Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

Entorno de Realidad Aumentada para Rehabilitación de Miembro Superior

Autor:

Matilla Plaza, David

Tutor:

De La Fuente López, Eusebio

Ingeniería de Sistemas y Automática

Valladolid, Julio 2023.

RESUMEN

El presente Trabajo Fin De Grado se basa en el desarrollo de dos videojuegos en el motor de desarrollo multiplataforma Unity, con una programación en el lenguaje C# y Python. El objetivo de estos videojuegos es la implementación en terapias de rehabilitación para pacientes con disfuncionalidades en el miembro superior. El programa segmentará y monitorizará la mano del paciente, con la cual interactuará con las distintas funcionalidades de los videojuegos, esto hace que las terapias de rehabilitación sean más amenas y amigables al paciente. Este sistema permite una reducción en los gastos típicos de estos sistemas utilizando tan solo una pantalla y una cámara web tradicional.

Palabras Clave: Realidad Aumentada, Rehabilitación, Segmentación de Imagen, Terapia, Videojuego

ABSTRACT

This Final Degree Project is based on the development of two video games in the Unity cross-platform development engine and programming in the C# and Python language. The objective of these video games is the implementation in rehabilitation therapies for patients with dysfunctions in the upper limb. The program will segment and monitor the patient's hand, with which it will interact with the different functionalities of the video games, this makes the rehabilitation therapies more pleasant and friendly to the patient. This system allows a reduction in the typical expenses of these systems using only a screen and a traditional webcam.

Key Words: Augmented Reality, Rehabilitation, Image Segmentation, Therapy, Video Game

Índice Principal

Índice de Figuras

1. INTRODUCCIÓN Y OBJETIVOS

1.1. Presentación del tema

Con los avances tecnológicos en alza, la sociedad ha sufrido un cambio en su mentalidad en relación con los dispositivos inteligentes. Hoy en día es normal encontrarse este tipo de dispositivos en los lugares más comunes, pedido automático en establecimientos, detección automática de matrículas, pago con el dispositivo móvil...etc.

Esto nos demuestra que el ser humano cada vez le resulta más normal el uso de las TICs, y no solo eso, si no que cada vez lo prefiere más. En la siguiente gráfica veremos el avance de las tecnologías Tics en los hogares,

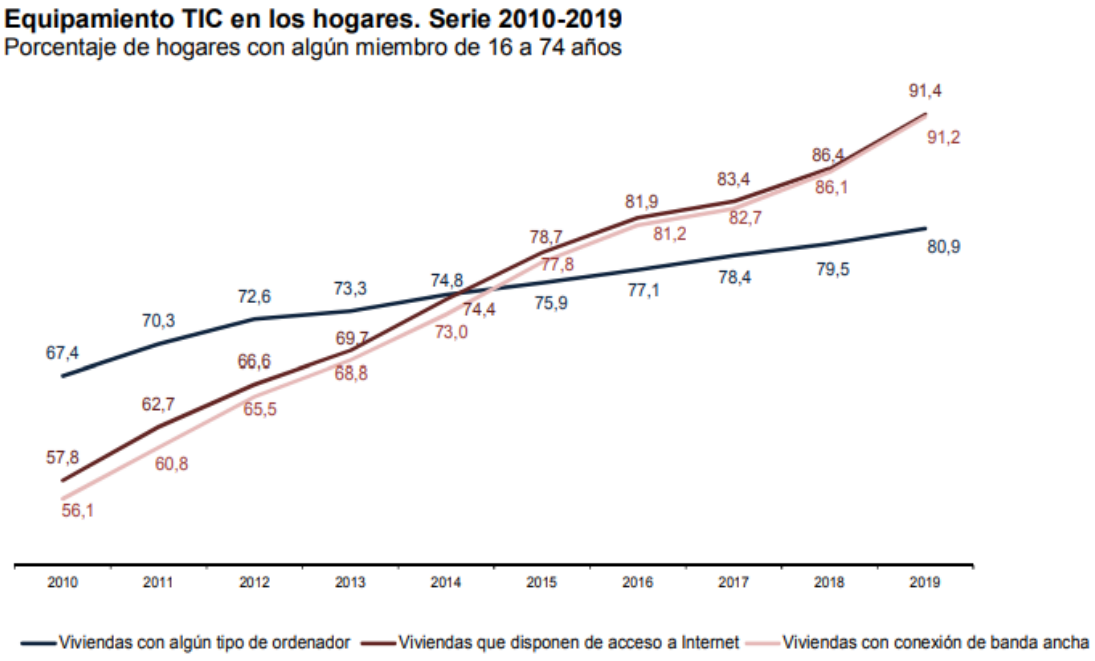


Figura 1. Equipamiento TIC en hogares. Fuente: (INE, 2019)

Como vemos en la gráfica anterior, obtenida del Instituto Nacional de Estadística (INE, 2019), vemos el número de hogares con algún tipo de ordenador ha aumentado en un 13,5% entre 2010 y 2019.

En este proyecto nos centraremos en un tipo de tecnología, la realidad aumentada. Utilizaremos esta novedosa técnica para desarrollar un entorno de realidad aumentada que será utilizado en la rehabilitación del miembro superior.

La realidad aumentada en este caso permite al paciente interactuar con elementos virtuales, pero, utilizando y viendo su propia mano o brazo.

Creemos que el uso de esta tecnología en el ámbito de la rehabilitación puede llevar a mejorar la versatilidad de los tratamientos y mejorar la experiencia del paciente durante estos, utilizando los videojuegos como base.

1.2. Objetivos

Este proyecto está realizado en el Instituto de las Tecnologías Avanzadas de la Producción (itap, 2023), en un convenio de colaboración con el Instituto de Rehabilitación Funcional La Salle en Madrid, y será utilizado en sus instalaciones en un futuro cercano.

El objetivo principal del proyecto ha sido desarrollar un entorno de realidad aumentada basado en dos videojuegos, que consigan una sinergia entre las necesidades del paciente y del profesional de la rehabilitación.

Con esto, conseguiremos que el paciente se sienta cómodo y estimulado durante la sesión, y a su vez, que el profesional a su cargo obtenga información continua del proceso. Además de esto, el proyecto solo necesita una cámara, una pantalla, un soporte y un ordenador, lo que facilita mucho la instalación.

1.2.1. Objetivos específicos

El proyecto se realizará utilizando Unity, un programa que no es comúnmente enseñado en el grado cursado, por lo que deberemos familiarizarnos con el motor de desarrollo de videojuegos, y aprender sobre el entorno de creación de juegos en realidad aumentada.

Los objetivos específicos de este proyecto son los siguientes:

1. Desarrollo de juego del reloj: En este juego realizaremos una terapia guiada basada en un reloj modelado de manera digital, y una serie de objetivos que indican al paciente hacia dónde dirigirse con la mano.

2. Desarrollo de juego del ratón: Este segundo juego tiene una base similar al del reloj, salvo que en este el paciente tendrá una libertad mayor de movimiento. No es un juego tan guiado, y el paciente deberá seguir un objetivo que huye de él, en una zona de la pantalla

3. Análisis de resultado: Comprobar si la experiencia de juego es la deseada y analizar los gráficos resultantes generados con la información obtenida durante las partidas. Comprobar si los datos generados se corresponden a la realidad y si son válidos para una valoración profesional de un paciente.

2. FUNDAMENTOS TEÓRICOS

2.1. Realidad Aumentada

La realidad aumentada se puede definir como una visión directa o indirecta del mundo físico en tiempo real, que ha sido mejorada añadiendo información virtual generada por ordenador.

El objetivo de la realidad aumentada es el de simplificar la vida del usuario, aportándole información adicional a lo que ve a su alrededor, mejorando su interacción y percepción del mundo real.

Es comúnmente confundido con la realidad virtual, lo cual no es del todo preciso. La particularidad que diferencia ambas tecnologías es que, en el caso de la realidad virtual, el usuario se encuentra inmerso en un mundo sintético sin ningún estímulo del mundo real.

Se podría decir que la realidad que en los extremos se encuentran la realidad y la realidad virtual, y en un punto medio estarían la realidad y la virtualidad aumentadas respectivamente.

Comúnmente pensamos en realidad aumentada con una idea simplemente visual, pero hay expertos que piensan que esta tecnología tiene potencial para aplicarse a todos los sentidos humanos como el olfato, el tacto o el oído, Esto permitiría a personas que han perdido alguno de los sentidos, sustituir esos estímulos que no son capaces de procesar, por otros que afecten a sentidos funcionales que les queden. (Carmigniani & Furht, 2011)

Figura 2. Gráfico de diferencias entre realidad aumentada, realidad virtual y realidad mixta. Fuente: (Carmigniani & Furht, 2011)

2.1.1. Progreso de la Realidad Aumentada

La realidad aumentada es una tecnología en auge, desde su desarrollo no ha dejado de ampliar su zona de actuación y popularidad. A pesar de que no sea nuestro caso, vamos a ver una estadística sobre la venta de cascos de realidad aumentada para hacernos una idea de este avance.

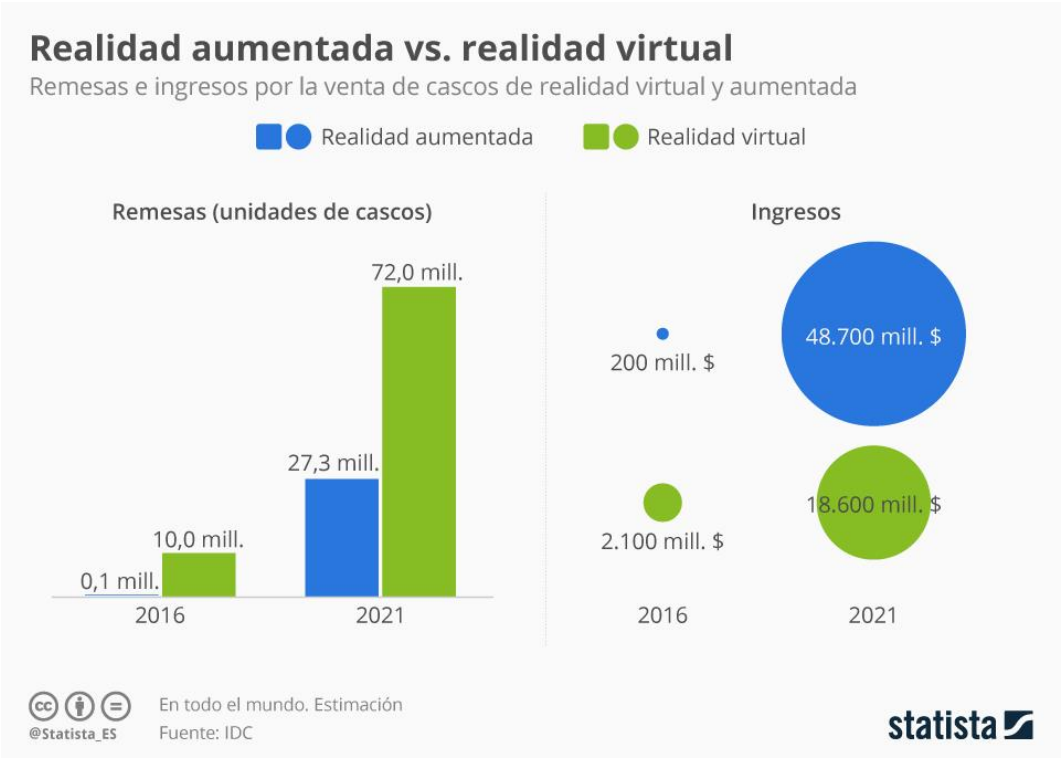


Figura 3. Progresión en los ingresos por venta de cascos de realidad aumentada y virtual. Fuente: (Moreno, 2023)

En este gráfico vemos el aumento de cascos de realidad aumentada y realidad virtual desde el 2016 al 2021. Como vemos hay un aumento del 24350% en el caso de la cotización de la realidad aumentada. (Fuente IDC).

Esto nos indica que la evolución de la tecnología tiene una tendencia creciente, lo que nos puede indicar que los proyectos de realidad aumentada no solo no van a parar de desarrollarse, si no que va a haber una mejoría en estos.

Como este proyecto tiene su principal objetivo en la rehabilitación, nos parece conveniente ver su evolución en el sector sanitario, al que va dirigido.

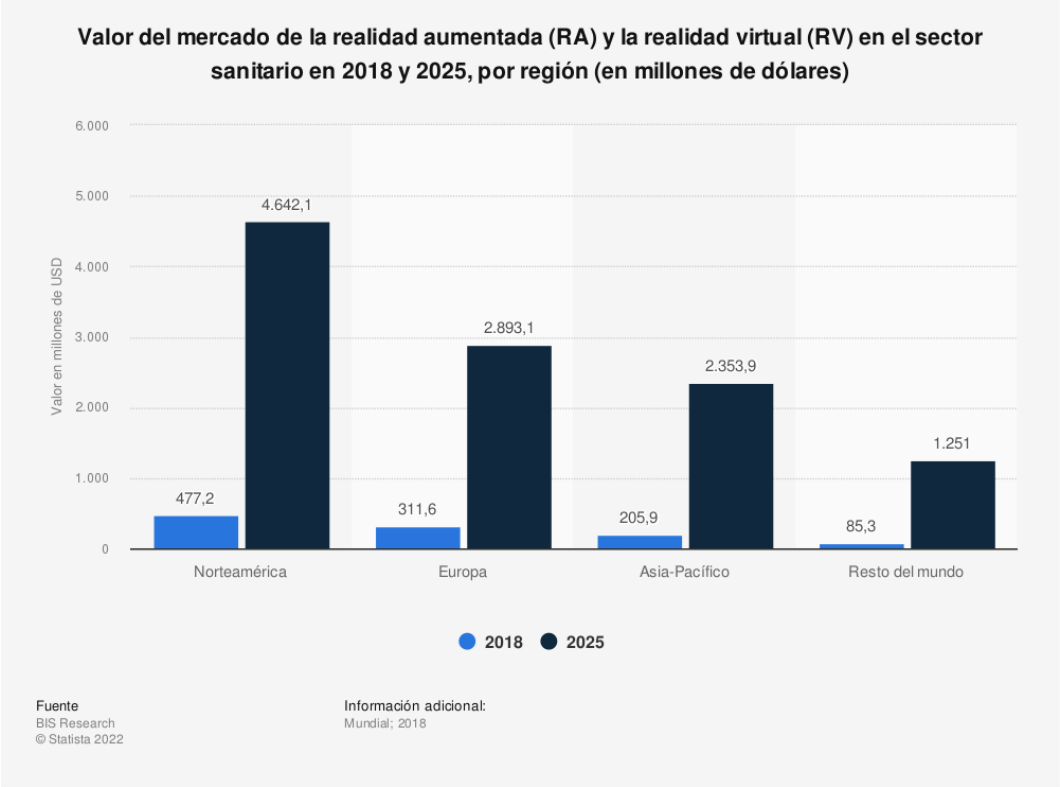


Figura 4. Valor del mercado de la realidad aumentada y virtual en el sector sanitario en 2018 y 2025. Fuente: (Fernández, 2023)

Como vemos en la gráfica anterior obtenida de Statista, en colaboración con BIS Research, el valor del mercado de la realidad aumentada y la realidad virtual en el sector sanitario en Europa se prevé que sea de 2.800 millones de dólares en 2025, siendo de 311 millones en 2018, esto supondría un aumento del 900% en 7 años.

También podemos ver como el crecimiento en Norte América, a pesar de no ser el objetivo de este proyecto, es previsiblemente mucho mayor que en Europa.

2.1.2. Aplicaciones de la Realidad Aumentada

La realidad aumentada se ha expandido en muchos sectores como el de la medicina, el militar, la fabricación, el entretenimiento, la visualización, el entretenimiento y la robótica. (Mekni & Lemieux)

En este apartado nos centraremos en el sector médico y el del entretenimiento o videojuegos, y mencionaremos de manera general algunos usos en los demás sectores.

2.1.2.1. Aplicaciones en el Sector Médico

En el sector médico la función principal de la realidad aumentada es la necesidad de visualizar datos médicos al paciente en el mismo espacio físico, fue en torno al 1988 cuando se realizó el primer sistema de realidad aumentada.

Imágenes ultrasonido

Una aplicación de la realidad aumentada en el sector médico es la visualización de imágenes de ultrasonido. Esta tecnología, por ejemplo, es capaz de visualizar imágenes volumétricas de un feto sobre el abdomen de la mujer embarazada, de manera que parece como si la imagen estuviera dentro de su cuerpo. (Mekni & Lemieux)

Cirugía Cardiaca

Además de sistemas de visualización de información centradas en el paciente, existen sistemas que tienen como objetivo asistir a los médicos en su labor, vamos a ver algunos de ellos.

Un ejemplo de este caso puede ser el uso de la realidad aumentada en el ámbito de la cirugía, en este ejemplo vemos la cirugía cardiaca. En la cirugía tradicional depende altamente de las imágenes 2D de rayos X y de la experiencia del cirujano. Esto hace que la capacidad del cirujano para ajustar la posición y orientación de su mano sobre el paciente, con la información que extrae de la imagen 2D.

A pesar de la experiencia del especialista, se pueden producir fallos en la identificación de la arteria coronaria, es por eso por lo que se han propuesto sistemas de visualización 3D en las imágenes obtenidas.

A pesar de que hay numerosos retos que quedan por resolver para poder utilizar estos sistemas, es una alternativa prometedora. (Ha & Hong, 2016)

2.1.2.2. Aplicaciones en los Videojuegos

A lo largo de la historia se han desarrollado gran cantidad de video juegos que incorporan la realidad aumentada como base de su funcionamiento, pero aquí se mencionarán dos de los más importantes:

Pokemon GO

Pokemon Go es un juego de realidad aumentada capaz de combinar el uso del teléfono móvil con la exploración física del mundo real. Con el uso del GPS y de Google Maps, proporcionó a los usuarios una experiencia en realidad aumentada única, permitiéndoles encontrar, capturar y coleccionar especies de *Pokemon* mientras exploraban el mundo real. Se generó un fenómeno a su alrededor en 2016, donde tanto adultos como jóvenes se juntaron en su comunidad. (Chong, Sethi, Loh, & Lateef, 2018)

El principal atractivo de este videojuego fue su capacidad de incorporar modelos de los personajes en el mundo real:

Figura 5. Captura de pantalla del videojuego Pokemon GO. Fuente: (Dorward, Mittermeier, Sandbrook, & Spooner, 2016)

Invizimals

Un juego publicado por la compañía de videojuegos Sony, en colaboración con una desarrolladora española, fue uno de los juegos de realidad aumentada más exitosos de su momento.

En este caso el programa generaba los personajes del videojuego en la imagen que tomaba la cámara de la consola, y para ello se ayudaba de unas cartas con

un patrón impreso. Esto permitía al sistema identificar donde colocar el modelo y que personaje es el que se debía modelar.

Figura 6. Vista del juego Invizimals en la consola PSP. Fuente: (Tyni, Kultima, & Mäyrä, 2013)

2.1.2.3. Otras Aplicaciones

La realidad aumentada se ha utilizado en otros sectores como pueden ser: (Mekni & Lemieux)

1- Militar: Mostrar el campo de batalla con información adicional para planificación

2- Robótica: Colaboración hombre-máquina

3- Educación: Visualizar conceptos abstractos con los que se puede interactuar

4- Marketing: Publicidad y visualización de objetos modelados en realidad aumentada

2.2. Rehabilitación del Miembro superior

Este proyecto se basa en una terapia de rehabilitación del miembro superior. Para entender bien los juegos, primero debemos entender para que parte del cuerpo están orientados.

2.2.1. Miembro Superior

El miembro superior está caracterizado por su capacidad para agarrar, golpear y realizar actividades motoras finas de manipulación. En las articulaciones de la extremidad superior se produce la interacción sincronizada, que permite realizar movimientos suaves y eficientes para tareas específicas. (Moore, Dalley, & Agur, 1992)

El miembro superior consta de cuatro partes:

1- Hombro: Segmento más próximo de la extremidad al tórax y al cuello.

2- Brazo: Primer segmento del miembro superior libre y segmento más largo del miembro.

3- Antebrazo: Segundo segmento más largo de la extremidad y se extiende entre el codo y la muñeca.

4- Mano: Parte del miembro superior distal al antebrazo. Está formado por la muñeca, la palma, el dorso y los dedos, y esta provisto de terminaciones nerviosas.

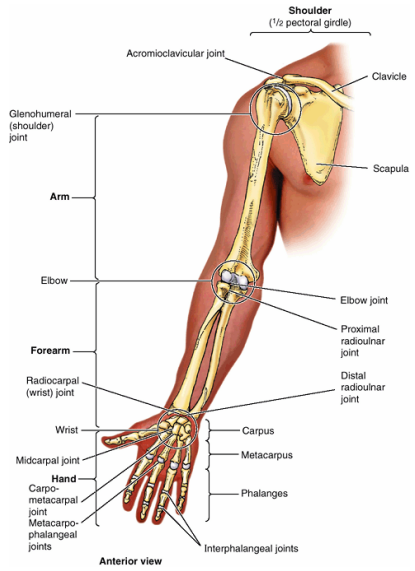


Figura 7. Partes del miembro superior. Fuente: (Moore, Dalley, & Agur, 1992)

El derrame es una de las causas más comunes de invalidez y de pérdida de calidad de vida. Esto es debido a que el miembro superior y el miembro inferior están involucrados en la mayor parte de situaciones de la vida diaria, por lo que realizar una correcta rehabilitación es muy importante en su recuperación. (Lam Phan, Huong Le, Min Lim, Ho Hwang, & Koo, 2022)

La rehabilitación con realidad aumentada muestra mucho potencial para mejorar esta extremidad tras un derrame, que se podría extrapolar a otras lesiones funcionales del miembro.

En un estudio realizado por una cooperación de universidades coreanas, se ha demostrado cómo el uso de tratamientos con realidad aumentada ha influenciado significativamente las funciones del miembro superior. En este estudio se ha realizado una comparación entre la mejora de un grupo de pacientes tratados con métodos tradicionales, y otro grupo tratado con métodos combinados entre tradicionales y con realidad aumentada. El estudio brindó los siguientes datos resultantes: "(SMD = 0.657; 95% CI = 0.287 to 1.026; $p < 0.001$)".

Este estudio demuestra que la rehabilitación combinada de realidad aumentada y rehabilitación tradicional es más beneficiosa que realizar únicamente la convencional. (Lam Phan, Huong Le, Min Lim, Ho Hwang, & Koo, 2022)

El SMD representa a la diferencia media estandarizada, y es muy utilizada debido a que tiene una unidad uniforme en los diferentes estudios. Resulta de la división de las medias de diferencias con sus respectivas desviaciones estándar. Se podría determinar como que una SMD de 0.2, 0.5 y 0.8 se consideran pequeña, media y alta respectivamente. (Andrade, 2020).

El CI indica el Intervalo de confianza del experimento y el rango de valores estimado de la media. En nuestro caso vemos que la confianza de 95% se encuentra en el rango de 0,287 a1,026, por lo que es probable que la verdadera diferencia de medias se encuentre entre esos valores. El hecho de que el rango comience en 0,287 nos indica que la diferencia mínima es esa, por lo cual es estadísticamente significativa.

En el caso del valor p, tenemos que "p<0,001". El valor p es una medida de la evidencia en contra de la hipótesis nula. En nuestro caso podemos decir que hay una evidencia bastante fuerte en contra de que la hipótesis sea nula, pudiendo afirmar que la diferencia entre los grupos no se ha debido al azar.

Todos estos datos nos permiten afirmar que existe una diferencia estadísticamente significativa entre ambos grupos,

2.2.2. Método de Rehabilitación

Para este proyecto nos centraremos en la rotación externa e interna del hombro, y en la flexión y extensión del codo. De manera que se realicen movimientos paralelos a la superficie de apoyo, con una disposición similar a la de la ilustración.

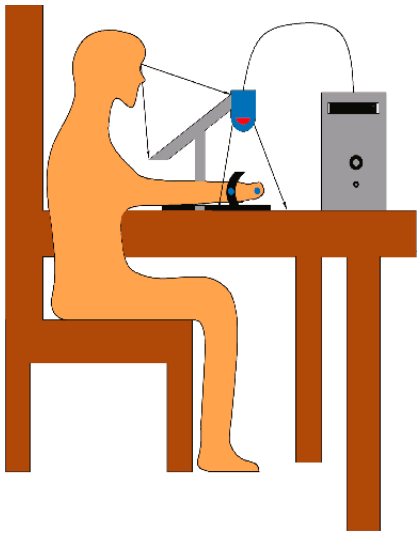


Figura 8. Disposición de los elementos para la rehabilitación. Fuente: (Lam Phan, Huong Le, Min Lim, Ho Hwang, & Koo, 2022)

En nuestro caso la cámara estará apuntando hacia la parte trasera del soporte en el que se encuentra la pantalla, en la cual se encuentra un espejo. Esta reformulación de la estructura permite al paciente ver la mano desde un punto de vista muy similar al que tendría si no estuviera presente la pantalla.

Figura 9. Imagen de soporte para la pantalla. Fuente: (Elaboración Propia)

Figura 10. Imagen de soporte de la pantalla desde vista al espejo trasero. Fuente: (Elaboración Propia)

3. HERRAMIENTAS UTILIZADAS

3.1. Unity

3.1.1. Historia de Unity

Unity es un motor de juegos y un entorno de desarrollo integrado que se utiliza principalmente para la creación de videojuegos. Fue creado por tres programadores llamados Nicholas Francis, Joachim Ante y David Helgason

Inicialmente se propusieron hacer negocio realizando videojuegos, pero tras ver la necesidad de una mejor tecnología de desarrollo, terminaron creando una herramienta para crear los juegos, que ha terminado siendo lo que es ahora Unity.

El atractivo principal de Unity fue el soporte a desarrolladores independientes que no eran capaces de afrontar licencias costosas para la tecnología de desarrollo.

Para la comercialización del motor, les pareció buena idea desarrollar un juego con él, para así probar los límites de éste. Con este motivo desarrollaron "Gooball" en marzo de 2005. (Haas, 2014).

Figura 11. Captura de pantalla del videojuego Gooball. Fuente: (Haas, 2014)

3.1.2. Editor de Unity

El editor de Unity está formado por gran cantidad de ventanas, las cuales se utilizan para diversas funciones. Las más comunes son las siguientes:

1- *Project*
2- *Scene*
3- *Hierarchy*
4- *Game*
5- *Inspector*
6- *Console*

Project

Esta ventana es la encargada de visualizar todos los archivos que están importados en nuestro proyecto. Todo lo que tenga que estar en nuestro juego, tiene que estar previamente en la ventana Project del editor.

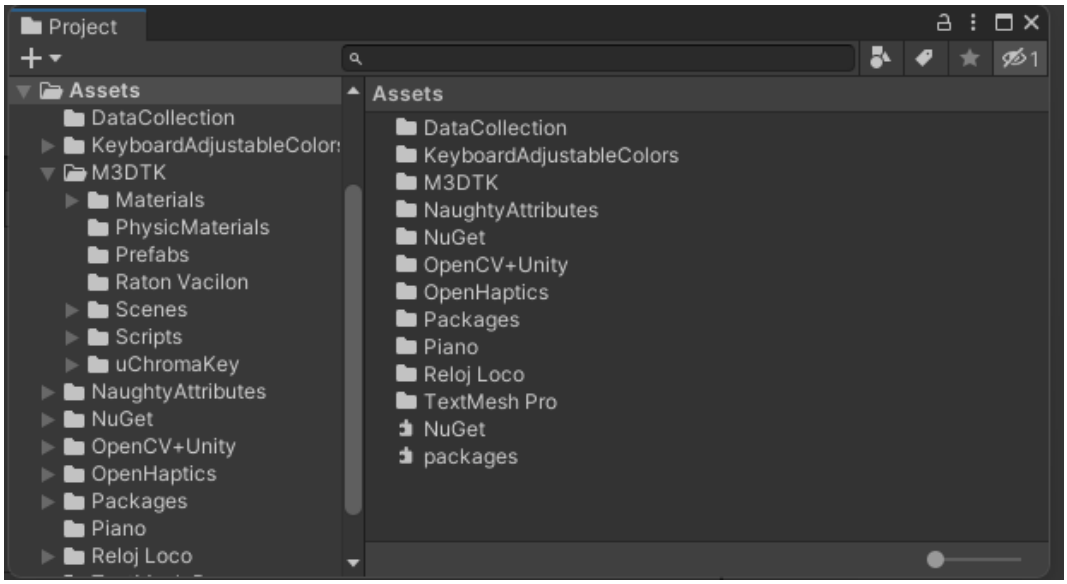


Figura 12. Ventana Project del editor de Unity. Fuente: (Elaboración Propia)

Esta ventana se podría asemejar al administrador de archivos de *Windows*, en la cual podemos desplazarnos entre carpetas.

Scene

Esta ventana es una de las que más se utilizan en el editor. En esta ventana podemos ver la disposición de los objetos en escena.

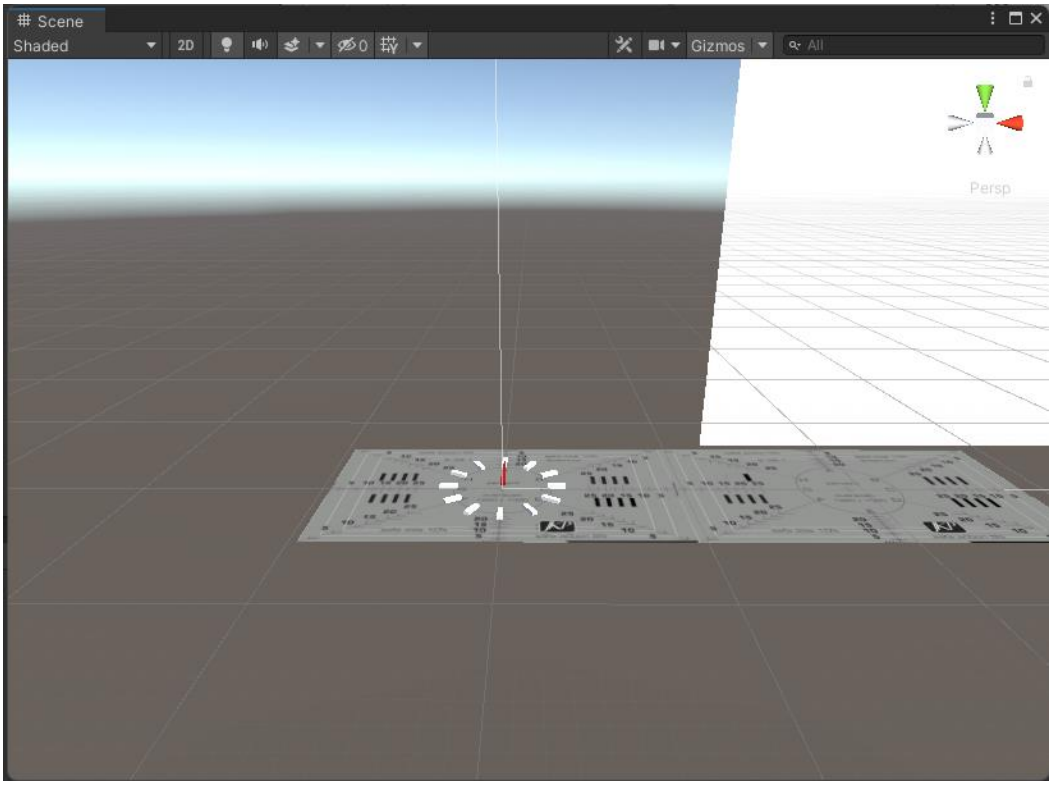


Figura 13. Ventana Scene del editor de Unity. Fuente: (Elaboración Propia)

Desde la propia ventaja podemos modificar la posición, orientación y tamaño de los objetos. Para movernos en ella utilizamos el ratón y el teclado según la configuración elegida.

Esta escena tiene un eje de coordenadas global, referenciado en la parte superior derecha de la ventana. El eje Y se representa en color verde, el eje X en color rojo y el eje z en color azul.

Para mover un objeto deberemos hacer *click* sobre él, y nos aparecerá otros tres ejes de coordenadas, los cuales son relativos al objeto, y pueden ser diferentes a los ejes globales de la escena. Al aparecer el sistema de coordenadas, hacemos *click* sobre el eje sobre el que queremos hacer el movimiento, y se arrastra el objeto.

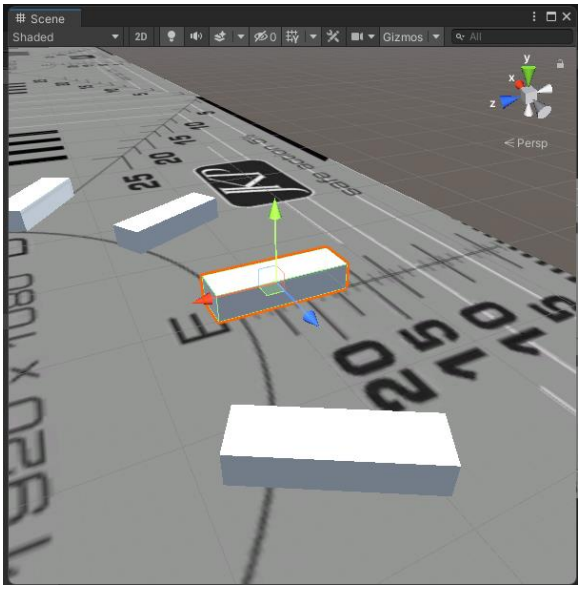


Figura 14. Selección de un elemento en la ventada Scene del editor de Unity. Fuente: (Elaboración Propia)

Hierarchy

Esta ventana se utiliza en conjunto con la ventana *Scene*. En esta ventana aparecen todos los objetos que se encuentran en la escena. El nombre de la ventana viene dado a que organiza jerárquicamente los objetos del juego *(GameObjects)* con el método Padre-Hijo.

Este método permite realizar diversas funciones como:

7- Modificar atributos del padre que afecten al hijo
8- Activar o desactivar padre e hijo simultáneamente
9- Acceder al objeto hijo a través del objeto padre

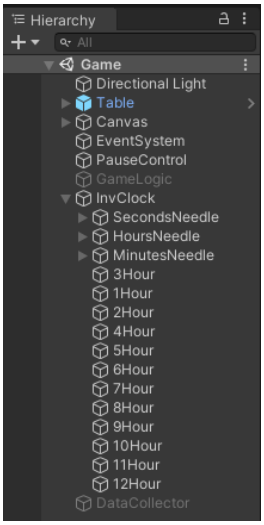


Figura 15. Ventana Hierarchy del editor de Unity. Fuente: (Elaboración Propia)

En esta ventana también podemos visualizar que componentes hay en cada escena, dado que hay juegos de cierta complejidad que necesitan el uso de varias de ellas. En este caso esto nos permite realizar una descomposición estructurada del juego, reduciendo la dificultad.

Game

Un juego de Unity representa la visualización de la escena en tiempo real desde el punto de vista de una cámara. Esta cámara se puede mover a lo largo del tiempo, y su posición es elección del desarrollador.

La ventana *Game*, muestra lo que se vería en el juego si estuviera activo. Esto permite al desarrollador modificar la escena teniendo en cuenta lo que se vería en la versión final.

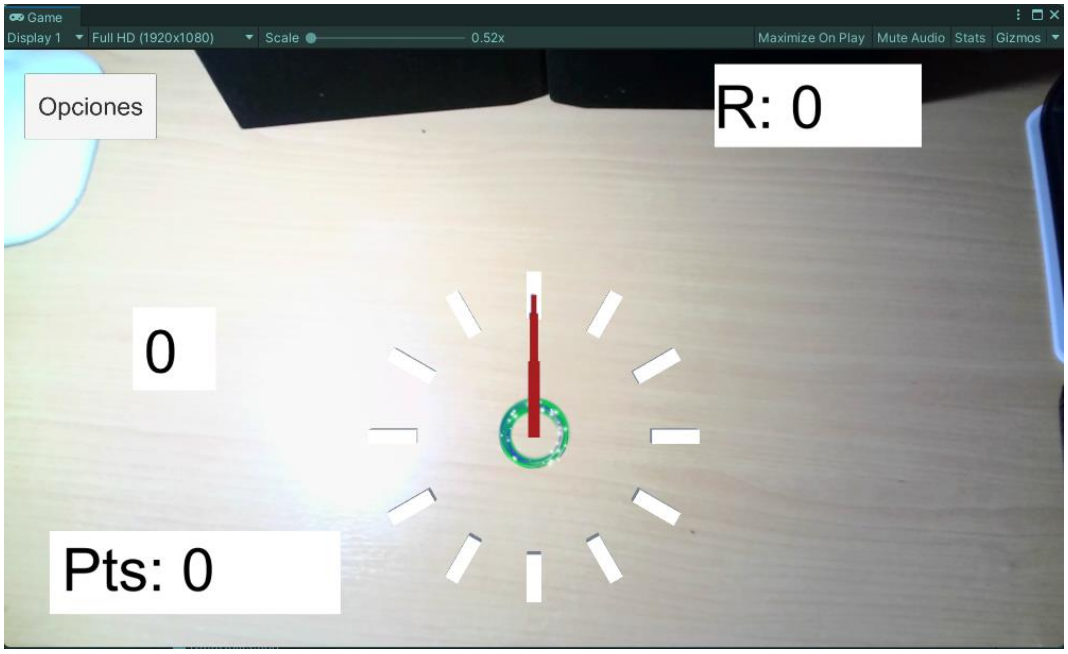


Figura 16. Ventana Game del editor de Unity. Fuente: (Elaboración Propia)

Inspector

Esta ventana nos permite ver la configuración del objeto del juego *(GameObject)* que esté seleccionado en ese momento. Desde esta ventana podemos ver, modificar o añadir atributos a los objetos. Algunos de estos atributos puedes ser Scripts, Posición, Escala, Rotación, Audio, Renderización, Atributos gráficos…

Lo normal es utilizar esta ventana para asignar los valores iniciales de los objetos, dado que, las modificaciones de estos se suelen realizar utilizando ficheros de código con instrucciones.

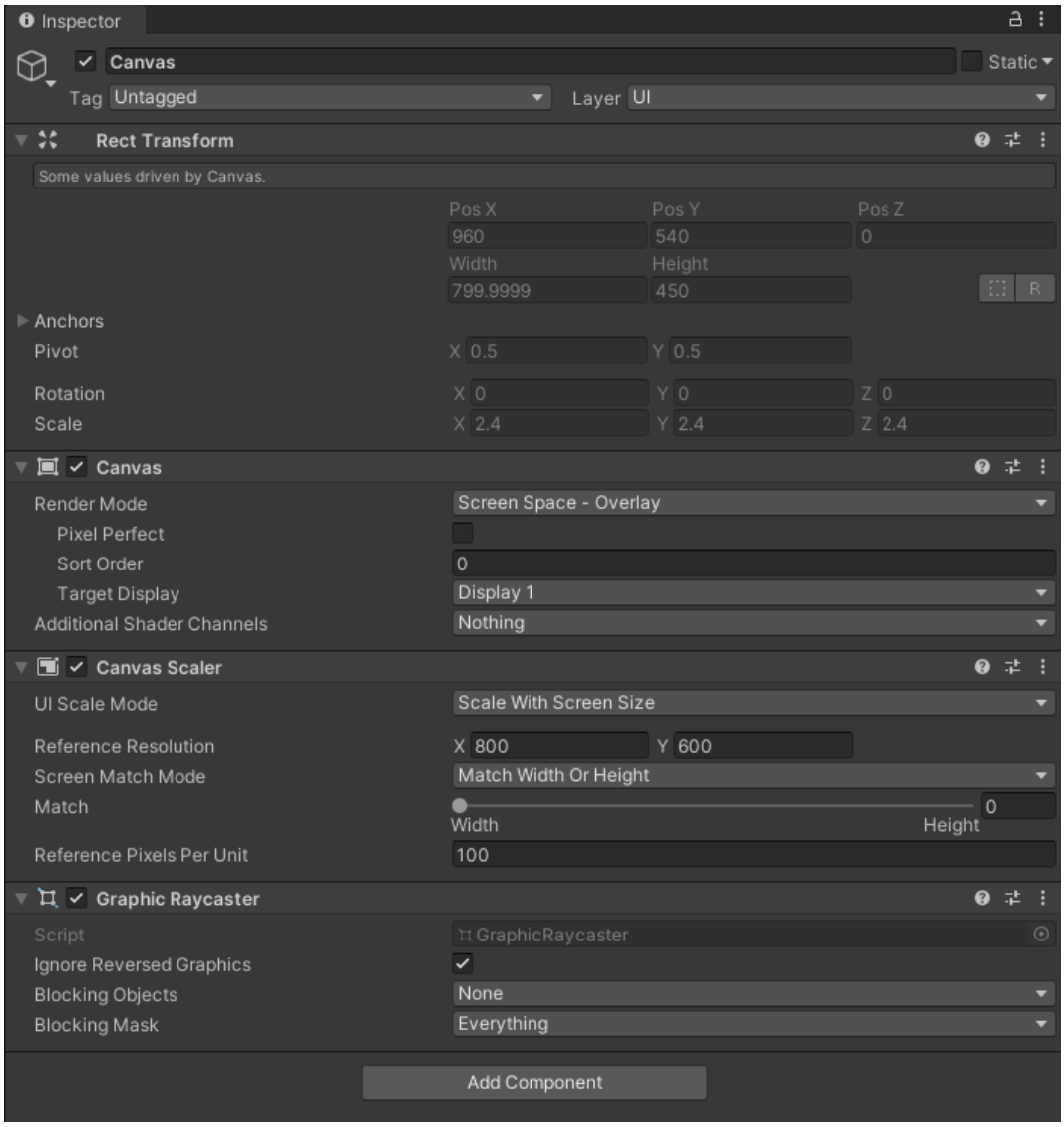


Figura 17. Ventana Inspector del editor de Unity. Fuente: (Elaboración Propia)

Console

Esta ventana es la consola del programa, o terminal. Se utiliza para visualizar los fallos, advertencias o mensajes que ocurran en el transcurso del juego. El programador puede utilizarla para visualizar parámetros o para obtener información útil de la progresión.

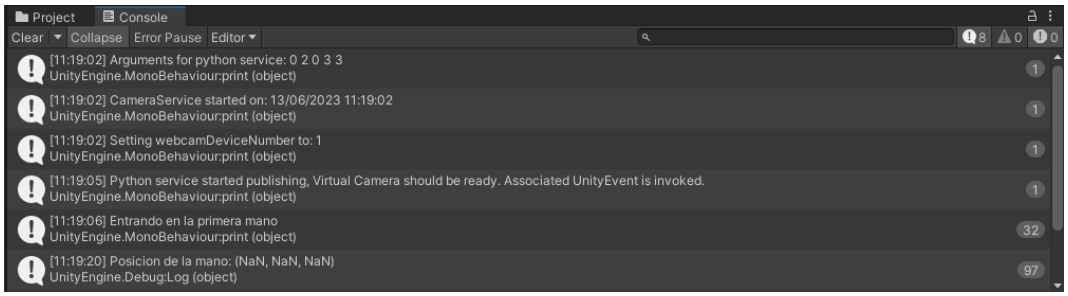


Figura 18. Ventana Console del editor de Unity. Fuente: (Elaboración Propia)

La disposición que me ha resultado más cómoda para visualizar toda la información de las ventanas durante el desarrollo ha sido la siguiente:

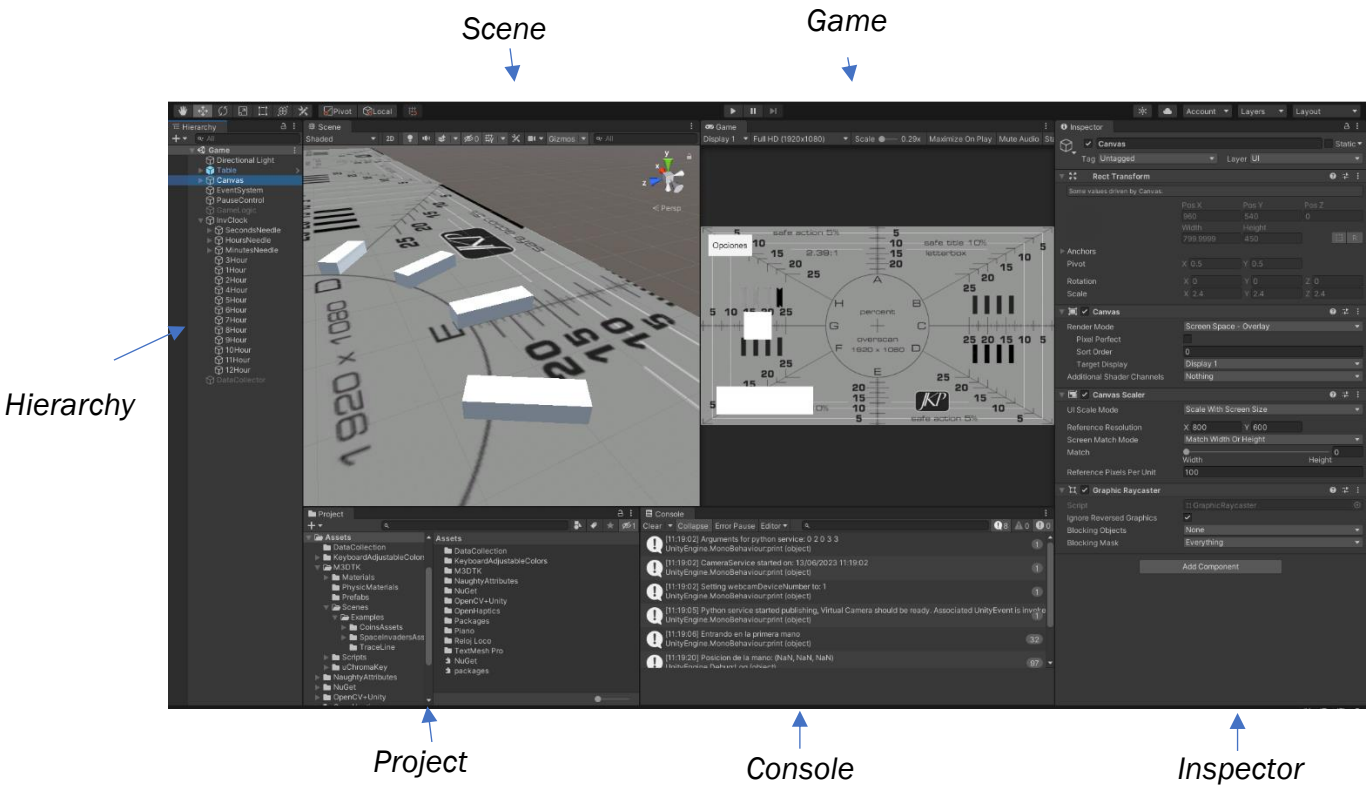


Figura 19. Disposición general de las ventanas en el editor de Unity. Fuente: (Elaboración Propia)

Con esta configuración damos más importancia a la ventana de *Scene* y la de *Game*.

3.1.3. Entornos de Realidad Aumentada

Unity contiene diversos paquetes que soportan el desarrollo de videojuegos en realidad aumentada. Uno de los más usados es AR Foundation.

AR Foundation permite crear juegos multiplataforma en Unity, seleccionando que características de la realidad aumentada se desea que la aplicación tenga, simplemente añadiendo estos componentes a la escena del juego.

Cuando se ejecuta esta aplicación en un dispositivo compatible con la realidad aumentada, AR Foundation habilita estas características que han sido añadidas a la escena, facilitando mucho el desarrollo de este tipo de videojuegos. (Technologies, 2023)

El problema que tenemos en este proyecto es que la localización de los elementos y el tratamiento de las imágenes es muy específica, no nos interesa realizar un videojuego en realidad aumentada común, si no uno pensado y desarrollado para la rehabilitación del miembro superior.

Es por esto por lo que en nuestro caso vamos a utilizar otro entorno de realidad aumentada que no se encuentra entre los paquetes nativos de Unity

3.1.4. Entorno de AR utilizado

Este entorno ha sido realizado en el Instituto de las Tecnologías Avanzadas de la Producción (itap, 2023), de la Universidad de Valladolid, y ha sido desarrollado por Guillermo Sánchez Brizuela (Sánchez Brizuela, Guillesanbri, 2023). En este apartado se van a explicar las partes de las que consta:

<u>Componentes:</u>

1- Archivo de Gestión de Python
2- Proyecto de Unity
3- Servicio de Python

En este apartado vamos a centrarnos en explicar cómo funciona el entorno, más que en cómo está desarrollado, dado que eso no es tan importante para el proyecto en el que estamos.

Archivo de gestión de Python

Este archivo de tipo *.batch*, es el encargado de abrir el entorno de anaconda, utilizar el servicio de Python, y cerrar el entorno de anaconda.

Este archivo se abre desde uno de los ficheros de *Unity* en el momento que se inicia la transmisión de la imagen. Vamos a ver lo que se hace con el servicio de Python

Coordenadas de los puntos de referencia de las manos

Se produce una comunicación entre el Servicio de Python y el Proyecto de Unity. El Servicio de Python obtiene las coordenadas de los puntos de referencia de las manos, y se lo envía a través de un mensaje por socket al proyecto de Unity con un patrón publicador/subscriptor. En el proyecto tenemos un script que modifica las posiciones de unos objetos del juego *(GameObjects)* en función de estas coordenadas.

Oclusión

En el proyecto de Unity se produce la transmisión de imagen obtenida por la cámara. En esta imagen se diferencia entre el fondo y las manos. Estas dos imágenes se muestran en distintas zonas del plano, permitiéndonos diferenciar lo que está por encima y por debajo de las manos en el videojuego.

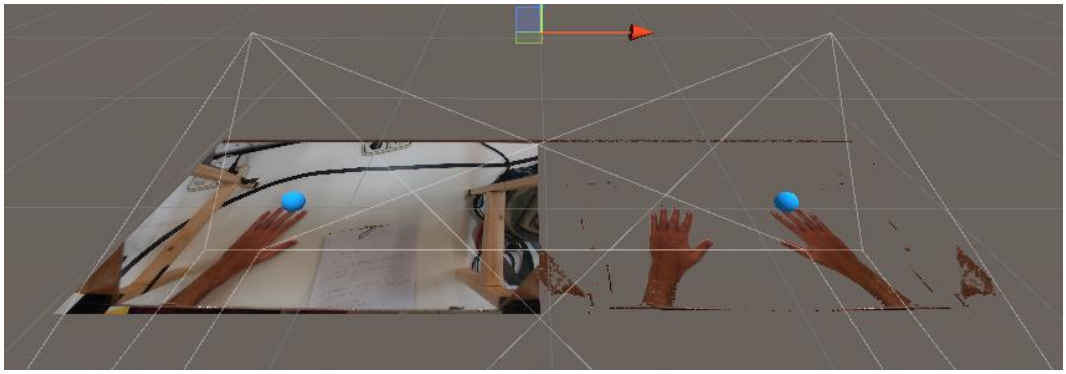


Figura 20. Muestra de la oclusión en la escena de Unity. Fuente: (Sánchez Brizuela, Documentación M3Display - Toolkit de desarrollo, 2021)

Unity

El objeto principal del proyecto es el *Table*. Los objetos del juego hijos de este objeto son los siguientes:

1- *Plane*: Plano físico que retransmite la imagen capturada por la cámara.

2- *WebcamManager*: Este componente se encarga de la gestión de la captura de imagen y las webcams virtuales.

3- *ComunicationsManager*: Este componente gestiona la comunicación con el servicio de Python

4- *LandmarksManager*: Recibe la información de las coordenadas de los puntos de referencia que le ha llegado al *ComunicationsManager*, y actualiza los puntos de la escena con esas posiciones.

5- *TrackerManager*: Es el encargado de permanecer en escucha por si llega un mensaje, si ese es el caso, se lo envía al *ComunicationsManager*

6- *PythonServiceManager*: Ejecuta y detiene el servicio de Python que extrae la información de las cámaras.

7- *UnityCameras*: Agrupa dos cámaras en la escena:

- *HandsCamera*: Captura los elementos segmentados (Las manos)
- *BackgroundCamera*: Obtiene el fonde del vídeo

8- *Origins*: Almacena los objetos que se encuentran en el plano

- *MainPlane*: Lo que se coloca aquí se encontraría debajo de las manos
- *SecondaryPlane*: Lo que se sitúa aquí se encontraría por encima de las manos.

3.1.5. Mediapipe Hand Tracking

Mediapipe Hand Tracking es una solución para la monitorización de las manos desarrollada por Google. Este algoritmo emplea un "ML Pipeline", con dos modelos trabajando simultáneamente:

1- Un detector de la palma, capaz de localizar la palma de la mano con un cuadro delimitador. El detector de palma emplea un *"Single-Shot detector model"*, que es un modelo de detección de una pasada, reduciendo mucho el tiempo de procesamiento manteniendo una gran precisión en la detección.

2- Un modelo que, utilizando la imagen recortada del detector de la palma, es capaz de proporcionar puntos de referencia en la mano. Este m odelo es muy preciso, siendo capaz de detectar manos parciamente visible e incluso oclusiones. Este modelo tiene tres salidas:

- 21 puntos de referencia, con su correspondiente (x, y)
- Un valor de probabilidad de presencia de una mano en la imagen
- Una clasificación binaria de la mano dominante. (Diestro, Zurdo)

Para que los modelos sean capaces de detectar eficientemente los objetos, se ha utilizado una base de datos con diferentes aspectos del problema:

1- Imágenes en la calle: Esta base de datos contiene seis mil imágenes con gran variedad de ambientes y diversidad geográfica, con distinta luminosidad y apariencia de las manos. La limitación de esta base de datos es que las manos no presentan una configuración articulada compleja.

2- En casa: Esta base de datos tiene diez mil imágenes desde varios ángulos de los gestos típicos de las manos. La limitación de esta base de datos es que fueron obtenidas tan solo de treinta personas, por lo que el fondo de las imágenes no es muy variado.

3- Base de datos sintética: Para mejorar la detección de otras posibilidades de configuración articular de las manos, se utilizó un modelo sintético de alta calidad con varios fondos. Este modelo permitía distintas texturas y tonos de color de piel. Esto permitió obtener cien mil imágenes de la mano a partir de vídeos.

El hecho de utilizar la imagen de la palma recortada permite reducir el número de datos necesarios y permite que la red dedique la mayor parte de su capacidad en la localización de los puntos de referencia.

Otra manera de hacer más eficiente la detección en video en directo, es usar la detección del fotograma i-1 como argumento en la detección del fotograma i, esto hace que el detector de palma solo actúe en el instante inicial, o si en algún momento se ha perdido la señal de la mano. (Zhang, y otros, 2020)

Veamos una tabla de resultados en su uso en dispositivos móviles, dado que la gran ventaja de este método es que busca la ligereza y velocidad en el procesamiento de la imagen.

Model	Params (M)	MSE	Time(ms) Pixel 3	Time(ms) Samsung S20	Time(ms) iPhone11
Light	1	11.83	6.6	5.6	1.1
Full	1.98	10.05	16.1	11.1	5.3
Heavy	4.02	9.817	36.9	25.8	7.5

Figura 21. Tabla de resultados de Mediapipe Hand Tracker en dispositivos móviles. Fuente: (Zhang, y otros, 2020)

Esta tabla muestra, según el tamaño del modelo, cuales son el tiempo de procesamiento medio, y el error cuadrático medio en la detección. Se podría decir que el mejor será el modelo *Full*, dado que es el más equilibrado en cuanto a relación tamaño, error y tiempo de procesamiento.

4. FUNDAMENTO DEL DESARROLLO DEL TFG

4.1. Instalación del entorno

El entorno utilizado para el desarrollo ha sido el mencionado en el apartado anterior. Vamos a analizar los pasos que se han realizado para su instalación en el equipo.

Instalación de Anaconda

El primer paso es realizar la instalación de *Anaconda*. *Anaconda* es una distribución de libre acceso, que facilitará la gestión de librerías y aplicaciones de Python. En el caso de que el dispositivo asociado en el que se requiera la instalación no tenga los recursos necesarios para la ejecución de anaconda, existe una versión reducida de la distribución llamada *Miniconda*, la cual igualmente funcional.

Una vez instalado *Anaconda* o *Miniconda*, debemos instalar una versión de *Python*. En nuestro caso, todo el sistema está optimizado para su ejecución en *Python 3.9.7*.

Instalación de entorno de Anaconda

En anaconda deberemos instalar todas las librerías necesarias para que el servicio de Python funcione correctamente. Estas librerías se pueden comprimir en un archivo de instalación para introducirlas automáticamente, o por el contrario realizar la instalación de forma manual una a una.

La lista completa de todas las librerías con su correspondiente versión se podrá encontrar en los anexos.

Instalación de carpetas

Como mencionamos en el apartado anterior, el entorno de Python tiene diversos ficheros encargados de la localización de posiciones de las manos, y el envío de estas mediante socket al programa de Unity.

Estos ficheros se almacenan en la carpeta “C:\ProgramData” con el nombre M3DPythonService.

Posteriormente se descargará *UnityCapture-master*, encargado de la gestión de las cámaras virtuales del entorno de Unity, y se guardará de nuevo en "C:\ProgramData".

Lo siguiente que debemos realizar es la instalación del entorno virtual de librerías de Python, como vimos en el apartado anterior, se puede hacer librería a librería de manera individual o con un ejecutable.

Con esto realizado deberían funcionar correctamente todos los juegos realizados con este entorno de realidad aumentada.

Para facilitar la instalación se ha realizado una carpeta de instalación automática que utiliza archivos por lotes que ejecutan comandos de Windows para realizar toda esta tarea más fácil.

4.2. Método de desarrollo

Para este proyecto se ha utilizado la metodología incremental de desarrollo. Este método parte de un diseño inicial con características básica, y según avanza el desarrollo se van realizando versiones cada vez más completas del sistema, hasta llegar a una versión final que satisfaga las necesidades del usuario, y cumpla con todos los requerimientos de ésta. (León Yacelga, Acosta Espinoza, & Díaz Vásquez, 2021).

Se ha seleccionado esta metodología dado que, la comunicación con el Instituto de Rehabilitación Funcional La Salle en Madrid, ha sido continuado, esto ha permitido ir añadiendo requerimientos y especificaciones adicionales según avanzaba el desarrollo del proyecto.

Figura 22. Métodología de desarrollo del proyecto. Fuente: (Elaboración Propia)

5. JUEGO DEL RELOJ

5.1. Idea inicial

El juego del reloj está pensado con el objetivo de ofrecer una rehabilitación de movimientos guiados, buscando un movimiento de tipo punto a punto con el centro como punto común. El movimiento aproximado sería el siguiente:

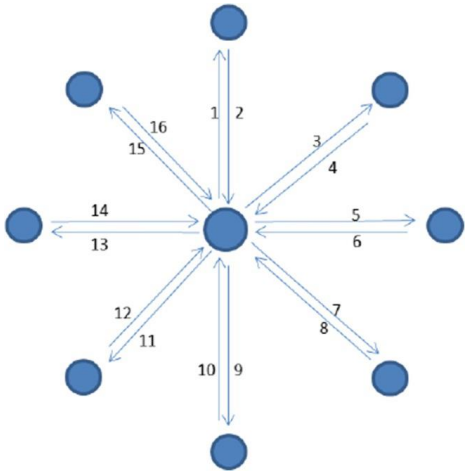


Figura 23. Ilustración de movimiento en juego del Reloj. Fuente: (Nordin, Quan Xie, & Wünsche, 2014)

El objetivo del juego es que el paciente vaya alcanzando objetivos marcados con un círculo llamativo. La localización de estos objetivos será seleccionada de manera aleatoria entre las posiciones de las horas de un reloj, y cada vez que se llegue a uno de esos objetivos se deberá volver al centro.

El juego tiene varios parámetros de configuración para que el responsable de la terapia los seleccione en función de la capacidad del paciente. Estos parámetros serán, la dificultad asociada al tamaño del reloj, la duración en tiempo, la duración en rondas y finalmente, el tiempo para que el paciente alcance el objetivo antes de que se busque uno nuevo.

5.2. Desarrollo

En este apartado vamos a ir analizando las distintas funcionalidades desarrolladas. El juego se ha realizado con dos escenas encadenadas, Menú Principal y Juego.

5.2.1. *Menú Principal*

Esta es la primera escena que aparece al iniciar el juego. En esta escena aparecen dos botones:

Figura 24. Ventana del Menú Principal del juego del Reloj. Fuente: (Elaboración Propia)

Esta ventana es un panel de *Canvas*, en el cual se han añadido dos botones interactivos, que tienen las siguientes funcionalidades.

El botón de *Opciones* inhabilita el panel de *Menú Principal*, y habilita el panel de *Menú Principal de Opciones*, como veremos en la siguiente figura:

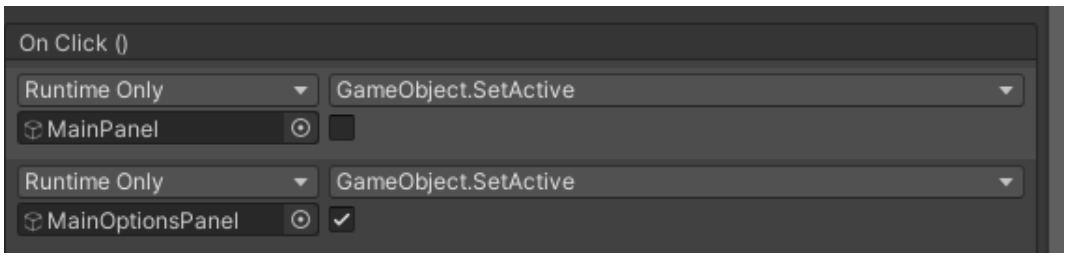


Figura 25. Ventana de acción al hacer click en boton de opciones del Menú Principal del juego del Reloj. Fuente: (Elaboración Propia)

El botón de *Iniciar Partida* está pensado para que traslade al usuario a la siguiente escena, como podemos ver en la siguiente imagen.

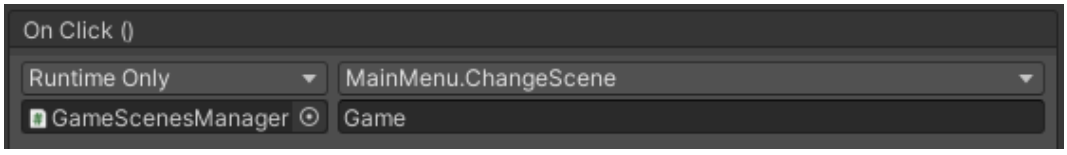


Figura 26. Ventana de acción al hacer click en boton de iniciar partida del Menú Principal del juego del Reloj. Fuente: (Elaboración Propia)

Cuando el usuario hace *click* sobre el botón de *Iniciar Partida*, se ejecuta la función *ChangeScene* del script *MainMenu*, el cual está en el objeto del juego *GameScenesManager*, y se introduce como argumento a la función la variable de texto *(string)* "*Game*".

```
public void ChangeScene(string SceneName)
{
    SceneManager.LoadScene(SceneName);
}
```

Es una función muy sencilla que cambia a la escena con el nombre introducido como argumento.

5.2.2. *Menú Principal de Opciones*

Al pulsar el botón de *Opciones*, el juego nos manda a este panel de edición de parámetros del juego.

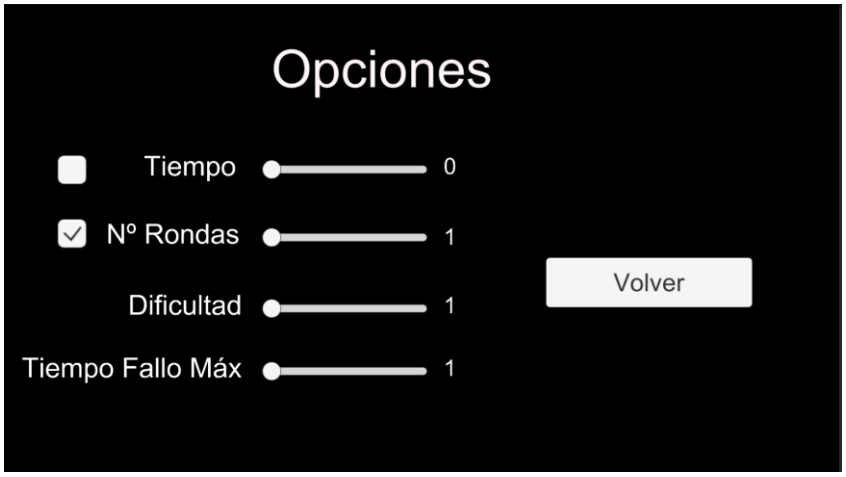


Figura 27. Ventana de Menú Principal de Opciones. Fuente: (Elaboración Propia)

En esta ventana vemos que hay cuatro parámetros editables, El tiempo, el número de rondas, la dificultad y el tiempo máximo de fallo. Todas las variables se controlan con una slider horizontal, y en todas ellas aparece el valor en tiempo real a su derecha.

Vamos a ir en orden descendente:

1- *Tiempo y Rondas*: Las dos variables controlan la duración del ejercicio. El usuario puede elegir con los seleccionadores de la izquierda que parámetro le parece más adecuado. Se ha realizado la programación de manera que en todo momento haya un parámetro seleccionado, pero también pueden estar los dos actuando. En el caso de que los dos parámetros hayan sido elegidos, el que suceda antes finalizará el juego.

   Se han introducido unos rangos de selección de uno a cinco minutos en el caso del tiempo, y de ocho a ochenta en el caso del número de rondas

2- *Dificultad*: La dificultad en este juego se basa en la distancia necesaria que tienen que recorrer el paciente para llegar al objetivo, por lo cual este parámetro modifica el tamaño del reloj.

   Tenemos tres valores posibles a elegir entre *Fácil*, *Medio* y *Difícil*.

3- *Tiempo de Fallo Máximo*: Cuanto el paciente tarda demasiado en llegar a un objetivo, se asume que es incapaz de llegar a dicho punto, por lo cual se asigna un tiempo máximo en el que tiene que alcanzarlo. Si transcurre este tiempo, y el usuario aún no ha alcanzado el punto, se eliminará y aparecerá otro aleatorio automáticamente.

   El rango de actuación en este caso irá de los diez a los treinta segundos de tiempo máximo.

Inicialmente se seleccionarán todos los parámetros en el mínimo valor del rango, y aparecerá seleccionado como duración el número de rondas.

El botón de la parte de la derecha que tiene inscrito *"Volver",* hace la función inversa al botón *Opciones* del panel de *Menú Principal*, desactivando el panel de opciones y activando el de inicio.

5.2.3. *Escena Juego*

Lo primero que aparece al pasar la escena de juego, es la ventana de carga. El servicio de *Python* que gestiona la visualización de la cámara necesita un tiempo de calibración, en el que se realiza la selección de diversos parámetros. Es por esto, que nos ha parecido conveniente introducir una pantalla de carga para que el usuario no se sienta contrariado por el tiempo de espera.

Figura 28. Ventana de carga juego del reloj. Fuente: (Elaboración Propia)

Cuando el servicio de *Python* completa la inicialización del vídeo en *Unity*, se deshabilita esta ventana, y se habilita la ventana de juego y los archivos necesarios para la gestión de la partida.

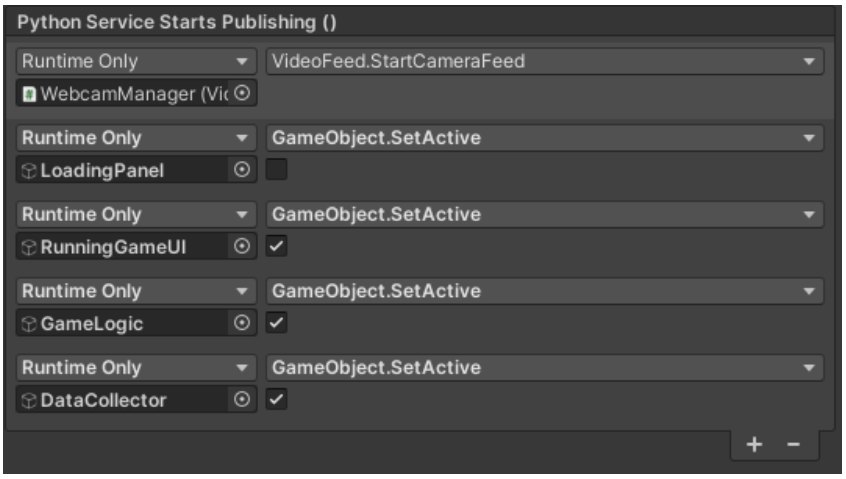


Figura 29. Funciones ejecutadas por el inicio del servicio de Python en juego del reloj. Fuente: (Elaboración Propia)

Lo siguiente que se visualiza tras ejecutar las funciones anteriores, es lo siguiente:

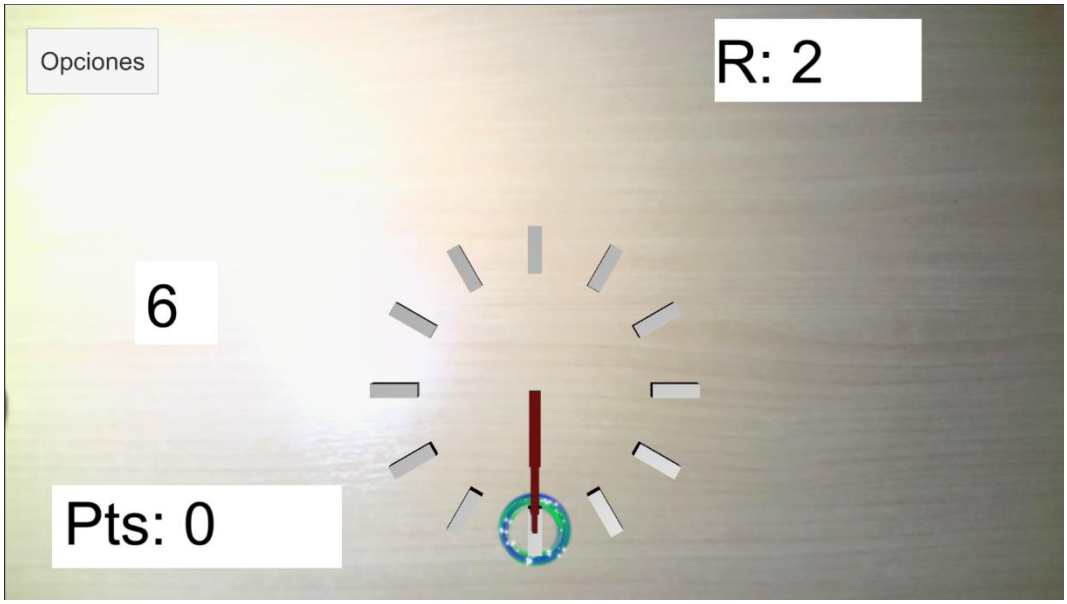


Figura 30. Escena de Juego del Reloj. Fuente: (Elaboración Propia)

Como vemos en la imagen anterior, aparecen varios elementos:

a) Un reloj en el medio de la escena.

b) Una marca circular señalando al objetivo, en este caso el 6.

c) Interfaz gráfica que ofrece información al usuario sobre distintos elementos

Diseño

A) El reloj

El objeto central de esta escena es el reloj “invisible”, denominado así debido a que solo han sido modeladas las agujas y los indicadores de la hora de este.

El reloj está formado por doce elementos en forma de prisma rectangular, indicando el número de hora, y tres agujas de reloj, que rotarán en función de la hora objetivo de la ronda actual.

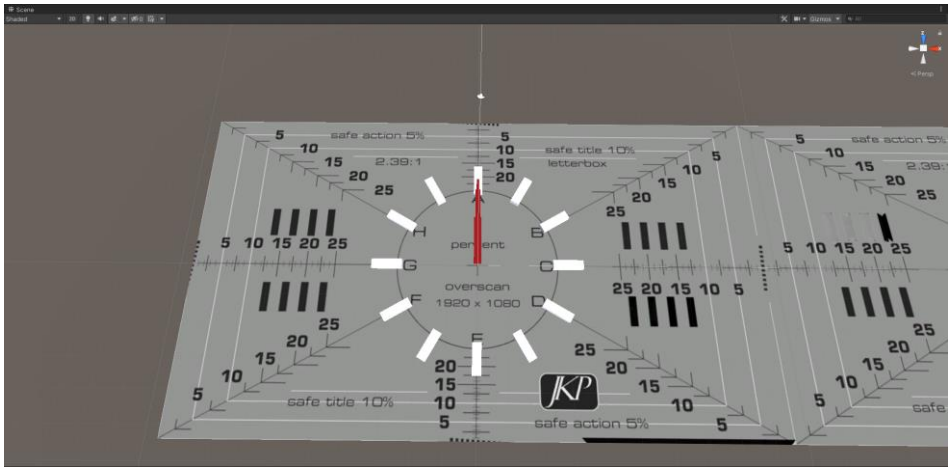


Figura 31. Reloj Invisible visto en la escena. Fuente: (Elaboración Propia)

El reloj se ha realizado con un objeto *(GameObject)* padre vacío, y un objeto hijo para cada prisma. Esto tiene una ventaja de programación, que permite desplazar todo el reloj modificando el padre, y a su vez tener la capacidad de ver y modificar los parámetros individuales de cada hijo.

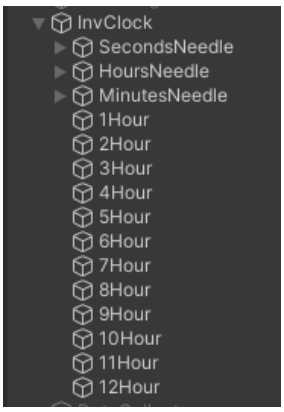


Figura 32. Reloj Invisible visto en el inspector. Fuente: (Elaboración Propia)

B) El objetivo circular

Este objeto es el que marca la meta que tiene que alcanzar el paciente para pasar de ronda. Este objeto ha sido realizado utilizando un sistema de partículas que rota alrededor de un punto. Todo el objeto se convirtió en un objeto prefabricado *(Prefab)* que nos permite generarlo en cualquier momento vía código.

Este sistema de partículas es un componente que simula un fluido de entidades como si fuera un líquido, esto lo realiza generando y animando gran cantidad de imágenes 2D. (Technologies, 2023)

A nivel de diseño se ha realizado una forma en tipo donut, formada por una línea de radio interno que gira en un sentido, una línea de radio externo que

gira en sentido contrario, y dos nubes de puntos entre ellas que giran en direcciones opuestas. Estas partículas varían su tamaño y velocidad en el tiempo, ofreciendo un efecto de fluidez muy interesante.

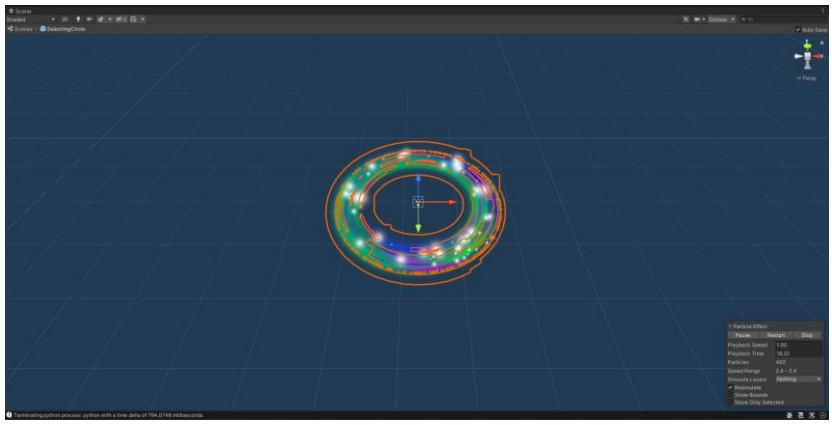


Figura 33. Objetivo circular del juego del Reloj. Fuente: (Elaboración Propia)

Algunos de los parámetros que han sido modificados en el atributo del sistema de partículas ha sido, la duración, el tamaño inicial y final de las partículas, la curva de tamaño a lo largo de su tiempo de vida, la forma de las partículas, el renderizado y el color.

Figura 34. Ventana del sistema de partículas del círculo del juego del Reloj. Fuente: (Elaboración Propia)

C) Interfaz gráfica de información

En esta parte vamos a comentar toda la información gráfica que recibe el jugador, tanto durante el juego, como los menús a los que puede acceder.

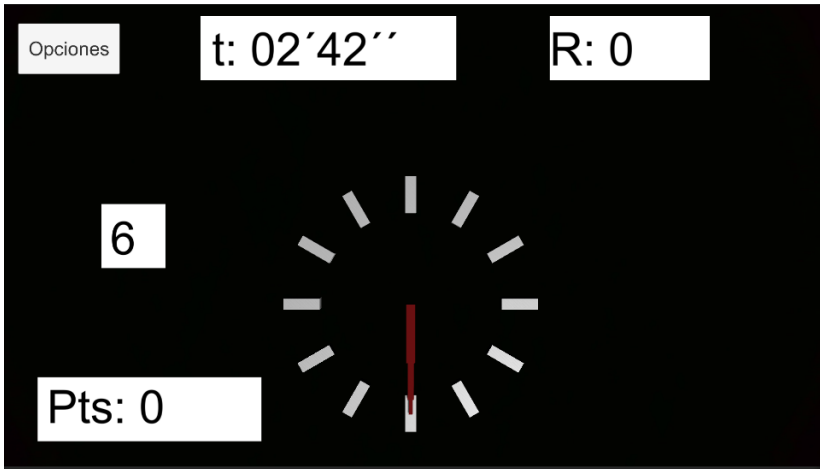


Figura 35. Interfaz gráfica durante el juego del Reloj. Fuente: (Elaboración Propia)

Mientras se está la pantalla de juego aparecen varias zonas que brindan información al usuario:

1- En la parte superior media-derecha de la imagen hay un recuadro que nos permite ver la duración de la partida. En el apartado anterior vimos como existen dos métodos de control de duración, mediante tiempo y mediante rondas.

   En cualquiera de los casos se mostrará la información en el método elegido y, en el caso de que se elijan las dos metodologías, aparecerán dos contadores.

2- En la parte media de la izquierda de la imagen aparece un recuadro con un valor numérico único en su interior. Este valor indica la hora objetivo a la que tiene que ir el usuario, esto permite una señalización triple del objetivo. Al final, la hora esta señalada por las agujas del reloj, por el círculo verde y morado y por el número de la izquierda, permitiendo al paciente comprender bien cuál es el objetivo.

3- En la parte baja de la izquierda de la imagen hay un recuadro que marca *"Pts: X"*, este recuadro indica la puntuación que tiene el usuario en tiempo real. Posteriormente en la programación veremos cómo se ha gestionado el tema de la puntuación.

4- Finalmente, vemos en la parte superior izquierda de la imagen el botón *Opciones*. Este botón se pude pulsar en cualquier momento de la partida. Al pulsar este botón nos aparece una ventana de pausa que nos permite reanudar la partida, iniciar una nueva o salir del juego. Además, al dar al botón de *Opciones*, se pausaría el tiempo de la partida, evitando que sigan apareciendo objetivos, o que el paciente elimine el objetivo mientras se esté en esta nueva ventana.

Figura 36. Ventana de pausa durante el juego del Reloj. Fuente: (Elaboración Propia)

El botón de *Salir del Juego* no necesita mucha explicación, su única función es dar una finalización amigable al programa cerrando la aplicación, con esto evitamos tener que cerrar la ventana manualmente al terminar el juego.

El botón de nueva partida nos envía a una ventana en la que podremos elegir de nuevo los parámetros propios del juego e iniciar una nueva partida, o en su defecto, se ha incluido otro pulsador para poder salir del juego, en caso de que el usuario no lo haya hecho en la ventana de pausa y quisiera hacerlo desde aquí.

Otra ventana que puede aparecer durante el juego que no se ha mencionado es la que surge en el momento en el que se finaliza la partida como consecuencia de que se haya llegado al valor máximo de la duración asignada por el usuario. Esto agrupa tanto al caso de que el paciente haya logrado superar el número de rondas elegido al inicio del juego, o que se haya cumplido el tiempo máximo determinado.

En esta última ventana aparecen las opciones de iniciar una nueva partida y avanzar a la ventana mencionada justo anteriormente, o la de salir del juego.

Figura 37. Ventana de finalización de partida del Juego del Reloj. Fuente: (Elaboración Propia)

Programación

Ya hemos visto todos los elementos gráficos que aparecen en la escena del juego, ahora vamos a analizar los archivos de código que hacen posible la interactuación de estos elementos con el usuario, brindando una experiencia fluida y amigable al paciente.

Cabe destacar que se ha intentado respetar en lo posible las convenciones de buenas prácticas asociadas al lenguaje de programación C# (Microsoft, 2023). El editor de texto utilizado para la edición de código ha sido Visual Studio.

Los ficheros más importantes de los que consta el juego son los siguientes:

- *GameLogic*
- *ClockLogic*
- *DataCollector*
- *CircleSpawner*

Todos estos ficheros actúan simultáneamente sobre la escena del juego, de forma que cada uno interactúa con una funcionalidad de la partida. Se podrían distribuir por la interfaz de la siguiente manera.

Todos estos códigos están íntegros en el ***Anexo 2***.

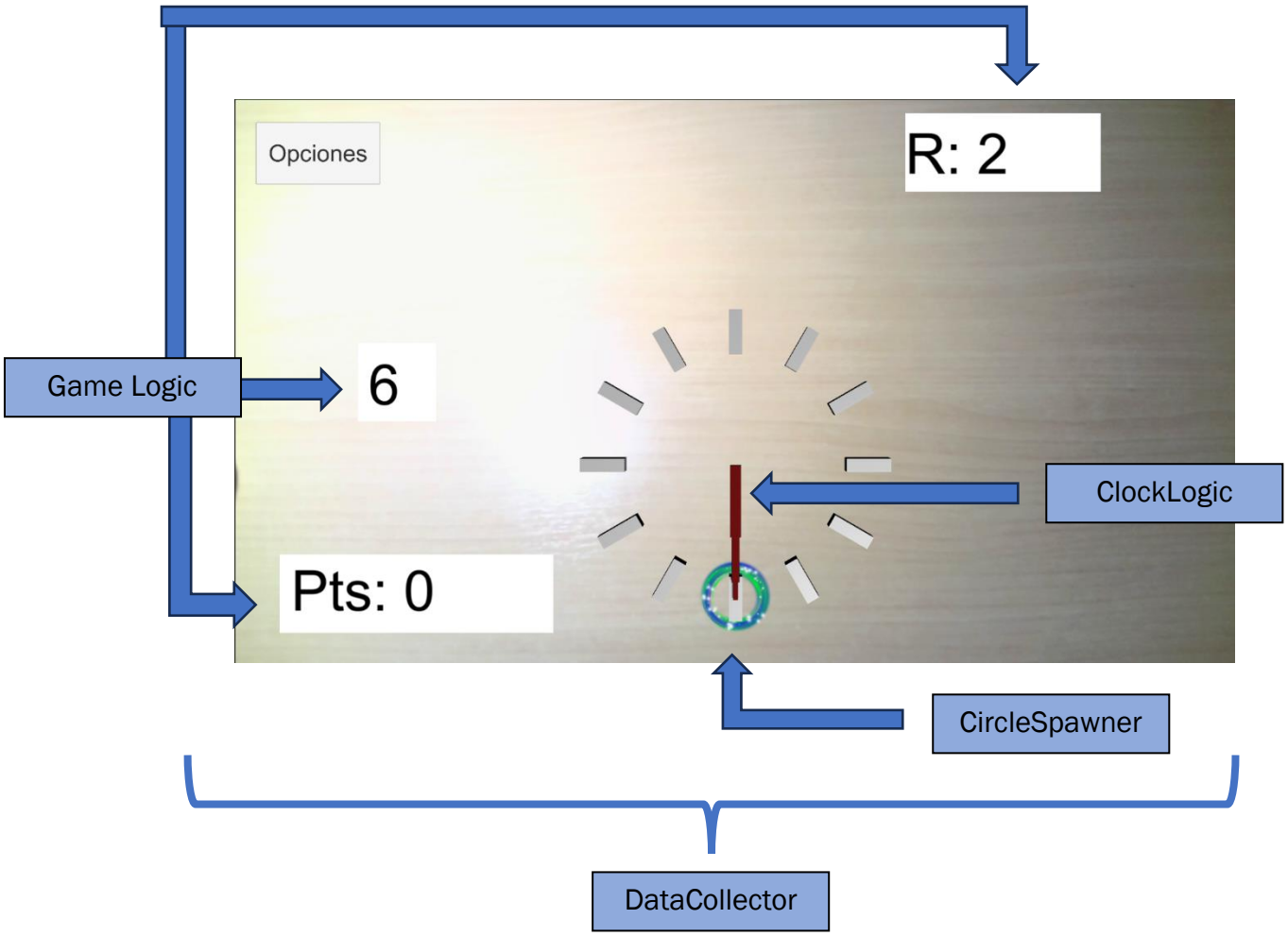


Figura 38. Distribución de funcionalidad de ficheros en juego del reloj. Fuente: (Elaboración Propia)

Antes de empezar explicando lo que hace realmente cada código, y explicar alguna de las funciones más complejas, mencionar que, en *Unity*, existen varios métodos nativos que son llamados en instantes concretos de la partida. Algunos de estos métodos que hemos usado, o son importantes:

- *Start:* Esta función se ejecuta cuando el objeto en el que está se activa, se ejecuta previa al primer *frame* del tiempo de juego.

- *Awake*: Se ejecuta antes de todos los métodos *Start*.

- *OnEnable*: Se ejecuta cada vez que se habilita el objeto en el que está el fichero.

- *Update*: Se ejecuta cada frame del tiempo de juego.

GameLogic: Es el que controla las funciones generales del juego.

Al ser un código tan general, necesitaremos utilizar gran cantidad de variables, tanto públicas, como privadas de la propia clase.

Las variables creadas se dividen entre públicas, privadas, los nombres de las variables *PlayerPrefs* y las variables privadas asociadas a las públicas.

La información utilizada por varios *Scripts* se almacena en *PlayerPrefs*, en esta memoria se almacenan los datos de la partida y puede ser accedida por cualquier fichero del entorno. Esto nos permite el almacenamiento y transmisión de información entre los objetos de una manera más sencilla.

Estas variables tienen un nombre único, que será necesario para acceder o modificar esa información.

Las funciones principales que se realizan en este fichero son las siguientes:

a) Inicializa todas las variables *PlayerPrefs*, comentadas anteriormente, para que, al acceder a ellas en los demás ficheros, estás tengan un valor correctamente asignado.

b) Modificamos la posición del reloj y su escala en función de la dificultad seleccionada en el menú de opciones.

c) Actualizamos la interfaz de usuario del panel. En esta interfaz nos encontramos el número de rondas, el tiempo de duración y la próxima hora a la que dirigirse.

d) Comprobamos si la partida ha terminado, y si ese es el caso, deshabilitaremos el panel de UI y habilitamos el panel de partida finalizada.

Las primeras dos funcionalidades se realizan en el método *OnEnable()*, de manera que cada vez que habilitamos el objeto, iniciando la partida, se inicializan las variables.

Las dos últimas funcionalidades se realizan en el método *Update()*, de manera que en cada *frame* se actualice la visualización de la interfaz gráfica y la finalización del juego.

Este fichero no es demasiado complejo a nivel de codificación, por lo cual no creemos necesaria la explicación de ninguna función en concreto, esto no lo hace menos importante, dado que sin este código la lógica de finalización y la interfaz gráfica del juego no se actualizaría.

ClockLogic: Es el encargado de que las manecillas del reloj señalicen la hora objetivo en tiempo real.

Este código es más sencillo que el anterior. Al igual que antes, utilizaremos los dos métodos nativos de *Unity* para conseguir el objetivo.

En el *Start()* inicializaremos los ángulos de Euler de las tres manecillas a cero, señalando al número doce del reloj.

En el *Update()* comprobaremos el valor de la próxima hora objetivo para el usuario, y operando con ese número conseguimos el valor del ángulo al que rotar las manecilla.

CircleSpawner: Este fichero gestiona todo lo relacionado con los círculos que marcan el objetivo.

Este es el fichero el más difícil de todos los que hemos utilizado, por lo cual en este caso nos detendremos algunas funciones algo más complejas.

Al igual que en los otros dos ficheros, empezamos declarando las variables necesarias para todas las instrucciones a realizar.

En este código, se han utilizado las funciones *Start()* y *Update()* al igual que en los anteriores.

El fichero realiza las siguientes funcionalidades:

a) Obtener las variables de *PlayerPrefs* de tiempo actual, número de errores, número de aciertos, hora objetivo actual..., y guardarlas en variables propias de la clase, con el objetivo de utilizarlas para realizar instrucciones con ellas.

b) Obtener los objetos de la escena. En este fichero utilizaremos información del reloj y de los elementos gráficos que tiene el sistema de partículas de los círculos, por lo cual deberemos encontrar estos objetos en la escena y guardarlos en variables de la clase.

c) Actualizar el tiempo de partida en cada *frame*, para poder así determinar cuándo se ha alcanzado el tiempo máximo de fallo.

d) Realizar el parpadeo del círculo cuando se ha alcanzado la mitad del tiempo máximo de fallo. Esta función tiene la peculiaridad de que no se puede modificar el efecto de renderizado del círculo al completo, por lo cual se debe habilitar o deshabilitar el renderizado de cada sistema de partículas de manera individual.

```
_centreParticles1Way.GetComponent<ParticleSystem>()
    .GetComponent<ParticleSystemRenderer>().enabled =
    !_circleVisible;
_centreParticlesOtherWay.GetComponent<ParticleSystem>()
    .GetComponent<ParticleSystemRenderer>().enabled =
    !_circleVisible;
_innerRadius.GetComponent<ParticleSystem>()
    .GetComponent<ParticleSystemRenderer>().enabled =
    !_circleVisible;
_outterRadius.GetComponent<ParticleSystem>()
    .GetComponent<ParticleSystemRenderer>().enabled =
    !_circleVisible;

_circleVisible = !_circleVisible;
```

Con esto invertimos el estado de renderizado de cada estado de partículas. Esto se realiza según un periodo, a partir del instante en el que se supera la mitad del tiempo máximo de fallo.

e) Cuando se ha llegado el tiempo máximo de fallo, o cuando se ha alcanzado el círculo actual, se genera un nuevo círculo. Esto se realiza en dos zonas diferentes.

   Cuando se ha alcanzado el objetivo se ejecuta la función *GenerateNextCircle(False)* desde el menú de *Unity.*

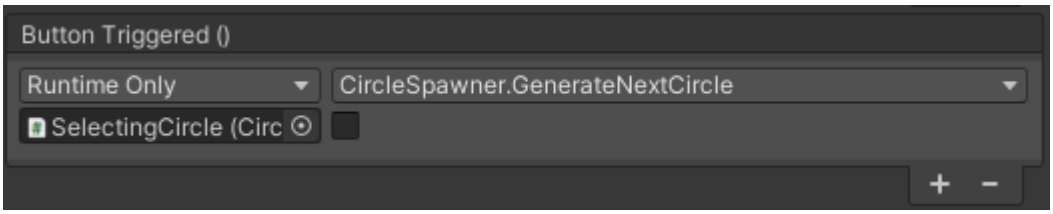


Figura 39. Ejecución de función de generación de círculos cuando se alcanza el objetivo en juego del Reloj. *Fuente: (Elaboración Propia)*

   Cuando se comete un fallo, se ejecuta la función *GenerateNextCircle(True)* desde el archivo de código.

   En esta función hay tres situaciones posibles.

   - La función se ejecuta por cometerse un fallo, por lo cual se debe generar el siguiente círculo en una posición aleatoria del círculo e incrementar el valor de cuenta de fallos.

- La función se ejecuta sin fallo, y el círculo actual se encontraba en el centro del círculo. En este caso se debe generar el siguiente círculo en una posición aleatoria del círculo.

- La función se ejecuta sin fallo, y el círculo actual se encontraba en una posición distinta del centro. En este caso se incrementa la cuenta de aciertos y se debe generar el siguiente círculo en la posición central.

```
Vector3 currentPosition = transform.position;
if(currentPosition == _transformClock.position || error == tr
ue)
{
    if(error == true && currentPosition !=
        _transformClock.position)
    {
        UpdateErrors();
        UpdateMistakes();
    }
    NewCircleInRandomPosition();
}
else
{
    NewCircleInClockCentre();
    UpdateSuccess();
}
if (error == true) { Instantiate(DestroySoundError); }
else { Instantiate(DestroySoundSuccess); }
Destroy(gameObject);
```

Las funcionalidades de inicialización de variables y la obtención de la escena se utilizan en el método *Start()*, las demás funcionalidades se utilizan en el método *Update()*.

DataCollector: Este archivo es capaz de recoger toda la información relevante de la partida e introducirla en un fichero .csv. La información que se va a guardar será la siguiente:

- Fecha del día de la prueba, con el formato “yyyy-MM-dd-HH-mm-ss”
- Tiempo total de la partida
- Rondas totales completadas
- Fallos y aciertos totales
- Puntuación total
- Dificultad
- Posición de la mano en cada instante

Durante el desarrollo de este fichero, surgieron diversas formas de realizar la adquisición y el guardado de datos. Finalmente, el código tiene las siguientes funcionalidades:

a) Inicializar listas de datos de la información necesaria.

```
_positionsList = new List<Vector3>();
_timeList = new List<float>();
_roundList = new List<int>();
_mistakesList = new List<int>();
_successList = new List<int>();
```

b) Cuando ya tenemos las listas inicializadas, procedemos a actualizar el tiempo en cada *frame*, cuando éste supera el periodo de guardado asignado, se procede a guardar el valor de las variables en su correspondiente lista.

Para evitar valores nulos que pueden afectar al posterior tratamiento de datos, solo realizaremos el guardado cuando la posición de la mano no sea nula.

```
if ((!float.IsNaN(handPosition.x)
     || !float.IsNaN(handPosition.y)
     || !float.IsNaN(handPosition.z)))
{
    _positionsList.Add(handPosition);
    Debug.Log("Posicion de la mano: " + handPosition);

    _currentTime = PlayerPrefs.GetFloat(CurrentTimeVariableName,
        0);
    _timeList.Add(_currentTime);

    _rounds = PlayerPrefs.GetInt(CurrentRoundVariableName,
        0);
    _roundList.Add(_rounds);

    _totalSuccess = PlayerPrefs.GetInt(TotalSuccessVariableName,
        0);
    _successList.Add(_totalSuccess);

    _totalMistakes = PlayerPrefs.GetInt(
        TotalMistakesVariableName, 0);
    _mistakesList.Add(_totalMistakes);

    _score = PlayerPrefs.GetInt(ScoreVariableName, 0);
}
```

Para el cálculo de la posición de la mano, se realiza la siguiente función:

```
private Vector3 GetHandMeanPosition()
{
    GameObject[] objectsInLayer = GetObjectsInLayer();
    (Vector3 positionSum, int nPos) =
        GetSummationOfPositions(objectsInLayer);
    _clockOffset = PlayerPrefs.GetFloat(ClockPositionOffset, 0f);


    Vector3 meanPosition = new Vector3(positionSum.x / (float)nPos
        , positionSum.y / (float)nPos
        , (positionSum.z / (float)nPos) - _clockOffset);

    return meanPosition;
}
```

En esta función obtenemos todos los puntos de referencia *(landmark)* que nos proporciona el algoritmo de segmentación. Para obtener el valor medio de posición de la mano, se obtiene la posición individual de cada punto que está detectándose en la mano, y se calcula la media.

Cuando finaliza la partida, se procede a realizar el volcado de datos al fichero *.csv*. Lo primero que se realiza es crear el fichero de datos, este fichero debe tener un nombre único para evitar problemas, por lo que el nombre de la ruta estará compuesto por la fecha de creación del archivo.

Cuando se crea el fichero, se añade a este la primera fila de datos generales.

Date	TotalTime	TotalRounds	TotalMistakes	TotalSuccess	Score	Difficulty
2023-06-30-13-36-00	194,9142	21	0	21	2100	Medium

Después de esta línea, se realiza un bucle iterativo para extraer todos los datos de las listas.

```
foreach (Vector3 position in _positionsList)
{
    string status = "";
    if ((i > 0) && (_mistakesList[i-1] !=
        _mistakesList[i])) { status = "Mistake"; }
    if ((i > 0) && (_successList[i-1] !=
        _successList[i])) { status = "Success"; }
    file.WriteLine($"{_timeList[i]};{_roundList[i]};" +
        $"{position.x};{position.z};" +
        $"{status}");
    i++;
}
```

Para finalizar se cierra el fichero y se inhabilita el objeto. Con esto conseguimos que, al habilitarlo en la siguiente partida, se volverán a inicializar las listas de nuevo.

5.3. Resultados Obtenidos

Con el archivo de datos generado por el fichero *DataCollector*, anteriormente explicado, seremos capaces de visualizar gráficos que permitan analizar la progresión del paciente.

Con los datos obtenidos se podrán calcular manualmente de velocidades parciales, velocidad media o aceleración. Esto queda en manos del especialista. Lo que si que podemos realizar son gráficas de la posición de la mano, las cuales resultan útiles para comprobar la precisión del movimiento del paciente.

Por ello, hemos realizado algunas sesiones de prueba con diferente tiempo, rondas y dificultad para ver estos gráficos.

1)

Date	TotalTime	TotalRounds	TotalMistakes	TotalSuccess	Score	Difficulty
2023-06-30-13-36-00	194,9142	21	0	21	2100	Medium

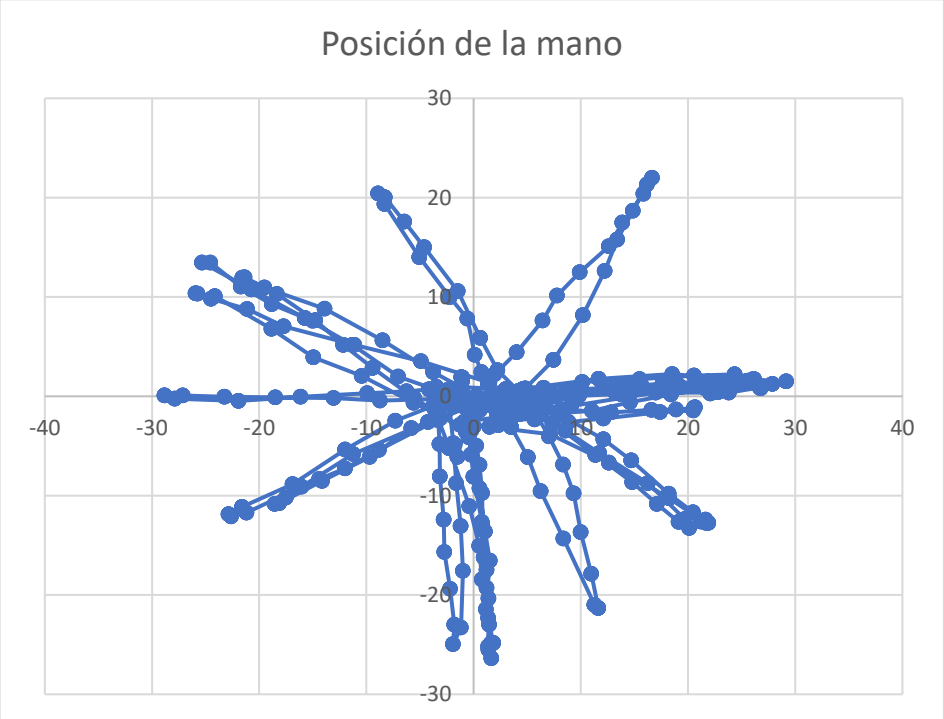


Figura 40. Gráfica 1 de posición de la mano en juego del reloj. Fuente: (Elaboración Propia)

Como vemos en esta prueba, se ha utilizado una dificultad media, por lo que se llega aproximadamente a ±30 de distancia máxima en el eje X. Vemos una gran concentración de puntos en el centro de la gráfica, dado que en cada ronda se debe volver al punto medio. Al no se un paciente real, vemos que el movimiento se realiza de forma precisa.

2)

Date	TotalTime	TotalRounds	TotalMistakes	TotalSuccess	Score	Difficulty
2023-06-30-13-24-46	194,9021	20	0	20	2000	Difficult

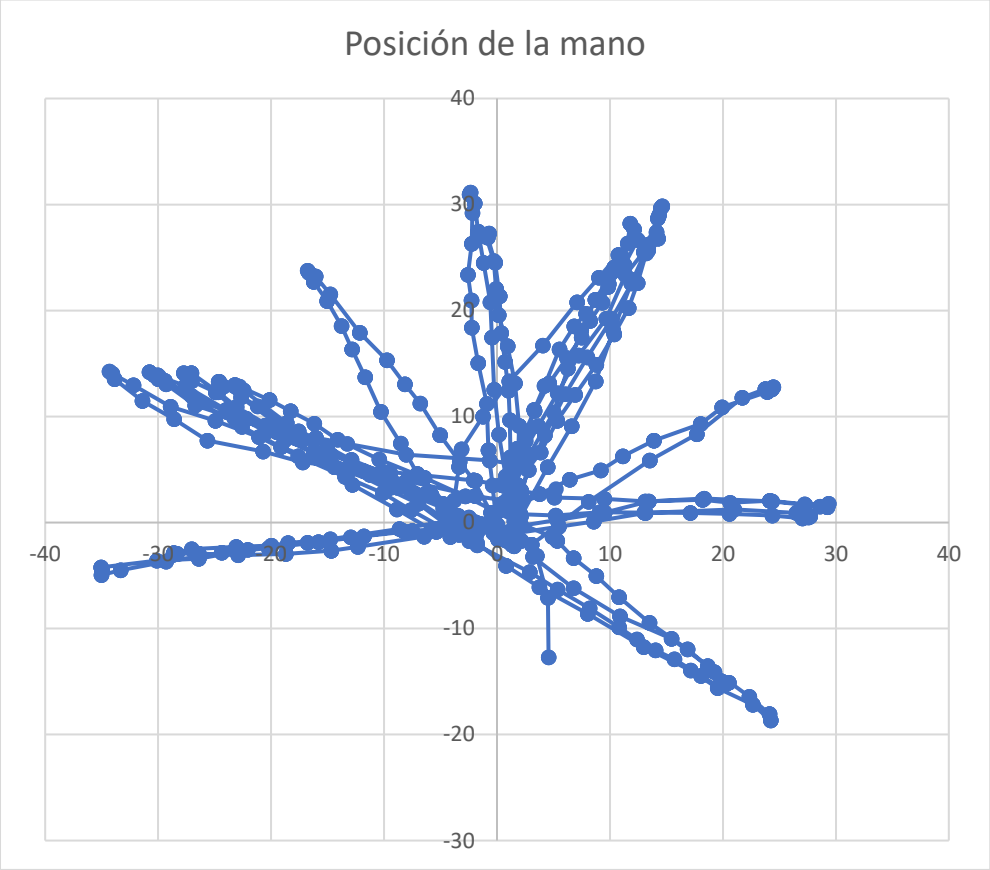


Figura 41. Gráfica 2 de posición de la mano en juego del reloj. Fuente: (Elaboración Propia)

En esta partida se utilizó la máxima dificultad, y se puede ver en la parte de la izquierda de la gráfica cómo aparecen valores más alejados de -30 en el eje X. Otra consideración que se podría hacer viendo la gráfica, es que las horas más transitadas han sido la 2 y la 10, dado que son en las que más acumulación de puntos hay. También podemos ver como las horas 6, 7 y 8 no se han utilizado.

3)

Date	TotalTime	TotalRounds	TotalMistakes	TotalSuccess	Score	Difficulty
2023-06-30-13-02-43	77,2506	7	0	7	700	Easy

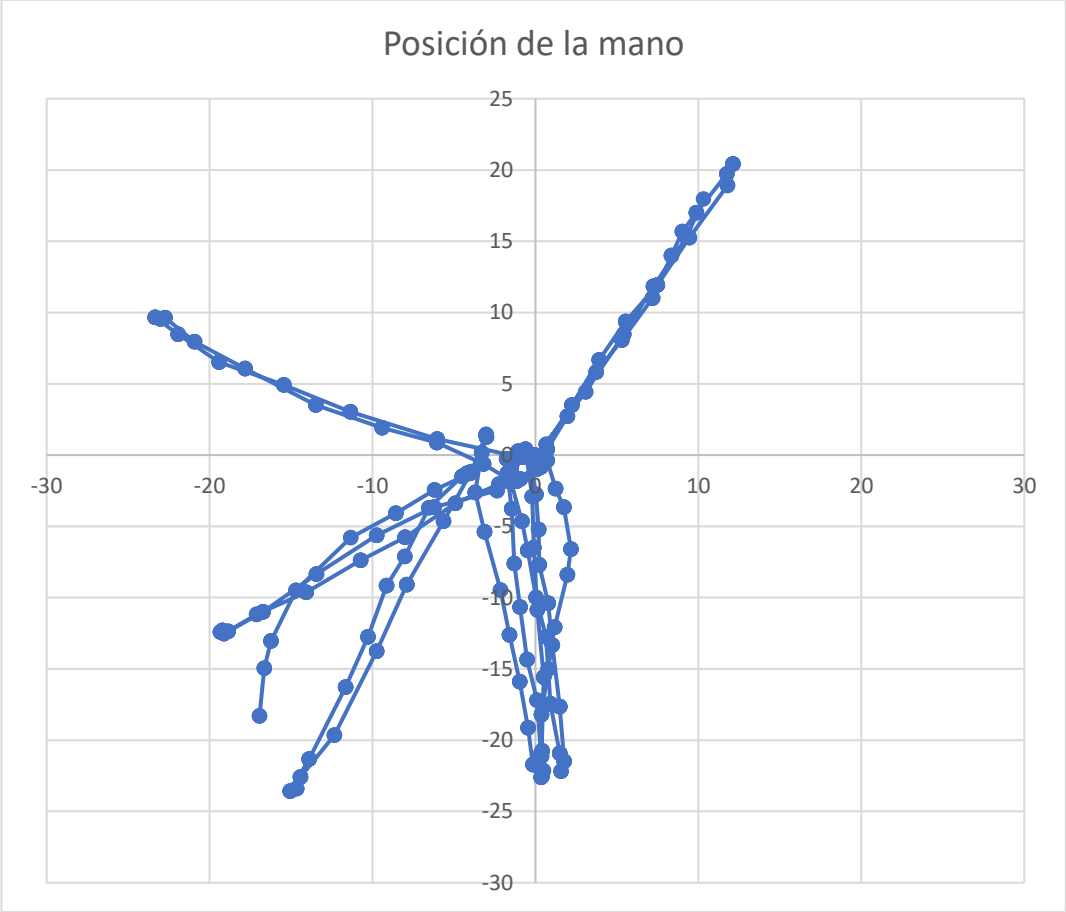


Figura 42. Gráfica 3 de posición de la mano en juego del reloj. Fuente: (Elaboración Propia)

En esta prueba final, se utiliza la menor dificultad de todas. Podemos ver como las distancias alcanzadas son menores. En este caso se puede ver como la hora más concurrida ha sido la hora 6, y al ser la prueba más corta de todas, la mayoría de los objetivos del reloj están vacíos.

6. JUEGO DEL RATÓN

6.1. Idea inicial

El juego del ratón ha sido pensado para realizar una rehabilitación en el que el paciente tenga un papel más activo en cuanto a las decisiones. En el juego del reloj el usuario tiene unas instrucciones muy claras que tenía que seguir para completar la partida, en cambio en este juego la única indicación que tiene es la de alcanzar el objetivo.

Al comenzar el juego, aparecería, en una zona media de la pantalla, el objetivo que debe alcanzar el paciente con la mano.

Este juego está formado por dos zonas diferenciadas, la zona de menú principal en la cual se modifica la configuración de la partida y se inicia desde ahí, y la ventana juego en la que se desarrolla la partida, y se finaliza esta.

6.2. Desarrollo

6.2.1. Ventana del Menú Principal

Esta es la primera ventana que aparece en el juego, en esta ventana se modifica la dificultad de la partida.

Figura 43. Ventana del Menú Principal del Juego del Ratón. Fuente: (Elaboración Propia)

Como hemos mencionado anteriormente, en esta ventana se modifica la dificultad del juego, la cual viene dada por el espacio disponible de huida del ratón. Se ha limitado la zona jugable de la parte inferior de la pantalla, debido a que en esa zona puede no obtenerse la mano completa, haciendo difícil la segmentación.

Figura 44. Ventana de Menú Principal del Juego del Ratón con dificultad aumentada. Fuente: (Elaboración Propia)

Si modificamos el valor de la slider, aumentará la superficie ocupada por la zona rojiza de la pantalla. Esta zona marca los límites jugables en los que podrá moverse el ratón.

Además de esto, en esta ventana tenemos dos botones interactivos que sacan al usuario del juego, y empieza la partida con la dificultad seleccionada en ese momento, respectivamente. Al pulsar el botón *Empezar Partida*, pasaremos a la siguiente ventana.

A nivel de programación, esta ventana es bastante sencilla, dado que solo está formada por la selección objetos que se activan y desactivan con la pulsación del botón, y modificar la variable del espacio al deslizar la slider.

6.2.2. Ventana de Juego

Esta ventana aparece al pulsar el botón *Empezar Partida*, en esta ventana transcurre todo el proceso del juego, y está formado por tres objetos de trabajo:

1- *El ratón:* El objeto objetivo, huye del usuario y tiene una apariencia de ratón.

2- *El Usuario:* Objeto del usuario, se localiza donde se encuentra la mano.

3- *Interfaz gráfica:* Botón de pausa

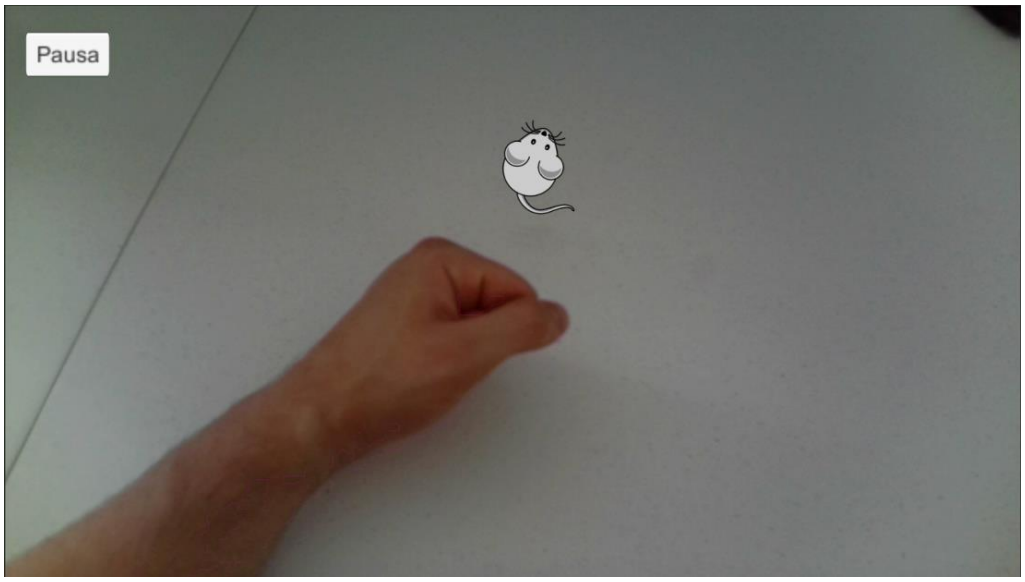


Figura 45. Ventana de Juego del Ratón. Fuente: (Elaboración Propia)

Diseño

Ratón: Este objeto del juego *(GameObject)* está formado por una secuencia de jerarquías de tres objetos.

El objeto de menor grado de poder se corresponde con la imagen del ratón.

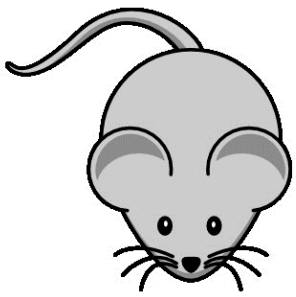


Figura 46. Imagen de ratón usado como personaje. Fuente: (Elaboración Propia)

Durante la huida del ratón, se realizan dos movimientos, uno de translación en dirección de huida y un movimiento de rotación para que la cara del ratón mire hacia la dirección de movimiento.

El uso de las transformaciones geométricas de rotación y translación se deberán hacer a objetos diferentes, la rotación al objeto hijo y la translación al objeto padre, de manera que la rotación no afecte a los ejes antes de realizar la translación.

Usuario: Para el usuario, hemos usado un objeto invisible que actualiza su posición en cada *frame* con la posición media de la mano. Se usa este objeto para que haya un objeto único en la escena marcando la posición de la mano, y que este punto sea el mismo en toda la partida. Se ha planteado también que el ratón huyera del punto del mano más cercano, pero esto haría que aparecieran conflictos en el punto de huida.

Interfaz gráfica: En la ventana de juego aparece un botón en la parte superior izquierda que contiene el texto *Pausa*, esto botón, como su propio nombre indica, permite al usuario pausar el tiempo de partida y activa la ventana de pausa.

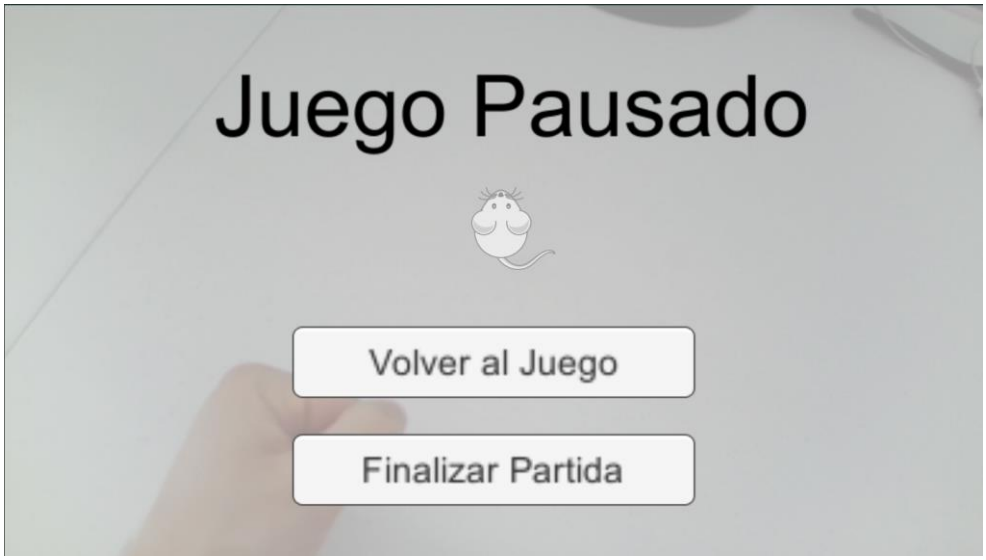


Figura 47. Ventana de Juego Pausado del Juego del Ratón. Fuente: (Elaboración Propia)

En la ventana de pausa el tiempo de proceso de Unity está parado, por lo cual no se actualiza la posición ni del ratón, ni del objeto que localiza la mano. En esta ventana el usuario puede elegir entre volver al juego de nuevo o, por el contrario, finaliza partida. Al finalizar la partida el usuario sería enviado a la ventana del menú principal.

Programación

Este juego tiene tres ficheros imprescindibles para el correcto funcionamiento del juego, cada uno se encarga del control de un sector del juego, que se podría representar con el siguiente esquema.

Los ficheros íntegros se encuentran en el ***Anexo 4.***

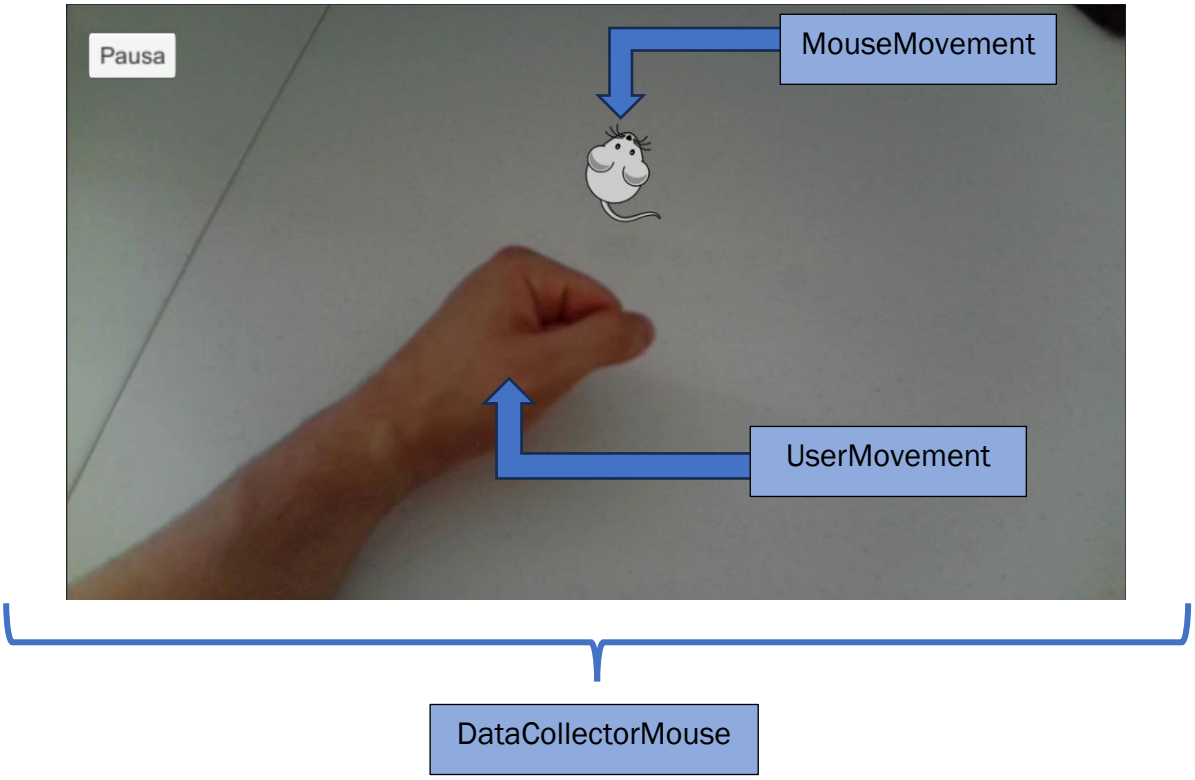


Figura 48. Distribución de funcionalidad de ficheros en juego del ratón. Fuente: (Elaboración Propia)

UserMovement: Este fichero se encarga de localizar los puntos de la mano, y colocar al objeto en el que está en una posición media. Hemos reutilizado gran parte del código del juego del Reloj, dado que la principal función de este fichero es la detección de la posición de la mano, y eso ya se hacía en el fichero *DataCollector*. Lo que se realiza en el *Update()* del fichero es lo siguiente.

```
Vector3 handPosition = GetHandMeanPosition();
if((!float.IsNaN(handPosition.x) ||
    !float.IsNaN(handPosition.y) ||
    !float.IsNaN(handPosition.z)))
{
    transform.position = new Vector3(
        handPosition.x, 5, handPosition.z);
}
```

Se obtiene el punto de la mano, y se actualiza la posición del objeto al vector calculado. Se utiliza una altura de cinco para mantenerlo ligeramente por encima del plano de visualización de la cámara.

MouseMovement: Este archivo es el encargado de todo el movimiento del ratón. En este fichero se elige la dirección de huida del ratón en cada instante.

En este caso, al ser un algoritmo algo más complejo que el anterior, necesitaremos más variables propias de la clase para mantenerlo controlado.

Esta clase utiliza las funciones nativas de Unity *OnEnable()* y *Update().*

OnEnable(): En esta función buscamos al objeto que sigue al usuario para poder guardarlo en una variable de la clase e inicializamos las demás variables.

Update(): En esta segunda función controlamos el movimiento del ratón, ya sea de huida o de tentación al usuario. Para ello calculamos la distancia entre el objeto que huye y el que persigue, y comprobamos si se supera o no un umbral.

```
_distanceToOtherObject = DistanceCalc();
PlayerPrefs.SetFloat(
    _distanceToUserVariableName, _distanceToOtherObject);
if (_distanceToOtherObject < _distanceThreshold)
{
    _timer = 0f;
    Escape();
}
else
{
    _timer += Time.deltaTime;
    TeaseUser();
}
```

En el caso de que haya que huir, se utiliza la función *Escape()*, la cual utiliza un algoritmo de huida que cuenta con tres casuísticas:

1- El usuario se acerca al ratón más de lo que permite el umbral, y se encuentra en una zona de no peligro, con espacio para escapar. En este caso, el ratón escapa en dirección contraria a la distancia con el usuario.

2- El usuario se acerca al ratón, y este se encuentra cerca del borde, por lo que no puede alejarse más. En este caso el ratón comprueba con un periodo de muestreo de dos grados, todas las direcciones a su alrededor. El ratón elige la dirección en la cual se situaría en una posición más alejada del paciente, pero manteniéndose en la zona permitida.

3- El usuario se acerca al ratón, y este se encuentra arrinconado. En este caso el ratón optará por un movimiento paralelo a alguno de los lados del paralelogramo que marca la zona, seleccionando la mejor opción. Este movimiento se seguirá hasta que el ratón haya salido de la zona crítica del rincón.

DataCollectorMouse: Este fichero es el encargado de almacenar la información relevante durante la partida. Para esta clase se ha utilizado como base el archivo de datos del juego del Reloj, pero se le ha hecho una readaptación para este caso.

En este caso, se ha reducido bastante el número de variables en la clase, esto es debido a que este juego recopila menos información. Durante la partida, la única información útil que hemos elegido es, la posición de la mano en cada instante, y su distancia al ratón.

Otra diferencia importante frente al recopilador de datos anterior es que en este no hay un límite de finalización de la partida, por lo cual, no sabemos cuántos elementos se van a introducir. En el anterior juego se utilizaban listas para guardar los datos, y al terminar la partida se realizaba un guardado en el archivo *.csv*.

El tamaño máximo de una lista de tipo *Vector3* de C# es muy grande, pero, aunque no lleguemos a llenar la lista, si aumenta mucho su nivel de llenado podría afectar al rendimiento del programa.

Para evitar este problema, hemos decidido abrir de inicio el archivo *.csv* e ir introduciendo los valores según se van calculando. Cuando la partida finaliza, se cierra el archivo de datos para evitar problemas de apertura.

Este fichero utiliza los métodos nativos de Unity *OnEnable() y Update()*

OnEnable(): Al activar el objeto inicializamos las variables y abrimos el fichero *.csv* escribiendo la primera línea de índice.

```
private void OnEnable()
{
    _currentTime = 0f;
    _time = 0f;
    CreateAndOpenCSV();
}
```

```
private void CreateAndOpenCSV()
{
    string folder = Path.Combine(
        Application.dataPath, "DataCollectionMouse");

    if (!Directory.Exists(folder))
    {
        Directory.CreateDirectory(folder);
    }
    string actualDate = DateTime.Now.ToString(
        "yyyy-MM-dd-HH-mm-ss");
    string fileName = actualDate + ".csv";
    _filePath = Path.Combine(folder, fileName);

    bool fileExist = File.Exists(_filePath);

    _file = new StreamWriter(_filePath, true);

    if (!fileExist)
    {
        _file.WriteLine($"Date;{actualDate}");
        _file.WriteLine($"Time;HandX;HandZ;DistanceToUser")
;
    }
}
```

Update(): En esta segunda función vamos a realizar el guardado de las variables.

```
private void SaveOnCSV()
{
    Vector3 handPosition = GetHandMeanPosition();
    float distanceToUser = PlayerPrefs.GetFloat(
        _distanceToUserVariableName, 0f);

    if ((!float.IsNaN(handPosition.x) ||
        !float.IsNaN(handPosition.y) ||
        !float.IsNaN(handPosition.z)))
    {
        _file.WriteLine($"{_currentTime};{handPosition.x};" +
            $"{handPosition.z};{distanceToUser}");
    }
}
```

Se utiliza la función de obtención de la posición de la mano utilizada en *UserMovement*, y se obtiene la distancia al usuario guardada por *MouseMovement*. Estas dos variables se guardan en el archivo de datos, salvo que no se detecte una posición de la mano no nula.

6.3. Resultados Obtenidos

Con el archivo de datos generado por el fichero *DataCollectorMouse*, anteriormente explicado, seremos capaces de visualizar gráficos que permitan analizar la progresión del paciente.

En este caso hay dos gráficas que se pueden sacar:

- Distancia de la mano al ratón a lo largo del tiempo: Esta gráfica puede ser útil para comprender la capacidad del paciente para alcanzar el objetivo

- Distancia a lo largo del tiempo: A diferencia del anterior juego, en el cual considerábamos esta gráfica como la más importante, en este caso cuando la partida dura demasiado tiempo, se pierde la perspectiva de la información. No queda claro en qué momento o dirección se desplaza la mano y en que instante lo hace

  En este caso sería más lógico analizar los datos de manera aislada, o hacer gráficos parciales de momentos concretos. De todas formas, vamos a mostrar ambos gráficos.

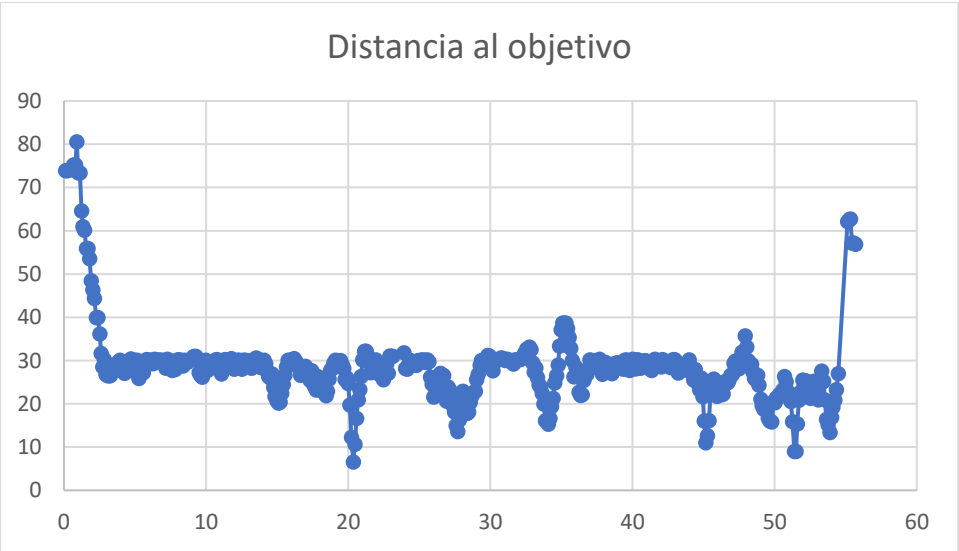


Figura 49. Gráfica de distancia respecto al objetivo a lo largo del tiempo en juego del ratón. Fuente: (Elaboración Propia)

En la gráfica anterior se puede ver como la mano se mantiene a una distancia que fluctúa en torno a 30, siendo esta la distancia umbral utilizada por el ratón para huir.

Además, vemos como aparecen mínimos por debajo de este umbral, esto es debido que, al llegar a las esquinas, el ratón se ve obligado a acercarse, para posteriormente alejarse del usuario. Algo positivo es que vemos que el objeto usuario en ningún momento alcanza al ratón.

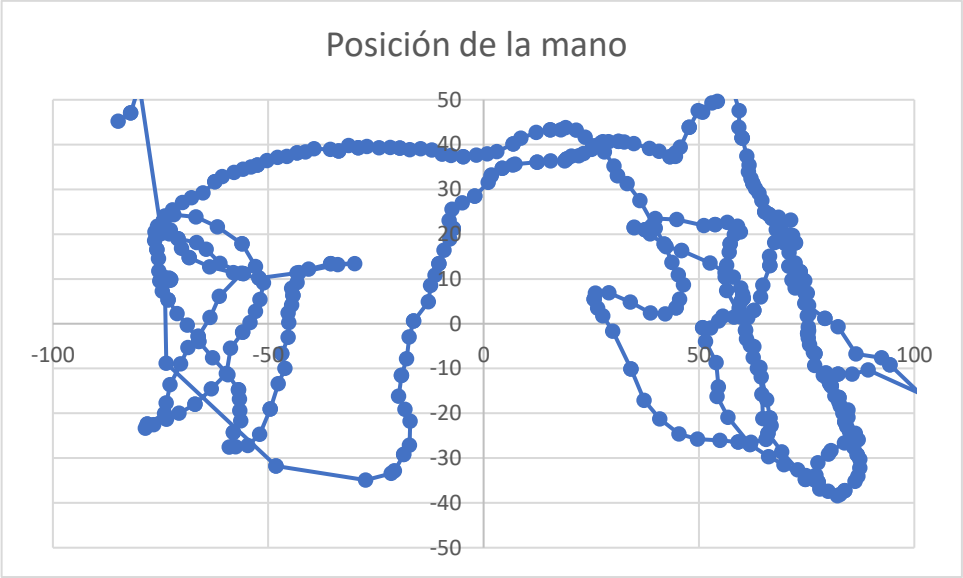


Figura 50. Gráfica de posición de la mano en juego del ratón. Fuente: (Elaboración Propia)

En esta gráfica vemos que el usuario se ha desplazado por toda la pantalla persiguiendo al objetivo. El movimiento parece suave y preciso.

7. CONCLUSIONES

En este trabajo de fin de grado hemos abordado como tema central, el desarrollo de videojuegos con el uso de la realidad aumentada para la rehabilitación del miembro superior. Durante el proceso, hemos encontrado obstáculos y limitaciones, pero finalmente se ha conseguido alcanzar los objetivos y requerimientos solicitados por el usuario final, tanto en efectividad como en experiencia de usuario.

Se han desarrollado dos aplicaciones o juegos, planteados con filosofías diferentes. En primer lugar, se ha implementado un entorno de realidad virtual en el que el paciente debe seguir unos objetivos marcados sobre las horas de un reloj analógico. El fisioterapeuta tiene la capacidad de adaptar la partida a las necesidades del paciente mediante la configuración de parámetros tales como la amplitud del reloj, la duración de la partida o la velocidad de aparición de nuevos objetivos.

En segundo lugar, se ha realizado una aplicación en la cual el paciente posee una mayor libertad. En este segundo juego, el paciente tendrá como meta el de alcanzar un objetivo que se mueve para escapar de él. Para mantener la línea de parametrización de la aplicación anterior, en este juego el fisioterapeuta tendrá la libertad de finalizar la partida cuando lo crea conveniente, y de modificar el espacio habilitado para el movimiento.

Ambos programas están habilitados con la capacidad de adquirir datos de la movilidad del usuario durante las pruebas. Esto permite al personal sanitario realizar gráficos interesantes para analizar el progreso de los pacientes.

Este proyecto ha permitido demostrar que la realidad aumentada es una tecnología viable en el campo de la rehabilitación. El uso de videojuegos con esta tecnología como añadido, permite al usuario tener una experiencia inmersiva única y motivadora, haciendo el proceso de rehabilitación más ameno. Además, se ha desarrollado pensando en que el paciente utilice movimientos comunes en el día a día con el objetivo de mejorar su recuperación.

Con la parametrización de los juegos, más en concreto en el juego del reloj, hemos conseguido una individualización de la terapia que se adecúa a cada paciente y las necesidades de su tratamiento. Esto permite al terapeuta diseñar terapias personalizadas y ajustadas a cada momento.

Las aplicaciones anteriormente mencionadas han sido implantadas para su evaluación en un centro de rehabilitación funcional, pero en el momento de finalización de este TFG, aún no se dispone de datos para valorar el efecto de las aplicaciones en la mejora de los pacientes.

Este proyecto no solo ha sido satisfactorio por la parte del resultado final y de los objetivos específicos, si no que ha sido útil a nivel educativo personal. En este trabajo fin de grado he podido adquirir nuevas habilidades, tanto blandas como duras, y mejorar las que ya había obtenido durante el transcurso del grado universitario. Cabe destacar la importancia de asignaturas como *Visión Artificial* o *Informática Industrial*, que me han permitido comenzar con una base de conocimientos de programación y análisis de código bastante amplia.

Para concluir, hay que mencionar que este trabajo ha demostrado el gran potencial que pueden tener los juegos en realidad aumentada como medio para la rehabilitación del miembro superior. Sin embargo, este sector tiene un amplio margen de recorrido por delante para mejorar.

7.1. Líneas futuras de trabajo y mejoras

Este proyecto no se ha realizado con la intención de que se cierre la línea de trabajo al terminar. Este trabajo puede servir como base de trabajo para futuros proyectos similares, dado que es complicado comenzar un desarrollo como este sin una base de la que partir.

Este entorno se podría considerar como una biblioteca de videojuegos, ampliable con otros juegos que contemplen otros movimientos útiles para otro tipo de terapias.

A su vez, existe mucho trabajo de mejora de los juegos ya realizados. En el futuro sería interesante mejorar el diseño gráfico de las aplicaciones con nuevos *sprites*, fuentes, formas o colores, efectos de sonido diferentes o una interfaz gráfica más atractiva. Todo esto haría que el usuario tenga una experiencia de juego más inmersiva.

A su vez, sería interesante realizar un estudio clínico más riguroso sobre el impacto que tiene esta tecnología en las terapias de rehabilitación, analizando la mejora que tienen los pacientes al utilizar terapias convencionales y mixtas.

Otra línea de trabajo podría ser incorporar este proyecto con otros del sector. Una idea podría ser implementar estos juegos con el exoesqueleto *RobHand*, que permite al paciente realizar terapias activas. Este exoesqueleto ha sido investigado por ITAP y parece tener una línea de progreso muy buena.

8. BIBLIOGRAFÍA

Andrade, C. (2020). *Mean Difference, Standardized Mean Difference (SMD), and Their Use in Meta-Analysis: As Simple as It Gets.* The Journal of clinical psychiatry.

Carmigniani, J., & Furht, B. (2011). Augmented Reality: An Overview. En *Handbook of augmented reality* (págs. 3-46).

Chong, Y., Sethi, D. K., Loh, C. H., & Lateef, F. (2018). Going Forward with Pokemon Go. *Journal of emergencies, trauma, and shock*.

Dorward, L. J., Mittermeier, J. C., Sandbrook, C., & Spooner, F. (2016). Pokemon Go: Benefits, Costs, and Lessons for the Conservation. *Conservation Letters*, 161.

Fernández, R. (6 de Julio de 2023). *Statista*. Obtenido de Statista Web site: https://es.statista.com/estadisticas/1067588/valor-del-mercado-de-la-ra-y-la-rv-en-el-sector-sanitario-por-region/

Ha, H.-G., & Hong, J. (2016). *Augmented Reality in Medicine.* Daegu, South Korea: Hanyang Med Rev.

Haas, J. (2014). *A History of the Unity Game Engine.* Worcester Polytechnic Institute.

INE. (2019). *Encuesta sobre Equipamiento y Uso de Tecnologías de Información y.* Instituto Nacional de Estadística. Recuperado el Junio de 2023, de https://ine.es/prensa/tich_2019.pdf

itap. (29 de Junio de 2023). Obtenido de https://www.itap.uva.es/

Lam Phan, H., Huong Le, T., Min Lim, J., Ho Hwang, C., & Koo, K.-i. (2022). *Effectiveness of Augmented Reality in Stroke Rehabilitation: A.* Applied Science.

León Yacelga, A. R., Acosta Espinoza, J. L., & Díaz Vásquez, R. A. (2021). Aplicación de la metodología incremental en el desarrollo de sistemas de información. *Revista Universidad y Sociedad*.

Mekni, M., & Lemieux, A. (s.f.). *Augmented Reality: Applications, Challenges and Future Trends.* Minesota, Quebec.

Microsoft. (20 de Junio de 2023). *Microsoft Learn*. Obtenido de https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/coding-style/coding-conventions

Moore, K. L., Dalley, A. F., & Agur, A. M. (1992). *Upper limb.* Clinically oriented anatomy.

Moreno, G. (6 de Julio de 2023). *Statista*. Obtenido de Statista: https://es.statista.com/grafico/8697/el-negocio-de-la-realidad-aumentada/

Nordin, N., Quan Xie, S., & Wünsche, B. (2014). *Assessment of movement quality in.* Journal of Neuroengineering and Rehabilitation.

Sánchez Brizuela, G. (2021). *Documentación M3Display - Toolkit de desarrollo.* Valladolid, España.

Sánchez Brizuela, G. (29 de Junio de 2023). *Guillesanbri*. Obtenido de https://guillesanbri.com/

Technologies, U. (22 de Junio de 2023). *Unity Documentation*. Obtenido de https://docs.unity3d.com/Manual/AROverview.html

Tyni, H., Kultima, A., & Mäyrä, F. (2013). Dimensions of Hybrid in Playful Products. *Proceedings of International Conference on Making Sense of Converging Media*, 237-244.

Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C.-L., & Grundmann, M. (2020). *MediaPipe Hands: On-device Real-time Hand Tracking.* Mountain View, California: Google Research.

ANEXO 1: Manual de Usuario Juego del Reloj

1- ¿Dónde se configurar los parámetros de la partida?

Al iniciar el juego nos aparece la pantalla de inicio, y pulsamos en el botón *Opciones*, si se pulsa este botón, nos aparece la ventana de configuración de partida.

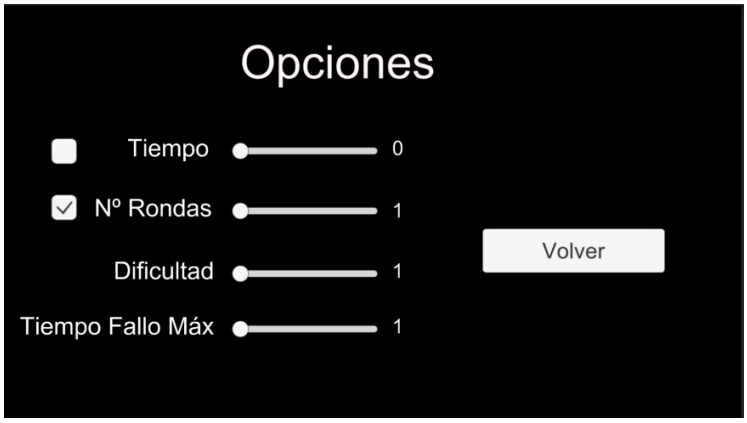
2- ¿Qué significa cada parámetro?

En la pantalla para la configuración de la partida, nos aparecen cuatro parámetros modificables:

4- *Tiempo y Rondas*: Los dos primeros parámetros configuran la duración de la partida. Se pueden seleccionar con los selectores de su izquierda, o simplemente modificando la barra. En el caso de que los dos estén seleccionados, finalizará la partida cuando se agote el primero.

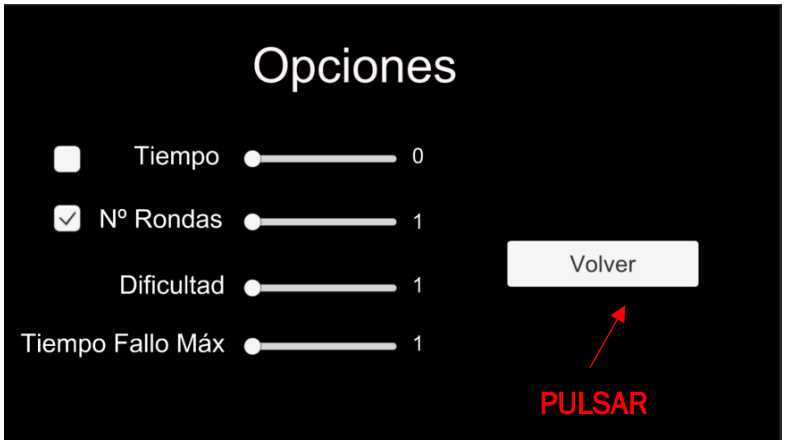
5- *Dificultad*: La dificultad de este juego viene dada por el tamaño del reloj. Tenemos tres valores posibles a elegir entre *Fácil*, *Medio* y *Difícil*.

6- *Tiempo de Fallo Máximo*: Es el tiempo máximo que debe tardar el paciente en alcanzar el objetivo, si llega la mitad de ese tiempo, el objetivo empezará a parpadear, si se llega al tiempo total, el objetivo desaparecerá y aparecerá uno nuevo en otra localización.



3- ¿Cómo se inicia la partida?

Cuando ya hemos modificado los parámetros a nuestro gusto, pulsamos el botón *Volver* de la pantalla de *Opciones*.



Esta acción nos enviará a la ventana inicial del juego, donde deberemos pulsa el botón *Iniciar Partida*.

La partida se iniciará con la última configuración que hayamos dejado.

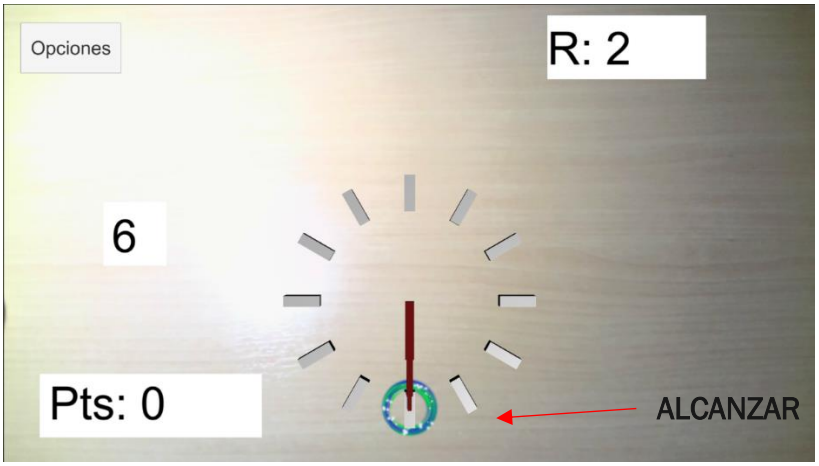
4- ¿Qué pasa si inicio la partida sin haber configurado los parámetros?

Si no se modifican los parámetros, estos tomarán el valor mínimo posible:

- Duración de 8 rondas
- Dificultad Fácil *(Reloj Pequeño)*
- Tiempo máximo de fallo de 10s

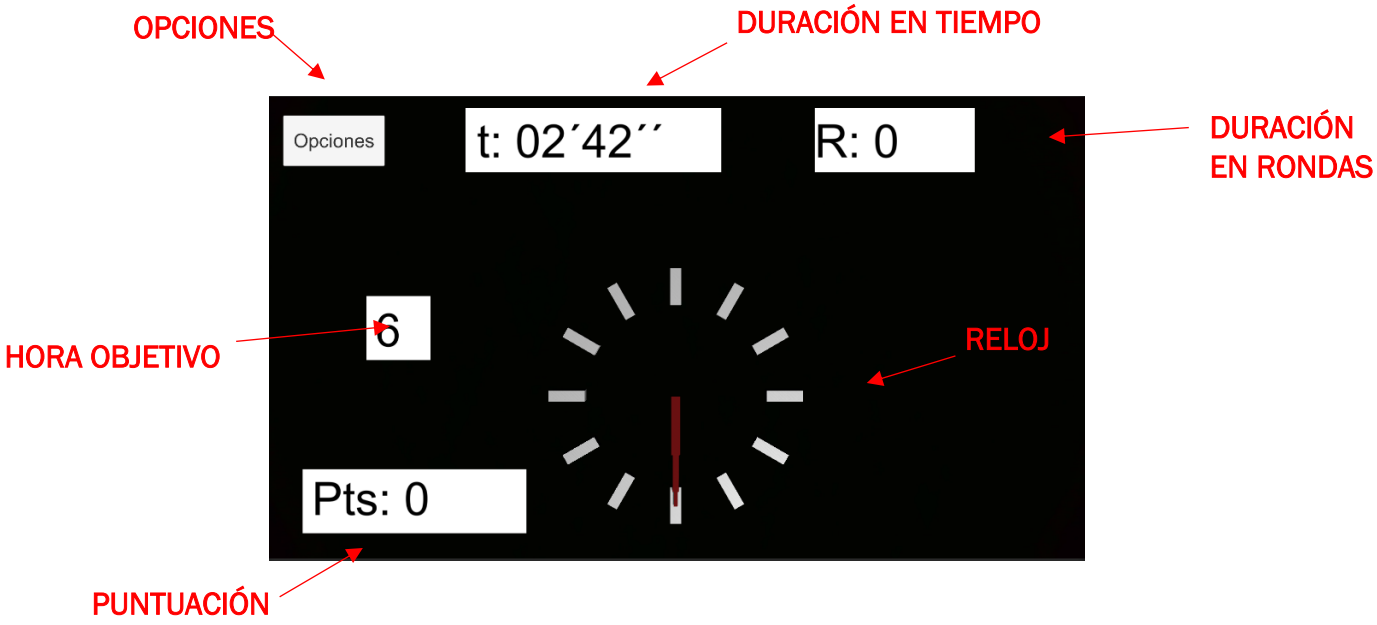
5- ¿Cuál es el objetivo de la partida?

El objetivo de la partida es que el paciente alcance la mayor cantidad de objetivos circulares que pueda con su mano. Cuando el usuario alcance un objetivo en una posición de las horas del reloj, aparecerá un círculo en su centro. Al eliminar este, aparecerá otro círculo en una hora aleatoria. Esto sucederá de manera cíclica mientras dure la partida.



6- ¿Qué significa cada elemento que aparece durante la partida?

Durante la partida aparecen los siguientes elementos:



7- ¿Cómo se consigue una ronda?

Una ronda se consigue tanto si se alcanza un círculo objetivo que no sea el que aparece en el centro, como si se sobrepasa el tiempo máximo de alcance y desaparece.

8- ¿Cómo se consigue la puntuación?

La puntuación se consigue alcanzando objetivos, cada objetivo alcanzado aumenta en 100 puntos el marcador.

9- ¿Qué pasa si se comenten muchos errores?

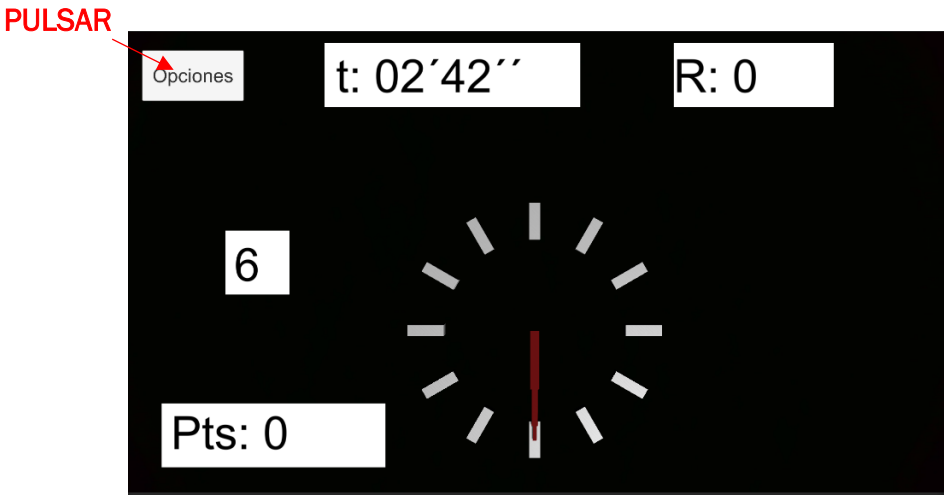
Al cometer errores pasan dos cosas para tener en cuenta:

- Se aumentan el número de rondas y se podría terminar la partida por alcanzar el máximo de rondas.

- Al alcanzar un cierto número de errores en una hora en concreto, el objetivo aparecerá más grande y cerca del centro cuando vuelva a salir esa hora como objetivo.

10- ¿Puedo pausar el juego a mitad de la partida?

El juego puede ser pausado pulsando el botón *Opciones* que aparece en la parte superior izquierda de la pantalla.



Al pulsar este botón, nos aparece la pantalla de *Pausa*, en la cual la partida pausada.

11- ¿Qué hace cada botón de la ventana de *Pausa*?

En la ventana de *Pausa* nos aparecen tres botones:

- *Nueva Partida:* Este botón nos lleva a una ventana que mezcla la ventana de inicio y la de opciones. En esta ventana podremos modificar los parámetros de la partida y empezar otra, o bien salir del juego definitivamente.

- *Volver:* Este botón reanuda el juego en la situación en la que estaba justo antes de pulsar el botón de *Opciones*.

- *Salir del Juego:* Este botón saca al usuario del juego.

12- ¿Qué sucede cuando se finaliza la partida por alcanzar la duración seleccionada?

Cuando el usuario alcanza el número de rondas seleccionado, o se alcanza el tiempo máximo de la partida, aparece una ventana de *Finalización de Partida*, en la cual el usuario puede elegir entre comenzar una nueva partida, o salir del juego.

13- ¿Se pueden visualizar datos de la partida?

Sí, durante la partida se recogen datos generales de la partida, como duración, rondas realizadas, puntuación y fallos y aciertos cometidos. Además, se guarda la posición de la mano en cada instante, lo que permite generar gráficos de posición. Estos datos se guardan en un archivo con formato *.csv (compatible con Excel)* en la carpeta *DataCollection*.

14- ¿Qué información se puede visualizar sobre la ejecución del paciente?

El conjunto de datos almacenados en esta aplicación se puede dividir en dos grupos:

- Datos globales de la partida: Tiempo total de la partida, rondas totales completadas, aciertos totales, errores totales cometidos, puntuación total y dificultad de la partida

- Datos instantáneos: Posición de la mano en cada instante, y el instante en el que se produce cada acierto o error.

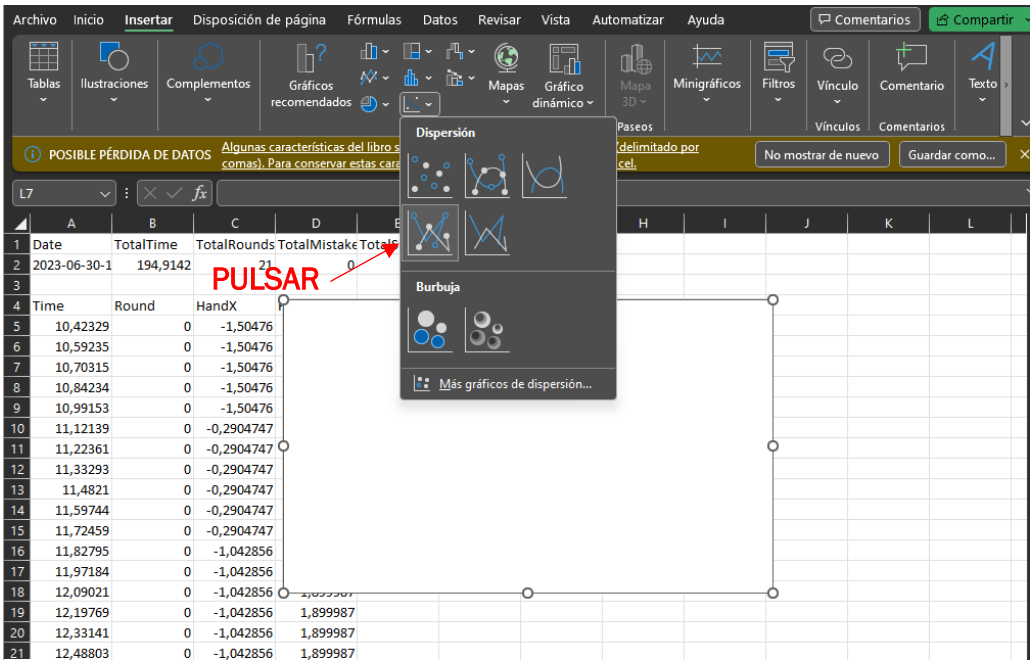
Los datos de los cuales podemos realizar una gráfica son los datos instantáneos, visualizando la trayectoria del usuario durante la partida.

15- ¿Cómo puedo generar un gráfico de las trayectorias que ha seguido el paciente?

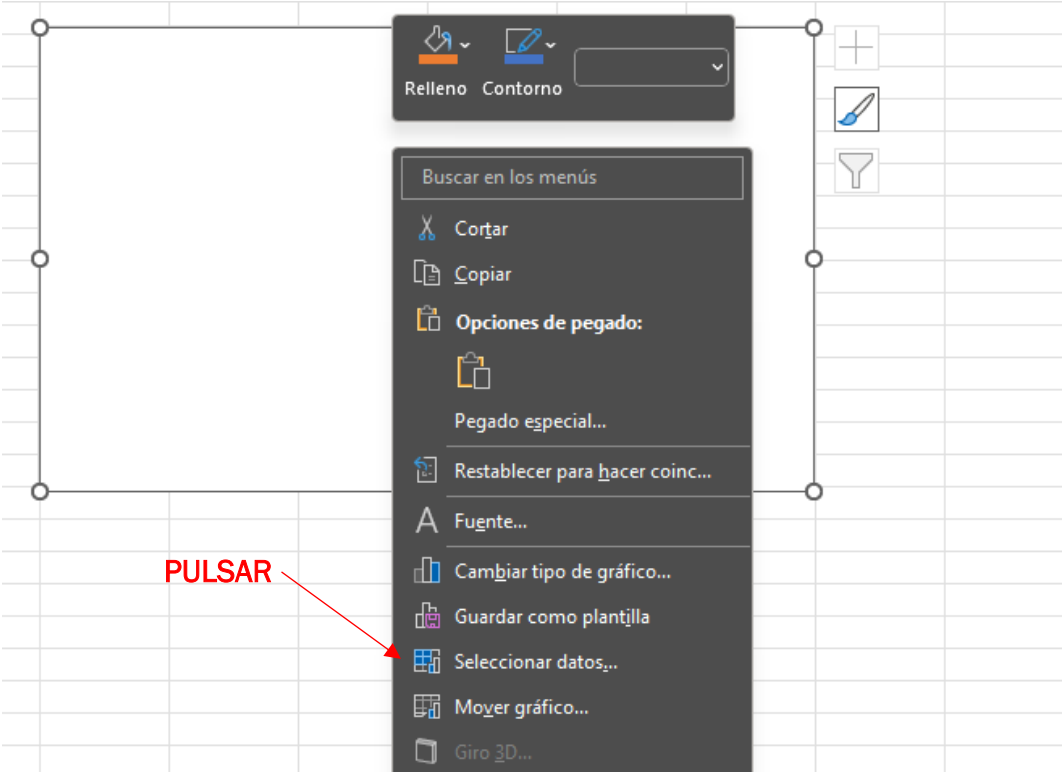
El documento *Excel* generado por el programa tiene este formato:

Date	TotalTime	TotalRounds	TotalMistake	TotalSuccess	Score	Difficulty
2023-06-30-1	194,9142	21	0	21	2100	Medium
Time	Round	HandX	HandZ	Status		
10,42329	0	-1,50476	-6,095251			
10,59235	0	-1,50476	-6,095251			
10,70315	0	-1,50476	-6,095251			
10,84234	0	-1,50476	-6,095251			
10,99153	0	-1,50476	-6,095251			
11,12139	0	-0,2904747	-1,100015			
11,22361	0	-0,2904747	-1,100013			
11,33293	0	-0,2904747	-1,100013			
11,4821	0	-0,2904747	-1,100013			
11,59744	0	-0,2904747	-1,100013			
11,72459	0	-0,2904747	-1,100013			
11,82795	0	-1,042856	1,899987			
11,97184	0	-1,042856	1,899987			

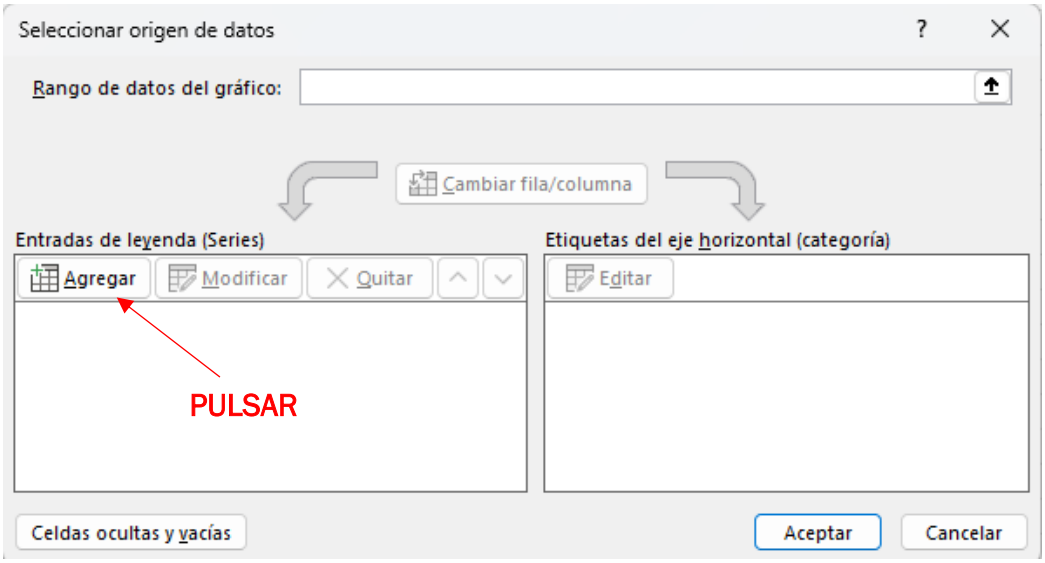
La parte superior muestra los datos globales de la partida, y en la parte inferior una lista de los datos instantáneos de la mano. Para mostrar un gráfico de las trayectorias, deberemos pulsar el botón de *Gráfico de dispersión con líneas rectas y marcadores*, de la ventana de *Inserta.*



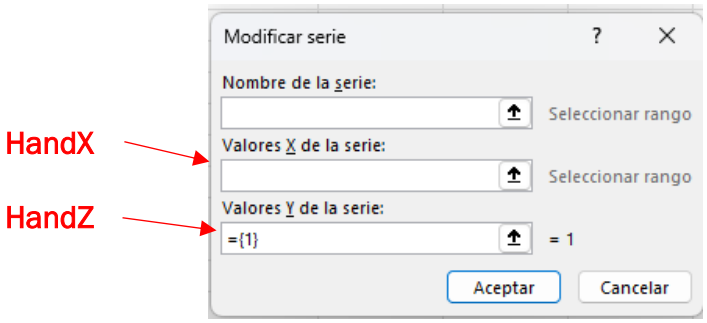
Pulsamos *click* derecho sobre el cuadro blanco que aparece en el documento de datos, y pulsamos sobre *Seleccionar datos*.



En la nueva ventana que aparece, pulsaremos el botón *Agregar* en la parte de la izquierda de la ventana.



Al pulsar el botón, nos surge el selector de datos, en el cual deberemos introducir la columna *HandX* en la sección x y la columna *HandZ* en la sección y.



Si se ha realizado bien el proceso, se debería poder visualizar el gráfico de trayectorias correctamente.

16- ¿Si se finaliza la partida manualmente desde el menú de pausa, se guardan los datos?

Sí, aunque no se deje finalizar la partida por los parámetros de duración elegidos de inicio, el juego guardará los datos como una partida normal.

ANEXO 2: Código Fuente de Juego del Reloj

GameLogic:

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


/// <summary>
/// Class Name: GameLogic
/// Author: David Matilla Plaza
/// Function: This class is used to control the operation of the game
/// and the UI
/// </summary>


public class GameLogic : MonoBehaviour
{
    //Private Variables
    private int _maxRounds;
    private int _hoursInRounds;
    private int _nextHour;
    private int _scale;
    private int _duration;
    private float _maxTime;
    private float _currentTime;
    private int _currentRound;
    private bool _finished;
    private int _score;

    //PlayerPrefs Variable Names
    public const string RoundsVariableName = "Rounds";
    public const string CurrentRoundVariableName = "CurrentRound";
    public const string MaxTimeVariableName = "Time";
    public const string CurrentTimeVariableName = "CurrentTime";
    public const string DurationTypeVariableName = "Duration";
    public const string NextHourVariableName = "nextHour";
    public const string ScoreVariableName = "Score";
    public const string ScaleVariableName = "Scale";
    public const string ClockPositionOffset = "ClockOffset";
    public const string FinishedVariableName = "Finished";
    public const string TotalMistakesVariableName = "TotalMistakes";
    public const string TotalSuccessVariableName = "TotalSuccess";
    public const string ErrorCounterVariablesName = "ErrorsIn";

    //Public Variables
    public GameObject CirclePrefab;
    public GameObject Clock;
    public Text ScoreText;
    public Text NextHourText;
    public GameObject RoundsPanel;
    public GameObject FinishedGamePanel;
    public Image DurationTime;
    public Image DurationRound;

    //Private Variables Associated with the public ones
    private Text _durationTimeText;
    private Text _durationRoundText;
```

```
private Transform _transformClock;

/// <summary>
/// Called when the object is activated, used to initialize the
/// variables of the class and the PlayerPrefs. Set the videogame to
/// start a new game
/// </summary>
void OnEnable()
{
    Time.timeScale = 1;

    InitializeErrorsCount();
    GetPlayerPrefsConstantValues();
    InitializeOwnVariables();
    ScaleAssignment();
    InitiatePlayerPrefsValues();
    SelectDurationFormatVisualized();
    DestroyAndInitializeCircle();
}

/// <summary>
/// Called every frame while runtime, update de variables value and
/// check if the game is finished
/// </summary>
void Update()
{
    _nextHour = PlayerPrefs.GetInt(NextHourVariableName, 0);
    _score = PlayerPrefs.GetInt(ScoreVariableName, 0);
    NextHourText.text = _nextHour.ToString();
    ScoreText.text = "Pts: " + string.Format("{0}", _score);

    RoundUpdate();
    TimeUpdate();
    CheckIfGameFinished();
}

/// <summary>
/// This function is used when the game is restarted
/// to destroy any existing circle and initialize a new one
/// </summary>
private void DestroyAndInitializeCircle()
{
    GameObject Posiblecircle = GameObject.FindWithTag("Circle");

    if (Posiblecircle != null)
    {
        Destroy(Posiblecircle);
    }

    Quaternion futureRotation = Quaternion.Euler(-90f, 0f, 0f);
    Vector3 futurePosition = _transformClock.position;
    GameObject circle = Instantiate(CirclePrefab,
        new Vector3(futurePosition.x,
                    futurePosition.y,
                    futurePosition.z), futureRotation);

    circle.tag = "Circle";
}

/// <summary>
/// Updates the round count and check if the game is finished by
/// rounds in case the duration selection was for rounds
/// </summary>
private void RoundUpdate()
```

```
{
    _currentRound = PlayerPrefs.GetInt(CurrentRoundVariableName, 0);

    if (_duration == 0 || _duration == 2)
    {
        _durationRoundText.text = "R: " + string.Format(
            "{0}", _currentRound);

        FinishGameByRounds();
    }
}

/// <summary>
/// Check if the game is finished by rounds
/// </summary>
private void FinishGameByRounds()
{
    if (_currentRound >= _maxRounds)
    {
        _finished = true;
        PlayerPrefs.SetInt(FinishedVariableName, 1);
    }
}

/// <summary>
/// Updates the time count and check if the game is finished by
/// rounds in case the duration selection was for time
/// </summary>
private void TimeUpdate()
{
    _currentTime += Time.deltaTime;
    int tiempo = (int)(_maxTime - _currentTime);
    PlayerPrefs.SetFloat(CurrentTimeVariableName, _currentTime);
    string TimeText;

    if (tiempo >= 60)
    {
        float minutes = tiempo / 60;
        float remainingSeconds = tiempo % 60;
        TimeText = string.Format(
            "{0:00}´{1:00}´´", minutes, remainingSeconds);
    }
    else
    {
         TimeText = string.Format("{0:00}''", tiempo);
    }

    if (_duration == 1 || _duration == 2)
    {
        _durationTimeText.text = "t: " + TimeText;

        FinishGameByTime();
    }
}

/// <summary>
/// Check if the game is finished by time
/// </summary>
private void FinishGameByTime()
{
    if (_currentTime >= _maxTime)
    {
        _finished = true;
        PlayerPrefs.SetInt(FinishedVariableName, 1);
```

```
    }
}

/// <summary>
/// Initialize the hour error count to zero
/// </summary>
private void InitializeErrorsCount()
{
    for (int i = 1; i <= 12; i++)
    {
        string VariableName = ErrorCounterVariablesName
                              + i.ToString();
        PlayerPrefs.SetInt(VariableName, 0);
    }
}

/// <summary>
/// Select the clock position and scale based on difficulty
/// selection
/// </summary>
private void ScaleAssignment()
{
    if (_scale == 1)
    {
        Clock.transform.localScale = new Vector3(0.8f, 0.8f, 0.8f);
        Clock.transform.position = new Vector3(0f, 0f, -15f);
    }
    else if (_scale == 2)
    {
        Clock.transform.localScale = new Vector3(0.9f, 0.9f, 0.9f);
        Clock.transform.position = new Vector3(0f, 0f, -13f);
    }
    else
    {
        Clock.transform.localScale = new Vector3(1f, 1f, 1f);
        Clock.transform.position = new Vector3(0f, 0f, -10f);
    }
}

/// <summary>
/// Update de local variables with the PlayerPrefs value
/// </summary>
private void GetPlayerPrefsConstantValues()
{
    _scale = PlayerPrefs.GetInt(ScaleVariableName, 0);
    _duration = PlayerPrefs.GetInt(DurationTypeVariableName, 0);
    _maxTime = (float)PlayerPrefs.GetInt(MaxTimeVariableName, 0);
    _maxRounds = PlayerPrefs.GetInt(RoundsVariableName, 0);
}

/// <summary>
/// Initialize the PlayerPrefs Variables
/// </summary>
private void InitiatePlayerPrefsValues()
{
    PlayerPrefs.SetInt(CurrentRoundVariableName, 0);
    PlayerPrefs.SetFloat(CurrentTimeVariableName, 0f);
    PlayerPrefs.SetInt(ScoreVariableName, 0);
    PlayerPrefs.SetInt(FinishedVariableName, 0);
    PlayerPrefs.SetInt(TotalMistakesVariableName, 0);
    PlayerPrefs.SetInt(TotalSuccessVariableName, 0);
    PlayerPrefs.SetFloat(
        ClockPositionOffset, _transformClock.position.z);
    Debug.Log("Offset del reloj" + _transformClock.position.z);
```

```
    }

    /// <summary>
    /// Select the UI duration info. If both durations ar selected,
    /// both texts are assingned
    /// </summary>
    private void SelectDurationFormatVisualized()
    {
        if (_duration == 0 || _duration == 2)
        {
            DurationRound.enabled = true;
            _durationRoundText =
                DurationRound.GetComponentInChildren<Text>();
        }
        if (_duration == 1 || _duration == 2)
        {
            DurationTime.enabled = true;
            _durationTimeText =
                DurationTime.GetComponentInChildren<Text>();
        }
    }

    /// <summary>
    /// Initialize the class variables
    /// </summary>
    private void InitializeOwnVariables()
    {
        _nextHour = 0;
        _currentRound = 1;
        _currentTime = 0f;
        _transformClock = Clock.transform;

        _finished = false;
    }

    /// <summary>
    /// Check the bool variable _finished to end the game
    /// </summary>
    private void CheckIfGameFinished()
    {
        if (_finished)
        {
            RoundsPanel.SetActive(false);
            FinishedGamePanel.SetActive(true);
        }
    }
}
```

ClockLogic:

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

/// <summary>
/// Class Name: ClockLogic
/// Author: David Matilla Plaza
/// Function: Used to control the rotation of the clock pointers
/// /// </summary>
public class ClockLogic : MonoBehaviour
{

    public GameObject pointerSeconds;
    public GameObject pointerMinutes;
    public GameObject pointerHours;
    private string nextHourVariableName = "nextHour";

    /// <summary>
    /// Called before the first frame. It initialize the pointers
    /// rotation to zero
    /// </summary>
    void Start()
    {
        pointerSeconds.transform.localEulerAngles = new Vector3(0.0f, 0.0f,
 0f);
        pointerMinutes.transform.localEulerAngles = new Vector3(0.0f, 0.0f,
 0f);
        pointerHours.transform.localEulerAngles = new Vector3(0.0f, 0.0f, 0
f);
    }

    /// <summary>
    /// Called every frame. It updates the pointers rotation to point
    /// the current hour selected
    /// </summary>
    void Update()
    {
        int hour = PlayerPrefs.GetInt(nextHourVariableName, 0);

        float rotationSeconds = (360.0f / 12.0f) * hour;
        float rotationMinutes = (360.0f / 12.0f) * hour;
        float rotationHours = (360.0f / 12.0f) * hour;

        pointerSeconds.transform.localEulerAngles = new Vector3(
                                            0.0f, rotationSeconds, 0.0f);
        pointerMinutes.transform.localEulerAngles = new Vector3(
                                            0.0f, rotationMinutes, 0.0f);
        pointerHours.transform.localEulerAngles = new Vector3(
                                            0.0f, rotationHours, 0.0f);
```

CircleSpawner:

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

/// <summary>
/// Class Name: CircleSpawner
/// Author: David Matilla Plaza
/// Function: This class is used to create and destroy circles during
/// the game. It also count errors and update variables
/// </summary>

public class CircleSpawner : MonoBehaviour
{
    //Public variables
    public GameObject CirclePrefab;
    public GameObject DestroySoundSuccess;
    public GameObject DestroySoundError;
    public GameObject CircleAppearanceSound;
    public GameObject CircleDisappearanceSound;

    //PlayerPrefs Variables Names
    public const string NextHourVariableName = "nextHour";
    public const string RoundsVariableName = "Rounds";
    public const string ScaleVariableName = "Scale";
    public const string CurrentRoundVariableName = "CurrentRound";
    public const string ErrorCounterVariablesName = "ErrorsIn";
    public const string MaxTimeErrorVariableName = "MaxErrorTime";
    public const string ScoreVariableName = "Score";
    public const string TotalMistakesVariableName = "TotalMistakes";
    public const string TotalSuccessVariableName = "TotalSuccess";

    //Private Variables
    private int _currentRound;
    private int _scale;
    private int _score;
    private int _totalSuccess;
    private int _totalMistakes;
    private float _maxTime;
    private float _currentTime;
    private float _blinkPeriod;
    private float _currentBlinkTime;
    private bool _circleVisible;
    private Vector3 _newScale = new Vector3(3f, 3f, 3f);
    private Quaternion _futureRotation = Quaternion.Euler(-90f, 0f, 0f);


    //GameObjects variables
    private GameObject _clock;
    private GameObject _centreParticles1Way;
    private GameObject _centreParticlesOtherWay;
    private GameObject _innerRadius;
    private GameObject _outterRadius;

    private GameObject _nextCentreParticles1Way;
    private GameObject _nextCentreParticlesOtherWay;
    private GameObject _nextInnerRadius;
    private GameObject _nextOutterRadius;

    private Transform _transformClock;
    private Transform _transformHour;
```

```
//Constants
private const int minHour = 1, maxHour = 12;
private const float startDelay = 4f;
private const float _errorDistanceReductionFactor = 0.7f;
private const int _numberOfErrorCounterLimit = 2;


/// <summary>
/// Called before the first frame has started. Initialize all the
/// variables of the class
/// </summary>
void Start()
{
    GetPlayerPrefsConstantValues();
    InitializeOwnVariables();
    GetGameObjectsOnScene();
}

/// <summary>
/// Called every frame while runtime, update the game time, and make
/// the circle blink in the period selected. When the max time error
/// is reached, another circle is created
/// </summary>
void Update()
{
    _currentTime += Time.deltaTime;
    _currentBlinkTime += Time.deltaTime;
    CircleBlinkFunction();
    if (_currentTime >= _maxTime)
    {
        GenerateNextCircle(true);
    }
}

/// <summary>
/// Generate a new circle in the correct position based of the
/// position of the current circle. It also destroy the current
/// circle and instantiate the sound
/// </summary>
/// <param name="error"></param>
public void GenerateNextCircle(bool error)
{
    Vector3 currentPosition = transform.position;
    if(currentPosition == _transformClock.position || error == true)
    {
        if(error == true && currentPosition !=
            _transformClock.position)
        {
            UpdateErrors();
            UpdateMistakes();
        }
        NewCircleInRandomPosition();
    }
    else
    {
        NewCircleInClockCentre();
        UpdateSuccess();
    }
    if (error == true) { Instantiate(DestroySoundError); }
    else { Instantiate(DestroySoundSuccess); }
    Destroy(gameObject);
}
```

```
/// <summary>
/// Initialize all the variables of objects in scene
/// </summary>
private void GetGameObjectsOnScene()
{
    _clock = GameObject.Find("InvClock");
    _transformClock = _clock.transform;

    _centreParticles1Way = transform.Find(
        "CentreParticles1Way").gameObject;
    _centreParticlesOtherWay = transform.Find(
        "CentreParticlesOtherWay").gameObject;
    _innerRadius = transform.Find("InnerRadius").gameObject;
    _outterRadius = transform.Find("OutterRadius").gameObject;

    _centreParticles1Way.GetComponent<ParticleSystem>().
        GetComponent<ParticleSystemRenderer>().enabled = true;
    _centreParticlesOtherWay.GetComponent<ParticleSystem>().
        GetComponent<ParticleSystemRenderer>().enabled = true;
    _innerRadius.GetComponent<ParticleSystem>().
        GetComponent<ParticleSystemRenderer>().enabled = true;
    _outterRadius.GetComponent<ParticleSystem>().
        GetComponent<ParticleSystemRenderer>().enabled = true;
}

/// <summary>
/// Initialize the class variables
/// </summary>
private void InitializeOwnVariables()
{
    _currentTime = 0f;
    _blinkPeriod = 0.5f;
    _currentBlinkTime = 0f;
    _circleVisible = true;
}

/// <summary>
/// Initialize the variables of PlayerPrefs
/// </summary>
private void GetPlayerPrefsConstantValues()
{
    _currentRound = PlayerPrefs.GetInt(CurrentRoundVariableName, 0);
    _score = PlayerPrefs.GetInt(ScoreVariableName, 0);
    _maxTime = (float)PlayerPrefs.GetInt(
        MaxTimeErrorVariableName, 0) + startDelay;
}

/// <summary>
/// Change the render status of the cirlce each in the period
/// </summary>
private void CircleBlinkFunction()
{
    if (_currentTime >= (_maxTime / 2) &&
        _currentBlinkTime >= _blinkPeriod)
    {
        _currentBlinkTime = 0f;
        if (_circleVisible)
        {
           Instantiate(CircleDisappearanceSound);
        }
        else { Instantiate(CircleAppearanceSound); }
        ChangeCircleRenderStatus();
    }
}
```

```
/// <summary>
/// Invert all the circle components render status
/// </summary>
private void ChangeCircleRenderStatus()
{
    _centreParticles1Way.GetComponent<ParticleSystem>()
        .GetComponent<ParticleSystemRenderer>().enabled =
        !_circleVisible;
    _centreParticlesOtherWay.GetComponent<ParticleSystem>()
        .GetComponent<ParticleSystemRenderer>().enabled =
        !_circleVisible;
    _innerRadius.GetComponent<ParticleSystem>()
        .GetComponent<ParticleSystemRenderer>().enabled =
        !_circleVisible;
    _outterRadius.GetComponent<ParticleSystem>()
        .GetComponent<ParticleSystemRenderer>().enabled =
        !_circleVisible;

    _circleVisible = !_circleVisible;
}

/// <summary>
/// Update the mistakes count of the current hour
/// </summary>
private void UpdateMistakes()
{
    _currentRound = PlayerPrefs.GetInt(CurrentRoundVariableName, 0);
    PlayerPrefs.SetInt(CurrentRoundVariableName, _currentRound + 1);

    _totalMistakes = PlayerPrefs.GetInt(
        TotalMistakesVariableName, 0);
    PlayerPrefs.SetInt(
        TotalMistakesVariableName, _totalMistakes + 1);
}

/// <summary>
/// Update the success count of the current hour
/// </summary>
private void UpdateSuccess()
{
    _currentRound = PlayerPrefs.GetInt(CurrentRoundVariableName, 0);
    PlayerPrefs.SetInt(CurrentRoundVariableName, _currentRound + 1);

    _score = PlayerPrefs.GetInt(ScoreVariableName, 0);
    PlayerPrefs.SetInt(ScoreVariableName, _score + 100);

    _totalSuccess = PlayerPrefs.GetInt(TotalSuccessVariableName, 0);
    PlayerPrefs.SetInt(TotalSuccessVariableName, _totalSuccess + 1);
}

/// <summary>
/// Update the errors of the current hour
/// </summary>
private void UpdateErrors()
{
    int currentHour = PlayerPrefs.GetInt(NextHourVariableName, 0);
    string ErrorName =
        ErrorCounterVariablesName + currentHour.ToString();
    int ErrorCount = PlayerPrefs.GetInt(ErrorName, 0);
    PlayerPrefs.SetInt(ErrorName, ErrorCount + 1);

    Debug.LogFormat("Se han comentido {0} errores en la hora {1}",
        ErrorCount + 1, currentHour);
}
```

```
/// <summary>
/// This function is called when the current circle is in the centre
/// of the clock or if the user didnt reach it. In those cases the
/// next circle must appear in a random clock position
/// </summary>
private void NewCircleInRandomPosition()
{
    int nextHour = Random.Range(minHour, maxHour + 1);
    string hourName = string.Format("{0}Hour", nextHour);
    _transformHour = _transformClock.Find(hourName);

    Vector3 futurePosition = NextPositionDependingOnNumberOfErrors(
        _transformHour.position,nextHour);
    GameObject circle = Instantiate(CirclePrefab, new Vector3(
        futurePosition.x, futurePosition.y, futurePosition.z)
        , _futureRotation);
    TransformCircleScale(circle, _newScale);

    circle.tag = "Circle";
    PlayerPrefs.SetInt(NextHourVariableName, nextHour);
}

/// <summary>
/// When the user made many errors in the same hour the circle
/// created has bigger scale. This function changes the scale
/// of the circle
/// </summary>
/// <param name="circle">The circle scale changing</param>
/// <param name="scale">The new scale to assign</param>
private void TransformCircleScale(GameObject circle, Vector3 scale)
{
    _nextCentreParticles1Way =
        circle.transform.Find("CentreParticles1Way").gameObject;
    _nextCentreParticlesOtherWay = circle.transform.Find(
        "CentreParticlesOtherWay").gameObject;
    _nextInnerRadius = circle.transform.Find(
        "InnerRadius").gameObject;
    _nextOutterRadius = circle.transform.Find(
        "OutterRadius").gameObject;

    circle.transform.localScale = scale;
    _nextCentreParticles1Way.transform.localScale = scale;
    _nextCentreParticlesOtherWay.transform.localScale = scale;
    _nextInnerRadius.transform.localScale = scale;
    _nextOutterRadius.transform.localScale = scale;
}

/// <summary>
/// When the user make many errors in the same hour, the next circle
/// appears in a position nearer to the centre of the circle. This
/// function checks the errors count an decide the position of
/// the circle
/// </summary>
/// <param name="hourPosition">The next goal hour position</param>
/// <param name="nextHour">The next hour number</param>
/// <returns>Next circle position Vector3</returns>
private Vector3 NextPositionDependingOnNumberOfErrors(
    Vector3 hourPosition, int nextHour)
{
    string ErrorName = ErrorCounterVariablesName +
        nextHour.ToString();
    int ErrorCount = PlayerPrefs.GetInt(ErrorName, 0);
    Vector3 nextposition;
```

```
        float hourToCentreDistance =
            Mathf.Sqrt(Mathf.Pow(
                hourPosition.x - _transformClock.position.x, 2)
            + Mathf.Pow(
                hourPosition.z - _transformClock.position.z, 2));

        if (ErrorCount >= _numberOfErrorCounterLimit)
        {
            nextposition.x = _errorDistanceReductionFactor *
                hourToCentreDistance *
                Mathf.Sin(((30f * (float)nextHour) *
                (2 * Mathf.PI)) / 360) + _transformClock.position.x;
            nextposition.y = hourPosition.y;
            nextposition.z = _errorDistanceReductionFactor *
                hourToCentreDistance *
                Mathf.Cos(((30f * (float)nextHour) *
                (2 * Mathf.PI)) / 360) + _transformClock.position.z;

            _newScale = new Vector3(4f, 4f, 4f);
        }
        else
        {
            nextposition = hourPosition;
            _newScale = new Vector3(3f, 3f, 3f);
        }
        return nextposition;
    }

    /// <summary>
    /// When the current circle was in another position different from
    /// the centre and non error was made, the next circle position
    /// may appear in the centre of the circle. This function make that
    /// circle appear.
    /// </summary>
    private void NewCircleInClockCentre()
    {
        Vector3 futurePosition = _transformClock.position;
        GameObject circle = Instantiate(CirclePrefab,
            new Vector3(futurePosition.x,
            futurePosition.y, futurePosition.z), _futureRotation);
        circle.tag = "Circle";
    }

}
```

DataCollector:

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

/// <summary>
/// Class Name: DataCollector
/// Author: David Matilla Plaza
/// Function: This class is used to get data from the game and save
/// it in a .csv file
/// </summary>


public class DataCollector : MonoBehaviour
{
    //PlayerPrefs Variables Names
    public const string CurrentTimeVariableName = "CurrentTime";
    public const string ScoreVariableName = "Score";
    public const string CurrentRoundVariableName = "CurrentRound";
    public const string FinishedVariableName = "Finished";
    public const string TotalMistakesVariableName = "TotalMistakes";
    public const string TotalSuccessVariableName = "TotalSuccess";
    public const string ScaleVariableName = "Scale";
    public const string ClockPositionOffset = "ClockOffset";


    //Private Variables
    private List<Vector3> _positionsList;
    private List<float> _timeList;
    private List<int> _roundList;
    private List<int> _mistakesList;
    private List<int> _successList;
    private int _totalSuccess;
    private int _totalMistakes;
    private LayerMask _layer;
    private const float _periodOfHandData = 0.1f;
    private float _time;
    private float _currentTime;
    private int _rounds;
    private float _clockOffset;
    private int _finished;
    private int _score;
    private string _scale;
    private string _currentDate;
    private Collider[] _colliders;


    /// <summary>
    /// Called when the object is activated, used to initialize the
    /// variables of the class and the PlayerPrefs. Set the videogame to
    /// start a new game
    /// /// </summary>
    private void OnEnable()
    {
        _positionsList = new List<Vector3>();
        _timeList = new List<float>();
        _roundList = new List<int>();
        _mistakesList = new List<int>();
        _successList = new List<int>();
        GetDifficulty();
        _time = 0f;
```

```
    _finished = 0;
    PlayerPrefs.SetInt(FinishedVariableName, _finished);
}

/// <summary>
/// Called every frame while runtime, update de variables value and
/// if the saving period is completed, then save the values in a
/// list. When the game is finished, the function make a backup of
/// the list info in the file
/// </summary>
private void Update()
{
    _finished = PlayerPrefs.GetInt(FinishedVariableName, 0);
    _time += Time.deltaTime;
    if((_time > _periodOfHandData) && (_finished != 1))
    {
        UpdateVariables();
        _time = 0f;
    }
    if(_finished == 1)
    {
        SaveOnCSV();
    }
}

/// <summary>
/// This function get the difficulty name from the PlayerPrefs var
/// </summary>
private void GetDifficulty()
{
    int scale = PlayerPrefs.GetInt(ScaleVariableName, 0);
    if(scale == 1) { _scale = "Easy"; }
    else if(scale == 2) { _scale = "Medium"; }
    else { _scale = "Difficult"; }

}

/// <summary>
/// Update every list of values
/// </summary>
private void UpdateVariables()
{
    Vector3 handPosition = GetHandMeanPosition();

    if ((!float.IsNaN(handPosition.x)
         || !float.IsNaN(handPosition.y)
         || !float.IsNaN(handPosition.z)))
    {
        _positionsList.Add(handPosition);
        Debug.Log("Posicion de la mano: " + handPosition);

        _currentTime = PlayerPrefs.GetFloat(CurrentTimeVariableName,
            0);
        _timeList.Add(_currentTime);

        _rounds = PlayerPrefs.GetInt(CurrentRoundVariableName,
            0);
        _roundList.Add(_rounds);

        _totalSuccess = PlayerPrefs.GetInt(TotalSuccessVariableName,
            0);
        _successList.Add(_totalSuccess);

        _totalMistakes = PlayerPrefs.GetInt(
```

```
            TotalMistakesVariableName, 0);
        _mistakesList.Add(_totalMistakes);

        _score = PlayerPrefs.GetInt(ScoreVariableName, 0);
    }

}

/// <summary>
/// This function get the position of the hand landmarks and
/// calculate the mean position of the hand
/// </summary>
/// <returns>The return is the mean position Vector3 </returns>
private Vector3 GetHandMeanPosition()
{
    GameObject[] objectsInLayer = GetObjectsInLayer();
    (Vector3 positionSum, int nPos) =
        GetSummationOfPositions(objectsInLayer);
    _clockOffset = PlayerPrefs.GetFloat(ClockPositionOffset, 0f);


    Vector3 meanPosition = new Vector3(positionSum.x / (float)nPos
        , positionSum.y / (float)nPos
        , (positionSum.z / (float)nPos) - _clockOffset);

    return meanPosition;
}

/// <summary>
/// Get the objects in the landmark layer and append them to a list
/// </summary>
/// <returns>The list of hand landmarks</returns>
private GameObject[] GetObjectsInLayer()
{
    LayerMask layer;
    int layerNumber = LayerMask.NameToLayer("Landmark");
    layer = 1 << layerNumber;
    _colliders = Physics.OverlapSphere(
        Vector3.zero, Mathf.Infinity, layer);
    GameObject[] objectsInLayer = new GameObject[_colliders.Length];
    for (int i = 0; i < _colliders.Length; i++)
    {
        objectsInLayer[i] = _colliders[i].gameObject;
    }
    return objectsInLayer;
}

/// <summary>
/// Get the summation of all landmarks position vector and count
/// the total number of objects in the list
/// </summary>
/// <param name="objectsInLayer">The list of hand landmarks</param>
/// <returns>Summation vector of positions and number of
/// landmark objects used for the hand</returns>
private (Vector3, int) GetSummationOfPositions(
    GameObject[] objectsInLayer)
{
    Vector3[] handPositions = new Vector3[_colliders.Length];
    Vector3 positionSum = new Vector3(0f, 0f, 0f);
    int nPos = 0;

    foreach (GameObject obj in objectsInLayer)
    {
        Vector3 position = obj.transform.position;
```

```
        //When the landmark is not detected, the landmark stays
        //in initial position (-106, 5, 54)
        if (position != new Vector3(-106f, 5f, 64f))
        {
            handPositions[nPos] = position;
            positionSum += position;
            nPos = nPos + 1;
            Debug.Log("Posicion encontrada " + position);
        }
    }
    return (positionSum, nPos);
}

/// <summary>
/// Create the .csv and save all the values of the lists
/// </summary>
public void SaveOnCSV()
{
    StreamWriter file = CreateCSV();

    //This writes the fist lines of the file
    file.WriteLine($"{_currentDate};{_currentTime};{_rounds};" +
        $"{_totalMistakes};{_totalSuccess};{_score};{_scale}");
    file.WriteLine("");
    file.WriteLine("Time;Round;HandX;HandZ;Status");

    int i = 0;
    foreach (Vector3 position in _positionsList)
    {
        string status = "";
        if ((i > 0) && (_mistakesList[i-1] !=
            _mistakesList[i])) { status = "Mistake"; }
        if ((i > 0) && (_successList[i-1] !=
            _successList[i])) { status = "Success"; }
        file.WriteLine($"{_timeList[i]};{_roundList[i]};" +
            $"{position.x};{position.z};" +
            $"{status}");
        i++;
    }

    file.Close();
    gameObject.SetActive(false);
    Time.timeScale = 0;
}

/// <summary>
/// Create the .csv with the date in the path name
/// </summary>
/// <returns>the file</returns>
private StreamWriter CreateCSV()
{
    string folder = Path.Combine(
        Application.dataPath, "DataCollection");
    if (!Directory.Exists(folder))
    {
        Directory.CreateDirectory(folder);
    }
    _currentDate = DateTime.Now.ToString("yyyy-MM-dd-HH-mm-ss");
    string fileName = _currentDate + ".csv";
    string filePath = Path.Combine(folder, fileName);
    bool fileExist = File.Exists(filePath);
    StreamWriter file = new StreamWriter(filePath, true);
    if (!fileExist)
    {
```

```
                file.WriteLine("Date;TotalTime;TotalRounds;TotalMistakes;" +
                    "TotalSuccess;Score;Difficulty");
            }
            return file;
        }

        /// <summary>
        /// This function is used when the user exit the game midtime and
        /// we need to save the data in the file
        /// </summary>
        public void ExitGameMidGame()
        {
            SaveOnCSV();
            Application.Quit();
        }
    }
```

ANEXO 3: Manual de Usuario de Juego del Ratón

1- ¿Cuáles son los parámetros de este juego y cómo se configuran?

En este juego hay un único parámetro para este juego. Este parámetro es la dificultad de la partida, y lo que modifica esta dificultad es la zona en la que el ratón puede huir.

Para configurar este parámetro debemos modificar el valor de la barra de la pantalla de inicio.

Al modificar esta barra varía la zona roja de la ventana, esta zona representa la zona en la que puede moverse el ratón.

2- ¿Cómo se inicia la partida?

Cuando ya hemos configurado la dificultad a nuestro gusto, iniciaremos la partida pulsando el botón iniciar partida que se encuentra debajo de la barra de dificultad.

3- ¿Cuál es el objetivo del usuario durante la partida?

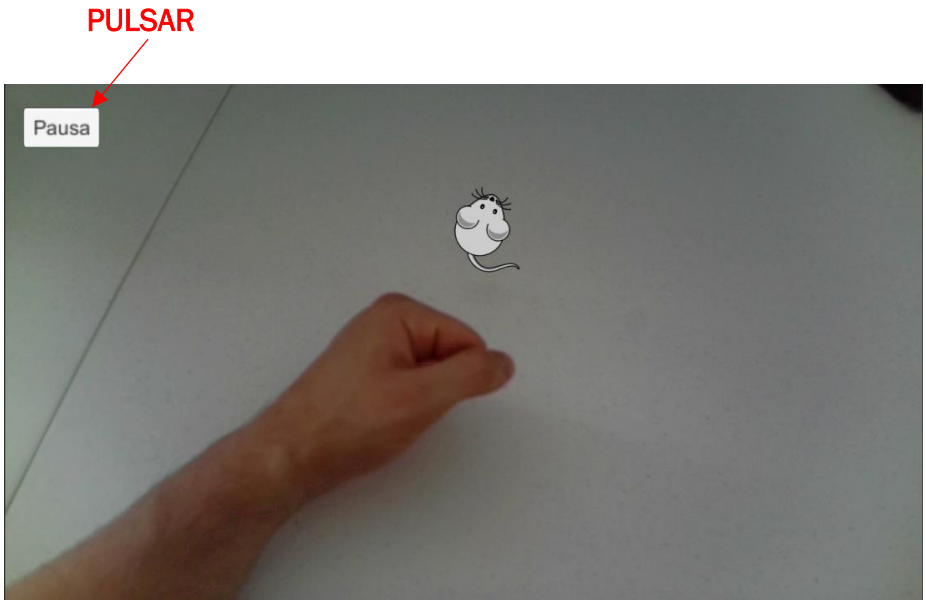
El usuario deberá intentar atrapar el objetivo con su mano, en este caso la imagen del ratón. El ratón tratará de huir de él en la zona delimitada según la dificultad que se haya elegido de inicio.

4- ¿Cuándo finaliza la partida?

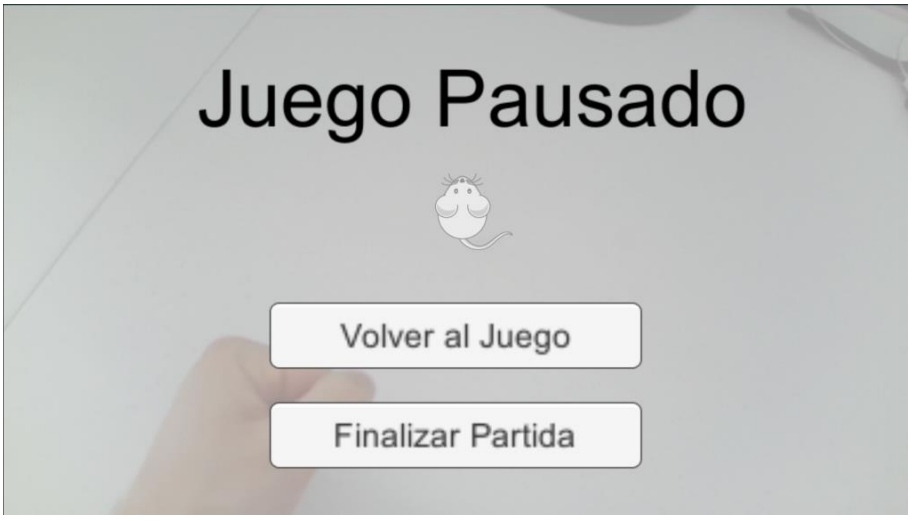
La partida no tiene fin como tal, el terapeuta es el encargado de finalizarla cuando haya determinado que se ha completado la sesión.

5- ¿Se puede Pausar la partida?

Sí, se puede Pausar la partida. Esto se puede realizar utilizando el botón *Pausa* de la parte superior izquierda de la pantalla.

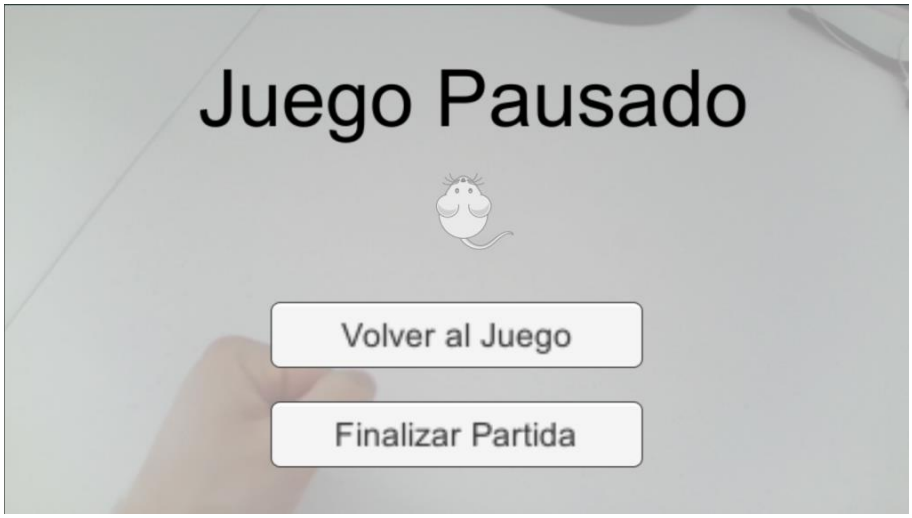


Este botón nos redirige a la ventana de *Pausa*, mientras nos encontremos en esta ventana, el tiempo de partida está detenido.



6- ¿Cómo se finaliza la partida de forma manual?

Para finalizar la partida de forma manual, deberemos pulsar el botón que se encuentra en la parte inferior de la pantalla de pausa. Este botón nos permite volver a la pantalla de inicio, finalizando la partida.



7- ¿Qué información se puede visualizar sobre la ejecución del paciente?

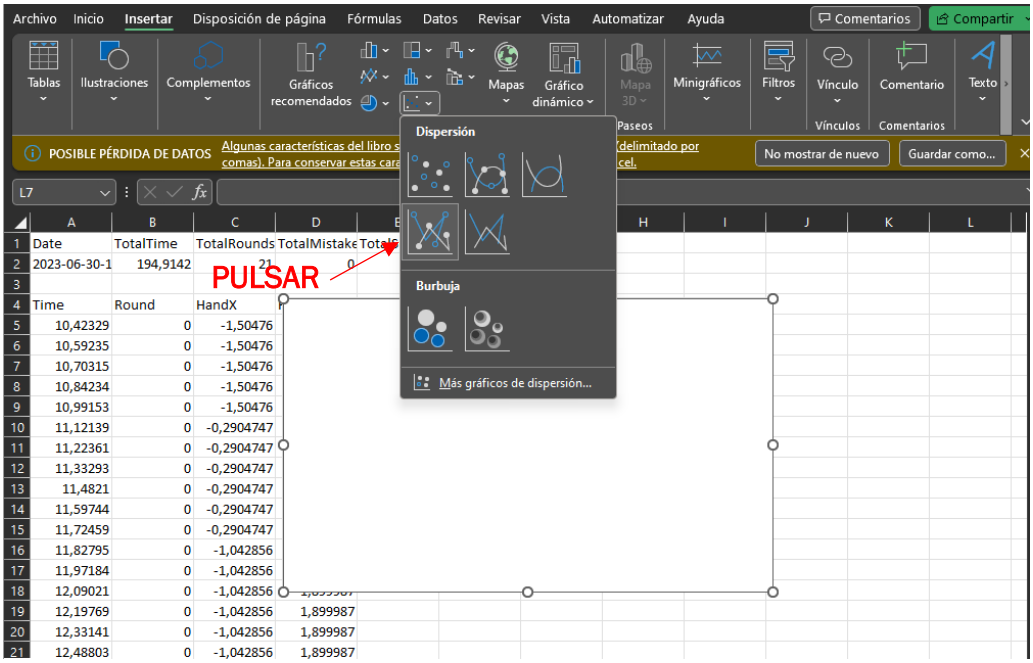
En esta aplicación se pueden realizar dos gráficas para visualizar el comportamiento del paciente durante la prueba. La primera representa la distancia del usuario al ratón a lo largo del tiempo, y la segunda muestra la trayectoria efectuada por el paciente durante la partida.

8- ¿Cómo puedo generar un gráfico de las trayectorias que ha seguido el paciente?

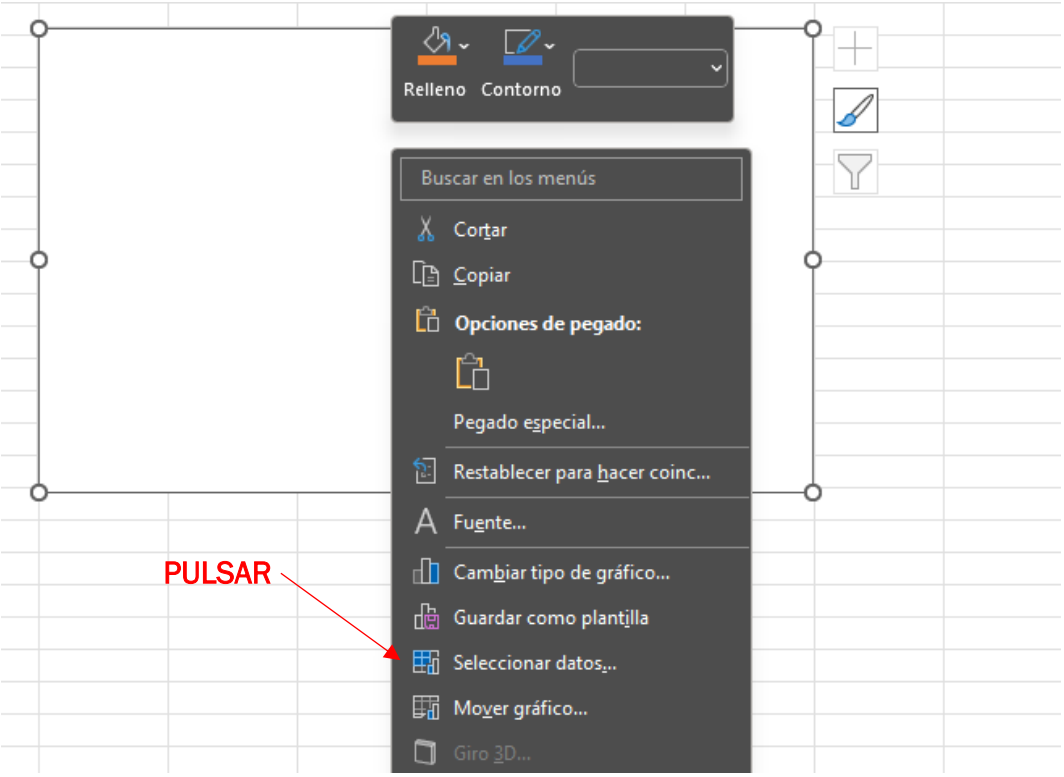
El documento *Excel* generado por el programa tiene este formato:

Date	2023-07-04-15-35-43		
Time	HandX	HandZ	DistanceToUser
0,1163492	-72,59523	9,909514	73,86747
0,2490866	-72,59523	9,909514	73,86747
0,3495329	-72,71905	9,695228	74,02899
0,4662359	-73,00952	10,15713	74,22955
0,5687112	-72,99524	10,07618	74,23017
0,6831695	-74,03809	10,27142	75,22118
0,7996004	-74,03809	10,27142	75,22118
0,899617	-73,7	-8,833348	80,50477
1,016488	-48,18095	-31,72859	73,33643
1,13286	-48,18095	-31,72859	73,33643
1,232919	-27,39047	-34,92383	64,58015
1,335388	-21,44286	-33,43811	60,89812
1,450176	-20,65238	-32,86668	60,08736

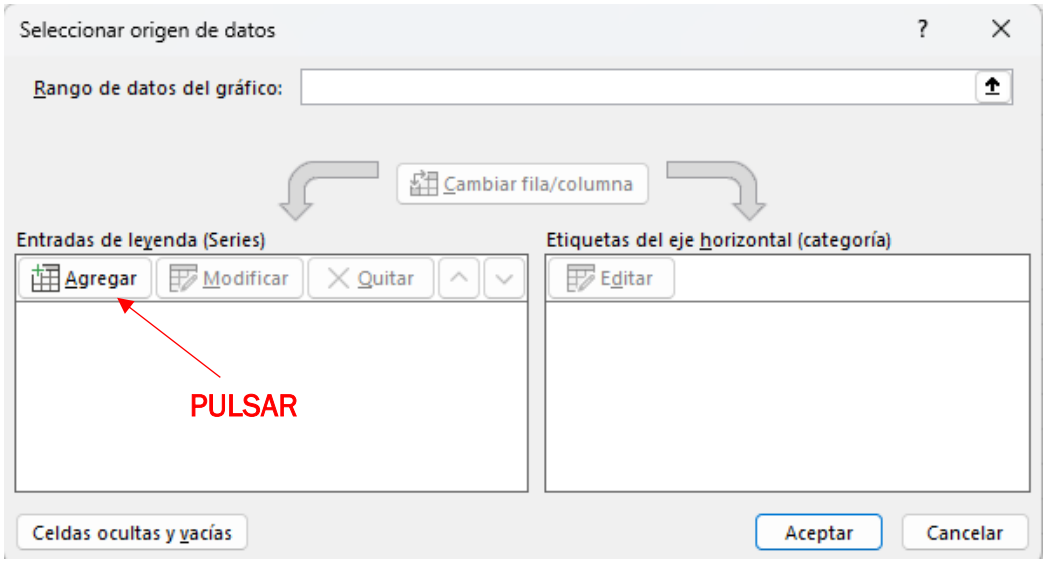
La parte superior muestra los datos globales de la partida, y en la parte inferior una lista de los datos instantáneos de la mano. Para mostrar un gráfico de las trayectorias, deberemos pulsar el botón de *Gráfico de dispersión con líneas rectas y marcadores*, de la ventana de *Inserta.*



Pulsamos *click* derecho sobre el cuadro blanco que aparece en el documento de datos, y pulsamos sobre *Seleccionar datos*.

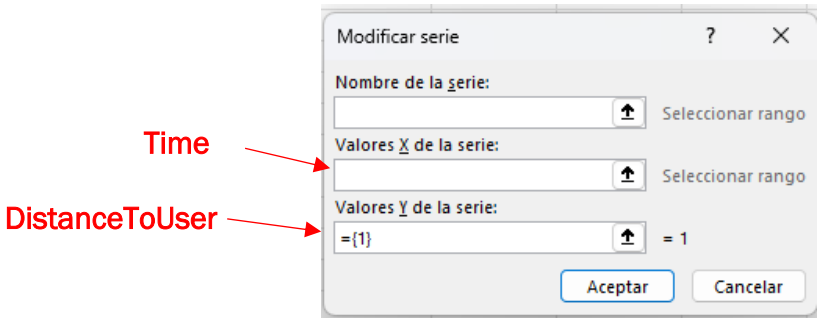


En la nueva ventana que aparece, pulsaremos el botón *Agregar* en la parte de la izquierda de la ventana.

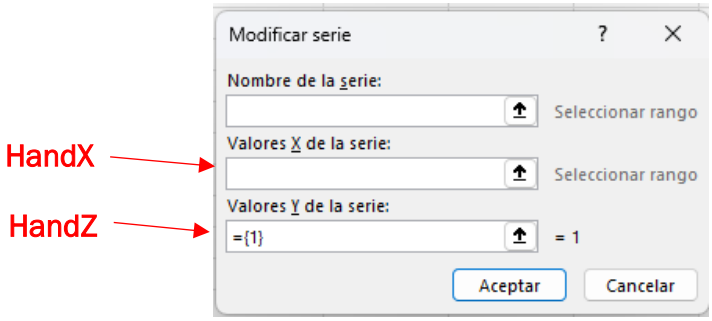


Hasta aquí, el proceso es el mismo tanto para el gráfico de la distancia, como para el de la trayectoria. En la siguiente ventana deberemos elegir una columna de datos u otra en función del gráfico que se quiera realizar.

En el caso de buscar el gráfico de la distancia a lo largo del tiempo, deberemos introducir la columna de datos *Time* en la sección de valores x y la columna de datos *DistanceToUser* en la sección de valores y.



En el caso de buscar el gráfico de la trayectoria del usuario a lo largo de la partida, deberemos introducir la columna de datos *HandX* en la sección de valores x y la columna de datos *HandZ* en la sección de valores y.



Si se ha realizado bien el proceso, se debería poder visualizar el gráfico de trayectorias correctamente.

ANEXO 4: Código Fuente de Juego del Ratón

UserMovement:

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

/// <summary>
/// Class Name: UserMovement
/// Author: David Matilla Plaza
/// Function: Control the user object position
/// </summary>
public class UserMovement : MonoBehaviour
{
    //Class variable
    private Collider[] _colliders;

    /// <summary>
    /// Called before the first frame. It initialize the object position
    /// </summary>
    private void Start()
    {
        transform.position = new Vector3(0, 5, 0);
    }

    /// <summary>
    /// Called every frame. It updates the posiiton of the object based
    /// on the hand mean position
    /// </summary>
    private void Update()
    {
        Vector3 handPosition = GetHandMeanPosition();
        if((!float.IsNaN(handPosition.x) ||
            !float.IsNaN(handPosition.y) ||
            !float.IsNaN(handPosition.z)))
        {
            transform.position = new Vector3(
                handPosition.x, 5, handPosition.z);
        }
    }

    /// <summary>
    /// This function get the position of the hand landmarks and
    /// calculate the mean position of the hand
    /// </summary>
    /// <returns>The return is the mean position Vector3 </returns>
    private Vector3 GetHandMeanPosition()
    {
        GameObject[] objectsInLayer = GetObjectsInLayer();
        (Vector3 positionSum, int nPos) = GetSummationOfPositions(
                                                objectsInLayer);

        Vector3 meanPosition = new Vector3(positionSum.x / (float)nPos
            , positionSum.y / (float)nPos, positionSum.z / (float)nPos);

        return meanPosition;
    }
```

```
        /// <summary>
        /// Get the objects in the landmark layer and append them to a list
        /// </summary>
        /// <returns>The list of hand landmarks</returns>
        private GameObject[] GetObjectsInLayer()
        {
            LayerMask layer;
            int layerNumber = LayerMask.NameToLayer("Landmark");
            layer = 1 << layerNumber;
            _colliders = Physics.OverlapSphere(
                Vector3.zero, Mathf.Infinity, layer);
            GameObject[] objectsInLayer = new GameObject[_colliders.Length];
            for (int i = 0; i < _colliders.Length; i++)
            {
                objectsInLayer[i] = _colliders[i].gameObject;
            }
            return objectsInLayer;
        }

        /// <summary>
        /// Get the summation of all landmarks position vector and count
        /// the total number of objects in the list
        /// </summary>
        /// <param name="objectsInLayer">The list of hand landmarks</param>
        /// <returns>Summation vector of positions and number of
        /// landmark objects used for the hand</returns>
        private (Vector3, int) GetSummationOfPositions(
            GameObject[] objectsInLayer)
        {
            Vector3[] handPositions = new Vector3[_colliders.Length];
            Vector3 positionSum = new Vector3(0f, 0f, 0f);
            int nPos = 0;

            foreach (GameObject obj in objectsInLayer)
            {
                Vector3 position = obj.transform.position;
                if (position != new Vector3(-106f, 5f, 64f))
                {
                    handPositions[nPos] = position;
                    positionSum += position;
                    nPos = nPos + 1;
                    Debug.Log("Posicion encontrada " + position);
                }
            }
            return (positionSum, nPos);
        }
    }
```

MouseMovement:

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

/// <summary>
/// Class Name: MouseMovement
/// Author: David Matilla Plaza
/// Function: Control the mouse position
/// </summary>
public class MouseMovement : MonoBehaviour
{
    //class Variables
    private GameObject _userObject;
    private Transform _objectToRunAwayFrom;
    private Transform _rotationObject;
    private float _factorSeparacion = 10f;
    private float _distanceThreshold = 30f;
    private float _distanceToCornerThreshold = 5f;
    private float _moveSpeed = 20f;
    private float _runTime = 5f;
    private string _heightVariableName = "RectSize";
    private float _timer = 0f;
    private float _rectSize;
    private float _width = 100f;
    private float _posZHigh;
    private float _posZLow;
    private float _posXRight;
    private float _posXLeft;
    private Vector3 _lastDirection = new Vector3(0f, 0f, 0f);
    private int _cornered;
    private float _distanceToOtherObject;
    private const string _childName = "Mouse";
    private const float _mouseWidth = 5f;
    private const string _distanceToUserVariableName = "DistanceToUser";

    /// <summary>
    ///Called when the object is enabled. It find the user object and
    ///initialize all the variables
    /// </summary>
    void OnEnable()
    {
        FindUser();
        InitializeVariables();
    }

    /// <summary>
    /// Called every frame. It calculate the distance to the user
    /// and act based on that
    /// </summary>
    void Update()
    {
        _distanceToOtherObject = DistanceCalc();
        PlayerPrefs.SetFloat(
            _distanceToUserVariableName, _distanceToOtherObject);
        if (_distanceToOtherObject < _distanceThreshold)
        {
            _timer = 0f;
            Escape();
        }
        else
        {
```

```
            _timer += Time.deltaTime;
            TeaseUser();
        }
    }

    /// <summary>
    /// Calculate the distance between the mouse and the user
    /// </summary>
    /// <returns>distance float type</returns>
    private float DistanceCalc()
    {
        float distance = Vector3.Distance(
            transform.position, _objectToRunAwayFrom.position);
        Debug.Log("La distancia entre los objetos es: " + distance);

        return distance;
    }

    /// <summary>
    /// This function increase the value of move speed based on
    /// the distance between the mouse and the user
    /// </summary>
    /// <param name="x"></param>
    /// <returns></returns>
    private float DistanceFactor(float x)
    {
        float resultado = Mathf.Pow((1f / 6f), (x - 1f));
        return resultado;
    }

    /// <summary>
    /// Finds the user object in scene an save it in a class variable
    /// </summary>
    private void FindUser()
    {
        _userObject = GameObject.Find("UserObject");
        if (_userObject != null)
        {
            _objectToRunAwayFrom = _userObject.transform;
        }
        else
        {
            Debug.LogError("Couldn't find gameObject named 'UserObject'");
        }
    }

    /// <summary>
    /// Initialize the class variables
    /// </summary>
    private void InitializeVariables()
    {
        _rectSize = PlayerPrefs.GetFloat(_heightVariableName, 0f);
        PlayerPrefs.SetFloat(_distanceToUserVariableName, 0f);
        _posZHigh = (_rectSize / 100f) * 65.656f - 27.096f;
        _posZLow = -39f;
        _posXRight = ((_rectSize - 50f) / 50f) * 31.25f + 64.75f;
        _posXLeft = -_posXRight;
        transform.position = new Vector3(0f, 5f, _posZHigh - 15);
        _distanceToOtherObject = DistanceCalc();
        _cornered = 4;
        _rotationObject = transform.Find(_childName);
    }
```

```
/// <summary>
/// Called when the user si further than x distance, the mouse
/// would tease the user by getting closer to the user object
/// </summary>
private void TeaseUser()
{
    if (_timer > 5f)
    {
        Vector3 newDirection = (
            _objectToRunAwayFrom.transform.position -
            transform.position);
        LookAtTheMovementDirection(newDirection);
        transform.Translate(
            newDirection.normalized * _moveSpeed * Time.deltaTime);
    }
}

/// <summary>
/// When the mouse is cornered, the variable "_cornered" takes a
/// value based on witch corner is in
/// </summary>
private void CheckIfCornered()
{
    float XDis, ZDist;
    (XDis, ZDist) = CalcMaxDistToBeCornered();

    if((_objectToRunAwayFrom.position.x < (_posXLeft + XDis))
        && (_objectToRunAwayFrom.position.z > (_posZHigh - ZDist)))
    {
        //Upper left corner
        _cornered = 0;
    }
    else if((_objectToRunAwayFrom.position.x < (_posXLeft + XDis))
        && (_objectToRunAwayFrom.position.z < (_posZLow + ZDist)))
    {
        //Bottom left corner
        _cornered = 1;
    }
    else if((_objectToRunAwayFrom.position.x > (_posXRight - XDis))
        && (_objectToRunAwayFrom.position.z < (_posZLow + ZDist)))
    {
        //Bottom right corner
        _cornered = 2;
    }
    else if(_objectToRunAwayFrom.position.x > (_posXRight - XDis)
        && (_objectToRunAwayFrom.position.z > (_posZHigh - ZDist)))
    {
        //Upper right corner
        _cornered = 3;
    }
    else
    {
        //Not cornered
        _cornered = 4;
    }
}


/// <summary>
/// When the mouse closer than x distannce, it must scape. Based on
/// the value of cornered it goes one or another direction
/// </summary>
private void Escape()
{
```

```
        Vector3 newDirection = (transform.position -
            _objectToRunAwayFrom.transform.position);
        Vector3 newPosition = transform.position +
            (transform.position -
            _objectToRunAwayFrom.transform.position).normalized *
            _moveSpeed *
            Time.deltaTime;
        float futureDistance = Vector3.Distance(
            _objectToRunAwayFrom.transform.position, newPosition);
        CheckIfCornered();

        if ((_cornered < 4) && _distanceToOtherObject < 30)
        {
           newDirection = DirectionToEscapeWhenCornered();
        }
        else if((_cornered < 4) && _distanceToOtherObject > 30)
        {
           _cornered = 4;
        }
        else
        {
           newDirection = NewParametersIfNewPositionOutOfBounds(
               newDirection, newPosition, futureDistance);
        }
        newPosition = transform.position
            + newDirection.normalized * _moveSpeed * Time.deltaTime;
        futureDistance = Vector3.Distance(
            _objectToRunAwayFrom.transform.position, newPosition);
        LookAtTheMovementDirection(newDirection);
        transform.Translate(newDirection.normalized *
            DistanceFactor(futureDistance / _distanceThreshold) *
            _moveSpeed * Time.deltaTime);
    }

    /// <summary>
    /// When the mouse is cornered, it goes in the same direction,
    /// escaping from the corner
    /// </summary>
    /// <returns>Move direction in Vector3</returns>
    private Vector3 DirectionToEscapeWhenCornered()
    {
        Vector3 newDirection = new Vector3(0f, 0f, 0f);
        switch (_cornered)
        {
            case 0:
                if ((_objectToRunAwayFrom.position.x - (_posXLeft)) <
                    (_posZHigh - _objectToRunAwayFrom.position.z))
                {
                    //The user is nearer to the side
                    newDirection = new Vector3(1f, 0f, 0f);
                }
                else { newDirection = new Vector3(0f, 0f, -1f); }
                break;
            case 1:
                if ((_objectToRunAwayFrom.position.x - (_posXLeft)) <
                    (_objectToRunAwayFrom.position.z - _posZLow))
                {
                    //The user is nearer to the side
                    newDirection = new Vector3(1f, 0f, 0f);
                }
                else { newDirection = new Vector3(0f, 0f, 1f); }
                break;
            case 2:
                if ((_posXRight - _objectToRunAwayFrom.position.x) <
```

```
                (_objectToRunAwayFrom.position.z - _posZLow))
            {
                //The user is nearer the side
                newDirection = new Vector3(-1f, 0f, 0f);
            }
            else { newDirection = new Vector3(0f, 0f, 1f); }
            break;
        case 3:
            if ((_posXRight - _objectToRunAwayFrom.position.x) <
                (_posZHigh - _objectToRunAwayFrom.position.z))
            {
                //The user is nearer to the side
                newDirection = new Vector3(-1f, 0f, 0f);
            }
            else { newDirection = new Vector3(0f, 0f, -1f); }
            break;
    }
    return newDirection;
}

/// <summary>
/// When the mouse is not cornered, there can be two cases of study.
///
/// When the new position is not out of the space bounds. In this
/// case the mouse go that direction
///
/// When the new position is out of the space bounds. In this case
/// the mouse have to look around and go to the furthest position
/// inside the bounds
/// </summary>
/// <param name="newDirection">Posible new direction</param>
/// <param name="newPosition">Posible new position</param>
/// <param name="futureDistance">Posible new distance</param>
/// <returns>New direction Vector3</returns>
private Vector3 NewParametersIfNewPositionOutOfBounds(
    Vector3 newDirection, Vector3 newPosition, float futureDistance)
{
    if (newPosition.x < _posXLeft ||
        newPosition.x > _posXRight ||
        newPosition.z < _posZLow ||
        newPosition.z > _posZHigh)
    {
        // If the new position is outside the zone,
        // move in the direction that is the furthest
        Vector3 farthestDirection = Vector3.zero;
        float farthestDistance = 0f;
        Vector3 farthestPosition = Vector3.zero;

        for (int i = 0; i < 360; i += 2)
        {
            Vector3 direction = new Vector3(
                Mathf.Cos(i * Mathf.Deg2Rad),
                0f,
                Mathf.Sin(i * Mathf.Deg2Rad));
            Vector3 testPosition = transform.position +
                direction.normalized * _moveSpeed * Time.deltaTime;
            float testDistance = Vector3.Distance(
                testPosition, _objectToRunAwayFrom.transform.position);

            if (testDistance > farthestDistance &&
                testPosition.x >= _posXLeft &&
                testPosition.x <= _posXRight &&
                testPosition.z >= _posZLow &&
                testPosition.z <= _posZHigh)
```

```
                {
                    farthestDistance = testDistance;
                    farthestDirection = direction;
                    farthestPosition = testPosition;
                }
            }
            return farthestDirection;
        }
        else
        {
            return newDirection;
        }
    }

    /// <summary>
    /// Called before the mouse move, it rotate the image for the mouse
    /// to look the direction is moving
    /// </summary>
    /// <param name="newDirection">New direction Vector3</param>
    private void LookAtTheMovementDirection(Vector3 newDirection)
    {
        Quaternion targetRotation = Quaternion.LookRotation(
            newDirection);
        _rotationObject.rotation = targetRotation;
    }

    /// <summary>
    /// It calculate the distance to the corner that make the mouse
    /// be cornered
    /// </summary>
    /// <returns>X,Z distance to corner float</returns>
    private (float XDistance, float ZDistance) CalcMaxDistToBeCornered()
    {
        float XDistance, ZDistance;
        XDistance = (_posXRight - _posXLeft) * 0.25f;
        ZDistance = (_posZHigh - _posZLow) * 0.3f;
        return (XDistance, ZDistance);
    }
}
```

DataCollectorMouse:

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

/// <summary>
/// Class Name: DataCollectorMouse
/// Author: David Matilla Plaza
/// Function: Save useful data of the game in a .csv file
/// </summary>
public class DataCollectorMouse : MonoBehaviour
{

    private LayerMask _layer;
    private const float Period = 0.1f;
    private float _time;
    private float _currentTime;
    private StreamWriter _file;
    private string _filePath = "";
    private Collider[] _colliders;
    private const string _distanceToUserVariableName = "DistanceToUser";

    /// <summary>
    /// Called when the object is enabled. Initialize the variables and
    /// create and opens the .csv
    /// </summary>
    private void OnEnable()
    {
        _currentTime = 0f;
        _time = 0f;
        CreateAndOpenCSV();
    }

    /// <summary>
    /// Called every frame. Updates the current time of the game and
    /// when the period has passed, we save de info in the .csv
    /// </summary>
    private void Update()
    {
        _currentTime += Time.deltaTime;
        _time += Time.deltaTime;

        if (_time > Period)
        {
            SaveOnCSV();
            _time = 0f;
        }
    }

    /// <summary>
    /// Called when the data save period has passed. It makes a backup
    /// of the current data in a line of the .csv file
    /// </summary>
    private void SaveOnCSV()
    {
        Vector3 handPosition = GetHandMeanPosition();
        float distanceToUser = PlayerPrefs.GetFloat(
            _distanceToUserVariableName, 0f);

        if ((!float.IsNaN(handPosition.x) ||
            !float.IsNaN(handPosition.y) ||
```

```
            !float.IsNaN(handPosition.z)))
        {
            _file.WriteLine($"{_currentTime};{handPosition.x};" +
                $"{handPosition.z};{distanceToUser}");
        }
    }


    /// <summary>
    /// This function get the position of the hand landmarks and
    /// calculate the mean position of the hand
    /// </summary>
    /// <returns>The return is the mean position Vector3 </returns>
    private Vector3 GetHandMeanPosition()
    {
        GameObject[] objectsInLayer = GetObjectsInLayer();
        (Vector3 positionSum, int nPos) = GetSummationOfPositions(
            objectsInLayer);

        Vector3 meanPosition = new Vector3(positionSum.x / (float)nPos
            , positionSum.y / (float)nPos, positionSum.z / (float)nPos);

        return meanPosition;
    }

    /// <summary>
    /// Get the objects in the landmark layer and append them to a list
    /// </summary>
    /// <returns>The list of hand landmarks</returns>
    private GameObject[] GetObjectsInLayer()
    {
        LayerMask layer;
        int layerNumber = LayerMask.NameToLayer("Landmark");
        layer = 1 << layerNumber;
        _colliders = Physics.OverlapSphere(
            Vector3.zero, Mathf.Infinity, layer);

        GameObject[] objectsInLayer = new GameObject[_colliders.Length];
        for (int i = 0; i < _colliders.Length; i++)
        {
            objectsInLayer[i] = _colliders[i].gameObject;
        }
        return objectsInLayer;
    }

    /// <summary>
    /// Get the summation of all landmarks position vector and count
    /// the total number of objects in the list
    /// </summary>
    /// <param name="objectsInLayer">The list of hand landmarks</param>
    /// <returns>Summation vector of positions and number of
    /// landmark objects used for the hand</returns>
    private (Vector3, int) GetSummationOfPositions(
        GameObject[] objectsInLayer)
    {
        Vector3[] handPositions = new Vector3[_colliders.Length];
        Vector3 positionSum = new Vector3(0f, 0f, 0f);
        int nPos = 0;

        foreach (GameObject obj in objectsInLayer)
        {
            Vector3 position = obj.transform.position;
            if (position != new Vector3(-106f, 5f, 64f))
            {
```

```
                handPositions[nPos] = position;
                positionSum += position;
                nPos = nPos + 1;
                Debug.Log("Posicion encontrada " + position);
            }
        }
        return (positionSum, nPos);
    }

    /// <summary>
    /// This function is called when the object is enabled and it
    /// creates and open a .csv with the current date in the path name
    /// </summary>
    private void CreateAndOpenCSV()
    {
        string folder = Path.Combine(
            Application.dataPath, "DataCollectionMouse");

        if (!Directory.Exists(folder))
        {
            Directory.CreateDirectory(folder);
        }
        string actualDate = DateTime.Now.ToString(
            "yyyy-MM-dd-HH-mm-ss");
        string fileName = actualDate + ".csv";
        _filePath = Path.Combine(folder, fileName);

        bool fileExist = File.Exists(_filePath);

        _file = new StreamWriter(_filePath, true);

        if (!fileExist)
        {
            _file.WriteLine($"Date;{actualDate}");
            _file.WriteLine($"Time;HandX;HandZ;DistanceToUser");
        }
    }

    /// <summary>
    /// Called when the game is finished. It close the .csv file
    /// </summary>
    public void CloseFile()
    {
        bool fileExist = File.Exists(_filePath);
        if (fileExist)
        {
            _file.Close();
            gameObject.SetActive(false);
        }
    }
}
```

ANEXO 5: Lista de librerías del entorno de Python.

absl-py	1.0.0
attrs	21.2.0
blas	1.0
brotli	1.0.9
brotli-bin	1.0.9
ca-certificates	2023.05.30
certifi	2023.5.7
cycler	0.11.0
fonttools	4.28.2
freetype	2.12.1
giflib	5.2.1
intel-openmp	2021.4.0
jpeg	9e
kiwisolver	1.3.2
lerc	3.0
libbrotlicommon	1.0.9
libbrotlidec	1.0.9
libbrotlienc	1.0.9
libdeflate	1.17
libpng	1.6.39
libtiff	4.5.0
libwebp	1.2.4
libwebp-base	1.2.4
lz4-c	1.9.4
matplotlib	3.5.0
matplotlib-base	3.4.3

mediapipe	0.8.9
mkl	2021.4.0
mkl-service	2.4.0
mkl_fft	1.3.1
mkl_random	1.2.2
munkres	1.1.4
numpy	1.21.4
numpy-base	1.23.5
opencv-contrib-python	4.6.0.66
opencv-python	4.6.0.66
openssl	1.1.1u
packaging	21.3
pillow	8.4.0
pip	21.2.4
powershell_shortcut	0.0.1
protobuf	3.19.1
pyparsing	3.0.6
python	3.9.7
python-dateutil	2.8.2
pyvirtualcam	0.8.0
pyzmq	22.3.0
setuptools	58.0.4
setuptools-scm	6.3.2
six	1.16.0
sqlite	3.36.0
tk	8.6.12
tomli	1.2.2
tornado	6.2
tzdata	2021e
vc	14.2

vs2015_runtime	14.27.29016
wheel	º0.37.0
wincertstore	0.2
xz	5.2.10
zlib	1.2.13
zstd	1.5.2