Universitat de les Illes Balears

TRABAJO DE FIN DE GRADO

MUSEU DEL VIDEOJOC EN REALITAT VIRTUAL

Miguel Ángel García Vich

Ingeniería en Informática

Escola Politècnica Superior

Any acadèmic 2021-22

MUSEU DEL VIDEOJOC EN REALITAT VIRTUAL

Miguel Ángel García Vich

Trabajo de Fin de Grado

Escola Politècnica Superior

Universitat de les Illes Balears

Any acadèmic 2021-22

Paraules clau del treball: TFG, memoria, Videojuegos, Realidad Virtual, Museo

Tutor: Antoni Oliver Tomàs, Antonio Bibiloni Coll

Gracias a todos los profesores que me han ayudado a llegar a este punto de mis estudios, especialmente a mis tutores Antoni Oliver Tomàs y Antonio Bibiloni Coll, por ayudarme y supervisar este proyecto. Gracias a mi familia y en especial a mi mejor amigo Josep, por acompañarme durante el recorrido y darme otro punto de vista. Y por último, darle las gracias a todos aquellos que se ofrecieron voluntarios para realizar las pruebas de usuario del proyecto.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRÓNIMOS

TFG Trabajo Final de Grado

RV Realidad Virtual

UIB Universidad de las Islas Baleares

PUN Photon Unity Networking

RPC Remote Procedural Call

IU Interfaz de Usuario

CC Creative Commons

SUS System Usability Scale

Resumen

Este proyecto se enmarca en el ámbito del desarrollo de videojuegos y la realidad virtual. En concreto, hemos desarrollado una aplicación multidispositivo y multijugador que funcione para ordenadores y dispositivos de realidad virtual.

El resultado del proyecto ha sido una aplicación donde se permite la exploración de un museo al cual se pueden conectar hasta un máximo de diez jugadores, con consolas y videojuegos de cinco compañías distintas. De estas consolas, se puede interactuar para conocer información y ver producciones audiovisuales sobre videojuegos de ellas, e incluso, poder jugar a un videojuego semejante a los que encontraríamos en estas.

La idea detrás de esta aplicación viene motivada por el interés en la historia de los videojuegos y el sentimiento de inmersión y nostalgia que puede producir la realidad virtual en este campo. Este tipo de mercado no ha sido muy explorado, por lo que es una buena fuente de investigación y desarrollo.

La aplicación ha sido desarrollada con el motor de videojuegos Unity, y se ha optado por utilizar las librerías OpenXR para la implementación de la realidad virtual, así como Photon como sistema de networking que ayuda con el matchmaking y la comunicación entre usuarios.

Para comprobar los resultados de la aplicación y obtener feedback de esta, se han llevado a cabo pruebas de usuario, las cuales hemos analizado usando un test SUS adaptado a la realidad virtual, y mediante un test ANOVA, comprobar si el tiempo transcurrido en la realización de las tareas entre dispositivos es significativo o no.

Han sido varias las conclusiones de este proyecto, predominando el sentimiento de nostalgia e inmersión que buscábamos y elaborando las posibles ampliaciones a futuro.

CAPÍTULO 1

INTRODUCCIÓN

1.1. Contextualización

En los últimos años, uno de los medios de entretenimiento que ha ido creciendo ha sido la Realidad Virtual (RV). Si bien la RV se remonta a principios del siglo pasado, no ha sido hasta la última década que se ha visto su crecimiento tanto a nivel comercial como tecnológico [1].

El punto de inflexión se considera que fueron las Oculus Rift [2], las cuales anunciaron en 2012 [1]. A partir de este punto, y con la compra de Oculus VR por parte de Facebook, otras empresas entraron en el mercado de los dispositivos de RV. Empresas como Sony con sus Sony Playstation VR [3], Samsung con Gear VR [4] y Valve con las HTC Vive [5] o SteamVR [6], han desarrollado nuevas tecnologías que se van actualizando y remodelando con el paso del tiempo.

Televisión, cine, medicina, educación, muchos han sido los sectores que se han adaptado a estas nuevas tecnologías [7]. El sector que más se ha desarrollado ha sido el del entretenimiento, sobre todo el de los videojuegos [8]. Los dispositivos de RV proporcionan al usuario una mayor sensación de inmersión en cualquiera de las áreas que hemos mencionado, pero es la del sector del entretenimiento en la que más destaca, haciendo que los usuarios tiendan más a buscar estas experiencias.

A día de hoy hay una gran oferta tanto de dispositivos de RV como de contenido para estos, y las previsiones de futuro indican que esto seguirá aumentando [9].

En el contexto del desarrollo de videojuegos, muchos frameworks y motores de videojuegos se han adaptado para dar soporte a la realidad virtual, otros han sido creados especialmente para ello. Algunos de los que se han adaptado son Unity [10], Unreal Engine [11] o Lumberyard [12], mientras que frameworks como A-Frame [13] o React VR [14] se han creado especialmente para ello, sobre todo en el desarrollo web.

Uno de los mencionados, Unity, ha crecido en popularidad en el desarrollo de RV gracias a su sencillez y comunidad, algo que veremos más adelante en este documento.

1.2. Motivación

La idea detrás de este proyecto viene motivada por el interés en la historia de los videojuegos y en la creación de algún lugar donde distintos jugadores se puedan reunir para disfrutar juntos de una experiencia inmersiva sobre el pasado y presente de los videojuegos. Por ello, un museo de la historia de los videojuegos, con capacidad para el multijugador y el uso de la RV para la inmersión, nos pareció la mejor manera para transmitir esta idea.

Investigando el mercado de este tipo de museos, de los cuales hablaremos en el correspondiente apartado de este documento, nos damos cuenta de que aquellos con una idea similar a la nuestra no ofrecen todo lo que ideamos ofrecer nosotros, y es por eso que queremos desarrollar este producto.

1.3. Objetivo

El objetivo de este proyecto es desarrollar una aplicación multidispositivo, que funcione para ordenadores y dispositivos de realidad virtual, y que permita la exploración de un museo de videojuegos en realidad virtual. Además, la aplicación cuenta con dos modalidades: una para un solo jugador y otra para más de un jugador. Los usuarios podrán interactuar con varias consolas, y cada una tendrá una de dos: o un videojuego parecido a los representativos de la consola o una producción multimedia relacionada con un videojuego de la consola.

1.4. Alcance del proyecto

La aplicación se tiene que poder instalar correctamente en Windows 10, y reconocer los dispositivos de RV de la marca Oculus, Oculus Quest. El museo tiene que poder soportar la conexión de mínimo un jugador y máximo de diez jugadores. De todas las consolas representadas dentro del museo, de dos a seis de ellas deben tener implementado un videojuego. De estos videojuegos, por lo menos dos de ellos deben poder jugarse en modo de dos jugadores. Las consolas de videojuegos que se representarán son las de las marcas Atari, Sony, Nintendo, Microsoft (XBOX) y Sega. Cada consola estará representada por un modelo 3D, acompañadas del modelo de un monitor o televisor en 3D. El proyecto se entregará antes de julio de 2022. En este proyecto no se prevé aumentar el número de videojuegos que se podrá jugar ni las consolas de videojuegos que se van a representar

A lo largo del presente documento vamos a comentar la situación actual del mercado de museos digitales, las diferentes tecnologías que existen y que usaremos, cuál es la gestión del proyecto, el análisis de los usuarios, el diseño e implementación de la aplicación, las pruebas de usuario que se han realizado, y por último, unas conclusiones del producto y del proyecto.

CAPÍTULO 2

ESTADO DEL ARTE

En este capítulo haremos un estudio de la situación actual del mercado de los museos digitales, donde los dividiremos en varias categorías y analizaremos cada una de ellas. También haremos un análisis y selección de las tecnologías disponibles para desarrollar nuestro producto.

2.1. Estudio del mercado

El mercado de experiencias audiovisuales en RV está en constante expansión. Para saber cuáles son las características que son tendencia en el resto de los museos digitales, nos fijaremos para determinar que es lo que están aportando ellos al mercado, aspectos positivos y negativos, y que podremos aportar nosotros que el resto no ofrezcan o que aspectos utilizaremos que también utilizan ellos. Para ello, dividiremos los museos que hemos analizado en cuatro grupos, que estarán relacionados con la temática que tiene nuestro producto o al público objetivo que compartiremos. Empezaremos yendo desde una temática más general, como podría ser un museo de acceso virtual y online, hasta un museo en RV que tengan que ver con videojuegos.

2.1.1. Museos digitales

Comencemos con los museos digitales de algunos de los museos más famosos del mundo. No todos ellos ofrecen las mismas experiencias, pero si podemos ver que la mayoría ofrecen experiencias similares. La mayoría de ellos, como por ejemplo The J.Paul Getty Museum [15], el Guggenheim Museum [16] o el Musée d'Orsay [17], se apoyan en la tecnología de Arts & Culture de Google [18], que utiliza una versión de street maps adaptada para ver el interior de estos museos. Una de las desventajas que tiene esta opción, es que a la hora de navegar por el interior del museo y ver los distintos cuadros u objetos, las descripciones o tarjetas informativas de estos se ven difuminadas o son dificultosos de leer.

Un museo que se destaca del resto es el British Museum [19], que cuenta con una web donde, mediante una línea temporal en 3D, muestra unos puntos repartidos en el tiempo y en distintos continentes, y al hacer clic en estos puntos se abre una pestaña mostrando una imagen del objeto o la obra, con una descripción de este, tanto en texto como en audio, su lugar de origen, que además se puede abrir en Google Maps y otras funcionalidades. Podemos ver la página principal en la figura 2.1.

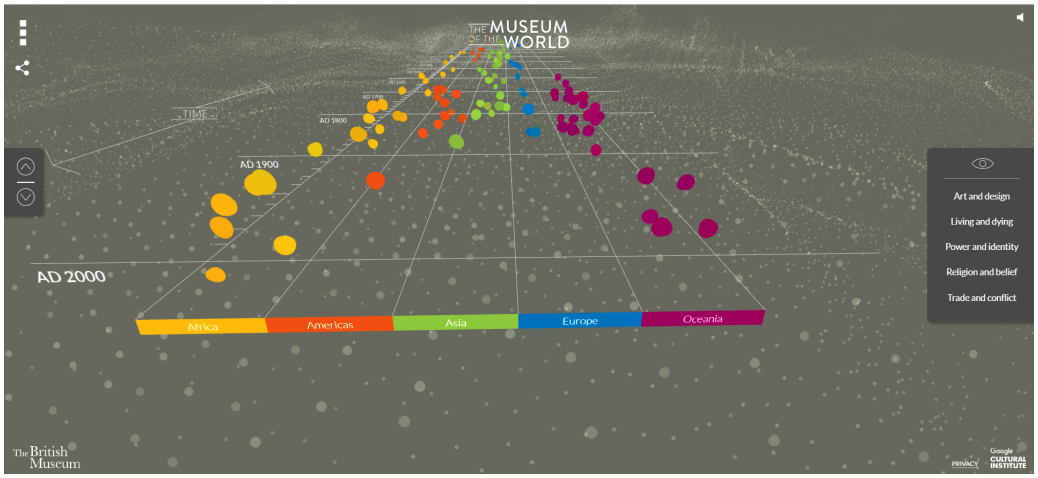

Figura 2.1: Pantalla principal British Museum

Por último, el museo de historia natural de Berlin [20] y Londres [21], ofrecen una experiencia en video 360º, que narra unos 5 minutos de un pequeño tour con modelos de animales 3D. Estos videos tienen opción de visualizarlos en RV, pero no destacan en nada en concreto, por lo que no los mencionaremos en el siguiente apartado.

Al observar estos museos, concluimos que no nos interesa implementar nada de los que estos ofrecen, dado que no compartimos la temática de nuestro producto, y además las tecnologías que utilizan no son interactivas y no se pueden usar para cumplir nuestras ideas.

2.1.2. Museos en RV

Hay algunos museos que no tienen versión física y que se han creado únicamente para experimentar lo que sería visitar un museo en RV. Uno de los ejemplos de museo que ha hecho esto, es el Museum of Fine Art [22]. Este museo contiene pocas esculturas u obras, pero se puede navegar libremente por el espacio tridimensional, utilizando los controles de los dispositivos de realidad virtual, pudiéndote acercar también con la cámara como si estuvieras a un palmo de la escultura o placa descriptiva, pudiendo apreciar los detalles y poder leer correctamente la descripción.

Otro ejemplo de museo en RV es el MAC Museum VR [23]. En este pequeño museo tenemos un recinto donde hay distintos dispositivos Macintosh producidos a lo largo del tiempo. Podemos interactuar con unos modelos 2D y nos muestran la información sobre ese dispositivo y un vídeo comercial de cuando se lanzaron al mercado. Podemos ver un ejemplo en la figura 2.2. Pese a su sencillez, cumple su cometido. Mostrar todos

los modelos Macintosh e información sobre ellos, semejante a lo que intentamos hacer nosotros pero con las consolas de videojuegos.

Figura 2.2: Entorno Mac Museum VR

Al igual que el apartado anterior, al no compartir la temática del museo no podemos quedarnos con muchas de las cosas que estos museos implementan. Una opción interesante que tienen estos museos es que cuando interactúas utilizando los inputs de la RV con alguna de las obras o ítem, estos te muestran información sobre ellos. Haremos algo similar a estos museos, ya que el usuario podrá interactuar con las consolas y podrán ver la información de estas.

2.1.3. Museos digitales de videojuegos

Este punto y el siguiente son los que más pueden afectar a nuestro producto, ya que la temática de los museos es la misma que nuestro sector de interés. Hay varias maneras de enfocar estos museos. Uno de ellos son los museos referentes a la historia de los videojuegos, y otros de ellos se refieren a un conjunto de videojuegos muy específico, como sería el museo de una compañía de videojuegos o incluso de algún desarrollador indie.

Hay muy pocos museos relacionados con la historia de los videojuegos que ofrezcan un tour online. Los dos que destacaremos, y de los pocos que existen, son el National Videogame Museum y el Museo Arcade Vintage [24].

Respecto al Museo Arcade Vintage, como su nombre indica, es un museo de videoconsolas arcade. Las videoconsolas arcade fueron realmente las pioneras en la historia de los videojuegos, las cuales más tarde dieron paso a las videoconsolas domésticas. Para poder realizar el tour online de este museo se requiere de una donación simbólica, de un mínimo de un céntimo. Cuando se consigue entrar nos encontramos con una visita del estilo de Google Street View. Podemos navegar por el establecimiento e ir leyendo los distintos carteles que hay colgados en las paredes. Aparte de esto, sobre cada consola arcade hay un globo de información, y que si le hacemos clic se abre una pestaña mostrándonos el juego de esa consola, así como las especificaciones técnicas

de estas. Podemos ver el entorno de este museo en la figura 2.3. Este estilo es parecido a lo que haremos en nuestro juego, donde mostraremos las consolas con algún vídeo de un videojuego y un poco de información sobre la consola.

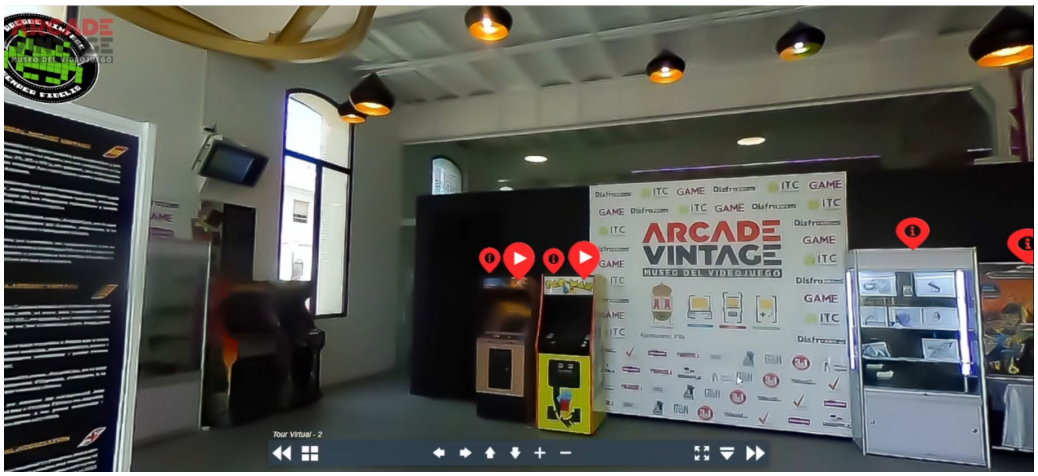

Figura 2.3: Entorno Museo Arcade Vintage

El otro ejemplo de museo de videojuegos es el National Videogame Museum. Este museo no tiene un tour online de acceso directo, pero sí podemos navegar por él gracias a que, con la ayuda del videojuego Doom 2 y utilizando la herramienta de mods GZDoomBuilder, podemos ver todo el museo a través de este videojuego y de forma oficial [25]. Este tour, quitando la interactividad que tiene el videojuego Doom 2 fuera de este espacio, carece totalmente de interacción alguna, únicamente podemos desplazarnos y ver el modelado de las consolas y carteles. Otra manera de ver este museo es gracias a usuarios que han grabado su propio tour en un vídeo de 360º, pero al no ser fuentes oficiales de estos museos simplemente los mencionamos [26].

Ahora pasamos a hablar de dos museos de videojuegos relacionados con algún autor o compañía de videojuegos.

El primero de ellos es Devolverland Expo [27]. Se trata de un museo de la compañía Devolver Digital, conocido por ser la distribuidora de videojuegos como Fall Guys, Hotline Miami o The Talos Principle. Devolverland Expo es un gran escenario 3D, cuyo objetivo es hacer publicidad de los juegos más destacados de su compañía a través de un videojuego interactivo, donde podemos navegar por el escenario y pasar por distintos niveles con temáticas igual que los videojuegos que intentan publicitar. Es una buena forma de mostrar al usuario los videojuegos que ofrece la empresa de forma interactiva.

Por otra parte, y con la misma premisa de publicitar los videojuegos, tenemos el caso de Majorariatto Museum [28]. En este museo se exponen los juegos de la empresa indie Majorariatto, a cuyo desarrollador principal se le conoce bajo el seudónimo de Alva Majo. Se conoce Majorariatto por contar con videojuegos sencillos, pero que se han hecho muy virales en internet, sobre todo en redes sociales como Youtube y Twitch. Algunos de estos videojuegos son Pureya, Shipped, o el más reconocido, Majotori. En Majorariatto Museum, aparte de poder ver obras y capturas de los videojuegos de

Majorariatto, también hay un juego interactivo, que para completar habrá que recorrer todo el museo resolviendo distintos puzzles y actividades, premiando la curiosidad y la exploración. También da pie a la rejugabilidad, término que se utiliza para volver a jugar una vez has completado el juego, ya que el museo contiene distintos finales y cosas que hacer o que tal vez en la primera vez que se juega no se llega a ver.

De la parte de museos dedicados a la historia de los videojuegos, cogemos la inspiración de la temática de nuestro museo, y muchas de las consolas que se ven en estos museos las tendremos en nuestro museo. Por otra parte, de la interactividad nos fijamos en los museos que tratan sobre empresas de videojuegos, ya que estas ofrecen una mayor interactividad, haciendo que el usuario tienda a quedarse y volver a entrar al museo en vez de hacerlo una única vez. La idea que no tendremos en cuenta es la importancia en la interactividad sobre el museo en general en lugar de las exposiciones del museo. En nuestro museo nos centraremos en las exposiciones que son las consolas y los videojuegos sin darle mucha importancia a la interactividad externa a estas exposiciones.

2.1.4. Museos de videojuegos en RV

En este punto es donde deberíamos tener más influencia, ya que compartimos tanto temática como dispositivo principal, pero veremos que no hay tantos museos de videojuegos en RV.

Uno de los que destacaríamos es el Kingdom Hearts Virtual Reality Experience [29]. Se puede acceder a este museo a través de una aplicación de PS4 o PS5 y utilizando las PlayStation VR. En esta aplicación nos vemos inmersos en distintas experiencias audiovisuales relacionadas con la saga de videojuegos Kingdom Hearts.

Iremos viendo la evolución de la saga y podremos acceder a algunos de los mundos en los que se desarrollan los videojuegos y vivir distintas experiencias, desde un túnel que recorremos viendo toda la saga, estar al lado de los protagonistas mientras hablan, o coger la famosa llave espada y golpear monstruos como si la estuviéramos sujetando nosotros mismos. De este museo nos fijamos en el factor nostalgia que nosotros también buscamos, y lo usan de forma que los usuarios que ya hayan jugado a esta saga de videojuegos se introduzcan de nuevo en la saga viviendo una experiencia inmersiva, sirviendo también como publicidad para que nuevos usuarios compren los videojuegos de la saga.

Otra aplicación en la que nos fijamos es EmuVR [30], una aplicación que permite emular en un entorno de RV distintas consolas, desde la PS1 hasta la WII, por nombrar algunas. Podemos ver el entorno en el que se mueven los usuarios en la figura 2.4. Junto con las consolas, podemos jugar infinidad de juegos, pero para jugarlos tendremos que tener una copia de estos. La tecnología que utilizan para conseguir esto es libretro, de lo cual hablaremos en la sección de consideraciones previas. Al igual que mencionamos con la aplicación de Kingdom Hearts, EmuVR busca una inmersión total, donde el usuario sienta que ha vuelto a su infancia, jugando consolas antiguas, e incluso ver series y películas.

En conclusión para este último apartado, el aspecto interesante de estos museos es la inmersión que proporciona la RV. Nos fijaremos en como intentan transmitir la sensación de inmersión, como interactúa el usuario con las exposiciones y en la emoción nostálgica que estos transmiten.

Figura 2.4: Entorno proyecto Emu VR

Para resumir el apartado de los museos que hemos analizado, podemos ver la tabla 2.1, que muestra distintos atributos analizados de cada museo estudiado, así como la figura 2.5 que agrupa los museos mas destacados de los que hemos hablado.

Figura 2.5: Continuum digital - virtual y videojuegos - no videojeugos

2.2. Usuarios en el mercado y objetivos

Para empezar a plantear cuáles son los usuarios de nuestro producto, lo primero que deberíamos hacer es fijarnos en las edades mínimas o recomendables que tienen los dispositivos de RV, ya que estos usuarios junto a los de PC serán nuestros principales

Museos	Digital o RV	Tecnología	Interaccion	Precio
The J.Paul Getty Museum	Digital	Google Arts & Culture	Poca	Gratis
Guggenheim Museum	Digital	Google Arts & Culture	Poca	Gratis
Musée d'Orsay	Digital	Google Arts & Culture	Poca	Gratis
British Museum	Digital	WebGL	Media	Gratis
Museo de historia natural de Berlin	Ambos	Google Arts & Culture	Media	Gratis
Museo de historia natural de Londres	Ambos	Google Arts & Culture	Media	Gratis
Museum of Fine Art	RV	Unity	Media Alta	Gratis
MAC Museum VR	Ambos	A-Frame	Media Alta	Gratis
National Videogame Museum	Digital	Doom 2	Baja	Gratis
Museo Arcade Vintage	Digital	Google Street View	Media	Donativo
Devolverland Expo	Digital	Unreal Engine 4	Alta	Gratis
Majorariatto Museum	Digital	Unity	Alta	Gratis
Kingdom Hearts Virtual Reality Experience	RV	Unreal Engine 4	Alta	Gratis
EmuVR	RV	Libretro Retroarch	Alta	Inscripción

Tabla 2.1: Atributos museos

clientes. Según lo que informan algunas de las grandes marcas que ofrecen dispositivos de RV, como puede ser Oculus, con sus modelos Quest, Rift y Go, o Sony con sus PlayStation VR, estos recomiendan que los niños menores de 14 [31] y 12 [32] años respectivamente no utilicen sus dispositivos. Así mismo, estas empresas no ponen un límite superior en la edad, por lo que no debería haber problema con las personas mayores.

Una vez tenemos en cuenta la edad mínima de nuestros usuarios, que será de 14 años, vamos a ver cuál o cuáles son los grupos de personas que consideraremos nuestros usuarios.

La temática de nuestro producto es de desarrollar un museo donde se puedan experimentar con consolas y videojuegos desde que la industria se inició, sin olvidar las consolas más recientes. Esto significa que los usuarios principales serán aquellas personas que les interese la industria de los videojuegos. Dentro de este gran grupo podemos identificar dos grupos diferentes.

Uno de ellos es el grupo juvenil. Los adolescentes y los jóvenes-adultos han demostrado un aumento en el interés de la industria de los videojuegos a medida que han pasado los años desde sus inicios. Este aumento en el interés se debe, entre otros factores, a la entrada de los videojuegos en dispositivos móviles, y sobre todo estos

últimos dos años, donde la pandemia ha jugado un papel fundamental en el cambio del sector del entretenimiento, siendo el de los videojuegos de los más afectados y que ha incrementado enormemente en número de usuarios. [33]

El segundo grupo sería el de los adultos, que una vez vivieron las experiencias que ofrecían las consolas y los videojuegos hace años, y pueden rememorar estas experiencias de nuevo gracias a las ofertas del museo en RV. Por lo que estos últimos verían un gran atractivo por la nostalgia y por volver a incorporarse a la industria de los videojuegos si no habían continuado consumiéndola.

2.3. Tecnologías. Análisis y selección

Como en todos los proyectos y productos, hay muchas tecnologías que se pueden utilizar. Antes de desarrollar una aplicación o juego, conviene considerar como implementar la idea que se tiene y las opciones que hay para poder aplicar estas ideas al proyecto. En este apartado principalmente trataremos estas posibilidades: ¿Los videojuegos serán emulados o implementados? ¿Cómo implementamos el multijugador online? ¿Motores de videojuegos o A-Frame? ¿RV o digital?

Vamos a empezar tratando una de las cuestiones más importantes, uno de los núcleos del museo, y son los videojuegos que se podrán jugar en las consolas. A la hora de pensar como implementar los videojuegos nos planteamos dos opciones.

Una de ellas era, en la misma escena que el museo y lejos de este, crear cada videojuego de forma separada. La zona en la que transcurre el videojuego está grabada por una cámara de Unity, y el output de la cámara se plasma en una textura que pondremos a la pantalla del televisor de la consola. Cuando el jugador interactúa con la consola, se renderiza la zona del videojuego, y los controles del movimiento del jugador quedan bloqueados para que estos controlen el personaje del videojuego. Esta opción requiere el esfuerzo de desarrollar los videojuegos que se podrán jugar, pero estos serán pocos, no más de 6, y además no serán videojuegos completos, sino que se tratará de un nivel o una pantalla de lo que sería el videojuego completo.

Otra opción que llegamos a plantear, se trata de emular los videojuegos. Para conseguir emular estos juegos utilizaríamos la librería de Unity, SK.Libretro [34]. Esta librería permite cargar los núcleos de las consolas de videojuegos y las propias ROM de los videojuegos, para poder ejecutarlos y emularlos.

Las ventajas que tendría esta opción frente a la anterior, seria que no hace falta desarrollar los videojuegos, además que todas las consolas podrían tener un videojuego emulado, mostrando mayor diversidad e interactividad que si tuviéramos la mayoría de vídeos en las consolas y unos pocos videojuegos para interactuar. Pero esta opción tiene unos cuantos inconvenientes.

La mayoría de los inconvenientes vienen de la librería Libretro adaptada a Unity [35]. Esta librería, como hemos dicho antes, nos permite emular el videojuego en el núcleo de la consola. Los inconvenientes que más destacaremos son que esta librería tiene muy poca documentación, y si queremos saber como actúa cada función, habría que analizarla en profundidad. Otro de los inconvenientes es que, en parte, debido a la falta documentación, es difícil determinar la manera en la que se podría adaptar los inputs de los controles del dispositivo RV. Además, la implementación de varios

jugadores en el mismo videojuego y que puedan entrar y salir cuando quieran también se ve cuestionada por los inputs.

Otra cuestión es la compatibilidad de los núcleos de las consolas y de algunos videojuegos. Si quisiéramos utilizar un emulador de la PS3, esto sería difícil, ya que actualmente los emuladores de PS3 no funcionan a la perfección.

Por último, el gran inconveniente es que la emulación ha generado mucha polémica a lo largo de los años. Esta polémica reside en el hecho de que muchos de los emuladores son usados para emular copias de videojuegos que no son legales. Y aquí viene el punto por el cual no podríamos emplear los emuladores dentro del proyecto, al menos no de forma legal con todos los videojuegos.

La mayoría de los videojuegos están sujetos a una licencia que pertenece a una empresa. Para poder emular estos videojuegos, se tiene que tener una copia legal del videojuego y hacer una depuración hacia el emulador para poder hacerlo legalmente. Este factor seguramente no sería el caso de algunas de las consolas de primera generación, cuyas empresas ya no existen y su licencia se ha perdido, pero en consolas más actuales esto no se podría hacer, ya que entrarían en juego temas legales y de copyright. En el caso de que consiguiéramos una copia legal de los juegos y hacer una depuración para poder utilizarlo en los emuladores dentro del museo, habría que considerar también si es legal el hecho de que otros usuarios utilicen copias de los videojuegos de forma remota sin que estos tengan una copia legal ni nosotros tengamos una licencia que nos permita compartir estos.

Pasemos ahora a hablar de la o las tecnologías que utilizaremos para desarrollar el museo. Para ello, nos hemos centrado en analizar cuatro opciones: Unity, Unreal Engine, A-Frame y Google Arts & Culture (Street view).

Empezaremos descartando la opción de Google Arts & Culture, ya que esta opción es inviable porque tendríamos que construir el museo en la realidad y posteriormente capturar las imágenes del museo con una cámara especial. A más inri, la interactividad que buscábamos en nuestro proyecto se verá disminuida por las limitaciones de esta tecnología.

Descartada esta opción, hablaremos ahora de Unity y Unreal Engine, que son dos motores de videojuegos con editor incorporado.

Estos dos motores de videojuegos son similares, pero tienen sus ventajas y desventajas. Empezando por Unity, el editor es limpio y visualmente atractivo para los nuevos usuarios que no están acostumbrados a este tipo de editores. Hay una gran cantidad de documentación disponible, tanto así como numerosos tutoriales publicados tanto por usuarios como profesionales de la aplicación y también muchos foros e hilos de usuarios que resuelven cualquier duda que uno pueda tener. Tiene integración nativa para el desarrollo multiplataforma. Los videojuegos y aplicaciones de Unity se programan principalmente en C#, un lenguaje intuitivo y con muchos recursos para aprenderlo. Debido en parte al lenguaje de programación, al no tratar el programador con la gestión de memoria, utiliza un colector de basura, por lo que la memoria está menos optimizada. Unity es conocido por ser usado en pequeños videojuegos con poco presupuesto, gracias a sus assets y extensiones gratuitas.

Unreal Engine, parece estar usado únicamente para videojuegos con mucho presupuesto, pero lejos de este mito, también se usa en videojuegos de tamaño medio o hechos por desarrolladores indies. La documentación disponible es similar a la de

Unity, pero el número de tutoriales y foros con ayudas son menores. Tenemos control completo del engine, se puede acceder de forma gratuita a su código fuente y modificarlo mientras se siga la licencia de uso. Ofrece calidad y potencia en el apartado de iluminación y en la creación de shaders y materiales. Hace uso de C++ lo cual aporta flexibilidad y control, aunque este lenguaje es más complicado de usar sin experiencia. Tiene un menor número de assets y extensiones, y pocos de ellos son gratuitos.

Por último, nos encontramos con A-Frame, un framework web de código abierto para crear experiencias de realidad virtual. A-Frame utiliza html y javacript para desarrollar las aplicaciones y juegos, siendo familiar para diseñadores y desarrolladores web a la vez que cuenta con un editor similar a los de los motores de videojuegos. A-frame es relativamente nuevo, comenzando en 2015. A diferencia de los motores de videojuegos antes mencionados, A-Frame no tiene tanta documentación ni una comunidad tan activa como estos, por lo que podría ser complicado iniciarse.

Otro de los puntos que tratamos es de que tecnologías disponemos para implementar el multijugador online. Nos hemos planteado utilizar estas tres tecnologías: Un servidor web dedicado y utilizar el Network Manager de Unity [36], utilizar Photon [37] o utilizar Mirror [38].

Empezaremos hablando de Photon y Mirror, ya que son dos opciones externas al propio Unity, pero que se pueden descargar desde la asset store e integrarlos.

Las ventajas que tiene Photon es que es fácil de usar y de configurar. Emplea un sistema de redes peer-to-peer, que consiste en una serie de nodos que se comportan como iguales entre sí y permiten el intercambio directo de información entre los dispositivos implicados [39]. Muchos desarrolladores utilizan Photon, y gracias a esto tiene una comunidad muy activa con muchos tutoriales.

La desventaja más destacable de Photon es que no permite la conexión simultánea de muchos usuarios, por lo que solo puede ser usado en proyectos pequeños donde no haya muchos usuarios conectados, como podría ser un juego cooperativo. En nuestro caso, como el servidor base permite la conexión simultánea de 20 usuarios y nosotros tendremos un máximo de 10 nos es suficiente.

Mirror, en cambio, convierte la desventaja de Photon en ventaja, ya que sus servidores permiten la conexión simultánea de muchos usuarios, por lo que está orientado a proyectos de gran escala donde se puedan conectar una gran cantidad de usuarios. Además, su red es de tipo cliente-servidor, por lo que previene de que los usuarios hagan un uso malintencionado de la aplicación (cheaters).

Las principales desventajas de Mirror son que no tiene tantos tutoriales como Photon y es más difícil de utilizar y configurar.

Por último dentro de las tecnologías para implementar el networking, podemos utilizar el propio Network Manager de Unity junto a un servidor propio. Esta opción es la que más desventajas tiene respecto a los dos anteriores. En primer lugar, hay poca documentación y tutoriales, además que es difícil de utilizar y configurar. Es la opción menos usada en pequeños y medianos proyectos, ya que requiere de tener un servidor propio. Además, la capacidad de usuarios simultáneos depende completamente de la capacidad de nuestro servidor, pero es bueno a la vez para aumentar o reducir la capacidad del servidor según la demanda de usuarios.

Algunas de las ventajas, aparte de la capacidad antes mencionada, es que se puede implementar el tipo de red que se quiera, por ejemplo una peer-to-peer o cliente-

servidor, como hemos visto en los otros.

Vistas estas opciones, terminaremos analizando si el desarrollo del museo será únicamente en digital, en RV, o ambos.

La principal ventaja que representaría desarrollar el museo en digital, es que la gran mayoría de usuarios cuentan con un PC, por lo que podrán utilizarlo para acceder al museo. Los periféricos que se necesitan son el teclado y el ratón, y unos auriculares para poder escuchar los sonidos, pero al igual que el PC, todos estos usuarios cuentan con ellos. Una de las pocas desventajas que tendría desarrollarlo únicamente en digital, sería el factor de inmersión, debido a que a través de la pantalla y con los periféricos mencionados no se obtiene la inmersión total en el museo.

Por otro lado, desarrollar el museo en RV tiene la ventaja de la inmersión del usuario y experimentar el museo casi de primera mano, importante para uno de nuestros objetivos como es el factor nostalgia. Una de las desventajas es el hecho de que no todos los usuarios cuentan con un dispositivo de RV, por lo que nuestro público se vería reducido en gran medida. En cambio, si se tiene el dispositivo de RV no se necesita ningún otro periférico, ya que este incluye todo.

Para finalizar este apartado, haremos la selección de las tecnologías y conceptos que hemos ido mencionando. Esta selección se puede ver gracias a la tabla 2.2

Tecnologias	Seleccionadas	No seleccionadas
Emulación		✓
Implementacion de los juegos	✓	
Unity	✓	
Unreal Engine		✓
A-Frame		✓
Google Arts & Culture		✓
Network Manager		✓
Mirror		✓
Photon	✓	
Digital	✓	
Realidad Virtual	✓	

Tabla 2.2: Tecnologías usadas

En primer lugar, uno de los conceptos que hemos analizado en extensión ha sido la emulación de los videojuegos, pero tras haber enumerado las ventajas y desventajas de la emulación, la conclusión es que se implementarán un máximo de 6 videojuegos cortos, de no más de un nivel o pantalla.

En segundo lugar, la tecnología principal que utilizaremos para desarrollar el videojuego. Tras plantearse las ventajas y desventajas que tienen aquellas en las que nos hemos centrado, finalmente nos decantamos por usar Unity. Esta decisión ha sido principalmente por la experiencia previa que tenemos empleando esta tecnología, la comodidad del editor, la extensa documentación, ayudas y contenido gratuito, y la integración con el desarrollo multiplataforma nativo.

Además, como ya habíamos comentado, el museo contará con la opción de mul-

tijugador. Usaremos Photon Networking, con su versión Pun2 para Unity. El máximo de usuarios que ofrece en su versión gratuita es de 20 usuarios, suficiente como para albergar los 10 usuarios máximos que teníamos planeados, además lo hemos elegido por todas las facilidades que proporciona para implementar el online.

Por último, la cuestión de las plataformas en las que se podrá acceder al museo. Hemos decidido utilizar ambas plataformas: Tanto en digital en PC como en dispositivos de RV. El número de usuarios al que podemos acceder con esta opción es mayor, y se podrá experimentar tanto en forma digital para aquellos que no cuenten con un dispositivo RV como para los que sí lo tengan. La diferencia entre estas dos plataformas, es que gracias al dispositivo de RV se experimentará una mayor inmersión en el museo.

CAPÍTULO 3

DESARROLLO

Dentro de este capítulo describiremos la gestión del proyecto, hablando de la metodología del desarrollo y las herramientas que se han usado para gestionarlo, la planificación y la gestión de los interesados. También discutiremos cuáles son los requisitos de usuario, funcionales y no funcionales y los requisitos de sistema. Por último, veremos el diseño, la implementación de la aplicación y las pruebas con usuarios durante el desarrollo.

3.1. Gestión del proyecto

3.1.1. Metodología de desarrollo y herramientas

En este proyecto se ha optado por seguir la línea de metodologías de desarrollo en cascada, dado que desde el principio se ha planteado el desarrollo del proyecto con unas ideas establecidas. Siguiendo esta línea de desarrollo, se comenzó el proyecto con una fase de análisis, posteriormente diseño, implementación y verificación. La implementación del proyecto se ha dividido en distintos bloques, de los cuales hablaremos en el apartado 3.3.4. Se ha hecho uso de la herramienta Trello [40] para establecer y gestionar las distintas tareas definidas por el tutor del Trabajo Final de Grado (TFG) y el alumno. Además, el tutor y el alumno han intercambiado correos explicando el avance del proyecto como mínimo una vez al mes, y teniendo reuniones presenciales cuando lo creían necesario para un intercambio de ideas más fluidas.

3.1.2. Planificación temporal

El proyecto se ha planificado desde el principio en distintas partes. Una fase inicial en la que se reúne información para poder redactar la introducción y el estado del arte del documento, así como la parte de gestión del proyecto y el análisis de usuarios de la parte de desarrollo.

Después, tendremos una fase en la que se empezará a plantear el diseño de la aplicación y se empezará el desarrollo en Unity. Una vez se haya desarrollado el primer videojuego e implementado el multijugador, se hará una prueba de funcionamiento, donde se verá si dos jugadores pueden jugar a dicho juego satisfactoriamente.

Seguidamente, se continuará con el desarrollo de las demás consolas y videojuegos. En la última fase se redactará la implementación de la aplicación, y se harán varias pruebas con usuarios que explorarán el museo y usaran las distintas consolas en una misma sala. Además, se redactará la conclusión y el resumen del documento, así como apartados adicionales que se hayan podido considerar a lo largo del proyecto.

A continuación vemos una imagen visual de la planificación temporal, con fechas de los meses límites para completar los bloques antes mencionados.

Figura 3.1: Planificación temporal

3.1.3. Gestión de los interesados

Durante el desarrollo del proyecto se han identificado distintos interesados, que tendrán que comunicarse para que el desarrollo sea fructífero. Los principales interesados son el tutor Antoni Oliver Tomàs y el alumno Miguel Ángel Gracia Vich.

La comunicación entre estos dos interesados se realizará a través de mensajería con sus respectivos correos electrónicos proporcionados por la Universidad de las Islas Baleares (UIB). Ambos interesados tienen que poder comunicarse al menos una vez

por semana, pero no de forma obligatoria. De media, intercambian información una vez al mes.

La expectativa de los interesados es que el proyecto se entregue cumpliendo todos los objetivos antes de julio de 2022.

3.2. Análisis

En esta sección hablaremos sobre cuáles son nuestros usuarios y cuáles son los requisitos de usuario, tanto funcionales como no funcionales y requisitos de sistema que hemos identificado.

3.2.1. Usuarios

Retomando lo analizado en la sección 2.2, nuestros principales usuarios son jóvenes interesados en los videojuegos y adultos que se quieren reincorporar a la industria o tener unos buenos recuerdos. Más allá de la personalidad, nos interesa de estos usuarios los dos tipos de sistema que vayan a emplear para utilizar la aplicación del museo. Así entonces, tendremos dos tipos de usuario principales: Aquellos usuarios que juegan a través de un sistema PC y aquellos que lo hacen a través de la RV. Estos dos tipos de sistemas que usan los usuarios además servirán para organizar los posibles requisitos de sistema, los cuales explicaremos a continuación junto a los requisitos funcionales y no funcionales.

3.2.2. Requisitos de usuario

- Requisitos funcionales

RF-01	**Control del avatar**
Descripción	El usuario tiene que poder rotar y mover la cámara y su avatar.
Objetivo	Dar control al usuario para desplazarse por el museo
Dependencias	-

RF-02	**Jugar videojuegos**
Descripción	El usuario tiene que poder jugar un videojuego de los implementados en el museo.
Objetivo	Permitir que el usuario pueda jugar un videojuego.
Dependencias	-

RF-03	**Controlar personajes**
Descripción	El usuario tiene que poder controlar un personaje u objeto mientras juega un videojuego.
Objetivo	Permitir que el usuario controle un objeto del videojuego con sus funcionalidades pertinentes.
Dependencias	RF-02

RF-04	**Encender o apagar televisores**
Descripción	El usuario tiene que poder encender o apagar un televisor.
Objetivo	Permitir que el usuario encienda un televisor si está apagado o apagarlo si está encendido.
Dependencias	-

RF-05	**Activar o desactivar globo informativo**
Descripción	El usuario tiene que poder activar o desactivar un globo informativo de las consolas.
Objetivo	Permitir que el usuario active el globo si está desactivado, o desactivarlo si está activado.
Dependencias	-

- Requisitos no funcionales

RNF-01	**Los controles no se pueden cambiar**
Descripción	Los controles que se utilizan tanto en teclado y ratón como en RV no se pueden cambiar ni añadir otros mandos de consolas.
Objetivo	Restringir el alcance para evitar complicaciones.
Dependencias	-

RNF-02	**Videojuego sin capacidad para más jugadores**
Descripción	No se podrá jugar a un videojuego si todos los mandos están siendo usados.
Objetivo	Dar prioridad para jugar al primero que haya llegado.
Dependencias	-

RNF-03	**Jugador quieto mientras juega en una consola**
Descripción	El usuario no podrá mover su avatar mientras está jugando un videojuego, pero sí podrá rotar la cámara.
Objetivo	Evitar que el jugador pierda de vista la pantalla de la consola.
Dependencias	-

RNF-04	**Sin prioridad en los televisores**
Descripción	El usuario podrá encender o apagar un televisor indistintamente si esta ha sido encendido o apagado por otro usuario recientemente.
Objetivo	Dejar elección a los jugadores para que interactúen con el televisor libremente.
Dependencias	-

RNF-05	**Sin prioridad en la información de las consolas**
Descripción	El usuario podrá activar y desactivar el globo informativo indistintamente si este ha sido activado o desactivado por otro usuario recientemente.
Objetivo	Dejar elección a los jugadores para que interactúen con la información de las consolas libremente.
Dependencias	-

- Requisitos de sistema
  Los requisitos de sistema que presentamos a continuación son la selección de requisitos que creemos que son más interesantes de comentar, y que pueden referirse a los requisitos de sistema en general, para PC o para RV

RS-01	**Sincronización de objetos y estados**
Descripción	Cuando un usuario modifique algún elemento de la escena, el cambio también se verá reflejado en las escenas del resto de usuarios. Un usuario que acabe de entrar al museo debe poder ver los cambios que se han realizado antes de que entrara.
Objetivo	Sincronizar los elementos de todas las instancias del museo, permitiendo la interacción entre distintos usuarios
Dependencias	-

RS-02	**Conexiones a la sala**
Descripción	Cuando un usuario se conecta al servidor, automáticamente se le asigna una sala. Cada sala tiene capacidad para 10 usuarios.
Objetivo	Conectar al usuario a una de las salas
Dependencias	-

RSPC-01	**Puntero del avatar en PC**
Descripción	Existe un puntero en forma de punto en el centro de la pantalla de un usuario de PC. Cuando el puntero se sitúa sobre un objeto interaccionable, este cambia a una circunferencia con cierto grosor.
Objetivo	Indicar al jugador cuando está mirando a un objeto interaccionable.
Dependencias	-

RSPC-02	**Controles bolos en PC**
Descripción	El personaje del juego de bolos se tiene que controlar con el teclado. Para que el jugador sepa hacia donde está apuntando el personaje, se dibuja una línea en el suelo con la trayectoria.
Objetivo	Adaptar el control del juego de bolos a los controles de PC.
Dependencias	-

RSRV-01	**Punteros RV**
Descripción	De las manos del avatar del jugador salen unos láseres de color rojo, que cambian a color blanco cuando apuntan a un objeto interactuable.
Objetivo	Indicar al jugador cuando está apuntando de un objeto interactuable que está al alcance.
Dependencias	-

RSRV-02	Controles bolos en RV
Descripción	El brazo del personaje de bolos tiene que imitar el movimiento que hace el jugador con sus controles de RV, así como lanzar la bola donde esté apuntando con la mano.
Objetivo	Permitir una máxima inmersión al tipo de videojuegos que es el juego de bolos de la Wii.
Dependencias	-

3.3. Diseño

En este apartado hablaremos del diseño de la aplicación. Principalmente, hablaremos de los controles, del funcionamiento de los juegos y de como interacciona el jugador con los elementos del museo.

3.3.1. Controles

Dado que la aplicación se ejecutará en dos dispositivos diferentes, tendremos que diferenciar los controles, ya que estos serán diferentes.

- **Controles en PC**: Para la asignación de teclas en PC, nos fijamos en la mayoría de videojuegos en primera persona [41]. En estos videojuegos el jugador se mueve casi continuamente mientras controla la cámara con el ratón, por lo que las manos derecha e izquierda tienen que poder realizar las dos funciones a la vez. Por ello, el control del personaje se suele realizar con la mano izquierda, mientras que la mano derecha se deja únicamente para controlar la cámara.

  Para ser más concreto, para el control de movimiento utilizaremos las teclas WASD, dado que son cómodas de utilizar con tres de los dedos izquierdos. Como hemos dicho, el movimiento de la cámara será únicamente con el ratón, que será manejado por la mano derecha. Además, para interactuar con las consolas se usará el botón izquierdo del ratón, utilizado generalmente en los juegos en primera persona para interactuar con objetos o disparar. Para saber con qué objeto interacciona el jugador, utilizaremos un raycast que sale del centro de su cámara hacia el objeto.

  Estos controles son únicamente para interactuar en el museo, pero debido a que los usuarios también interactúan con distintos videojuegos, estos tendrán controles únicos, pero no muy alejados de estos que hemos mencionado.

- **Controles en dispositivo RV**: El control para los dispositivos de RV se asemejarán en parte a los mandos gamepad. Los dispositivos cuentan con dos mandos, que utilizaremos para los controles junto a las gafas.

  La tarea de controlar la cámara se repartirá entre la parte de la cabeza y del mando derecho. Las gafas colocadas en la cabeza podrán rotar la cámara, y moverla únicamente en sus inmediateces. Junto con la rotación de la cabeza, se usa el joystick del mando derecho para rotar la cámara en un ángulo fijo, de 30º.

Para la parte del control de movimiento, contaremos principalmente con el mando izquierdo, y sobre todo con el joystick izquierdo. Este movimiento se llama continuo, y es uno de los más utilizados en el tipo de locomoción de los juegos en RV [42]. A pesar de que este método suele producir mareos, lo elegimos en lugar de otros tipos como el teletransporte, ya que aporta un mayor nivel de inmersión y si el usuario está acostumbrado a este modo de control es más cómodo. La interacción con las consolas podrá hacerse con el trigger superior de ambos mandos, apuntando hacia un elemento interactivo y estando en un rango de distancia no muy lejano.

Al igual que los controles en PC, estos controles son para interactuar con el museo. En la interacción con los distintos videojuegos, estos controles serán diferentes y dependiendo del juego que sea. Por ejemplo, en el juego de bolos el movimiento del mando será esencial, mientras que otros juegos más estáticos utilizaran únicamente los joysticks y los botones.

3.3.2. Transmisión de los juegos

Alguna de las consolas tendrá videojuegos para jugar, mientras otras tendrán únicamente vídeos sobre uno de los videojuegos de la consola. En ambos casos se hace uso de una textura plana para retransmitir este contenido. En el caso de los videojuegos, la transmisión a esta textura estará enlazada a una cámara virtual que está grabando el videojuego, que funciona en la misma instancia que el museo, pero lejos de la vista del jugador.

Por lo tanto, la única manera que tiene el jugador de saber lo que está pasando en estos videojuegos es a través de un monitor en el espacio virtual, y con los cuales podrá interactuar con los controles de su dispositivo. Mientras el usuario está jugando, los controles que antes servían para moverse por el museo, quedarán bloqueados hasta que deje de interactuar con el videojuego.

3.3.3. Distribución de los objetos y el espacio

La estructura principal del museo consistirá en una habitación *lobby*, que estará conectada a cinco habitaciones distintas. Podemos ver la distribución de las habitaciones en la figura 3.2. Cada una de estas habitaciones tendrá las consolas de las distintas empresas. Estas empresas son Sony, Atari, Microsoft, Nintendo y Sega y no habrá más de once consolas de cada compañía. Las consolas estarán separadas una distancia razonable, pero no estarán muy alejadas las unas de las otras para que el movimiento entre ellas no sea cansado. Respecto a como estarán colocadas las consolas, al ser un juego multijugador, que transcurrirá en una misma sala del servidor, los monitores y las consolas tendrán una posición y altura fija. Si bien la diferencia de altura de los distintos usuarios podría dificultarles interactuar con estas, al no poder modificar la altura según la necesidad de cada uno hemos decidido dejarlas fijas.

Respecto a los distintos videojuegos, estos estarán alejados de la estructura del museo, y a su vez, las estructuras de los distintos videojuegos estarán alejadas las unas de las otras para no crear interferencias entre ellas. Cada videojuego tendrá una única cámara que se usará para transmitir lo que está pasando en cada uno.

Figura 3.2: Estructura del museo

3.3.4. Información de las consolas y vídeos

Dentro de cada habitación, cada consola estará expuesta sobre un mueble y acompañada de un televisor, de tubo o de plasma según la época de la consola. Para interactuar con ellas, tanto si estamos con los controles de PC o RV utilizaremos los Raycast para detectar si estamos apuntando a ellas. Podemos interactuar tanto con las consolas como los televisores al lado de estas. Si interactuamos con la consola, a su izquierda aparecerá un globo informativo, mostrando el nombre de la consola, su año de lanzamiento y el número de consolas vendidas. En cambio, si interactuamos con el televisor, aparecerá en pantalla un vídeo demostración de un videojuego exclusivo de esa consola. Algunos videojuegos serán los más representativos de esa consola, mientras que otros, por falta de exclusivos de la consola, pueden no ser muy conocidos.

Los vídeos de las consolas no solo se podrán visualizar, sino que también contarán con sonido. Este sonido se manifestará en el espacio 3D en formato estéreo, es decir, si el jugador se sitúa a la derecha de la fuente de sonido, este escuchará el sonido desde sus auriculares izquierdos. Además, si el jugador se aleja de la fuente del sonido, percibirá como el volumen de este va decreciendo hasta que si se aleja lo suficiente, deje de escuchar el sonido.

En el modo multijugador, tanto la interactividad con la consola y los televisores no estará restringida para los jugadores. Es decir, si un jugador acaba de encender el televisor o acaba de activar el globo informativo, otro jugador puede apagarlos de forma inmediata, sin ningún tipo de prioridad.

3.3.5. Diseño modular

En este sub-apartado veremos los distintos módulos de la aplicación, sin entrar en detalles, ya que estos estarán explicados en el apartado de implementación.

- **Jugador**: Uno de los módulos principales será el jugador. Cada jugador tendrá un script principal que contiene la gestión de sus controles y el estado en el que se encuentra (explorando o jugando). Los scripts de los jugadores de PC y RV serán ligeramente distintos debido a la diferencia de tecnologías.
- **Multijugador**: Otro de los módulos será el que gestione el online y se encargará que todos los jugadores puedan conectarse a las salas.
- **Museo general**: Es el módulo que comprende la estructura física del museo, con las consolas, vídeos e información sobre estas.
- **Videojuego Pong**: En este módulo se desarrolla el videojuego para la consola Atari Pong. Consiste en el videojuego del mismo nombre, un clásico de la industria.
- **Videojuego Bolos**: En este módulo se desarrolla el videojuego para la consola Wii de Nintendo. Se trata de un videojuego de bolos similar al del videojuego Wii Sports.
- **Videojuego Plataformas**: En este módulo se desarrolla el videojuego para la consola Sega Saturn. Este es un videojuego del género de plataformas, similar a un Sonic o Mario, utilizando una estética pixel art similar a los títulos de la consola.

Finalmente, en la figura 3.3 podemos ver como los módulos interaccionan entre ellos.

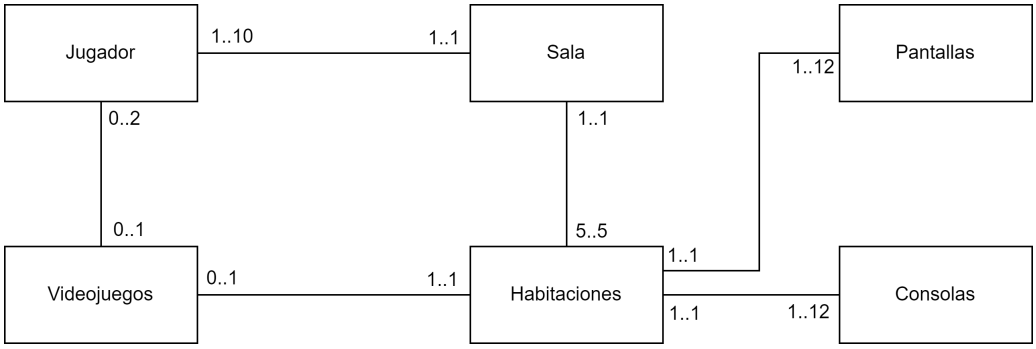

Figura 3.3: Interacción entre módulos

3.4. Implementación

En este apartado profundizaremos como se ha implementado cada módulo de la aplicación. El objetivo es ilustrar el funcionamiento del código sin tener que explicar

línea por línea todo lo que están haciendo los scripts. Antes de ello, trataremos también con las herramientas y plugins que se han utilizado para la implementación de la aplicación.

3.4.1. Herramientas y plugins

Como herramienta principal, la aplicación ha sido desarrollada con el motor de videojuegos Unity. En nuestro caso, utilizamos la versión de Unity 2020.3.21f1. La versión de Unity no ha sido actualizada en ningún punto del desarrollo, por lo que siempre se ha usado esta versión. Otra herramienta que hemos empleado para visualizar y editar el código C# ha sido Visual Studio 2019 [43].

Dentro de Unity, para el desarrollo de la aplicación hemos importado distintos plugins y assets del gestor de paquetes de Unity y de la tienda de assets. A continuación, tenemos una lista con estos, con una breve explicación de cuál es su propósito en el proyecto.

- **OpenXR**: El plugin de OpenXR permite simplificar el desarrollo en Realidad Aumentada/RV, dando a los desarrolladores herramientas para abarcar cualquier dispositivo [44].

  En el caso de nuestro proyecto, se utiliza este plugin para desarrollar toda la parte de RV, sobre todo para dar soporte a los dispositivos de Oculus Quest. Junto al plugin se instalan otros paquetes que complementan a este para que las distintas funciones de XR actúen correctamente según el dispositivo objetivo.

- **PUN 2**: Photon Unity Networking (PUN) es un paquete de Unity para juegos multijugador online [45]. Permite conectar distintos dispositivos en la nube bajo una misma sala donde los usuarios pueden interactuar y sincronizar el contenido de la aplicación. En esta aplicación hacemos un gran uso de PUN, ya que es nuestro principal y único método para permitir que distintos usuarios se conecten a nuestra aplicación y sincronicen todas sus acciones en el museo. Hablaremos más en profundidad de cada uso de PUN en los siguientes subapartados.

  Dos de los conceptos que mas estaremos tratando en todos los sub-apartados se tratará del Photon View y el PUNRpc, que denominaremos de ahora en adelante simplemente Remote Procedural Call (RPC).

  El Photon View sirve para identificar a un objeto a través del servidor y permitirá al poseedor de este actualizar al objeto pertinente y así sincronizarse en la aplicación de los otros usuarios. Dado que los scripts se ejecutan en cada usuario, utilizaremos el Photon View para restringir el acceso a otros usuarios que no posean el objeto al cual esta vinculado el script.

  Las RPC consisten en lo que su propio nombre implica, son métodos llamados en clientes remotos de la misma sala. Utilizaremos estos métodos para sincronizar variables en los distintos clientes que estén conectados en la misma sala.

- **Sketchfab for Unity**: Sketchfab para Unity te permite publicar contenido 3D directamente desde Unity. Además, te permite importar contenido 3D en formato GLTF directamente a tu proyecto de Unity [46].

Sketchab para Unity ha sido un plugin realmente útil para nuestra aplicación, ya que nos ha permitido importar directamente a nuestro proyecto Unity los modelos de las consolas descargados desde Sketchfab en formato GLTF. Con solo arrastrar el archivo a la ventana del plugin, importa automáticamente el modelo 3D a la carpeta deseada con el nombre deseado. No todos los modelos han sido importados de Sketchfab, también se han descargado de otros servicios web. Todos los orígenes de estos modelos se pueden ver en el apartado del apéndice A.2.

- **XR Interaction Toolkit**: El plugin XR Interaction Toolkit es un sistema de interacción de alto nivel basado en componentes, que proporciona un marco que hace que las interacciones 3D y de interfaz de usuario estén disponibles a partir de eventos de entrada de Unity.[47]

  Nosotros utilizamos este plugin para gestionar la interacción de los componentes RV del museo con el usuario. Trataremos este plugin más en profundidad en los apartados 3.4.2 y 3.4.4

3.4.2. Controlador de los jugadores PC y RV

En este sub-apartado estudiaremos el código interno de los distintos jugadores. En el caso de nuestro museo, hay dos tipos de jugadores o usuarios: el usuario que accede a través de un PC sin un dispositivo RV conectado o el que accede con un dispositivo RV conectado.

Controlador de jugador en PC

Empezaremos observando el comportamiento del jugador PC. Una vez la aplicación inicia, y se conecta al servidor utilizando PUN. Si se ha detectado que no hay ningún dispositivo RV conectado, procedemos a cargar la escena donde el usuario clicará en *Play*, para posteriormente conectarse a una sala y cargar la escena del museo.

Una vez el jugador ha sido instanciado en la escena del museo empleando la instanciación de PUN, la cual se utiliza para sincronizar la aparición del jugador en todos los nodos usuarios, podemos empezar a observar el comportamiento del personaje. El script principal que contiene todo el comportamiento se llama *JugadorFPPCBehaivour*, es decir, el comportamiento del jugador de PC en primera persona.

La primera función que se ejecuta dentro del script, *OnPhotonInstantiate*, guarda el objeto del jugador en una propiedad de la clase *Player* de Photon. Además, se guarda en otra propiedad un booleano que indica que no es un jugador RV.

Lo siguiente que hace el script del jugador en su función de *Start*, es centrar el cursor, bloquearlo y ocultarlo, para que este no moleste al observar con la cámara en primera persona. Acto seguido, guarda en una variable la referencia a la cámara del objeto y al Photon View, que como mencionamos en 3.4.1, sirve para identificar a un objeto a través del servidor y restringir el acceso a algunas funciones del script a otros usuarios que no posean el personaje. En el caso de que poseamos el personaje, inicializaremos un par de variables booleanas. Además, debido a que las cámaras de los personajes inician como desactivadas para evitar el acceso no deseado de otros jugadores, activaremos únicamente las del jugador poseedor en su propia instancia

del museo. Por último, activamos el puntero que utiliza el jugador para apuntar, que se sitúa en el centro de la pantalla.

Una de las funciones es *RecieveInput*, que como su propio nombre indica, recibe el input del usuario y en nuestro caso, actualiza únicamente un Vector2 del input horizontal, es decir, los ejes X e Z recibidos a través de las teclas W,S y A,D respectivamente.

Dentro de la función *Update*, la cual se llama en cada frame, tenemos una serie de condicionantes para gestionar las distintas acciones que puede realizar el jugador. Primero de todo, miramos que somos los poseedores del personaje, es decir, el Photon View es nuestro. Si no estamos jugando ningún videojuego, movemos al personaje con su Character Controller dependiendo del input que hayamos recibido. En todo momento, el jugador lanza un Raycast, con origen el centro de la cámara y con una distancia lo suficientemente lejana como para poder interactuar con los objetos cercanos. Si somos capaces de interactuar con ese objeto, el puntero básico se cambiará por otro distinto, para mostrar que ese objeto es interactuable. Este cambio del puntero lo podemos ver en la figura 3.4. Tenemos tres tipos de interacciones en el museo: interacción con las pantallas, con las consolas y con los mandos. El funcionamiento de estos objetos los comentaremos en uno de los siguientes apartados.

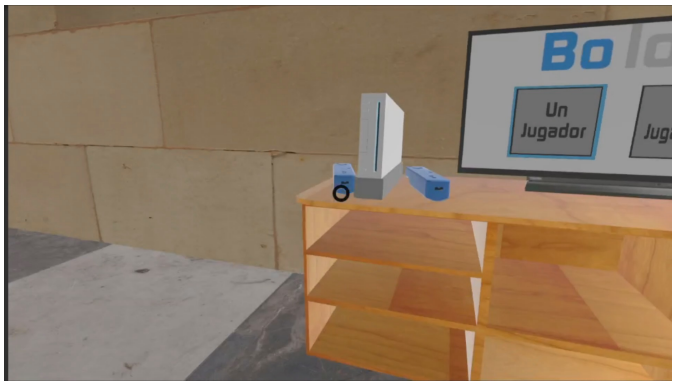

Figura 3.4: Puntero del jugador en PC al enfocar objetos interaccionables

Si el Raycast entra en colisión con una pantalla, y pulsamos el botón izquierdo del ratón, cogemos la componente del objeto que contiene el gestor de vídeo, y según si el vídeo ya se está reproduciendo, lo reproduce o lo para según la situación.

Si el Raycast colisiona con una consola y pulsamos el botón izquierdo, funciona parecido a la pantalla. La única diferencia es que cogemos la componente que contiene el gestor del globo informativo, y lo muestra o lo oculta según si está activo u oculto en ese momento.

Por último, si el Raycast colisiona con un mando y pulsamos el botón izquierdo del ratón, se iniciará un proceso que involucra distintas variables para que el jugador sostenga el mando y se posicione enfrente de la pantalla con el videojuego.

Para realizar esta transición, primero comprobamos que no haya nadie que ya esté sosteniendo ese mando. Después, guardamos el estado actual del mando, para devolverlo a su sitio al soltarlo. Indicamos al mando que alguien lo ha cogido para que actualice su información, esta actualización de información por parte del controlador del mando lo veremos en más detalle en la sub-sección 3.4.4. Hecho esto, indicamos

que el personaje está en estado *jugando*, por lo que no podrá moverse en el espacio, únicamente podrá rotar la cámara. Se posiciona el mando en el espacio designado para ello y a partir de entonces, el jugador ya puede interactuar con el videojuego del que ha cogido el mando.

Una vez estemos sosteniendo el mando, podremos dejar el mando, incluso si estamos en medio de una partida, con la tecla Q. Primero de todo, confirmamos que efectivamente estamos sosteniendo un mando, y se realiza el proceso inverso a cuando cogemos el mando. Recuperamos el estado original del mando y lo devolvemos a su sitio, con la correspondiente actualización de salida por parte del controlador del mando. Por último, el estado del jugador deja de ser *jugando* y es libre de continuar moviéndose por el museo.

Respecto a la sincronización para que los otros usuarios vean los cambios que ha provocado el jugador al interactuar con los objetos, utilizaremos las RPC. En el caso de las pantallas y las consolas, de eso se encargan sus gestores. En el caso de coger o soltar un mando, en el momento en el que actualizan los valores dentro de *Update*, se llaman a las RPC correspondientes a la entrada y salida del mando.

Con esto, concluiríamos el funcionamiento del gestor del comportamiento del jugador en PC, pero hay un pequeño script que tendríamos que revisar también, el controlador de la cámara, llamado *Mouse Look*. Como su propio nombre indica, se enfoca en el control del ratón. Al igual que el movimiento en los ejes X y Z que vimos en el anterior script, en este también recibimos un input, en este caso el input del ratón. Tan solo tenemos en la función *Update*, donde comprobamos inicialmente que poseamos el Photon View, y realizamos una actualización de la rotación de la cámara según el input recibido.

Controlador de jugador en RV

Ahora pasaremos a hablar del comportamiento del jugador en RV, el cual es bastante similar al del PC. Entonces, comentaremos los puntos en los que son diferentes. Primeramente, en el controlador de RV tenemos una serie de funciones para actualizar la posición de la cámara según la posición del dispositivo de RV y de la animación de las manos según el input del trigger y grip de los mandos derecho e izquierdo.

Aunque no todas las funciones de movimiento se realizan en este script. El movimiento con los joysticks, siendo el joystick derecho el encargado de rotar el cuerpo del personaje en ángulos fijos, queda relegado al *Snap Turn Provider* proporcionado por el XR Interaction Toolkit, así como el joystick izquierdo relega el movimiento en los ejes X y Z al *Continuous Move Provider*. Estos dos scripts vienen contenidos en el *Locomotion System*, el cual es hijo del objeto que contiene todo el personaje.

En la función *Start* guardamos e inicializamos las variables correspondientes al sistema XR, Photon View y la cámara del jugador.

En la función *Update* llamamos a las funciones que actualizan las animaciones de las manos, en caso de que no estemos jugando. Además, llama a dos funciones que controlan por separado las acciones de las manos izquierda y derecha. Esta diferenciación reside en que, a diferencia del controlador del PC, donde teníamos un Raycast que se lanzaba desde el centro de la cámara, en el caso del controlador RV tenemos dos Raycast: uno que se origina en la mano izquierda y otro que se origina en la mano derecha. A pesar de esta diferenciación, lo único que varía de una función a

otra es el input dependiendo de la mano que sea y de la colisión de los Raycast. Esto permite además que un jugador RV pueda activar, por ejemplo, un televisor con la mano derecha y una consola con la mano izquierda al mismo tiempo.

Respecto al funcionamiento de cada mano, empezaremos mencionando que el Raycast no es solo lógico, sino también visual. Gracias a los componentes *XR Ray Interactor* y al *Line Renderer*, un rayo de color rojo, con cierto grosor, emerge de cada una de las manos, lo que representa el alcance de los Raycast de cada mano. Cuando un Raycast impacta con un objeto interactuable, automáticamente cambia el color del rayo a uno blanco.

Profundizando más en la función que maneja cada mano, esta empieza guardando en una variable el valor del input del trigger y grip de los mandos.

Al igual que el controlador del jugador de PC, si el Raycast de la mano colisiona con una pantalla o consola, si pulsamos el botón de trigger se activará o desactivará el vídeo o globo informativo, según su estado actual.

La parte diferente reside cuando interactuamos con un mando. Aparte de lo ya visto en el controlador del jugador de PC, en el caso del jugador RV el objeto del mando hereda la mano que lo haya cogido, por lo que su posición en todo momento vendrá definida por como se esté moviendo la mano. Así mismo, el modelo de la mano desaparecerá, y la mano contraria tendrá bloqueado su Raycast, para no poder interactuar con otros objetos mientras estemos sosteniendo un mando. Además, el jugador RV verá limitado su desplazamiento, ya que únicamente podrá rotar la cámara o moverse en los ejes X y Z pero sin utilizar el control por movimiento, solo utilizando su cuerpo físico.

Por último, tenemos las mismas RPC que en el controlador de PC, sumando una nueva función llamada *ActualizarColorBase*, la cual actualiza en los otros clientes el color de los láseres del personaje en cuestión cuando hay algún cambio en el color. Para no estar continuamente sincronizando el color de los láseres, esta RPC se llama únicamente cuando ha habido un cambio al interactuar o dejar de interactuar con un objeto.

Con esto finalizamos el estudio del funcionamiento de los controladores. Como hemos podido ver, hay muchas similitudes entre los controladores de PC y RV, por lo cual la creación de uno de los controladores ha sido fácil gracias a la similitud con el otro controlador, a pesar de las diferencias.

3.4.3. Gestor del online

Dentro de este sub-apartado veremos en profundidad como se gestiona la conexión entre los usuarios y el servidor, que tipos de conexión tenemos, algunos detalles de las llamadas a RPC, etc.

Comenzaremos detallando como es la configuración de nuestro servidor proporcionado por Photon. Para comenzar, al ser la versión gratuita, el servidor tiene una capacidad máxima de 20 usuarios de forma concurrente. Estos 20 usuarios pueden estar divididos en el número de salas que se desee. En nuestro caso, el museo se desarrolla en una única sala, con un máximo de 10 usuarios conectados de forma concurrente. El término de sala se refiere a una porción del servidor que contiene un número determinado de usuarios, distinto al término de sala o habitación del museo, el cual tiene 5 habitaciones más el lobby. La región del servidor es Europa, y su protocolo de conexión es TCP (Transmission Control Protocol). Elegimos este protocolo frente a los demás, ya

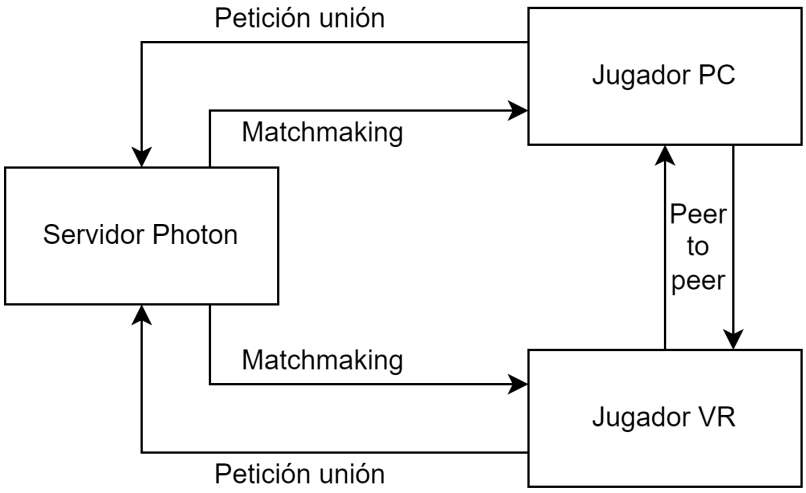

Figura 3.5: Diagrama de conexión entre los usuarios y el servidor

que permite la retransmisión de datos perdidos, en el caso de que los haya. Además, Photon permite la conexión mientras la aplicación se ejecuta en background, algo muy útil si el usuario desea poner el foco en otra aplicación y luego regresar a esta. A la hora de realizar la conexión, los usuarios mandan una petición al servidor, y este realiza un matchmaking con la sala. Durante el transcurso de la conexión, cuando hay más de un usuario, estos se comunican directamente con una conexión peer to peer. En la figura 3.5 podemos ver como estos efectúan la conexión.

En la aplicación hay cuatro escenas distintas de Unity, donde cada una se utiliza Photon de forma distinta.

En la escena *Loading*, donde tenemos un objeto contenedor del script nombrado *ConectarAlServidor*, se ejecuta automáticamente la conexión de PhotonNetwork usando la configuración antes mencionadas. Cuando el Master de la conexión se ha conectado, llama a la función para unirse al lobby del servidor (no confundir con el lobby del museo). Una vez el usuario se ha conectado al lobby, dependiendo de si el usuario está conectado con RV o no, salta a la escena de *UnirseVR* o *Unirse*.

De las dos escenas mencionadas, en esencia son muy parecidas. Una vez se carga la escena, tanto en RV como si no, tendremos un botón con el que podremos interactuar. Podemos observar como se ve esta escena con el botón en la figura 3.6, y también, como se organizan las escenas en la figura 3.7.

Una vez se pulsa el botón, clicando en él con el ratón o apuntando con una mano y pulsando el trigger del mando en RV, se llama a una función, donde se inicializa las opciones de la sala. En nuestro caso, como tenemos únicamente una sola sala, esta está siempre abierta, pero el número máximo de jugadores que pueden estar de forma concurrente es de diez. Una vez asignadas estas opciones, se llama a la función *JoinOrCreateRoom*, a la cual le pasamos el nombre de la sala y las opciones que hemos asignado. Cuando nos hemos unido correctamente a la sala del servidor, cargamos la escena *Museo*. Tanto el usuario con RV como con PC, acabarán en la misma sala.

Una vez hemos cargado correctamente la escena, el primer script que llamamos

Figura 3.6: Escena Unirse

es *ScenePlayers*, el cual se encarga de hacer la instanciación en Photon del prefab de PC o de RV. Primero miramos si el usuario tiene conectado el dispositivo RV, y si es el caso, instanciamos su prefab en una posición predeterminado. Lo mismo sucede con el usuario de PC.

Con esto concluiría la parte de conexión al museo, ahora trataremos algunos aspectos de la sincronización de variables y estados, aunque la sincronización de variables de los scripts se verán en sus respectivas sub-secciones de este apartado.

Cabe destacar respecto a la sincronización, que diferenciamos la sincronización de los transform de los objetos y las variables dentro de cada script y los objetos que están activos o desactivados.

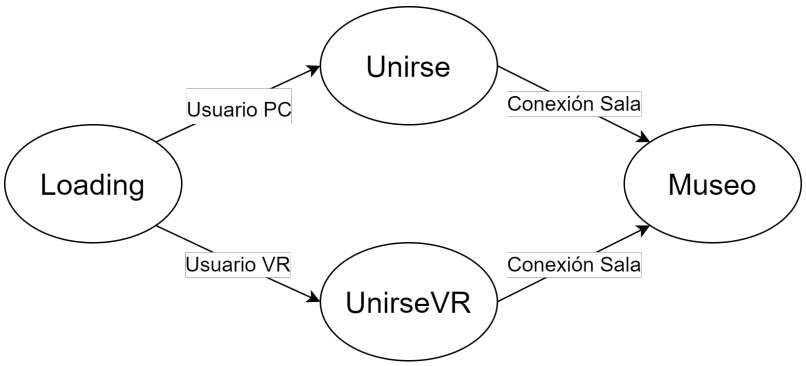

Figura 3.7: Diagrama de transición entre escenas

Los transform de los objetos se sincronizan automáticamente gracias al Photon View Transform, el cual se encarga de que siempre que el usuario local posea el Photon View de un objeto, este se sincronizará en la escena del resto de usuarios.

Cuando una variable que debería ser actualizada no solo para el usuario local, sino para los demás usuarios también, como podría ser algún booleano interno de un juego o si un mando ha sido cogido o no, hay que mandar un RPC con el valor de las variables que se tienen que actualizar. En el ejemplo de que no solo haya que actualizar una

variable, sino que también hay que activar o desactivar un objeto del que el script tiene una referencia, también se utilizan las RPC.

Una cuestión importante respecto a las RPC reside en qué momentos, la cantidad de veces y para quien estas son llamadas. Al llamar a una RPC, a la función se le pasan dos valores principales: los parámetros de entrada que mandamos a la función, es decir, los valores de las variables que queremos sincronizar y también a que usuarios va dirigida esta llamada. Esto último es fundamental, ya que algunas variables necesitan estar sincronizadas cuando un nuevo jugador entra a la sala, mientras que otras solo necesitan sincronizarse para los jugadores que están en ese momento en la sala. Sincronizar los estados de los objetos y variables referidas a una partida de un videojuego, si una pantalla está encendida o apagada, etc, debería guardarse los cambios en un buffer para que los jugadores que entren vean los cambios realizados. Por otro lado, el color de los láseres de las manos de los jugadores RV únicamente se sincroniza para los jugadores que estén en la sala cada vez que estos se actualizan.

Por último, una vez tratado de forma general la sincronización de variables en los distintos objetos del museo, trataremos que sucede cuando un jugador se desconecta de la sala y del servidor.

El principal problema que existía a la hora de que un jugador se desconectase, es que qué sucedía si ese jugador estaba jugando a un videojuego y estaba en posesión de un mando. Para solucionar este problema, creamos un script *Disconnected*, donde se implementan unas funciones callback para cuando el jugador abandona una sala o se desconecta.

Lo primero que hacemos en estas funciones, las cuales comparten el código, es mirar si el jugador era RV o no. Acto seguido, llamamos a una función auxiliar para realizar un código similar al que se encuentra en los scripts del controlador de comportamiento del jugador cuando este deja un mando. Guardamos la información del jugador y devolvemos el mando a su sitio como si fuera el propio jugador que lo está haciendo. Además, se llama a la función del mando que actualiza el estado de salida del mando, y que probablemente dará por finalizada la partida del videojuego que se estaba jugando. Una vez hecho esto, destruimos el objeto por si Photon no ha logrado destruirlo antes de la desconexión.

Como conclusión para este sub-apartado, mencionar que el potencial de las RPC ha sido de gran utilidad a la hora de sincronizar los estados y las variables de los objetos y al gestionar la desconexión de los jugadores de forma manual aparte de la gestión de Photon ha servido para asegurarse que no se crean errores irreparables dentro del museo.

3.4.4. Implementación del museo general

El museo consiste en una serie de suelos, paredes y tejados que juntos forman un total de cinco habitaciones y un lobby. En el lobby es donde aparecen los jugadores, y cada una de las habitaciones restantes contiene las consolas que pertenecen a cinco empresas de videojuegos distintas. Nada más aparecemos en el *lobby*, la primera habitación a mano izquierda tiene las consolas de la empresa Atari, la primera a la derecha la de XBOX, la segunda a la izquierda Sega, la segunda a la derecha Sony, y la del fondo Nintendo.

Los jugadores que entran en cada una de las habitaciones tiene la posibilidad de realizar diferentes tipos de actividades y cosas que ver.

Modelos y globos informativos

La primera y más sencilla, ya sea desde un PC o con un dispositivo RV, es la posibilidad de ver el modelo de cada uno de los modelos 3D de las consolas y algunos modelos de los mandos de estas. Todos los modelos han sido importados y descargados desde distintos sitios, estos se pueden ver en el apartado del apéndice A.2.

Otra de las actividades, relacionada con el modelo de la consola, ocurre cuando un jugador, mediante el puntero central de la versión de PC o del puntero de la mano de la versión RV, activa una función que mostrará el globo informativo vinculado a esa consola. Este globo informativo contiene la información del nombre de la consola, del año de su lanzamiento y del número de consolas vendidas aproximadamente. Podemos ver un ejemplo de globo informativo en la figura 3.8

Respecto al script que maneja la aparición o desaparición de este globo informativo, este es una componente del objeto que contiene el modelo de la consola. Si el Raycast, ya sea del jugador de PC o RV, colisiona con la caja de colisión de la consola, este recogerá la componente que contiene el script. Primero de todo mirará si este objeto está activo, es decir, está visible o no. En el caso de que esté inactivo, llamará a la función *ShowInfo* del script *InfoManager*, la cual aparte de utilizar la función *SetActive* de un objeto para ponerlo en verdadero, y por tanto activarlo, también llamará a la RPC para que en las instancias de los demás usuarios suceda lo mismo, a fin de sincronizar correctamente la escena. En el caso de que el objeto este activo, sucederá el proceso contrario. Se llamará a la función *HideInfo*, que empleará la función *SetActive* para ponerlo en falso, y por lo tanto, desactivarlo. Al igual que el otro caso, se llama a la RPC para que en las instancias de los demás usuarios el objeto quede inactivo.

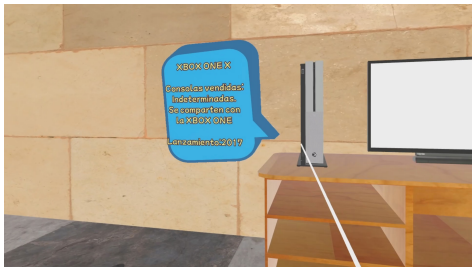

Figura 3.8: Jugador viendo un globo informativo

Figura 3.9: Jugador viendo un vídeo en el televisor

Vídeos demostración

Otra actividad parecida a esta que acabamos de ver, se trata de que al interactuar con las pantallas que se encuentran al lado de las consolas, estas muestran un vídeo de unos treinta segundos. Podemos ver como se muestra el vídeo en la pantalla gracias a la figura 3.9 Este vídeo es un recorte del recorrido de uno de los videojuegos más

representativos de la consola. La fuente de los vídeos se encuentra en el apartado del apéndice A.2.

Para activar o desactivar este vídeo el funcionamiento es muy parecido al del globo informativo, con una pequeña diferencia.

Para interactuar con la pantalla se utiliza el mismo funcionamiento con los Raycast, que una vez colisionan con la caja de colisión de la pantalla recogen el componente *VideoManager*. En este caso, el script contiene una variable pública *VideoPlayer*, un tipo que componente que maneja la reproducción de los vídeos, pudiendo conocer si el vídeo está en play o stop. El conocer si está en play o stop toma la misma función que cuando mirábamos si el objeto del globo informativo estaba activo o no. Una vez hemos determinado si el vídeo está en play o stop, podemos tomar el curso de la acción. Si está en play, llamaremos a la función *StopVideo* del script *VideoManager*, el cual llamará a la función *Stop* del video player, y llamará también a la RPC correspondiente para sincronizar el resto de escenas con la misma función de *Stop*. En el caso de que el vídeo esté en stop, se llama a la función *StartVideo*, la cual llama a la función *Play* del video player, además de llamar a la RPC correspondiente.

Es muy importante conocer y sincronizar el estado del vídeo, ya que cualquier usuario tiene que poder poner en play o en stop el vídeo en cualquier momento, dando igual si alguien justo acababa de ponerlo en play o acababa de ponerlo en stop. Además, cuando un vídeo se pone en stop, este vuelve al segundo cero del vídeo, por lo que la próxima vez que alguien le dé a play, el vídeo comenzará desde cero.

Fuentes de sonido

Otro elemento a tener en cuenta relacionado con los vídeos es la fuente del sonido. Tanto para los vídeos que se transmiten por pantalla, como los distintos sonidos que puedan generar los videojuegos, estos emitirán su sonido a través de una fuente de sonido, situada lo más cerca posible a la pantalla, sin que entre en conflicto con otras fuentes de sonido. Esta fuente tiene unas características especiales, que ayudan a la inmersión a la hora de interactuar con estos elementos que emiten sonido.

Para empezar, el sonido que emite esta fuente, se trata de un sonido estéreo, es decir, se reproduce por los dos canales de audio. En el caso de utilizar unos auriculares, se reproduciría en el auricular derecho e izquierdo, igual sucedería en cualquier dispositivo que permita el audio en estéreo.

Sumado al audio en estéreo, dependiendo de donde esté situado el receptor de sonido en comparación con la fuente de sonido, el volumen de audio será mayor o menor, disminuyendo de forma lineal hasta cero si superamos un cierto margen de distancia. Cabe mencionar, que normalmente el receptor de audio se encuentra incorporado en la cámara del jugador.

El factor del audio es muy importante para la inmersión del usuario en la aplicación, ya que si la fuente de sonido se encuentra a la izquierda de la cámara del jugador y contamos con auriculares, el audio sonará más fuerte en dicho auricular. También, cuando estemos más cerca de la pantalla, notaremos que el volumen está alto como si realmente estuviéramos en una pantalla, y que este disminuye a medida que nos alejamos de la pantalla.

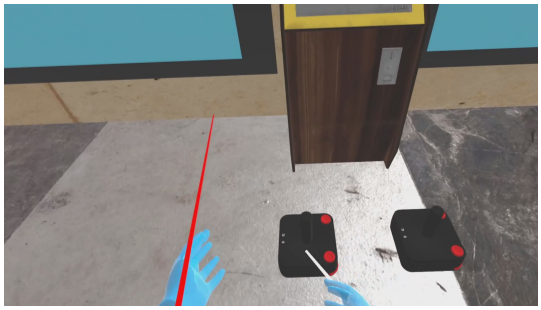

Figura 3.10: Jugador a punto de coger un mando

Mandos de las consolas

Un jugador puede coger un mando siempre y cuando no haya otro jugador sujetándolo previamente. Podemos observar un usuario de RV cogiendo un mando en la figura 3.10. Los mandos contienen un componente que hereda de una clase general para todos los videojuegos. Este script padre se trata de *PersonajeBehaviour*, en el cual guardamos distintas variables que serán comunes para todos los gestores de personaje de los videojuegos. Además de estas variables, también tenemos una funciones comunes.

En anteriores apartados de la implementación ya las hemos mencionado y explicado cuando se llaman, se tratan de *ActualizarEstadoEntrada* y *ActualizarEstadoSalida*. Cada una de las clases hijas sobreescribirán estas funciones para realizar la función que se requiera en cada videojuego.

Retomando las variables de esta clase, contamos con dos Photon View: Una variable almacena el Photon View del objeto que representa al personaje que controlará el jugador con ese mando, la otra variable guardará el Photon View del jugador, ya que solo este podrá controlar al personaje del videojuego. Tendremos también un booleano que indicará si un jugador ha cogido el mando o no, además de una variable string que guarda el nombre del mando. Este string es de utilidad para diferenciar las funciones del mando que actuará como jugador uno y el mando que actuará como jugador dos en el caso de los videojuegos multijugador. Por último, guardamos cuál es la posición donde el jugador quedará anclado una vez recoja el mando.

Para concluir este apartado debemos comentar que la implementación de todas las funcionalidades que hemos ido nombrando han sido separadas, subdivididas e implementadas de tal forma que el mal funcionamiento de unas no afecte a las demás, lo que además ha facilitado la implementación, ya que al construir pequeñas tareas era mejor para comprobar su correcto funcionamiento.

3.4.5. Videojuego tipo Pong

Uno de los videojuegos que hemos recreado para que los jugadores puedan jugar en el museo se trata del videojuego Pong. Este videojuego se considera uno de los primeros videojuegos que existieron y es una adaptación del tenis de mesa. El juego consiste en que dos *raquetas*, que son controladas por dos jugadores distintos, tienen que hacer rebotar una bola para que termine en el campo posterior a la raqueta del

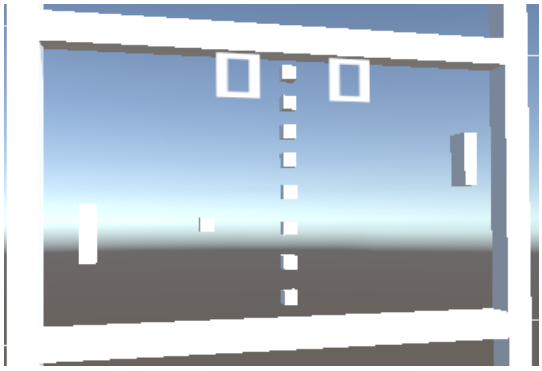

Figura 3.11: Estructura del campo en el videojuego Pong

rival.

Adaptación y estructura

Para adaptar este videojuego a Unity, hemos tenido que tomar una serie de consideraciones. Primeramente, la escena de Unity en la que transcurre el museo consiste en un entorno 3D. Este factor parece entrar en conflicto con la idea del videojuego Pong, el cual transcurre en un entorno 2D. Como comentamos en el sub-apartado 3.3.2, a pesar de que la estructura del videojuego transcurra en un entorno 3D, utilizando una cámara virtual con proyección ortográfica, se captura la imagen como funciona el videojuego, y se transmite la salida de tal cámara a una textura plana, situada en la pantalla del modelo 3D de la cabina Pong. Gracias a esta transmisión, el jugador ve en la pantalla como transcurre el videojuego en un formato 2D, aunque realmente se encuentre en un entorno 3D.

Una vez hemos solventado la adaptación del videojuego, pasamos a comentar cuál es la estructura básica del videojuego. El juego transcurre en un *campo* cerrado por cuatro paredes. Las paredes laterales izquierda y derecha hacen la función de una *porteria*. A demás de las cuatro paredes, nos encontramos con una línea puntillada cuya función es dividir los campos de los dos jugadores. En el campo de cada jugador encontramos dos elementos: la raqueta que utilizará el jugador y un contador con la puntuación que tiene dicho jugador. El último elemento importante es la bola, la cual irá rebotando por las paredes superiores o inferiores del campo o cuando una raqueta se interponga en su camino. Podemos ver el modelo de todo el campo en la figura 3.11

Ahora que hemos visto el concepto general del videojuego, vamos a tratar el comportamiento de cada una de las piezas que lo componen, las cuales son: las raquetas, la bola y el campo.

Comportamiento del mando y la raqueta

Empezaremos hablando de las raquetas. Hay dos raquetas en el videojuego: una situada al lado izquierdo del campo, la cual estará controlada por aquel jugador que coja el mando situado a la izquierda, la otra está situada en el lado derecho del campo y será controlada por el jugador que coja el mando situado a la derecha.

Cada raqueta está vinculada a uno de los mandos, como si se tratara de un mando real que manda los inputs y actuadores a la consola. Cada mando tiene distintas variables que ayudan a su correcto funcionamiento. Cuenta con distintos punteros: un puntero a la bola para actualizar los Photon View, un puntero al comportamiento general del Pong y otro al objeto que representa la propia raqueta. Además, cuenta con dos booleanos, uno para ver si el jugador que ha cogido el mando es de PC o RV y otro para indicar si el Pong está en marcha. Por último, tenemos un string constante que indica el nombre del mando nº 1 y que utilizaremos para saber si estamos en posesión del primer o el segundo mando.

Como comentamos en los sub-apartados 3.4.2 y 3.4.4, las clases que heredan de *PersonajeBehaivour* como es en este caso *RaquetaBehaivour* cuentan con dos funciones que actualizan el estado del videojuego cuando un jugador coge o suelta un mando. Vamos a ver entonces, como sobreescribe esta clase dichas funciones, y como actúan en el videojuego de Pong.

En *ActualizarEstadoEntrada*, el cual se llama cuando un jugador coge el mando, nos encontramos con una serie de acciones. Primero, comprobamos que vamos a ser los únicos poseedores del mando gracias al Photon View que le hemos pasado anteriormente. A continuación, nos apropiamos del Photon View del objeto representativo de la raqueta. Además, verificamos si el jugador poseedor del mando es RV o no, y actualizamos el booleano en el que almacenamos esta información.

Por otro lado, en *ActualizarEstadoSalida* debemos realizar una serie de acciones más extensas. Primero de todo miramos cuál ha sido la causa de la llamada a esta función, ya que a diferencia de la anterior función, esta también puede ser llamada durante la desconexión de un jugador. Empezaremos tratando el caso de que haya sido el jugador que voluntariamente ha soltado el mando. A continuación, tendremos en cuenta si somos poseedores del primer o el segundo mando.

En el caso de que seamos poseedores del segundo mando, podremos soltar el mando sin afectar al juego en dos casos: el juego aún no ha sido iniciado o el juego está en marcha, pero el jugador uno está jugando contra la IA. Si no se cumple ninguno de los dos casos anteriores, entonces indicaremos al controlador del Pong que pare el juego, llamando a sus funciones correspondientes.

En el caso de que seamos poseedores del primer mando, tendremos únicamente dos casos que tratar: si el juego no ha sido puesto en marcha, dejaremos el mando sin ver ningún cambio en el juego, mientras que si el juego está en marcha, tanto si jugamos contra una IA o contra otro jugador, indicaremos al controlador del Pong que pare el juego.

En el caso de que esta función haya sido llamada a causa de la desconexión de un jugador, no podremos saber si somos poseedores del mando, pero si sabemos si se trata del primer o del segundo mando, por lo que podemos actuar igual que en el estado de salida normal, sin tener en cuenta el Photon View del jugador local.

Antes de continuar con la función *Update*, mencionar que contamos con dos funciones auxiliares llamadas *EmpiezaJuego* y *QuitarJugando*, el cual actualiza el booleano *jugando* del mando. Estas funciones son llamadas por el controlador del Pong, que veremos más adelante.

Por último, veremos la función *Update*, la cual controla la raqueta en función de los inputs del usuario. Lo primero que haremos será mirar si el juego está en marcha y si en efecto un jugador ha cogido el mando. Dependiendo de si se trata de un jugador de

PC o RV trataremos los inputs de forma diferente, además, si el segundo jugador tiene cogido el mando, el juego está en marcha, pero el modo de juego es contra la IA, este no podrá controlar la raqueta.

Si el jugador que tiene el mando es un usuario de PC, recogeremos los inputs de las teclas W y S. Estos inputs modificarán la posición de la raqueta en dirección hacia arriba o hacia abajo respectivamente. Si el jugador que tiene el mando es un usuario de RV, similar al de PC, recogeremos los inputs del joystick del mando izquierdo. Si el valor del eje Y del joystick es positivo, la raqueta modificará su posición en dirección hacia arriba, si el valor es negativo irá en dirección hacia abajo.

Cabe tener en cuenta que en ambos casos, la raqueta no puede superar un cierto margen tanto superior como superior. Este margen representa la pared superior e inferior del campo de Pong, el cual no pueden atravesar por razones lógicas.

Una vez hemos visto el comportamiento del mando y la raqueta, veremos el comportamiento de la bola.

Comportamiento de la bola

La bola consiste en un cuadrado o cubo, la cual rebota en las paredes superior, inferior y en las raquetas. La bola es uno de los objetos que más variables maneja. Estas variables comprenden desde la velocidad en X e Y que tiene la bola, punteros a la fuente de audio, al controlador del Pong y las posiciones de reaparición, distintos Photon View y temporizadores para la animación.

Empezaremos viendo las distintas situaciones cuando la bola colisiona con un objeto. Tenemos tres posibles casos: si colisiona con una raqueta, si colisiona con la pared superior o inferior, y si colisiona con las paredes izquierda y derecha.

Si la bola colisiona con una raqueta, aleatoriamente cogerá una velocidad distinta para X e Y. Además, emitirá un pequeño sonido que saldrá en la fuente de sonido que se encuentra justo sobre la cabina. Dependiendo de si la bola se dirige a posterior hacia una raqueta u otra, el Photon View se transferirá a dicho jugador para que pueda ver venir la bola sin retraso, facilitando un juego fluido para ambos jugadores.

Si la bola colisiona con la pared inferior o superior, invertirá su velocidad en Y, para que vaya en dirección de arriba a abajo o viceversa. También, emitirá un sonido, pero distinto al que suena al golpear una raqueta.

Si la bola colisiona con la pared izquierda o derecha, significa que se ha marcado un punto. Para actualizar la puntuación, se llama a la función del controlador de Pong con el mismo nombre. Posteriormente, se emite un sonido dando a entender que se ha marcado punto, y se reinicia la posición de la bola, junto con una animación. Si puntúa en la derecha, la bola saldrá de la derecha hacia la izquierda, y si puntúa en la izquierda, viceversa. Cuando la bola se reinicia, también lo hace su velocidad, por lo que sale con cualquier ángulo.

Respecto a la velocidad de la bola, cuando cambia la velocidad, esta lo hace de forma aleatoria, pero limitada a un conjunto de direcciones, según se dirija a la izquierda o a la derecha.

Comentaremos la función *Update*. Esta función es bastante sencilla, ya que únicamente controla la posición de la bola según su velocidad, y también controla la animación de reaparición según el tiempo restante del temporizador dedicado a la animación.

Por último, mencionar que las RPC se utilizan para sincronizar entre usuarios los sonidos, el Photon View y la animación.

Comportamiento del Pong general

El script *Pong Behaivour* es el encargado de gestionar lo que ocurre en la selección de modo, actualizar la puntuación, gestionar la pantalla, etc. Contiene diferentes variables, sobre todo apuntadores a los distintos componentes del juego, como pueden ser los mandos, la bola y los distintos lienzos. Además, tenemos variables para saber si el primer jugador es de PC o RV, el temporizador de la ventana que anuncia al ganador, y booleanos para saber el modo de juego, si ha habido un ganador o si el juego está en marcha.

Empezaremos explicando los tres estados que puede tener lo que muestra la pantalla del juego: la selección de modo contra la IA o contra otro jugador, el visualizado del transcurso del juego y la pantalla del ganador.

La selección de modo sucede durante la función de *Update*. Pare ello, se comprueba que el juego no haya arrancado y que el primer jugador haya cogido el mando. Entonces, este espera el input de dicho jugador, ya que el primer jugador es el único que puede elegir el modo, el segundo jugador no puede. Dependiendo de si el jugador es de PC o RV, tendrá unos inputs u otros. El jugador de PC puede cambiar la selección con las teclas W o S para subir o bajar respectivamente, y para seleccionar la elección pulsar la tecla E. En el caso del jugador de RV, este cambia la selección con el joystick del mando izquierdo. Un valor positivo del eje Y para subir y un valor negativo para bajar, para seleccionar la elección pulsaremos el botón principal del mando derecho.

Una vez hemos hecho la selección del modo, se realizan una serie de acciones para arrancar el juego. Para empezar, ocultamos la pantalla de selección de modo, dando paso al campo del Pong. A continuación, indicamos a los mandos y a la bola que empieza el juego, además, en el caso de la bola, llamamos a su función de reinicio, para darle tiempo a los jugadores a posicionar sus raquetas antes de empezar la partida. También llamamos a las RPC para sincronizar todas las escenas.

Una vez el juego ha empezado, en el caso de que se haya elegido el modo de dos jugadores, se cede el control para que las raquetas y la bola controlen el estado del juego. En el caso del modo contra la IA, el controlador del Pong es el encargado de controlar la raqueta del segundo jugador. Esta IA tiene un comportamiento simple, ya que sigue el movimiento de la bola y sube o baja la raqueta dependiendo de su posición.

Miramos ahora las funciones que restan, que son la de actualizar la puntuación y la de reiniciar el juego.

Para empezar, cogeremos el texto de las puntuaciones para poder manejarlos directamente en vez de acceder cada vez a ellos a través del lienzo. Cuando la puntuación se tiene que actualizar, la función recibe un booleano que indica si ha marcado el jugador uno o el jugador dos. Se suma un punto al contador del jugador en cuestión, y se comprueba si el contador ha llegado a tres.

Si el contador ha llegado a tres, realizamos dos acciones: activamos la pantalla de ganador, con un mensaje indicando cuál de los dos jugadores ha ganado y llama a la función de reiniciar el juego, la cual explicaremos a continuación. Una vez se ha acabado el tiempo de la pantalla del ganador, esta desaparece dando pasa de nuevo a la pantalla de selección de modo.

Figura 3.12: Dos jugadores jugando al videojuego Pong

Respecto a la función de reiniciar el juego, simplemente informa a los mandos y a la bola que el juego ha terminado. Aparte de cuando la puntuación de uno de los jugadores llega a tres, esta función también se llama en la función *ActualizarEstadoSalida*.

Por último, las RPC de esta clase sincronizan cuál de las tres pantallas están activas, las puntuaciones de los jugadores y actualizar el estado del juego para las raquetas y la bola de las instancias del resto de usuarios.

Con esto hemos explicado todo el funcionamiento del videojuego Pong. Como hemos podido ver, el juego está estructurado de tal forma que tenemos un controlador central que se encarga de mandar y recibir información de los implicados en el campo, que son la bola y las raquetas, y actualiza el juego en consecuencia. Podemos ver a dos jugadores jugando a este videojuego en la figura 3.12

3.4.6. Videojuego de bolos

El bowling es uno de los juegos de salón y recreativos más populares. Este tipo de juego fue adaptado por Nintendo en su videojuego Wii Sports, que pese a ya existir otros similares creados anteriormente, este fue el que mayor éxito alcanzó gracias a este conjunto exclusivo para la Wii. El objetivo de recrear este videojuego en el museo es de volver a reunir a personas para que disfruten de la experiencia vivida en familia hace años.

Adaptación y estructura

Para adaptar este videojuego al entorno de Unity hemos necesitado cinco elementos diferentes. Primero de todo, y a diferencia de los otros dos videojuegos, utilizaremos dos cámaras con perspectiva, ya que tendremos un entorno 3D donde iremos intercambiando las dos cámaras: una para cuando el personaje vaya a lanzar la bola y otra para cuando se cambie el turno donde se presenta al siguiente personaje. Utilizaremos cuatro tipos de modelos distintos: Un modelo de una bolera donde transcurrirá el videojuego, una bola de bolos, un conjunto de pinos y el cuerpo de los personajes. Podemos ver una captura de como se ve la bolera gracias a la figura 3.13 Además, tendremos varios paneles con la Interfaz de Usuario (IU), entre ellos el panel de inicio, la tabla de puntuaciones o la pantalla de espera.

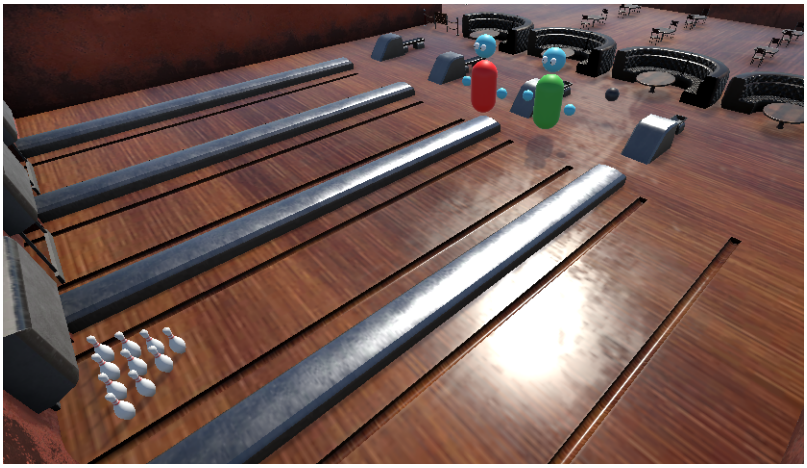

Figura 3.13: Captura de la bolera

Sistema de puntuación y turnos

El videojuego transcurre en una sucesión de turnos, dependiendo si se trata de uno o dos jugadores. En general, podemos ver la transición de turnos gracias a la figura 3.14. Un turno se compone de dos partes. Si en la primera parte se consigue derribar todos los pinos, se contará como pleno, y la puntuación final de ese turno será de veinte. Si no se consiguen derribar todos los pinos en la primera parte, habrá otro intento para volver a tirar la bola. Si en el segundo intento se consiguen derribar todos los bolos, se contará como semi pleno, y la puntuación final será de quince. Si no se consiguen derribar todos los pinos en el segundo intento, la puntuación final del turno será únicamente el número de pinos derribados. La puntuación final de la partida será la suma de los cinco turnos, hasta un máximo de cien puntos, y en el caso de que la partida sea de dos jugadores, se anunciará quien de los dos ha ganado.

Si la partida es de solo un jugador, los turnos serán sucesivos hasta el final. Como cada turno tiene 2 partes, es posible que el jugador complete la partida con solamente cinco tiradas o hasta un máximo de diez tiradas.

En el caso de que la partida sea de dos jugadores, los turnos de cada uno de ellos se irán intercalando hasta completar un total de diez turnos. Cada jugador tiene en su turno dos partes. Si en la primera parte el primer jugador consigue derribar todos los pinos, se pasará al turno del segundo jugador, y viceversa. Lo mismo ocurre con la segunda parte. Tanto si se consiguen derribar todos los pinos o no, al acabar la segunda parte continuará el siguiente jugador con su turno. Una vez se completan los diez turnos, aparece la tabla con todas las puntuaciones, incluida la puntuación total de la partida, y un mensaje indicando que jugador ha ganado

Comportamiento del mando y los personajes

El personaje de este videojuego consta de un conjunto de figuras básicas como pueden ser un cilindro y varias esferas. Al poder ser dos jugadores, hay dos jugadores diferenciados, uno de color verde y otro de color rojo. Podemos ver el modelo del personaje en la figura 3.15.

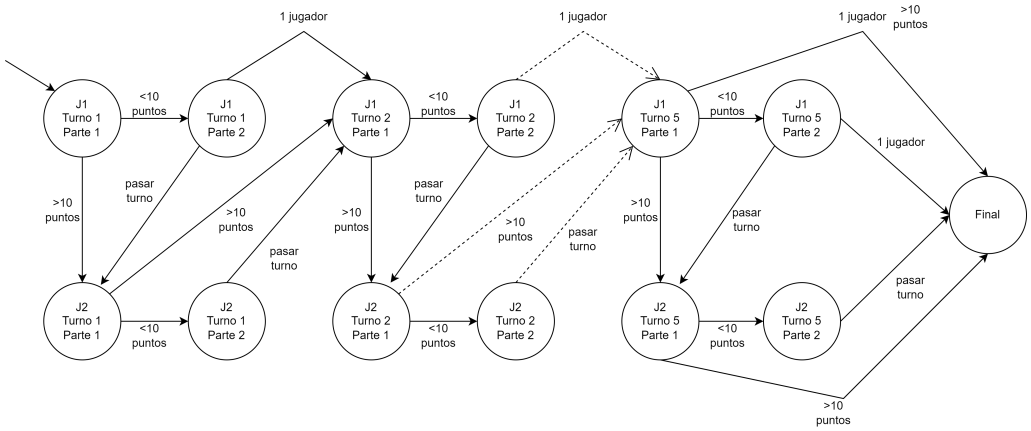

Figura 3.14: Diagrama turnos videojuego bolos

Para comprender el comportamiento de estos personajes y como reciben el input del jugador, veremos el script principal *MandoBolosBehaivour*. Para empezar, veremos las variables, algunas de las cuales son similares a las de otros mandos. Tendremos un booleano para indicar si el jugador utiliza input RV. Además, a diferencia de otros videojuegos en los que la diferencia entre los inputs de RV o teclado y ratón se basaban en las teclas o botones que había que recibir, aquí también necesitaremos saber, en el caso de RV, con cuál mano se ha cogido el mando, ya que esto se traducirá en la mano con la que el personaje cogerá la bola de bolos. Dicho esto, necesitaremos un puntero al cuerpo del personaje y de sus manos. Además, necesitaremos guardar el objeto de la mano con la que hemos cogido el mando, para así poder luego reproducir el movimiento con el que lanzará la bola el personaje. Junto a estas variables principales, necesitamos unas auxiliares como pueden ser un booleano para saber si el juego está en marcha, si ya hemos tirado la bola, un puntero a la bola y al comportamiento del juego general. Por último, tenemos también una variable float con la que recogeremos la rotación inicial del jugador cuando coge el mando, ya que de ello dependerá la rotación relativa a la hora de lanzar la bola, lo cual analizaremos más adelante en este mismo apartado.

Siguiendo con las funciones que hemos visto en otros scripts como son *ActualizarEstadoEntrada* y *ActualizarEstadoSalida*, decir que la funcion de salida la comentaremos en más profundidad más adelante, en el sistema de espera. Respecto a la función de entrada, vemos varias diferencias respecto al comportamiento de los otros videojuegos. Esta vez, cogemos con que mano va a lanzar el personaje. Esto dependerá del jugador, ya que si el jugador usa los inputs de teclado y ratón, automáticamente se le asignará la mano derecha, mientras que el usuario de RV hará que el personaje coja y lance la bola con la misma mano que el jugador ha cogido el mando. Además, en el caso del jugador RV recogeremos la rotación con la que cogió el mando, utilizando los ángulos de Euler, en concreto el valor en Y.

Ahora pasaremos a hablar de la función *Update* y como se controlan los personajes segun el input del jugador.

Primero de todo, comprobaremos que el juego está en marcha, y dependiendo de

si tenemos input de teclado y ratón o RV actuaremos de forma totalmente diferente.

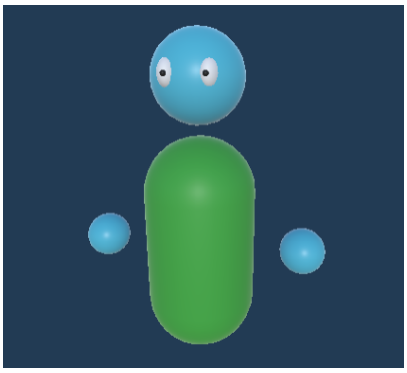

Figura 3.15: Modelo personaje juego de bolos

- **Teclado y ratón**: El jugador que esté utilizando teclado y ratón no tendrá tanta libertad como el jugador RV a la hora de lanzar la bola, ya que este no podrá mover directamente la mano del personaje, para ello tendremos dos modos que podrá utilizar para ayudarle a apuntar, además de un modelo auxiliar para saber la dirección. Este modelo se trata de una figura plana de una flecha, que va desde el centro del personaje, situada en el suelo, y apunta hacia donde saldrá la bola. Los dos modos se tratan de uno en el que el personaje se puede mover de izquierda a derecha, para poder posicionarse donde quiera de la pista. El otro modo de movimiento se trata de uno rotatorio, con el que el personaje rotara sobre su eje central Y, para poder apuntar a izquierda o derecha de la pista pero sin moverse de su posición. Estos dos modos se intercambian pulsando la tecla de espacio.

  Por otro lado, para lanzar la bola, lo que hacemos es liberar la bola de las manos del personaje y hacerla hija de la bolera. Siguiente, la bola adquiere la rotación que tenía el personaje, para así poder apuntar hacia donde este estaba apuntando. Por último, llamamos a la función *LanzarBola*, la cual añadirá una fuerza con la que se impulsa la bola.

- **RV**: A diferencia del input del teclado y ratón, el personaje se quedará quieto en su sitio y no se podrá desplazar. A cambio, dispondremos del control directo de su mano a través de uno de nuestros mandos del dispositivo RV como si estuviéramos realmente jugando con un mando de Wii. Para ello, cogeremos la posición local de las manos del jugador respecto al centro del modelo, y la trasladaremos a la mano del personaje, haciendo así que se mueva al mismo tiempo que el jugador mueva su mano.

  Para lanzar la bola, y de forma similar a como se hace en teclado y ratón, pulsaremos el botón principal del mando del dispositivo RV con el que hayamos cogido el mando de la Wii. Entonces, liberaremos la bola de la mano del personaje y calcularemos la rotación que esta debe tomar respecto a la rotación actual del jugador y su rotación iniciar a cuando cogió el mando de la Wii. Como estamos utilizando ángulos de Euler para este cálculo, deberíamos ver como funcionan

estos. En la figura 3.16 podemos observar que, si miramos hacia adelante, dejando la parte roja a la izquierda y la azul a la derecha, es donde tenemos nuestro origen del ángulo cero, dando la vuelta completa sería 360. Así entonces, para poder lanzar la bola hacia delante, deberemos ver cuál es el ángulo de Euler. En el caso de que fuera un ángulo cercano a cero, la bola se lanzaría adelante, pero ligeramente a la derecha. Si fuera un ángulo cercano a 360 desde la izquierda (358, por ejemplo), significaría que hay que lanzar la bola hacia adelante, pero ligeramente a la izquierda. Siguiendo esta lógica, los ángulos entre 0 y 180, la bola tendría que dirigirse hacia el lado positivo del eje X, mientras en los ángulos de 180 a 360 la bola se dirige al lado negativo del eje X.

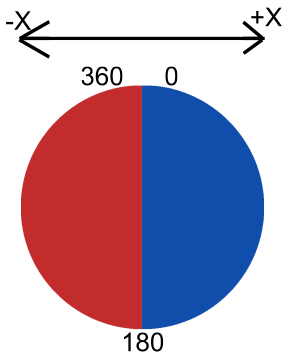

Figura 3.16: Ángulos de Euler en Unity

Para finalizar la explicación del comportamiento del mando del juego de los bolos, hablaremos de dos funciones auxiliares y de las funciones RPC.

Como hemos dicho en la parte de los inputs de teclado y ratón, para facilitar el saber hacia donde está apuntando el personaje, contamos con una flecha que apunta en su dirección. Esta flecha se tiene que activar cuando el jugador vaya a lanzar, y desactivarse cuando otro jugador está lanzando o cuando se está presentando a su personaje. Para ello, la función *ActualizarFlecha* tiene el deber de activar o desactivar el objeto de la flecha según un parámetro booleano de entrada.

La otra función auxiliar se llama *CogerBola*, y su cometido es el de vincular la bola con la mano del personaje, así como parar cualquier movimiento que esta pudiera tener.

Para terminar, tenemos un gran número de funciones RPC, las cuales analizaremos por encima, dado que la mayoría son copias de otras funcionalidades ya vistas que se ejecutarán en las instancias del resto de usuarios, a excepción de las funciones de espera, las cuales veremos en su correspondiente sub apartado.

De las funciones RPC más destacadas, tenemos *RPCActualizarFlecha*, cuya funcionalidad es la misma que la función auxiliar mencionada anteriormente. *RPCLanzarBola*, la cual libera la bola de la mano del personaje para que posteriormente el Photon View Transform se encargue de sincronizar su posición a la del resto de usuarios. *RPCCogerBola*, la cual funciona igual que el método auxiliar antes mencionado. *ActualizarManoElegida*, la cual se encarga de, pasado un parámetro booleano para

indicar si se trata de la mano izquierda o derecha, guarda en la variable *manoElegida* la mano izquierda o derecha del personaje para la correcta sincronización de este.

El resto de funciones son las ya vistas en los otros videojuegos, como actualizar el estado de jugando, sincronizar la entrada del mando, etc, por lo que con esto, terminamos la explicación de la implementación del comportamiento de los personajes y mandos.

Comportamiento de la bola de bolos

La bola de bolos se trata de prácticamente una esfera completa, sin contar los tres agujeros que suele tener este tipo de bolas. El comportamiento de la bola proviene únicamente del script *BolaBolosBehaivour*, y a continuación explicaremos lo que contiene este script.

Aparte de la función *Update* este script tiene dos funciones más que son importantes: *LanzarBola* y *PararBola*. La función que lanza la bola es llamada por el script del mando, y con el paso de parámetros para la velocidad en X, para saber si la bola ha de ir a la izquierda o la derecha, añade una fuerza al Rigidbody de la bola.

La función para parar la bola, como su propio nombre indica, pone la velocidad normal a 0, así como su velocidad angular.

La función *Update* se encarga de manejar el desplazamiento de la bola. Para ello, simplemente se asegura de que la velocidad de la bola sea la adecuada, y deja que el Rigidbody haga el resto, dado que es un objeto con cuerpo y de forma esférica, esta rodará naturalmente.

Con esto concluimos que el control de la bola se basa en el momento en el que la bola es lanzada, después de eso sigue su movimiento natural hasta que haya que volver a pararla.

Sistema de espera

A modo de resumen, este sistema consiste en que durante una partida de dos jugadores en el juego de los bolos, si uno de los jugadores deja el mando, se abrirá una pantalla en la que se indica la espera a que otro jugador recoja el mando para continuar la partida. En la figura 3.17 podemos ver la transición entre los distintos estados.

Para implementar este sistema necesitaremos de la conexión de ambos scripts, tanto el comportamiento del mando como el museo general. El script del mando se encargará de ver el estado en el que se encuentra el sistema de espera en *BolosBehaivour*, y en el método de *ActualizarEstadoSalida* veremos las distintas posibilidades. Si somos el jugador uno y soltamos el mando: Si estamos jugando solos, finaliza el juego. Si no estamos jugando solos, pero nos indican que el otro jugador ya ha soltado el mando, finaliza el juego. Si no hay otro jugador desconectado, indicamos que el jugador uno estará en espera, y lanzamos la ventana con el mensaje. Si somos el jugador dos y soltamos el mando: Si el jugador uno ya había soltado el mando, terminamos el juego. En caso contrario, indicamos que el jugador dos estará en espera y lanzaremos la ventana con el mensaje.

Ahora bien, para comprobar si se puede continuar la partida una vez un jugador ha soltado el mando, no será el mando al entrar quien notificará que ha vuelto, será

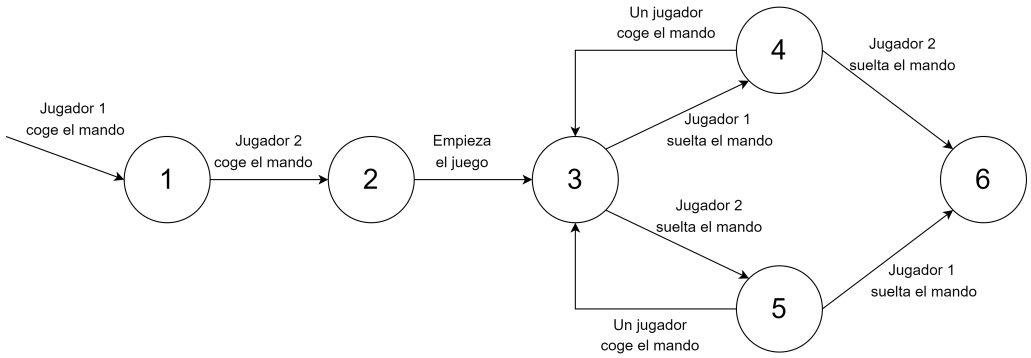

Figura 3.17: Diagrama de estados Sistema de espera

el script principal del comportamiento del juego quien cerrará la ventana de espera y notificará a los mandos que el juego se ha reanudado.

Comportamiento del juego de bolos general

El comportamiento del juego general está dividido en varios scripts, pero principalmente en *BolosBehaviour*. El resto de pequeñas funcionalidades están repartidas en los scripts de *CongelarPinos*, *PinoAux*,*TriggerBola* y *TriggerPinos*. Aunque la mayoría de ellos se explican con su propio nombre, vamos a explicarlos uno por uno.

- **CongelarPinos**: Como su propio nombre indica, congela o descongela los pinos según se necesite. Por norma general, los pinos están congelados, y no es hasta que la bola de bolos pasa por un trigger, al cual va vinculado este script, que los bolos se descongelan para que estos puedan ser derribados por la bola.

- **PinoAux**: Este script está incorporado en todos los pinos, y se encarga de modificar su Rigidbody en el caso de que los pinos tengan que estar absolutamente quietos. Para facilitar esta tarea, hacemos uso del atributo *RigidbodyConstraints* del Rigidbody. Con este atributo, podemos bloquear el movimiento del pino en todos los ejes y rotaciones. Si le indican que no tienen que estar quietos, quita estas restricciones.

- **TriggerBola**: Este script, vinculado a un trigger al final de la pista de bolos, detecta cuando la bola ha pasado más allá donde se encuentran los pinos. Cuando esto ocurre, la función empieza un contador, para darle tiempo a los bolos que estén a medio caer a que terminen de caer. Cuando termina el tiempo, llama a la función *BolaTrigger* del script principal.

- **TriggerPinos**: Este script está vinculado a un trigger que rodea la parte superior de todos los pinos, de tal manera que cuando uno de ellos cae derribado por la bola de bolos, este detecta su salida, aumentando la puntuación de esa parte en uno. La suma de la puntuación la hace directamente sobre la variable *puntuaciónActual* del script principal.

A continuación, pasamos a explicar el script principal del comportamiento del juego de bolos, *BolosBehaivour*. Empezaremos comentando las variables más comunes, hasta las más importantes.

En primer lugar, tenemos el booleano que indica si el primer jugador tiene input RV o no, si el juego está en marcha y si la partida es de uno o dos jugadores. Tenemos punteros a distintos objetos, ya que este videojuego es uno de los que requieren la coordinación de un gran número de ellos. Tenemos punteros a los distintos objetos de la IU como el menú de selección, la tabla de puntuaciones, las ventanas de espera y el panel del ganador. Como dijimos anteriormente, en este videojuego contamos con dos cámaras con perspectiva, estas van cambiando según haya que presentar al jugador o tirar la bola. También tenemos punteros a distintos prefab y objetos, como pueden ser el conjunto de pinos, la bola de bolos, los personajes del juego, el mando de los jugadores y el trigger de *CongelarPinos*. A esto, se le suma distintas variables para controlar los timers entre cambios de cámaras, control de rondas, segundas tiradas, posiciones de los objetos, etc. Por último, los booleanos para conocer si hay algún jugador a la espera.

La mayoría de funciones que manejan el script tienen un gran impacto, por ello vamos a explicar desde las más extensas a las menos extensas, empezando por *Update*. Podemos dividir esta función en tres grandes bloques. Uno se centra en controlar las transiciones de cámaras, otro en manejar el sistema de espera, y el último en recibir el input en la pantalla de selección de juego y empezar la partida. En el bloque de manejo de transiciones y cámaras, tenemos el cambio de cámaras entre presentación y lanzamiento. Durante la presentación, el cuerpo del personaje es sólido y se muestra la puntuación actual de la partida en la tabla. Cuando la cámara cambia a la de lanzamiento, el personaje se vuelve semi transparente, para así poder hacia donde estamos a punto y hacia donde se dirige la bola. El bloque del sistema de espera se encarga de comprobar que si hay alguno de los dos jugadores en espera, pero detecta que los dos mandos están cogidos, entonces reanuda la partida, indicando al jugador que tenga su turno que puede jugar. Por último, el bloque que recibe los inputs es similar a otros que hemos visto. Tenemos una distinción entre los inputs de teclado y ratón y los inputs en RV. Podemos seleccionar, moviendo a izquierda y derecha, el modo para un jugador y el modo para dos jugadores. Una vez seleccionado el modo con la tecla E o uno de los botones del mando RV se inicia la partida. Al empezar la partida, se recolocan los pinos y se inicia el cambio de cámaras a presentación y luego lanzamiento. Podemos observar en la figura 3.18 como se ve la partida en la cámara de lanzamiento. Una vez la partida está en marcha, la función principal del comportamiento del juego recae sobre la función *PasarTurno*, la cual explicaremos a continuación.

En la función de pasar turno verificaremos principalmente dos cosas para actuar de una forma u otra. Primero, miraremos si la partida es de uno o dos jugadores. Si la partida es de solo un jugador, simplemente pasaremos al siguiente turno, moviendo las cosas a su sitio y guardando la puntuación del turno del jugador. En el caso de dos jugadores, cambiaremos el turno al del siguiente jugador y realizando los mismos pasos que en un solo jugador. Cuando los dos jugadores hayan hecho su turno, se pasa también de ronda, aunque a efectos prácticos se cambia el turno al del otro jugador, solo que las siguientes puntuaciones se guardaran en la siguiente casilla de la tabla de puntuaciones. Esto se repite hasta que hayan pasado las 5 rondas, en cuyo caso se finaliza la partida. Al finalizar la partida calculamos la puntuación total de ambos

jugadores y se muestra el panel del ganador.

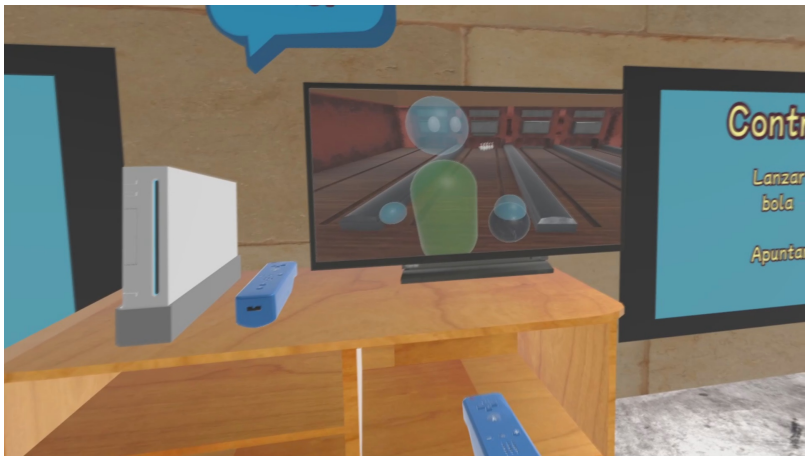

Figura 3.18: Captura de la pantalla de lanzamiento en el juego de bolos

La siguiente función que explicamos es *BolaTrigger*, la cual llamamos cuando la bola ya ha derribado a los pinos y han pasado cierto tiempo. En esta función se consideran distintos escenarios, los cuales explicamos en la sub sección del sistema de puntuación y turnos. Lo único que añadiremos es que en el caso de que saquemos menos de diez puntos volveremos a tirar, en cualquier otro caso pasaremos de turno.

Las siguientes funciones se explican con su propio nombre. *CambiarCamara* activa una cámara u otra según se necesite, y retransmite la cámara que toque a la pantalla. *MoverPersonaje* únicamente desplaza a un personaje u otro al centro de la pista según el turno de la partida. *VolverATirar* simplemente le da la bola al jugador que tenga su turno. *PersonajeVisible* y *PersonajeSemitransparente* tienen una función similar. Ambos modifican el material de los objetos que componen al personaje, cambiando su valor de transparencia. Para hacerlo completamente visible, el valor de transparencia se pone a uno, para hacerlo semitransparente lo ponemos a 0.3.

VentanaEspera y *OcultarVentanaEspera* cumplen la funcionalidad de sus propios nombres. Mostrando u ocultando la ventana del sistema de espera, y cambiando su contenido según si ha sido el primer o el segundo jugador quien se ha desconectado.

También, tenemos dos funciones auxiliares, llamadas *CalcularPuntuaciónTotal* y *VaciarTabla*, que mediante el uso de for, calcula la puntuación total de los dos jugadores en la tabla o la vacía, respectivamente.

Por último, la función *FinalizarJuego* será llamada una vez se ha anunciado el ganador de la partida, y transcurrido cierto tiempo, reiniciará todas las variables y paneles, dejando a la vista de nuevo el panel de selección de dificultad, para así poder iniciar una nueva partida.

Para terminar con la explicación del script principal del comportamiento del juego de bolos, comentar las funciones RPC, las cuales son una copia de las funciones que hemos ido mencionando, las cuales se llaman normalmente dentro de estas mismas funciones para que puedan ser ejecutadas en diferentes estancias del resto de jugadores.

3.4.7. Videojuego plataformas

El último videojuego que hemos creado para que los jugadores puedan jugar en el museo se trata de un juego sencillo de plataformas 2D. En este juego controlamos a un personaje cuyo objetivo es llegar a la meta, recorriendo un nivel con plataformas fijas, con un tiempo máximo para lograrlo.

En este nivel, encontraremos el mapa de nivel, compuesto por un suelo por el cual el personaje no puede caer y un fondo meramente visual. A lo largo del nivel encontraremos cuatro objetos interactuables: unos enemigos en forma de slimes, un número de monedas, un trampolín en forma de seta y una meta.

Adaptación y estructura

Para adaptar este videojuego a Unity, hemos planteado un acercamiento en un entorno 2D. Para ello, hemos utilizado las herramientas que proporciona Unity para los entornos 2D como son el grid, los tilesets y los sprites 2D. Además, para capturar la imagen y transmitirla a la pantalla al lado de la consola empleamos una cámara ortográfica.

La estructura básica del videojuego se compone de un grid y un panel que contiene la interfaz de usuario.

Dentro de la interfaz de usuario, tenemos tres paneles distintos: el panel de inicio, donde tenemos el título del juego y el botón de jugar, el panel mientras está el juego en marcha, con el número de vidas del jugador, el número de monedas y el tiempo restante, y el panel de victoria/derrota. En el panel de victoria, además, veremos el tiempo restante de la partida y el número de monedas conseguidas.

Por otro lado, los elementos que componen la jugabilidad del videojuego están dentro del grid. Dentro del grid encontramos dos tilemaps: uno que compone el suelo del nivel, y otro que compone el fondo y no es interactuable. En el nivel del suelo utilizamos un tileset que se asemeja a la tierra, y es por donde el jugador podrá pisar. Así mismo, aunque nos referimos al suelo, este tilemap también compone paredes y techos, por lo cual el jugador no podrá avanzar si se choca con una pared o techo, ya sea caminando o saltando.

Otro elemento diferenciado que encontramos dentro del grid son el conjunto de slimes. Estos slimes se mueven de derecha a izquierda en un intervalo de tiempo. Si el jugador toca a un slime con su parte inferior los derrota, si lo toca con uno de sus lados, recibe daño y es repelido.

El otro conjunto de elementos que tenemos dentro del grid son las monedas. Hay varias de ellas en todo el escenario, y el jugador puede recogerlas, aumentando el contador de monedas.

Por último, los tres únicos elementos individuales dentro del grid son el jugador, el trampolín en forma de seta y la meta. El jugador puede interactuar con estos elementos de manera que si su parte inferior colisiona con la seta, este se verá impulsado hacia arriba, y si llega a la meta habrá completado el juego.

Comportamiento del mando y el personaje

Al ser el videojuego de un solo jugador, únicamente habrá un mando que controlará al personaje. La función principal del mando será recibir el input del usuario y mover

al personaje acorde a su entorno y estado.

Los scripts que conforman al personaje están divididos en dos tipos: El script general y central que se encarga de recibir la información del segundo grupo, así como de actuar en función de los inputs del jugador, y los scripts vinculados a unos trigger que detectan la entrada de un elemento abajo, izquierda, derecha y arriba del personaje. Estos trigger pueden verse en la figura 3.19.

Los scripts que se vinculan a estos trigger son, como su mismo nombre indica, *DetectarSuelo*, *DetectarTecho*, *DetectarParedIzq* y *DetectarParedDer*. Cada uno de ellos cumplen una función distinta, pero todos tienen en común el uso de *OnTriggerEnter2D* y *OnTriggerExit2D*. A continuación explicamos cuál es la función de cada uno:

- **DetectarSuelo**: El uso principal de este script es para conocer si el personaje está tocando suelo o incluso otro elemento que no sea suelo, pero si sea interactuable. Si el trigger vinculado a este script detecta que está colisionando con algo, miraremos distintas posibilidades: Si el objeto colisionado está en la capa 9 (suelo) guardamos en el booleano *isGrounded* el valor true, indicando que estamos tocando el suelo. Si el objeto está en la capa 10 (enemigo) indicamos al script del enemigo que como le estamos pegando desde arriba, entonces lo hemos derrotado, además, daremos un pequeño salto como respuesta al impacto. Por último, si el objeto está en la capa 11 (trampolín) indicamos al personaje que de un salto mayor de lo normal. Por otro lado, si el trigger ha dejado de detectar el suelo, guardamos en el booleano *isGrounded* el valor false, indicando que no estamos tocando el suelo.

- **DetectarTecho**: La única función de este script es que si el trigger entra en contacto con el suelo, en este caso seria que estamos tocando un techo, indique al personaje que su velocidad es 0 y, por lo tanto empezará a decaer nada más choque contra el techo, haciendo así que no se pueda mantener mucho tiempo en el aire.

- **DetectarParedIzq y DetectarParedDer**: Ambos scripts tienen el mismo comportamiento, con la única diferencia en el nombre del booleano y en el salto. Si el trigger detecta un objeto en la capa de suelo, pone el valor true al booleano isIzqTouch o isDerTouch. Si el trigger detecta un objeto de la capa enemigo, quitaremos una vida al personaje y daremos un pequeño salto hacia atrás en respuesta a la colisión. En el caso de que salgamos de la colisión con una pared, pondremos el valor de dicho booleano a false.

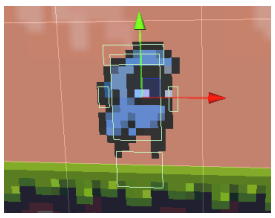

Figura 3.19: Sprite y colliders del personaje

Para terminar con la implementación del personaje, vamos a hablar de su script principal: *PersonajePlataformasBehaivour*. Empezaremos hablando de su similitud con el resto de controladores de personajes, como son las funciones *ActualizarEstadoEntrada* y *ActualizarEstadoSalida*. En la función de entrada, como siempre, comprobamos si el input que estamos usando es el de RV o no y actualizamos el valor de dicho booleano, luego transferimos el Photon View del personaje y el mando al jugador. En la función de salida, al ser un videojuego de un único jugador, en el caso de que soltemos el mando, simplemente la partida se acabará, llamando a la correspondiente función para terminar la partida.

Si miramos ahora las variables de este script, podremos ver algunas similares al resto de scripts de los mandos, como el booleano para saber si el input es RV o no, punteros al script del comportamiento general del videojuego, al objeto que representa el cuerpo del personaje, si el juego está en marcha... Por otro lado, tenemos variables únicas, como por ejemplo un puntero al animator del personaje, su velocidad en Y (salto), un puntero al contenedor de corazones, el estado actual del animator, un conjunto de constantes con los nombres de los estados del animator y unos timers para la invulnerabilidad y para controlar que no se abuse del salto en los primeros milisegundos.

Pasando a las funciones únicas de este script, tenemos las cuatro principales: *Update*, *CambiarAnimacion*, *Salto* y *QuitarVida*. Empezaremos hablando de la función que cambia las animaciones. Esta recibe como parámetro de entrada el string con el nombre del nuevo estado de animación, este estado dependerá de los inputs y el entorno del personaje, que veremos mejor cuando expliquemos el método *Update*. El comportamiento de esta función es sencillo. Si el estado actual es igual que el nuevo estado al que se quiere cambiar la animación, no hace nada. Si por otro lado es distinto, ejecutaremos el nuevo estado con la función *Play* del animator, y asignaremos al estado actual, el nuevo estado.

La función de salto únicamente reinicia las velocidades del Rigidbody2D del personaje a 0, después añade la fuerza al Rigidbody según los dos parámetros de entrada de la función, que son las velocidades en X e Y.

Para acabar con las funciones sencillas, tenemos la función de quitar vida, que como su propio nombre indica, quitamos una vida del personaje. Para hacer un control de las vidas, miramos directamente los objetos que se encuentran en la IU, por lo que si el objeto contenedor de los corazones no tiene más hijos, significará que no nos quedan vidas. Cuando nos quitan una vida, ponemos además el booleano de invulnerabilidad a true, e iniciamos el timer. Cuando este timer termine, la invulnerabilidad se terminará

Por último, tenemos la función *Update*, la cual se encarga de recoger los inputs del usuario y de procesar el tiempo de invulnerabilidad. Primero de todo, recoge el estado de los booleanos de los scripts de detección antes mencionado, así como del input *RV*. Después de recoger estos datos, los procesa y según el estado: Si se pulsa a la derecha, el personaje camina a la derecha siempre y cuando no colisione con una pared. Si se pulsa a la izquierda, pasaría lo mismo que la derecha. Ahora bien, a la hora de estar tocando el suelo, pueden pasar varias cosas. La primera seria que estemos tocando el suelo pero no recibimos ningún input para saltar, entonces el personaje se queda quieto mostrando su animación de idle. Si estamos tocando suelo y pulsamos el botón de saltar, se cambia a la animación de salto hacia arriba, y añadimos fuerza al Rigidbody2D para que se impulse hacia arriba. El último caso que tenemos, es que el personaje tenga una velocidad negativa (está cayendo), entonces le cambiamos la

animación a la de caída.

Respecto a la sincronización de variables, aparte de las funciones ya vistas en otros scripts, tenemos los RPC correspondientes a las funciones *ActualizarAnim* y *QuitarVida*. Además, tenemos otra para cambiar la dirección a la que mira el personaje, puesto que esa variable no se actualiza desde el *Photon View Transform* al tratarse de un sprite 2D. Con esto concluiría la explicación del comportamiento del personaje.

Comportamiento de los enemigos y las monedas

Empezaremos este apartado hablando del comportamiento de las monedas. Podemos saber como se ven estas gracias a la figura 3.20. Estas tienen un puntero al comportamiento general del juego y un puntero a su animator. La moneda tiene dos animaciones, una idle y otra cuando es cogida. Lo único destacable de su script es la función *OnTriggerEneter2D*, donde si el trigger de la moneda recibe una colisión, y esta se trata del personaje, llamará a la función *IncrementaMoneda* del juego y cambiará a su animación de que ha sido cogida. Además, el objeto se destruirá al acabar la animación y se elimina el collider para que así esta función no se pueda activar más veces antes que se destruya.

Por otra parte, tenemos el comportamiento de los enemigos. Los enemigos de este juego son slimes, y estos se mueven a izquierda y derecha, acompañados de animaciones mientras están quietos, cuando se mueven y cuando son derrotados. Podemos ver un slime en la figura 3.21.

Podemos observar que tiene distintas variables para poder controlar estos comportamientos, principalmente tenemos el nombre de las animaciones, un booleano para saber si está caminando o no, la dirección a la que va, unos timers para controlar el tiempo en el que está caminando y otro para cuando está quieto, y un puntero al comportamiento general del juego.

En la función *Update*, mientras el juego está en marcha, iremos intercalando el comportamiento de caminar o estar quieto. Si está caminando irá avanzando en la dirección que lleve en ese momento, si está quieto simplemente dejaremos pasar el tiempo hasta que tenga que volver a caminar, en cuyo caso cambiaremos la dirección a una distinta de la anterior e iniciaremos de nuevo el tiempo en el que está caminando.

La última función de este script se trata de *MatarSlime*, la cual es llamada como vimos en los scripts del personaje, cuando este cae sobre su parte inferior sobre el slime. Esta función simplemente ejecuta la animación de muerte del slime, y al igual que las monedas, se destruirá cuando acabe la animación y desactivamos su colisión.

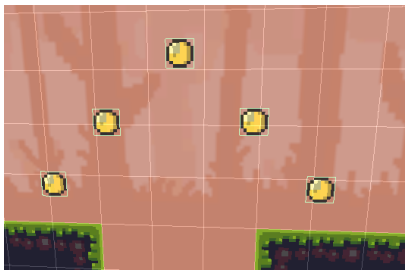

Figura 3.20: Sprite y collider de las monedas

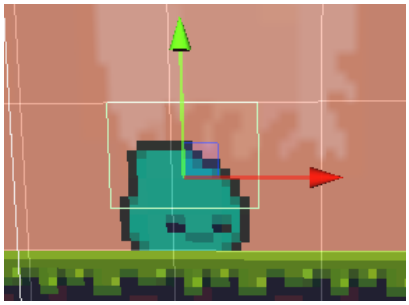

Figura 3.21: Sprite y collider del slime

Comportamiento del plataformas general

Empezaremos esta sección hablando de la cámara que graba el transcurso del juego. Se trata de una cámara ortográfica que sigue al personaje, pues lo que hacemos en la función *Update* es que la posición de la cámara sea la misma que la del jugador, exceptuando del valor z de la posición, el cual siempre es constante. Hay una excepción al seguimiento del personaje, y es que si el personaje cae por un precipicio, la cámara no le seguirá hasta que vuelva a reaparecer. Podemos ver una captura de como se ve el juego desde la pantalla en la figura 3.22

Antes de hablar del script principal del juego, vamos a comentar dos scripts incluidos en unos triggers. Estos son *DetectarCaida* y *DetectarMeta*.

Estos son scripts sencillos, que como su propio nombre indica, detectan la caída del jugador por un precipicio o su llegada a la meta, respectivamente. El script de detectar caída utiliza *OnTriggerEnter2D* y si lo que colisiona con él es el jugador, entonces lo teletransporta a una posición segura indicada posteriormente en el editor, además le quita una vida. Por otro lado, la función de detectar meta es únicamente, también utilizando *OnTriggerEnter*, si es el jugador quien colisiona con la meta, indica al script general del juego que ejecute la función *Win*.

Para finalizar la implementación del videojuego de plataformas, veremos su script principal que controla la mayor parte del comportamiento del juego. Este se trata de *PlataformasBehaviour*, y vamos a empezar viendo sus variables más importantes.

Al igual que otros comportamientos de otros videojuegos, este tiene el booleano que indica si el juego está en marcha, si el primer jugador usa input RV, el puntero a su Photon View y al mando del jugador. Además, tenemos punteros a distintos elementos de la IU, como los paneles de inicio, game over, victoria y el contenedor de corazones y monedas, el texto del timer... También tenemos los prefabs de monedas, enemigos y corazones para instanciarlos cada vez que se inicie la partida, así como las posiciones en las que se instanciarán. Por último, tenemos un timer para mostrar las pantallas de victoria y game over.

Dado que la función principal de este script es ir recibiendo los cambios de estado que le indiquen el jugador y otros objetos del nivel, la función principal *Update* no es muy extensa, por lo que empezaremos por ella. Primero de todo, si nos encontramos en la pantalla de inicio, leeremos los inputs del jugador. Si este pulsa el botón de jugar, el juego comenzará escondiendo el panel de inicio. Por otro lado, mientras estamos jugando comprobaremos distintos estados: Si la partida no se ha acabado, decrementaremos el timer que encontramos arriba a la derecha de la pantalla, el cual indica el tiempo que nos queda antes de que perdamos el juego. Si comprobamos que el timer ha llegado a 0, o si perdemos las tres vidas que tenemos, el juego acabará con un game over. Por otro lado, si conseguimos terminar la partida con éxito, con el booleano finalizando y su timer mantendremos la pantalla de victoria hasta que, finalmente, se termine la partida y volvamos a la pantalla de inicio.

Habiendo visto la función principal, veremos ahora los metodos auxiliares: *IncrementaMoneda*, *Win*, *GameOver* y *TerminarPartida*. En el caso de la función para incrementar monedas, esta únicamente recoge el entero guardado en el texto de la IU y lo aumenta un número. Las funciones de victoria y game over son muy parecidas. La única diferencia es que en el panel de victoria mostramos las monedas que hemos recogido en la partida y cuantos segundos nos han sobrado. Para ello, activamos los

respectivos paneles, y además asignamos a las variables pertinentes que la partida ha finalizado.

La última función, tiene como objetivo realizar una serie de pasos para terminar la partida y dejar todo preparado para cuando empiece la siguiente. Primero, mandamos al personaje a su posición inicial. Seguido de esto, destruimos todos los objetos como las monedas, enemigos y corazones, para luego instanciarlos de nuevo, así evitamos tener que instanciar únicamente aquellas monedas que hemos recogido, enemigos que hemos derrotado o corazones que hemos perdido. Por último, restablecemos los paneles de la IU, indicamos que ya no estamos jugando y reiniciamos el timer de segundos restantes en la partida a cien.

Para terminar con la explicación de esta implementación, decir que al igual que el comportamiento del jugador, hay que sincronizar algunas de las variables y objetos del juego general, por lo que tendremos unas funciones RPC equivalentes a *IncrementaMonedas*,*TerminarPartida*,*GameOver*, *Win* y otra auxiliar para decrementar el timer de la partida.

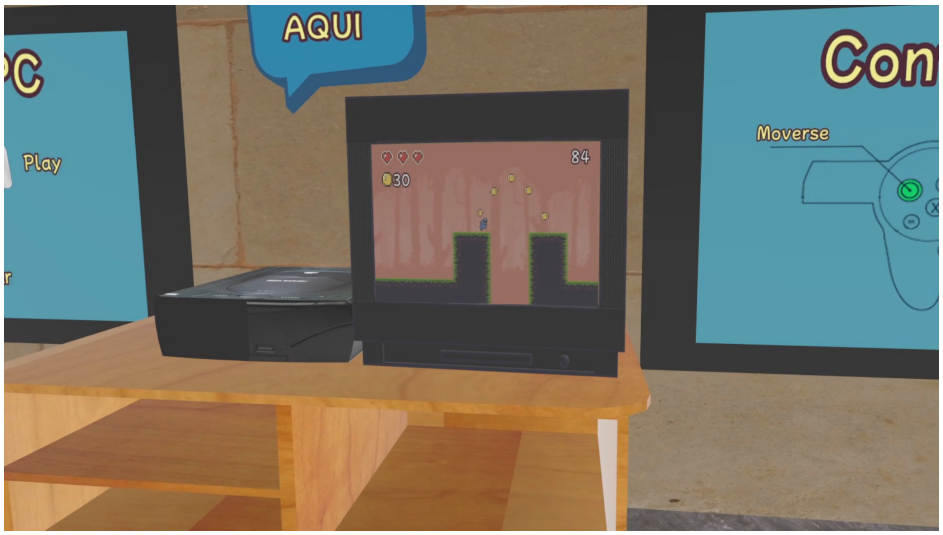

Figura 3.22: Captura videojuego plataformas

3.5. Pruebas de usuarios a lo largo del desarrollo

Durante el desarrollo de la aplicación, múltiples usuarios entraron al museo para poder testear los distintos módulos que se programaron. Entonces, por cada módulo, como se puede ver en el apartado 3.3.4 y subsiguientes apartados de implementación, estos fueron testeados por distintos usuarios para comprobar el correcto funcionamiento de estos. Durante el testeo, si los usuarios encontraban alguna irregularidad o pensaban que algo se podía mejorar, lo comentaban en voz alta mientras probaban la aplicación.

Por ello, vamos a comentar los puntos más importantes durante el testeo de estos distintos módulos, sin entrar en detalle, ya que las soluciones o reconsideraciones se

tuvieron en cuenta en las distintas iteraciones de nuestra metodología en cascada y han quedado reflejados en la parte de implementación.

- **Jugador**: El módulo del jugador funcionaba correctamente desde un principio en la mayoría de sus aspectos. Los únicos aspectos que se corrigieron fueron la velocidad del avatar del jugador, y en el caso del avatar RV un arreglo en el modelo de las manos, ya que se duplicaban
- **Multijugador**: El multijugador en general se implementó de forma correcta. Los aspectos específicos que afectan a otros apartados los explicamos en los siguientes módulos.
- **Museo general**: La interacción de distintos usuarios con el museo funcionaba correctamente, sin embargo, cuando estos interactuaban repetidamente sobre un televisor o modelo de consola, manteniendo pulsado el botón de interacción, provocaba que este se encendiese y apagase continuamente. Simplemente, hicimos que este encendido-apagado no se tuviera en cuenta si se mantenía pulsado el botón.
- **Videojuego Pong**: Este caso fue el primero que provocó un gran cambio a la hora de programar la parte multijugador, ya que los usuarios reportaban que cuando jugaban al juego veían con cierto delay el movimiento de la bola, lo cual les dificultaban el jugar. Este problema se arregló y el delay que había desapareció casi por completo.
- **Videojuego Bolos**: Los usuarios se quejaron de que la dificultad entre las versiones de teclado y ratón y las de RV era bastante dispareja.
- **Videojuego Plataformas**: En el videojuego de plataformas los usuarios se quejaron de varios aspectos, entre ellos las colisiones con paredes y suelos, los cuales tenían comportamientos un tanto extraños, y la sensación del control del personaje. Estos no se han solucionado dado que fue en la fase final del desarrollo, y lo veremos también próximamente en el apartado de conclusiones.

La mayoría de los usuarios han estado contentos con la evolución que ha tenido la aplicación en cada fase del desarrollo, y comentaron que les gustaría volver a utilizar la aplicación una vez estuviera acabada y más pulida.

3.6. Problemas después del desarrollo

Por último, comentar algunos de los fallos más notables que no se han podido solucionar durante el desarrollo, pero que tampoco impiden la experiencia completa del museo. En primer lugar, el mapa del videojuego de plataformas tiene las colisiones un poco irregulares, por lo que existe la probabilidad, de alrededor de un 5 % de partidas, que la colisión entre el personaje y el suelo falle en un salto y este atravesé el suelo. Como solución provisional a este problema, se ha extendido el trigger de los precipicios debajo del suelo, para que el jugador regrese a una posición segura en el caso de que suceda este bug. En segundo lugar, la versión en RV del videojuego de los bolos tiene

una peculiaridad, y es que debido a la dificultad de trabajar con ángulos de Euler y Quaternions, a veces los ángulos con los que se tira la bola respecto a la mano del jugador, no se registra correctamente y lanza la bola hacia la dirección que no toca.

Estos problemas se podrían arreglar con un cambio de perspectiva o usando un grupo de elementos distintos, pero como no suponen un problema grave en la experiencia del usuario quedarían fuera del alcance del proyecto.

CAPÍTULO 4

VERIFICACIÓN

En este capítulo haremos un estudio de los usuarios que han probado nuestra aplicación al final de la fase de desarrollo. Analizaremos su experiencia y obtendremos los resultados para verificar si se han cumplido las expectativas del programa y el sistema, además de realizar una hipótesis de investigación para averiguar si hay alguna relación significativa en realizar las tareas en un sistema u otro de los cuales hemos hecho el estudio. Posteriormente, en el capítulo de conclusiones, hablaremos de los comentarios de los usuarios durante las pruebas.

4.1. Realización de las pruebas de usuario

4.1.1. Contexto de uso

Para definir el contexto de uso tendremos que definir quienes son los usuarios que han realizado la prueba, así como que tareas tienen que realizar y cuál es su entorno.

Nuestros usuarios no cumplen un perfil concreto, varían entre los 21 y 54 años, con estudios y trabajos distintos. El detalle que si queremos destacar es que, de los once usuarios, cinco de ellos tomaron la prueba de forma telemática.

Los usuarios inician la aplicación sentados en el caso de utilizar input de teclado y ratón, y de pie con el input de RV. Durante el transcurso de la prueba, los usuarios pueden comentar sus ocurrencias en voz alta, de las cuales hablaremos más adelante. El encargado les irá anunciando cada una de las tareas, y cronometra estas tareas desde que termina de enunciarlas hasta que las considera cumplidas. Estas tareas son las mismas tanto para el input de teclado y ratón como en RV. Las pruebas son las siguientes:

1. Mira los controles y pulsa el botón de *Play*. Contexto: El usuario se encuentra al inicio de la aplicación

2. Coge el mando de la consola a la que se puede jugar en la sala de *Sega*.

3. Juega al videojuego de la consola hasta que consigas la pantalla de *Has Ganado*. Limitación: Si pasados 99 segundos no lo ha conseguido, se da por completada la tarea

4. Enciende el televisor de la consola *Nintendo DS*.

5. Mira la información de la consola *XBOX 360*.

El entorno será tanto un PC donde se usará el teclado y ratón, como unas gafas de RV que estarán conectadas a un PC para poder ejecutar el programa. Cualquier modelo de dispositivos de RV podía realizar esta prueba, siempre y cuando contara con soporte para OpenXR y tuviera dos mandos con joystick y botones.

4.1.2. Documento de las pruebas de usuario

El documento que se ha utilizado consta de dos páginas. En estas dos páginas encontramos dos apartados distintos: Un apartado donde se anota el tiempo que ha tardado el usuario en realizar cada una de las tareas, así como con cuál de el input ha empezado, y otro apartado donde el usuario anota sus datos (edad y género), dos preguntas sobre su experiencia con los videojuegos y la RV y por último, veinte preguntas que constituyen el System Usability Scale (SUS) adaptado a la RV [48]. Estas veinte preguntas son una mezcla de preguntas que encontraríamos en el test SUS convencional y preguntas orientadas a los sistemas de Realidad Virtual que planteamos nosotros mismos y de las cuales nos interesaba investigar, como el nivel de inmersión, la nostalgia, si producía mareos, etc.

4.2. Análisis de los datos obtenidos

4.2.1. System Usability Scale adaptado a la RV

En este apartado vamos a estudiar, según los datos recopilados de las experiencias de los usuarios, el nivel de usabilidad del sistema, en nuestro caso, del uso de las gafas de RV.

Primeramente, mostraremos como se han repartido las puntuaciones entre todas las preguntas que se han realizado en el SUS. En la figura 4.1 podemos observar dicha distribución. Como podemos ver, en las preguntas con número impar predominan los valores del tres al cinco, dando a entender que los usuarios están de acuerdo con las afirmaciones de estas preguntas de carácter positivo. Por otro lado, las preguntas pares, con carácter negativo, predominan valores del uno al tres, dando a entender que los usuarios no están de acuerdo con las afirmaciones de las preguntas.

A simple vista, podríamos decir que el test SUS da como resultado que el sistema es usable, según las respuestas de los usuarios. Para cerciorarnos, vamos a explicar el sistema de puntuación del SUS y como lo hemos aplicado a nuestras veinte preguntas.

En el formulario clásico de SUS se contemplan diez preguntas. Estas diez preguntas pueden responderse con una puntuación de un intervalo entre el uno y el cinco, siendo uno donde el usuario está en desacuerdo con la afirmación de la pregunta y cinco donde está de acuerdo.

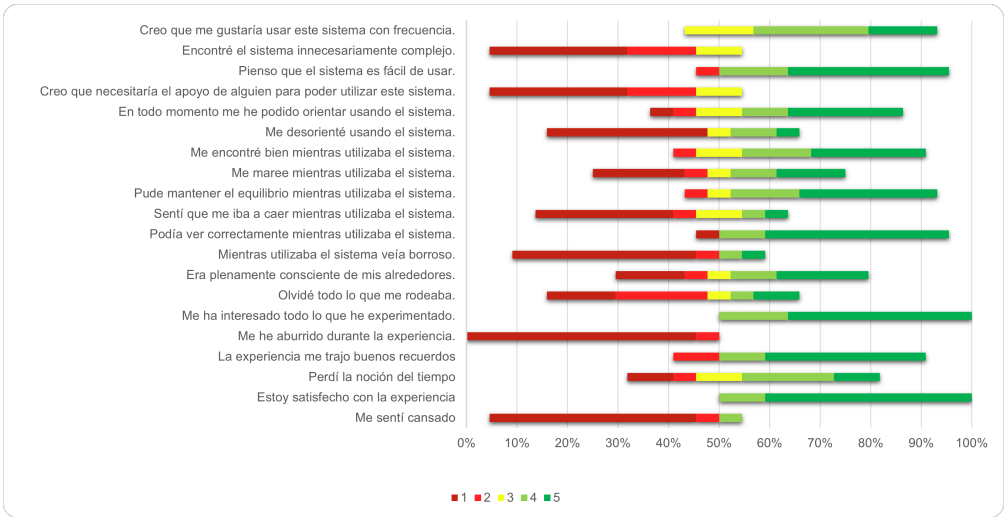

Figura 4.1: Distribución de las respuestas SUS

Con este sistema, la máxima puntuación que se puede conseguir en el test es de 40, dado que en las preguntas positivas hay que restar uno a la puntuación, dando lugar a que si la puntuación es de cinco, el resultado sea cuatro, mientras que en las negativas se le resta la puntuación a cinco, por lo que si la puntuación de la pregunta es uno, el resultado es cuatro. Así entonces, para conseguir la puntuación en un intervalo del cero al cien, el resultado del test es la suma de las puntuaciones de las preguntas positivas y negativas, multiplicando por 2.5 [48].

En el formulario que nosotros hemos utilizado no se plantean diez preguntas, sino que se plantean veinte. Así mismo, la proporción de preguntas positivas y negativas, intercaladas en pares e impares, es la misma. Por lo tanto, si hiciéramos el proceso de sumatorio de la puntuación de las preguntas, el resultado sería una serie de puntuaciones por cada uno de los usuarios, tal como se puede ver en la tabla 4.1. Al hacer la media de estas puntuaciones con los once usuarios, la puntuación resultante seria de 154.55. Debido a que nuestro modelo SUS no tiene un punto en concreto para demostrar que el sistema es usable, no podemos afirmar nada de este, por lo que adaptaremos ligeramente el modelo para obtener una conclusión con el modelo clásico de SUS.

Para adaptar nuestro modelo, dividimos las veinte preguntas en dos grupos de diez, uno tiene las preguntas de la número uno a la diez, y el otro de la once a la veinte. La suma de las puntuaciones se realiza igual que hemos explicado anteriormente, y la media de las puntuaciones de cada grupo son 75.68 para el primer grupo de preguntas y 78.86 para el segundo grupo. Solo de ver estas puntuaciones es claro que mediante esta simplificación vemos que el sistema es usable. Si hiciéramos la media de estas dos puntuaciones de los dos grupos, el resultado sería 77.27, el cual también cumple con el mínimo de 68 que indica el modelo clásico de SUS para afirmar que el sistema es usable.

ID	Puntuación total
1	170
2	182.5
3	152.5
4	157.5
5	185
6	195
7	162.5
8	125
9	117.5
10	142.5
11	110

Tabla 4.1: Resultado puntuaciones SUS adaptado a la RV

4.2.2. Evaluación de las tareas sobre los modelos

A continuación explicaremos como se ha evaluado la prueba realizada por los usuarios. Primero realizaremos una hipótesis de investigación, que variables y datos se han usado y la interpretación de los resultados que nos dará el método ANOVA [49].

Nuestra hipótesis de investigación consiste en ver si existe una diferencia entre el tiempo que ha tardado el usuario en hacer las tareas con los controles de teclado y ratón con los de RV. Así pues, nuestra hipótesis nula y alternativa seria:

- **H0**: No hay diferencia en los segundos que tardan los usuarios en realizar las tareas entre el uso de los controles de teclado y ratón y los de RV.
- **H1**: Sí hay diferencia en los segundos que tardan los usuarios en realizar las tareas entre el uso de los controles de teclado y ratón y los de RV.

La variable dependiente que principalmente usaremos es la suma de los segundos que el usuario ha tardado en realizar las cinco tareas. Por otro lado, la variable independiente sería el sistema en el que el usuario ha hecho las tareas.

Además, para organizar los datos asignaremos un ID a cada uno de los usuarios, así como tendremos en cuenta si tienen experiencia con RV o no. Con estas consideraciones, haremos la hipótesis sobre el grupo de once usuarios que han tomado la prueba.

Podemos ver en la tabla 4.2 una representación de los datos de los usuarios.

Primero de todo, antes de poder realizar el test ANOVA, vamos a explicar por qué hemos decidido utilizar este test y que pruebas hemos realizado para demostrar la normalidad de los datos.

Para ver la normalidad de los datos empleamos el *Shaphiro-Wilk normality test* [50], dado que se trata de una muestra menor a cincuenta. Hemos elegido el test ANOVA dado que necesitábamos un test paramétrico por nuestro tipo de datos. Tanto para poder aplicar este test como el ANOVA, utilizaremos la herramienta de R Studio [51].

- **Grupo completo de usuarios**: Vamos a analizar el grupo completo de usuarios. En el caso de que viéramos alguna irregularidad con el grupo grande, podríamos subdividirlo en dos grupos según su experiencia con la RV

ID	¿Está acostumbrado a la RV?	¿Con qué sistema empezó?	Tiempo total RV (s)	Tiempo total PC (s)
1	Si	VR	106	73
2	Si	VR	97	55
3	No	VR	180	127
4	No	VR	170	61
5	Si	VR	99	81
6	Si	PC	136	71
7	No	PC	235	187
8	No	VR	140	126
9	No	PC	97	92
10	Si	PC	82	79
11	No	PC	203	239

Tabla 4.2: Datos de las tareas y usuarios

Aplicando el *Shaphiro-Wilk normality test* a los datos, vemos que tenemos unos datos no normales. En este caso se trata de las instrucciones en PC, cuyo p-value nos da 0.018, inferior a los 0.05 del nivel de significancia que tomamos. Para resolver este conflicto, veremos si normalizando los datos cambia el resultado. Aplicando una función de logaritmo en base diez a los datos sobre el tiempo, y volver a realizar el test, esta vez obtenemos un p-valor de 0.442 para las instrucciones en RV y 0.3074 para las instrucciones PC. Así pues, no podemos rechazar la hipótesis nula, por lo que asumiremos la normalidad de los datos.

Seguidamente, para realizar el test ANOVA, agrupamos los datos por filas, de tal manera que en una columna nos informa del modelo, el cual se refiere a si las tareas se han realizado en PC o RV, y otra columna con el tiempo que han durado las tareas. Una vez hecho esto, aplicamos el test ANOVA, pasando por parámetro el tiempo como variable dependiente, el modelo del sistema como variable independiente y la tabla con los datos. Aplicaremos el test ANOVA, el cual nos da como resultado un p-valor de 0.0188. Al ser el valor menor a 0.05, hemos de rechazar H0, y asumiremos la hipótesis alternativa H1.

Pese a que finalmente hemos conseguido analizar los datos, estos han presentado cierta irregularidad, por lo que trataremos de ver con los grupos separados que es lo que ha podido pasar.

- **Usuarios con experiencia en RV**: Comenzaremos con este grupo analizando la normalidad que presentan los datos. Una vez aplicado el *Shaphiro-Wilk normality test*, el p-valor que nos muestra da como resultado es de 0.4888 para los datos de las instrucciones en RV y 0.2964 para las instrucciones en PC. Al ser estos valores mayores a 0.05, no podemos rechazar H0, así que asumiremos que los datos son normales.

  Realizando el test ANOVA, nos da como resultado 0.035. Al ser el valor menor a 0.05, hemos de rechazar H0, así que asumiremos la hipótesis alternativa H1.

- **Usuarios sin experiencia en RV**: Siguiendo en la misma línea que el anterior

grupo, aplicaremos el *Shaphiro-Wilk normality test*. El resultado que nos da el test es de unos p-valor de 0.987 para las instrucciones en RV y 0.7692 para las instrucciones en PC. Siendo los valores superiores a 0.05, no podemos rechazar H0, así que asumiremos la normalidad de los datos.

De la misma manera, aplicando el test ANOVA al igual que antes, obtenemos un p-valor de 0.173. En este caso, tendremos que aceptar la hipótesis nula para este grupo.

Como conclusión, aun viendo que el grupo de usuarios no acostumbrados a la RV ha dado algunas disparidades, podemos decir que en el conjunto global, aceptamos la hipótesis alternativa. Indicamos así, que sí hay diferencia en los segundos que tardan los usuarios en realizar las tareas entre el uso de los controles de teclado y ratón y los de RV.

Esta conclusión podría dar lugar a confusión, ya que se esperaría que el grupo de las personas que tenían poca experiencia con la RV, tuviera una mayor diferencia entre los tiempos que tardan en completar las tareas en los diferentes dispositivos. Esta irregularidad puede deberse a la muestra de población que tomamos. Estos usuarios no estaban acostumbrados a la RV, pero tampoco sabían utilizar del todo los controles de teclado y ratón, provocando que los tiempos fueran más semejantes que los usuarios que sí estaban acostumbrados a la RV y además solían utilizar el PC para jugar.

CAPÍTULO 5

CONCLUSIONES

En este último apartado comentaremos cuáles han sido las conclusiones del proyecto: La revisión de la estimación temporal, resultados de los test SUS y ANOVA, resultados generales del museo y ampliaciones de este proyecto.

5.1. Revisión de la estimación temporal

Como pudimos ver en el apartado 3.1.1, el proyecto se planificó en 4 fases bien diferenciadas entre ellas.

El desarrollo del proyecto comenzó a finales de septiembre, y la primera fase consistía en empezar a reunir información para posteriormente redactar la introducción, el estado del arte y la gestión del proyecto. Esta fase se estimó correctamente, ya que se pasó a la siguiente fase antes de acabar octubre.

En la segunda fase se empezó a desarrollar el juego, con la estructura general del museo, los avatares de PC y RV, la implementación del multijugador y hasta finales de noviembre, con el desarrollo del videojuego del Pong. Al ser la primera fase del desarrollo de la aplicación y tener que construir las bases del museo desde cero, se alargó el desarrollo un poco más de lo esperando, finalizando a mediados de diciembre.

Al comienzo de la tercera fase se hizo la primera prueba con dos usuarios jugando a un mismo juego. Después de unos cambios menores, se dio por finalizado el juego del Pong. En esta fase se tenían que desarrollar el resto de juegos, estos eran el videojuego de bolos y el de plataformas. Esta tercera fase en un principio se tenía que finalizar a finales de enero, pero se retrasó el desarrollo de tal forma que únicamente el juego de bolos estaba terminado para esa fecha. Tras el desarrollo del juego de bolos, el juego de plataformas se alargó hasta principios de marzo, por lo que la tercera fase finalizó en ese momento. La principal causa que llevó al retraso de esta fase fue el modo multijugador del juego de bolos, así como el control de juego en RV.

Finalmente, en la última fase se realizarían las pruebas con usuarios, cuyos resultados se ven reflejados en todo el capítulo 4. Al tener que contactar con un mínimo

de diez personas para poder realizar correctamente las pruebas y el inconveniente de que durante dos semanas no se pudieron realizar las pruebas debido al Covid, esto retrasó el desarrollo y posteriores resultados de las pruebas con usuarios. Una vez efectuadas las pruebas, el análisis de los resultados finalizó a principios de mayo, por lo que únicamente quedaban la documentación de las conclusiones, resumen y corrección del documento antes de la entrega. Esta fase terminó finalmente a principios de junio, dando también por finalizado el proyecto.

5.2. Resultados test SUS y ANOVA

Como pudimos ver en la sección 4.1.2, ambos análisis del test SUS adaptado a la RV y el test ANOVA dieron un resultado positivo.

Por un lado, el test SUS que adaptamos para la RV, al no tener un formato totalmente estándar, tuvimos que interpretar los resultados de distinta forma. Si volvemos a echar un vistazo a la figura 4.1, podemos ver que la distribución de las respuestas indica una mayor puntuación en las preguntas impares, de carácter positivo, así como una mejor puntuación en las preguntas pares, de carácter negativo, al estar en desacuerdo con estas. Por otro lado, una vez procesado el cálculo de las puntuaciones, la media de estas seria de 154'55 para nuestro modelo adaptado de SUS. Este dato no es del todo útil, ya que por nuestra adaptación, no tenemos una forma de medir lo que se acepta en usabilidad. Por otro lado, si dividimos las respuestas en dos mitades, de diez y diez preguntas, siguiendo la estructura de intercalar una positiva y una negativa, obtenemos que las puntuaciones serían 75'68 para el primero grupo de preguntas y 78'86 para el segundo grupo de preguntas, dando una media de 77'27. Según el percentil cincuenta que nos proporciona el modelo estándar SUS [48], podemos concluir que el modelo del sistema de RV es usable.

Por otro lado, si observamos el test ANOVA que aplicamos a los distintos grupos de usuarios, finalmente rechazamos nuestra hipótesis nula, que consistía en que el tiempo necesario para realizar las tareas en RV era la misma que en PC. Asegurando la normalidad de los datos, y aplicando el test ANOVA con el tiempo de las tareas y los dos modelos de sistema, el resultado ha sido que dado los p-valores que nos proporcionó el test, no podíamos aceptar la hipótesis nula, aceptando la hipótesis alternativa y, por lo tanto, diciendo que el tiempo transcurrido en las tareas realizadas en RV y PC son distintos.

5.3. Resultados del museo

En este apartado veremos, no solo todo lo que se ha terminado implementando en el museo, sino también si todo aquello que se especificó en el apartado de objetivos y alcance del proyecto 1.4 se ha cumplido.

El resultado del proyecto consiste en una aplicación multidispositivo, que funciona para Windows 7,8,10 y 11 y distintos dispositivos RV que puedan conectarse al PC y cuenten con soporte de OpenXR [44]. Esta aplicación permite la exploración de un museo de videojuegos, además cuenta con dos modalidades: un único jugador o más de un jugador. Los usuarios pueden interactuar con varias consolas: mirando información

sobre la consola, viendo un vídeo demo de un videojuego de la consola, e incluso jugar a un videojuego que se asemeje a los de esta.

Los dispositivos que se han confirmado que pueden realizar la mayoría de las funcionalidades del museo han sido: Oculus Rift S, Oculus Quest 1 y 2, HTC Vive, Valve Index y Acer ah101.

Nuestro museo se compone de cinco habitaciones y un lobby. En cada una de las habitaciones, se encuentran las cinco empresas de videojuegos que elegimos. La habitación de Nintendo, la más grande, cuenta con un total de once consolas, una de ellas, la Wii, siendo una de las consolas en las que se puede jugar un videojuego. La siguiente habitación con más consolas es la de Playstation, con siete consolas, seguida de la de Atari con seis, una de ellas con videojuego. Por último, tenemos la habitación de XBOX y Sega con cinco consolas, teniendo Sega otras de las consolas con videojuegos.

Como vimos en nuestro alcance del proyecto, la idea era implementar de dos a seis videojuegos, además de que al menos dos de ellos tenían que ser multijugador. Finalmente, se han implementado tres videojuegos: Un juego de plataformas, uno de bolos y un Pong, siendo estos dos últimos multijugador. Por lo tanto, cumplimos con esta parte del alcance del proyecto.

Además de la implementación base de los videojuegos, hemos implementado un sistema especial para uno de ellos, el videojuego de bolos. Este consiste en un sistema de espera, el cual explicamos en el sub-apartado 3.4.6, que permite que si un jugador deja de jugar, la partida se suspende hasta que el mismo jugador u otro vuelve a coger el mando, entonces se reanuda la partida.

Respecto al apartado multijugador, se ha comprobado que diez jugadores han podido estar en una misma sala del servidor. Estos han podido interactuar con los elementos interaccionables y los videojuegos sin que ocurra ningún fallo. No se ha realizado ningún estudio formal, pero podemos decir que estamos satisfechos con los resultados del modo multijugador.

5.4. Desarrollo personal

Con la finalización de este proyecto, siendo este el TFG que marca el final de una larga carrera, me gustaría recalcar algunas conclusiones personales, de todo aquello que se ha aprendido, y de la satisfacción con los resultados.

Si bien es cierto que se contaba con cierta experiencia en el uso del motor de videojuegos Unity, gracias a este proyecto se ha alcanzado un nivel superior de entendimiento y uso de tecnologías relativamente recientes. El aprendizaje de las tecnologías OpenXR y como funcionan algunos servicios de Networking ha sido fructífero tanto para poder implementar las funciones que necesitábamos para el museo como para poder utilizarlas en futuros proyectos personales o laborales. Además, la implementación de los videojuegos desde cero ha servido para entender como funcionan los estados por los que pasan y como diseñar e implementar distintos tipos de juegos, aunque no se haya podido profundizar mucho en ellos.

Al haber realizado pruebas con distintos usuarios, se ha podido comprobar que cosas que se daban por hecho que los usuarios entenderían a la primera o no les costaría deducir, ha dado como resultado que no es así, dando un cambio de perspectiva a futuro sobre como presentar las funcionalidades para que se entiendan mejor.

El resultado general del museo me ha dejado muy satisfecho. Mientras distintas personas probaban la aplicación, se cumplían mis expectativas y objetivos. La gran mayoría hablaba de lo a gusto que se habían sentido viviendo esa experiencia, ya fuera por la nostalgia o por el descubrimiento de consolas y videojuegos que no conocían o que les parecían interesantes. También se consiguió que los videojuegos desarrollados gustasen a la gente que los jugaba, sobre todo los que eran de dos jugadores, que incitaban a jugar varias partidas para ver quien ganaba más.

En general y para concluir, me quedo muy satisfecho con los resultados del proyecto. Siento que he tenido un gran progreso en el desarrollo de aplicaciones, más concretamente en videojuegos, y que esta experiencia me servirá en un futuro.

5.5. Posibles ampliaciones

Para cerrar el desarrollo y documentación de este proyecto, vamos a ver que posibilidades tendría de ampliarse en futuros desarrollos, lo que nosotros hemos implementado e investigado.

La mayoría de estas ideas han sido obtenidas a través de la prueba con usuarios, tanto en la fase de desarrollo como en las pruebas finales.

Algo que pudimos comprobar al realizar las pruebas con distintos dispositivos de RV, es que aquellos cuyos mandos no tenían botones aparte del trigger y el grip, no podían jugar con los videojuegos debido al input por defecto. Una solución sería adaptar el input para que se usaran únicamente los joysticks, el trigger y el grip. Otro problema parecido es sobre aquellos dispositivos que ni siquiera cuentan con un mando, y que utilizan gestos, muy común en dispositivos de realidad mixta. Para ello, podríamos usar el hand-tracking para saber hacia donde está apuntando, por ejemplo, por el dedo índice, y con un gesto conseguir interactuar con un objeto.

Otra función relacionada con el input es que los controles principales que se utilizan en el museo, como pueden ser los controles para caminar, interactuar y soltar un mando, no se vuelven a mostrar en otro lugar aparte del menú principal. Una forma de solucionar este defecto, sería poder acceder a un menú durante la estadía en el museo, donde se pudieran ver los controles según el dispositivo. Además de los controles en el menú, también se podrían añadir otras funcionalidades como el control del volumen del sonido, salir de la sala sin cerrar la aplicación, etc.

Uno de los comentarios que más han repetido los usuarios ha sido la confusión a la hora de buscar una consola sin saber de cuál se trataba hasta el momento de interactuar con ella y ver su información. Por ello, un cambio en el museo que arreglaría esta confusión podría ser añadir unos carteles con los logos de cada consola para poder identificarlas incluso desde lejos. Podemos ver en la figura 5.1 un ejemplo de como se podrían poner los carteles.

Actualmente, las únicas funcionalidades relacionadas con el sonido que están implementadas en el museo son los sonidos de los vídeos de cada una de las consolas, y los sonidos del videojuego Pong. Una ampliación para el museo sería añadir sonido a los videojuegos de bolos y plataformas.

Para finalizar este apartado, el último rasgo que pensamos que se podría ampliar, que a su vez es la base de todo el museo, es ampliar el catálogo de consolas, empresas y videojuegos que se pueden ver e interaccionar, así como los videojuegos que se

Figura 5.1: Ejemplo de carteles para identificar consolas

pueden jugar. Aparte de las consolas de otras empresas distintas a las cinco que hemos seleccionado, también se podrían exponer modelos alternativos de las consolas que ya tenemos, como la *PSP Go* o la *Nindento DSi*.

En futuras ampliaciones del proyecto podría volver a considerarse la emulación de los videojuegos en vez de implementarlos desde cero. Aun así, se desaconseja esta opción por los inconvenientes que presenta, tal como explicamos en el sub-apartado 2.3.

Este apartado concluye la documentación del proyecto y deja abierta la puerta para que este se amplíe en el futuro. A continuación se incluye un apéndice que contiene todas las atribuciones de los assets del proyecto.

APÉNDICE A

ASSETS EN EL PROYECTO

En el apéndice trataremos el origen de los bienes digitales utilizados durante el desarrollo del TFG y la validez de estos. En el primer apartado daremos una visión general de la legalidad del uso de estos bienes, mientras que en los demás tendremos constancia de su procedencia.

A.1. Legalidad del uso de bienes

En esta sección del apéndice explicaremos de forma general que puntos legales se han tenido en cuenta para el uso de los bienes digitales utilizados en el desarrollo de este TFG.

El primer punto sería asegurarse que la licencia Creative Commons (CC) permita la explotación de la obra del autor. La mayor parte de los bienes utilizados rinden bajo una licencia CC de ”Atribution” y ”Non comercial”. Dichas condiciones se cumplen en el desarrollo, ya que el TFG tiene un fin no comercial y los distintos bienes utilizados vienen con su respectiva atribución en los siguientes sub-apartados de este apéndice.

El principal problema de estos bienes es que algunos de ellos no vienen definidos por una licencia CC o no está clara. Para los bienes cuya licencia de uso no conocemos nos hemos acogido a la regla comúnmente denominada ”Fair use”, la cual indica que cualquier bien digital puede ser utilizado a pesar del tipo de licencia que tenga, siempre y cuando sea con fines educativos y no comerciales [52]. Este término anglosajón surgió de la legislación de Estados Unidos y aunque el estado español no tiene una regla directamente equivalente, si hay algo similar.

La actual legislación vigente recoge ciertos límites respecto a los derechos de autor que, bajo ciertas circunstancias, estos tienen unas excepciones de uso de la propiedad intelectual. Según recoge el Centro Español de Derechos Reprográficos, una de las excepciones de uso a la que nos podríamos acoger es la de ilustración con fines educativos o de investigación científica [53]. En este TFG se cumplen las condiciones adscritas a esta excepción, siendo estas: que tal acto se haga únicamente con una finalidad de

investigación, y en la medida justificada por la finalidad no comercial perseguida, que se trate de una obra ya divulgada, que las obras no tengan la condición de libro de texto, manual universitario o publicación asimilada y que se incluyan el nombre del autor y la fuente (lo cual hacemos en los siguientes sub-apartados).

A.2. Consolas

En esta sección del apéndice podremos ver la procedencia de los bienes digitales referentes a las consolas de videojuegos utilizados en el proyecto de Unity desarrollado en este TFG.

A.2.1. Sony

- **PS1**:
  - **Modelo**: https://sketchfab.com/3d-models/playstation-1-96bed8283f2940d7bb4b1866ab0637a2
  - **Video**: https://www.youtube.com/watch?v=VjEchWx-ZaA
- **PS2**:
  - **Modelo**: https://sketchfab.com/3d-models/playstation-2-8c992e9f5c86496ca7bd7dfe109d3cc2
  - **Video**: https://www.youtube.com/watch?v=x64859qvAHw
- **PSP**:
  - **Modelo**: https://sketchfab.com/3d-models/sony-psp-dca89d10ec304d0cab76837750df7761
  - **Video**: https://www.youtube.com/watch?v=_GIrIqhqUaE
- **PS3**:
  - **Modelo**: https://sketchfab.com/3d-models/cabinet-with-tv-dvd-player-vhs-ps3-4a9f29cfdfa64592bfeafaec59c7bf39
  - **Video**: https://www.youtube.com/watch?v=_Mn-vjJ6n0M
- **PS Vita**:
  - **Modelo**: https://3dwarehouse.sketchup.com/model/7d43d670-07a7-42fb-b55c-516b5404487f/PS-vita
  - **Video**: https://www.youtube.com/watch?v=Xkp05L5cPJY
- **PS4**:
  - **Modelo**: https://sketchfab.com/3d-models/ps4-fat-db08563414cf4bfdbdd1f40074b869b3
  - **Video**: https://www.youtube.com/watch?v=tKf_J4be6dw

- **PS5**:
  - **Modelo**: https://sketchfab.com/3d-models/ps5-d788de3735964151a3e24fd59c0f1956
  - **Video**: https://www.youtube.com/watch?v=PoZQ5SybSPs

A.2.2. XBOX

- **XBOX**:
  - **Modelo**: https://sketchfab.com/3d-models/xbox-first-generation-c586f9956989480fb14f11447fb1d42c
  - **Video**: https://www.youtube.com/watch?v=mtRmzN_W7W8
- **XBOX 360**:
  - **Modelo**: https://sketchfab.com/3d-models/xbox-360-black-3d-model-by-rob-bryant-jr-f65ac4721fd34dd8a8b63adfa712f5b8
  - **Video**: https://www.youtube.com/watch?v=fWJSzIZIowI
- **XBOX ONE**:
  - **Modelo**: https://sketchfab.com/3d-models/xbox-one-bfe7d60a29a24dc18c5bc17bd4779cc7
  - **Video**: https://www.youtube.com/watch?v=eF9-JTkxIxo
- **XBOX ONE X**:
  - **Modelo**:https://sketchfab.com/3d-models/xbox-one-xs-294f13ab554f4d4d832b271045bda4fa
  - **Video**: https://www.youtube.com/watch?v=K_0u0098RaI
- **XBOX Series S**:
  - **Modelo**: https://sketchfab.com/3d-models/xbox-series-s-c382c3f9f9ea4dc0afcd0ed177bde3f7
  - **Video**: https://www.youtube.com/watch?v=6sc9tFOpnUc

A.2.3. Nintendo

- **NES**:
  - **Modelo**: https://sketchfab.com/3d-models/nintendo-nes-original-6de5054340184509af6f301c7f3fac57
  - **Video**: https://www.youtube.com/watch?v=K5yqXTh1IUw
- **Game Boy**:
  - **Modelo**: https://sketchfab.com/3d-models/game-boy-classic-0ae80019e6f046168923286d7e628f6f

    - **Video**: https://www.youtube.com/watch?v=G1bu5fTMKEs

- **SNES**:

    - **Modelo**: https://sketchfab.com/3d-models/snes-console-cb62a1ddd75841e0a6b143e53c9240bd
    - **Video**: https://www.youtube.com/watch?v=bIWvEFPdEmM

- **Nintendo 64**:

    - **Modelo**: https://3dwarehouse.sketchup.com/model/u1749c274-7f89-4a49-b984-b8cedb45c344/Nintendo-64-with-Super-Mario-64-in-it
    - **Video**: https://www.youtube.com/watch?v=h6tFoIUyOdU

- **Game boy advance**:

    - **Modelo**: https://3dwarehouse.sketchup.com/model/5c187001-891f-4118-ae81-0febff1197cd/Game-Boy-Advance
    - **Video**: https://www.youtube.com/watch?v=nPzBeIATN74

- **Nintendo DS**:

    - **Modelo**: https://sketchfab.com/3d-models/nintendo-ds-lite-ca529eb7208746e89b7a28fd2246659d
    - **Video**: https://www.youtube.com/watch?v=ppMCqcf4Y80

- **Nintendo 3DS**:

    - **Modelo**: https://sketchfab.com/3d-models/new-nintendo-3ds-1e820d14445d45edbda0434dff8fe037
    - **Video**: https://www.youtube.com/watch?v=zRirNjXI760

- **Game Cube**:

    - **Modelo**: https://sketchfab.com/3d-models/game-cube-free-download-424ac8af08394ac7bc03d843f331feb7
    - **Video**: https://www.youtube.com/watch?v=_oCRjJYPXoE

- **WII**:

    - **Modelo**: https://sketchfab.com/3d-models/wii-7a96a38cf0684aecb42a58d1e3b65cb9
    - **Mando**: https://sketchfab.com/3d-models/wii-controller-test-346927e0ef4f47baa0b2d69117a21c1c

- **WII U**:

    - **Modelo**: https://sketchfab.com/3d-models/wii-u-a2e2e62fd88e43b1a5232daf34aed850

  - **Video**: https://www.youtube.com/watch?v=N8IkLfYDsaY

- **Switch**:

  - **Modelo**: https://sketchfab.com/3d-models/nintendo-switch-b30e0a74899b4f9baf030d02f45ab599
  - **Video**: https://www.youtube.com/watch?v=CE8PS7r3IGM

A.2.4. Atari

- **Atari 400**:

  - **Modelo**: https://sketchfab.com/3d-models/atari-400-0a34fed51f4241ffa867ee65de721d5f
  - **Video**: https://www.youtube.com/watch?v=vNuSMDeAO4s

- **Atari Pong**:

  - **Modelo**: https://sketchfab.com/3d-models/pong-arcade-cabin-1f0f0d21ea5e4f8dbc415acde9997696
  - **Mando**: https://sketchfab.com/3d-models/atari-2600-joystick-cd1f700e4895400d8e8ad7ee46fb28bf

- **Atari 2600**:

  - **Modelo**: https://sketchfab.com/3d-models/atari-2600-a72bfe5653f2458fb8be1c6165ed0d8a
  - **Video**: https://www.youtube.com/watch?v=QmrQkQsM9FU

- **Atari Lynx**:

  - **Modelo**: https://3dwarehouse.sketchup.com/model/ff3cd7cc-9e24-479a-9876-28d6dc1b9ee6/Atari-LYNX-Mk1
  - **Video**: https://www.youtube.com/watch?v=wVcDpuPYXNU

- **Atari Jaguar**:

  - **Modelo**: https://3dwarehouse.sketchup.com/model/6bd1a21172f19b0a3ca8b71b28ce88bb/Atari-Jaguar
  - **Video**: https://www.youtube.com/watch?v=o4u8dxpm3-4

- **Atari VCS**:

  - **Modelo**: https://www.turbosquid.com/es/3d-models/atariish-3d-1760002
  - **Video**: https://www.youtube.com/watch?v=KEsCbwFWyPc

A.2.5. Sega

- **Sega Master System**:
  - **Modelo**: https://sketchfab.com/3d-models/sega-master-system-c2de422f22234b209aaae7a377846995
  - **Video**: https://www.youtube.com/watch?v=gWo-Lm0FQZo
- **Sega Mega Drive**:
  - **Modelo**: https://sketchfab.com/3d-models/lowpoly-sega-genesis-gaming-console-bceba7d09cd147da8e25ff3f5c238b37
  - **Video**: https://www.youtube.com/watch?v=Gsa_4s3CjmI
- **Sega Game Gear**:
  - **Modelo**: https://3dwarehouse.sketchup.com/model/bf8bbe00-546d-4b1e-91a3-059d5067db8d/Sega-Game-Gear
  - **Video**: https://www.youtube.com/watch?v=0OyyxFOVaYA
- **Sega Saturn**:
  - **Modelo**: https://3dwarehouse.sketchup.com/model/86e4194e00424702d261b72fcd57b375/Sega-saturn?hl=es
  - **Mando**: https://sketchfab.com/3d-models/lowpoly-sega-genesis-gaming-console-bceba7d09cd147da8e25ff3f5c238b37
- **Sega Dreamcast**:
  - **Modelo**: https://3dwarehouse.sketchup.com/model/91bcea3a-3c1d-443f-a33c-86e1809958cd/Sega-Dreamcast-mit-Controller
  - **Video**: https://www.youtube.com/watch?v=0ESGFcbc-B8

A.3. Estructura museo

En esta sección del apéndice podremos ver la procedencia de los bienes digitales referentes a los materiales utilizados en la estructura del museo del proyecto de Unity desarrollado en este TFG.

- **Suelo y paredes**: https://assetstore.unity.com/packages/2d/textures-materials/floor-textures-4k-179126
- **Techo**: https://assetstore.unity.com/packages/2d/textures-materials/roofing/stylized-roof-material-150521
- **Mueble**: https://sketchfab.com/3d-models/cabinet-with-tv-dvd-player-vhs-ps3-4a9f29cfdfa64592bfeafaec59c7bf39
- **Televisor pantalla plana**:https://sketchfab.com/3d-models/cabinet-with-tv-dvd-player-vhs-ps3-4a9f29cfdfa64592bfeafaec59c7bf39

- **Televisor de tubo**: https://sketchfab.com/3d-models/lowpoly-sega-genesis-gaming-console-bceba7d09cd147da8e25ff3f5c238b37
- **Globo informativo**: https://svgsilh.com/es/image/156056.html
- **Cartel Sony**: https://1000marcas.net/playstation-logo/
- **Cartel XBOX**: https://logos-download.com/10046-xbox-logo-download.html
- **Cartel Nintendo**: https://logos-download.com/wp-content/uploads/2016/06/Nintendo_logo_red.png
- **Cartel Nintendo Game Cube**: https://1000marcas.net/nintendo-gamecube-logo/
- **Cartel Nintendo DS**: https://es.m.wikipedia.org/wiki/Archivo:Nintendo_DS_Logo.svg
- **Cartel Nintendo 64**: https://flyclipart.com/es/nintendo-64-logo-nintendo-word-symbol-trademark-hd-png-download-1157097
- **Cartel Sega**: https://es.m.wikipedia.org/wiki/Archivo:SEGA_logo.svg
- **Cartel Atari**: https://logos-marcas.com/atari-logo/

A.4. Otros bienes digitales

En esta sección del apéndice podremos ver la procedencia de los bienes digitales que no encajan en ninguna de las anteriores categorías, como pueden ser los modelos y tilemaps de los videojuegos, las imágenes utilizadas en los carteles de controles, etc.

- **Teclas W A S D y raton**: https://www.pngitem.com/middle/iTRxwTx_configuracin-de-teclas-para-el-demo-w-a/
- **Tecla E**: https://www.pngegg.com/en/png-fjihv
- **Tecla Q**: https://www.hiclipart.com/free-transparent-background-png-clipart-qrnlt
- **Oculus controller**: https://unrealrox.readthedocs.io/en/master/_site/project/controllers.html
- **Tecla espacio**: https://docs.google.com/document/d/1r9qPOGg9_UiOLucaDtGRGyh95O7Y0bod6PvQ6gEIUdM/mobilebasic
- **Bola de bolos**: https://sketchfab.com/3d-models/bowling-ball-5d31d395e0e24138900065b642ac8299
- **Pino de bolos**: https://sketchfab.com/3d-models/bowling-pin-7b4f3c1ca2f1451ebe3cf7ea984f69fc
- **Pista de bolos**: https://sketchfab.com/3d-models/bowling-alley-mozilla-hubs-room-644609db6a5a435098b305413b162266

- **Flechas movimiento controles VR bolos**: https://icon-icons.com/es/icono/flechas-grupo-los-cuatro-las-direcciones/57005
- **Flecha suelo bolos**: https://thypix.com/pic/white-arrow-73/
- **Tileset plataformas**: https://o-lobster.itch.io/

APÉNDICE B

MANUALES DE USUARIO Y DESARROLLO

En este segundo apéndice veremos el manual de instalación, tanto del usuario como el de desarrollo para futuros usos de proyecto.

B.1. Manual de usuario

Para poder ejecutar e instalar la aplicación, el usuario debe seguir unos pasos muy sencillos e intuitivos. Una vez descargada la carpeta comprimida y descomprimirla, únicamente hay que entrar en la carpeta y ejecutar *Museo del videojuego en realidad virtual*. Para poder jugar la versión de PC, se ha de tener cualquier otro dispositivo desconectado, refiriéndonos a los de RV. Para jugar la versión de RV se ha de conectar el dispositivo al PC mediante un cable USB, y posteriormente ejecutar la aplicación desde el PC o desde un escritorio virtual.

En el caso de los dispositivos de RV se recomienda instalar en el PC la aplicación de Steam, así como SteamVR dentro de la propia aplicación. Esto ayuda a la compatibilidad de algunos dispositivos con el sistema OpenXR que utiliza nuestro videojuego.

La carpeta comprimida se encuentra en el siguiente enlace:

https://drive.google.com/file/d/1Y8-kIZAvRmo69n4eEvD3XhBqdi9Pe22U/view?usp=sharing

B.2. Manual de desarrollo

En esta sección explicaremos como alguien puede seguir desarrollando la aplicación del *Museo del videojuego en realidad virtual*. Para empezar, deberíamos instalar en Unity Hub, la versión de Unity 2020.3.21f1. Esta es la versión con la que se ha desarrollado la aplicación, pero siempre se puede utilizar una versión posterior, asegurándose que el contenido de librerías funciona también para la versión que se elija. Una vez se ha instalado correctamente la versión de Unity, hay que descargar el proyecto de Unity sin compilar, y desde la pestaña *Projects* de Unity Hub, importar la carpeta.

La carpeta con el proyecto sin compilar se puede encontrar en el siguiente enlace: https://github.com/MiangaESP/UIB-TFG-Museo-del-videojuego-en-realidad-virtual

Una vez está importado, podemos acceder al proyecto y navegar por él usando el editor de Unity. Para confirmar que todos los paquetes de librerías y assets no se han corrompido, convendría reimportarlos todos. Para ello, si seleccionamos la pestaña de assets y le damos a la opción de *Reimport All*, se iniciará el proceso.

Cuando hayamos comprobado que todo está listo y que no hay ningún error por consola, podemos proceder a terminar con la instalación de uno de los componentes más importantes del museo, el multijugador.

Dado que la cuenta de Photon está vinculada al anterior desarrollador, convendría hacerse una cuenta nueva, o hacerla de forma obligatoria si se ha eliminado la cuenta anterior. Para ello, debemos dirigirnos a la página de Photon [37]. Allí, hemos de crear una nueva cuenta. Una vez creada, nos desplazamos a la pestaña *Dashboard* y clicamos en el botón de *Create a new app*. Rellenamos los apartados que nos indiquen. Una vez hagamos esto, en la misma página de *Dashboard* encontraremos un panel con la app que hayamos creado. Justo debajo del nombre de nuestra app, aparece un texto que indica *App id:*, y si copiamos este id, luego lo podemos introducir en el módulo de Photon de nuestro Unity. Para configurar los ajustes de Photon, hemos de buscar el archivo *PhotonServerSettings* desde el buscador del editor. Entonces, en el inspector podremos cambiar varios aspectos de Photon, como es el ID que mencionamos antes, la región del servidor, el protocolo, etc.

Una vez hemos realizado todos estos pasos y está todo instalado y configurado, podremos continuar con el desarrollo de la aplicación.

BIBLIOGRAFÍA

[1] Historia de la realidad virtual. [Online]. Available: https://xperimentacultura.com/historia-de-la-realidad-virtual/ 1.1

[2] Oculus vr. [Online]. Available: https://es.wikipedia.org/wiki/Oculus_VR 1.1

[3] Playstation vr. [Online]. Available: https://www.playstation.com/es-es/ps-vr/ 1.1

[4] Samsung gear with vr controller. [Online]. Available: https://www.samsung.com/global/galaxy/gear-vr/ 1.1

[5] Htc vive. [Online]. Available: https://www.vive.com/eu/ 1.1

[6] Steam vr. [Online]. Available: https://store.steampowered.com/app/250820/SteamVR/?l=spanish 1.1

[7] La realidad virtual en los diferentes sectores económicos. [Online]. Available: https://talentocorporativo.com/la-realidad-virtual-en-los-diferentes-sectores-economicos/ 1.1

[8] El mercado de los videojuegos de realidad virtual. [Online]. Available: https://i-amvr.com/el-mercado-de-los-videojuegos-de-realidad-virtual/ 1.1

[9] estrategiasdeinversion. ¿cuáles son los planes de facebook en realidad virtual y aumentada? [Online]. Available: https://www.estrategiasdeinversion.com/actualidad/noticias/bolsa-eeuu/cuales-son-los-planes-de-facebook-en-realidad-n-480791 1.1

[10] Unity. [Online]. Available: https://unity.com/es 1.1

[11] Unreal engine. [Online]. Available: https://www.unrealengine.com/en-US/unreal 1.1

[12] Lumberyard. [Online]. Available: https://aws.amazon.com/es/lumberyard/ 1.1

[13] A-frame. [Online]. Available: https://aframe.io/ 1.1

[14] React vr. [Online]. Available: https://github.com/facebookarchive/react-360 1.1

[15] The j. paul getty museum. [Online]. Available: https://artsandculture.google.com/partner/the-j-paul-getty-museum?hl=en 2.1.1

[16] Museo guggenheim bilbao. [Online]. Available: https://artsandculture.google.com/partner/guggenheim-bilbao 2.1.1

[17] Musée d'orsay. [Online]. Available: https://artsandculture.google.com/partner/musee-dorsay-paris 2.1.1

[18] Arts & culture google. [Online]. Available: https://artsandculture.google.com/ 2.1.1

[19] Museum of the world. [Online]. Available: https://britishmuseum.withgoogle.com/ 2.1.1

[20] Giraffatitan dinosaur: Back to life in 360 vr. [Online]. Available: https://www.youtube.com/watch?v=p86gh2HEsp0 2.1.1

[21] Rhomaleosaurus sea dragon: Back to life in 360 vr. [Online]. Available: https://www.youtube.com/watch?v=BH1AvqYXwHQ 2.1.1

[22] The vr museum of fine art. [Online]. Available: https://www.viveport.com/3a7fbcc0-5bcd-4ba3-81b7-0662c1247e12 2.1.2

[23] Mac museum vr. [Online]. Available: https://ltim.uib.es/mac/ 2.1.2

[24] Museo arcade vintage. [Online]. Available: https://museoarcadevintage.com/ 2.1.3

[25] R. Grosso. National videogame museum goes virtual in doom 2. [Online]. Available: https://techraptor.net/gaming/news/national-videogame-museum-goes-virtual-in-doom-2 2.1.3

[26] N. Wang. 360 national video game museum vr tour part 1 frisco texas usa. [Online]. Available: https://www.youtube.com/watch?v=nIHv7Rh_y50 2.1.3

[27] Devolverland expo. [Online]. Available: https://devolverdigital.com/games/devolverland-expo 2.1.3

[28] Majorariatto museum. [Online]. Available: https://www.majorariatto.com/es/museum 2.1.3

[29] Kingdom hearts vr experience. [Online]. Available: https://gaminguardian.com/analisis-kingdom-hearts-vr-experience/ 2.1.4

[30] Emuvr. [Online]. Available: https://www.emuvr.net/ 2.1.4

[31] oculus. Diseñado para mayores de 14 años. [Online]. Available: https://www.oculus.com/safety-center/?locale=es_ES 2.2

[32] C. Infante. Playstation vr: la guía de preguntas frecuentes definitiva (actualizado). [Online]. Available: https://blog.es.playstation.com/2016/10/03/playstation-vr-la-gua-de-preguntas-frecuentes-definitiva/ 2.2

[33] nowideas. Videojuegos en tiempo de coronavirus: una oportunidad para invertir en publicidad. [Online]. Available: https://blogs.unsw.edu.au/nowideas/blog/2020/04/videojuegos-coronavirus-oportunidad-invertir-publicidad/ 2.2

[34] Sk.libretro. [Online]. Available: https://github.com/Skurdt/SK.Libretro 2.3

[35] Libretro unity. [Online]. Available: https://github.com/Skurdt/LibretroUnityFE 2.3

[36] Network manager. [Online]. Available: https://docs.unity3d.com/Manual/UNetManager.html 2.3

[37] Photon - pun. [Online]. Available: https://www.photonengine.com/en-US/PUN 2.3, B.2

[38] Mirror networking. [Online]. Available: https://mirror-networking.com/ 2.3

[39] Peer-to-peer. [Online]. Available: https://es.wikipedia.org/wiki/Peer-to-peer 2.3

[40] Trello. [Online]. Available: https://trello.com 3.1.1

[41] Fps game keybind setup. [Online]. Available: https://linustechtips.com/topic/51216-fps-game-keybind-setup/ 3.3.1

[42] Common types of artificial vr locomotion. [Online]. Available: https://developer.oculus.com/resources/artificial-locomotion/ 3.3.1

[43] Visual studio 2019. [Online]. Available: https://visualstudio.microsoft.com/es/ 3.4.1

[44] Openxr plugin. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.2/manual/index.html 3.4.1, 5.3

[45] Pun 2. [Online]. Available: https://www.photonengine.com/ 3.4.1

[46] Sketchfab for unity. [Online]. Available: https://assetstore.unity.com/packages/tools/input-management/sketchfab-for-unity-14302 3.4.1

[47] Xr interaction toolkit. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@0.9/manual/index.html 3.4.1

[48] System usability scale. [Online]. Available: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html 4.1.2, 4.2.1, 5.2

[49] Anova. [Online]. Available: https://www.qualtrics.com/experience-management/research/anova/ 4.2.2

[50] L. F. P. Guachalla. Prueba de normalidad de shapiro-wilk. [Online]. Available: https://rpubs.com/F3rnando/507482 4.2.2

[51] R studio. [Online]. Available: https://www.rstudio.com/ 4.2.2

[52] Fair use. [Online]. Available: https://es.wikipedia.org/wiki/Uso_justo A.1

[53] Límites a los derechos de autor. [Online]. Available: https://www.cedro.org/propiedad-intelectual/limites-y-excepciones A.1