica para la inmersión física. Así, los sistemas de VR proporcionan *retroalimentación sensorial* directa a los participantes basada en su posición física. En la mayoría de los casos esta retroalimentación es visual, aunque también puede ser a través de dispositivos hápticos, o de sonido. [8]
- **Interactividad:** este elemento se basa en que el entorno debe responder a las acciones del usuario para que la VR se vea auténtica. Es decir, que el usuario no solo tenga la capacidad de interactuar con objetos si no también que pueda moverse dentro del mundo y que consiga nuevos puntos de vista al mover la cabeza [8].

3.1.3 Tipos de realidad virtual

Existen distintos tipos de VR, que se han ido desarrollado a lo largo su historia. Por ello, es importante saber cuáles son los diferentes tipos de VR que se encuentran a día de hoy.

- **No inmersiva:** como su nombre indica es la menos inmersiva y, además la menos sofisticada, por lo que también es la más barata. En este tipo de VR se muestra un mundo virtual a través de un monitor, y se permite interaccionar con él mediante teclado y ratón, aunque se pueden utilizar otro tipo de dispositivos. Este tipo de VR es muy utilizado en videojuegos y programas de diseño como los de AutoCAD. [14]
- **Semi-inmersiva:** da un mayor nivel de inmersión, aunque mantiene la simplicidad del tipo no inmersivo, ya que se basa en utilizar grandes pantallas o salas como es el caso del sistema CAVE. [14]
- **Inmersiva:** es el tipo que produce mayor sensación de inmersión en el entorno virtual. Para ello se basa en el uso de HMDs, y demás dispositivos de posicionamientos y sensores. Es en este tipo de VR en la que el usuario se siente más ligado al entorno virtual, ya que interactúa con él de una forma más directa. [14]

Cabe destacar también la **telepresencia** debido a que en algunos documentos se ha clasificado como un tipo de VR. Sin embargo, no está clara su pertenencia a las clases de VR, ya que esta se basa en el manejo de un robot u otro dispositivo a distancia.

Por último, es importante conocer el rendimiento de los diferentes tipos de VR en las características principales (Tabla 2) a tener en cuenta a la hora de desarrollar una aplicación de este tipo. Ya que, por ejemplo, si se quiere hacer un programa para ver el interior de un edificio, y se pretende que tenga un alto nivel de detalle y que el movimiento sea fluido, interesará más hacerlo en VR no inmersiva o semi-inmersiva, ya que tienen mayor resolución y menor retardo que la VR inmersiva [15].

Tabla 2: Rendimiento cualitativo de los distintos tipos de VR para determinadas características [15]

Qualitative Performance			
Main Features	Non- Immersive VR (Desktop)	Semi-Immersive VR (Projection)	Full Immersive VR (Head-coupled)
Resolution	High	High	Low - Medium
Scale (perception)	Low	Medium - High	High
Sense of situational awareness (navigation skills)	Low	Medium	High
Field of regard	Low	Medium	High
Lag	Low	Low	Medium - High
Sense of immersion	None - low	Medium - High	Medium - High

3.2 Historia de la realidad virtual

Los comienzos de la VR, aunque se pueden remontar a las primeras representaciones visuales, podrían establecerse en 1838 con los estudios de Sir Charles Wheatstone sobre el fenómeno de la estereopsis. En estos estudios se demostraba que el cerebro puede procesar dos imágenes bidimensionales muy similares una en cada ojo para crear un único objeto en tres dimensiones, dando sensación de profundidad e inmersión [7].

Casi 100 años más tarde, en 1929, Edwin Link desarrolló un simulador de vuelo, que permitía enseñar a un piloto a volar, en una posición estática dentro de una sala. Estos simuladores están considerados como una forma temprana de VR, ya que como su nombre indica, solo crean un entorno simulado [8]. Un par de décadas después, en los años 50, cabe destacar el *Sensorama* desarrollado por Morton Heiling. Este aparato era un sistema de visualización de experiencia multimodal que estimulaba los sentidos del usuario mediante sonido, olor, vibración y viento, para hacer que la persona sintiera una sensación de inmersión

total en lo que estaba viendo. Poco después, Heiling presentó la Máscara Teleférica, que tenía bastante similitud con los HMDs que se desarrollaron posteriormente, así que podría considerarse el primer prototipo de estos [7][8].

En 1965, Ivan E. Sutherland explica el concepto de *The Ultimate Display*. Dispositivo que permitía recrear la realidad e incluso moldearla para que no se ajustase a las leyes físicas, y que además dejaba al usuario interactuar con los objetos de ese mundo. Así, en 1968, presenta su HMD (La Espada de Damocles) siendo el primero en estar conectado a un ordenador y utilizando los gráficos de este para recrear los objetos que se veían a través de él [7][8].

Sin embargo, no es hasta 1989, cuando VPL Research (Virtual Programming Languages), empresa fundada por Jaron Lanier para crear un lenguaje visual de programación, lanza al mercado el primer sistema de VR, acuñando de esta manera el término *realidad virtual* propiamente dicho. De este modo, se empieza a utilizar el término, para abarcar todo lo que implica esta disciplina dentro de la computación [8].

En 1992, se presenta la VR estacionaria como una alternativa al paradigma de los HMDs. La tecnología para este paradigma es una sala cúbica con proyectores apuntando al suelo, al techo y a las diferentes paredes, de tal manera que según el usuario se mueve en ella, se van ajustando las perspectivas que se muestran en estas [9]. El nombre que se le dio fue “CAVE” y fue desarrollada por el Laboratorio de Visualización Electrónica, de la Universidad de Illinois. A lo largo del resto de esta década y comienzos de la siguiente, se van haciendo sucesivos avances en los sensores de posicionamiento, respuestas hápticas (vibración en los mandos), tarjetas gráficas para mejorar el realismo y velocidad de respuesta de los entornos virtuales, y en el sistema CAVE. En este periodo cabe destacar 2006, año en el que Nintendo lanza su consola Wii, siendo esta el primer producto masivo en usar posicionamiento físico para mundos generados por ordenador. Y 2010, cuando Microsoft publica Kinect, proveyendo así a la comunidad de usuarios de VR con un sistema de posición de bajo coste. Sin embargo, no es hasta 2012 cuando se crea una nueva ola de interés sobre el tema, debido a la campaña de Kickstarter de Oculus Rift, que lanza Palmer Luckey y cuyo objetivo se consigue en menos de 24 horas. A partir de aquí, todas las compañías tecnológicas importantes se lanzan a la carrera de la VR sacando sus propios HMDs. Así, en 2014 Facebook compra Oculus VR y lanza el segundo prototipo de HMD de la marca, el DK-2. Google presenta sus *Google Cardboard* que es una versión DIY (*do it yourself*) de gafas de VR para smartphones Android. Samsung anuncia las *Samsung Gear VR*, que también es un HMD basado en teléfonos para llevar el sistema de posicionamiento, procesamiento y pantalla. Un año más tarde, Valve se junta con HTC para presentar las *HTC Vive*, siendo un HMD con un sistema de posicionamiento por sensores. Finalmente, en 2016, se llega la entrada definitiva de productos de VR al mercado, estando en primera línea, las Oculus Rift, los HTC Vive, y las PlayStation VR de Sony diseñadas para la consola PlayStation 4.

En la actualidad, se encuentran una gran variedad de dispositivos conviviendo en el mercado, y cada día las empresas siguen mejorando los sistemas para hacerlos más potentes y, a su vez, más asequibles.

3.3 Motores gráficos para desarrollar VR

Los motores gráficos son *frameworks* que proporcionan un motor de renderizado, así como detectores de colisiones, y manipuladores de físicas, entre otras características. Estos motores han ido ganando popularidad en los últimos años, ya que permite a los desarrolladores centrarse en la creación de los videojuegos sin necesidad de preocuparse de la programación de detalles que supondrían un gran coste.

A día de hoy, existen una gran cantidad de motores disponibles por internet, se van solo a considerar los que permiten el desarrollo en VR. Además, otro factor a tener en cuenta es que permita desarrollar en él con una licencia gratuita ya que, en la mayoría de los casos, estos motores son utilizados por las propias empresas que los crean, para desarrollar sus propios videojuegos, por lo que o los mantienen en privado, o cobran una licencia por su uso. Bajo estas dos restricciones y basándose en su popularidad en internet, se encuentran las siguientes tres claras opciones a la hora de desarrollar una aplicación de VR:

3.3.1 Unity 3D

Aunque lanzado inicialmente en 2005 para el sistema operativo Mac OS X, en la actualidad Unity 3D (Figura 3) es el entorno de desarrollo integrado (IDE) más compatible y famoso del mundo. Unity tiene un interfaz simple que permite desarrollar experiencias tanto 2D como 3D, así como aplicaciones XR, es decir, de AR y VR. El lenguaje de programación que utiliza para dar control total sobre los objetos es C#, el cual está estandarizado en la industria del videojuego y además tiene una curva de aprendizaje más sencilla que otros lenguajes como C++. También cabe destacar que en la actualidad es la IDE compatible con más sistemas (móvil, escritorio, consola, web, etc...) y que posee una gran cantidad de documentación disponible, lo que facilita comenzar a aprender a utilizarlo. [16][17]



Figura 3: Logo de Unity 3D

A continuación, se muestran algunos de los pros y contras más relevantes de esta IDE:

- Pros:
 - Soporte en una gran cantidad de plataformas.
 - Gran cantidad de documentación y una comunidad muy activa.
 - Curva de aprendizaje fácil.
 - Mercado repleto de *assets*⁶ debido a su gran popularidad.
- Contras:
 - La documentación de algunas características está desfasada.
 - Su código fuente no está abierto para que los usuarios lo personalicen.

3.3.2 Unreal Engine 4

Unreal Engine (Figura 4) fue presentado inicialmente en 1998 con su juego en primera persona de nombre homónimo. En la actualidad, y desde 2015, se encuentra en su cuarta versión (UE4). Este motor se caracteriza por el alto nivel de calidad de gráficos que posee, lo que lo ha llevado a ser uno de los más utilizados a la hora de desarrollar juegos AAA. Tanto su código fuente, como sus scripts para desarrollar con él, están hechos de manera íntegra en C++. Aunque a la hora de desarrollar, también permite el uso de un sistema gráfico creado por la propia compañía, conocido como Blueprint. Los Blueprints, permiten implementar comportamientos para los objetos sin necesidad de tener nociones de programación en C++, ya que se basa en un sistema de caja negra, donde la persona simplemente conecta cajas con funcionalidades, unas con otras [18].



Figura 4: Logo de Unreal Engine 4

⁶ Un *asset* es una representación de cualquier ítem que pueda ser utilizado en un proyecto. Estos pueden venir de un archivo externo, y puede ser desde un modelo 3D, hasta un archivo de audio o imagen. [19]

Algunos de los pros y contras de UE4 son:

- Pros:
 - Código fuente abierto que permite personalizar la IDE.
 - Blueprints: lenguaje de scripts visual para gente sin conocimientos en programación.
 - Actualizaciones constantes para añadir nuevas características.
 - Gran comunidad de desarrolladores que ayudan a solucionar problemas.
- Contras:
 - Menos *assets* disponibles que Unity.
 - Curva de aprendizaje complicada para usar eficientemente la IDE.
 - Licencia de pago si el producto creado se comercializa y alcanza unos ingresos de \$3000.

3.3.3 CryEngine 5

En la GDC 2016 (Game Developers Conference), la empresa Crytek anunciaba la quinta versión de su motor gráfico, CryEngine (Figura 5), en la cual este pasaba a seguir un modelo de negocio de “paga lo que quieras”, con el que se permitía utilizarlo gratuitamente [20]. En la actualidad es el motor con mejor rendimiento, y teniendo tanto los gráficos como las físicas más realistas. Aunque a nivel de plataformas para las que se puede desarrollar con él es uno de los más reducidos, permitiendo solo consolas de última generación, Windows, Linux, y los cascos de VR más populares del mercado. En cuanto a los lenguajes que soporta, principalmente se basa en C++, aunque también acepta Lua, y C# para plantillas y *assets*. A su vez, también tiene un lenguaje visual, llamado Flowgraph, que permite programar sin necesidad de código, al igual que los Blueprints de UE4. [21][22][23]



Figura 5: Logo de CryEngine

Los pros y contras más destables de CryEngine 5 son:

- Pros:
 - Flowgraph: permite programación mediante lenguaje visual.
 - Alto rendimiento de renderizado.
 - Gráficos con alto nivel de calidad.
 - Código fuente de la IDE abierto para que se pueda personalizar.
- Contras:
 - Poca documentación y comunidad debido a que es gratuito desde hace poco.
 - Soporte de pocas plataformas.
 - Curva de aprendizaje compleja.

3.4 Hardware para VR

En la actualidad, conviven una gran cantidad de dispositivos de todo tipo para interaccionar con la VR. El rango de estos va desde los HMDs hasta los trajes hápticos de cuerpo completo que aumentan el nivel de inmersión del usuario capturando todos sus movimientos. Así como otra gran cantidad wearables como Myo, brazalete capaz de leer la actividad muscular con el objetivo de controlar la tecnología de forma inalámbrica, o Boogio, sensores para las zapatillas que rastrean el movimiento físico desde el suelo transmitiendo este a la VR. Así pues, aunque hay una gran cantidad de dispositivos, se van a revisar solamente los que son de interés para los propósitos del proyecto: HMDs y Leap Motion.

3.4.1 Análisis de HMDs

En el mercado actual, se pueden encontrar una gran cantidad de HMDs, y aunque todos tiene el mismo fin, siempre hay características que los diferencian. Así, se han revisado los HMDs más populares para usar con VR: Oculus Rift S, HTC Vive y PlayStation VR.

3.4.1.1 Oculus Rift S

Salido al mercado este mismo año, las Oculus Rift S (Figura 6) es el sucesor del Oculus Rift. Lo más novedoso de este HMD, es que su sistema de posicionamiento es de dentro hacia afuera, es decir, no necesita sensores externos para saber dónde está la persona. Esto es así por el sistema de 5 cámaras integrado que lleva el casco. En lo referente a las lentes, han pasado de utilizar una pantalla OLED a una LCD, con lo que se reduce el efecto “rejilla” y produce unos colores más nítidos. Aunque la velocidad de refresco ha sido reducida de 90 a 80 Hz. En cuanto a las conexiones, el número necesario de estas se ha reducido, ya que ahora no hacen falta sensores externos, por lo que solo son necesarios un Display Port y un USB 3.0. Y finalmente, los auriculares que tenía su antecesor han sido removidos, de modo que ahora el audio sonará por el dispositivo que se le tenga conectado al ordenador para ello [24][25].



Figura 6: Modelo Oculus Rift S

3.4.1.2 HTC Vive

HTC Vive (Figura 7) es el resultado del esfuerzo conjunto de las empresas HTC y Valve. En su momento, antes de que saliesen las Oculus Rift S, no tenían nada que envidiarle a las Oculus Rift. De hecho, en cuanto al sistema de posicionamiento, HTC Vive ofrecía mejores sensores, ya que mientras que el rango de las Rift era de unos 2,5 metros, el de las Vive es de unos 4,5. En cuanto al resto de características son bastante similares, las Vive van equipadas con pantallas AMOLED, la velocidad de refresco es de 90Hz, y el ángulo de visión es de 110° como el de las Rift. En cuanto a las conexiones, para poder conectar el casco son necesarios solamente un puerto USB y otro HDMI, además de una toma de corriente. Mientras que los sensores son inalámbricos y solamente necesitan una toma de corriente cada uno. Finalmente, el audio está integrado en el casco, pero simplemente con un jack 3.5, para que se le conecten auriculares [26][27].



Figura 7: Modelo HTC Vive

3.4.1.3 PlayStation VR

Por último, el modelo de PlayStation VR (PSVR) (Figura 8) sigue en la misma línea que sus competidores Rift y Vive, y utiliza pantallas OLED con una velocidad de refresco de 90 Hz, aunque puede llegar hasta una velocidad de 120 Hz si se pone en el modo cine. Su ángulo de visión es de 100° siendo así ligeramente inferior que el de Vive y Rift. En cuanto al posicionamiento, PSVR utiliza la cámara de la misma marca, que lo que hace es registrar el movimiento de los puntos de luz del casco y de los mandos. Aunque su rango de movimiento es bastante más corto que el de su competencia, siendo de 1x2 metros. Por otro lado, la parte buena de este modelo es que funciona simplemente con la GPU que tiene la PS4 integrada, por lo que no requiere de tarjetas gráficas de alto rendimiento como las otras dos. Para terminar, en el tema del audio, este modelo también cuenta con un jack 3.5 para que se le conecten los auriculares deseados al igual que Vive. [28][29]



Figura 8: Modelo PlayStation VR

3.4.2 Leap Motion

Leap Motion (Figura 9) es un dispositivo USB, que mediante el uso de luces LED y sensores de cámara (Figura 10), es capaz de escanear un área de unos 60 cm³ sobre él. Lo que logra con este escaneo es hacer un seguimiento de las manos y los dedos que se mueven sobre su espacio de alcance. Los datos generados por dicho seguimiento son traducidos por el software Orion, desarrollado por la misma empresa, y convertidos en información para el ordenador, y es mediante este sistema que las manos y dedos se pueden ver virtualizados dentro de la pantalla [30].



Figura 10: Dispositivo Leap Motion



Figura 9: Leap Motion por dentro

Es con la salida del software Orion hace un par de años cuando Leap Motion se reinventa y pasa a estar optimizado para la VR/AR. La mejora en su sistema de seguimiento de manos y dedos, haciendo que este sea más rápido y preciso, permitiendo diferenciar las manos del entorno incluso si están en contacto con otra superficie, hace que los desarrolladores puedan recrear interacciones físicas mucho más detalladas. En palabras de Michael Buckwald, cofundador y CEO de Leap Motion: *“The holy grail of virtual reality is a sense of total presence and immersion. [...] People can use their own hands and fingers to interact with digital content in VR with the same ease and nuance they use in the real world”*⁷ [31].

Esta renovación en el software del dispositivo es lo que lo hace interesante para este proyecto, ya que permite virtualizar las manos dentro del programa de VR con un buen nivel de precisión, consiguiendo así una mayor sensación de inmersión por parte del usuario. De esta manera, se evita el uso de los mandos de Oculus Rift, ya que la interacción se hace mediante el movimiento natural de las manos. Y, de cara al paradigma de la ilusión de la mano de goma, es mucho más interesante, ya que el usuario ve su mano extendida dentro de la pantalla, cuando en el mundo real también la tiene extendida, sin necesidad de sostener ningún mando en ella para que aparezca en el mundo virtual. Esta correlación entre lo que pasa en el mundo real y en el virtual es necesaria para reforzar la sensación de *embodiment* que se pretende estudiar con el software.

En lo que se refiere a proyectos desarrollados con Leap Motion, se ha formado una gran cultura DIY alrededor suyo, debido a su bajo precio de venta al público y a su facilidad de integración con Unreal y Unity, gracias a los *assets* que ofrece la propia empresa. Pero a parte de esta gran cantidad de pequeños proyectos, cabe destacar el proyecto North Star desarrollado por la empresa fabricante.

North Star es un proyecto de AR, que permite desplegar una serie de elementos virtuales por encima del mundo real. Esto lo consigue gracias a un casco de AR diseñado por la propia compañía, en el cual se engancha el dispositivo Leap Motion para hacer el seguimiento de las manos y saber dónde desplegar los menús en cada momento. El casco, aunque es de bajo coste para seguir la filosofía de la empresa, tiene unas características muy buenas, usando dos pantallas LCD con baja persistencia y resolución 1600x1440, una velocidad de refresco de 120 Hz y un campo de visión de 100°. Además, el proyecto es de código abierto, y tanto el diseño del casco como el software que hace falta para desplegarlo se pueden encontrar de manera totalmente gratuita en el GitHub de Leap Motion [32][33].

⁷ Traducción de la cita: “El santo grail de la realidad virtual es un sentido de presencia total e inmersión. [...] La gente puede usar sus propias manos y dedos para interactuar con el contenido digital en VR con la misma facilidad y matiz que en el mundo real.”

Capítulo 4

“Reality is a construct of the neurons.”

— Abhijit Naskar, *What is Mind?*

Embodiment. Ilusión de la mano de goma

Este capítulo tratará otro pilar fundamental del proyecto: el *embodiment*. En la introducción se dio una definición, pero no se concretaron sus bases ni qué se conoce sobre este concepto. En este capítulo se pretende profundizar en la definición del *embodiment* (SoE), explicando de qué sentidos está compuesto, y cuál es su papel en la autoconciencia del cuerpo. A su vez, se explicará en que consiste el paradigma de la ilusión de la mano de goma y la virtualización del mismo.

4.1 ¿Qué es el *embodiment*?

Definición: “El SoE (pertenencia) hacia un cuerpo B es el sentido que surge cuando las propiedades de B se procesan como si fueran las propiedades del propio cuerpo biológico.” [3]

Esta definición, la que se utilizará en este proyecto, se basa en dos definiciones clásicas de *embodiment*, la dada por Vignemont: “*E (object) is embodied if and only if some properties of E are processed in the same way as the properties of one’s body.*”⁸ [34] y la dada por Blanke y Metzinger, en la que afirman que el *embodiment* incluye la “*experiencia subjetiva de usar y ‘tener’ un cuerpo.*” [35]. Sin embargo, en la definición que se utiliza, es necesario también conceptualizar las propiedades de B, que son: el sentido de autolocalización, el sentido de voluntad y el sentido de propiedad del cuerpo. Además, dependiendo de en qué grado se experimente cada uno de estos términos se puede considerar si el SoE es parcial o total (Tabla 3).

Tabla 3: Proposiciones para el SoE hacia un cuerpo B [3]

One experiences SoE toward a body B,	if one feels <i>self-located</i> inside B at least in a minimal intensity (P1) if one feels to be an <i>agent</i> of B at least in a minimal intensity (P2) if one feels B as one’s <i>own</i> body at least in a minimal intensity (P3) if and only if one experiences at least one of the three senses at least in a minimal intensity (P4)
One experiences full SoE toward a body B,	if one experiences all of the three senses at the maximum intensity (P5)

⁸ Traducción de la definición: “E es pertenecido, sí y solo sí, algunas propiedades de E son procesadas de la misma manera que las propiedades del propio cuerpo.”. La traducción es un poco confusa ya que *embodied* realmente se traduce como encarnado, pero en el concepto de *embodiment*, a lo que hace referencia es al sentido de pertenencia, como ya se explicó anteriormente.

4.1.1 Sentido de autolocalización

“Es un volumen determinado en el espacio donde uno se siente ubicado.” [3]. Normalmente coincide con el espacio corporal, ya que uno se siente localizado dentro de su propio cuerpo físico, con la excepción de las experiencias extracorporales, en las que un individuo tiene la sensación de estar fuera de su cuerpo [3]. El sentido de autolocalización se refiere a la experiencial espacial de estar dentro de un cuerpo y no a la de estar dentro de un mundo (con o sin cuerpo). Esta distinción es necesaria, ya que este sentido puede ser confundido con el concepto de presencia, el cual sí que se refiere a la relación entre uno mismo y el entorno. Aunque, ambos términos pueden considerarse complementarios y al juntarlos se obtendría la representación espacial de uno mismo. [3]

Es importante mencionar que la autolocalización viene altamente determinada por la percepción visuoespacial ya que esta es normalmente egocéntrica. Así como por las señales vestibulares debido a que contienen información sobre la “translación y rotación del cuerpo, además de orientación con respecto a la gravedad.” [35]. Y, también es influenciada por el tacto, ya que la piel es lo que separa el cuerpo del entorno, lo que crea una diferencia espacial en referencia a la proximidad al cuerpo. De esta manera, el espacio personal es el que ocupa el propio cuerpo, el espacio peripersonal es el adyacente al cuerpo, es decir, el que está dentro del alcance de los brazos, y el espacio extrapersonal que es el que no se puede alcanzar [3].

4.1.2 Sentido de voluntad

Se refiere al sentido de tener “control motor global, incluyendo la experiencia subjetiva de acción, control, intención, selección motora y la experiencia consciente de voluntad.” [35]. En otras palabras, este sentido hace referencia a que uno se siente el agente de sus propias acciones cuando las consecuencias predichas de una acción coinciden con las consecuencias reales de dicha acción. Además, varios estudios han revelado que la discrepancia entre la retroalimentación visual de una acción y el movimiento real, afecta negativamente a este sentimiento de voluntad [3].

4.1.3 Sentido de propiedad del cuerpo

La propiedad del cuerpo se refiere a la auto atribución de un cuerpo [36]. Profundizando más, se puede decir que el sentido de propiedad del cuerpo emerge de una combinación de influencias *bottom-up* (de abajo arriba) y *top-down* (de arriba abajo) [37]. La información *bottom-up* se refiere a la información sensorial aferente que va desde los órganos sensoriales hasta el cerebro. Mientras que la *top-down* consiste en los procesos cognitivos que pueden modular el procesamiento de los estímulos sensoriales [3].

4.2 Ilusión de la mano de goma

La ilusión de la mano de goma (*Rubber Hand Illusion*, RHI) fue inicialmente demostrada por Botvinick y Cohen en 1998. En su estudio, los participantes estaban sentados con el brazo izquierdo extendido sobre una mesa, y una mampara para ocultarlo de la vista del sujeto. A su vez, en el lado de la mampara que los sujetos sí podían ver, había una mano de goma (izquierda) de tamaño real. La tarea consistía en estimular ambas manos (la real y la de goma) de manera sincronizada durante un tiempo determinado. Al transcurrir dicho período, los sujetos comenzaban a percibir que la mano de goma era la suya, experimentando la ilusión de que los estímulos que veían eran los que realmente sentían (Figura 11) [38].

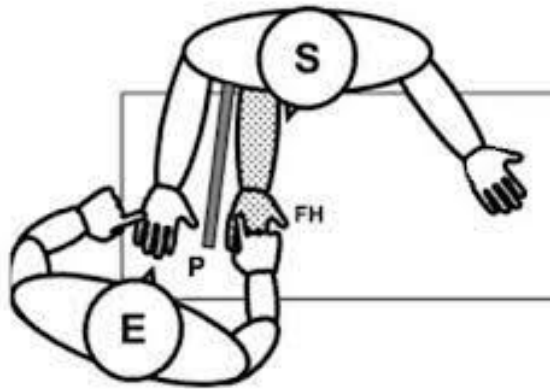


Figura 11: Ilusión de la mano de goma. S->sujeto. E->experimentador. FH->mano falsa. P->mampara

A día de hoy, se ha comprobado que es necesario que se cumplan una serie de restricciones para que se dé la RHI. Estas restricciones vienen determinadas para todo el sentido de autoconciencia del cuerpo, el cual incluye una percepción centrada en el cuerpo (mano, cara y torso), y son muy importantes para la integración multisensorial de señales, por lo tanto, es necesario tenerlas en cuenta a la hora de la RHI. Dichas restricciones son [41]:

- **Propioceptiva (posición del cuerpo):** la mano de goma tiene que estar posicionada en una postura que sea coherente con la mano real, es decir, si la mano real está con la palma hacia arriba, la de goma también debe estarlo [41].
- **Información visual relacionada con el cuerpo:** la mano artificial tiene que tener forma antropomórfica, es decir, que no se podría usar como mano de goma un trozo de madera. Por otro lado, se ha comprobado que el color de la piel de dicha mano no es relevante para que se produzca el *embodiment* [41].
- **Espacio peripersonal (PPS):** la mano falsa tiene que estar dentro del PPS de la persona para que se pueda generar la ilusión. El PPS es el espacio que hay inmediatamente alrededor de las partes del cuerpo, y que se mantienen unido a estas cuando se mueven, y donde la información relacionada con estímulos externos interactúa con el procesamiento de estímulos somatosensoriales⁹ en el cuerpo. Además, se ha verificado que la respuesta a un estímulo táctil en la mano se mejora si hay algún sonido dentro del PPS de dicha mano, como el del lápiz vibrador que la estimule. [41]
- **Embodiment:** más que una restricción es la consecuencia de que se apliquen señales multisensoriales sincronas de manera prolongada sobre una parte física del cuerpo biológico y su réplica artificial. Esto crea un vínculo temporalmente entre los estímulos de las diferentes modalidades y entre el cuerpo físico y el artificial, con lo que se altera la autoconciencia del cuerpo, al remodelar los límites del PPS e inducir la en objetos no corporales [41].

Por consiguiente, se puede ver que “a nivel estructural, la RHI requiere la participación de varios sentidos, por lo que se necesita una correcta integración multisensorial para que se produzca.” [39]. En cuanto a lo que el cerebro respecta, la corteza premotora y la corteza parietal se conoce que están involucradas en la representación multisensorial de la posición de las extremidades [40]. Por lo que donde se ha observado una mayor actividad cerebral durante la inducción de la RHI es en ambas cortezas, junto con el cerebelo y la ínsula (Figura 12) [39].

⁹ Los estímulos somatosensoriales son aquellos relacionados con el tacto, la temperatura, la propiocepción y la nocicepción (dolor) [42].

- **Corteza premotora:** en concreto la corteza premotora ventral es la encargada de la representación multisensorial del propio cuerpo (información visual, táctil e interoceptiva). Además, se ha encontrado que la activación en esta corteza se produce selectivamente debido a la integración multisensorial, por lo que se descarta que se deba solamente al procesamiento visual de objetos en el PPS [39].
- **Corteza parietal:** relacionada con la integración de estimulación visual, táctil y propioceptiva [39].
- **Cerebelo:** asociado al análisis temporal de las entradas sensorial. En concreto para la RHI estaría implicado en el análisis de la sincronía de la estimulación táctil. Adicionalmente, esta región recibe y envía conexiones tanto a la corteza premotora como a la parietal [39].
- **Ínsula:** específicamente es la ínsula posterior derecha la responsable del sentido de propiedad de la mano durante la RHI. Aunque la ínsula también se encarga de la termorregulación corporal, por lo que probablemente la temperatura modifique su actividad neuronal y así, module el sentido de propiedad de la mano en la RHI [39].

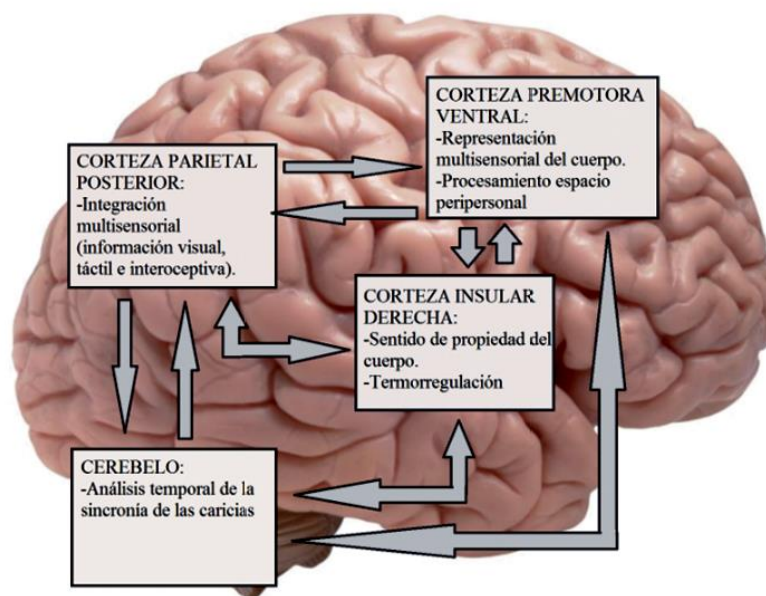


Figura 12: Representación del circuito que participaría en la RHI [39]

4.2.1 Procedimiento clásico del paradigma de la RHI

El experimento tradicionalmente consiste en dos fases: entrenamiento y evaluación. Ambas presentadas en el mismo orden y repitiéndose en dos condiciones diferentes. Las condiciones se diferencian en la posición de la mano artificial: condición congruente (la mano artificial se coloca en la misma posición que la mano del sujeto) y condición incongruente (la mano artificial se coloca de manera diferente a la del sujeto).

La fase de entrenamiento, consiste en estimular los dedos de la mano del sujeto y los de la mano artificial, acariciándolos de manera síncrona con algún tipo de pincel o similar. Mientras que en la fase de evaluación se evalúa la actividad cerebral y las respuestas fisiológicas consecuencia del entrenamiento mediante la estimulación de la mano de goma o la mano real de manera alternativa.

4.2.2 Virtualización del paradigma de la RHI

Para el propósito del proyecto, el procedimiento explicado en la sección anterior ha tenido que ser adaptado a fin de ser utilizado en un entorno de VR, donde el experimentador no interactúa de manera directa con el sujeto estimulado su mano y la artificial que, además en este caso sería virtual. De esta manera, se ha respetado la distinción de las dos fases separadas, la de entrenamiento con el objetivo de conseguir que el sujeto se acostumbre a las manos virtuales para lograr la inducción del *embodiment*, y la de evaluación para medir hasta qué punto se ha logrado dicho fenómeno.

La fase de entrenamiento se ha modificado de tal manera que consiste en un pequeño juego de puntería, en el cual el sujeto lanza una serie de proyectiles contra unos enemigos con la finalidad de eliminarlos. El objetivo de este juego es incentivar el uso de las manos por parte del participante, para que se produzca la asimilación y la identificación con la mano virtual. El programa realizado tiene también la capacidad de recrear la condición incongruente como condición de control.

La fase de evaluación, se ha mantenido lo más fiel posible a como se hace en el experimento clásico, recreado una simulación de esta. Así pues, se estimula la mano del participante mediante unos emisores de vibración y temperatura colocados en un guante que tiene puesto y, a su vez, se estimula la mano virtual mediante un lápiz vibrador representado en el mismo entorno. La condición incongruente ha sido mantenida en este caso, de manera que la mano virtual se puede poner al revés de como la tenga el participante, así como también ha sido mantenida la condición de que se estimulen manos o solo la mano virtual pero no la real y viceversa.

4.2.2.1 Estudio piloto realizado

Una vez adaptado el paradigma de la RHI al entorno de VR, se ha realizado un estudio piloto con X personas pertenecientes a la Universidad de Granada. El objetivo fue hacer una prueba para comprobar la eficacia de la asimilación de la mano virtual comparada con el procedimiento clásico de la mano de goma. En este estudio no se empleó la condición experimental de mano invertida, ya que no se correspondía con el objetivo de comparar el *embodiment* inducido por ambas situaciones.

En este estudio piloto un grupo de participantes, grupo A, fue sometido a la fase de entrenamiento del procedimiento tradicional durante 4 minutos. El grupo B fue asignado al procedimiento de la mano virtual, permaneciendo en la fase de entrenamiento durante 4 minutos. Tras finalizar el entrenamiento, cada participante respondió en una escala pictográfica su nivel de *embodiment* que experimentó con respecto a la mano falsa (ya fuera virtual o de goma), para así poder determinar hasta qué punto es eficiente el procedimiento virtual con respecto al clásico. Los datos del estudio y sus resultados se pueden consultar en el Apéndice II.

Capítulo 5

“When you want to know how things really work, study them when they’re coming apart.”

— William Gibson, *Zero History*

Análisis y diseño del sistema

Finalmente, una vez explicadas las bases teóricas del proyecto y explicado que se pretende conseguir con él, es el momento de pasar a analizar cuáles han sido los requisitos necesarios para lograrlo, desde el punto de vista de la ingeniería del software. Así, como cuál ha sido el diseño creado para poder cumplir dichos requisitos. De esta manera, se va a hacer un análisis de los requerimientos funcionales, no funcionales y de información del sistema. Posteriormente, se especificarán los casos de uso, en los cuales se detallan las acciones que pueden realizar los usuarios con el sistema y cuáles son las implicaciones de estas. Acto seguido se explicará cual ha sido el lenguaje diseñado para que el experimentador pueda comunicarse con el sistema, y como funciona dicho lenguaje. Finalmente, se mostrarán los diagramas de clases correspondientes a cada una de las escenas del programa, en los cuales se pueden ver las clases pertenecientes a cada una y que relación tienen unas clases con otras.

5.1 Especificación de requerimientos

La Ingeniería de Requerimientos (comprendida dentro de la Ingeniería del Software) se encarga de extraer, analizar, especificar y validar las necesidades de los clientes y comunicársela a los desarrolladores para que puedan implementar un sistema acorde a estas. Por lo tanto, la especificación de requerimientos del software (SRS) es una de sus tareas principales, ya que en ella se van a precisar los requisitos que van a describir los servicios que tiene que ofrecer el sistema y cuáles son las restricciones asociadas a su funcionamiento.

Los requerimientos que se van a detallar en las siguientes secciones estarán identificados por un código único que indicará a qué escena pertenece y de qué tipo es. De este modo, se utilizará el código para hacer referencia a cada requisito concreto, cada vez que sea necesario a lo largo del resto de proyecto.

Debido a la necesidad primaria de que el sistema tenga dos fases claramente diferenciadas (entrenamiento y simulación), los requerimientos se pueden separar en dos grupos, cada uno correspondiente a cada una de ellas. Así, el código de los requerimientos que estén relacionados con la fase de entrenamiento se expresará como RI-1.X, RF-1.X o RNF-1.X dependiendo del tipo. Mientras que los que describan la fase de simulación se numerarán como RI-2.X, RF-2.X y RNF-2.X.

5.1.1 Requerimientos de información

Los requerimientos de información describen las necesidades de almacenamiento de información en el sistema, es decir, aquella información relevante que debe gestionar y almacenar el software.

5.1.1.1 Fase de entrenamiento

- **RI-1.1:** Duración del juego: tiempo que durará una ejecución del juego.
- **RI-1.2:** Proyectiles: información de cada uno de los proyectiles que hay en la escena.
- **RI-1.3:** Modelo del proyectil: propiedades (estado, aspecto, tamaño del *collider*, propiedades del cuerpo rígido, fuente de sonido y propiedades táctiles) que tienen los proyectiles.
- **RI-1.4:** Sonidos del proyectil: sonidos que emite el proyectil al colisionar con otros objetos o destruirse.
- **RI-1.5:** Enemigos: información sobre las posiciones en las que pueden aparecer los enemigos (si están ocupadas o no) y sobre el enemigo que las ocupa en cada momento.
- **RI-1.6:** Enemigos de ejemplo: información sobre cada uno de los enemigos de ejemplo que hay en la escena.
- **RI-1.7:** Posiciones: lugares en los que puede aparecer un enemigo y la información de estos (posición espacial, orientación, estado de la luz de alerta).
- **RI-1.8:** Modelos de los enemigos: mallas utilizadas para darle forma definida a los enemigos.
- **RI-1.9:** Animaciones de los enemigos: efectos de animación que hacen los enemigos al aparecer, morir y mientras están en reposo.
- **RI-1.10:** Sonidos de los enemigos: sonidos que emiten los enemigos al aparecer.
- **RI-1.11:** Número de enemigos: cantidad de enemigos que han sido creados a lo largo de una ejecución del juego.
- **RI-1.12:** Número de enemigos destruidos: cantidad de enemigos destruidos por proyectiles a lo largo de una ejecución del juego.
- **RI-1.13:** Transparencias de información: información que se le muestra al usuario por la pantalla para explicarle como interaccionar con la escena.

5.1.1.2 Fase de simulación

- **RI-2.1:** Modelo del lápiz vibrador: información (aspecto, tamaño del *collider*, sistema de animación, fuente de audio) que tiene el lápiz vibrador.
- **RI-2.2:** Sonidos del lápiz vibrador: sonidos que emite el lápiz en sus distintas formas.
- **RI-2.3:** Animaciones del lápiz: animaciones que utiliza el lápiz para adaptarse a las órdenes y para ejecutarlas.
- **RI-2.4:** Órdenes: órdenes que tiene que ejecutar la simulación. Se almacenará por cada orden su tipo, estado, valor, tiempo y temperatura (solo en caso de ser de calor).
- **RI-2.5:** Imágenes: imágenes que el programa carga desde el exterior para mostrarlas por pantalla durante la ejecución de la simulación.

5.1.2 Requerimientos funcionales

Los requerimientos funcionales describen el funcionamiento del sistema, es decir, como interacciona el sistema con su entorno, proporcionando servicios o indicando de qué manera reacciona ante diferentes situaciones.

5.1.2.1 Fase de entrenamiento

- **RF-1.1:** El sistema debe iniciar la escena de entrenamiento al ejecutarse.
- **RF-1.2:** El sistema permitirá controlar la duración del juego para cada una de sus ejecuciones.
 - **RF-1.2.1:** Permitir introducir el tiempo de duración del juego en segundos por el teclado.
 - **RF-1.2.2:** Eliminar automáticamente la duración del juego almacenada al terminar cada ejecución de este.
 - **RF-1.2.3:** Aceptar el tiempo introducido solo si es mayor que 0.
 - **RF-1.2.4:** Verificar los caracteres introducidos para solo aceptar números entre 0 y 9 de manera que conformen la cadena que finalmente será el tiempo.
 - **RF-1.2.5:** Permitir corregir los valores introducidos mediante la tecla de retroceso.
 - **RF-1.2.6:** No aceptar la introducción de nuevos caracteres hasta que el tiempo introducido haya sido utilizado en una ejecución del juego.
 - **RF-1.2.7:** Dar información visual de que el tiempo ha sido introducido o eliminado satisfactoriamente.
- **RF-1.3:** El juego debe iniciarse por una acción del usuario desde dentro de la VR.
- **RF-1.4:** El sistema debe eliminar automáticamente todos los proyectiles o enemigos que pueda haber en la escena antes de iniciar el juego.
- **RF-1.5:** El sistema debe gestionar los proyectiles que hay en la escena y la información conocida sobre estos de manera autónoma.
 - **RF-1.5.1:** Añadir los nuevos proyectiles que se creen a la lista de proyectiles existentes.
 - **RF-1.5.2:** Modificar el estado de los proyectiles, desactivándolos cuando colisionen contra el suelo.
 - **RF-1.5.3:** Eliminar un proyectil cuando este colisione contra un enemigo. Dando clara información visual de que la colisión se ha producido contra un enemigo y no otro objeto.
 - **RF-1.5.4:** Eliminar todos los proyectiles almacenados, independientemente de su estado.
 - **RF-1.5.5:** Dar información acústica cuando un proyectil colisiones contra cualquier otro objeto a excepción de las manos del usuario.
 - **RF-1.5.6:** Mostrar información visual del estado de los proyectiles en todo momento.
- **RF-1.6:** El sistema debe permitir que el usuario cree nuevos proyectiles a su libre voluntad.
- **RF-1.7:** El sistema debe gestionar automáticamente los enemigos que hay en la escena.
 - **RF-1.7.1:** Mostrar información visual de cuantos enemigos han sido eliminados por proyectiles durante la ejecución, y al final de esta, una estadística comparativa entre cuantos han sido creados y cuantos han sido destruidos.
 - **RF-1.7.2:** Crear enemigos en las posiciones disponibles de manera aleatoria, con un tiempo aleatorio de espera entre uno y el siguiente, y con un tiempo aleatorio de vida. Avisando de en qué posición se va a generar antes de que

- aparezca, y dando información acústica cuando el enemigo se cree.
- **RF-1.7.3:** Eliminar los enemigos de la escena cuando estos son alcanzados por un proyectil activo o cuando su tiempo de vida se termina.
- **RF-1.8:** El sistema permitirá al experimentador crear enemigos de ejemplo en cualquier momento, pulsando la barra espaciadora.
- **RF-1.9:** El sistema debe mostrar al usuario información sobre como interaccionar con la escena hasta que el juego sea lanzado por primera vez.
 - **RF-1.9.1:** Detectar el gesto del pulgar hacia arriba en la mano derecha para avanzar en la información mostrada.
 - **RF-1.9.2:** Detectar el gesto del pulgar hacia arriba en la mano izquierda para retroceder en la información mostrada.
 - **RF-1.9.3:** Crear un enemigo de ejemplo de manera automática cuando la información mostrada lo especifique.
 - **RF-1.9.4:** Destruir automáticamente el enemigo de ejemplo si se avanza o se retrocede en la información indica su creación, en caso de seguir este existiendo.
- **RF-1.10:** El sistema debe desplegar un menú de opciones al pulsar la tecla ESC.
 - **RF-1.10.1:** Pausar la escena al desplegar el menú.
 - **RF-1.10.2:** Reanudar la escena al salir del menú.
 - **RF-1.10.3:** Cambiar el programa a la escena de simulación.
 - **RF-1.10.4:** Cerrar el sistema satisfactoriamente.

5.1.2.2 Fase de simulación

- **RF-2.1:** El sistema debe crear el lápiz vibrador cuando la mano del usuario esté en la posición correcta, la cual es con la palma hacia arriba y todos los dedos extendidos.
- **RF-2.2:** El sistema destruirá el lápiz vibrador cuando la mano deje de estar en la posición correcta, pero solo si la secuencia de órdenes no se está ejecutando.
- **RF-2.3:** El sistema leerá la secuencia de órdenes que tiene que seguir la simulación en su ejecución desde un fichero txt externo al programa, y almacenará dichas instrucciones hasta el cierre del sistema.
 - **RF-2.3.2:** Dar información visual una vez que las órdenes hayan sido cargadas y estén listas para ejecutarse
- **RF-2.4:** El sistema debe cargar y almacenar, hasta el cierre del sistema, imágenes que estén fuera del programa, para que así el experimentador pueda usar las que quiera en la ejecución.
- **RF-2.5:** El sistema ejecutará las órdenes cuando estas estén cargadas, la mano izquierda del usuario esté en posición correcta, de manera que el lápiz vibrador exista, y el experimentador pulse la tecla Intro.
- **RF-2.6:** El sistema procesará automáticamente los distintos tipos de órdenes para ejecutarlos correctamente.
 - **RF-2.6.1:** Adaptar el lápiz vibrador al tipo de orden en cada momento que sea necesario.
 - **RF-2.6.2:** Mover el lápiz vibrador a la posición que indica la instrucción o a la posición de parada en caso de ser una orden de stop.
 - **RF-2.6.3:** Ocultar el lápiz vibrador si la orden así lo especifica.
 - **RF-2.6.4:** Mostrar por pantalla la imagen que indique la orden de pantalla
 - **RF-2.6.5:** Dar información acústica cuando se está ejecutando una orden de vibración o calor.
- **RF-2.7:** El sistema se comunicará de manera automática con el controlador del guante y con el convertidor del polígrafo cuando sea necesario.

- **RF-2.7.1:** Traducir la orden para que el controlador o el convertidor la entienda.
- **RF-2.7.2:** Mandar orden de que cese la actividad del controlador o convertidor una vez haya transcurrido el tiempo establecido para la orden en ejecución.
- **RF-2.8:** Es sistema situará el lápiz vibrador en la posición de parada de manera automática una vez se haya terminado de ejecutar toda la secuencia de órdenes.
- **RF-2.9:** El sistema debe permitir que el experimentador modifique los modelos de las manos.
 - **RF-2.9.1:** Ampliar o reducir el modelo de ambas manos de manera independiente, mediante el uso del teclado.
 - **RF-2.9.2:** Voltar la mano izquierda para que se vea invertida a como la tiene puesta el usuario, es decir, si el usuario la tiene con la palma hacia arriba, que el modelo se vea con la palma hacia abajo y viceversa. Esta acción debe poder deshacerse.
- **RF-2.10:** El sistema debe desplegar un menú de pausa con las mismas funcionales y características que en RF-1.10, a excepción de RF-1.10.3, ya que en este caso la escena a la que permitirá cambiar es a la de entrenamiento.

5.1.3 Requerimientos no funcionales

Los requerimientos no funcionales describen cualidades o restricciones del sistema que no se relacionan de manera directa con el comportamiento de funcionamiento del mismo. Por lo tanto, estos requerimientos restringen temas como la facilidad de uso, el rendimiento, la fiabilidad, el soporte (facilidad de mantenimiento y portabilidad), restricciones hardware, e interfaces con otros sistemas.

En este caso además de requerimientos específicos para cada fase, también hay requerimientos que tiene cumplir el sistema de manera general, así que estos se van a denotar con el código RNF-X.

5.1.3.1 Requerimientos no funcionales generales

- **RNF-1:** El sistema debe funcionar sobre la plataforma Windows.
- **RNF-2:** Es necesario que el sistema virtualice las manos del usuario con la mayor precisión posible, es decir, que el movimiento de las manos virtuales no tenga retardo al de las reales. Por lo que se requiere de la integración del dispositivo Leap Motion en el sistema.
- **RNF-3:** Se precisa de la integración de Oculus Rift para poder ver el entorno desde una perspectiva de primera persona en VR, para así favorecer el *embodiment* del usuario.
- **RNF-4:** Debido a la necesidad de integrar Oculus Rift y Leap Motion en el sistema, se requiere de un ordenador potente, que tenga como mínimo una tarjeta gráfica NVIDIA GTX 960, y al menos 6 puertos USB para poder conectar todos los dispositivos.
- **RNF-5:** En caso de que alguno de los dispositivos externos necesario para el correcto funcionamiento del programa no esté conectado, se registrará el error, pero no se cerrará el programa.
- **RNF-6:** El modelo de las manos utilizado tiene que ser lo más realista posible para que se pueda producir el *embodiment*.
- **RNF-7:** El sistema debe informar mediante elementos visuales, acústicos o ambos cuando una de sus posibles acciones es realizada con éxito.
- **RNF-8:** El entorno tiene que emular la habitación en la que se encuentra el usuario.
- **RNF-9:** Es necesario que el experimentador tenga el control del sistema en todo momento, pudiendo pausar la escena y cambiar de una a otra a su libre voluntad.

5.1.3.2 Fase de entrenamiento

- **RNF-1.1:** El entorno tiene que incentivar el uso de las manos por parte del usuario, por eso tienen que facilitarse gestos con funciones y permitir que el usuario coja objetos con las ellas.
- **RNF-1.2:** El usuario debe acostumbrarse a como se crean objetos y se interacciona con los proyectiles con un pequeño tutorial y después de una primera ejecución del juego ya no debería tener problemas en agarrar los objetos correctamente.
- **RNF-1.3:** El nivel de dificultad del juego tiene que ser lo suficientemente alto como para que el usuario no se aburra, pero no demasiado alto para que no se cree un sentimiento de frustración en este.
- **RNF-1.4:** El juego tiene que poder lanzarse tantas veces como el experimentador desee dentro de la misma ejecución del sistema, y tiene que permitirse que este controle el tiempo que va a durar cada vez que lo lance.
- **RNF-1.5:** Los enemigos tienen que tener una forma llamativa para que se distingan claramente.
- **RNF-1.6:** Los modelos y sonidos de los enemigos se le asignan de manera aleatoria al crearse y no tienen ninguna relevancia a nivel de puntuación.

5.1.3.3 Fase de simulación

- **RNF-2.1:** Las órdenes se importan al sistema a través de un fichero txt en el que estas deben estar redactadas correctamente, ya que el sistema no tiene que encargarse de verificarlas.
- **RNF-2.2:** Las imágenes tienen que estar en formato png para que puedan ser leídas desde el programa.
- **RNF-2.3:** El sistema tiene que conectarse al controlador del guante que transmitirá los impulsos al usuario. Para transferir los datos desde el programa se utiliza una interfaz de puerto de serie con las configuraciones detalladas en la documentación del controlador, para que así el retraso en el envío sea mínimo.
- **RNF-2.4:** El sistema transfiere los datos al convertidor del polígrafo mediante un puerto de serie, con una configuración estándar por defecto.
- **RNF-2.5:** El lápiz vibrador tendrá unas características bien diferenciadas para que se pueda distinguir cuando va a ejecutar una orden de vibración y cuando una de temperatura.

5.2 Modelado de casos de uso

El modelado de casos de uso es otra parte de la Ingeniería de Requerimientos, que se utiliza con la finalidad de ayudar a crear la base para el proceso de diseño, facilitando la construcción de prototipos. El modelado está compuesto por actores, casos de uso y relaciones (entre actores, entre casos de uso y entre ambos). Para la representación de estos se utilizan diagramas en lenguaje UML (*Unified Modeling Language*).

Los actores son las entidades externas que interactúan directamente con el sistema. Los casos de uso describen el modo en el que los actores interactúan con el sistema y explican el funcionamiento de este. Finalmente, las relaciones sirven para organizar los casos de uso, mejorando su comprensión.

5.2.1 Descripción de los actores

Dadas las necesidades del sistema, se precisa que este tenga 3 actores diferentes: participante (Tabla 4), experimentador (Tabla 5), tiempo (Tabla 6).

Tabla 4: Definición del actor 1 - Participante

Actor	Participante	ACT-1			
Descripción	Persona sobre la cual se realiza el experimento				
Características	Actor que interacciona directamente desde la VR con el sistema, es decir, es el que ve el entorno en primera persona e interactúa con los objetos directamente con sus manos.				
Relaciones					
Referencias					
Autor	Andrés Molina López	Fecha	27-08-19	Versión	1.0

Tabla 5: Definición del actor 2 - Experimentador

Actor	Experimentador	ACT-2			
Descripción	Investigador que realiza el experimento				
Características	Actor que controla el sistema desde fuera de la VR. Es la persona que ve la aplicación en el monitor y la maneja desde el teclado, asignado los valores de los parámetros.				
Relaciones					
Referencias					
Autor	Andrés Molina López	Fecha	27-08-19	Versión	1.0

Tabla 6: Definición del actor 3 - Temporizador

Actor	Temporizador				ACT-3	
Descripción	Actor que realiza acciones cada determinado tiempo en el sistema					
Características	Actor que realmente está dentro del sistema pero que realiza tareas cuando se cumple alguno de los periodos de tiempo determinados.					
Relaciones						
Referencias						
Autor	Andrés Molina López		Fecha	27-08-19	Versión	1.0

5.2.2 Diagramas de casos de uso

En el sistema se pueden diferenciar claramente dos subsistemas que funcionan de manera independiente, como ya se ha mencionado anteriormente. Por lo tanto, se ha elaborado un diagrama de casos de uso individual para cada uno, con el fin de establecer que actores y casos de uso intervienen en cada uno de ellos y simplificar el diseño.

5.2.2.1 Subsistema de la fase de entrenamiento

Los casos de uso de que se muestran en el diagrama (Figura 13) solo pueden ocurrir en la fase de entrenamiento.

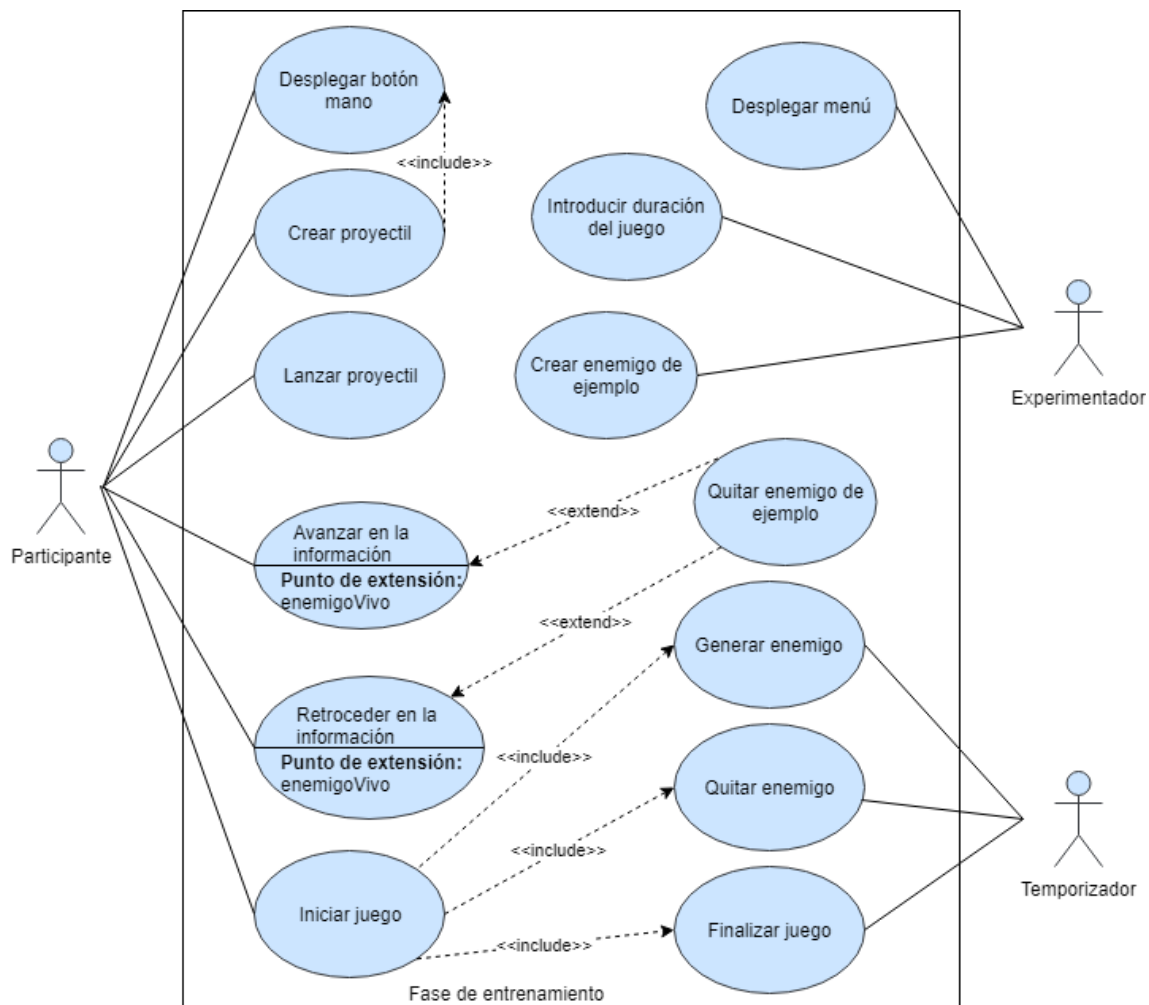


Figura 13: Diagrama de casos de uso de la fase de entrenamiento

A continuación, se muestran la descripción de cada uno de ellos.

Tabla 7: CU-1 Desplegar botón mano

Caso de Uso	Desplegar botón mano				CU-1
Actores	ACT-1				
Tipo	Primario extendido esencial				
Referencias	RF-1.6				
Precondición					
Postcondición	Un pequeño botón rojo aparecerá al lado de la mano izquierda				
Autor	Andrés Molina López	Fecha	28-08-19	Versión	1.0
Propósito					
Permitir que el participante genere proyectiles cuando quiera e incentivar el uso de las manos.					
Resumen					
El participante voltea su mano izquierda hacia arriba para que un pequeño botón aparezca junto a esta y pueda ser pulsado con la otra mano.					
Curso Normal					
1	Participante pone la palma izquierda hacia arriba	2	Genera un botón al lado de la mano izquierda		

Tabla 8: CU-2 Crear proyectil

Caso de Uso	Crear proyectil				CU-2		
Actores	ACT-1						
Tipo	Primario extendido esencial						
Referencias	RF-1.5.1,6; RF-1.6				CU-1		
Precondición	Se tiene que haber realizado el CU-1						
Postcondición	Aparece un proyectil en la escena						
Autor	Andrés Molina López			Fecha	28-08-19	Versión	1.0
Propósito							
Crear proyectiles que pueda utilizar el participante.							
Resumen							
El participante pulsa el pequeño botón rojo junto a su mano izquierda, un cubo rojo (proyectil), se genera un poco por encima de la vista del usuario y cae encima de la mesa que este tiene delante para que quede al alcance de sus manos.							
Curso Normal							
1	Incluir CU-1: Desplegar botón mano						
2	Participante pulsa botón junto a su mano			3	Genera un cubo rojo un poco por encima de la vista del usuario		
				4	Almacena el proyectil creado en la lista de proyectiles de la escena		

Tabla 9: CU-3 Lanzar proyectil

Caso de Uso	Lanzar proyectil				CU-3	
Actores	ACT-1					
Tipo	Primario extendido esencial					
Referencias	RF-1.5.2,3,5,6; RF-1.7.1,3					
Precondición	Tiene que existir al menos un proyectil en la escena al alcance del usuario					
Postcondición	La posición y estado del proyectil cambia, pudiendo ser destruido por el camino al destruir algún enemigo					
Autor	Andrés Molina López		Fecha	28-08-19	Versión	1.0
Propósito						
Destruir enemigos.						
Resumen						
El participante coge uno de los proyectiles que haya creado y tenga a su alcance. Lo arroja mediante un gesto natural, y se actualiza la posición y el estado de este, además de calcular las colisiones que pueda tener para saber si tiene que ser destruido o no.						
Curso Normal						
1	Participante coge uno de los proyectiles					
2	Participante arroja el proyectil		3	Comprueba las colisiones		
			4	Emite sonido de choque		
			5	Desactiva el proyectil ya que acaba chocando contra el suelo por la gravedad		
Cursos Alternos						
5.a	El proyectil encuentra un enemigo en su trayectoria por lo que no llega a impactar contra el suelo.					

1	El sistema crea una explosión en la zona del impacto con el enemigo que se autodestruye al segundo
2	El sistema elimina al enemigo y actualiza la lista correspondiente de enemigo o enemigo de ejemplo
3	El sistema elimina el proyectil y actualiza la puntuación del participante y la lista de proyectiles

Tabla 10: CU-4 Avanzar en la información

Caso de Uso	Avanzar en la información				CU-4	
Actores	ACT-1					
Tipo	Primario extendido esencial					
Referencias	RF-1.9.1,3,4					
Precondición						
Postcondición	Se cambia la imagen que se muestra en la pantalla					
Autor	Andrés Molina López		Fecha	28-08-19	Versión	1.0
Propósito						
Explicar al usuario como interaccionar con la escena.						
Resumen						
El participante hace el gesto del pulgar hacia arriba con la mano derecha para que la información que se muestra en la pantalla que ve se actualice y le siga explicando como interactuar con los objetos de la escena.						
Curso Normal						
1	Participante hace gesto de pulgar hacia arriba con la mano derecha		2	Cambia la imagen de la pantalla a la siguiente que tiene almacenada		
	Punto de extensión: enemigoVivo					
Cursos Alternos						
2.a	No quedan más imágenes a las que avanzar					
1	El sistema no hace nada y se termina el caso de uso					
2.b	La imagen a la que se avanza implica que se cree un enemigo					
1	El sistema genera automáticamente un enemigo de ejemplo					
2	El sistema almacena el enemigo en la lista de enemigos de ejemplo					

Tabla 11: CU-5 Retroceder en la información

Caso de Uso	Retroceder en la información			CU-5	
Actores	ACT-1				
Tipo	Primario extendido esencial				
Referencias	RF-1.9.2,3,4				
Precondición					
Postcondición	Se cambia la imagen que se muestra en la pantalla				
Autor	Andrés Molina López	Fecha	28-08-19	Versión	1.0
Propósito					
Recordar al usuario como interaccionar con la escena.					
Resumen					
El participante hace el gesto del pulgar hacia arriba con la mano izquierda para que la información que se muestra en la pantalla que ve se actualice y le vuelva a mostrar información que ya había visto antes y puede recordarla.					
Curso Normal					

1	Participante hace gesto de pulgar hacia arriba con la mano izquierda	2	Cambia la imagen de la pantalla a la anterior que tiene almacenada
	Punto de extensión: enemigoVivo		
Cursos Alternos			
2.a	No quedan más imágenes a las que retroceder		
1	El sistema no hace nada y se termina el caso de uso		
2.b	La imagen a la que se retrocede implica que se cree un enemigo		
1	El sistema genera automáticamente un enemigo de ejemplo		
2	El sistema almacena el enemigo en la lista de enemigos de ejemplo		

Tabla 12: CU de extensión

CU de extensión: Quitar enemigo de ejemplo	
Segmento 1	
Precondición: el enemigo de ejemplo sigue vive cuando se cambia de imagen	
Curso normal de eventos	
Sistema	
1	El sistema elimina automáticamente al enemigo de ejemplo

Tabla 13: CU-6 Iniciar juego

Caso de Uso	Iniciar juego				CU-6	
Actores	ACT-1 (principal) y ACT-3 (secundario)					
Tipo	Primario extendido esencial					
Referencias	RF-1.3; RF-1.4; RF-1.5.4; RF-1.7.1			CU-10, CU-11, CU-12		
Precondición	Se tiene que haber realizado el CU-8					
Postcondición	Comienza a crearse enemigos en la escena					
Autor	Andrés Molina López		Fecha	28-08-19	Versión	1.0
Propósito						
Hacer que el participante se acostumbre al uso de las manos virtuales.						
Resumen						
El participante pulsa el botón rojo que tiene a su derecha, haciendo que este desaparezca junto con todos los enemigos y proyectiles que pueda haber en la escena, y el sistema comienza a generar enemigos en bucle en posiciones aleatorias cada cierto tiempo (en un rango aleatorio) para que el participante los destruya antes de que desaparezcan.						
Curso Normal						
1	Participante pulsa el botón a su derecha		2	Elimina todos los proyectiles de la escena		
			3	Desactiva las imágenes de explicación y muestra la puntuación actual del participante en su lugar		
4	Participante suelta el botón pulsado		5	Oculta el botón para desactivarlo		
			6	Elimina todos los enemigos de ejemplo de la escena		
			7	Incluir CU-10: Generar enemigo		
			8	Incluir CU-11: Quitar enemigo		
			9	Incluir CU-12: Finalizar juego		
Cursos Alternos						

8a	El participante destruye el enemigo antes de que se acabe su tiempo de vida
1	El temporizado no hace nada ya que el enemigo ya no existe
2	El sistema actualiza la puntuación del jugador

Tabla 14: CU-7 Desplegar menú

Caso de Uso	Desplegar menú			CU-7	
Actores	ACT-2				
Tipo	Primario extendido esencial				
Referencias	RF-1.1; RF-1.10				
Precondición					
Postcondición	Se despliega un menú de opciones y se pausas el programa				
Autor	Andrés Molina López	Fecha	28-08-19	Versión	1.0
Propósito					
Permitir pausar el programa, cambiar de escena y cerrar el programa correctamente.					
Resumen					
El experimentador pulsa la tecla ESC para desplegar el menú de opciones, el sistema se pausa y espera a que el experimentador pinche sobre alguna de las opciones o vuelva a pulsar la tecla ESC para cerrarlo.					
Curso Normal					
1	El experimentador pulsa la tecla ESC	2	El sistema despliega el menú de opciones, pausando el programa		
3	El experimentador pincha sobre la opción reanudar o vuelve a pulsar ESC	4	El sistema cierra el menú de opciones y reanuda la ejecución del programa		
Cursos Alternos					
3a	El experimentador pincha sobre la opción de simulación				
1	El sistema cierra la escena actual y cambia a ejecutar la escena de simulación				
3b	El experimentador pincha sobre la opción de salir				
1	El sistema cesa toda actividad y se cierra				

Tabla 15: CU-8 Introducir duración del juego

Caso de Uso	Introducir duración del juego				CU-8	
Actores	ACT-2					
Tipo	Primario extendido esencial					
Referencias	RF-1.2.1,3,4,5,6,7					
Precondición						
Postcondición	El CU-6 puede realizarse					
Autor	Andrés Molina López		Fecha	28-08-19	Versión	1.0
Propósito						
Dar control sobre el tiempo que va a durar la fase al experimentador.						
Resumen						
El experimentador introduce el tiempo que va a durar la ejecución del juego, se almacena y se notifica visual que ha sido introducida.						
Curso Normal						
1	El experimentador introduce el tiempo por el teclado		2	Guarda cada tecla en una cadena		
3	El experimentador pulsa la tecla Intro		4	Traduce la cadena a un número.		

		5	Almacena el número traducido
		6	Notifica que el número ha sido guardado correctamente
Cursos Alternos			
1a	El experimentador introduce un carácter no numérico		
1	El sistema no guarda el carácter en la cadena		
3a	El experimentador pulsa la tecla de retroceso		
1	El sistema borra el último carácter introducido en la cadena		
5a	El número a almacenar es 0		
1	El sistema descarta el número y reinicia la lectura de la cadena		
5b	La variable de tiempo ya tiene un número positivo almacenado		
1	El sistema no hace nada con los nuevos valores introducidos		

Tabla 16: CU-9 Crear enemigo de ejemplo

Caso de Uso	Crear enemigo de ejemplo				CU-9
Actores	ACT-2				
Tipo	Primario extendido esencial				
Referencias	RF-1.8				
Precondición					
Postcondición	Aparece un enemigo de ejemplo en la escena				
Autor	Andrés Molina López	Fecha	28-08-19	Versión	1.0
Propósito					
Permitir que el experimentador le ponga más enemigos al participante para que practique.					
Resumen					
El experimentador pulsa la barra espaciadora, se crea un enemigo en una posición determinada y se almacena en la lista de enemigos de ejemplo.					
Curso Normal					
1	Experimentador pulsa la barra espaciadora		2	Crea un enemigo en una posición determinada	
			3	Almacena el enemigo en la lista de enemigos de ejemplo	

Tabla 17: CU-10 Generar enemigo

Caso de Uso	Generar enemigo				CU-10	
Actores	ACT-3					
Tipo	Primario extendido esencial					
Referencias	RF-1.7.2			CU-6		
Precondición	El juego tiene que estar iniciado y quedarle tiempo de duración					
Postcondición	Aparece un enemigo en la escena					
Autor	Andrés Molina López			Fecha	28-08-19	Versión 1.0
Propósito						
Darle al participante un objetivo en el juego.						
Resumen						
Una vez transcurrido el tiempo necesario para crear un nuevo enemigo, el temporizador notifica al sistema que tiene que generarlo, y este lo crea, almacena y asigna un nuevo tiempo al temporizador.						

Curso Normal			
1	Temporizador notifica al sistema que tiene que crear un nuevo enemigo	2	Elige aleatoriamente una posición para crear al enemigo
		3	Enciende la luz de esa posición para avisar al participante
		4	Genera al enemigo en la posición elegida
		5	Asigna al temporizador un nuevo tiempo de espera

Tabla 18: CU-11 Quitar enemigo

Caso de Uso	Quitar enemigo			CU-11	
Actores	ACT-3				
Tipo	Primario extendido esencial				
Referencias	RF-1.7.3		CU-6		
Precondición	Tiene que existir al menos enemigo cuyo tiempo de vida se haya agotado				
Postcondición	Se elimina el enemigo con tiempo de vida agotado				
Autor	Andrés Molina López		Fecha	28-08-19	Versión 1.0
Propósito					
Dificultar el juego para hacerlo más interesante.					
Resumen					
Cuando el tiempo de vida de un enemigo ha finalizado, el temporizador notifica al sistema que lo elimine, y este lo quita de la escena y lo elimina de la lista de enemigos.					
Curso Normal					
1	Temporizador notifica al sistema que tiene que quitar un nuevo enemigo		2	Elimina al enemigo de la escena	
			3	Elimina al enemigo de la lista, marcando su posición como libre	

Tabla 19: CU-12 Finalizar juego

Caso de Uso	Finalizar juego				CU-12	
Actores	ACT-3					
Tipo	Primario extendido esencial					
Referencias	RF-1.2.2,7; RF-1.7.1				CU-6	
Precondición	El juego tiene que estar iniciado y su tiempo de duración finalizado					
Postcondición	Se termina el juego y se borra el tiempo de duración establecido					
Autor	Andrés Molina López			Fecha	28-08-19	Versión 1.0
Propósito						
Terminar el juego.						
Resumen						
Cuando la duración del juego se termina, el temporizador se lo notifica al sistema para que este deje de generar enemigos, muestre la estadística de la partida y borre el tiempo de duración que tiene establecido.						
Curso Normal						
1	Temporizador notifica al sistema que la duración del juego se ha terminado			2	Deja de asignarle tiempos al temporizador	

		3	Calcula estadística entre enemigos creados y destruidos por proyectiles
		4	Muestra la estadística por pantalla al participante
		5	Elimina el tiempo de duración que tiene almacenada y limpia el buffer de entrada
		6	Avisar visualmente que el tiempo ha sido eliminado

5.2.2.2 Subsistema de fase de simulación

Los casos de uso que se muestran en el diagrama (Figura 14) corresponden a la fase de simulación y solo pueden ocurrir cuando esta se está ejecutando. El caso de uso “Mostrar menú” es muy similar al explicado en la fase de entrenamiento bajo el nombre “Desplegar menú” y código CU-7, lo cual es debido a que en ambas escenas se tiene que poder ver un menú para cambiar de una a otra e interconectar de esta manera ambos subsistemas en uno.

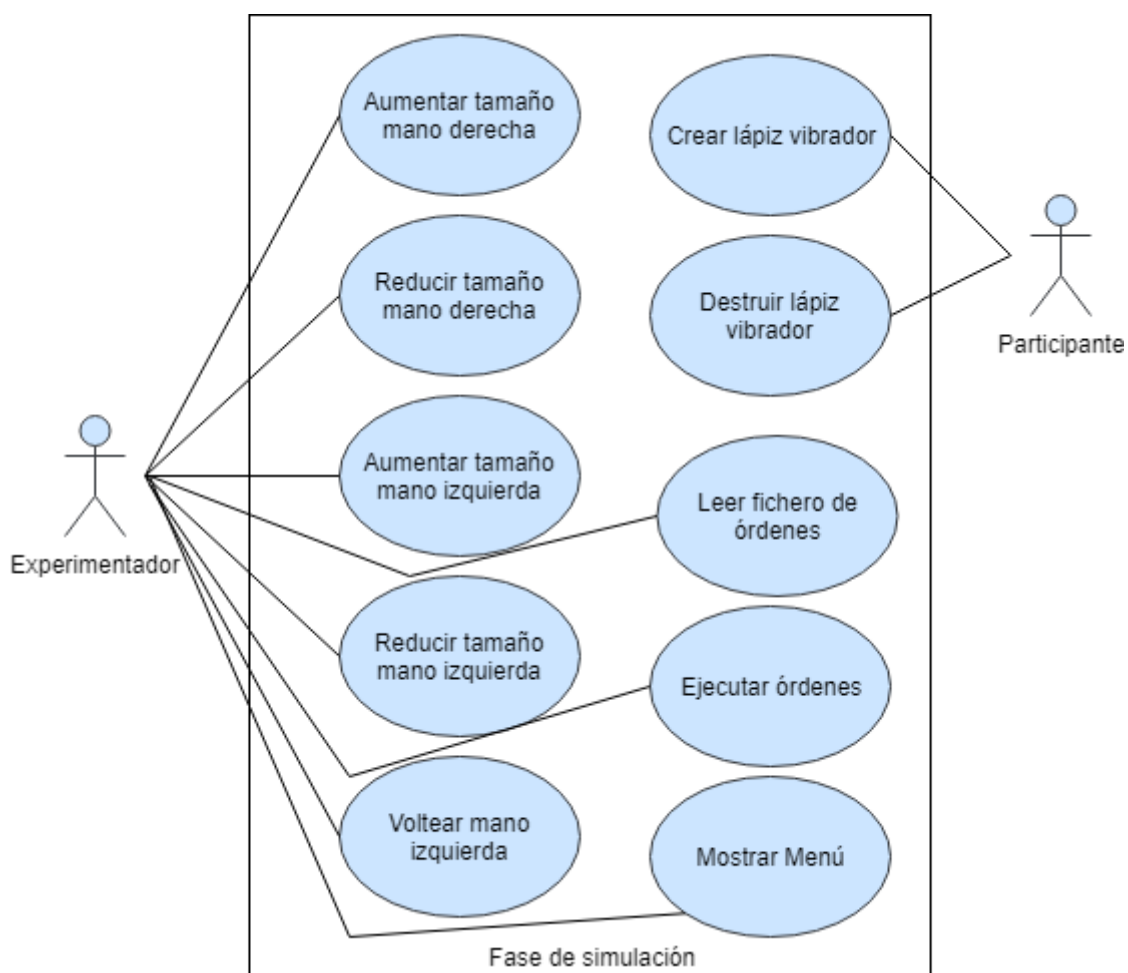


Figura 14: Diagrama de casos de uso de la fase de simulación

La definición de cada caso de uso junto con su secuencia de acciones se puede ver en las tablas mostradas a continuación:

Tabla 20: CU-13 Aumentar tamaño mano derecha

Caso de Uso	Aumentar tamaño mano derecha				CU-13
Actores	ACT-2				
Tipo	Primario extendido esencial				
Referencias	RF-2.9.1				
Precondición					
Postcondición	El modelo de la mano derecha es más grande				
Autor	Andrés Molina López	Fecha	29-08-19	Versión	1.0
Propósito					
Aumentar el tamaño de la mano derecha para enfatizarla.					
Resumen					
El experimentador pulsa la tecla numérica 1 para escalar el modelo de la mano derecha sumándole 0.1 en cada dimensión.					
Curso Normal					
1	El experimentador pulsa la tecla 1		2	Escala el modelo de la mano derecha sumándole 0.1	

Tabla 21: CU-14 Reducir tamaño mano derecha

Caso de Uso	Reducir tamaño mano derecha				CU-14
Actores	ACT-2				
Tipo	Primario extendido esencial				
Referencias	RF-2.9.1				
Precondición					
Postcondición	El modelo de la mano derecha es más pequeño				
Autor	Andrés Molina López		Fecha	29-08-19	Versión 1.0
Propósito					
Reducir el tamaño de la mano derecha para atenuar la atención del participante en ella.					
Resumen					
El experimentador pulsa la tecla numérica 2 para escalar el modelo de la mano derecha restándole 0.1 en cada dimensión.					
Curso Normal					
1	El experimentador pulsa la tecla 2		2	Escala el modelo de la mano derecha restándole 0.1	

Tabla 22: CU-15 Aumentar tamaño mano izquierda

Caso de Uso	Aumentar tamaño mano izquierda				CU-15
Actores	ACT-2				
Tipo	Primario extendido esencial				
Referencias	RF-2.9.1				
Precondición					
Postcondición	El modelo de la mano izquierda es más grande				
Autor	Andrés Molina López		Fecha	29-08-19	Versión 1.0
Propósito					
Aumentar el tamaño de la mano izquierda para enfatizarla.					
Resumen					
El experimentador pulsa la tecla numérica 3 para escalar el modelo de la mano izquierda sumándole 0.1 en cada dimensión.					
Curso Normal					
1	El experimentador pulsa la tecla 3		2	Escala el modelo de la mano izquierda sumándole 0.1	

Tabla 23: CU-16 Reducir tamaño mano izquierda

Caso de Uso	Reducir tamaño mano izquierda				CU-16
Actores	ACT-2				
Tipo	Primario extendido esencial				
Referencias	RF-2.9.1				
Precondición					
Postcondición	El modelo de la mano izquierda es más pequeño				
Autor	Andrés Molina López		Fecha	29-08-19	Versión 1.0
Propósito					
Reducir el tamaño de la mano izquierda para atenuar la atención del participante en ella.					
Resumen					
El experimentador pulsa la tecla numérica 4 para escalar el modelo de la mano izquierda restándole 0.1 en cada dimensión.					
Curso Normal					
1	El experimentador pulsa la tecla 4		2	Escala el modelo de la mano izquierda restándole 0.1	

Tabla 24: CU-17 Voltear mano izquierda

Caso de Uso	Voltear mano izquierda			CU-17	
Actores	ACT-2				
Tipo	Primario extendido esencial				
Referencias	RF-2.9.2				
Precondición					
Postcondición	El modelo de la mano izquierda queda invertido a como estaba				
Autor	Andrés Molina López	Fecha	29-08-19	Versión	1.0
Propósito					
Crear una condición incongruente para que no se produzca el <i>embodiment</i> .					
Resumen					
El experimentador pulsa la tecla numérica 5 para invertir el modelo de la mano izquierda dándole la vuelta a este, y el sistema lo gira de manera que la mano acabe girada 180° en el eje Z con respecto de cómo estaba.					
Curso Normal					
1	El experimentador pulsa la tecla 5		2	Gira el modelo de la mano izquierda 180° con respecto al eje Z	
			3	Reorienta el detector de la palma para que lápiz vibrador se cree cuando el modelo tiene la palma hacia abajo.	
Cursos Alternos					
2a	El modelo ya había sido invertido				
1	Gira el modelo de la mano izquierda 180° con respecto al eje Z devolviéndolo a su posición inicial				
2	Reorienta el detector de la palma para crear el lápiz vibrador cuando esta apunta hacia arriba				

Tabla 25: CU-18 Leer fichero de órdenes

Caso de Uso	Leer fichero de órdenes				CU-18		
Actores	ACT-2						
Tipo	Primario extendido esencial						
Referencias	RF-2.3; RF-2.4						
Precondición							
Postcondición	Las imágenes y órdenes están almacenadas y listas para utilizarse						
Autor	Andrés Molina López			Fecha	29-08-19	Versión	1.0
Propósito							
Permitir al experimentador personalizar cada ejecución de la simulación.							
Resumen							
El experimentador pulsa la barra espaciadora, se leen las órdenes desde el fichero externo y se almacenan.							
Curso Normal							
1	El experimentador pulsa la barra espaciadora			2	Abre el fichero de órdenes		
				3	Lee el fichero línea a línea		
				4	Almacena la orden leída y traducida en la lista de órdenes		
				5	Cierra el fichero una vez se han leído todas las líneas		
				6	Avisa visualmente que las órdenes e imágenes se han cargado		
Cursos Alternos							
4a	En lugar de una orden lee la dirección absoluta de un archivo png						
1	El sistema comprueba que el archivo existe						
2	El sistema lee todos los bytes del archivo						
3	El sistema convierte los bytes en una textura y ésta en un sprite						
4	El sistema añade el sprite creado a la lista de sprites						

Tabla 26: CU-19 Ejecutar órdenes

Caso de Uso	Ejecutar órdenes				CU-19
Actores	ACT-2				
Tipo	Primario extendido esencial				
Referencias	RF-2.5; RF-2.6; RF-2.7; RF-2.8				
Precondición	El CU-21 tiene que haberse realizado				
Postcondición	La secuencia de órdenes ha sido ejecutada una vez				
Autor	Andrés Molina López	Fecha	29-08-19	Versión	1.0
Propósito					
Llevar a cabo el experimento.					
Resumen					
El experimentador pulsa la tecla Intro, se leen las órdenes de una en una y se van ejecutando.					
Curso Normal					
1	El experimentador pulsa la tecla Intro	2	Selecciona las órdenes en la secuencia que han sido almacenadas		
		3	Comprueba el tipo de orden que es		
		4	Ejecuta la orden		
		5	Devuelve el lápiz vibrador a la posición inicial una vez todas las ordenes han sido ejecutadas		

Tabla 27: CU-20 Mostrar menú

Caso de Uso	Mostrar menú				CU-20
Actores	ACT-2				
Tipo	Primario extendido esencial				
Referencias	RF-2.10				
Precondición					
Postcondición	Se despliega un menú de opciones y se pausas el programa				
Autor	Andrés Molina López	Fecha	28-08-19	Versión	1.0
Propósito					
Permitir pausar el programa, cambiar de escena y cerrar el programa correctamente.					
Resumen					
El experimentador pulsa la tecla ESC para desplegar el menú de opciones, el sistema se pausa y espera a que el experimentador pinche sobre alguna de las opciones o vuelva a pulsar la tecla ESC para cerrarlo.					
Curso Normal					
1	El experimentador pulsa la tecla ESC		2	El sistema despliega el menú de opciones, pausando el programa	
3	El experimentador pincha sobre la opción reanudar o vuelve a pulsar ESC		4	El sistema cierra el menú de opciones y reanuda la ejecución del programa	
Cursos Alternos					
3a	El experimentador pincha sobre la opción de juego				
1	El sistema cierra la escena actual y cambia a ejecutar la escena de entrenamiento				
3b	El experimentador pincha sobre la opción de salir				
1	El sistema cesa toda actividad y se cierra				

Tabla 28: CU-21 Crear lápiz vibrador

Caso de Uso	Crear lápiz vibrador				CU-21
Actores	ACT-1				
Tipo	Primario extendido esencial				
Referencias	RF-2.1				
Precondición					
Postcondición	Se crea un lápiz vibrador enfrente de la mano				
Autor	Andrés Molina López	Fecha	29-08-19	Versión	1.0
Propósito					
Crear el lápiz que simulará la estimulación en la mano virtual.					
Resumen					
El participante voltea su mano izquierda real de manera que la palma apunte hacia arriba y extiende los dedos, se detecta la dirección de la palma y la posición de los dedos y se crea un lápiz vibrador enfrente de la mano izquierda virtual que acompaña su movimiento.					
Curso Normal					
1	El participante pone su mano izquierda real con la palma hacia arriba y los dedos extendidos		2	Crea un lápiz vibrador enfrente del dedo corazón de la mano izquierda virtual	

Tabla 29: CU-22 Destruir lápiz vibrador

Caso de Uso	Destruir lápiz vibrador				CU-22
Actores	ACT-2				
Tipo	Primario extendido esencial				
Referencias	RF-2.2				
Precondición	El CU-21 tiene que haberse realizado				
Postcondición	El lápiz vibrador que hay en la escena desaparece				
Autor	Andrés Molina López	Fecha	29-08-19	Versión	1.0
Propósito					
Quitar el lápiz vibrador de la escena cuando no sea necesario.					
Resumen					
El participante hace que la palma de su mano izquierda real deje de apuntar hacia arriba o cierra los dedos, el lápiz vibrador es destruido de la escena.					
Curso Normal					
1	El participante cierra los dedos o gira la palma de la mano izquierda para que no apunte hacia arriba		2	Destruye el lápiz vibrador	

5.2.3 Diagramas de secuencia

Los diagramas de secuencia describen la colaboración entre actores y objetos para llevar a cabo una determinada operación, remarcando la ordenación temporal de los mensajes que se mandan unos con otros. Son útiles porque permiten explicar en un mayor nivel de detalle que los casos de uso como se realiza una función. Así que, debido a que algunos casos de uso no permiten explicar en detalle cómo se realiza la acción, detallando todas las acciones que realiza

el sistema, se muestran los diagramas de secuencia de los casos de uso más complejos: CU-6, CU-18 y CU-19.

5.2.3.1 CU-6: Iniciar juego

En el diagrama de secuencia (Figura 15) se puede ver el proceso de iniciar el juego, lo que ocurre mientras este se ejecuta y la parte final en la cual se termina. La zona dentro del bucle serían los casos de uso 10 y 11, y lo que hay justo al final del bucle corresponde al CU-12. Adicionalmente, dentro del bucle, aunque no se muestra en el diagrama, también hay que considerar que el CU-3 se va a estar realizando, por lo que la condición de “!destruido” es necesaria en el condicional del *opt*.

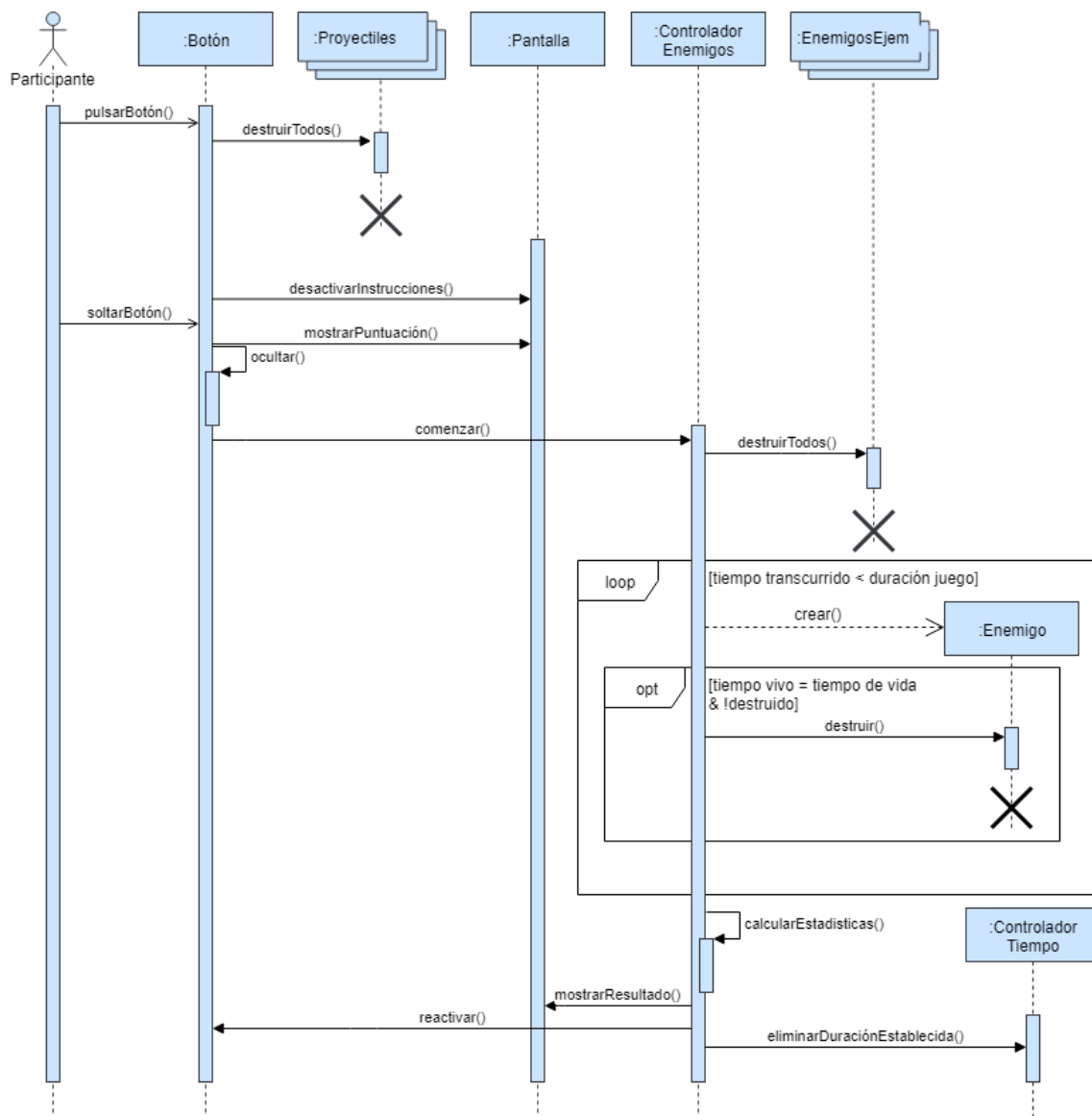


Figura 15: Diagrama de secuencia del CU-6

5.2.3.2 CU-18: Leer fichero de órdenes

En el diagrama (Figura 16) se puede ver como se trata la lectura del fichero de órdenes, línea a línea, comprobando si estas son direcciones absolutas de imágenes u órdenes, y como actúa el sistema en cada caso.

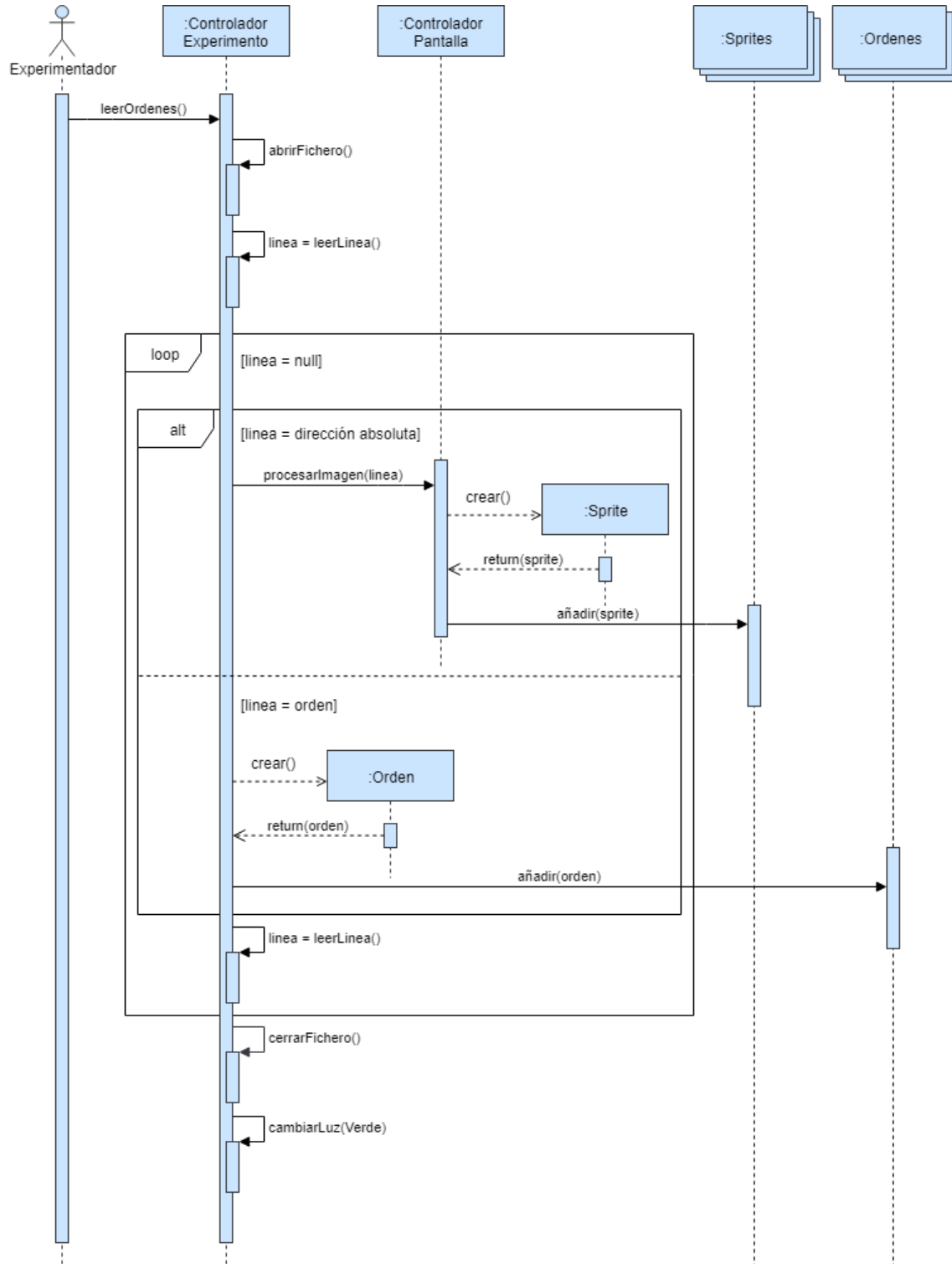


Figura 16: Diagrama de secuencia del CU-18

5.2.3.3 CU-19: Ejecutar órdenes

En el diagrama (Figura 17) se puede ver la secuencia que se genera cuando el experimentador pulsa la tecla Intro para que se ejecuten las órdenes. En el diagrama aparecen el controlador del guante y del polígrafo porque, aunque están en dispositivos externos al sistema es necesario mandarles mensajes.

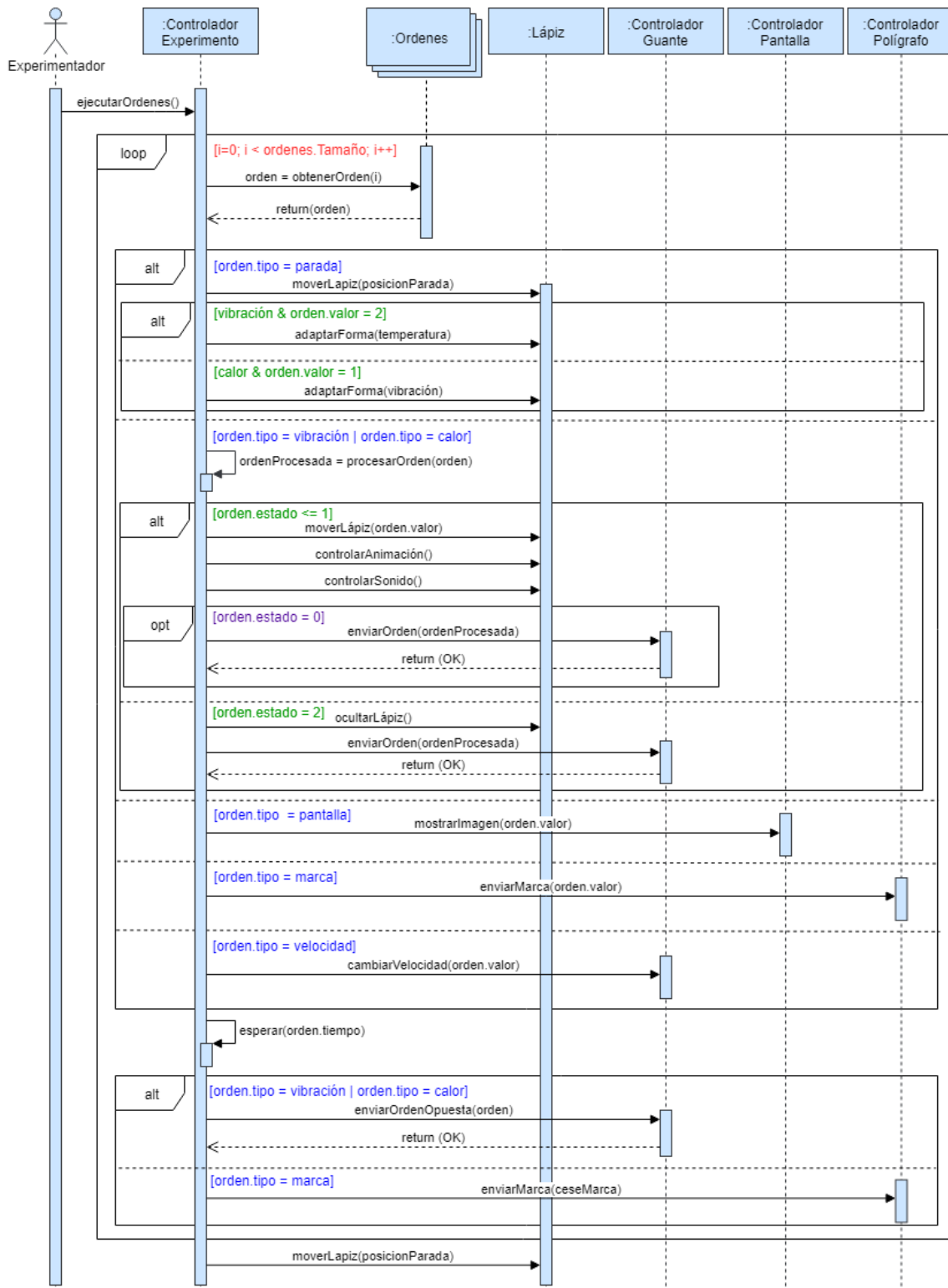


Figura 17: Diagrama de secuencia del CU-19

5.3 Diseño del fichero de órdenes

Como ya se ha comentado anteriormente, permitir que las órdenes sean completamente personalizables por parte de los investigadores es un requisito fundamental del sistema. Además, esto se tiene que poder realizar sin necesidad de acceder y modificar el código fuente ya que los investigadores no están obligados a tener nociones de programación, por lo tanto, las órdenes tienen que cargarse desde un fichero externo al sistema. Adicionalmente, dicho fichero tiene que ser de tipo txt por decisión del grupo de investigadores, para así poder trabajar con él de manera cómoda y que les sirva con otras finalidades.

Teniendo en cuenta estas restricciones, lo primero es establecer una serie de convenciones sobre como llamar al fichero y donde situarlo, para que el programa pueda localizarlo sin problema. Así, el nombre de este tiene que ser *Ordenes.txt* y debe estar en una carpeta llamada *VRExperiment* situada en el escritorio del ordenador donde se ejecute el programa. Lo siguiente a establecer es la estructura que va tener el fichero internamente. Para esto, se ha decidido que en cada línea solo se puede una imagen u orden, y no puede haber líneas en blanco. Inicialmente, se añaden las direcciones absolutas de las imágenes que se quieren mostrar por pantalla (RF-2.4 y RF-2.6.4), las cuales tienen que estar en formato .png (RNF-2.2), y las “\” de Windows en las direcciones hay que cambiarlas a “/” para que así el programa pueda leerlas. Las imágenes se almacenan en el orden en el que están escritas sus direcciones en el fichero, por lo tanto, cuando haya que utilizar una, se especificará usando el número de posición que tiene la dirección en el fichero en orden descendente, y comenzando los números por el 0. Una vez introducidas todas las imágenes, se introduce la secuencia de órdenes, poniendo una por línea, como ya se ha mencionado.

Las órdenes permiten definir 6 tipos de acciones diferentes: poner el lápiz en posición de parada (delante de la mano alineado con el dedo corazón o casi), dar vibración en una posición determinada, dar calor en una posición determinada, mostrar una imagen por pantalla, cambiar la velocidad de los vibradores, mandar una marca al convertidor del polígrafo. La sintaxis diseñada para unificar todas estas acciones con una sola estructura es:

Acción;Estado;Valor;(Temperatura);Tiempo

A cada parámetro le corresponde una letra en mayúscula o un valor numérico, y el parámetro de temperatura solo hay que rellenarlo cuando la acción es para transmitir calor, ya que es necesario especificar hasta que temperatura se quiere calentar el termopar, según indica la documentación del controlador del guante.

5.3.1 Parámetros de las órdenes

Cada parámetro define un aspecto de la orden, y tiene un significado y valores específicos, los cuales son los siguientes:

- **Acción:** este primer parámetro es el que define qué tipo de instrucción de las seis posibles se quiere realizar. Su valor es una letra en mayúscula que indica que acción se quiere llevar a cabo:
 - S -> poner el lápiz en posición de parada.
 - V -> dar vibración en una posición determinada.
 - C -> dar calor en una posición determinada.
 - P -> mostrar una imagen en la pantalla.

- A -> determinar la velocidad de los vibradores
- M -> mandar una marca al polígrafo.
- **Estado:** este parámetro solo es importante para las acciones de dar vibración o calor, ya que define si se pretende que la acción se haga completa, es decir, tanto en la VR como el guante, o parcial, solo en la VR o solo en el mundo real. Para el resto de acciones este parámetro carece de repercusión, pero aun así tiene que ser rellenado para que la orden se procese correctamente. Los valores posibles son:
 - 0 -> la acción se hace en la VR y se transmite al guante. Además, es el valor que se le asigna en el resto de acciones.
 - 1 -> la acción solo se hace en la VR.
 - 2 -> para que solo se mande la acción al guante, pero en el mundo virtual no pase nada. Además, el lápiz deja de verse en el mundo virtual.
- **Valor:** es el parámetro más importante junto con el de acción, ya que para cada tipo de esta indica una cosa:
 - S -> en esta acción sirve para saber qué tipo de acción se va a realizar en la siguiente orden, ya que es en esta acción es en la que el lápiz se adapta para dar vibración o calor. Así pues, para esta acción, el parámetro tiene los siguientes valores:
 - 0 -> la siguiente acción no requiere del uso del lápiz.
 - 1 -> la siguiente acción es una acción de vibración.
 - 2 -> la siguiente acción es una acción de calor.
 - V -> en este caso indica el número de la posición donde se quiere colocar el lápiz para que de la vibración (Figura 18).
 - C -> el parámetro tiene los mismos valores que para la acción de dar vibración.
 - P -> en esta acción especifica la imagen que se quiere mostrar por la pantalla que tiene el participante enfrente dentro de la VR. Las imágenes están almacenadas en un vector, en el orden en el que se han introducido en el fichero de órdenes, así que, por ejemplo, para mostrar por pantalla la primera imagen que está puesta en el fichero, hay que asignarle el valor 0 al parámetro, y así sucesivamente.
 - A -> para esta acción lo que indica es la velocidad a la que se quiere que funcionen los vibradores del guante. Por lo que, en este caso, está delimitado por las capacidades del guante, de tal manera que solo puede tomar valores entre 40 y 80 sin decimales.
 - M -> el valor del parámetro es la marca que se le manda al convertidor del polígrafo, por lo tanto, tiene que ser un entero entre 0 y 255.
- **Temperatura:** como ya se ha mencionado anteriormente, este parámetro solo se utiliza cuando la acción es de tipo C, y sus valores pueden oscilar entre 0 y 70 sin decimales, aunque tal y como está diseñado los termopares, estos se apagan una vez se alcanza la temperatura deseada, y como están pegado a la mano, solo se emite calor si se manda una temperatura superior a la de la mano.
- **Tiempo:** finalmente, este último parámetro especifica el tiempo que se quiere estar realizando la acción. Por lo tanto, le daremos un valor distinto de 0 en las acciones:
 - S -> el tiempo que el lápiz va a estar en reposo sin producir ninguna estimulación, y aprovechando para adaptarse a la siguiente orden. El

tiempo tiene que ser de al menos 1 segundo ya que es lo que tarda la animación de la adaptación en efectuarse.

- V -> es el tiempo que se está estimulando la zona indicada.
- C -> igual que para V.
- M -> si se quiere prolongar el valor de la marca en el tiempo, aunque esto impide que se sigan ejecutando órdenes hasta que el tiempo indicada haya transcurrido.

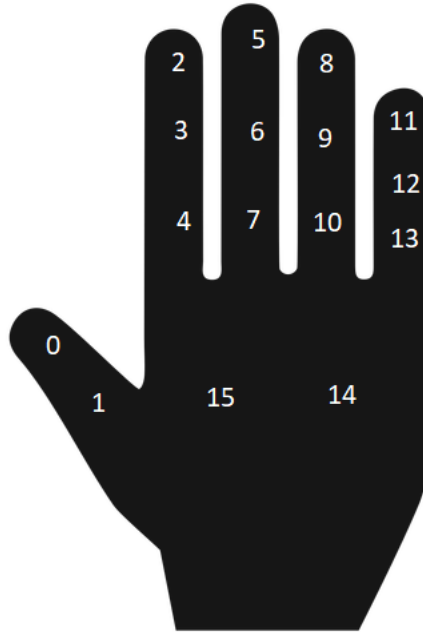


Figura 18: Posiciones donde se puede colocar el lápiz con sus correspondientes valores

Una vez explicados los posibles valores para cada parámetro, cabe destacar que después de cada acción V, C y M, la cuales requieren de la especificación de un tiempo 0 o positivo, el sistema manda automáticamente una acción opuesta a la descrita (se puede ver en la Figura 17) para ahorrar líneas en el fichero y que sea más cómodo para el investigar. Es decir, por ejemplo, si se manda una vibración, una vez transcurrido el tiempo se manda la señal de que se pare, si se manda una temperatura, transcurrido el tiempo se manda una temperatura 0 al mismo termopar para que deje de emitir calor, y si se manda una marca, acto seguido se manda una marca 0 que es la que detiene la marca en el polígrafo.

Finalmente, la última consideración a tener en cuenta a la hora de redactar el fichero, es que antes de cada orden de vibración o calor, tiene que ir una orden de parada, para permitir que el lápiz se adapte correctamente a la acción y se eviten errores de posicionamiento. Adicionalmente, aunque no es necesario, es recomendable poner también la orden de parada detrás de cada una de estas acciones, para así devolver el lápiz a su posición inicial y que se ancle a la mano mientras se ejecutan otras órdenes (Figura 19).

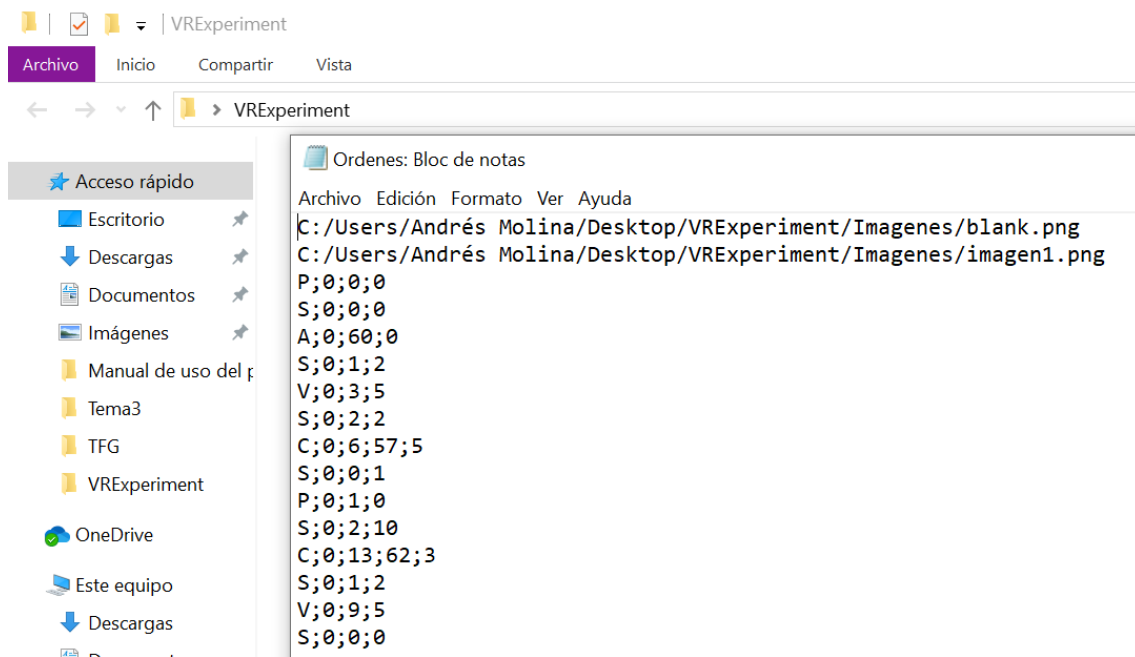


Figura 19: Ejemplo de fichero de órdenes

5.4 Diagramas de clase

Por último, para completar el diseño del programa se van a ver los diagramas de clase de este. Los diagramas de clase describen la estructura del sistema mostrando un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Por lo tanto, está formado por:

- **Clases:** descripciones de conjuntos de objetos que comparten atributos, operaciones, relaciones y semántica.
- **Interfaces:** especifican el servicio de las clases.
- **Relaciones:** modelan la forma en la que los elementos estructurales (clases e interfaces) se conectan entre sí. Pueden ser relaciones de asociación, generalización, dependencia o realización.

Una vez más, la aplicación se va a dividir en sus dos fases para establecer las relaciones entre sus clases, ya que los objetos de una escena no existen en la otra en la mayoría de los casos, o en caso de existir no tienen el mismo comportamiento. Así, va a haber un diagrama de clases para la fase de entrenamiento, y otro para la fase de simulación, siendo ambos totalmente independientes uno del otro. Aunque, excepcionalmente, hay una clase que se utiliza en ambas escenas por igual, que es la que se encarga de manejar el menú de pausa. Por lo tanto, esta clase se muestra separada en un diagrama único para ella (Figura 20).

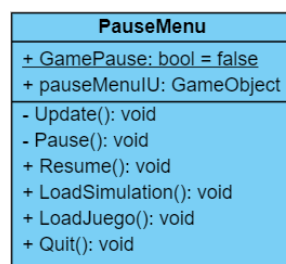


Figura 20: Diagrama de clases - clase común para ambas fases

5.4.1 Fase de entrenamiento

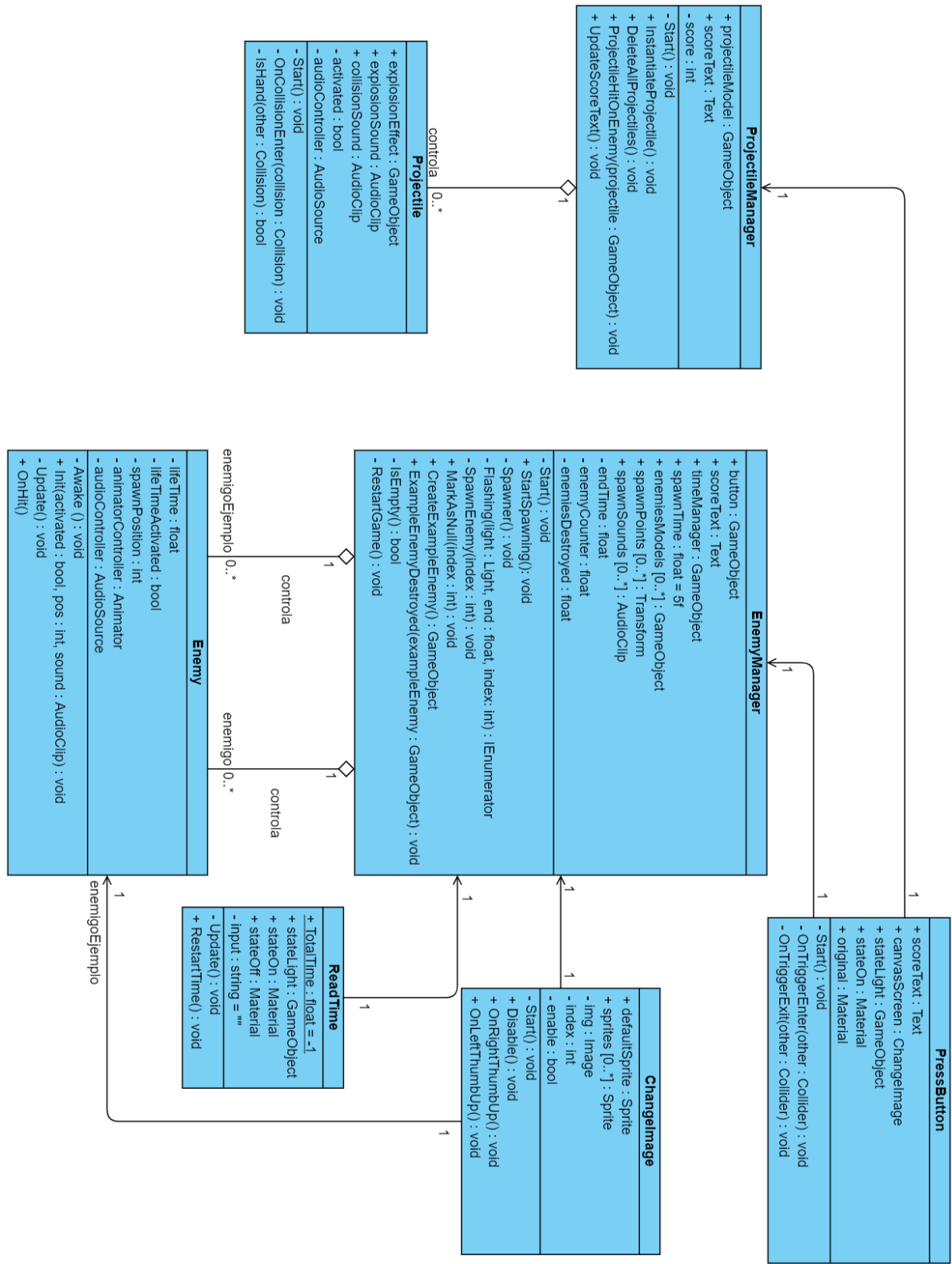


Figura 21: Diagrama de clases de la fase de entrenamiento

5.4.2 Fase de simulación

Como se puede ver en el diagrama de clase (Figura 22), algunas de las clases no se relacionan con ninguna de las demás. Esto se debe a que el diagrama solo incluye las clases que hace falta implementar, obviando las que viene implementadas en los paquetes importados al proyecto. Además, la mayoría de las funcionalidades de esta fase son independientes unas de otras en lo que respecta a su programación, por lo tanto, es normal que no se relacionen. Así, por ejemplo, la clase *MyMessageListener* es simplemente un punto de escucha que se actualiza cada *frame* para ver si se ha recibido algún mensaje por el puerto de serie del controlador del guante, pero es independiente del resto de funcionalidades ya que el mensaje recibido solo se ha utilizado durante el desarrollo para comprobar que el guante funcionaba correctamente.

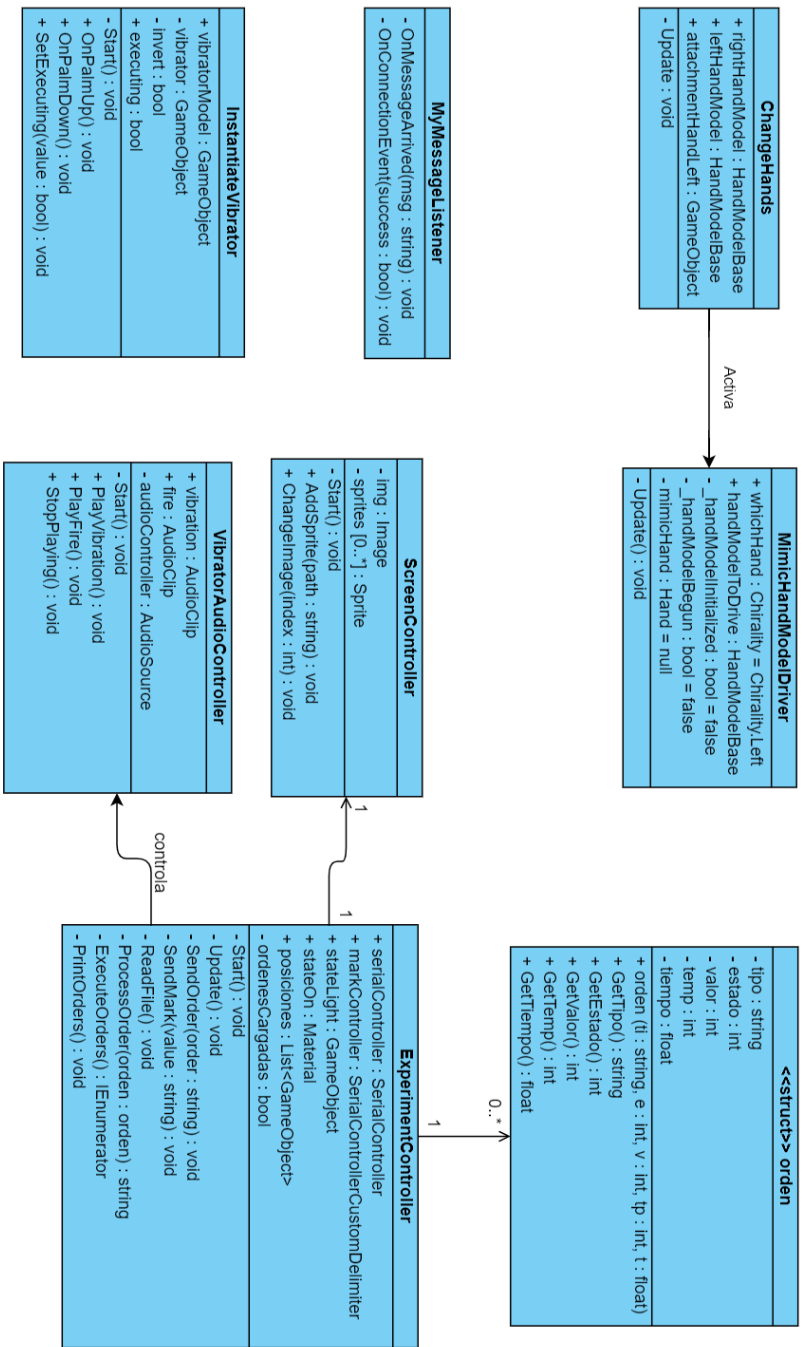


Figura 22: Diagrama de clase de la fase de simulación

Capítulo 6

“I created the OASIS because I never felt at home in the real world”

— Ernest Cline, *Ready Player One*

Implementación

En este capítulo se van a comentar las herramientas usadas para el desarrollo del sistema, así como las bibliotecas externas que han sido necesarias integrar en él. Además, se verán algunas particularidades del código desarrollado para cumplir con los objetivos especificados.

6.1 Herramientas utilizadas

En el capítulo 3 se presentaron varios motores gráficos que facilitaban la tarea de desarrollar aplicaciones de VR, así pues, para la implementación de ésta, se ha decidido aprovechar la potencia de estos y utilizar Unity 3D. La decisión de utilizar Unity por encima de UE4 o CryEngine 5 es debido a la mayor facilidad de su curva de aprendizaje, así como a la disponibilidad de una mejor documentación, y de una mayor cantidad de *assets* en su tienda, que permiten darle mayor nivel de detalles al sistema sin necesidad de conocimientos de *sculpting* digital o modelado 3D. Adicionalmente, Unity también proporciona una herramienta para crear animaciones, con lo que hace innecesario el uso de otros programas como Maya o 3ds Max para este propósito. Finalmente, la versión utilizada ha sido la 2017.4.6 (64-bits), la cual se puede encontrar en <https://unity3d.com/es/get-unity/download/archive>, y se ha utilizado al licencia personal gratuita, para cumplir con el propósito de reducir los costes al mínimo.

Como editor de código se ha utilizado Microsoft Visual Studio Community 2017, en su versión 15.9.6, el cual permite crear y modificar scripts en lenguajes como C++, C#, JS, etc... de manera sencilla. Además, Unity permite asignar esta IDE como su herramienta por defecto para editar scripts. De hecho, desde la versión 2018.1 de Unity viene por defecto.

El lenguaje de programación usado ha sido C#, ya que Unity lo maneja de manera nativa, debido a que es uno de lenguajes estándar de la industria. C# es un lenguaje de programación orientado a objetos, con compatibilidad para programación orientada a componentes, cuya sintaxis viene de la familia de lenguajes C y utiliza el modelo de objetos de la plataforma .NET, lo que hace de este una mezcla entre Java y C/C++.

En cuanto a los temas de virtualización, las gafas de VR utilizadas ha sido las Oculus Rift, ya que eran de las que disponía el equipo de investigación en el CIMCYC. Y para la virtualización de las manos se ha utilizado Leap Motion como ya se ha comentado en secciones anteriores.

Finalmente, para confeccionar la documentación se ha utilizado Microsoft Word, junto con las herramientas gratuitas online Visual Paradigm y Draw.io para hacer los diagramas UML.

6.2 Bibliotecas externas

Unity Core Assets 4.4.0 es la biblioteca más importante utilizada ya que es la que permite integrar Leap Motion en Unity, adicionalmente, junto con esta se ha integrado también Leap Motion Interaction Engine ya que es la que habilita la interacción natural con los objetos, es decir, cogerlos, lanzarlos y demás acciones que hacen que la experiencia sea lo más similar a la real. Ambas pueden encontrarse de manera gratuita en <https://developer.leapmotion.com/unity#5436356>.

Ardity 1.1.0 es la segunda biblioteca utilizada con mayor interés para el sistema, ya que es la que permite conectar Arduino a Unity mediante un puerto de serie, lo cual es necesario para comunicar el sistema con el guante que estimula la mano del participante, cuyo controlador es una placa de Arduino Uno. Asimismo, la comunicación con el convertidor del polígrafo también es realizada con esta biblioteca, ya que este también es otra placa de Arduino. Esta biblioteca la podemos encontrar gratuitamente en <https://ardity.dwilches.com/> junto con su documentación o incluso en la tienda de Unity.

El resto de bibliotecas utilizadas son para mejorar el nivel de detalle del sistema y todas se pueden encontrar en la tienda de Unity:

- Level 1 Monster Pack (gratuita) -> se ha utilizado para darle forma a los enemigos y que estos sean llamativos para el usuario.
- Simple FX – Cartoon Particles (gratuita) -> su finalidad dentro del sistema es permitir crear una explosión cuando un enemigo es eliminado por un proyectil y generar una llama en la punta del lápiz vibrador cuando este pase al modo de dar calor, para que el efecto visual advierta al usuario de lo que pasa.
- Leap Motion Realistic Hands (de pago) -> aunque esta biblioteca no cumple con el requisito de utilizar recursos gratuitos, es necesaria incluirla en el proyecto, porque como se comentó en el capítulo 4, la mano falsa (virtual en este caso) tiene que ser lo más similar posible a la humana. Así pues, la falta de modelos gratuitos compatibles con Leap Motion en internet ha hecho que se requiera de esta biblioteca para resolver este problema ya que ofrece unos modelos de manos bien modelados e idénticos a una mano real.

Finalmente, aunque no son bibliotecas, también se ha importado un modelo gratuito de una lámpara de pie, que se puede encontrar en <https://www.turbosquid.com/3d-models/3d-floor-lamp-1161065>, para darle más realismo a la iluminación de la escena y que el participante pueda ver de dónde viene el foco de luz. Los sonidos de las escenas para los proyectiles, los enemigos, las explosiones, la vibración del lápiz y el crepitar de las llamas han sido descargados gratuitamente de <https://freesound.org/> y retocados en caso de necesidad con Audacity, el cual también se puede descargar gratis desde su página web.

6.3 Implementación de funciones

A continuación, se van a ver detalladamente las funciones cuya implementación es compleja pero necesaria para cumplir con los objetivos del proyecto.

6.3.1 Leer duración del juego

Esta función (Figura 23) pertenece a la fase de entrenamiento y es la encargada de leer las teclas que se pulsán en el teclado. Su implementación es interesante, ya que se requiere que se puedan introducir valores en cualquier momento, y que solo acepte determinados caracteres.

Para cumplir con el requisito de que se puedan introducir valores en cualquier momento, todo el funcionamiento ha sido creado dentro de la función nativa de Unity, *Update()*. Esta función es invocada una vez cada *frame*, lo que significa que se va a estar comprobando periódicamente cada poco tiempo. A continuación, el procedimiento consiste en crear un bucle que lea cada carácter que se ha introducido en el *frame* del momento. Para ello, se recogen los datos de la función *Input.inputString*, la cual contiene los caracteres en ASCII introducidos en el *frame*. Finalmente, se filtra cada carácter con una estructura *if-esle*, ya que hay que controlar los 2 caracteres especiales almacenados en *Input.inputString*, los cuales son “\b” y “\n” o “\r” y que van a servir para borrar valores de la cadena que se pretende generar con la duración del juego, y para indicar que se ha terminado de introducir la cadena y se quiere convertir en número, respectivamente. Adicionalmente, también se controlan los espacios introducidos ya que es la tecla especial asignada para que el experimentador pueda crear enemigos de ejemplo. Para cada otro carácter que no sea ninguno de los mencionados anteriormente, se comprueba que su código ASCII está comprendido entre el 48 y el 57 ya que son los valores que corresponden a los números del 0 al 9 y si se cumple esta condición, se añade a un *string* inicializado en un principio a vacío (“”).

Cuando se lee un salto de línea (“\n” o “\r”), lo primero que se hace es comprobar si el valor de la duración del juego es negativo, ya cuando su valor no está establecido o se reinicia al juego por haberse completado, este se marca como -1. En caso de que así sea, la cadena generada con los valores filtrados introducidos, se convierte en número y se comprueba que sea mayor que 0, ya que el tiempo de juego tiene que ser estrictamente positivo. Así pues, en caso de ser 0 todo lo introducido será descartado y se comenzará de nuevo con la cadena. En caso de no ser 0 se avisará al jugador de puede comenzar cambiando un led que tiene situado encima de la pantalla blanca que tiene enfrente de rojo a verde.

```
void Update () {
    foreach (char c in Input.inputString)
    {
        if (c == '\b') // has backspace/delete been pressed?
        {
            if (input.Length != 0)
            {
                input = input.Substring(0, input.Length - 1);
            }
        }
        else if ((c == '\n') || (c == '\r')) // enter/return
        {
            if (TotalTime < 0)
            {
                print("Duración introducida: " + input);
                TotalTime = float.Parse(input);
                if (TotalTime > 0)
                {
                    stateLight.GetComponent<Renderer>().material = stateOn;
                }
                else
                {
                    print("Duración introducida invalida - Entrada limpiada");
                    RestartTime();
                }
            }
            else
            {
                print("El tiempo ya ha sido introducido");
            }
        }
        else if (c == ' ') {
            enemyManager.GetComponent<EnemyManager>().CreateExampleEnemy();
        }
        else
        {
            if (c >= 48 && c <= 57)
            {
                input += c;
            }
        }
    }
}
```

Figura 23: Función para leer la duración del juego

6.3.2 Leer fichero de órdenes

Leer un fichero externo es un requisito primario del proyecto, para permitir la personalización de la secuencia de órdenes que sigue la fase de simulación. Así, aunque se han establecido varias restricciones sobre donde tiene que estar el fichero situado y como tiene que llamarse este, es posible que el programa se utilice en diferentes ordenadores, por lo tanto, tiene que ser capaz de localizar el escritorio de la sesión activa en cada momento. Debido a esto, la función (Figura 24) lo primero que hace es establecer la dirección de la cual se quiere leer. Para ello se hace uso de la función disponible en .NET Core a partir de la versión 2.0, `Environment.GetFolderPath(Environment.SpecialFolder.Desktop)`, que mediante la variable de entorno obtiene la ruta de acceso a las carpetas especiales del sistema que tiene definidas en la enumeración *SpecialFolder*, entre las que se encuentra el escritorio. Una vez estable el directorio, se intenta leer el fichero mediante la clase *StreamReader* de .NET, y si se consigue, se van leyendo línea a línea hasta que este se termine y se cierra el objeto, liberando todos los recursos del sistema que tiene asociados. Para cada línea leída, se comprueba si contiene la cadena “.png”, para saber si es una dirección a una imagen de las que se quiere usar, dado que en tal caso se invoca a la función que la va leer, convertir en *sprite* para que pueda ser utilizable por el programa, y almacenarla. Y, en caso de no ser una dirección a una imagen, tiene que ser una orden porque son los dos únicos tipos de datos que puede haber en el fichero según el diseño establecido. Así que la orden se va a separar por los “;” y dependiendo de su longitud se detecta si es una orden de temperatura o no, ya que, en caso de no serlo, dicho valor se tiene que inicializar a -1 para que no haya inconsistencias en la estructura de datos encargada de almacenar las órdenes. Finalmente, en caso de que no se pueda leer el fichero, se manejará la excepción mostrando por consola el mensaje de error, cosa que solo se ve cuando el sistema se está utilizando desde Unity para comprobar errores.

```
void ReadFile()
{
    //string path = "C:/Users/rv-lab/Desktop/VRExperiment/Ordenes.txt";
    string dir = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
    string path = dir + @"\VRExperiment\Ordenes.txt";
    Debug.Log("Lectura de ordenes desde: " + path);
    try
    {
        using (StreamReader sr = new StreamReader(path))
        {
            string line = sr.ReadLine();
            while (line != null)
            {
                if (line.Contains(".png"))
                {
                    screenCanvas.AddSprite(line);
                }
                else
                {
                    string[] entries = line.Split(';');
                    if(entries.Length == 4) // if the order hasn't temp value we assign a -1 to it
                        ordenes.Add(new orden(entries[0], int.Parse(entries[1]), int.Parse(entries[2]), -1, float.Parse(entries[3])));
                    else
                        ordenes.Add(new orden(entries[0], int.Parse(entries[1]), int.Parse(entries[2]), int.Parse(entries[3]), float.Parse(entries[4])));
                }
                line = sr.ReadLine();
            }
            sr.Close();
        }
        ordenesCargadas = true;
        //PrintOrders();
    }
    catch (Exception e)
    {
        Console.WriteLine("The process failed: {0}", e.ToString());
    }
}
```

Figura 24: Función para leer el fichero de órdenes

La función para crear el *sprite* y almacenarlo a partir de la dirección de una imagen (Figura 25), la cual está directamente asociada a esta función, también es interesante de explicar, ya que su implementación no es trivial. Así, lo primero que hace esta función es comprobar que la dirección que se le ha pasado existe, y en tal caso, se leen todos sus bytes y se almacenan en un vector. Acto seguido se crea una textura 2D vacía, con cualquier tamaño

para las dimensiones, ya que seguidamente se va a invocar a la función pública *LoadImage()* perteneciente a la clase de *Texture2D*, la cual reemplaza las dimensiones con las de la imagen que se le pasa a través de la secuencia de bytes. Una vez se tiene la textura bien configurada, se crea un *sprite* y a su constructor se le pasa dicha textura, se le especifica que las dimensiones sean las de la textura y se establece el centro del *sprite* de manera que corresponda con el centro de la textura, mediante un *vector2d(0.5f, 0.5f)*, ya que el (0f, 0f) corresponde a la esquina inferior izquierda, y el (1f, 1f) corresponde a la esquina superior derecha. Finalmente, el *sprite* generado se añade a lista de sprites disponibles.

```
public void AddSprite(string path) // Es necesario poner el path absoluto
{
    Texture2D tex = null;
    byte[] fileData;

    if (File.Exists(path))
    {
        fileData = File.ReadAllBytes(path);
        tex = new Texture2D(2, 2);
        tex.LoadImage(fileData);

        //Create new Sprite from the Loaded PNG
        Sprite sp = new Sprite();
        sp = Sprite.Create(tex, new Rect(0.0f, 0.0f, tex.width, tex.height), new Vector2(0.5f, 0.5f));

        //Save sprite in the list
        sprites.Add(sp);
    }
}
```

Figura 25: Función para crear un sprite a partir de la dirección de una imagen

6.3.3 Voltar modelo de la mano

Finalmente, la función para voltear el modelo de la mano izquierda (Figura 26), la cual sirve para crear la situación incongruente necesaria en la fase de simulación, requiere de la utilización de la SDK de Leap Motion para Unity. Aunque en principio solo es necesaria para la mano izquierda, se ha programado de tal manera que se pueda intercambiar su uso al de la mano derecha mediante el editor de Unity sin necesidad de modificar el código de esta función.

Para empezar la función debe actualizar el estado en el que se ve el modelo de la mano constante, lo cual implica que tiene que ejecutarse cada *frame*, por lo que nuevamente, se ha introducido toda la lógica de la transformación dentro de la función *Update()* de Unity, ya que se ejecuta al comienzo de cada *frame*. Lo primero que se va a hacer dentro de la función es obtener las propiedades de la mano indicada en el editor de Unity, mediante el *enum Chirality* disponible en la biblioteca *Leap.Unity*, la cual permite especificar si es la izquierda o la derecha (para el objetivo del proyecto es la izquierda). Para obtenerlo se va usar la función *Get()* que provee la clase *Hands* dentro de la misma biblioteca. Para continuar, hay que asegurarse de que se han obtenido bien los datos y que la variable no es nula. En dicho caso, se comprueba si la variable *mimicHand* de tipo *Hand* (clase disponible en *Leap.Unity*) inicializada a nula al comienzo de la ejecución del programa, lo sigue siendo, ya que en tal caso habrá que usar el constructor por defecto de la clase *Hand* para inicializarla correctamente. Independientemente, si se requiere inicializarla o no, el siguiente paso será copiar en ella tanto las propiedades de la mano izquierda obtenidas como las del brazo unido a ella, mediante las funciones de copia, *CopyFrom()*, que proporcionan las clases *Hand* y *Arm* (por extensión de la clase *Bone*, ambas pertenecientes a *Leap.Unity*), respectivamente. El siguiente paso es obtener las transformaciones de la mano, su posición y rotación, para así, a continuación, poder aplicarle una transformación al modelo de mano que se está construyendo (*mimicHand*), mediante la función *SetTransform*, a la cual se le pasa la misma posición que se ha obtenido (el centro de la

posición de la mano), y la rotación obtenida multiplicada por 180 en el eje Z para que así la mano se dé la vuelta, que es lo que se pretende. Una vez configurado el modelo de la mano y transformado correctamente, solo falta asignárselo al modelo que desde el editor se le indique que corresponde a la mano izquierda, el cual está almacenado en la variable *handModelToDrive*. Para esto, primero se comprueba si ambas variables son distintas de nulo para evitar posibles errores, y por si acaso, es la primera vez que se ejecuta esta función, se inicializa el modelo almacenado en *handModelToDrive* con la función *InitHand()* y se añade a las representaciones de manos que maneja Leap Motion internamente para hacer su seguimiento, con la función *BeginHand()*. Por último, se llama a la función *UpdateHand()* sobre el modelo especificado ya modificado, para que Leap Motion lo actualice correctamente, y se vea en la pantalla con la modificación de inversión aplicada.

```
private void Update()
{
    // Get a hand from the standard tracking pipeline.
    var sourceHand = Hands.Get(whichHand);

    if (sourceHand != null)
    {
        // Copy data from the tracked hand into the mimic hand.
        if (mimicHand == null) { mimicHand = new Hand(); }
        mimicHand.CopyFrom(sourceHand);
        mimicHand.Arm.CopyFrom(sourceHand.Arm); // Capsule Hands like to have Arm data too.

        // Figure out what rotation to use for the mimic hand.
        var handRotation = mimicHand.Rotation.ToQuaternion();
        var handPosition = mimicHand.PalmPosition.ToVector3();

        // Transform the copied hand so that it's centered on this object's transform.
        mimicHand.SetTransform(handPosition, handRotation * Quaternion.Euler(0, 0, 180f));
    }

    // Drive the attached HandModel.
    if (mimicHand != null && handModelToDrive != null)
    {
        // Initialize the handModel if it hasn't already been initialized.
        if (!_handModelInitialized)
        {
            handModelToDrive.InitHand();
            _handModelInitialized = true;
        }

        // Set the HandModel's hand data.
        handModelToDrive.SetLeapHand(mimicHand);

        // "Begin" the HandModel to represent a 'newly tracked' hand.
        if (!_handModelBegun)
        {
            handModelToDrive.BeginHand();
            _handModelBegun = true;
        }

        // Every Update, we call UpdateHand. It's necessary to call this every update
        // specifically for CapsuleHands, which uses manual GL calls to render rather than
        // a standard MeshRenderer.
        handModelToDrive.UpdateHand();
    }
}
```

Figura 26: Función para voltear el modelo de la mano

Capítulo 7

“Nobody dares to solve problems-because the solution might contradict your philosophy”

— Michael Crichton, *State of Fear*

Conclusión

Finalmente, en este último capítulo, se repasan los objetivos planteados al comienzo del proyecto comprobando si ha sido posible cumplirlos. Asimismo, se exponen los resultados obtenidos por el estudio piloto realizado con el software implementado para verificar la eficacia de este. Por último, se revisan las habilidades adquiridas en el desarrollo del proyecto, complementándolas con las obtenidas en la consecución del grado, y se proponen posibles vías futuras del proyecto.

7.1 Revisión de los objetivos del proyecto

En el primer capítulo se propusieron unos objetivos clave que tenían que ser realizados para que este fuese funcional y abordase todas las necesidades requeridas por el equipo de equipo de investigación de Psicofisiología Humana y Salud con la finalidad de poder recrear el paradigma de la ilusión de la mano de goma en un entorno virtual. Así, tras meses de desarrollo, y a la vista de los resultados obtenidos, se puede llegar a concluir que todos los objetivos planteados han sido cumplidos con éxito (Tabla 30).

Tabla 30: Consecución de los objetivos del proyecto

Objetivo	Prioridad	Cumplido	Detalles
Emular paradigma de la RHI	Principal	Sí	Con el programa es posible recrear el paradigma de la RHI incluyendo sus dos fases tradicionales.
Integrar de Oculus Rift	Secundario	Sí	El programa no solo reconoce el dispositivo Oculus Rift como visor de entorno de VR si no que requiere de un HMD reconocible por Unity VR para su correcto funcionamiento.
Integración de Leap Motion	Secundario	Sí	Gracias a la biblioteca de Unity Core Assets 4.4.0 del propio fabricante del dispositivo, este se ha podido integrar sin dificultad.
Implementación fase de entrenamiento con un juego	Secundario	Sí	A lo largo del capítulo 5 se ha visto el diseño de este, así como en que consiste, y en el capítulo 6 se muestra una de sus funciones, más el hecho de que el estudio piloto se ha hecho sobre esta fase.
Introducción manual del tiempo del juego	Terciario	Sí	En la sección 6.3.1 se explica la implementación de esta funcionalidad.
Aparición de enemigos aleatoria, avisando de su aparición	Terciario	Sí	El juego crea enemigos en las posiciones disponibles de manera aleatoria, avisando con una luz roja la posición elegida.

Modelo aleatorio de enemigo	Terciario	Sí	Los enemigos se crean con un modelo aleatorio de entre los que tienen disponibles en un vector.
Sonido aleatorio al aparecer enemigo	Terciario	Sí	Cada enemigo reproduce un sonido elegido de manera aleatoria, de entre los que tiene el programa almacenados en una lista, cuando es creado.
Implementación de la fase de evaluación	Secundario	Sí	En el capítulo 5 se muestra el diseño de esta escena junto con su funcionamiento, además de explicarse como se le configuran sus órdenes para estimular las manos.
Leer instrucciones desde un fichero externo	Terciario	Sí	En la sección 6.3.2 se explica la implementación de esta función con todos sus detalles.
Flexibilidad para decidir las imágenes a mostrar	Terciario	Sí	La sección 5.3 explica cómo se ha aprovechado el fichero de órdenes para lograr este objetivo, y en la sección 6.3.2 se explica su implementación.
Enviar señales por el puerto de serie a Arduino	Terciario	Sí	Gracias a la biblioteca externa Ardity, se ha podido implementar esta funcionalidad fácilmente ya que se han aprovechado sus funciones de paso de mensajes.
Agradar/reducir tamaño de las manos	Terciario	Sí	Mediante el uso del teclado, el experimentador puede modificar el tamaño de los modelos sin problemas, como se explica en el diseño en el capítulo 5.
Invertir la posición de la mano izquierda	Terciario	Sí	En la sección 6.3.3 se explica con todo detalle cómo ha sido necesario la utilización de la SDK de Leap Motion para implementar esta funcionalidad.
Permitir cambiar de una escena a otra	Secundario	Sí	Mediante un menú de pausa desplegable se permite que el experimentador cambie de una escena otra entre las 2 disponibles a su libre voluntad.

7.2 Resultados obtenidos

Al final del capítulo cuarto, se pone de manifiesto que el sistema ha sido probado en un estudio piloto con un reducido grupo de participante, en el cual se comprueba la capacidad de este para inducir el *embodiment* con respecto al procedimiento tradicional.

La hipótesis desde la que se partía a la hora de realizar el estudio, es que los resultados que diese el programa tenían que ser como mínimo iguales que los que se consiguen con el procedimiento tradicional. Tras realizarlo, los resultados obtenidos han demostrado la eficacia del programa para inducir a la RHI, alcanzando este un promedio mejor en la sensación de *embodiment* de los participantes que el obtenido con el procedimiento tradicional. Además, la desviación típica también es menor en los resultados obtenidos por el programa, lo que implica que con el uso del software la sensación de pertenencia de la mano falsa lograda es más similar para todos los participantes. En consecuencia, se puede afirmar que el sistema cumple perfectamente con su propósito, permitiendo el estudio del *embodiment* mediante su utilización.

7.3 Habilidades adquiridas

A lo largo del desarrollado del proyecto, se han puesto en práctica diferentes competencias adquiridas a lo largo del grado. Dado el carácter de Ingeniería del Software del proyecto, se han utilizado las habilidades de análisis y diseño aprendidas en las diferentes asignaturas. A su vez, también se han aprovechado los conceptos de computación gráfica aprendidos en las asignaturas más técnicas de la especialización de software para la correcta jerarquización de los objetos y aplicación de transformaciones a estos.

No obstante, ha sido necesaria la adquisición de conocimientos en el motor gráfico Unity 3D y el lenguaje C# para la creación de sus scripts de cara a personalizar los comportamientos de los objetos y lograr los efectos deseados, aunque gracias a los conocimientos en los lenguajes C++ y Java y del motor gráfico Unreal Engine 4 aprendidos en el grado, la curva de aprendizaje de estas herramientas ha sido más suave al partir de una base sólida. Además, ha sido necesario aprender acerca del funcionamiento de Oculus Rift y profundizar en las capacidades de Leap Motion, tecnologías que está creando nuevos paradigmas de interacción como bien se apunta en la correspondiente asignatura perteneciente a la especialidad de computación y sistemas inteligentes, demostrando así la capacidad para adaptarse a las tecnologías de vanguardia.

Por otra parte, ha sido necesario aprender una serie de conceptos relacionados con la neuropsicología con el fin de entender su funcionamiento y poder aplicarlos virtualmente, así como para poder integrarse en el equipo y comprender las necesidades de este, poniendo de manifiesto la capacidad para trabajar en equipos multimodales.

7.4 Vías futuras

Como se ha podido observar, los objetivos de este proyecto se han cumplido plenamente, permitiendo incluso realizar un pequeño estudio piloto con el software desarrollado. Aunque, esto no implica que tanto el programa como los temas de investigación que permite indagar este, estén cerrados. Así pues, a nivel de software se plantearon un par de funcionalidades que por restricciones de tiempo y su complejidad fueron descartadas, y puestas de cara a futuras investigaciones. Dichas funcionalidades son:

- Mostrar al sujeto un estímulo de vibración en el programa, pero mandar un estímulo de temperatura al guante. Esta funcionalidad permitiría trabajar con temas como el umbral del dolor.
- Hacer que una de las manos imite el comportamiento de la otra, creando la mano que imita en caso de que esta no esté en la escena. Es decir, si solo se utiliza la mano derecha y la izquierda ni siquiera sale en la escena, se crea el modelo de la mano izquierda y se hace que este haga los mismos movimientos que la mano derecha, o los movimientos contrarios. Con esta funcionalidad se podría trabajar con miembros amputados, y avanzar en campos de investigación como las prótesis y síndrome del miembro fantasma.

En cuanto, a temas de VR, el mayor problema que se ha encontrado, sobre todo, en el pequeño estudio piloto realizado, es la dificultad para agarrar objetos virtuales con las manos reales. Esto se debe a que cada persona tiende a coger los objetos de una manera diferente y, además, cuando el objeto no existe realmente la cosa se complica ya que hay quien intenta ajustar su mano a la forma de lo que ve y hay quien cierra el puño introduciendo la mano dentro del objeto. Así pues, aun cuando se avisó a los participantes de como tenían que agarrar los

cubos, estos en última instancia acababan cogiéndolos de cualquier manera, ocasionando que los cubos se quedasen atrapados en los colisionadores de las manos y no se lanzasen o incluso sin por llegar a coger un cubo durante varios segundos al no llegar a tocar el cubo propiamente y por lo tanto no detectarse la colisión entre mano y cubo para producirse el agarre. Por lo tanto, un tema en el que habría que profundizar para futuros desarrollos de VR con una interfaz manual, sin mandos, es en encontrar una solución a este problema, de manera que el hecho de agarrar un objeto sea de la forma más natural posible.

Finalmente, aunque la aplicación desarrollada se ha centrado solo en la virtualización de las manos para su estudio. En posteriores estudios, se podría ampliar este rango, virtualizando otras extremidades o llegando a virtualizar todo el cuerpo y produciendo una ilusión de cuerpo entero, con la que se podría estudiar los efectos del *embodiment* a mayor escala, y tratar temas como la empatía con las minorías.

Bibliografía

- [1] SUTHERLAND, IVAN E., “*The ultimate display*”, in International Federation of Information Processing IFIPS Congress, vol. 2, New York, NY, USA, May 1965, pp. 506–508 [en línea][Fecha de consulta 26 julio 2019]. Disponible en: <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.136.3720>
- [2] RAE ASALE. *Diccionario de la lengua española* [en línea][Fecha de consulta 26 julio 2019]. Disponible en: <https://dle.rae.es/?id=VH7cofQ>
- [3] KILTENI, K., GROTEN, R. and SLATER, M. *The Sense of Embodiment in Virtual Reality* [en línea] Presence Teleoperators & Virtual Environments, 2012 [Fecha de consulta 9 julio 2019] DOI: 10.1162/PRES_a_00124. Disponible en: https://www.researchgate.net/publication/243056510_The_Sense_of_Embodiment_in_Virtual_Reality
- [4] *Fundamentos de Scrum* [en línea][Fecha de consulta 31 de julio 2019]. Disponible en: <https://proyectosagiles.org/fundamentos-de-scrum/>
- [5] *Beneficios de Scrum* [en línea][Fecha de consulta 31 de julio 2019]. Disponible en: <https://proyectosagiles.org/beneficios-de-scrum/>
- [6] CRAIG, ALAN B, SHERMAN, WILLIAM R and WILL, JEFFREY D, Introduction to Virtual Reality. En: *Developing virtual reality applications*. 2009. [en línea] pp. 1-32 [Fecha de consulta 8 de agosto 2019]. DOI: 10.1016/C2009-0-20103-6. Disponible en: <https://www.sciencedirect.com/science/article/pii/B978012374943700001X>
- [7] *History Of Virtual Reality - Virtual Reality Society* [en línea][Fecha de consulta 6 de agosto 2019]. Disponible en: <https://www.vrs.org.uk/virtual-reality/history.html>
- [8] SHERMAN, WILLIAM R and CRAIG, ALAN B, Introduction to Virtual Reality. En: *Understanding Virtual Reality*, Segunda Edición. 2018 [en línea] pp. 4-58 [Fecha de consulta 6 de agosto 2019]. DOI: 10.1016/B978-0-12-800965-9.00001-5. Disponible en: <https://www.sciencedirect.com/science/article/pii/B9780128009659000015>
- [9] CRUZ-NEIRA, CAROLINA, SANDIN, DANIEL, DEFANTI, THOMAS, KENYON, ROBERT and HART, JOHN, *The CAVE: audio visual experience automatic virtual environment*. [en línea]. 1992. [Fecha de consulta 6 de agosto 2019]. Disponible en: <https://go.galegroup.com/ps/anonymous?id=GALE%7CA12353475&sid=googleScholar&v=2.1&it=r&linkaccess=abs&issn=00010782&p=AONE&sw=w>
- [10] VAN KREVELEN, RICK and POELMAN, RONALD. A Survey of Augmented Reality Technologies, Applications and Limitations. *International Journal of Virtual Reality*. [en línea] 2010. [Fecha de consulta 8 de agosto 2019] (ISSN 1081-1451). Disponible en: https://www.researchgate.net/publication/279867852_A_Survey_of_Augmented_Reality_Technologies_Applications_and_Limitations
- [11] *What is mixed reality? - Mixed Reality*. [en línea][Fecha de consulta 8 de agosto 2019] Disponible en: <https://docs.microsoft.com/en-us/windows/mixed-reality/mixed-reality>

- [12] WHITWORTH, BRIAN. *The Physical World as a Virtual Reality* [en línea]. Nueva Zelanda: Universidad de Massey, Diciembre 2007. [Fecha de consulta 8 de agosto 2019]. Disponible en: <https://arxiv.org/ftp/arxiv/papers/0801/0801.0337.pdf>
- [13] BIERBAUM, ALLEN and JUST, CHRISTOPHER. *Software Tools for Virtual Reality Application Development* [en línea]. Iowa, 1998. [Fecha de consulta 8 de agosto 2019]. Disponible en: <https://pdfs.semanticscholar.org/a650/7b8406973e035ce54fc670a77398144e8554.pdf>
- [14] BAMODU, OLULEKE and YE, XUMING. Virtual Reality and Virtual Reality System Components. En: *Advanced Materials Research* [en línea]. Atlantis Press, París, Francia, 2013. Vols. 765-767, pp. 1169-1172 [Fecha de consulta 8 de agosto 2019]. DOI: 10.4028/www.scientific.net/AMR.765-767.1169. Disponible en: <https://pdfs.semanticscholar.org/ee45/af43d1373f2564c49a0c15502fea5f67d8e3.pdf>
- [15] *Types of VR system*. Agocg.ac.uk [en línea][Fecha de consulta 8 de agosto 2019]. Disponible en: <http://www.agocg.ac.uk/reports/virtual/37/chapter2.htm>
- [16] *Products - Unity*. [en línea][Fecha de consulta 10 de agosto 2019]. Disponible en: <https://unity3d.com/es/unity>
- [17] *Programming in Unity: For programmers new to Unity - Unity*. [en línea][Fecha de consulta 10 de agosto 2019]. Disponible en: <https://unity3d.com/es/programming-in-unity>
- [18] *Unreal Engine | What is Unreal Engine 4*. [en línea][Fecha de consulta 11 de agosto 2019]. Disponible en: <https://www.unrealengine.com/en-US/>
- [19] TECHNOLOGIES, UNITY, *Flujo de trabajo de los Assets (Asset Workflow) - Unity Manual*. [en línea][Fecha de consulta 11 de agosto 2019]. Disponible en: <https://docs.unity3d.com/es/current/Manual/AssetWorkflow.html>
- [20] PARRILLA RUIZ, JAVIER. CryEngine 5, la nueva versión del motor gráfico de Crytek. En: *HobbyConsolas* [en línea]. 2016. [Fecha de consulta 11 de agosto 2019]. Disponible en: <https://www.hobbyconsolas.com/industria/cryengine-5-nueva-version-motor-grafico-crytek-1609>
- [21] *Sandbox & Tools - CryEngine*. [en línea][Fecha de consulta 11 de agosto 2019]. Disponible en: <https://www.cryengine.com/features/sandbox-tools>
- [22] *C# Programming - CRYENGINE Programming - Documentation*. [en línea][Fecha de consulta 11 de agosto 2019]. Disponible en: <https://docs.cryengine.com/display/CEPROG/C%23+Programming>
- [23] *Platforms - CryEngine*. [en línea][Fecha de consulta 11 de agosto 2019]. Disponible en: <https://www.cryengine.com/features/platforms>
- [24] *Oculus Rift S | Features, 2019*. [en línea][Fecha de consulta 11 de agosto 2019]. Disponible en: <https://www.oculus.com/rift-s/features/>
- [25] *Oculus Rift S Review: First-Gen VR Gets a Reboot, 2019*. Tom's Hardware [en línea][Fecha de consulta 11 de agosto 2019]. Disponible en: <https://www.tomshardware.com/reviews/oculus-rift-s-vr-headset,6148.html>

- [26] *VIVE™ / VIVE Virtual Reality System*. [en línea][Fecha de consulta 11 de agosto 2019]. Disponible en: <https://www.vive.com/us/product/vive-virtual-reality-system/>
- [27] *HTC Vive: Análisis, requisitos y características - MGRV. Mejores Gafas Realidad Virtual* [en línea][Fecha de consulta 11 de agosto 2019]. Disponible en: <https://mejoresgafasrealidadvirtual.com/gafas-vr/gafas-de-realidad-virtual-para-pc/htc-vive/>
- [28] *Especificaciones técnicas. Playstation* [en línea][Fecha de consulta 11 de agosto 2019]. Disponible en: <https://www.playstation.com/es-es/explore/playstation-vr/tech-specs/>
- [29] *Sony Playstation VR - Review, características técnicas, accesorios, juegos. Mejores Gafas Realidad Virtual* [en línea][Fecha de consulta 11 de agosto 2019]. Disponible en: <https://mejoresgafasrealidadvirtual.com/gafas-vr/gafas-de-realidad-virtual-para-videoconsolas/sony-playstation-vr/>
- [30] *What is the Leap Motion Controller?*, 2017. Leap Motion Support [en línea][Fecha de consulta 12 de agosto 2019]. Disponible en: <https://support.leapmotion.com/hc/en-us/articles/223783728-What-is-the-Leap-Motion-Controller->
- [31] *Leap Motion Introduces Orion, Its Next-Generation Hand Tracking Product for Developers in VR/AR* - Leap Motion, 2017. Leap Motion [en línea][Fecha de consulta 12 de agosto 2019]. Disponible en: <https://www.leapmotion.com/news/leap-motion-introduces-orion-its-next-generation-hand-tracking-product-for-developers-in-vr-ar/>
- [32] HOLZ, DAVID, *Unveiling Project North Star*. Leap Motion Blog [en línea]. 2018. [Fecha de consulta 12 de agosto 2019]. Disponible en: <http://blog.leapmotion.com/northstar/>
- [33] MOTION, LEAP, *Project North Star is Now Open Source* - Leap Motion Blog. Leap Motion Blog [en línea]. 2018. [Fecha de consulta 12 de agosto 2019]. Disponible en: <http://blog.leapmotion.com/north-star-open-source/>
- [34] VIGNEMONT, FRÉDÉRIQUE. Embodiment, ownership and disownership. En: *Consciousness and Cognition*. [en línea]. 2011, volumen 20, issue 1, pp 82-93. [Fecha de consulta 14 de agosto 2019]. DOI: 10.1016/j.concog.2010.09.004. Disponible en: <https://www.sciencedirect.com/science/article/pii/S1053810010001704?via%3Dihub>
- [35] BLANKE, OLAF and METZINGER, THOMAS. Full-body illusions and minimal phenomenal selfhood. En: *Trends in Cognitive Science*. [en línea]. 2009, volumen 13, issue 1, pp 7-13. [Fecha de consulta 15 de agosto 2019]. DOI: 10.1016/j.tics.2008.10.003. Disponible en: <https://www.sciencedirect.com/science/article/abs/pii/S1364661308002507>
- [36] TSAKIRIS, M., PRABHU, G. and HAGGARD, P. Having a body versus moving your body: How agency structures body-ownership. En: *Consciousness and Cognition*. [en línea]. 2006, volumen 15, issue 2, pp 423-432. [Fecha de consulta 15 de agosto 2019]. DOI: 10.1016/j.concog.2005.09.004. Disponible en: <https://www.sciencedirect.com/science/article/pii/S1053810005001200>

- [37] TSAKIRIS, MANOS. My body in the brain: A neurocognitive model of body-ownership. En: *Neuropsychologia*. [en línea] 2010, volumen 48, issue 3, pp 703-712. [Fecha de consulta 15 de agosto 2019]. DOI: 10.1016/j.neuropsychologia.2009.09.034. Disponible en: <https://www.sciencedirect.com/science/article/abs/pii/S002839320900390X>
- [38] BOTVINICK, MATTHEW and COHEN, JONATHAN. Rubber hands 'feel' touch that eyes see. En: *Nature* [en línea] 1998, 391, pp. 756 [Fecha de consulta 19 de agosto 2019]. DOI: 10.1038/35784. Disponible en: <http://www.nyu.edu/gsas/dept/philo/courses/consciousness/papers/Botvinick.pdf>
- [39] VILLÉN, JOSE A., MARTÍN, ANA C., PÉREZ-DÍAZ, FRANCISCO J. and LÓPEZ, JUAN C. *La ilusión de la mano de goma: factores implicados, bases neurales y aplicaciones clínicas*. [en línea] 2015 [Fecha de consulta 3 de septiembre 2019]. DOI: 10.4067/S0717-92272015000400008. Disponible en: https://www.researchgate.net/publication/292671642_La_ilusion_de_la_mano_de_goma_factores_implicados_bases_neurales_y_aplicaciones_clinicas
- [40] LLOYD, DM., SHORE, DI., SPENCE, C. and CALVERT GA. Multisensory representation of limb position in human premotor cortex. En: *Nature Neuroscience* [en línea] 2003, volumen 6, pp. 17-18 [Fecha de consulta 3 de septiembre 2019]. DOI: 10.1038/nn991. Disponible en: <http://www.p.u-tokyo.ac.jp/~kyama/kitazawaz/Lloyd2003.pdf>
- [41] BLANKE, O., SLATER, M., and SERINO, A. Behavioral, Neural, and Computational Principles of Bodily Self-Consciousness. En: *Neuron*. [en línea]. 2015, volumen 88, issue 1, pp. 145-166 [Fecha de consulta 21 de agosto 2019]. DOI: 10.1016/j.neuron.2015.09.029. Disponible en: [https://www.cell.com/neuron/fulltext/S0896-6273\(15\)00818-1?returnURL=https%3A%2F%2Flinkinghub.elsevier.com%2Fretrieve%2Fpii%2FS0896627315008181%3Fshowall%3Dtrue](https://www.cell.com/neuron/fulltext/S0896-6273(15)00818-1?returnURL=https%3A%2F%2Flinkinghub.elsevier.com%2Fretrieve%2Fpii%2FS0896627315008181%3Fshowall%3Dtrue)
- [42] *Know your brain: Primary somatosensory cortex* — Neuroscientifically Challenged, 2016. Neuroscientifically Challenged [en línea][Fecha de consulta 23 de agosto 2019]. Disponible en: <https://www.neuroscientificallychallenged.com/blog/know-your-brain-primary-somatosensory-cortex>

Apéndice I

Manual de uso del programa

1 Introducción

El programa desarrollado para la recreación del experimento clásico de la ilusión de la mano de goma (RHI) ha sido desarrollado con el motor de videojuegos Unity (v. 2017.4.6f1) ya que permite integrar de manera cómoda y sencilla las herramientas de realidad virtual requeridas para este experimento.

En primer lugar, se necesita de Oculus Rift, ya que es la herramienta que nos permite envolver al usuario en un entorno de realidad virtual. Y, en segundo lugar, se necesita de Leap Motion, el cual es un sistema de integración que nos permite virtualizar las manos del usuario dentro del ordenador, mientras estas estén dentro del rango de sus sensores.

1.1 Librerías externas

Para integrar estas herramientas dentro de Unity se ha necesitado una librería externa que provee el propio desarrollador de Leap Motion y modificar la configuración del jugar dentro de Unity para habilitar la SDK de Oculus.

Para Leap Motion tendremos que descargar la librería externa desde su página web bajo el nombre de Unity Core Assets (v. 4.4.0). Adicionalmente, de Leap Motion necesitaremos la librería Leap Motion Interaction Engine (v. 1.2.0), que también podremos encontrar en su página web, y que utilizaremos para poder hacer que las manos interactúen con otros objetos de la escena.

Para hacer que Unity reconozca que queremos usar vamos a cambiar la configuración del jugador desde *Edit > Project Setting > Player > XR Settings* (en el inspector que se nos ha abierto) *> Virtual Reality Supported > Oculus*, tendremos que marcar Virtual Reality Supported como activo y cuando se nos despliegue que SDK de realidad virtual queremos usar, dejar el de Oculus como el primero, por encima de OpenVR para que sea el primero que busque al arrancar. Finalmente, como también hay que integrar el controlador del guante que produce la estimulación en la mano del sujeto, se requiere de una librería más para hacer que Unity pueda controlar los puertos de serie (COM) de manera sencilla. Esta librería es Ardity, y la podremos encontrar en su página web o en su GitHub ya que ha sido creada por usuarios y no directamente por la empresa desarrolladora de Unity, aunque se distribuye bajo licencia Creative Commons Attributions para que se pueda usar libremente.

1.2 Desarrollo básico

Una vez importadas las distintas librerías podemos pasar al desarrollo de las diferentes escenas que compondrán el programa. En este caso, ajustándonos a los requisitos especificados por el equipo de investigación, ha sido necesario desarrollar dos escenas. Una primera en la que el usuario se adapta al uso de las manos virtuales mediante la realización de un pequeño juego que requiere del movimiento de estas. Y, una segunda en la cual se pasa a la fase experimental en la que el sujeto simplemente mantiene la mano izquierda extendida y ve como un lápiz

vibrador o de calor va estimulando diferentes zonas de la mano.

Ha sido también necesario incluir un canvas a modo de menú de pausa, para conectar ambas escenas, este menú no será visible a través del visor de Oculus Rift en ningún momento, aunque los objetos de la escena, a excepción de las manos, se congelaran ya que el tiempo de esta estará pausado. Por lo tanto, el menú solo lo verá el investigador en la pantalla del ordenador donde se esté ejecutando el programa y se desplegará pulsando la tecla de escape (ESC). En este menú se muestran tres opciones, una primera para continuar (acción que también podemos realizar volviendo a pulsar la tecla ESC), una segunda para cambiar a la otra escena a la que estemos y, finalmente, una última para cerrar el programa. Este canvas, por tanto, será desplegable desde ambas escenas.



Ilustración 1. Canvas para conectar ambas escenas

1.3 Correcto funcionamiento del programa

Para que el programa se ejecute correctamente hace falta cumplir con los siguientes requisitos. El más esencial es tener instalados los programas que controlan las herramientas ya mencionadas, es decir, Oculus Setup para que las Oculus Rift funcionen, y Orion Beta para que Leap Motion funcione correctamente, además, estos programas deberán estar ejecutándose en segundo plano al mismo tiempo que el programa desarrollado para la recreación de la RHI.

También, será necesario que el controlador del guante se conecte al COM 3 y el convertidor para el polígrafo se conecte al COM 4, ya que es la configuración establecida en el programa, y en caso de no estar en estos puertos, no se podría llegar a conectar con ellos. Si no hubiera manera de conectar a estos puertos porque los dispositivos siempre se pusieran en otro por algún motivo, habría que abrir el programa en Unity y en la escena llamada *Simulación*, en la jerarquía habría que buscar el objeto llamado *SerialController* y en su inspector cambiar el nombre del puerto al que se esté usando para el controlador del guante. Para el convertidor del

polígrafo habría que hacer lo mismo, pero con el objeto llamado *MarkController*.

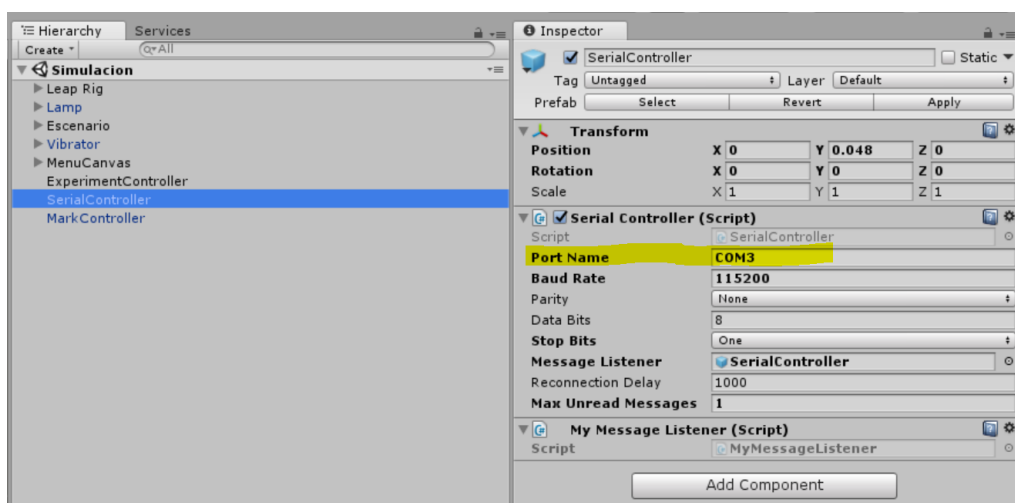


Ilustración 2. Cambiar nombre del puerto para el controlador del guante

Adicionalmente, puede ser también necesario modificar la altura de la cámara de la escena a través de la cual el usuario ve el entorno, ya que las personas tienen diferentes alturas, pero queremos que las manos siempre queden por encima de la mesa, aunque a una altura en la que se puedan “apoyar” sobre ella (realmente se simula que se apoyan ya que las manos atravesarían el objeto en caso de colisionar contra él). Para modificar esta altura, lo que haremos será buscar en la jerarquía de la escena un objeto llamado Leap Rig, y al seleccionarlo se nos abrirá en el inspector sus propiedades, por lo que desde su componente *Transform>Position* podremos modificar el valor del parámetro Y para subir o bajar dicho objeto. Otra opción es al seleccionarlo, desde la ventana de escena, arrastrar la flecha verde, que aparece en el objeto, hacia arriba o hacia abajo para ajustarlo, aunque este segundo método es menos preciso que el anterior.

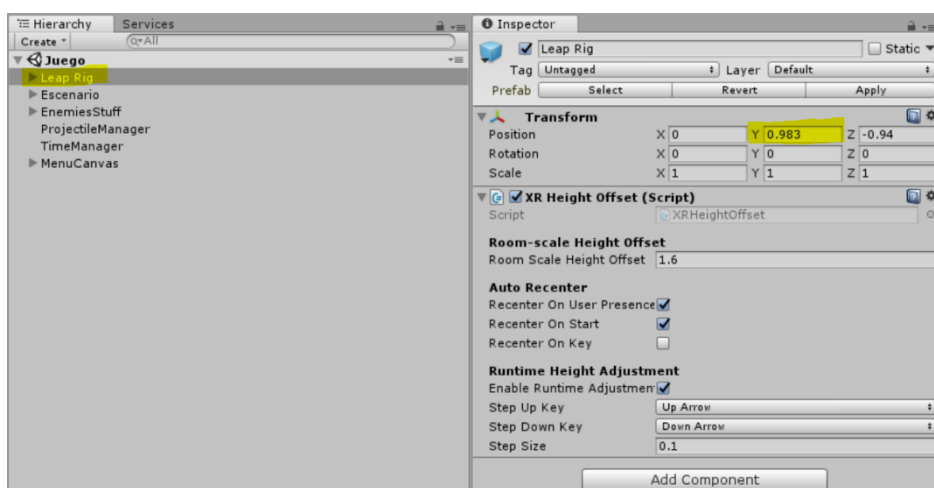


Ilustración 3. Modificar altura de la cámara

Una vez modificados los puertos y/o la altura se tendría que volver a compilar el programa desde *File > Build Setting...* asegurarnos de que ambas escenas estén marcadas con el juego como 0 y la simulación como 1, y que la plataforma elegida es Windows con arquitectura x86_64. Finalmente, darle a *build* para que compile el programa y nos cree un ejecutable .exe dentro de la carpeta Builds que será la que seleccionaremos cuando nos pregunte donde queremos crearlo.

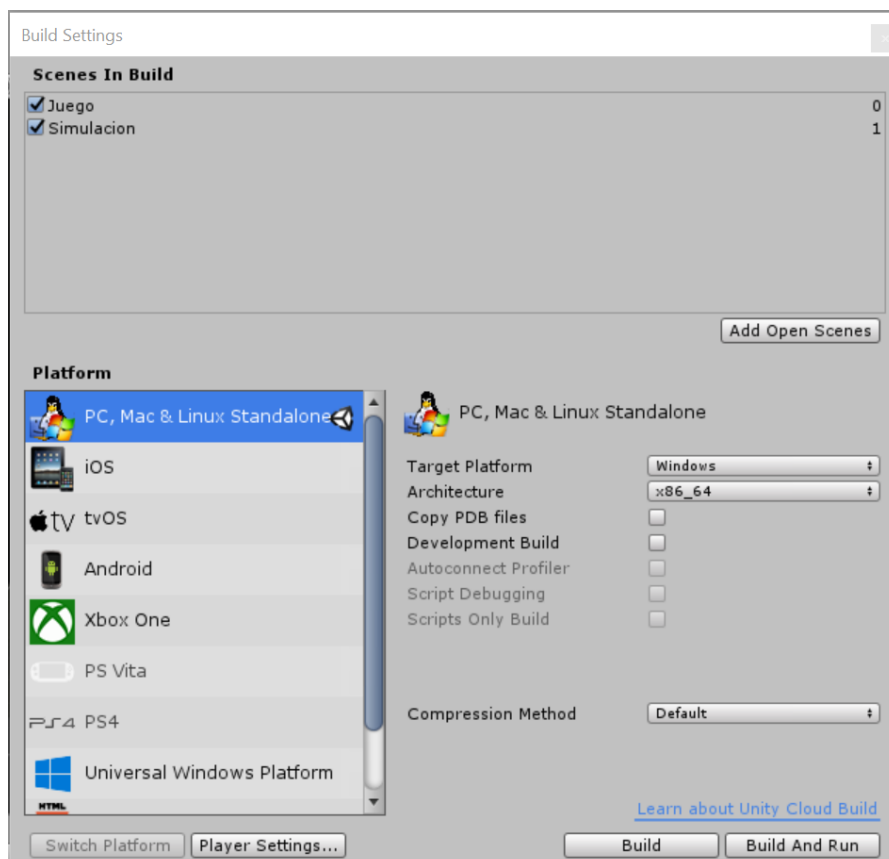


Ilustración 4. Compilar programa

En las siguientes secciones vamos a ver el funcionamiento de cada una de las escenas explicadas con detalle.

2 Juego

Esta es la primera escena del programa, y la cual arrancará el mismo una vez se inicie. En ella lo que pretendemos es que el usuario se acostumbre a las manos virtuales para que luego la simulación de la RHI tenga mayor efectividad. Para alcanzar este objetivo de adaptación al medio, se ha diseñado un pequeño minijuego inspirado en el clásico juego de golpear a los topos cuando estos aparecen, salvo que en lugar de topos son unos pequeños monstruos importados de otra librería externa (no mencionada anteriormente ya que se podría usar cualquier tipo de figura) que podemos encontrar en Unity Asset Store con el nombre de Level 1 Monster Pack.

Cuando el usuario aparece inicialmente en la escena, el entorno con el que se encuentra es una habitación oscura con una pantalla blanca de bordes negros en la pared que tiene enfrente, un pequeño cuadrado de luz roja (o verde, dependiendo de las acciones que haya hecho el investigador a cargo de ejecutar el programa) encima de esta pantalla (a partir de ahora LED Estado), una mesa negra justo delante suya, a la altura de los brazos como si estuviera sentado en una silla (como realmente está el sujeto en la habitación) y, un botón rojo con bordes

grises flotando a su derecha.

A partir de este momento, el usuario podrá mover las manos libremente por delante del sensor de Leap Motion para que estas se virtualicen dentro de la escena y pueda interactuar con ella. Para que las manos y brazos se vean de la forma más realista posible se ha utilizado una librería externa (tampoco mencionada anteriormente ya que no es necesaria para el funcionamiento del programa porque se podrían usar otros modelos del propio Leap Motion) que se podemos encontrar también en Unity Asset Store bajo el nombre de Leap Motion Realistic Hands (aunque esta no es gratuita como las demás).

2.1 Acciones realizables por el usuario

Las acciones que puede realizar el usuario son:

- Hacer el gesto del pulgar hacia arriba con la mano **derecha** para que las instrucciones que tiene en la pantalla delante suya avancen a la siguiente diapositiva.
- Hacer el gesto del pulgar hacia arriba con la mano **izquierda** para que las diapositivas de las instrucciones retrocedan.
- Extender la mano **izquierda** con la palma hacia arriba para que se genere un botón al lado de su palma. Este botón es el que le servirá para crear los cubos que se usarán a modo de proyectiles para eliminar a los monstruos que aparezcan.
- Coger los cubos con las manos y lanzarlos.
- Pulsar el botón rojo flotante que tiene a su derecha, el cual funcionará o no dependiendo de si el LED Estado está verde o rojo respectivamente. En caso de funcionar se iniciará el juego y caso de que no, simplemente no hará nada.

2.2 Acciones realizables por el investigador

Las acciones que puede realizar el investigador con el teclado:

- Pulsar la tecla ESC para mostrar el menú como ya se ha dicho anteriormente.
- Pulsar la barra espaciadora para crear enemigos de ejemplo en el punto que hay determinado para que aparezcan el código.
- Introducir un número por teclado para decir cuánto tiempo quiere que dure el minijuego. Una vez introducido el número, se pulsará la tecla Entrar (Intro o Enter) para que el programa entienda que la cadena introducida es el tiempo deseado. Una vez el tiempo se ha establecido, el LED Estado pasará a ser de color verde. También se puede pulsar la tecla de retroceso para borrar los caracteres introducidos en la cadena, siempre y cuando se haga antes de pulsar Enter. Cualquier otro carácter introducido diferente de un número o las teclas mencionadas, será obviado como si no hubiera sido pulsado.

2.3 Funcionamiento del juego

En lo referente al funcionamiento del minijuego, el usuario debe pulsar el botón rojo a su derecha para que este empiece, y en ese momento este botón desaparecerá, no sin antes haber cambiado su color a azul para denotar que ha sido pulsado correctamente.

El primer monstruo aparecerá un segundo después de ser destruido el botón, y, a partir de aquí, hasta que se acabe el tiempo introducido por el investigador, los monstruos irán

apareciendo en distintas posiciones aleatoriamente (las posiciones han sido definidas con anterioridad en el código) con un intervalo de entre tres y cinco segundos desde que aparece uno hasta que aparece el siguiente.

Siempre, antes de que aparezca un monstruo, una luz roja parpadea en el punto en el que va a parecer durante un segundo, para avisar al usuario, pero cuidado, ya que los monstruos también pueden aparecer en las paredes laterales y es importante estar atento.

El tiempo de vida de los monstruos será de entre seis y nueve segundos o, hasta que un cubo rojo los golpee (en las instrucciones que se le pasan al usuario por pantalla se le explica que los cubos rojos están activos y los verdes inactivos, ya que significa que ya han chocado al menos una vez contra el suelo, y en tal caso ya no sirven para eliminar a los monstruos), en este segundo caso el usuario verá que el marcador de puntos que se le muestra ahora por pantalla aumenta en diez puntos.

Todos los enemigos dan diez puntos independientemente de la forma o color, cosa que también se determina de manera aleatoria entre los diferentes modelos que hay. A su vez, los monstruos cuando aparecen también producen un sonido aleatorio que no está relacionado de ninguna manera con la posición en la que aparezca o el modelo del monstruo.

Finalmente, una vez terminado el tiempo de juego, se mostrará en pantalla el resultado de cuantos monstruos han sido eliminados, con su correspondiente puntuación. Y, además, el LED Estado volverá a ponerse en rojo para que se pueda introducir otro tiempo distinto y el botón que había sido destruido reaparecer para que el juego pueda volver a iniciarse. Sin embargo, las instrucciones que se mostraban al principio en la pantalla ya no volverán a aparecer ni, aunque el usuario haga los gestos de los pulgares.

Mencionar también, que los tiempos establecidos para la aparición de los monstruos y el tiempo de vida de los mismos, aunque dentro de un rango aleatorio, los límites de este rango han sido probados para que la sensación del juego no sea agobiante para el usuario por falta de tiempo, ni sea extremadamente sencillo de tal manera que el usuario se aburra mientras juegue porque le sobra tiempo en exceso para realizar las acciones.

2.4 Flujo de funcionamiento

En esta subsección se explica lo que se considera un flujo de funcionamiento normal de la escena:

Inicialmente se deja que el usuario lea las instrucciones y las vaya pasando para entender cómo funciona el programa. En la esquina inferior derecha de cada diapositiva sale una flecha apuntando a la derecha en caso de haber más transparencias después de en la que se está. Una de las transparencias autogenera un enemigo de prueba para que el usuario trate de destruirlo sin restricción de tiempo alguna. Además, el usuario puede volver a esta transparencia tantas veces como quiera, lo que volverá a generar al monstruo en el mismo punto.

Una vez el usuario ha terminado este pequeño tutorial para entender el funcionamiento del juego, es cuando el investigador debe introducir el tiempo que desea que dure el juego por teclado y pulsar Enter para cambiar el LED Estado a verde y que el juego pueda comenzar. Realmente, es irrelevante el momento en el que el investigador introduzca el tiempo, pero podría darse la situación de que el usuario le diese al botón rojo antes de terminar el tutorial y entonces al estar el LED Estado en verde, el juego comenzaría interrumpiendo el tutorial sin

importar porque punto se encuentre este. Es por esto, que recomiendo que el tiempo se introduzca solo una vez finalizado el tutorial, ya que, al comenzar el juego, todos los cubos y monstruos que pudiera haber en la escena desaparecerán para que empiece limpia, y las instrucciones de pantalla se cambiarán por la puntuación actual que lleve el usuario.

Para finalizar, una vez, el juego se haya terminado, ya queda a libre disposición del investigador si quiere reiniciarlo con otra duración o si desea cambiar a la siguiente escena usando el menú que aparece al pulsar la tecla de escape.

3 Simulación

Esta escena es la principal del programa, ya que en ella es en la que se va a realizar toda la experimentación para la RHI, pero para llegar hasta ella tendremos que usar el menú de pausa desde la escena del juego como ya se ha explicado anteriormente.

El entorno en este caso es igual que el anterior a excepción de que no hay ningún tipo de botón, es decir, solo está la habitación con la pantalla, el LED Estado encima de esta y la mesa justo enfrente de donde aparece el usuario.

3.1 Acciones realizables por el usuario

En esta escena la única acción que puede realizar el usuario es poner la palma de la mano izquierda hacia arriba con los dedos extendidos, para que delante de estos se genere el lápiz vibrador que va a causar la estimulación simulada. Si los dedos del usuario se cierran o la palma deja de apuntar hacia arriba, el lápiz desaparecerá (se destruye el objeto), y volverá a aparecer (se crea un nuevo objeto) solo cuando ambas condiciones se vuelvan a cumplir.

Esta regla tiene una excepción, y es que una vez la secuencia de instrucciones de la simulación (se explica más adelante de que es) se haya comenzado, ya no importa que el usuario cierre o voltee la mano, que el lápiz vibrador seguirá funcionando, aunque, obviamente, el efecto visual será incongruente a los estímulos que recibe el usuario.

Es necesario hacer que el lápiz no desaparezca una vez iniciada la simulación, porque se está accediendo al objeto constantemente durante esta, y debido a la sensibilidad de Leap Motion, a veces, pequeños movimientos hacen que las condiciones para que el lápiz esté en pantalla dejen de cumplirse durante milisegundos, lo que causa la destrucción y creación de uno nuevo, y esto causaría un error en ejecución que detendría el programa.

Finalmente, a modo de recomendación personal, es aconsejable que el usuario abra y cierre la mano varias veces con la palma apuntando hacia arriba antes de empezar la simulación para que así el lápiz vibrador se posicione correctamente, ya que en ocasiones la primera vez que el lápiz se crea aparece bastante desviado de la posición de para que debería tener, que es alineado con el dedo corazón más o menos.

3.2 Acciones realizables por el investigador

En esta escena el investigador puede y debe realizar unas cuantas acciones más, ya que es el núcleo del experimento. Entre las acciones realizables nos encontramos:

- Pulsar la tecla ESC para mostrar el menú de pausa.

- Pulsar la tecla numérica 1 para hacer que el modelo de la mano **derecha** se haga más grande.
- Pulsar la tecla numérica 2 para hacer que el modelo de la mano **derecha** se haga más pequeño.
- Pulsar la tecla numérica 3 para hacer que el modelo de la mano **izquierda** se haga más grande.
- Pulsar la tecla numérica 4 para hacer que el modelo de la mano **izquierda** se haga más pequeño.
- Pulsar la tecla numérica 5 para que el modelo de la mano **izquierda** se voltee y se vea al revés de como el usuario tiene colocada la mano. Es decir, si el usuario tiene la mano con la palma hacia arriba, el modelo se verá con la palma hacia abajo y viceversa. Volviendo a pulsar la tecla, haremos que este efecto pare y la correlación entre la mano del usuario y el modelo vuelva a ser normal.
- Pulsar la barra espaciadora para que el programa lea las instrucciones del fichero de órdenes que tendrá que seguir la simulación y las almacene en sus estructuras internas. Esto cambiará el color del LED Estado de rojo a verde indicando que la simulación está lista para empezar. En la siguiente sección se explicará cómo funciona el fichero de órdenes que aquí se menciona.
- Pulsar la tecla Enter para que las instrucciones que conforman la simulación comiencen a ejecutarse. Esto solo pasará si las instrucciones están cargadas dentro del programa, es decir, si el LED Estado está verde debido a que antes se ha pulsado la barra espaciadora.

3.3 Fichero de instrucciones para la simulación

El fichero de órdenes ya mencionado anteriormente en este documento, es un fichero .txt normal en el que se especificarán las instrucciones que seguirá la simulación, pero con una estructura especial diseñada para que el programa entienda dichas instrucciones.

Es importante que este fichero de ordenes esté nombrado como *Ordenes* con extensión .txt y que se encuentre en una carpeta llamada VRExperiment que esté situada en el escritorio del ordenador, ya que es donde el programa lo buscará al pulsar la barra espaciadora para leer las órdenes.

Inicialmente, en la cabecera del fichero se añadirán las direcciones absolutas de las imágenes que se quieren ir mostrando por pantalla, las cuales tienen que estar en formato .png, y las “\” de Windows en las direcciones hay que cambiarlas a “/” para que así el programa pueda leerlas, convertirlas en sprites y almacenarlas para su posterior uso.

Se recomienda que las imágenes se guarden en una subcarpeta dentro de VRExperiment para que sea más fácil localizarlas y, además, que la primera imagen que se añada al fichero sea la imagen que se va a usar como fondo de pantalla cuando no se quiere mostrar nada por esta, ya que si no, al principio, la pantalla saldrá con fondo blanco, pero una vez se muestre una imagen ya no se podrá volver a ese fondo por defecto, así que hay que añadirle manualmente un fondo por defecto al que se pueda volver.

Una vez añadidas las imágenes al fichero, podemos pasar a añadir las instrucciones en la siguiente línea, sin dejar ningún tipo de separación más allá del salto de línea para empezar una nueva. Solo se puede poner una instrucción por línea (al igual que las imágenes), y estas nos permitirán definir seis tipos diferentes de acciones, las cuales son: poner el lápiz en posición de

parada (delante de la mano alineado con el dedo corazón o casi), dar vibración en una posición determinada, dar calor en una posición determinada, mostrar una imagen (.png) por pantalla, cambiar la velocidad de los vibradores, mandar una marca al convertidor del polígrafo. La estructura diseñada para las instrucciones es la siguiente:

Acción;Estado;Valor;(Temperatura);Tiempo

Como vemos hay que definir cuatro parámetros obligatoriamente para todas las acciones y, además, para la acción de dar calor en algún dedo, hay que especificar la temperatura que queremos que alcance el emisor de calor del guante. Es importante que este parámetro solo se especifique en las instrucciones de dar calor, ya que ponerlo en cualquier otra o no ponerlo en estas, daría lugar a un fallo en la lectura del fichero.

A continuación, vamos a ver cómo definir los diferentes parámetros de la estructura de las instrucciones:

- **Acción:** este primer parámetro es el que nos va a definir qué tipo de instrucción de las seis posibles queremos realizar. Para rellenarlo usaremos solamente una letra en mayúsculas dependiendo de la acción deseada:
 - S -> poner el lápiz en posición de parada.
 - V -> dar vibración en una posición determinada.
 - C -> dar calor en una posición determinada.
 - P -> mostrar una imagen en la pantalla.
 - A -> determinar la velocidad de los vibradores
 - M -> mandar una marca al polígrafo.
- **Estado:** este parámetro solo es importante para las acciones de dar vibración o calor, ya que define si queremos que la acción se haga completa, es decir, tanto en la realidad virtual como el guante, o parcial, en uno de los dos lados, para ello, usaremos los siguientes valores:
 - 0 -> para que se haga en la realidad virtual y se transmita al guante.
 - 1 -> para que solo se haga en la realidad virtual.
 - 2 -> para que solo se mande la acción al guante, pero en el mundo virtual no pase nada. De hecho, el lápiz deja de verse en el mundo virtual.

Importante: para el resto de acciones este parámetro es como si no existiera, así que se recomienda dejarlo a 0 para evitar confusión.
- **Valor:** este es el parámetro más importante junto con el de acción, ya que para cada tipo de esta nos va a indicar una cosa:
 - S -> en esta acción nos sirve para saber qué tipo de acción se va a realizar a continuación, ya que en este estado es en el que se modifica el lápiz para que dé vibración o calor. De esta manera tenemos que este parámetro puede rellenarse con los siguientes valores:
 - 0 -> si la siguiente acción no requiere del uso del lápiz.
 - 1 -> si la siguiente acción es una acción de vibración.
 - 2 -> si la siguiente acción es una acción de calor.
 - V -> en este caso indica el número de falange donde se quiere posicionar el lápiz para que de la vibración. En la *Ilustración 5* podemos ver el valor de las distintas posiciones.

- C -> para esta acción se usa de la misma manera que para la de dar vibración.
- P -> en esta acción va a indicar la imagen que queremos que se muestre por pantalla. Las imágenes han sido guardadas en un vector, en el orden en el que se han introducido en el fichero de texto, de esta manera si queremos que se muestre por pantalla la primera imagen que hemos puesto en el fichero *Ordenes* (que debería ser la imagen que queremos poner por defecto en la pantalla), tendríamos que pasarle el valor 0, si queremos que se muestre la segunda el valor sería 1, y así sucesivamente.
- A -> para esta acción lo que nos va a indicar es la velocidad a la que queremos que funcionen los vibradores, en este caso, este parámetro viene delimitado por las capacidades del guante, de tal manera que este solo puede tomar valores entre 40 y 80 sin decimales.
- M -> el valor es la marca que se le va a mandar al convertidor del polígrafo, por lo tanto, tiene que ser un entero entre 0 y 255.
- **Temperatura:** como ya se ha mencionado anteriormente, este parámetro solo lo usaremos cuando la acción sea de tipo C, y sus valores pueden oscilar entre 0 y 70 sin decimales, aunque tal y como está diseñado el emisor de calor, este se apaga una vez se alcanza la temperatura deseada, y como está pegado a la mano, solo emitirá calor si se le manda una temperatura superior a la de la mano. **Importante:** no poner los paréntesis que salen arriba en la definición, ya que solo están puestos a modo aclaratorio para indicar que el parámetro no siempre va a ser requerido.
- **Tiempo:** finalmente, este último parámetro nos indicará el tiempo que queremos que se esté realizando la acción. Por lo tanto, le daremos un valor distinto de 0, en las acciones:
 - S -> ya que es el tiempo que queremos que no haya estimulación más que hay que considerar que la transición entre lápiz vibrador a de calor y viceversa tarda un segundo en realizarse.
 - V -> es el tiempo que se va a estar estimulando la zona.
 - C -> igual que para V.
 - M -> si se quiere prolongar el valor de la marca en el tiempo, aunque esto no nos dejará realizar otra acción hasta que el tiempo indicada haya transcurrido.

Para las acciones P y A se usará el valor 0 ya que es algo que sucede de manera instantánea y no queremos que el programa se detenga en una acción que no requiere hacer nada más.

Importante: después de cada acción V, C y M, la cuales requieren o tiempo, se manda automáticamente una acción opuesta a la descrita para ahorrar líneas en el fichero *Ordenes* y que sea más cómodo para el investigar. Es decir, si se manda una vibración, una vez transcurrido el tiempo se manda la señal de que se pare, si se manda una temperatura, transcurrido el tiempo se manda una temperatura 0 al mismo sitio para que se deje de emitir calor, y si se manda una marca, acto seguido se manda una marca 0 que es la que para la señal.

Una vez explicado como rellenar todos los parámetros para crear instrucciones que se ajusten a las necesidades de investigador, queda por mencionar que, la acción de parada es una instrucción que siempre vamos a necesitar poner de manera obligatoria antes de una instrucción de vibración o calor, para que así el lápiz se adapte a la acción que se quiere realizar a

continuación. Y que, además, para evitar errores de ejecución, deberíamos ponerla también detrás de estas instrucciones, para que así el lápiz volviese a su estado de parada, aunque no se modifique su estado. Por lo tanto, la mejor manera de evitar errores es ir intercalando instrucciones de parada.

Finalmente, en la *Ilustración 6* se muestra un pequeño ejemplo de cómo sería un fichero *Ordenes*, en su carpeta correspondiente.

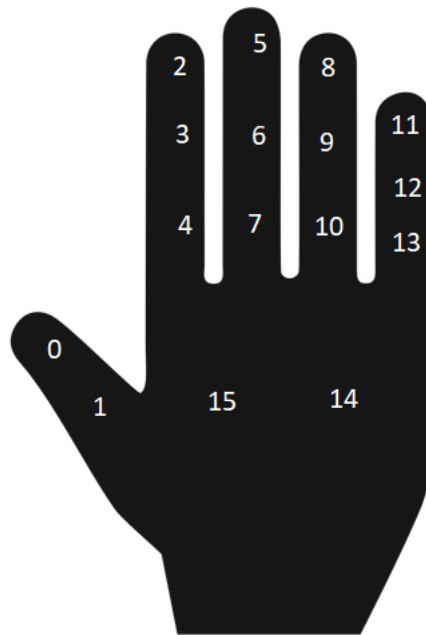


Ilustración 6. Posiciones de la mano

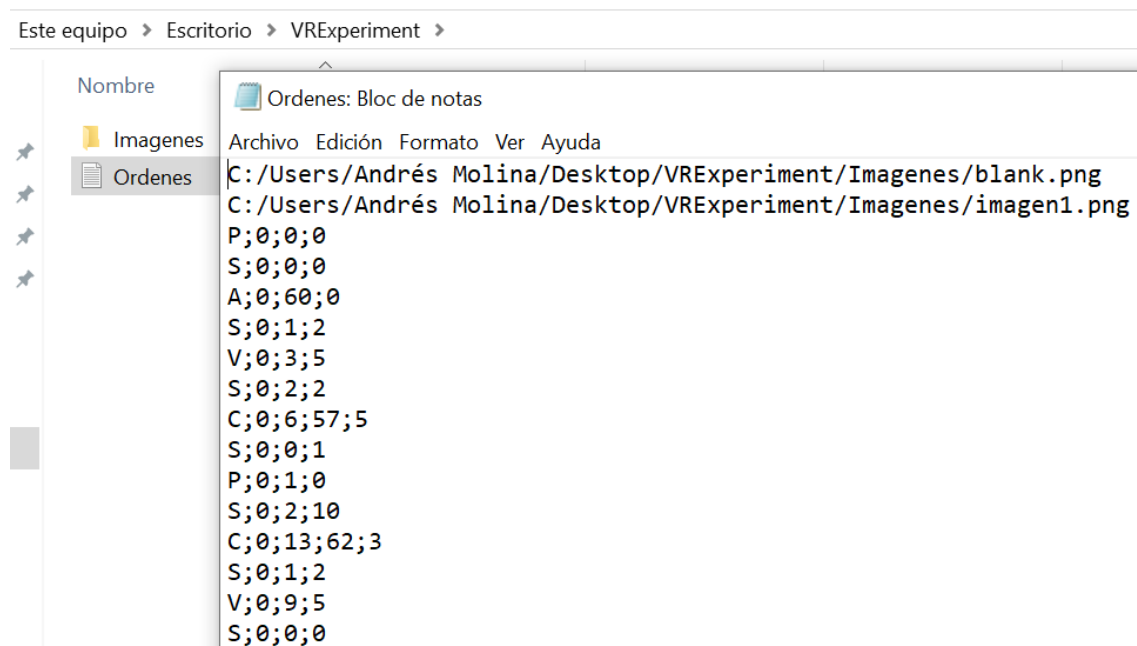


Ilustración 5. Ejemplo de fichero Ordenes en su carpeta correspondiente

3.4 Flujo de funcionamiento

En esta escena el flujo de funcionamiento es bastante más sencillo que en la anterior, ya que, como hemos visto, la complejidad está en la redacción del fichero de órdenes.

Esta vez, cuando la escena se haya iniciado, el usuario podrá pasar las manos por delante del lector de Leap Motion, mientras que el investigador pulsa la barra espaciadora para que se carguen las órdenes del fichero. Una vez el LED Estado esté en verde, y el investigador haya decidido el tamaño y correspondencia de las manos con los reales, este puede pasar a pulsar la tecla Enter para que comience la ejecución de las mismas, siempre asegurándose de que el usuario tiene la mano izquierda extendida y con la palma hacia arriba, para que el lápiz esté creado.

Una vez más, recordar que es recomendable que el usuario abra y cierre la mano izquierda varias veces para que el lápiz vibrado pueda posicionarse bien en la escena, es decir, alineado con el dedo corazón más o menos.

Finalmente, una vez la ejecución haya terminado, las ordenes permanecerán cargadas en el programa (el LED Estado queda en verde), para que se puedan volver a ejecutar, simplemente volviendo a pulsar la tecla Enter.

Antes de lanzar otra ejecución, el investigador puede volver a modificar la correspondencia de las manos virtuales con las reales, pero no podrá modificar el fichero de *Ordenes* ya que una vez las instrucciones ya están cargadas dentro del programa, no se pueden volver a cargar, así que tendría que cerrar el programa, modificar el fichero y luego volver a ejecutar el programa, en caso de querer modificar las órdenes.

Apéndice II

Estudio piloto

En este apéndice se va a mostrar el procedimiento seguido para realizar el pasamiento de la fase de entrenamiento en los participantes del grupo B (realidad virtual), así como el resultado promedio obtenido comparado con los participantes del grupo A (tradicional).

Los datos que se han tomado de los participantes de ambos grupos han sido la edad y el sexo únicamente (Tabla 31).

Tabla 31: Distribución participantes

Procedimiento	Sexo		Edad
	Hombres	Mujeres	
Virtual	4	2	26,33
Clásico	3	3	20,5

A cada participante del grupo B, se le ha explicado en que consiste la fase de entrenamiento antes de empezar, y se le han dado una serie de indicaciones sobre como interaccionar con la escena, y como debía mover las manos para que estas fuesen detectadas correctamente. Las indicaciones han sido dadas por el experimentador de manera verbal, además de estar acompañadas de gestos por su parte, para explicar de forma clara y precisa como realizar cada una de las interacciones. Así pues, las indicaciones han sido las siguientes:

“El juego consiste en crear cubos, cogerlos, y lanzarlos contra los objetivos que irán apareciendo. Una vez dentro de la realidad virtual, verá una pantalla blanca delante suya con una serie de instrucciones. Mientras estas instrucciones tengan una flecha negra apuntando a la derecha en la parte inferior, significa que hay más instrucciones a continuación. Para pasarlas, debe hacer el gesto del pulgar hacia arriba con la mano derecha. Si en algún momento quiere retroceder en las instrucciones, solamente deberá hacer el mismo gesto, pero con la mano izquierda. Estas instrucciones explican todo lo que necesita saber para jugar. Para agarrar los cubos, tiene que aproximar la mano a ellos y cuando quiera cogerlo tendrá que cerrar la mano sobre estos a modo de pinza juntando las yemas de los dedos. Por último, tenga en cuenta que el dispositivo que virtualiza sus manos, está situado delante del casco de realidad virtual, por lo que está a la altura de sus ojos. Esto implica que para que las manos se virtualicen, deben de estar dentro del campo de visión que hay justo delante de su cabeza. Este dispositivo detecta los movimientos de las manos siempre cuando se hagan suavemente, es decir, lanzamientos bruscos o movimientos bruscos podrían producir un fallo en la detección, causando que las manos dejasen de verse, o no se produzca el efecto deseado.”

Una vez, se dadas las indicaciones, se ha procedido a colocarle las Oculus Rift al participante, y a cargar el programa en su fase de entrenamiento. Dentro del programa, se le han dado un par de minutos al participante para que lea las instrucciones que se muestran inicialmente como se le había indicado y, acto seguido, se han introducido 4 minutos de duración para el tiempo del juego y se le ha notificado al participante que podía comenzar a jugar. Terminados los 4 minutos de juego, se le ha pedido al participante que se retirase las gafas de realidad virtual, y se le ha pedido que responda en una escala pictográfica de 9 valores (Ilustración 7) cómo de suya había sentido la mano virtual. Asimismo, se le ha explicado que la

escala está dividida en 9 posiciones que van de máximo a mínimo, y como funciona:

“En un extremo de la escala sientes la mano virtual como tuya, real, como si perteneciera a tu cuerpo. El otro extremo de la escala es para cuando sientas la mano virtual, completamente independiente, que no pertenece a tu cuerpo, completamente ajena a ti. Además, puedes representar niveles intermedios de pertenencia de la mano virtual marcando el número de cualquiera de las otras figuras o entre ellas. Esto te permite una evaluación más fina de cómo sientes la mano virtual como propia. Observa que la figura central indica que sientes más tuya la mano virtual que en cualquiera de las posiciones anteriores, pero menos que en cualquiera de las posiciones siguientes.”

¿Cómo sientes de tuya la mano artificial?

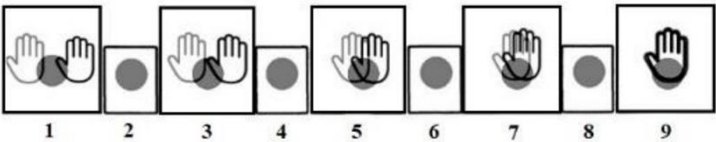


Ilustración 7: Escala pictográfica

Finalmente, una vez obtenidos los resultados de la escala por parte de todos los participantes, grupo A y grupo B, se ha calculado la media aritmética de los resultados de cada grupo, así como su desviación típica, para poder compararlos (Tabla 32).

Tabla 32: Media aritmética y desviación típica de cada grupo

Escala RHI		
Procedimiento	Media	Desv. Típica
Virtual	7,2	0,89
Clásico	5,8	3,82