Control Remoto de Videojuegos con Smartphones

MEMORIA DE TRABAJO DE FIN DE GRADO

Pablo Gómez Calvo
Sergio J. Higuera Velasco

Grado de Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

Diciembre 2020

Documento maquetado con TEXIS v.1.0+.

Control Remoto de Videojuegos con Smartphones

Memoria de Trabajo de Fin de Grado
Grado de Desarrollo de Videojuegos
Director: Carlos León Aznar
Director: Pedro Pablo Gómez Martín

Versión 0.1.6

Grado de Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

Diciembre 2020

A todo aquel que confió en nosotros

Agradecimientos

Nadie es innecesario.
Yitán, Final Fantasy IX

El primer agradecimiento hay que dárselo a la Universidad Complutense por aceptar la creación de este grado, un grado que demuestra la importancia del mundo de los videojuegos en la sociedad actual. Con este grado se han conseguido romper muchas barreras, entre ellas está poder especializarse y adoptar los videojuegos como nuestra profesión.

Dar gracias a los profesores que nos han acompañado estos años y que han contribuido en el desarrollo del grado. Una mención aparte para las dos personas que han hecho posible la realización de este Trabajo de Fin de Grado, Carlos León Aznar y Pedro Pablo Gómez Martín.

Nuestro último agradecimiento va dirigido a nuestras familias por soportarnos en todos nuestros momentos durante el grado.

Resumen

¡No estáis preparados!
Illidan Tempestira, World of Warcraft

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

La relación entre los humanos y las máquinas se ha ido consolidando en las últimas décadas hasta el punto que son imprescindibles para algunas de nuestras tareas cotidianas. La tecnología extiende nuestras capacidades y durante las últimas décadas se han ido forjando y mejorando las interfaces de comunicación hombre-máquina. Las pantallas táctiles y los controles por voz son algo a lo que la sociedad se ha acostumbrado y una gran parte de la responsabilidad de este hecho reside en los dispositivos móviles. Junto con esta constante mejora en las interfaces humano-máquina también puede encontrarse una introducción a la tecnología háptica en modo de pulso para saber que el teléfono ha registrado una orden o pulsación en la pantalla.

1.1. Motivación

La industria del videojuego desde su inicio nos ha enseñado que la innovación a la hora de crear experiencias nuevas para los usuarios es algo que enriquece a muchos jugadores, por ende se está ayudando a que la experiencia de juego sea cada vez más cómoda y flexible para los jugadores.

Es por esta razón que empresas como Electronic Arts, Ubisoft, Kunos Simulazioni y Polyphony Digital,entre otras, dedican gran cantidad de sus recursos a hacer realidad muchas experiencias que los usuarios quieren tener como, por ejemplo, jugar a juegos deportivos realistas como en FIFA o conducir automoviles de competición por un circuito famoso como en Assetto Corsa.

Pero para invertir en crear este nuevo tipo de estilos de juego se necesita una gran financiación, cosa que estudios pequeños no tienen. Estos estudios independientes usan motores como Unity3D o Unreal Engine 4. Estos motores, cuyo pago es porcentual a las ganancias obtenidas por tu juego o en algunos casos mensual, ofrecen una gran cantidad de herramientas a disposición de sus usuarios para que los desarrolladores puedan ahorrar tiempo de implementación de una nueva características y lo utilicén para que su juego

siga creciendo. En el caso de Unity el pago es mensual dependiendo de la cantidad de dinero generado por el usuario o su empresa; los precios van desde gratuito si hace menos de 100 mil dólares anuales hasta 125 dólares mensuales si el usuario o su empresa genera más de 200 mil dólares anuales. Si hablamos del pago de Unreal Engine, si cualquiera de los productos realizados por el estudio se comercializa de forma oficial, Epic Games obtendría el 5 % de los beneficios de la obra cada trimestre cuando este producto supere sus primeros 3000 dólares. El desarrollo para dispositivos móviles en estudios independientes ha crecido de manera exponencial gracias a que motores como Unity lo hacen bastante accesible.

Unity es utilziado por el 34 % del top mil de juegos para dispositivos móviles. Algunos de estos juegos son *Alto's Adventure*, *Monument Valley* o el famoso *Hearthstone* de Blizzard en su versión de Android e iOS.

Entonces, ¿y si un juego se pudiese jugar en el teléfono móvil pero en verdad el juego se estuviese ejecutando en el ordenador?

Con esta pregunta se pretende poner encima de la mesa las tecnologías desarrolladas por diferentes empresas como **Nintendo** con Nintendo Switch que alterna modo portátil con modo sobremesa, lo que da una flexibilidad a la hora de jugar donde quieras que nunca se había experimentado, como **Sony** con sus aplicaciones **PS4 Remote Play** y **PS4 Second Screen** que te dan la posibilidad de controlar de manera remota e incluso jugar desde tu dispositivo iOS o Android.

Para la conexión de una consola o un ordenador a un dispositivo móvil o tablet es necesaria una conexión a internet estable. El tiempo de respuesta, lo que se conoce como *Input Lag*, puede llegar a convertirse en un quebradero de cabeza para un estudio pequeño ya que requiere de pruebas de rendimiento, pruebas con usuarios y pruebas con diferentes anchos de banda de red para buscar posibles cuellos de botella.

Este trabajo pretende aplicar los modelos propuestos anteriormente y crear una herramienta libre para el motor de videojuegos Unity con la finalidad de dar a estudios independientes y con escasa o nula financiación la oportunidad de habilitar nuevo gameplay para sus usarios. La finalidad es dar todas estas herramientas de una forma rápida e intuitiva con documentación y una prueba de implementación de este Plugin con uno de los juegos que se ofrecen de manera gratuita en la tienda de Unity, la Asset Store. Además de esto, la herramienta consta de una licencia libre, lo que da la posibilidad de ampliación y modificación dependiendo de las necesidades de cada usuario/estudio.

1.2. Objetivos

Los objetivos de este proyecto son:

- Desarrollar y publicar un *plug-in open source* para Unity que permita establecer conexión y recibir *input* desde otro dispositivo.

- Desarrollar y publicar una aplicación *open source* para Android que permita establecer conexión con otro dispositivo para ser usado como dispositivo de entrada.

- Evidenciar y realizar un estudio posterior de los resultados del proyecto através de una serie de pruebas con usuarios.

La finalidad del proyecto es la conexión entre 2 dispositivos, uno de ellos ejecuta el juego y el otro funciona como un dispositivo de entrada / mando para controlar el videojuego. Dentro de Unity existe una plataforma en la que los diferentes creadores de contenido suben sus creaciones para que otros desarrolladores las utilicen para sus proyectos. Estas herramientas pueden ser tanto de pago como gratuitas y entre ellas se encuentran modelos 3D, recursos de audio como música y efectos de sonido y *plug-ins*. Un *plug-in* es *software* que se introduce dentro de un programa para añadir nuevas funcionalidades.

1.3. Metodología

Como metodología de desarrollo se ha decidido usar una metodología ágil de producción que es habitual en la industria del desarrollo de software y videojuegos. Se ha elegido una metodología ágil por la experiencia positiva en proyectos previos.

Scrum, propuesto por Schwaber and Sutherland (2018), es un framework para la gestión de proyectos de trabajo en equipo. El paradigma se basa en un principio simple: comenzar con metas a corto plazo que formen parte del resultado final, tras esto se sigue el progreso y modifica según se avance en el proyecto.

Debido a los problemas de disponibilidad durante el desarrollo del proyecto, se realizaba una pequeña reunión de 10 minutos cada 1-3 días para ver el progreso de cada uno de los integrantes. En estas reuniones se revisaba la planificación para la siguiente reunión o la siguiente semana. Al principio del desarrollo fueron necesarias reuniones mucho más largas que en muchas ocasiones incluian a los directores del TFG en las que se discutían las diferentes *features* que deberían tener las aplicaciones que se estaban desarrollando. Estas reuniones más extensas servían como cierre de *Sprint* y como preparación del siguiente. El seguimiento de las diferentes tareas a realizar se realizaba usando la herramienta online **Pivotal Tracker** donde se marcaban las tareas con 3 posibles estados: “Open”, “In Progress” o “Done”.

1.4. Planificación

La planificación del desarrollo de este proyecto se ha dividido en 3 fases:

Documentación y diseño: Durante esta fase tratamos de delimitar claramente el alcance y objetivos del TFG, reunir fuentes y referencias y preparar las herramientas que se utilizarían durante el resto del desarrollo. Además, en esta primera fase se realizó un diseño de las aplicaciones que se desarrollarían en los meses posteriores.

Desarrollo: Durante esta fase del desarrollo se desarrollaron todas las funcionalidades especificadas en la fase anterior del proyecto. Al hacerse revisiones periódicas de la implementación, algunas de las funcionalidades iniciales sufrieron cambios o fueron eliminidas y se añadieron otras que encajaban más con el rumbo que estaba tomando el desarrollo. Esta fase ha ocupado la mayor parte del tiempo que ha tomado realizar este proyecto. Durante esta fase se realizaron 2 aplicaciones en forma de demo con las que poder probar la herramienta y demostrar la viabilidad del proyecto.

Cierre: Durante la fase final del desarrollo se realizaron las mejoras finales de las aplicaciones y se refinaron los últimos detalles de rendimiento. En esta fase también se realizaron las pruebas con usuarios para extraer datos tanto de rendimiento de las aplicaciones como de posibles fallos y mejoras de las aplicaciones. Estos datos se han refinado, filtrado y analizado y han servido para extraer las conclusiones finales de este trabajo. Además, durante esta última fase del desarrollo se ha trabajado en terminar la redacción de este documento junto con la revisión por los directores de este TFG.

1.5. Estructura del documento

Este proyecto se divide en 6 capítulos, cada uno de ellos dedicado a una temática. Esta sección está situada en el capítulo de introducción donde se ha definido la motivación y los objetivos del proyecto.

El capítulo 2 recoge el estudio inicial del estado del arte en el cual se exponen los antecedentes de la temática del proyecto. Está dividido en varias secciones, la primera hace un repaso de la evolución que han sufrido los dispositivos de entrada desde que desarrolló el teclado hasta la última generación de consolas. La siguiente sección trata la tecnología háptica y la retroalimentación en los controladores de videojuegos. La última sección expone el funcionamiento de un sistema de streaming en red.

En el capítulo 3 se explica todo lo relacionado con la especificación de las aplicaciones que se van a desarrollar. Esta especificación incluye una descripción del protocolo de comunicación entre dispositivos necesario para

este proyecto.

En el capítulo 4 se explica de manera detallada la implementación de las aplicaciones que se han desarrollado para este proyecto. Junto a la implemetación, se explica la demo desarrollada para probar la viabilidad del *plug-in* desarrollado y como caso práctico para la presentación de este proyecto.

El capítulo 5 recopila el pequeño experimento que se ha llevado a cabo con diferentes usuarios para probar la aplicación y recopilar *feedback* de los diferentes usuarios que han probado la demo. También se describen los participantes, los resultados obtenidos y la discusión sobre estos resultados.

En el último capítulo se explican de manera detalla las conclusiones obtenidas después de realizar el proyecto y una visión general del trabajo futuro que inspira este proyecto.

Capítulo 2

Estado del arte en entrada de usuario para videojuegos

A lo largo de las diferentes generaciones de computadores y de consolas se han ido desarrollando una serie de dispositivos de entrada que permiten al usuario interactuar con la máquina. Estos dispositivos van desde teclados y ratones hasta cámaras que permiten transformar tus movimientos físicos en movimientos dentro de un entorno virtual pasando por detectores de aceleración y pantallas táctiles. En este capítulo se presentan muchos de los trabajos pasados en el ámbito de los dispositivos de entrada de usuario.

2.1. Evolución de los dispositivos de entrada

Los dispositivos de entrada son aquellos que permiten introducir datos o información en un ordenador para que este los procese u ordene. Otro término usado para estos dispositivos es periférico. A pesar de que este término implica a menudo el concepto de adicional y no esencial, en muchos sistemas informáticos son elementos fundamentales. Estos dispositivos de entrada pueden clasificarse según el tipo de entrada, ya sea sonora, visual, de movimiento mecánico, etc. o dependiendo de si su forma de entrada es discreta (pulsaciones de teclas) o continua (una posición).

Todos estos dispositivos de entrada son conocidos como **HID, Human Interface Device**. La principal motivación para HID era la de permitir innovaciones en los dispositivos de entrada a los ordenadores y así simplificar el proceso de instalación de este tipo de dispositivos. Antes de HID, los dispositivos de entrada se ajustaban a unos estrictos protocolos diseñados

para ratones, teclados y joysticks. Con cualquier innovación en el hardware se requería de sobrecargar el protocolo existente o la creación de un driver. Un solo driver HID analiza los datos de entrada y permite una asociación dinámica de estos datos con la funcionalidad descrita por la aplicación. En el protocolo HID existen 2 enidades: el host y el dispositivo.El dispositivo es la entrada que intercatúa directamente con el humano, un ejemplo puede ser el teclado o ratón. El host es el que recibe los datos de entrada del dispositivo con las acciones ejecutadas por el humano, el host suele ser un ordenador. Los dispositivos definen sus paquetes de datos y luego presentan un descriptor HID al host. El descriptor HID es codificado como un grupo de bytes que describen los paquetes de datos del dispositivo. Esto incluye: cuantos paquetes soporta el dispositivo, el tamaño de los paquetes, y el propósito de cada byte y bit en el paquete. Se espera que el host sea la entidad más compleja de las 2 ya que el host necesita obtener el descriptor HID del dispositivo y analizarlo antes de establecer la comunicación. El HID también define el modo arranque, este modo es limitado ya que paquetes de datos son utilizados durante ese modo. Los únicos dispositivos que soportan el modo arranque son teclados y ratones.

Un teclado es un HID que se representa como una disposición de botones o teclas. Cada una de estas teclas se puede utilizar para ingresar cualquier carácter lingüístico o hacer una llamada a cualquier función particular del ordenador. Los teclados tienen su inspiración en las máquinas de escribir que tienen su origen en el año 1877, cuando la marca Remington comenzó a comercializar de manera masiva las máquinas de escribir. Uno de los primeros avances de estas máquinas de escribir ocurrió en la década de 1930, cuando se combinaron la tecnología de la entrada e impresión de las máquinas de escribir con la tecnología de la comunicación del telégrafo. Además, los sistemas de tarjetas perforadas también se combinaron con las máquinas de escribir para crear perforadoras de teclado que fueron la base para las primeras calculadoras. La emergente máquina de escribir eléctrica mejoró aún más la unión tecnológica entre la máquina de escribir y el ordenador. Así, en 1955, el Whirlwind del MIT, se convierte en el primer ordenador del mundo que permite a sus usuarios introducir comandos a través de un teclado y confirma lo útil y conveniente que puede ser un dispositivo de entrada de teclado. En 1964 se impuló el desarrollo de una nueva interfaz de usuario llamada terminal de visualización de video que permitió a los usuarios de las primeras computadoras ver qué carcacteres estaban escribiendo. En los años 80, IBM lanzó su primer ordenador personal que incluía un teclado mecánico. Actualmente las principales mejoras que han sufrido los teclados de ordenador se basan en la eliminación de cables gracias al Bluetooth o WI-FI. Con la llegada de los dispositivos táctiles se añadió además el concepto de teclado virtual. Este teclado virtual elimina el uso de un teclado hardware

para pasar a un teclado software que imita el teclado tradicional QWERTY pero en una pantalla táctil.

Además del teclado, el segundo dispositivo de entrada por excelencia es el ratón. La primera maqueta fue diseñada durante los años 60, disponía de 2 ruedas metálicas que al desplazarse por una superficie movían 2 ejes. Cada uno de estos ejes controlaba el movimiento tanto vertical como horizontal del cursor en la pantalla. Durante lo sños posteriores se añadieron botones, una rueda central o lateral para el desplazamiento, el sensor de movimiento óptico por diodo led o un sensor basado en un láser no visible. En los primeros años de la informática, el teclado era el dispositivo más popular para la entrada de datos pero la aparición y éxito del ratón, junto con la evolución de los sistemas operativos, lograron facilitar y mejorar la comodidad a la hora de manejar ambos periféricos a la vez. Los ratones suelen estar preparados para manejarse con ambas manos pero algunos fabricantes también ofrecen modelos especícos para zurdos y diestros. Los sistemas operativos también han ayudado a que ambos tipos de personas tengan un uso satisfactorio de los ratones. Con el avance de los nuevos ordenadores, el ratón se ha convertido en un dipsositivo esencial a la hora de jugar videojuegos, sirviendo no solo para seleccionar y accionar objetos en pantalla en juegos de estrategia, sino para controlar la cámara o cambiar la dirección del personaje en juegos de primera y tercera persona.

Los videojuegos han sido los principables responsables de la evolución de los dispositivos de entrada en las décadas posteriores a los años 70. En 1972 fue lanzada de manera oficial la **Magnavox Odyssey** y fue considerada la primera videoconsola. El dispositivo de entrada para poder jugar consitía de 2 diales que se utilizaban para el movimiento horizontal y vertical del personaje, un cuadrado blanco. Como juego para la consola Odyssey sacaron el Magnavox Odyssey Shooting Gallery en 1972. El mando que se usaba para poder jugar a este juego tenÃa la forma de un rifle y la peculiaridad que tenÃa este accesorio/mando era que los disparos se registraban siempre y cuando el rifle apuntase a una luz intensa por lo que era muy fácil de engañar si no fuese porque el juego no disponía de un registro de puntos. En 1976 la empresa Fairchild Semiconductor sacó al mercado la **Fairchild Channel F** cuya característica principal a nivel de entrada de usuario fue la incorporación de un joystick de 8 direcciones. Además de ofrecer un movimiento en 8 direcciones, la parte de arriba de este mando podía girarse para ser compatible con juegos como *Pong* y también podía ser pulsado y usarse normalmente como botón de disparo. En 1977 salió al mercado uno de los joysticks más famosos. Este joystick es el que se utilizaba en la consola **Atari 2600**. Este joystick se conocía como el **Atari CX40** y consistía de una palanca que permitía un movimiento en 8 direcciones y un botón. Junto con

este modelo, Atari sacó al mercado un tipo de conexión que se convertiría en el estándar de la indústria y que sería compatible con sistemas posteriores. Unos pocos años después, en 1982, Atari lanzó su nueva consola Atari 5200. Además de mejorar gráficamente, el controlador que llevaba incorporado proviene de un kit de controlador de un avión RC. El sistema combinaba un diseño mecánico demasiado complejo con un sistema de circuito flexible interno de muy bajo coste. Este controlador incluyó un botón de pausa, una característica única en ese momento.

En 1983 Nintendo sacó al mercado su **Nintendo Entertainment System (NES)** cuyo controlador sentó unas bases en el control del movimiento gracias a la cruceta que incorporaba. Esta cruceta permitía un movimiento en 4 direcciones y que pretendía reemplazar a las voluminosas palancas de mando de los controladores. Además de la cruceta, el mando disponía de 2 botones redondos (A y B) y otros 2 botones rectangulares (START y SELECT). En lo sucesivo, se lanzaron varios dispositivos especiales diseñados precisamente para usarse con juegos específicos, aunque muy pocos de estos se volvieron populares. Uno de estos dispositivos era el **Power Glove**, el que sería considerado como uno de los primeros periféricos de interfaz en recrear los movimientos de la mano en una pantalla de televisión o de un ordenador en tiempo real. En 1990 Nintendo hizo evolucionar a la Nintendo NES y lanzó la **Super Nintendo Entertainment System (SNES)**, la cual dejó atrás un diseño cuadrado del controlador y se inclinó por un diseño más ergonómico, se mejoró la cruceta y se añadieron otrós 2 botones (X e Y). El tiempo de respuesta del controlador era de 16 milisegundos. Dentro de la evolución de los controladores, en 1993 la compañia SEGA sorprendió con el lanzamiento de un nuevo accesorio para su consola **Sega Mega Drive**. Este accesorio consistía en un aro octogonal que se colocaba en el sueloy se conectaba directamente al puerto de controlador de la consola. Lo llamaron **Sega Activator** y fue el primer controlador que utilizaba el cuerpo completo. El jugador se tenía que situar en el centro del aro, el cual emitía rayos infrarrojos hacia arriba para detectar los movimientos del jugador. Los juegos orientados para el Sega Activator eran juegos que involucrasen el movimiento de brazos y piernas para que el jugador cruzase los rayoss infrarrojos y así se detectase el movimiento. Al tratarse de 8 segmentos, cada uno de estos segmentos estaba mapeado como si fuera un botón en el mando tradicional, el cual se "pulsaríaçada vez que el jugador cruzase un segmento de los rayos infrarrojos.

Durante los años 90 **Sony** entró al terreno del desarrollo de consolas y por consecuencia, de modelos diferentes de controladores de videojuegos. Con su primera consola, la **Sony PlayStation**, incluyeron un nuevo diseño de mando que recogía muchos de los diseños vistos hasta el momento. A diferencia

de Nintendo, este controlador cambió la nomenclatura de los botones A,B,Y y X por las figuras △, O, X y □, mantenía la cruceta y los botones START y SELECT y además añadió 4 botones más en la parte lateral del mando para los dedos índice y corazón. 3años más tarde Sony sacaría una re-edición del mando al que le incorporaron 2 stickts analógicos junto con un botón con un LED para cambiar entre los diferentes modos usados para el control del personaje. Este modelo fue el predecesor del famoso **DualShock** y únicamente la versión japonesa presentaba una función de retroalimentación de vibración. Por el lado de Nintendo, la consola sucesora de la Super Nintendo fue la **Nintendo 64** que fue acompañada por un nuevo diseño de mando que no pasó desapercibido. Disponía de una cruceta en la parte izquierda del mando, un stick de 360 grados y un botón START en el centro del mando y 6 botones en su parte derecha. Complementario a esto, en la parte trasera del mando había 2 botones más y también en la parte trasera se daba la opción de introducir un dispositivo extraible que proporcionaba retroalimentación de vibración. Este accesorio se activaba en ocasiones concretas como al disparar un arma y servía para sumergir al jugador en el videojuego.

En los años posteriores las compañias siguieron sacando modelos direferentes mandos que modificaban tamaño y posiciones de los botones pero no salieron cambios significativos hasta que en 2002 Nintendo lanzó al mercado un nuevo mando alternativo para su consola **GameCube**, este controlador tenía la peculiaridad de ser inalámbrico. Lo llamaron **WaveBird Wireless Controller** y sentó las bases para los próximos mandos inalámbricos. Contaba con una cruceta, 6 botones digitales, 2 botones híbridos ya que hacian la función de gatillos y 2 palancas analógicas para el movimiento del personaje y la cámara normalmente. Como alimentación usaba 2 pilas AA y para comunicarse con la consola usaba radiofrecuencia, lo que permitía al jugador alejarse hasta 6 metros de la consola. Poco tiempo después tanto Sony con su PlayStation 3 como Microsoft con su Xbox 360 añadirían las baterias a sus mandos para convertirlos en inalámbricos.

En 2006 Nintendo volvió a sorprender a la sociedad con la llegada de la **Nintendo Wii** y su nuevo mando. **Nintendo Wiimote** es el mando principal de la consola Wii y las características destacables que trae son la de la detección de movimiento en el espacio y la habilidad de poder apuntar a objetos en la pantalla. El diseño del mando de Wii deja de lado todos los modelos tradicionales de mandos para videojuegos y se acerca más a un control remoto de televisión para que este pueda usarse con una sola mano y sea más intuitivo ya que lo que pretendía era cautivar a un público más casual y ser una consola para todos los miembros de la familia. En la cara frontal del mando se encuentran los botones “A”, “1”, “2”, “+”, “-”, “HOME” y la cruceta, más un botón “POWER” para apagar la consola, algo inédito hasta enton-

ces. En la parte anterior sólo presenta el botón "B", en un formato similar a un gatillo. Además de los botones, en la parte frontal lleva incorporado un altavoz y 4 luces numeradas que indican el júmero de jugador al que corresponde cada mando durante una partida. Lleva una correa de seguridad udia al mando por laparte inferior de este para poder atar el mando a la muñeca y evitar asi que el mando se resbale durante una sesión de juego y o el mando o la televisión se vieran dañados. El Wii Remote tiene la capacidad de detectar la aceleración a lo largo de tres ejes mediante la utilización de un acelerómetro. El Wiimote también cuenta con un sensor óptico PixArt, lo que le permite determinar el lugar al que el Wiimote está apuntando; además de agregar una brújula electrónica en su posterior versión mejorada, el **WiiMotionPlus.** A diferencia de controladores que detectan la luz de la pantalla del televisor, el Wii Remote detecta la luz de una barra sensor que venía incorporada con la consola y se colocaba encima de la pantalla. No es necesario señalar directamente a la barra sensor, pero apuntar significativamente fuera de la barra de posición perturbará la capacidad de detección debido al limitado ángulo de visión del Wiimote. La posición y seguimiento del movimiento del Wii Remote permite al jugador imitar las acciones reales de juego, como blandir una espada o una pistola con objetivo, en lugar de simplemente pulsando los botones. El Wii Remote tiene incorporado un altavoz que emite sonidos diferentes que los emitidos por lo altavoces de la televisión y que se utilizan para mejorar la ambientación del videojuego. El ejemplo de esto se mostró en el E3 de 2006 cuando uno de los desarrolladores disparó un arco en "The Legend of Zelda: Twilight Princess" y por el mando sonó como tensaba y soltaba la flecha mientras que en la televisión únicamente se escuchaba el impacto de la flecha. Esto daba la sensación de que la flecha viajaba desde el jugador hasta la televisión. Esto se veía acompañado por la vibración del mando y ambas funciones podían deshabilitarse desde el menú HOME de la consola, en ese caso todos los sonidos saldrían por los altavoces de la televisión. En cuanto a la duración de las baterias, con las funciones más básicas de algunos juegos tenía la autonomía de 60 horas y en caso de usar todas sus capacidades llegaba a aguantar 25 horas funcionando. Al tratarse de un mando simétrico su uso estaba pensado tanto para zurdos como para diestros y se utilizaba tanto de manera vertical con una sola mano como de manera horizontal en caso de usar ambas manos. Además de todo lo anterior, en la parte inferior del mando se encuentra un puerto de expansión para conectar diferentes periféricos. Los más usado fueron el **Wii Balance Board** que se trataba de una tabla capaz de calcular la presión que se ejercía sobre ella, el **Wii Guitar** que permitía jugar al juego Guitar Hero en su versión de Wii, **Nunchuck** que agregaba un joystick analógico y 2 botones más para el movimiento de personajes, **Wii Wheel** que hacía que el Wii Remote se convirtiese en un volante y asi poder tener una mejor experiencia en juegos como Mario Kart y **Wii MotionPlus** que incorporaba 3 sensores nuevos de movimiento para los ejes vertical, longitudinal y lateral

lo que ayudó a mejorar la precisión del mando original.

En 2010 Microsoft dio el salto a un nuevo controlador de videojuegos para su consola Xbox 360. Este nuevo periférico es conocido con el nombre de **Kinect** y permite a los usuarios controlar e interactuar con la consola sin necesidad de tener contacto físico con un mando tradicional. Este control se realiza por gestos y reconocimiento de voz. El sensor Kinect es una barra horizontal de unas 9 pulgadas conectada a una pequeña base circular con un eje que permiteque esta rote y además está diseñado para ser colocado por encima o por debajo de la televisión. El dispositivo cuenta con una cámara RGB, un sensor de profundidad, un micrófono de múltiples matrices y un procesador personalizado que ejecuta el software patentado, que proporciona captura de movimiento de todo el cuerpo en 3D, reconocimiento facial y capacidades de reconocimiento de voz. El micrófono de matrices del sensor de Kinect permite a la Xbox 360 llevar a cabo la localización de la fuente acústica y la supresión del ruido ambiente, permitiendo participar en el chat de Xbox Live sin utilizar auriculares. El sensor de profundidad es un proyector de infrarrojos combinado con un sensor CMOS monocromo que permite a Kinect ver la habitación en 3D en cualquier condición de luz ambiental. El rango de detección de la profundidad del sensor es ajustable gracias al software de Kinect capaz de calibrar automáticamente el sensor, basado en la jugabilidad y en el ambiente físico del jugador, tal como la presencia de sofás, mesas y otro tipo de muebles.

PlayStation también lanzó al mercado su dispositivo de control de videojuegos por movimiento, ellos lo llamaron **PlayStation Move** y es compatible con los sitemas PS3 y PS4. PlayStation Move compitió tanto con el Kinect de Xbox como con el WiiMote de Nintendo. El diseño del mando consiste en un mando similar al de Wii ya que ambos se controlar con una mano en vertical. PlayStation Move unió los conceptos de sensores de movimiento que usaba el WiiMote y la cámara que usaba el Kinect, por lo que PlayStation Move usa sensores de movimiento en el mando,una esfera en su extremo que se ilumina y la cámara PlayStation Eye que detecta la posición del mando.Al igual que en el resto de controladores inalámbricos para PlayStation, tanto el mando principal de PlayStation Move como el Navigation Controller usan la conexión inalámbrica Bluetooth 2.0 y una batería de ion de litio, que se carga mediante un puerto USB Mini-B. El Naigation Controller es un mando que complementa al mando principal y tiene una función similar a la del Nunchuck de Wii. Se pueden conectar hasta 4 PlayStation Move de manera simultánea.

Coincidiendo con la salida al mercado del PlayStation Move, Nintendo lanzó su nueva consola en 2012; la **Wii U** y con ella un Controlador de

videojuegos híbrido. Este controlador híbrido es el **Wii U GamePad** y es el mando principal de la consola. La principal distinción con respecto a los mandos tradicionales es la incorporación de una pantalla táctil, la cual se utiliza para mostrar información adicional durante una partida y además puede usarse como pantalla principal en caso de no disponer de un televisor mientras se juega. Además de como controlador de videojuegos, el Wii U GamePad es utilizado como control remoto independiente para cotrolar la pantalla de la televisión u otro aparato via infrarrojos sin tener que tener la consola encendida. El mando constaba de altavoces y micrófono e incorporaba una cámara frontal de 1.3 megapixeles. Los sensores que incorporaba eran: acelerómetro, giroscopio, geomagnético e infrarrojo e incorporaba vibración. La conexión a la consola se hacía mediante bluetooth y disponía de NFC para futuros accesorios que incorporaron a los diferentes videojuegos.

En 2013 Sony hizo una revisión de su DualShock y como mando de la consola PlayStation 4 lanzó el **DualShock 4**. Este mando añadió al diseño anterior una pantalla táctil en la parte frontal, lo que hizo que la distribución de los botones que antes eran centrales cambiasen. En la parte trasera se añadió una barra LED que se ilumina en varios colores para diferenciar e identificar a los diferentes jugadores. Microsoft por su parte en 2015 puso a la venta su nuevo **Xbox One Elite Controller**. Una de las principales características es que tiene un diseño modular por lo que cada pieza es intercambiable para que cada jugador pueda adaptarlo a su medida. La personalización es el principal aliciente en este mando ya que permite reprogramar tanto los cuatro botones tradicionales de la parte frontal como los gatillos. También se da la posibilidad de establecer curvas de sensibilidad en las palancas analógicas. Gracias a la posibilidad de añadir piezas, este mando incluye 4 palancas más en la parte trasera.

En 2017 Nintendo presentó su nueva consola híbrida que se basa en su predecesora, Wii U. La consola es **Nintendo Switch** y es una consola que puede ser jugada tanto de manera portátil como de sobremesa en una televisión o un monitor. Loa mando diseñados para esta consola son los **Joy Con** y consisten en 2 unidades, cada uno de ellos contiene una palanca analógica y una matriz de botones. Estos mandos tienen la peculiaridad de que pueden usarse tanto acoplados a la consola cuando esta se utilice en modo portátil o pueden ser desacoplados para cuando la consola se utilice en una televisión. Cuando se separan, un par de Joy-Con pueden ser utilizados por un solo jugador, o dividido entre dos como controladores individuales. Los Joy-Con se distribuyen en pares, designados como “Joy-Con L” y “Joy-Con R”, respectivamente. Este diseño viene heredado de Wii con su Wiimote y el accesorio principal el Nunchuk. Una misma Nintendo Switch puede tener conectados hasta un total de 8 Joy-Con. La comunicación que tienen con

la consola se realiza por Bluetooth y disponen de una correa igual a la del mando de Wii para evitar la caida del mando durante su uso. Los Joy-Con contienen baterías no extraibles de 525 mAh, que se cargan cada vez que se conectan a la consola. Ambos controladores contienen una palanca analógica, cuatro botones de cara, dos botones superiores, dos botones laterales accesibles cuando se sueltan y designados como SL y SR, un botón "+" o "-", un botón de sincronización y un indicador de jugador luces LED. Cada uno de los Joy-Con contiene un acelerómetro y un giroscopio, que pueden ser utilizados para los juegos que incluyen movimiento. La diferencia con el mando de Wii es que estos mandos no necesitan una barra de sensores para su correcto funcionamiento. Además, el Joy-Con R contiene un sensor de seguimiento de profundidad infrarroja, que puede leer objetos y movimientos sostenidos delante de él. Cada uno de los Joy-Con incorpora un motor para la vibración del mando durante las sesiones de juego. Poco después de su salida, se descubrió que los Joy-Con pueden conectarse y utilizarse con otros ordenadores personales y con dispositivos móviles a través de la conexión Bluetooth.

También durante 2017, Sony anunció una serie de juegos nuevos que se jugarían de una forma totalmente diferente ya que el mando utilizado sería cada uno de los teléfonos móviles de los usuarios. A esta serie de juegos se la conoce como **PlayLink**. La idea detrás de PlayLink es que todos los usuarios de la consola PlayStation 4 y sus familiares y amigos disponen de un dispositivo móvil pero no todos disponen de varios mandos para poder jugar con más personas en juegos cooperativos. PlayLink es una aplicación móvil que cada usuario se descarga en su Android o iOS y así puede usar su teléfono como un mando más de PlayStation 4. El requisito para que la conexión sea efectiva es que tanto la consola PS4 como los dispositivos móviles que se vayan a usar estén conectados a la misma red WIFI.

A finales del año 2020 Sony lanzó al mercado su nueva consola, la PS5 y con ella un nuevo mando al que bautizaron como **DualSense**. Como ya pasaba con el DualShock 4, este mando funciona con batería y lleva un altavoz integrado. Además de esto, el DualSense incorpora un acelerómetro y un giroscopio para aquellos juegos compatibles con esta tecnología. La gran novedad que trae este mando es la retroalimentación háptica. Se han sustituido los motores de vibración tradicionales por 2 activadores que emiten vibraciones dinámicas capaces de simular todo tipo de sensaciones. Además de la nueva retroalimentación háptica, el nuevo DualSense incorpora 2 gatillos adaptativos. Estos gatillos emiten diferente fuerza de resistencia contra el jugador dependiendo del arma que se esté utilizando. El ejemplo que pusieron los desarrolladores de Sony fue con la cuerda de un arco, inicialmente no tiene resistencia pero cuanto más se tense la cuerda del arco más fuerza

es necesaria.

2.2. *Feedback* en los controladores

En los inicios de los videojuegos y de las consolas el objetivo fue la creación de nuevos tipos de juegos con diferentes mecánicas y mejoras gráficas. En la era actual de los videojuegos se ha ido mucho más lejos de los primeros juegos como Pong y Tetris, la tendencia ha llevado a la creación de escenarios virtuales más realistas y a que el jugador formase parte de ese entorno virtual. Particularmente la respuesta que se da al usuario del videojuego al realizar acciones es conocida como **tecnología háptica**. Este tipo de tecnología se refiere al conjunto de interfaces tecnológicos que interaccionan con una persona mediante el sentido del tacto. La tecnología háptica tiene sus inicios en los dispositivos de los sistemas servo para controlar grandes aviones con la intención de poder hacerlo de manera remota. En estos sistemas se instaló un sistema de control que proporcionaba una resistencia a la palanca del piloto proporcional al ángulo de ataque del avión. Otro ejemplo destacable de esta tecnología se encuentra en la película **4-D Honey, I Shrunk the Audience!** del año 1994 que simulaba que los ratones se soltaban por el auditorio y corrian por toda la sala. Para conseguir esta simulación se bombeaba aire a través de un pequeño tubo de plástico y al agitarse, imitaba la sensación de las colas de los ratones rozando las piernas de los espectadores.

En los videojuegos esta tecnología fue introducida a través de los controladores. Al inicio estos sistemas de vibración se introdujeron en los controladores como dispositivos que se acomplaban al mando por separado pero con la salida de la versión japonesa del controlador de PlayStation, el Dualshock. Este mando incorporaba un sistema de vibración conocido como *tabletas vibradoras (rumble packs)* que tenían ese efecto de vibración al conducir vehículos o disparar armas de fuego. Con la llegada de las pantallas táctiles también aparecieron las pantallas hápticas. Estas pantallas son aquellas que transmiten una vibración al tocarla. El ejemplo más actual de tecnología háptica puede encontrarse en la reciente consola de Nintendo, Nintendo Switch. Nintendo no ha denominado a la tecnología que usan sus Joy-Cons como tecnología háptica sino como **Rumble HD** pero Nintendo se ha asegurado en demostrar todas las capacidades que sus nuevos Joy-Cons pueden ofrecer en cuanto a este tipo de tecnología. Con su juego 1-2 Switch recogen una serie de minijuegos que tienen como objetivo el de mostrar las capacidades de los nuevos controladores, los Joy-Con. Entre estos minijuegos se encuentra uno en el que el jugador tiene que agitar el Joy-Con que simula un vaso con hielos y tiene que averiguar cuántos hielos hay en el vaso. Microsoft por su parte añadió a sus controladores de la consola Xbox One 4 motores de vibración que daban una mejor experiencia a los usuarios cuando

estos conducían, disparaban armas de fuego o producían explosiones. Sony, por su parte, ha introducido un nuevo modelo de mando al que ha llamado **DualSense** para su nueva consola **PlayStation 5**. Con este mando Sony ha dejado atrás la vibración tradicional de los motores que llevaban incorporados los DualShock 4 y han introducido una retroalimentación háptica mucho más definida y precisa. Esta tecnología recuerda al Rumbre HD que Nintendo incorporó en sus Joy-Con pero en este caso siguiendo la estética de los mandos tradicionales de Sony. Para complementar este *feedback*, el DualSense incorpora 2 gatillos adaptativos que ofrecen diferentes niveles de resistencia. Esto permite al mando simular efectos de dentro del juego directamente en las manos del jugador como la tensión de un arco al disparar una flecha o la diferencia entre disparar una escopeta y una ametralladora. Sony ha incluido con la consola el juego **Astro's Playroom** en el que con una serie de niveles y minijuegos se explotan al máximo las nuevas características que trae el nuevo DualSense a la PlayStation 5.

2.3. Sistemas de *streaming*

Una transmisión en directo consiste en una distribución digital de contenido multimedia a través de una red de ordenadores de manera que el usuario consume el producto a la vez que se descarga. El término *retransmisión* se refiere a una corriente continua que fluye sin interrupciones, normalmente la difusión es de audio o vídeo. Para referirse a una retransmisión se suele utilizar el anglicismo *streaming*. Esta tecnología funciona gracias a un búfer de datos que almacena el flujo de descarga en la estación del usuario e inmediatamente mostrar el material descargado. En contraposición a esto se encuntra la descarga de archivos completos que requiere que el usuario descargue los archivos necesarios al completo para poder acceder al contenido. En el año 1993 se consiguió que el grupo de música *Severe Tire Damage* actuase en directo a través de Internet. Este concierto tuvo lugar en California y pudo verse hasta en Australia gracias a la tecnología **Mbone**. Mbone es la abreviatura que se dio a *Multicast backbone (Red troncal de multidifisión)* y consistía en una red troncal experimental y una red virtual que se contruyó sobre internet para el transportal el tráfico de la multidifusión de IPs. Dado que los operadores de la mayoría de los enrutadores de Internet tenían deshabilitado la multidifusión IP debido a preocupaciones relacionadas con el seguimiento del ancho de banda y la facturación, Mbone fue creado para conectar redes con capacidad de multidifusión a través de la infraestructura de Internet existente en el momento. El propósito principal de Mbone era el de minimizar la cantidad de datos requeridos para realizar una videoconferencia a diferentes puntos simultáneamente. Algunas de las características de Mbone son:

- El protocolo de enrutado de multidifusión utilizado fue **DVMRP (Dis-**

tance Vector Multicast Routing Protocol). El funcionamiento de este protocolo consiste en la recepción de paquetes a un router y si este proviene de un camino que podría utilizarse para alcanzar al emisor del mensaje original, este paquete se difunde por todos los demás caminos activos. En caso contrario se elimina el paquete.

- La IP utilizada por Mbone fue 224.2.0.0
- Los requisitos de banda ancha para la transmisión de audio eran 32-64 kbits/s y para video 120 kbits/s.

En 1994 el grupo británico *The Rolling Stones* decidió retransmitir en directo y de forma gratuita 20 minutos de uno de sus conciertos. Con el paso de los años empresas como QuickTime, ActivePlayer,Adobe, etc siguieron realizando experimentos para mejorar esta tecnología. Solamente faltaba que las conexiones a internet a nivel global mejoraran.

Otro hito destacable en esta evolución fue el streaming ejecutado por Justin Kan, fundador de Justin.tv, ajustando una cámara web a su gorra y personalizando su portátil en su maleta para transmitir 24 horas durante los 7 días de la semana su vida cotidiana. Más adelante la compañia creada por Justin se transformaría en la famosa plataforma de streaming **Twitch.tv** que en el año 2019 alcanzó un pico de 982 millones de espectadores simultáneos en los diversos canales que emiten en la plataforma. Para poder proporcionar un acceso claro, convincente, continuo y sin interrupciones ni cambios, la retransmisión se apoya en las siguientes tecnologías:

- **Códecs**: Códec es el acrónimo de codificador-decodificador. Sirven para codificar el flujo o la señal y recuperarlo o descifrarlo para la reproducción en un formato adecuado para estas operaciones. Algunos códecs son MP3, formato de compresión de audio digital un algorítmo de pérdida para conseguir un menor tamaño de archivo. Para video se suele usar el códec H.264, el usado por la plataforma *Youtube* para la visulaización de sus videos y directos.
- **Secuencia de bits**: Las emisiones de audio y vídeo en códecs se ensamblan en un contenedor de secuencia de bits como **FLV (Flash Video)**. FLV fue ampliamente utilizado para transmitir vídeo por Internet sobre el complemento Adobe Flash Player. Flash Video puede ser visto en la mayoría de los sistemas operativos mediante el plugin Adobe Flash Player, disponible para la mayoría de navegadores web o de otros programas de terceros.
- **Protocolos de transporte** : El uso de protocolos de transporte ligeros como **UDP** o **RTSP (Real Time Spread Protocol)** es algo que se busca en la retrasnmisión de información para que el ancho de banda a usar sea menor. UDP y RTSP hacen que las entregas de paquetes de

datos desde el servidor a quien reproduce el archivo se hagan con una velocidad mucho mayor que la que se obtiene por **TCP** y **HTTP**. Esta eficiencia es alcanzada por una modalidad que favorece el flujo continuo de paquetes de datos. Cuando TCP y HTTP sufren un error de transmisión, siguen intentando transmitir los paquetes de datos perdidos hasta conseguir una confirmación de que la información llegó en su totalidad. Sin embargo, UDP continúa mandando los datos sin tomar en cuenta interrupciones, ya que en una aplicación multimedia estas pérdidas son casi imperceptibles. Aunque UDP no haga un control de transmisión, la aplicación que use este protocolo para la retransmisión tendrá que ser la encargada de realizarlo para tomar decisiones sobre qué hacer ante un posible extravío de información.

- **Ancho de banda**: Los anchos de banda recomendados para cada una las resoluciones incrementa de manera directamente proporcional con la calidad a la que se reproduce el contenido. Para la transmisión de un vídeo en definión estándar como la usada por Google TV, los discos Blue-Ray o Apple TV es de 2 Mbit/s. En caso de querer aumentar la calidad a alta deficinión se piden 5 Mbit/s y para el contenido con definición ultra se piden 9 Mbit/s. Si el archivo se almacena en un servidor para transmisión bajo demanda y 1000 personas ven este flujo al mismo tiempo usando un protocolo Unicast, el requisito es 300 kbit/s x 1000 = 300000 kbit/s = 300 Mbit/s de banda ancha. Esto equivale a alrededor de 135 GB por hora. Usando un protocolo de multidifusión, el servidor envía una sola transmisión que es común a todos los usuarios. Por lo tanto, dicha transmisión solo usaría 300 kbit/s de ancho de banda de servicio. El cálculo para la transmisión en vivo es similar. Suponiendo que la semilla en el codificador es de 500 kbit/s y si la emisión dura 3 horas con 3000 espectadores, los cálculos serían (500 kbit/s x (3 x 3600) segundos x 3000 personas) / (8 x 1024 x 1024) lo que da un resultado de 1,977,539 MB consumidos para la visualización de un directo por 3000 personas con un ancho de banda de 500kbit/s.

En España los servicios de retransmisión son populares entre la población y un estudio realizado por el Ministerio de Cultura de España durante los años 2018-2019 realizó una encuenta a 16000 personas sobre si se hacían descargas ilegales de contenidos o si por el contrario estaban suscritos a servicios con suscripción como Amazon Prime Video, Spotify, Netflix, etc.

Plataformas	%
Películas y series	38.9 %
TV	28.8 %
Música	26.8 %
Libros	3.4 %
Videojuegos	4.1 %

Tabla 2.1: Porcentaje de suscripciones totales a plataformas digitales en España en 2018-2019

La conclusión final fue que el 52.2 % de los usuarios encuestados tenía una suscripción activa a alguna plataforma digital.

Capítulo 3

Especificación del protocolo de comunicación

3.1. Introducción

En este capítulo se describen las diferentes decisiones tecnológicas involucradas en el proyecto y ver las decisiones tomadas en base a las necesidades para la realización de este proyecto. La finalidad del proyecto es la conexión entre 2 dispositivos, uno de ellos ejecuta el juego y el otro funciona como un dispositivo de entrada / mando para controlar el videojuego. Para conseguir esta comunicación entre ambos dispositivos es necesario plasmar la funcionalidad que se quiere dar y la posterior creación e implementación de un protocolo de red.

3.2. Funcionalidad

En los últimos años se ha visto un aumento en la incorporación de elementos táctiles en los controladores de videojuegos. Empresas como Sony añadieron una pantalla táctil en el Dualshock4, Nintendo con las consolas portátiles lleva varias generaciones añadiendo pantallas táctiles a sus consolas y los dispositivos móviles cada vez son más usados para jugar a videojuegos. Es por esto que en este proyecto, la comunicación entre 2 dispositivos será más precisa y será entre un dispositivo en el cual se ejecuta el juego y un dispositivo móvil que servirá como mando. Una de las principales diferencias entre un mando tradicional y un dispositivo móvil es la pulsación de botones. El feedback recibido en la pulsación de una pantalla táctil es inexistente a no ser que no se especifique en la funcionalidad de la aplicación y debe ser lo suficientemente sutil como para no ser molesto a la hora de jugar.

Otras de las características que tiene un dispositivo móvil frente a un mando tradicional es el uso de gestos específicos. Una gran parte de estos

gestos son utilizados de manera natural ya que son asociados al uso de dispositivos móviles.

Al usar un dispositivo móvil, tenemos a nuestra disposición una pantalla que nos permite añadir imágenes, videos o incluso gameplay como lo hacía Nintendo con su consola WiiU.

Con estas características contempladas, las principales funcionalidades que debe tener este proyecto son:

- Envío de pulsaciones en pantallas táctiles.
- Envío de imágenes estáticas.
- Envío de imágenes de manera constante, simulando un streaming de video.
- El uso de vibración como feedback de la pulsación de la pantalla.

3.3. Protocolo de comunicación entre juego y dispositivo de entrada

Un protocolo de comunicación es un sistema de reglas que permiten a 2 o más dispositivos comunicarse entre ellos. Estas reglas se establecen para permitir la transmisión de datos y la forma en la que la información debe ser procesada. Cada mensaje tiene un significado exacto destinado a obtener una respuesta de un rango de posibles respuestas predeterminadas para esa situación en particular. Una de las características principales de un protocolo de comunicación es que ambas partes tienen que acordar los mensajes que se van a enviar y a recibir.

En este proyecto ambos dispositivos envían y reciben mensajes por lo que no puede definirse un servidor y un cliente, sin embargo se puede hacer una distinción entre quién envía los botones que se han pulsado y quién ejecuta el juego. El criterio utilizado para este proyecto ha sido el de un sistema ***little-endian***. Al inicio de la comunicación, el dispositivo encargado de ejecutar el juego debe quedarse a la escucha en un puerto asignado a la espera de un primer mensaje. El tamaño de este mensaje es de 8 bytes y su estructura es la siguiente:

- Los 4 primeros bytes son un número de tipo *int* que simboliza el ancho del dispositivo móvil.
- Los 4 últimos bytes son un número de tipo *int* que simboliza el alto del dispositivo móvil.

La comprobación que se realiza en este mensaje es si se reciben 8 bytes en el lado del dispositivo que ejecuta el juego.

Bits	0-31	32-63
0	Ancho	Alto

Tabla 3.1: Primer mensaje del controlador al ejecutor de juego

Una vez la conexión se ha establecido correctamente, el dispositivo que ejecuta el juego envía al controlador la duración de la vibración que debe realizar para que el usuario reciba un feedback háptico. Este mensaje contiene un *int* de 4 bytes que representa el tiempo que debe durar la vibración en milisegundos.

Bits	0-31
0	Tiempo de vibración

Tabla 3.2: Tiempo de vibración del dispositivo movil en milisegundos

Tras el envío de este mensaje comienza un bucle de juego en el que ambos dispositivos intercambian mensajes de una manera no ordenada. El dispositivo que se encarga de ejecutar el juego envía los siguientes mensajes:

- En caso de que se decida enviar imágen de manera constante, ya sea en forma de streaming de video o una imágen estática, esta imágen es comprimida en formato ***PNG***.
- Se envía el flag de vibración al controlador. Este flag es un *int* de 4 bytes cuyo valor tiene que ser 0 para que vibre.

Además de enviar estos mensajes, el dispositivo encargado de ejecutar el juego recibirá mensajes de 9 bytes cada uno cuya estructura será:

- El primer byte es el tipo de la pulsación. El valor 0 indica el comienzo de la pulsación y el valor 1 indica el fin de la pulsación.
- Los siguentes 4 bytes representan un *int* cuyo valor indica la posición X de la pantalla del dispositivo móvil donde se ha pulsado el botón.
- Los últimos 4 bytes representan un *int* cuyo valor indica la posición Y de la pantalla del dispositivo móvil donde se ha pulsado el botón.

Bits	0-7	8-39	40-71
0	Tipo de pulsación	Posición X	Posición Y

Tabla 3.3: Pulsación enviada desde el dispositivo móvil al ejecutor del juego

Para el cierre ordenado de la comunicación desde el dispositivo móvil se envía un mensaje de tamaño 1 byte que tiene el valor 2. La recepción de

este mensaje hace que el dispositivo que ejecuta el juego envíe 1 byte con el valor 1 al dispositivo móviles y ambos cierran la conexión. Estos mensajes puedes enviarse y recibirse en cualquier momento en caso de que sucediese cualquier error durante la comunicación. Para la detección de una pérdida de conexión entre el dispositivo móvil y el dispositivo que ejecuta el juego, este dispone de un tiempo máximo en el que si no se reciben mensajes se finaliza la conexión (Time Out).

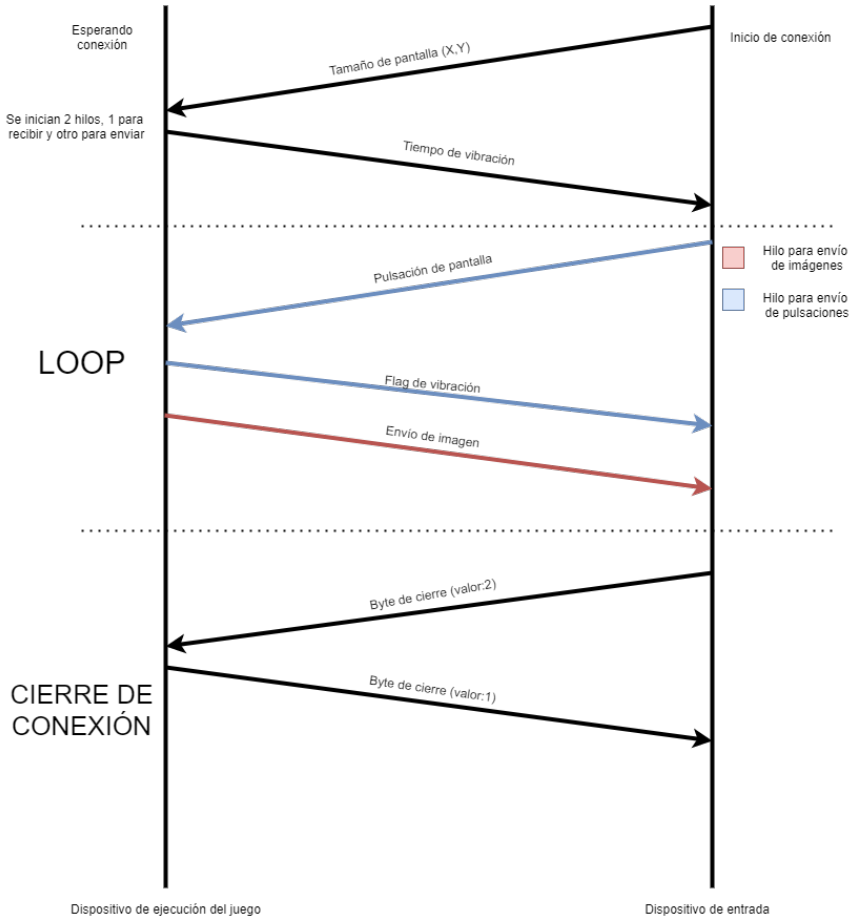

Figura 3.1: Diagrama del protocolo de comunicación entre ambos dispositivos

Capítulo 4

Implementación de las aplicaciones

En este capítulo se detalla todo el proceso de creación y desarrollo de las diferentes aplicaciones y la demo que posteriormente se utiliza como ejemplo de implementación de la herramienta. Se ha divido este capítulo en 3 secciones:

- Implementación de la aplicación de Android.
- Implementación de la aplicación de Unity.
- Inclusión de la herramienta en un juego cerrado.

4.1. Implementación de la aplicación de Android

Este proyecto se ha desarrollado utilizando principalmente 2 tecnologías: el motor de Unity y Android. Al tratarse de un proyecto de creación de un controlador para videojuegos y tomando como base la especificación desarrollada en el capítulo 3 se ha optado por el desarrollo de una aplicación para Android que sirva como dispositivo de entrada y una aplicación en Unity que sirva como ejecutor del juego.

Android Studio es el entorno de desarrollo integrado (IDE) oficial para la plataforma de Android. Hasta finales del 2014 para desarrollar en Android se utilizaba Eclipse como IDE oficial y está disponible para las plataformas Microsoft Windows, macOS y GNU/Linux. Android Studio incluye una gran variedad de herramientas para facilitar el desarrollo en Android entre las que se incluyen:

- Plantillas para crear diseños comunes de Android y otros componentes.
- Un editor de diseño enriquecido que permite a los usuarios arrastrar y soltar componentes de la interfaz de usuario.

- Soporte para construcción basada en Gradle.
- Un dispositivo virtual de Android que se utiliza para ejecutar y probar aplicaciones.
- Consola de desarrollador: consejos de optimización, ayuda para la traducción y estadísticas de uso.
- Integración de ProGuard y funciones de firma de aplicaciones.

Los lenguajes de programación aceptados por Android Studio son Kotlin, Java y C++. En concreto para este proyecto se ha usado Java como lenguaje de programación.

4.1.1. Ciclo de vida de Android

Las aplicaciones en Android se rigen por una serie de componentes que se llaman ***Activity***. Estos componentes son claves a la hora de manejar los estados de una aplicación en Android. A diferencia de otros paradigmas de programación que comienzan sus aplicaciones con un método ***main()***, la instancia de una Actividad invoca métodos de devolución de llamada que se corresponden con etapas específicas de su ciclo de vida. La experiencia de una aplicación para un dispositivo móvil difiere mucho de la versión de escritorio de esa misma aplicación ya que la interacción del usuario con la aplicación no siempre comienza en el mismo lugar. Un claro ejemplo de esto sucede con las aplicaciones de mensajería instantanea. Un usuario puede estar navegando por cualquier red social y encontrarse una publicación interesante y compartirla por correo electrónico o por una aplicación de mensajería instantanea. La aplicación de correo electrónico no se abre en el mismo estado si se abre desde la opción de compartir de la red social o si se abre desde el menú de aplicaciones instaladas en el dispositivo. Las Actividades están diseñadas para facilitar este paradigma.

La mayoría de las aplicaciones contienen varias pantallas, lo cual significa que contienen varias actividades. Este concepto se pondrá posteriormente de manifiesto ya que para el desarrollo de la aplicación han sido necesarias 2 Actividades. Una actividad proporciona una ventana en la que la aplicación dibuja la interfaz de usuario. Esta ventana puede estar a pantalla completa o puede ser más pequeña y flotar sobre otras ventanas como si fuese un *pop-up*.

Cuando un usuario navega por una aplicación, la cierra, la vuelve a abrir o la minimiza, las instancias de las Actividades de la aplicación pasan por una serie de estados de su ciclo de vida. Estos estados pueden tener comportamientos definidos por los desarrolladores de la aplicación. Esto permite tener un control sobre lo que ocurre en cada uno de los estados de la aplicación. Para navegar por las transiciones entre las etapas del ciclo de vida de

una actividad, la clase Activity proporciona un conjunto básico de seis devoluciones de llamadas: **onCreate() ,onStart(), onResume(), onPause(), onStop() y onDestroy().** El sistema invoca cada una de estas devoluciones de llamada cuando una operación entra en un nuevo estado.

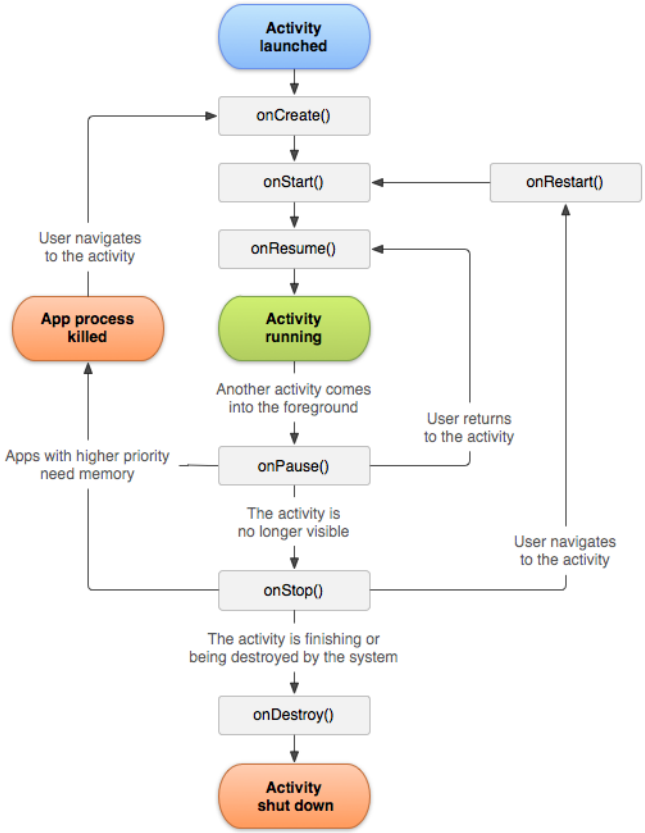

Figura 4.1: Ciclo de vida de una Actividad de un sistema Android

Cada uno de los estados que tiene la actividad es llamado en un momento concreto de la ejecución de la Actividad. Las características de cada uno de estos estados son las siguientes:

- **onCreate()** → Este método es el primero que llama cuando se crea la Actividad. Este estado es utilizado para ejecutar la lógica de la aplicación que debe ocurrir únicamente una vez en todo el ciclo de vida. Este método recibe el parámetro ***savedInstanceState***, que es un objeto de tipo ***Bundle*** que contiene el estado ya guardado de la actividad. Si la actividad nunca existió, el valor del objeto *Bundle* es nulo.

- **onStart()** → Cuando la actividad entra en el estado Started, el sistema

invoca esta devolución de llamada. La llamada *onStart()* hace que el usuario pueda ver la actividad mientras la app se prepara para que esta entre en primer plano y se convierta en interactiva. Por ejemplo, este método es donde la app inicializa el código que mantiene la IU.

- **onResume()** → La aplicación permanece en este estado hasta que ocurre algún evento que la quita de foco. Tal evento podría ser, por ejemplo, recibir una llamada telefónica, que el usuario navegue a otra actividad o que se apague la pantalla del dispositivo.

- **onPause()** → Este estado se utiliza cuando se ha perdido el foco de una aplicación. Sin embargo una actividad con el estado Paused puede ser completamente visible si está en el modo multiventana. En este estado no deben guardarse datos de la aplicación ya que es un estado que dura poco tiempo.

- **onStop()** → En este estado es donde los componentes del ciclo de vida pueden detener cualquier funcionalidad que no necesite ejecutarse mientras el componente no sea visible en la pantalla. Este estado debe usarse para liberar o ajustar recursos que no son necesarios mientras no sea visible para el usuario.

- **onDestroy()** → Se llama a este método antes de que se finalice la actividad. El sistema invoca esta devolución de llamada cuando el dispositivo rota o cuando la aplicación se cierra. En este estado es donde los componentes del ciclo de vida pueden recuperar cualquier elemento que se necesite antes de que finalice la Actividad.

4.1.2. Arquitectura de la aplicación Android

Para que la aplicación Android cumpla los requisitos expuestos en el apartado de especificación del proyecto, se han implementado 3 clases:

- **Controller** → Esta es la actividad principal. Desde esta actividad se recogen los datos de IP y puerto al que debe conectarse el dispositivo Android para ser utilizado como controlador.Al crearse la Actividad se lanza la ejecución de una hebra que se encargará de recibir, leer e interpretar las imágenes que lleguen por red una vez la aplicación se conecte al juego. Desde esta clase se controla la pulsación del usuario en la pantalla. De esta pulsación se guardan 3 datos: posición (x,y) donde se ha realizado la pulsación y el tipo de pulsación (presionar,levantar o arrastrar). Para cada una de estas pulsaciones se dispara la ejecución de hilo (**UdpClientThread**). La última función de esta clase es la de cambiar la imagen que se muestra en la aplicación.

- **UdpClientThread** → Esta clase tiene como función el envío de datos a la aplicación donde se ejecuta el juego. La primera vez que el usuario realice una pulsación se enviará el tamaño de la pantalla del dispositivo Android. Además de este mensaje, esta hebra se encarga del envío de paquetes que incluyen el tipo de pulsación que se ha realizado, la coordenada x y la coordenada y de la pantalla del dispositivo donde se ha realizado la pulsación. Una vez que la aplicación se cierre, el packete de cierre de conexión se envía desde esta hebra.

- **Receive_Image** → Esta clase se ejecuta desde una hebra distinta a la de la Actividad principal y la función que desempeña es la de recibir información que mande el juego. Esta clase se queda escuchando en un puerto designado, normalmente el mismo que abre el juego para recibir los datos de la aplicación Android. Los primeros mensajes en llegar son el tiempo de vibración del dispositivo cada vez que se realice una pulsación en la pantalla y posteriormente las imágenes que se envíen desde el juego. Estas imágenes se esperan en formato PNG ya que se realiza una descompresión de este formato.

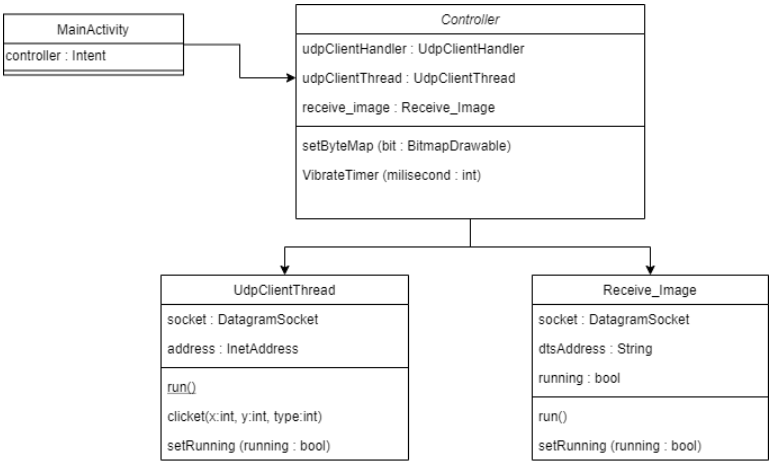

Figura 4.2: Arquitectura de la aplicación Android implementada

Se ha optado por una aplicación cerrada para que el desarrollador no tenga que implementar nueva funcionalidad en Android. El coste computacional de la aplicación es bajo, lo que permite ser utilizado en una gran cantidad de dispositivos. La división en 2 actividades es debido a que el flujo de Android se basa en ir cambiando entre actividades para que cada una tenga un uso específico. La primera actividad de usa para capturar un QR con la cámara y la segunda es utilizada para simular un mando.

4.2. Implementación de la aplicación de Unity

Como se comentó al principio de este capítulo, el motor de videojuegos elegido para la realización de este proyecto ha sido Unity. Unity es un motor de videojuegos multiplataforma creado por *Unity Technologies* en 2005. Unity está disponible como plataforma de desarrollo para Microsoft, Mac OS y Linux y tiene soporte de compilación con múltiples plataformas:

- **Web** → WebGL.
- **PC** → Windows, SteamOS, Linux, OS X y Windows Store Apps.
- **Dispositivos móviles** → iOS, Android, Windows Phone.
- **Smart TV** → tvOS, Samsung Smart TV, Android TV.
- **Consolas** → PlayStation Vita, PlayStation 4, Xbox 360, Xbox One, Wii U, Nintendo 3DS, Nintendo Switch.
- **Dispositivos de realidad virtual** → Oculus Rift, Google Cardboard, HTC Vive, PlayStation VR, Samsung Gear VR

Además de contar con soporte para múltiples dispositivos, Unity puede usarse junto con múltiples herramientas de desarrollo como **Blender**, **Autodesk 3ds Max** y **ZBrush** entre otros.

4.2.1. Funcionamiento de Unity

Unity es un motor de videojuegos que aglutina una gran variedad de herramientas para el desarrollo. Estas herramientas van desde inclusiones de **Scripts** para dar comportamientos específicos a cada una de las **Entidades** del juego hasta elementos más visuales como diagramas de estado para el control de las animaciones de un modelo. Para que todos estos sistemas tan diferentes puedan convivir, hay una serie de funciones que se ejecutan en un orden determinado. Unity a su vez se compone de varios elementos clave:

- **Escena** → Las escenas contienen los objetos del juego. Pueden usarse para crear niveles, menús o cualquier estado del juego.
- **GameObjects** / **Entidades** → Cada una de las escenas contiene objetos. Estos objetos se llaman GameObjects. Cualquier elemento es considerado un GameObject, no tiene por qué tener una representación visual (música, cámara, etc).
- **Componentes** → Los componentes son los diferentes atributos que se le dan a los GameObjects para que tengan funcionalidad (movimiento, posición, animación, colisión física, etc).

Unity ofrece una serie de componentes que dan una funcionalidad ya definida a un objeto, esta funcionalidad va desde tener una posición definida en el mundo hasta emitir un sonido y realizar una animación. Los desarrolladores pueden desarrollar sus propios componentes usando Scipts. Estos scripts indican a las diferentes entidades cómo comportarse. El lenguaje seleccionado para este sistema de *scripting* es C# y un script debe estar vinculado a una entidad para que este se ejecute. Al tratarse de un sistema basado en la ejecución e interacción entre scripts, Unity tiene una serie de funciones que se ejecutan automáticamente por el motor. Estas funciones son:

- **Awake** → Awake se invoca solo una vez al inicio de la ejecución. Si un GameObject está inactivo, entonces no podrá invocarse hasta que se le active. Sin embargo, Awake se invoca incluso si el GameObject está activo pero el componente no está habilitado. Se puede utilizar Awake para inicializar todas las variables a las que es necesario asignar un valor.
- **Start** → Al igual que Awake, Start se invocará si un GameObject está activo, pero solo si el componente está habilitado.
- **Update** → Update se invoca una vez por frame. En esta función se debe definir la lógica que se ejecuta continuamente, como animaciones, inteligencia artificial y otras partes del juego que tienen que actualizarse continuamente.
- **FixedUpdate** → Esta función es muy similar a Update pero su uso queda reservado para el desempeño de acciones en las que esté involucrada la física. Unity tiene un sistema basado en frames y FixedUpdate es la única función en la que se asegura que se va a ejecutar en un *frame-rate* fijo.
- **LateUpdate** → Esta es una función que es similar a Update, pero LateUpdate se invoca al final del frame. Unity analizará todos los objetos de juego, encontrará todas las Updates, e invocará las LateUpdates. Esto es bueno para entidades como la cámara.

4.2.2. Arquitectura de la API en Unity

Para que la aplicación desarrollada en Unity cumpla los requisitos expuestos en el apartado de especificación del proyecto, se han realizado 2 clases:

- **UDPSocket** → Esta clase se utiliza para la creación de todo lo necesario para hacer funcionar esta herramienta. Con el método **init()** se inician 2 hebras de ejecución diferentes. Una de ellas se encarga de enviar los datos necesarios al dispositivo de entrada. Estos datos son

tanto la vibración como la imagen a renderizar en el dispositivo. La otra se encarga de recibir los datos de entrada del dispositivo y avisar a los diferentes *listeners*. Estos listeners utilizan esa información para los propósitos desigandos por el desarrollador del juego (mover al personaje, pausar el juego, salir, etc). Esta clase también se encarga de cerrar la conexión.

- **InputMobileInterface** → Esta interfaz sirve para dar soporte a la conexión. **EndOfConnection()** debe ser utilizado para finalizar la conexión. **ReceiveTouch()** recibe las coordenadas donde se ha realizado la pulsación en el dispositivo de entrada. La misión de esta función es comprobar si estas coordenadas corresponden con la posición de un botón u objeto interactuable dentro del juego. Por último **ScreenSize()** recibe las dimensiones del dispositivo de entrada para posteriormente poder enviar imágenes y hacer las transformaciones de la posición de los botones.

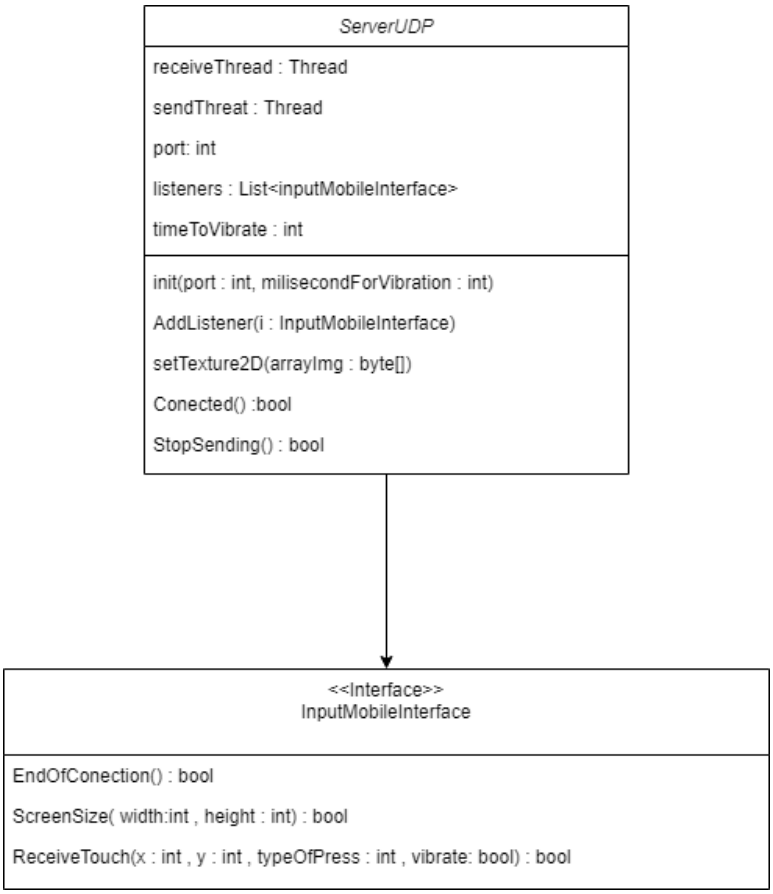

Figura 4.3: Arquitectura de la aplicación de Unity implementada

Con esta implementación se quiere dar al desarrollador una API que permita al usuario tener un nivel de abstracción superior al que tendría en caso de tener que manejar directamente la conexión entre 2 dispositivos y el tratamiento de input. Esto se ha conseguido manejando los hilos de manera individual y a la funcionalidad dada por la interfaz **InputMobileInterface**. La interfaz permite que el desarrollador decida en cada momento de la ejecución lo que se realiza en cada una de sus fases.

La primera fase consiste en el reescalado de las imágenes que se envían. En la segunda fase se realiza el tratamiento del input. En esta fase el desarrollador deberá decidir las coordenadas de los botones que quiere que tenga su mando y decidir qué acción se va a realizar con la pulsación de estos botones. La última fase involucra volver a lanzar el servidor en caso de que se pierda la conexión ya sea porque el dispositivo se apaga, porque se cierre la conexión de manera controlada o por inactividad.

Capítulo 5

Pruebas con usuarios

En las fases finales del desarrollo de la herramienta se llevó a cabo otro desarrollo en paralelo. Este desarrollo era la implementación de la herramienta desarrollada en un juego ya terminado y cerrado para realizar pruebas de rendimiento y usabilidad con usuarios. Los usuarios seleccionados no habían tenido contacto previo con la herramienta ni con el juego elegido. En este capítulo se recogen los objetivos de las pruebas, se analizarán los resultados obtenidos y se sacarán conclusiones al respecto.

5.1. Realización de Demo

Durante las fases finales de la implementación de la herramienta se vio la necesidad de integrar esta herramienta en un proyecto ya terminado en el que poder realizar pruebas de rendimiento, comprobar si era necesaria la implementación de más módulos que los descritos en la especificación de a herramienta y probarlo en diferentes configuraciones. Para poder realizar estas pruebas era necesario desarrollar un proyecto en paralelo donde poder probar o buscar uno ya terminado. Tras hacer una búsqueda de proyectos que pudieran aprovechar la herramienta se propuso la utilización de uno de los proyectos que ofrece Unity en su plataforma de aprendizaje ***Unity Learn***.[1] En esta plataforma se encuentran varios proyectos en los cuales pueden incluirse diferentes modificaciones explicadas en la propia plataforma para aprender a utilizar algunos aspectos de Unity. El proyecto escogido de la plataforma ha sido **Karting Microgame**[2], un juego de conducción arcade muy parecido a la saga de ***Mario Kart*** desarrollada por Nintendo. Se escogió este juego por su similitud a una saga qla cual ha jugado una gran cantidad de población en algún momento ya que cuenta con 17 juegos para una gran variedad de plataformas (iPhone, Android, Nintendo Switch,

[1] Unity Learn - https://learn.unity.com/projects

[2] Enlace de descarga en Asset Store - https://assetstore.unity.com/packages/templates/karting-microgame-150956?

Wii U, Nintendo 3DS, Wii, Nintendo DS, GameCube, Game Boy Advance y Nintendo 64).

Una vez se han incluido en el proyecto los scripts pertenecientes a la herramienta, se necesitaba una forma de conectar el dispositivo móvil al juego y para esto se implementó en la demo de Unity un generador de códigos QR utilizando la libreria **ZXingNet**[3]. Esta librería es un port del proyecto ZXing desarrollado en java para leer y generar códigos de barras. Con este código QR se envía a la aplicación móvil los datos de IP y puerto al que debe conectarse para poder ser usado como mando.

En el proyecto de Unity se debe añadir una nueva cámara para poder enviar a la aplicación Android la imagen del mando. Junto con esta imagen debe incluirse un nuevo script que defina la posición, el alto y el ancho del botón y la acción que se debe realizar cuando el jugador lo pulse. Con esto, cuando las pulsaciones del usuario lleguen al juego podrán ser tratadas como si fuesen teclas.

Para hacer el juego juegable tanto con el nuevo input como con el original, se ha añadido un pequeño menú al inicio del juego para elegir qué input utilizar. Para poder salir del juego de manera controlada también se ha añadido un botón para poder salir de la aplicación.

Para leer este QR desde la aplicación móvil se ha añadido una Activdad nueva que se ejecuta al inicio de la aplicación. Esta Actividad tiene como función utilizar la cámara del dispositivo Android para leer el QR y guardarse esos datos. Estos datos se mandan a la siguiente Actividad donde la aplicación los utilizará para iniciar la conexión y poder ser utilizada como dispositivo de entrada.

Figura 5.1: Inicio Demo Unity

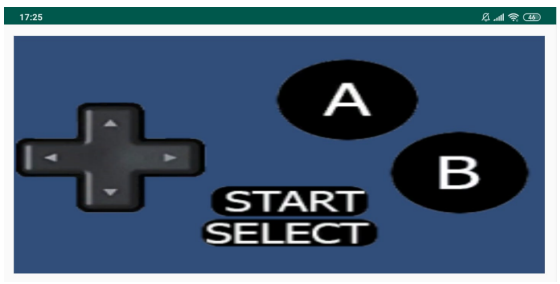

Figura 5.2: Mando Demo Android

5.2. Objetivos y organización de las pruebas

Previo a las pruebas con usuarios se definieron una serie de objetivos que cubrir durante la evaluación. Estos objetivos son los siguientes:

[3]ZXingNet - https://archive.codeplex.com/?p=zxingnet

- Comprobación del funcionamiento de ambas aplicaciones (Android y Unity) en diferentes configuraciones.
- Rendimiento de la parte de red, sobretodo en el envío de imágenes y el tiempo de envío de las pulsaciones.
- Valorar la intuitividad del uso de la herramienta.

Debido a la pandemia mundial que tuvo lugar durante la publicación de esta memoria debido al confinamiento por el virus SARS-COV-2, también conocido como *Coronavirus*, las pruebas de usuario han tenido que modificarse y adaptarse para ser realizadas de manera online en lugar de físicamente. Todas las pruebas se han realizado siguiendo las siguientes pautas:

- Se ha informado al usuario de los datos técnicos que se van a extraer de esta prueba (modelo de tarjeta gráfica, modelo de procesador, memoria RAM) y del posterior formulario a rellenar.
- Se ha subido el ejecutable y el APK a un repositorio público para que el usuario pueda hacer las pruebas.
- Se ha indicado al usuario que el ordenador y el móvil deben estar conectados a la misma red WIFI.
- Se ha utilizado la aplicación de **Discord** para realizar una llamada con el usuario y que este compartiese la pantalla donde se estaba ejecutando el juego.
- Se ha explicado al jugador que tiene que escanear el código QR que aparece en el juego con la aplicación que se ha descargado del repositorio.
- Se ha indicado al usuario que debe dar varias vueltas al circuito para que los datos puedan recogerse.
- Se ha realizado una entrevista con el usuario de entre 5 y 10 minutos para rellenar el formulario y comentar cualquier tipo de *feedback* sobre la herramienta y el juego.

Al final de la prueba se realiza una charla con el usuario donde este nos indica todas las observaciones, sugerencias de mejora y puntos positivos. En esta charla también se hacen algunas preguntas como por ejemplo el modelo de móvil con el que ha realizado la prueba, experiencia jugando a videojuegos y cuales juega normalmente, opinión sobre la fluidez de la prueba y el *feedback* recibido (visual y háptico gracias a la vibración).

5.3. Resultados de las pruebas

En total, este experimento ha contado con 5 participantes que se han ofrecido voluntariamente a probar la herramienta. A pesar de contar con un número muy limitado de usuarios se han obtenido opiniones variadas. Entre los usuarios se encuentran tanto jugadores habituales de videojuegos como usuarios que no juegan. Los resultados individuales de las pruebas son:

1. **Usuario 1**

   Este usuario tiene 22 años y es un jugador habitual. Suele jugar en PC a juegos como World of Warcraft, Crossfire, Monster Hunter y Starcraft 2. Las pruebas se han realizado con los siguientes dispositivos:

Tarjeta Gráfica	**Procesador**	**RAM**	**Modelo Móvil**
NVIDIA GeForce GTX 980	Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz	32594 MB	One plus 6

   Se han recogido los diferentes tiempos que tarda Unity en convertir la imagen que se optiene de la cámara a formato PNG, enviarse por red al dispositivo Android y el tiempo que este tarda en descomprimir el PNG y mostrar la imagen en la pantalla.

   Este usuario comentó durante la prueba que la vibración era adecuada a la hora de pulsar los botones. Señaló también que los botones le parecian muy pequeños y que necesitarían ser más grandes ya que hay veces que no se pulsan bien. En cuanto a si la experiencia fue fluida, el usuario contestó un 7 sobre 8 siendo 1 - Nada fluida y 8 - Completamente fluida. Cabe destacar que la latencia de red en el momento de la prueba era de 2 milisegundos.

   Con los dispositivos utilizados se obtiene una media en milisegundos de descompresión de PNG de XX y de transformación de textura de Unity a PNG de XX. La moda en la descompresión del PNG es de 2 milisegundos y la moda en el tiempo de conversión de textura a PNG es de 5 milisegundos. Sumado a esto tenemos los 2 milisegundos de latencia por lo que el proceso completo estaría en los 9 milisegundos en la mayoría de los casos. El umbral en el que el ojo humano detecta un cambio en las imágenes es de 14 milisegundos por lo que al encontrarse por debajo podemos asegurar que la fluidez durante la sesión fue la óptima.

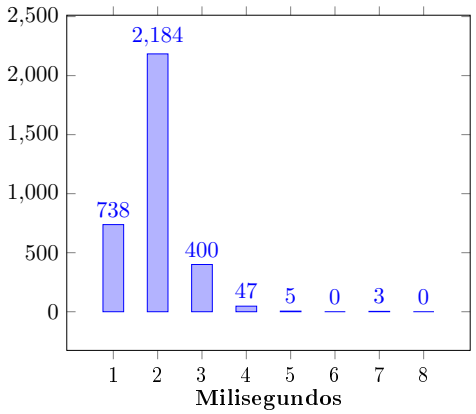

Figura 5.3: Tiempo de descompresión PNG en dispositivo Android Usuario 1

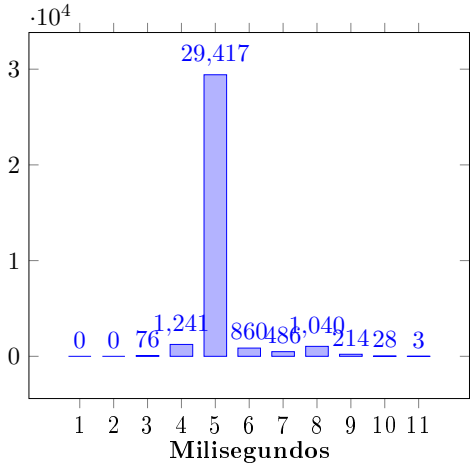

Figura 5.4: Tiempo de conversión de cárama de Unity a PNG Usuario 1

2. **Usuario 2**

   Este usuario tiene 23 años y es un jugador habitual. Suele jugar a juegos como Call Of Duty: MW, Dragalia Lost y Mario Kart 8. Las pruebas se han realizado con los siguientes dispositivos:

Tarjeta Gráfica	**Procesador**	**RAM**	**Modelo Móvil**
NVIDIA GeForce GTX 1080	Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz	16327 MB	One plus 6

   Se han recogido los diferentes tiempos que tarda Unity en convertir la

imagen que se optiene de la cámara a formato PNG, enviarse por red al dispositivo Android y el tiempo que este tarda en descomprimir el PNG y mostrar la imagen en la pantalla.

Este usuario comentó durante la prueba que la vibración durante los derrapes era intermitente en vez de mantenida. Esto le desagradó. En cuanto a si la experiencia fue fluida, el usuario contestó un 8 sobre 8 siendo 1 - Nada fluida y 8 - Completamente fluida. Cabe destacar que la latencia de red en el momento de la prueba era de 3 milisegundos.

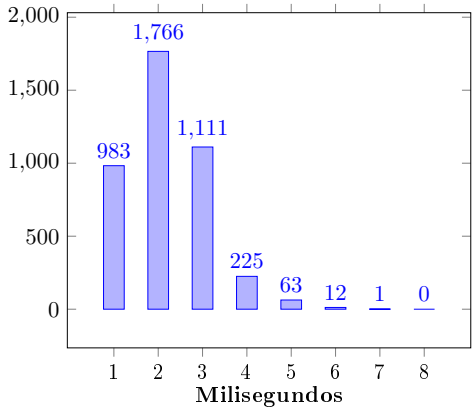

Figura 5.5: Tiempo de descompresión PNG en dispositivo Android Usuario 2

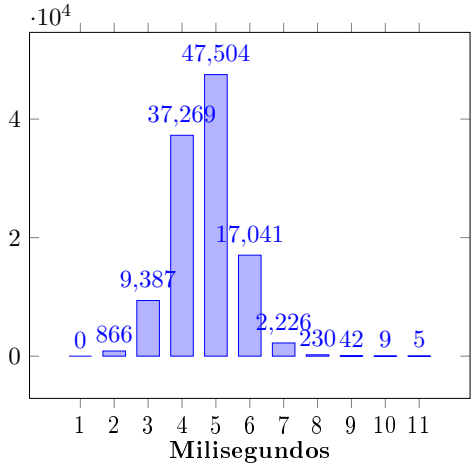

Figura 5.6: Tiempo de conversión de cárama de Unity a PNG Usuario 2

Con los dispositivos utilizados se obtiene una media en milisegundos de descompresión de PNG de XX y de transformación de textura de Unity a PNG de XX. La moda en la descompresión del PNG es de 2-3 milisegundos y la moda en el tiempo de conversión de textura a PNG

es de 4 y 5 milisegundos la mayor parte de las veces. Sumado a esto tenemos los 3 milisegundos de latencia por lo que el proceso completo estaría entre 9-11 milisegundos en la mayoría de los casos. El umbral en el que el ojo humano detecta un cambio en las imágenes es de 14 milisegundos por lo que al encontrarse por debajo podemos asegurar que la fluidez durante la sesión fue la óptima.

3. **Usuario 3**

   Este usuario tiene 49 años y es un jugador casual. Suele jugar a juegos como Age of Empires, Destiny, ARK y La Batalla por la Tierra Miedia. Las pruebas se han realizado con los siguientes dispositivos:

Tarjeta Gráfica	**Procesador**	**RAM**	**Modelo Móvil**
NVIDIA GeForce GTX 960M	Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz	16289 MB	Samsung Galaxy S9+

   Se han recogido los diferentes tiempos que tarda Unity en convertir la imagen que se optiene de la cámara a formato PNG, enviarse por red al dispositivo Android y el tiempo que este tarda en descomprimir el PNG y mostrar la imagen en la pantalla.

   Este usuario comentó durante la prueba que la vibración era demasiado fuerte por lo que debería de disminuir la intensidad. Esto le desagradó. En cuanto a si la experiencia fue fluida, el usuario contestó un 5 sobre 8 siendo 1 - Nada fluida y 8 - Completamente fluida. Cabe destacar que la latencia de red en el momento de la prueba era de 2 milisegundos.

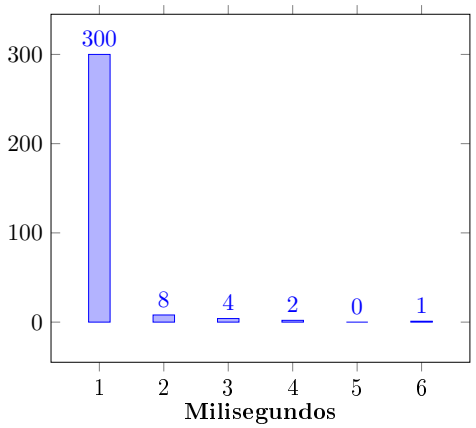

Figura 5.7: Tiempo de descompresión PNG en dispositivo Android Usuario 3

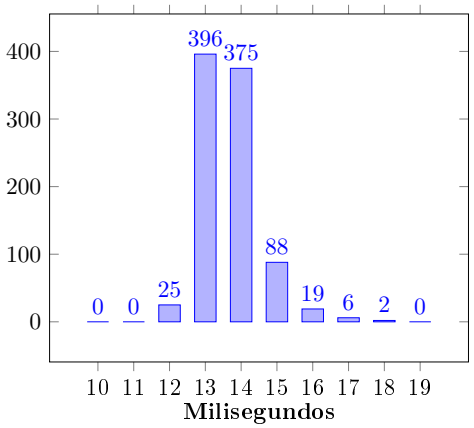

Figura 5.8: Tiempo de conversión de cárama de Unity a PNG Usuario 3

Con los dispositivos utilizados se obtiene una media en milisegundos de descompresión de PNG de XX y de transformación de textura de Unity a PNG de XX. La moda en la descompresión del PNG es de 1 milisegundos y la moda en el tiempo de conversión de textura a PNG es de 13 y 14 milisegundos la mayor parte de las veces. Sumado a esto tenemos los 2 milisegundos de latencia por lo que el proceso completo estaría entre 16 y 17 milisegundos en la mayoría de los casos. El umbral en el que el ojo humano detecta un cambio en las imágenes es de 14 milisegundos por lo que al encontrarse por encima la fluidez no ha sido la óptima.

4. **Usuario 4**

Este usuario tiene 51 años y no juega a videojuegos salvo en momentos muy concretos. Los juegos a los que ha jugado en algún momento han sido Mario Kart de Wii y Scrabble para Android. Las pruebas se han realizado con los siguientes dispositivos:

Tarjeta Gráfica	**Procesador**	**RAM**	**Modelo Móvil**
AMD Radeon HD 7660D	AMD A10-5800K APU with Radeon(tm) HD Graphics	7367 MB	Samsung Galaxy S8

Se han recogido los diferentes tiempos que tarda Unity en convertir la imagen que se optiene de la cámara a formato PNG, enviarse por red al dispositivo Android y el tiempo que este tarda en descomprimir el PNG y mostrar la imagen en la pantalla.

Este usuario comentó durante la prueba que los botones no se correspondían por completo con los mostrados en la imagen, tenía que pulsar un poco fuera del botón para que este se detectase. En cuanto a si la experiencia fue fluida, el usuario contestó un 7 sobre 8 siendo 1 - Nada fluida y 8 - Completamente fluida. Cabe destacar que la latencia de red en el momento de la prueba era de 2 milisegundos.

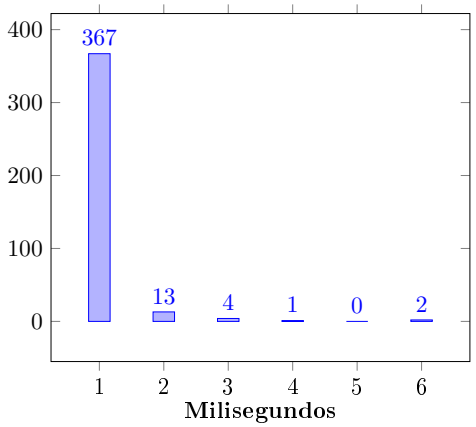

Figura 5.9: Tiempo de descompresión PNG en dispositivo Android Usuario 4

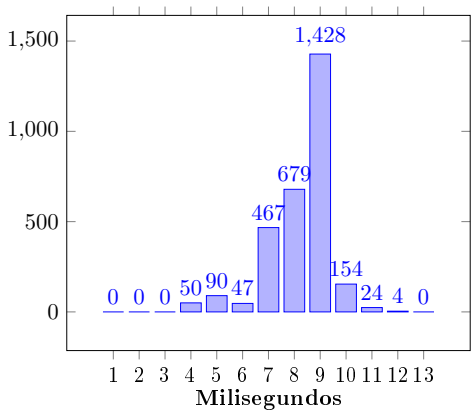

Figura 5.10: Tiempo de conversión de cárama de Unity a PNG Usuario 4

Con los dispositivos utilizados se obtiene una media en milisegundos de descompresión de PNG de XX y de transformación de textura de Unity a PNG de XX. La moda en la descompresión del PNG es de 1 milisegundos y la moda en el tiempo de conversión de textura a PNG es de 7-9 milisegundos la mayor parte de las veces. Sumado a esto tenemos los 2 milisegundos de latencia por lo que el proceso completo estaría entre 10-12 milisegundos en la mayoría de los casos. El umbral

en el que el ojo humano detecta un cambio en las imágenes es de 14 milisegundos por lo que al encontrarse por debajo podemos asegurar que la fluidez durante la sesión fue la óptima.

5. **Usuario 5**

   Este usuario tiene 19 años y es un jugador habitual de juegos en todas las plataformas actuales (PC, PS4, Switch y Android). Los juegos a los que suele jugar son World Of Warcraft, The Legend of Zelda, Mario Kart, Total War y Assassin's Creed. Las pruebas se han realizado con los siguientes dispositivos:

Tarjeta Gráfica	**Procesador**	**RAM**	**Modelo Móvil**
NVIDIA GeForce GTX 1080	Intel(R) Core(TM) i7-6800K CPU @ 3.40GHz	32668 MB	Samsung Galaxy S9+

   Se han recogido los diferentes tiempos que tarda Unity en convertir la imagen que se optiene de la cámara a formato PNG, enviarse por red al dispositivo Android y el tiempo que este tarda en descomprimir el PNG y mostrar la imagen en la pantalla.

   Este usuario comentó durante la prueba que los botones no se correspondían por completo con los mostrados en la imagen, tenía que pulsar un poco fuera del botón para que este se detectase. En cuanto a si la experiencia fue fluida, el usuario contestó un 7 sobre 8 siendo 1 - Nada fluida y 8 - Completamente fluida. Cabe destacar que la latencia de red en el momento de la prueba era de 1 milisegundos.

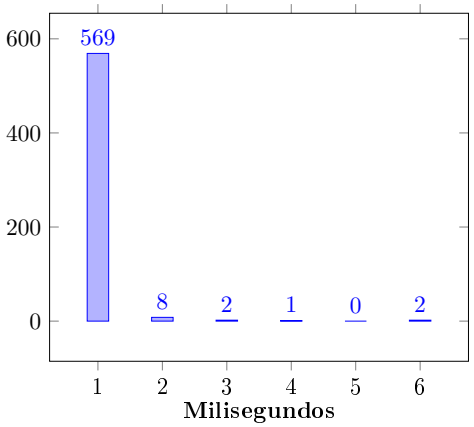

Figura 5.11: Tiempo de descompresión PNG en dispositivo Android Usuario 5

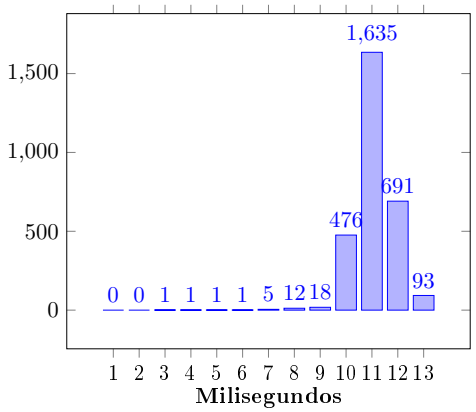

Figura 5.12: Tiempo de conversión de cárama de Unity a PNG Usuario 5

Con los dispositivos utilizados se obtiene una media en milisegundos de descompresión de PNG de XX y de transformación de textura de Unity a PNG de XX. La moda en la descompresión del PNG es de 1 milisegundos y la moda en el tiempo de conversión de textura a PNG es de 10-12 milisegundos la mayor parte de las veces. Sumado a esto tenemos los 1 milisegundos de latencia por lo que el proceso completo estaría entre 12-14 milisegundos en la mayoría de los casos. El umbral en el que el ojo humano detecta un cambio en las imágenes es de 14 milisegundos por lo que al encontrarse por debajo podemos asegurar que la fluidez durante la sesión fue la óptima.

Capítulo 6

Conclusiones

El objetivo del proyecto es diseñar e implementar una herramienta con la que poder utilizar un dispositivo móvil como dispositivo de entrada para videojuegos. Para probar que esta herramienta funciona, se ha implementado la herramienta desarrollada en un proyecto ya finalizado de Unity. La integración de la herramienta en esta demo ha requerido de la implementación de scripts auxiliares y modificaciones específicas del proyecto desarrollado por Unity. Las pruebas realizadas con usuarios muestran unos datos positivos ya que todas las ejecuciones han podido realizarse sin inconvenientes a pesar de la situación actual. Esto demuestra que la herramienta se comporta de manera favorable en entornos diferentes y con configuraciones no preestablecidas. Esta situación ha reflejado que existe un problema cuando las dimensiones de los dispositivos Android no son las mismas que las que se pensó inicialmente.

Como puede verse en los resultados de las pruebas, gran parte de la fluidez depende del procesador del ordenador donde se ejecute el juego. Esto indica que el proceso de compresión de imagen es muy lento en algunas de las pruebas realizadas. La latencia de red y la velocidad de descompresión de la imagen en Android son muy óptimas debido a que los dispositivos móviles utilizados en las pruebas eran de gama media-alta.

6.1. Trabajo Futuro

Como se ha comprobado, el proyecto y la herramienta cumplen los objetivos definidos en el capítulo 1 y 3. Sin embargo, debido a los problemas para realizar las pruebas con usuarios en un entorno controlado se han descubierto una serie de fallos. A continuación se proporcionan una serie de tareas a realizar en una posible revisión de la herramienta:

- Mejorar los tiempos de descompresión de imagen en sistemas móviles de gamas más bajas.

- Mejorar los tiempos de compresión de textura a PNG en ordenadores con procesadores de gamas bajas.
- Eliminar la restricción que obliga a ambos dispositivos a estar conectados a la misma red.
- Tener en cuenta las dimensiones del dispositivo móvil para modificar la posición de los botones.
- Añadir una lista de posibilidades para elegir con qué mando se desea jugar.