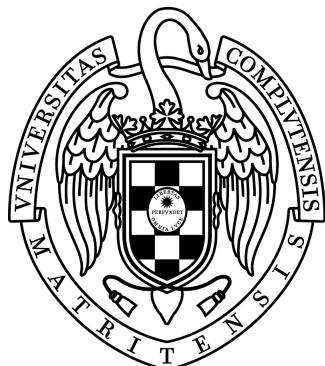


---

# 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

Enero 2021

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 0.1.8*

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

**Enero 2021**

Copyright © Pablo Gómez Calvo y Sergio J. Higuera Velasco

*A todo aquél 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

<b>Agradecimientos</b>	<b>VII</b>
<b>Resumen</b>	<b>IX</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.2. Objetivos . . . . .	3
1.3. Metodología . . . . .	4
1.4. Planificación . . . . .	5
1.5. Estructura del documento . . . . .	6
<b>2. Estado del arte en entrada de usuario para videojuegos</b>	<b>7</b>
2.1. Dispositivos de entrada en ordenadores de propósito general . . . . .	7
2.1.1. Evolución de los teclados . . . . .	7
2.1.2. Evolución de los ratones . . . . .	8
2.2. Evolución de los gamepads a través de las generaciones de consolas . . . . .	10
2.2.1. Primera generación (1972-1976) . . . . .	11
2.2.2. Segunda generación (1976-1983) . . . . .	11
2.2.3. Tercera generación (1983-1987) . . . . .	12
2.2.4. Cuarta generación (1987-1993) . . . . .	12
2.2.5. Quinta generación (1993-1998) . . . . .	14
2.2.6. Sexta generación (1998-2005) . . . . .	14
2.2.7. Septa generación (2005-2012) . . . . .	15
2.2.8. Octava generación (2012-2020) . . . . .	16
2.2.9. Novena generación (2020-Actual) . . . . .	18
2.3. <i>Feedback</i> en los controladores . . . . .	18
2.4. Sistemas de <i>streaming</i> en videojuegos . . . . .	19
<b>3. Especificación del protocolo de comunicación</b>	<b>23</b>
3.1. Arquitectura del proyecto a alto nivel . . . . .	23
3.2. Protocolo de comunicación entre juego y dispositivo de entrada	26

<b>4. Implementación de las aplicaciones</b>	<b>29</b>
4.1. Implementación de la aplicación de Android . . . . .	29
4.1.1. Conceptos básicos de Android . . . . .	30
4.1.2. Desarrollo de la aplicación Android . . . . .	32
4.2. Implementación de la aplicación de Unity . . . . .	35
4.2.1. Funcionamiento de Unity . . . . .	35
4.2.2. Desarrollo de la API en Unity . . . . .	36
<b>5. Pruebas con usuarios</b>	<b>39</b>
5.1. Realización de Demo . . . . .	39
5.2. Objetivos y organización de las pruebas . . . . .	40
5.3. Resultados de las pruebas . . . . .	42
<b>6. Conclusiones</b>	<b>51</b>
6.1. Trabajo Futuro . . . . .	51

# Índice de figuras

2.1. Primer ratón desarrollado por Douglas Engelbart y Bill English . . . . .	9
2.2. Ratón del Xerox Alto de los años 70 . . . . .	9
2.3. Joystick utilizado para jugar a Spacewar! . . . . .	10
2.4. Dispositivos de entrada relevantes en la 1. <sup>a</sup> generación de consolas . . . . .	11
2.5. Dispositivos de entrada relevantes en la 2. <sup>a</sup> generación de consolas . . . . .	12
2.6. Dispositivos de entrada relevantes en la 3. <sup>a</sup> generación de consolas . . . . .	13
2.7. Dispositivos de entrada relevantes en la 4. <sup>a</sup> generación de consolas . . . . .	13
2.8. Dispositivos de entrada relevantes en la 5. <sup>a</sup> generación de consolas . . . . .	14
2.9. Dispositivos de entrada relevantes en la 6. <sup>a</sup> generación de consolas . . . . .	15
3.1. Arquitectura del proyecto . . . . .	24
3.2. Diagrama del protocolo de comunicación entre ambos dispositivos . . . . .	28
4.1. Estados de las aplicaciones Android . . . . .	30
4.2. Ciclo de vida de una Actividad de un sistema Android . . . . .	31
4.3. Diagrama de clases de la aplicación Android . . . . .	34
4.4. Diagrama de clases de la aplicación de Unity . . . . .	37
5.1. Inicio Demo Unity . . . . .	40
5.2. Mando Demo Android . . . . .	40
5.3. Tiempo de descompresión PNG en dispositivo Android Usuario 1 . . . . .	43
5.4. Tiempo de conversión de cámara de Unity a PNG Usuario 1 . . . . .	43
5.5. Tiempo de descompresión PNG en dispositivo Android Usuario 2 . . . . .	44

---

5.6. Tiempo de conversión de cámara de Unity a PNG Usuario 2 . . . . .	44
5.7. Tiempo de descompresión PNG en dispositivo Android Usuario 3 . . . . .	45
5.8. Tiempo de conversión de cámara de Unity a PNG Usuario 3 . . . . .	46
5.9. Tiempo de descompresión PNG en dispositivo Android Usuario 4 . . . . .	47
5.10. Tiempo de conversión de cámara de Unity a PNG Usuario 4 . . . . .	47
5.11. Tiempo de descompresión PNG en dispositivo Android Usuario 5 . . . . .	48
5.12. Tiempo de conversión de cámara de Unity a PNG Usuario 5 . . . . .	49

# Índice de Tablas

2.1. Ancho de banda necesario para jugar a Stadia vs GeForce Now	20
3.1. Envío de la resolución del móvil al juego.	26
3.2. Tiempo de vibración del dispositivo móvil en milisegundos	27
3.3. Pulsación enviada desde el dispositivo móvil al ejecutor del juego	27



# Capítulo 1

## Introducción

La tecnología se ha vuelto indispensable en prácticamente todos los ámbitos de nuestra sociedad. Son pocos los escenarios en los que no se vea involucrado un aparato tecnológico. Tareas tan cotidianas como preparar un café por la mañana o levantar una persiana ahora son posibles con una aplicación en nuestro móvil o con un comando por voz.

Además de para facilitar nuestras tareas, los dispositivos móviles se han unido a los ordenadores y las consolas de videojuegos en la tarea de entretener a una gran cantidad de usuarios. Con la progresión en la calidad de los dispositivos móviles, la industria del videojuego ha decidido unirse y comenzar a lanzar juegos con grandes presupuestos al mercado móvil.

Como veremos, la entrada de los dispositivos móviles en la industria del entretenimiento no ha sido solo en forma de plataforma de juego sino como un aliado de los dispositivos ya existentes. Los dispositivos móviles han ayudado a la industria del entretenimiento a ampliar el abanico de opciones que los usuarios tienen para jugar e interactuar con las consolas actuales.

Con este proyecto se pretende explorar la situación actual del uso de dispositivos móviles como dispositivo de entrada para videojuegos ejecutados en otra plataforma y las motivaciones que existen para invertir e investigar en este nuevo modelo de interacción de usuario. Junto con esta investigación, se llevará a cabo con un proceso de desarrollo de una herramienta que haga posible jugar utilizando un dispositivo móvil como dispositivo de entrada para videojuegos.

### 1.1. Motivación

Uno de los pilares que siempre ha caracterizado a la industria del videojuego es la innovación a la hora de crear nuevas experiencias para los

usuarios. Son experiencias que enriquecen a muchos jugadores y que cada vez se disfrutan de una manera más cómoda y flexible. Uno de los culpables del aumento de esta flexibilidad a la hora de jugar son los dispositivos móviles.

En esta última década el mercado del *gaming* ha acogido al teléfono móvil como su nuevo integrante. Los juegos para móviles han tenido mucho éxito entre las nuevas generaciones de jugadores. Tal y como se muestra en el informe publicado por AEVI (2019), el 45 % de los ingresos de la industria del videojuegos en el año 2019 provino de los videojuegos móviles donde se incluyen tanto dispositivos móviles como tablets. Estapé (2018a) plantea si estos nuevos dispositivos móviles son una amenaza para las consolas actuales y presenta el caso de varios fabricantes de periféricos *gaming* que se han lanzado al mundo de la fabricación de **smartphones** orientados a jugar.

Títulos relevantes dentro de la industria como *League of Legends*, *Call of Duty* o *Fortnite* ya tienen su versión para dispositivos móviles. Gracias a que estos títulos son gratuitos y se encuentran a la cabeza de los *e-sports* en sus modalidades de PC y consola (TEO, 2020), su relevancia dentro de las plataformas móviles ha sido aun mayor. A pesar de esto, tal y como cuenta Sanmartín (2019) en un artículo, existen varios limitantes en el *gaming* para dispositivos móviles. Algunos de estos limitantes son la batería de los dispositivos y la necesidad de una conexión estable a internet.

Para resolver estos inconvenientes, las principales desarrolladoras de dispositivos móviles comenzaron a fabricar las gamas altas de estos dispositivos que suplían los problemas de batería, calentamiento y frecuencia de refresco de las pantallas. Sony en particular ha apostado por el desarrollo de una serie de dispositivos móviles pensados para largas sesiones de juego<sup>1</sup>. Además de esto, Sony ha desarrollado su aplicación para poder jugar de manera remota a PlayStation 4 y PlayStation 5, **PS Remote Play**<sup>2</sup>. Además de para poder jugar, Sony ha desarrollado otra aplicación que permite a los usuarios controlar la interfaz de la consola PlayStation 4 desde su dispositivo móvil haciendo que este simule ser un mando y una segunda pantalla, **PS4 Second Screen**<sup>3</sup>.

Nintendo por su parte ha desarrollado la aplicación **Nintendo Switch Online**<sup>4</sup> para llevar a cabo la comunicación en sus juegos online. Esta apli-

---

<sup>1</sup>Xperia - <https://www.sony.es/electronics/xperia-mobile-gaming>

<sup>2</sup><https://remoteplay.dl.playstation.net/remoteplay/lang/es/index.html>

<sup>3</sup><https://play.google.com/store/apps/details?id=com.playstation.mobile2ndscreen&hl=es&gl=US>

<sup>4</sup><https://www.nintendo.es/Familia-Nintendo-Switch/Nintendo-Switch-Online/Aplicacion-para-moviles-1374628.html>

cación convierte tu dispositivo móvil en un chat de voz y texto, lo que suple la falta de micrófono incorporado en la consola y permite a los jugadores comunicarse con sus compañeros en los juegos online. Microsoft ha desarrollado su aplicación **Xbox**<sup>5</sup> con la que poder jugar de manera remota a su consola. Gracias a esta aplicación, el usuario puede descargarse los juegos en su Xbox, ejecutarlos y conectar su móvil o tablet para poder jugar directamente en su smartphone a través de internet.

Después de desarrollar sus aplicaciones, Sony sacó al mercado una serie de juegos conocidos como **PlayLink**<sup>6</sup>. Este nuevo tipo de juegos se concentran en una colección de títulos que tienen como característica común que no es necesario usar un mando convencional de PlayStation. Estos títulos se juegan directamente usando el teléfono móvil como mando y el único requisito es tener cada uno de los dispositivos móviles conectados a la consola vía Wi-Fi. Esto soluciona por completo el problema de la escasez de mandos de consola en los hogares ya que únicamente serán necesarios los teléfonos móviles de las personas que vayan a jugar.

El propósito de este trabajo consiste en lograr utilizar un móvil como dispositivo de entrada en un juego de PC, imitando las características de algunas de las aplicaciones anteriormente mencionadas. Para poder lograrlo, se propone crear una librería de uso libre para el motor de videojuegos Unity. Esto permitirá la posibilidad de ampliación y modificación de la librería dependiendo de las necesidades de cada desarrollo.

## 1.2. Objetivos

La finalidad del proyecto es la conexión entre 2 dispositivos, uno de ellos que ejecuta el juego y el otro funciona como un dispositivo de entrada / mando para controlar el videojuego. Esta conexión debe ser estable, con el mínimo retraso posible y que sea fácil de incorporar en proyectos ya desarrollados.

El dispositivo móvil tiene la función de ser el dispositivo de entrada del videojuego. Para lograr esto, en la pantalla del móvil se muestra un mando virtual. Una vez el usuario interactúe con este mando virtual, las pulsaciones de los botones que se encuentran en la pantalla se envían al juego a modo de entrada de usuario. Además de esto, el mando virtual que se muestra puede configurarse desde el juego enviándole al dispositivo móvil la imagen del mando que mostrar.

---

<sup>5</sup><https://www.xbox.com/es-ES/consoles/remote-play>

<sup>6</sup><https://www.playstation.com/es-es/accessories/playlink/>

Con respecto a las plataformas donde realizar ambas aplicaciones, hemos decidido utilizar Android de manera nativa para el dispositivo de entrada y Unity para la parte del juego. Se ha decidido hacer el plug-in para Unity ya que es uno de los motores de juegos más utilizado actualmente. En España el 83 % de las empresas de videojuegos utilizan este motor para desarrollar sus juegos (Libro blanco del desarrollo español de videojuegos 2019). El uso de Android para la aplicación de móvil se debe a las facilidades que ofrece Google para subir aplicaciones a la plataforma Play Store y al amplio parque de dispositivos Android que existen actualmente.

Para conseguir este objetivo se ha dividido el objetivo final en los siguientes pasos:

- 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 a través de una serie de pruebas con usuarios.

Para el último punto, se va a realizar un juego simple que pueda ser controlado con un dispositivo móvil para probar que la librería desarrollada funciona. Una vez el juego esté terminado, se va a realizar una posterior prueba con usuarios para obtener una serie de datos como latencia de red, uso del procesador y tiempos de compresión y descompresión de imágenes. Los datos recolectados nos permitirán averiguar si el sistema desarrollado es apto para uso en juegos comerciales y futuros desarrollos.

### 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 (1995) es un framework para la gestión de proyectos de trabajo en equipo. El texto original proviene de un congreso (OOPSLA 1995) pero en él no se incluyen muchos de los términos que hoy se asocian con la metodología SCRUM como las reuniones diarias (Mei, 2020). Scrum introduce el término *sprint* para referirse a períodos de tiempo de entre 2 y 4 semanas de duración en la que el equipo

de desarrollo se compromete a realizar una serie de tareas. Estas tareas no deben ser cambiadas durante la duración del sprint y el progreso de estas tareas debe ser compartido en las diferentes reuniones de scrum diarias. Una vez finalizado el sprint, debe de planificarse el siguiente con el objetivo de mejorar el producto ya existente. Clinton Keith explica cómo introducir esta metodología en cada uno de los equipos de desarrollo que configuran un estudio de videojuegos. Actualmente los videojuegos son productos que tardan varios años en desarrollarse e involucran grandes presupuestos, es por esto que revisar el producto sobre el que se está trabajando cada poco tiempo ayuda a las diferentes partes del equipo a tener una visión más global del videojuego (Keith, 2010).

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 incluían a los directores del TFG en las que se discutían las diferentes características 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 realizaron 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 eliminadas 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. En este capítulo se mencionan y explican los cambios que han sufrido los diferentes dispositivos de entrada para videojuegos desde los inicios. En este capítulo se incluye también el funcionamiento de los sistemas de streaming y la retroalimentación en los controladores de videojuegos.

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.

El capítulo 5 se explica la demo desarrollada para probar la viabilidad del *plug-in* desarrollado y como caso práctico para la presentación de este proyecto. Además, se 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. Dispositivos de entrada en ordenadores de propósito general

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. Beekman (1999) expone que los más comunes de estos dispositivos de entrada son el teclado y ratón. Pero no existen únicamente estos 2 dispositivos de entrada, a lo largo de la historia de la informática se han ido desarrollando diversos dispositivos de entrada tanto sonora como visual y de movimiento mecánico.

#### 2.1.1. Evolución de los teclados

La historia de los teclados actuales tiene su origen en las máquinas de escribir. Estas primeras máquinas de escribir tienen su origen en el año 1877, cuando la empresa Remington comenzó a comercializar de manera masiva

las máquinas de escribir. En un primer momento las teclas se dispusieron en orden alfabético, algo que cambiaría un año después con la aparición de la primera patente del teclado QWERTY. Tal y como señala Stamp (2013) en su artículo, esta primera versión de las máquinas de escribir tenían un defecto que fue notorio cuando los usuarios escribían rápidamente una sucesión de letras que se encontraban cerca en el teclado. Este defecto consistía en que las barras que conectaban cada una de las teclas chocaban si se pulsaban teclas que estuvieran demasiado cerca. Para evitar este fallo en el modelo inicial se desarrolló el sistema QWERTY, el cual distancia los pares de letras que se suelen escribir juntas.

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. Este dispositivo fue muy popular durante el siglo XX y sus funciones eran las de enviar y recibir mensajes mecanografiados de un punto a otro a través de un canal de comunicación. Más adelante fue utilizado en conjunto con las cintas perforadas para almacenar datos en los primeros ordenadores. 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. Además confirmó lo útil y conveniente que puede ser un dispositivo de entrada como el teclado.

Actualmente las principales mejoras que han sufrido los teclados de ordenador se basan en la eliminación de cables gracias al Bluetooth, la disminución de la presión que hay que ejercer sobre la tecla para que esta sea detectada y la ergonomía. 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.

### **2.1.2. Evolución de los ratones**

Además del teclado, el segundo dispositivo de entrada por excelencia es el ratón. La primera maqueta [2.1] 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 y disponía de un botón en la parte superior con el que se permitía hacer clic en la posición en la que se encontraba el cursor.

El siguiente avance del dispositivo fue cambiar su carcasa de madera por una de plástico y añadir más botones. Tal y como describió Entrialgo (2020)



Figura 2.1: Primer ratón desarrollado por Douglas Engelbart y Bill English

en su artículo, este avance suele ser atribuido a Microsoft o a Apple. Lejos de ser así, fue la empresa **Xerox** la que realizó el nuevo diseño del ratón [2.2] y del que es considerado el primer ordenador personal de la historia junto al Altair 8080. Este nuevo dispositivo sustituía las 2 ruedas que marcaban la posición del cursor por una única bola de metal. La posición relativa de esta bola era la que determinaba la posición del cursor en la pantalla.



Figura 2.2: Ratón del Xerox Alto de los años 70

En el año 1992 Microsoft decidió vender en un mismo paquete su última versión de MS-DOS y Windows 3.1, lo que hizo que el ratón pasase a ser un periférico fundamental ya que se dejaba atrás el mundo del texto y se abrían a todos los públicos las interfaces gráficas del PC. Este cambio hizo que el ratón siguiera evolucionando y durante la década de los 90 e inicios del 2000 los ratones sufrieron algunas mejoras. Algunas de estas mejoras son: 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. Un sector que aprovechó mucho esta estandarización del ratón y de las interfaces gráficas fue el de los videojuegos. El ratón se ha convertido en una parte esencial del mundo de los videojuegos de PC, 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.

## 2.2. Evolución de los gamepads a través de las generaciones de consolas

Los videojuegos de PC fueron, son y serán muy importantes para el desarrollo de la industria pero las precursoras de este existo son sin dudas las máquinas recreativas. Se dice que el primer intento de videojuego es una patente de 1947 sobre la simulación del lanzamiento de misiles pero no fue hasta 1958 y la salida del famoso “Tenis para Dos” creado por William Higginbotham que se puede comenzar a hablar de videojuegos (Estapé, 2018b). En este juego se recreaba una pista de tenis en la que la pelota iba de un lado a otro de la pista. Poco después, en 1962 vio la luz **Spacewar!**. Este título fue desarrollado por Steve Russell y posteriormente modificado por sus estudiantes. En este juego lo que se quería era simular una pelea entre 2 naves en el espacio, fue el primer paso de los juegos multijugador en un mismo dispositivo. El dispositivo de entrada de este juego consistía de 2 ejes de rotación que se usaban para controlar la rotación y el empuje de la nave. Además de esto, el mando disponía de un botón para disparar misiles [2.3].

Será en la década de los 70 cuando las máquinas recreativas comiencen a tener repercusión con la salida de **Pong** en 1972 y **Space Invaders** en 1978. Muchos títulos y sagas tienen su inicio en esta época en la que las salas de recreativas estaban abarrotadas de jóvenes dispuestos a pasar allí todas las horas posibles jugando y compitiendo con otros jugadores. La salida de las consolas, las rápidas mejoras del hardware y el avance la tecnología hizo que las máquinas recreativas se quedasen obsoletas en poco tiempo. Además de esto la proliferación de los cibercafés y los juegos en linea hicieron que el consumo de máquinas arcade se estancara.



Figura 2.3: Joystick utilizado para jugar a Spacewar!

### 2.2.1. Primera generación (1972-1976)

En 1972 fue lanzada de manera oficial la **Magnavox Odyssey** y fue considerada la primera videoconsola. El dispositivo de entrada para poder jugar consistía de 2 diales que se utilizaban para el movimiento horizontal y vertical del personaje [2.4a]. Poco después de la salida de la consola la misma Magnavox sacó su juego de disparos conocido como *Magnavox Odyssey Shooting Gallery* en 1972. Este juego tenía la particularidad de que tenía que ser jugado con un mando diferente al original de la consola. Este accesorio nuevo era una pistola de luz [2.4b]. El funcionamiento de este nuevo mando era que los disparos se registraban siempre y cuando la pistola apuntase a una luz intensa por lo que si el jugador apuntaba hacia una bombilla, el juego marca como “alcanzado” el primer objetivo de la pantalla. La primera solución de este problema fue dibujar la pantalla en negro una vez el objetivo es alcanzado para así comprobar que se está apuntando a la pantalla.



(a) Magnavox Odyssey joystick

(b) Rifle de luz Magnavox Odyssey Shooting Gallery

Figura 2.4: Dispositivos de entrada relevantes en la 1.<sup>a</sup> generación de consolas

### 2.2.2. Segunda generación (1976-1983)

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 [2.5a].

En 1977 salió al mercado uno de los joysticks más famosos. Este joystick es el que se utilizaba en la consola **Atari VCS** que posteriormente sería conocida como **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 [2.5b]. Junto con este modelo, Atari sacó al mercado un tipo de conexión que se convertiría en el estándar de facto. Los sistemas posteriores eran compatibles con estos joysticks ya que lo tomaron como referencia. Unos pocos años después, en 1982, Atari lanzó su nueva consola Atari 5200. 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 [2.5c].

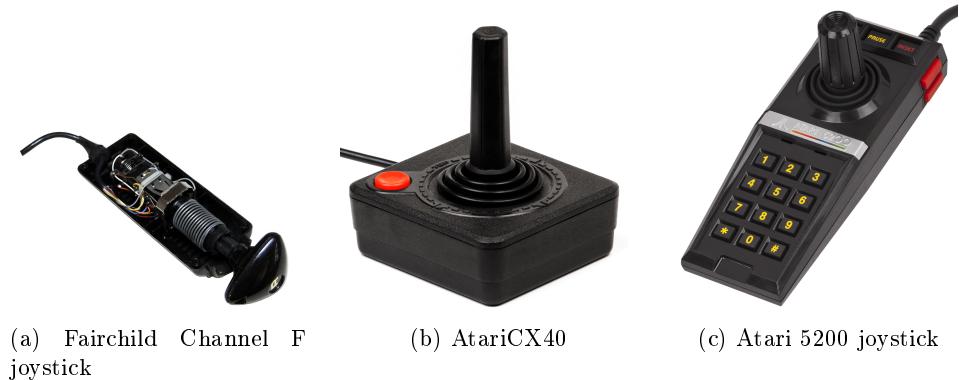


Figura 2.5: Dispositivos de entrada relevantes en la 2.<sup>a</sup> generación de consolas

### 2.2.3. Tercera generación (1983-1987)

En 1983 Nintendo sacó al mercado su **Nintendo Entertainment System (NES)**. El controlador de esta consola no fue el primer dispositivo donde se utilizó pero el éxito de la consola lo popularizó. 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) [2.6a]. 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** [2.6b], 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.

### 2.2.4. Cuarta generación (1987-1993)

En 1990 Nintendo hizo evolucionar a la Nintendo NES y lanzó la **Super Nintendo Entertainment System (SNES)**, la cual dejó atrás un



(a) Mando NES

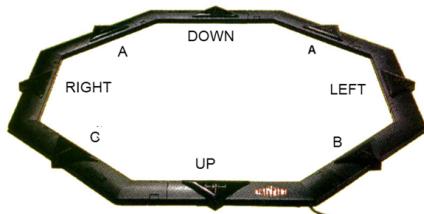


(b) Power Glove NES

Figura 2.6: Dispositivos de entrada relevantes en la 3.<sup>a</sup> generación de consolas

diseño cuadrado del controlador y se inclinó por un diseño más ergonómico, se mejoró la cruceta y se añadieron otros 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 compañía 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 suelo y se conectaba directamente al puerto de controlador de la consola. Lo llamaron **Sega Activator**[2.7a] 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 rayos 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 “pulsaría” cada vez que el jugador cruzase un segmento de los rayos infrarrojos.



(a) Sega Activator

Figura 2.7: Dispositivos de entrada relevantes en la 4.<sup>a</sup> generación de consolas

### 2.2.5. Quinta generación (1993-1998)

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  $\Delta$ ,  $O$ ,  $\times$  y  $\square$ , 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. 3 años más tarde Sony sacaría una re-edición del mando al que le incorporaron 2 sticks 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**[2.8a] 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 extraíble que proporcionaba retroalimentación de vibración [2.8b]. Este accesorio se activaba en ocasiones concretas como al disparar un arma y servía para sumergir al jugador en el videojuego.



Figura 2.8: Dispositivos de entrada relevantes en la 5.<sup>a</sup> generación de consolas

### 2.2.6. Sexta generación (1998-2005)

En los años posteriores las compañías siguieron sacando diferentes 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**[2.9a] y sentó las bases para los próximos mandos inalámbricos. Contaba con una cruceta, 6 botones digitales, 2 botones híbridos ya que hacían 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 baterías a sus mandos para convertirlos en inalámbricos.



(a) WaveBird Wireless Controller

Figura 2.9: Dispositivos de entrada relevantes en la 6.<sup>a</sup> generación de consolas

### 2.2.7. Septa generación (2005-2012)

#### **FALTA TODO LO DE WII QUE HAY QUE VOLVER A ESCRIBIR MEJOR.**

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 permite que 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, **PlayStation Move**, y es compatible con los sistemas 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 controlan 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 Navigation 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.

#### 2.2.8. Octava generación (2012-2020)

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 controlar la pantalla de la televisión u otro aparato vía 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 un panel táctil en la parte frontal, lo que hizo que la distribución de los botones que antes eran centrales cambiase. 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. Los mandos 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. 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. Los Joy-Con contienen baterías no extraíbles 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. 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.

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 dispo-

sitivo 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.

### 2.2.9. Novena generación (2020-Actual)

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. 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.

## 2.3. *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 corrían 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 acoplaban al mando por separado como pasaba con el **Rumble Pack** de Nintendo 64. Con la salida del **Dualshock** en su versión japonesa esto cambió. 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**.

## 2.4. Sistemas de *streaming* en videojuegos

Con la extensión de los dispositivos móviles en el mundo de los videojuegos y las mejoras en las conexiones a internet, la industria de los videojuegos se mueve a pasos acelerados hacia el *gaming* en la nube. Servicios de retransmisión via streaming como *Netflix* o *Amazon Prime Video* ofrecen la posibilidad de disponer de un catálogo de series a películas a través de internet. Con este precedente, empresas como Google<sup>1</sup> y Nvidia<sup>2</sup> se han adentrado al mundo de los juegos en la nube.

Esta nueva forma de forma no solamente consiste en retransmitir al usuario final la partida que está jugando sino que el proceso va mucho más allá. Esta diferencia tiene lugar a la hora de tratar el input del usuario. En un videojuego tradicional, ejecutado en la misma máquina en la que se está jugando, una vez el usuario realiza una entrada el videojuego lo procesa y genera una respuesta apropiada. En los videojuegos en la nube una vez el usuario realiza una acción, esta debe ser enviada al servidor donde el juego está siendo ejecutado. Una vez en el servidor, la entrada que generó el usuario es tratada y se realizan los cambios en el estado del juego que haya provocado el usuario con su entrada. Cuando el estado cambia en el servidor se prepara el frame, se codifica y es enviado de vuelta al usuario que tendrá que decodificar el frame una vez le llegue al dispositivo donde esté jugando (Krishna, 2016).

Los anchos de banda requeridos para estas dos tecnologías varían con las resoluciones a las que se quiera jugar [2.1]. Ambos sistemas aconsejan una conexión Ethernet o un router que disponga de una red de frecuencia 5 GHz.

---

<sup>1</sup><https://stadia.google.com/>

<sup>2</sup><https://www.nvidia.com/es-es/geforce-now/>

	Google Stadia	Nvidia GeForce Now
<b>720p 60fps</b>	10 Mbps	15 Mbps
<b>1080p 60fps</b>	20 Mbps	25 Mbps
<b>4K 60fps</b>	35Mbps	No Disponible

Tabla 2.1: Ancho de banda necesario para jugar a Stadia vs GeForce Now





## Capítulo 3

# Especificación del protocolo de comunicación

La finalidad de este trabajo es conseguir la conexión entre 2 dispositivos con la creación de una librería para un motor de videojuegos. Uno de ellos va a funcionar como un controlador de videojuegos y otro como ejecutor del juego. Para que esta comunicación se consiga es necesario analizar y definir las funcionalidades que se quieren ofrecer a los desarrolladores que usen la librería.

En este capítulo se va realizar un análisis de las diferentes funcionalidades que son necesarias para que la comunicación pueda realizarse. Se expondrán las ventajas y las desventajas de cada una de estas funcionalidades y por último se definirá el protocolo de comunicación entre los 2 dispositivos involucrados en este trabajo.

### 3.1. Arquitectura del proyecto a alto nivel

Como ya hemos mencionado, el objetivo a alto nivel consiste en lo siguiente (figura 3.1).

Para conseguir usar el dispositivo móvil como mando para videojuegos se plantearon 3 posibles diseños. Estas opciones difieren en la versatilidad que se le ofrece al usuario de la librería, la complejidad de programación y el ancho de banda usado durante la ejecución del juego.

La primera de estas opciones se basa en el uso de imágenes estáticas de mandos cargados previamente en la aplicación del móvil. La comunicación entonces serían las pulsaciones que se envían desde el móvil al ordenador. En este esquema el feedback viene dado por la aplicación que se ejecuta en el dispositivo móvil. La principal ventaja de este diseño de arquitectura es

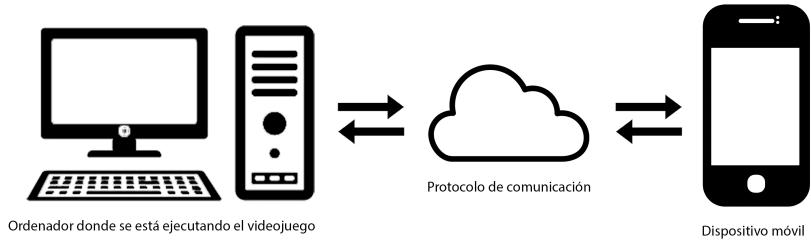


Figura 3.1: Arquitectura del proyecto

el poco uso de ancho de banda. Con este diseño la única comunicación que se tiene por red es el envío de las pulsaciones del usuario en la pantalla del móvil al ordenador. Esta comunicación puede realizarse utilizando TCP y conseguir así que no se pierda ninguna pulsación del usuario. Otra ventaja que ofrece este diseño es la poca complejidad que supone para el usuario final de la librería. Con este sistema, el juego será notificado de la llegada de pulsaciones desde el teléfono gracias a la librería desarrollada. El desarrollador del juego únicamente tendrá que encargarse de decidir qué hacer con la llegada de cada pulsación. Esto lleva directamente a la primera desventaja, la poca versatilidad que se le da al usuario de la librería. Al no haber una comunicación desde el videojuego hasta el móvil, perdemos la oportunidad de que el desarrollador del juego modifique el mando que quiere utilizar o que ordene al teléfono activar la vibración a su gusto.

Para resolver el problema de la modificación del mando que usar se planteó el segundo diseño: permitir que el juego envíe imágenes en formato .PNG a lo largo de la sesión de juego. Para que esto sea posible se tiene que habilitar el envío de datos en ambas direcciones. Hacer esto supone un aumento en el uso necesario del ancho de banda a cambio de dar más versatilidad al usuario de la librería. Para conseguir la modificación de las imágenes, Android ofrece un sistema de capas en las imágenes que se describan en el archivo de manifiesto. Estas capas se pintan en orden descendiente, siendo la última imagen declarada la que taparía al resto.

Para disminuir el impacto en el ancho de banda se planteó la opción de que el envío del mando desde el juego al dispositivo móvil se hiciese únicamente

mente al inicio de la conexión. Hacerlo de esta manera permitiría al usuario de la librería modificar el mando con el que se quiere jugar pero este no se podría cambiar en ningún momento de la ejecución. Esta decisión aumenta el coste en el ancho de banda al inicio de la conexión pero disminuye drásticamente una vez que la imagen del mando ha sido guardada, lo que permite seguir usando TCP como protocolo de transmisión.

El último de los diseños es el más versátil para el desarrollador del videojuego pero también el más costoso en cuanto al ancho de banda que usa. Este diseño da la posibilidad al usuario de la librería enviar streaming de video al dispositivo móvil y que este lo muestre por pantalla. Para que esto sea posible la comunicación entre ambos dispositivos tiene que realizarse en ambos sentidos durante toda la ejecución con un intercambio constante de datos. Como este diseño es el más versátil para el usuario de la librería, en esta arquitectura se daría la posibilidad de controlar la vibración también al desarrollador. Darle todas estas características al desarrollador implica un aumento en la complejidad del código que este tiene que implementar. Dado que el consumo del ancho de banda sube de manera considerable, en este diseño se ha optado por modificar el protocolo de transmisión de TCP a UDP para disminuir el consumo. Esto implica que se asume una posible pérdida de pulsaciones realizadas por el usuario o una posible pérdida de frames en el envío del streaming de video.

Este último diseño ha sido el utilizado para este proyecto. Las funcionalidades que se ofrecen al usuario de la librería son:

- Envío de pulsaciones desde el dispositivo móvil al juego.
- Envío de streaming de imágenes en formato .PNG desde el juego al móvil.
- Envío de orden de vibración del juego al móvil.

Para que el desarrollador tenga un control total de lo que se envía en cada momento, se da la opción de enviar un mensaje de vibración al dispositivo móvil. Esto permite que el móvil vibre no solo por las pulsaciones, sino, por ejemplo, cuando el jugador reciba daño o se choque si está jugando a un juego de conducción. En la siguiente sección se detalla el protocolo desarrollado y en el próximo capítulo se detallará la implementación de la parte de móvil y PC.

### 3.2. 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.

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. Este primer mensaje es enviado por el móvil y tiene de tamaño 1 byte y define el número de versión del protocolo. Este número de versión es utilizado para comprobar si el cliente y el servidor son compatibles y así evitar errores en la interpretación de los datagramas. Instantáneamente después, el dispositivo móvil envía al juego la resolución de su pantalla. Este dato viene dado en un mensaje de 4 bytes donde los 2 primeros se tratan como el ancho de la pantalla y los 2 siguientes como el alto. Se han utilizado 2 bytes porque no se esperan resoluciones de pantalla que superen el valor de  $2^{16}$ .

Bits	0-15	16-32
0	Ancho	Alto

Tabla 3.1: Envío de la resolución del móvil al juego.

En caso de que estos mensajes lleguen con un formato incorrecto son descartados. En caso de que la versión del protocolo no coincida, el servidor descarta el mensaje y sigue a la espera de una versión que coincida.

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 haptico. Este mensaje contiene 3 bytes que se distribuyen de la siguiente forma:

- Primer byte con valor 5 a modo de cabecera para que se sepa el tipo del mensaje.
- 2 bytes que indican la duración de la vibración en milisegundos.

Este mensaje puede ser enviado de nuevo en caso de que quiera cambiarse

Bits	0-7	8-23
0	0000101	Tiempo de vibración

Tabla 3.2: Tiempo de vibración del dispositivo móvil en milisegundos

la duración de la vibración. Además de modificar el tiempo, se da la opción de hacer peticiones al móvil para que este vibre por una determinada acción del juego. Esto puede hacerse con el envío de un mensaje de tamaño 1 byte cuya cabecera contenga el valor 2, valor designado para activar la vibración en el tiempo determinado por la estructura anterior.

Tras el envío de este mensaje comienza un bucle de juego en el que ambos dispositivos intercambian mensajes de una manera no ordenada. Además de la vibración, el juego tiene la posibilidad de enviar imágenes. Estas imágenes pueden enviarse en forma de imágenes estáticas o en forma de streaming de video. El formato de compresión utilizado será **PNG** y la cabecera que diferencia este tipo de mensajes es un byte con valor 3.

Además de enviar estos mensajes, el dispositivo encargado de ejecutar el juego recibirá mensajes de 6 bytes cuya estructura será:

- El primer byte tiene el valor 0 e indica el tipo del mensaje.
- El segundo 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 últimos 4 bytes representan las coordenadas X e Y de la pulsación del usuario. Los 2 primeros bytes indican la coordenada X en la pantalla del móvil y los 2 últimos la coordenada Y.

Bits	0-7	8-15	16-31	32-47
0	00000000	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 como cabecera del mensaje el valor 4 que es el valor que indica cierre de conexión. Para este mensaje no se espera nada de vuelta ya que es siempre el móvil el que cierra la conexión. Para la detección de pérdidas de conexión se utilizan paquetes de tipo *Keep Alive* cuya cabecera tiene el valor 6. Este paquete consta del envío de 1 solo byte y una vez que este tipo de mensaje se envía el receptor está obligado a responder para que la conexión no se corte.

Tal y como hemos visto en este capítulo, para la realización de este proyecto surgieron 3 alternativas de las posibles características que debería incorporar. Tras analizar las ventajas y desventajas se escogió la que mayor

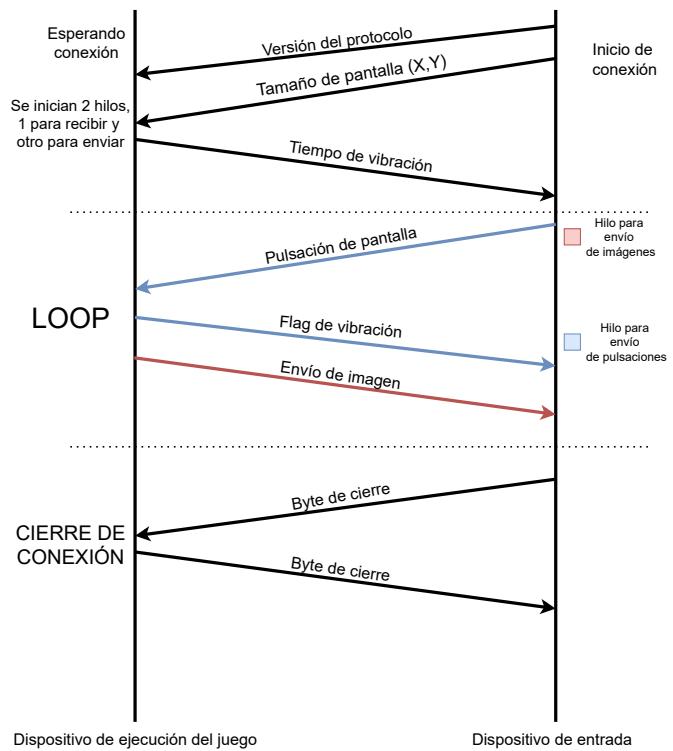


Figura 3.2: Diagrama del protocolo de comunicación entre ambos dispositivos

versatiidad daba al desarrollador a cambio de un mayor coste en el ancho de banda. Para solventar esto se decidió utilizar UDP como protocolo de transmisión. En el próximo capítulo se procede a explicar la implementación tanto de la aplicación de Android como de la aplicación de Unity.

## Capítulo 4

# Implementación de las aplicaciones

En el capítulo anterior se analizaron las diferentes opciones de diseño del proyecto y se detalló el protocolo de comunicación entre el juego y el dispositivo móvil. Además de esto se expusieron las ventajas y desventajas de usar los diferentes conjuntos de características expuestas. En este capítulo se detallará todo el proceso de creación y desarrollo de las diferentes aplicaciones que utilizan el protocolo de comunicación anteriormente expuesto. Se ha dividido este capítulo en 2 secciones, una para cada aplicación desarrollada:

- Implementación de la aplicación de Android (sección 4.1)
- Implementación de la aplicación de Unity (sección 4.2)

### 4.1. Implementación de la aplicación de Android

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. Actualmente Android Studio 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.
- Dispositivos virtuales 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. Conceptos básicos 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 instantánea. Un usuario puede estar navegando por cualquier red social y encontrarse una publicación interesante y compartir la por correo electrónico o por una aplicación de mensajería instantánea. 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.

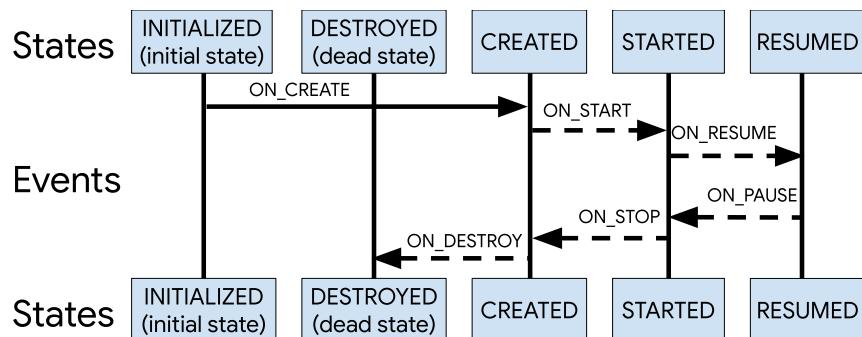


Figura 4.1: Estados de las aplicaciones Android

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. 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 (figura 4.1). 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 actividad entra en un nuevo estado.

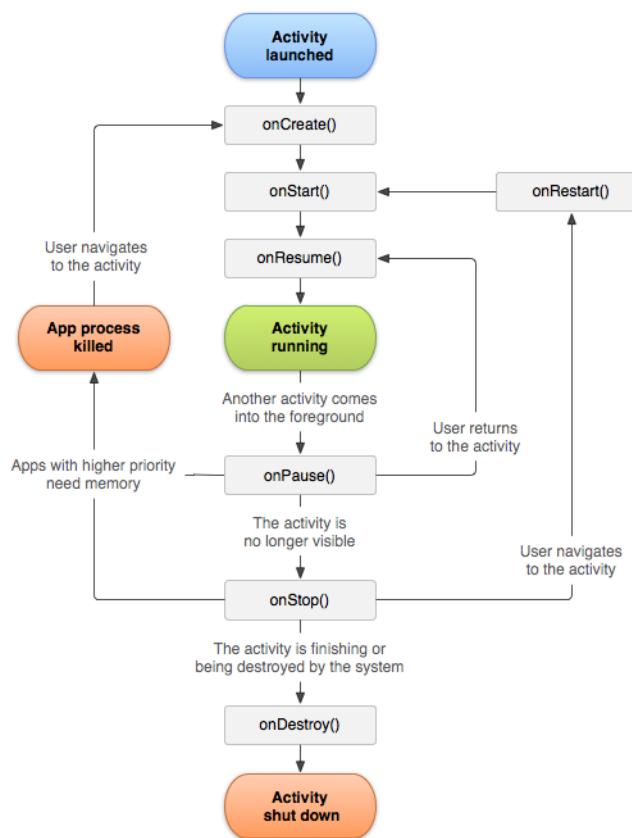


Figura 4.2: Ciclo de vida de una Actividad de un sistema Android

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.
- **onResume()** → La aplicación permanece en el 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 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.

En el siguiente apartado se explicará el uso de estas llamadas del sistema Android dentro del proyecto y la utilización de diferentes actividades. Además se expondrá la arquitectura de la aplicación desarrollada para Android.

#### 4.1.2. Desarrollo de la aplicación Android

Una vez se tienen claras las diferentes llamadas que realiza el sistema Android y cuándo las realiza, es el momento de ver la aplicación que tienen en el desarrollo de este proyecto. Tal y como se planteó en la descripción del protocolo de comunicación, la aplicación de Android necesita saber la IP y el puerto al que debe conectarse para enviar las pulsaciones. Para este primer paso se decidió utilizar un código QR que al leerlo con el móvil, este contenga la dirección IP y el puerto al que la aplicación se debe conectar. La lectura

del código QR y el envío de las pulsaciones en pantalla son 2 procesos muy distintos e independientes, por esta razón se implementaron 2 actividades diferentes.

La primera de estas actividades tiene como función activar la cámara del dispositivo Android para leer un código QR. El contenido de este código QR viene dado con el formato IP:Puerto (por ejemplo 192.168.1.1:5000). Una vez el código QR es correcto, Android ofrece la posibilidad de lanzar una actividad nueva con una serie de parámetros. Para este proyecto uno de estos parámetros será el contenido del código QR para que sea la actividad hija la que realice la conexión y el intercambio de datos con el videojuego.

En esta segunda actividad se realiza la conexión con el videojuego, lo que implica el envío de pulsaciones y la recepción de imágenes y mensajes de vibración. Para que estos procesos se realicen de manera independiente y asíncrona se han implementado 2 hilos de ejecución diferentes. Uno de ellos se encarga de gestionar la recepción de datos y otro del envío de pulsaciones. Como esto debe realizarse al inicio de la ejecución de la actividad, el proceso de inicialización de estos hilos junto con los *listeners* de las pulsaciones de la pantalla se realizan en el método *onCreate()*.

La actividad llamará a la función *onPause()* en caso de que se pierda el foco de la aplicación, es por eso que para parar por completo el consumo de la aplicación se cierran los hilos y se manda el mensaje de cierre de conexión al juego. Esto ocurre para que no sea el sistema Android el que cierre los hilos de una manera inesperada.

Para que la aplicación Android cumpla los requisitos expuestos en el apartado de especificación del proyecto, se han implementado 4 clases:

- **MainActivity** → Esta primera clase corresponde a la primera actividad. En esta primera actividad se utiliza la cámara para leer un código QR y guardar como parámetro el contenido del código. Este contenido es la IP del equipo donde se está ejecutando el juego y el puerto al que deben ser enviadas las pulsaciones del usuario. Estos datos son pasados a la actividad hija para comenzar a usar el móvil como dispositivo de entrada.
- **Controller** → Esta es la actividad principal. Desde esta actividad se recogen los datos de IP y puerto al que debe conectarse el dispositivo Android gracias a la lectura de un código QR realizada con la actividad padre. 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). Este dato permite soportar el *multitouch* en la aplicación. El comportamiento de este hilo viene dado en la clase **UdpClientThread** que se encarga de preparar el datagrama de la pulsación y enviarlo. 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. 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 datagrama 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 manda el juego. En este hilo se espera la llegada del *streaming* de imágenes desde el juego. Estas imágenes se esperan en formato PNG ya que se realiza la descompresión de este formato. El tiempo de vibración puede ser modificado en cualquier momento por el desarrollador por lo que ese mensaje también es tratado en este hilo.

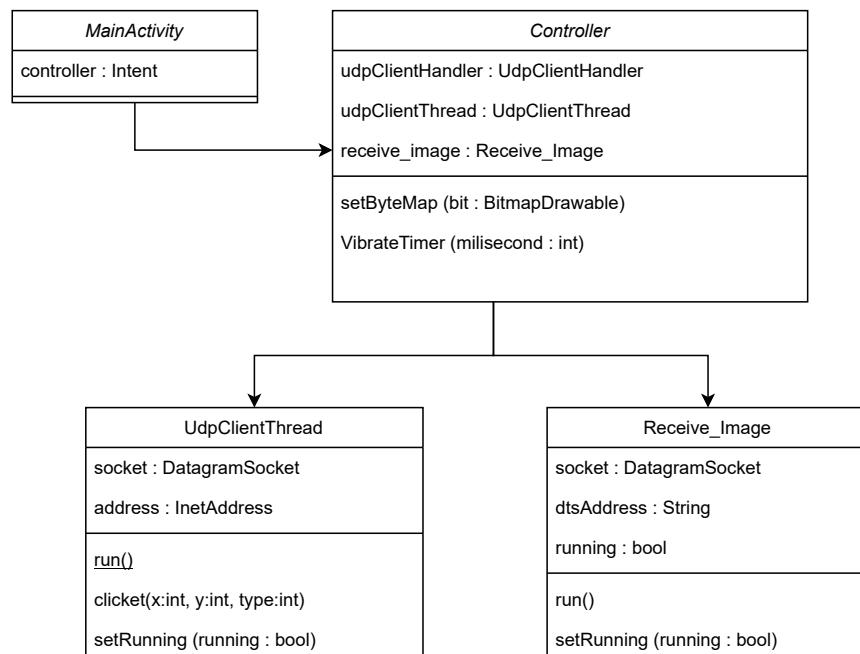


Figura 4.3: Diagrama de clases de la aplicación Android

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

Actualmente en la versión 2021.1 de la documentación de Unity, no existe soporte para la nueva generación de consolas.

### 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 Scripts. 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.

#### 4.2.2. Desarrollo de la API en Unity

Las características específicas de Unity deben tenerse en cuenta para la integración de la librería dentro del motor pero la librería está desarrollada en .NET. El inicio de la comunicación entre en juego y el móvil se realiza con la lectura de un QR que lleva los datos de IP del PC donde se está ejecutando el juego y el puerto donde el juego va a estar escuchando y por donde llegarán los datos del móvil. Para conseguir que el móvil disponga de estos datos se ha decidido utilizar un código QR. Este código se genera utilizando la librería **ZXing**<sup>1</sup> en su versión de .NET.

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 móvil. Estos datos son tanto la vibración como la imagen a renderizar en el dispositivo o los mensajes de tipo *keepalive* en caso de que el usuario no interactue con el juego en un tiempo determinado. 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 designados por el desarrollador del juego (mover al personaje, pausar el juego, salir, etc). Esta clase también se encarga de cerrar la conexión.

---

<sup>1</sup><https://archive.codeplex.com/?p=zxingnet>

- **InputMobileInterface** → Esta interfaz permite al desarrollador del juego recibir los eventos que llegan desde el móvil. Estos eventos son las coordenadas de las pulsaciones, las dimensiones del móvil y el aviso del cierre de la conexión por parte del móvil. El método **EndOfConnection()** comunica la pérdida de conexión al *listener* para que el usuario de la librería realice una acción, por ejemplo pausar el juego hasta que la conexión vuelva a iniciarse.

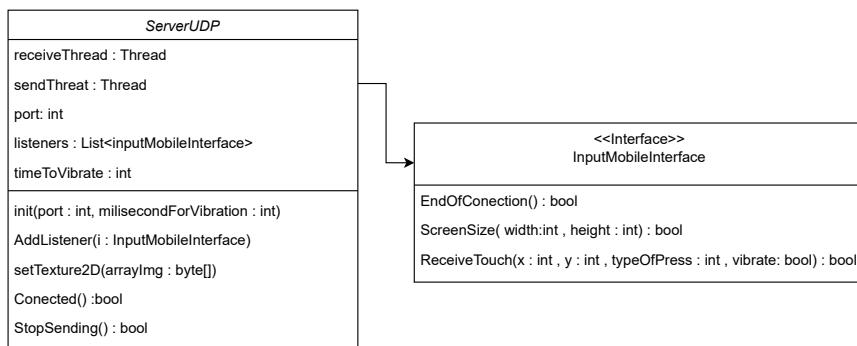


Figura 4.4: Diagrama de clases de la aplicación de Unity

En este capítulo se ha expuesto el desarrollo de las aplicaciones de Android y Unity. Además se han explicado algunos de los conocimientos básicos para usar ambas herramientas. En el próximo capítulo se explicará la integración de la librería en un juego ya terminado con el que se realizarán una serie de pruebas con usuarios. Estas pruebas servirán para obtener datos del rendimiento del proyecto. Aplicando una serie de baremos se determinará si el uso de la librería desarrollada cumple con las expectativas.



# 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 la 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**.<sup>1</sup> 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**<sup>2</sup>, 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 que 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,

---

<sup>1</sup>Unity Learn - <https://learn.unity.com/projects>

<sup>2</sup>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 librería **ZXingNet**<sup>3</sup>. 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 Actividad 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:

<sup>3</sup>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 llenar.
- 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 llenar 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 obtiene 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 parecían 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.

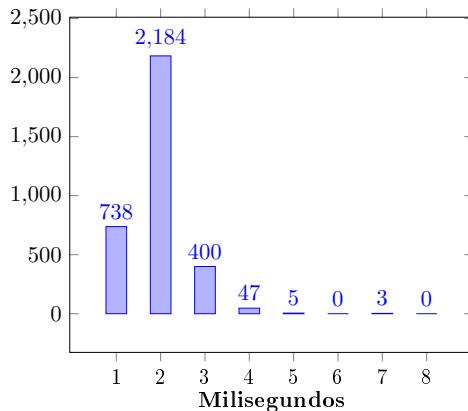


Figura 5.3: Tiempo de descompresión PNG en dispositivo Android Usuario 1

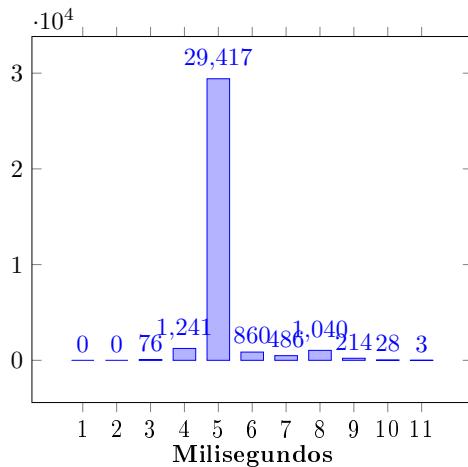


Figura 5.4: Tiempo de conversión de cámara de Unity a PNG Usuario 1

## 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.

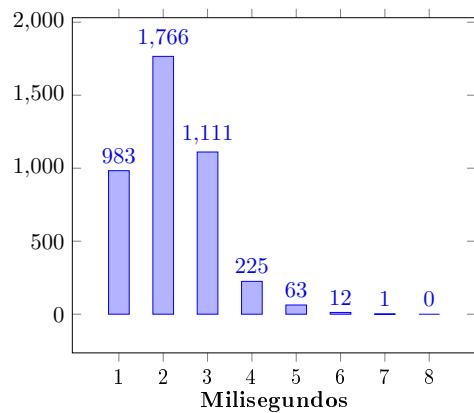


Figura 5.5: Tiempo de descompresión PNG en dispositivo Android Usuario 2

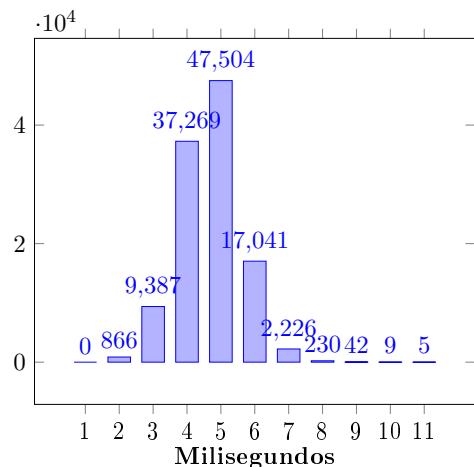


Figura 5.6: Tiempo de conversión de cámara de Unity a PNG Usuario 2

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 Media. 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.

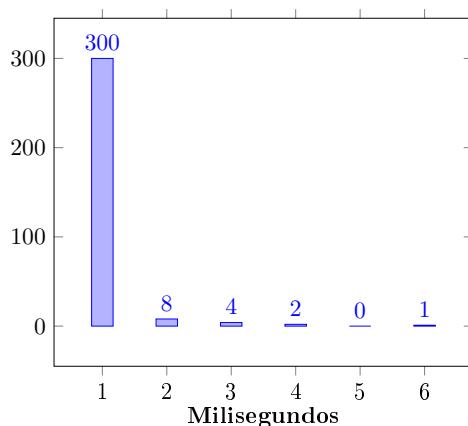


Figura 5.7: Tiempo de descompresión PNG en dispositivo Android Usuario 3

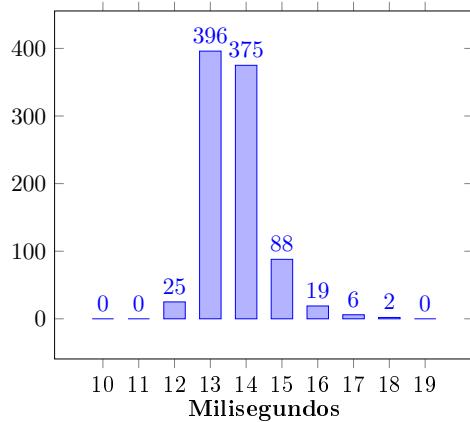


Figura 5.8: Tiempo de conversión de cámara de Unity a PNG Usuario 3

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 obtiene 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.

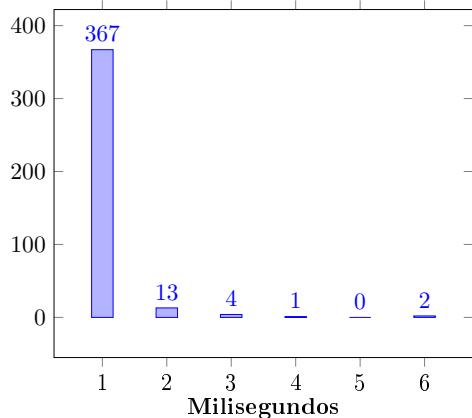


Figura 5.9: Tiempo de descompresión PNG en dispositivo Android Usuario 4

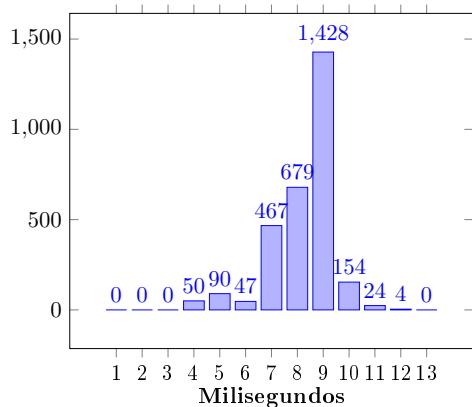


Figura 5.10: Tiempo de conversión de cámara de Unity a PNG Usuario 4

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 milisegundo 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 milisegundo.

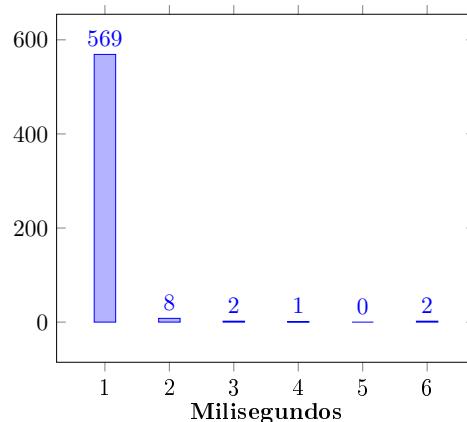


Figura 5.11: Tiempo de descompresión PNG en dispositivo Android Usuario 5

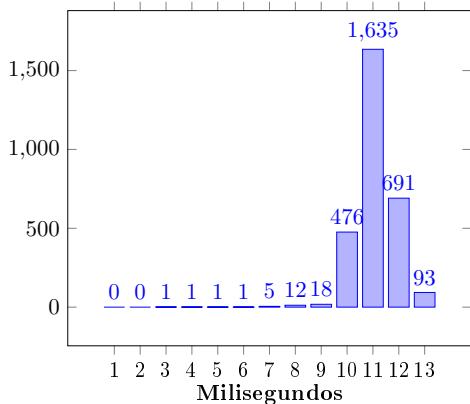


Figura 5.12: Tiempo de conversión de cámara de Unity a PNG Usuario 5

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.

# Bibliografía

AEVI. La industria del videojuego en España anuario 2019, 2019. URL <http://www.aevi.org.es/web/wp-content/uploads/2020/04/AEVI-ANUARIO-2019.pdf>.

George Beekman. *Introducción a la Computación*. 1999.

DEV. Libro blanco del desarrollo español de videojuegos, 2019. URL <https://esportsobserver.com/q3-2020-impact-index/>.

Daniel Entrialgo. Xerox alto, el ordenador maldito que nunca se vendió pero “inspiró” (e hizo ricos) a Steve Jobs y Bill Gates, 2020. URL <https://www.economista.es/status/noticias/10455084/04/20/Xerox-Alto-el-ordenador-maldito-que-nunca-se-vendio-pero-inspiro-e-hizo-ricos-a-Steve-Jobs-y-Bill-Gates.html>.

Juan Antonio Pascual Estapé. Móviles gaming: ¿son una amenaza para las consolas?, 2018a. URL <https://www.hobbyconsolas.com/listas/moviles-gaming-son-amenaza-consolas-342681>.

Juan Antonio Pascual Estapé. Tennis for two, el primer videojuego, cumple 60 años, 2018b. URL <https://computerhoy.com/noticias/gaming/tennis-two-primer-videojuego-cumple-60-anos-318133>.

Clinton Keith. *Agile Game Development with Scrum*. 2010. URL <http://index-of.co.uk/Agile/Agile%20Game%20Development%20With%20Scrum.pdf>.

P. Venkata Krishna. *Emerging Technologies and Applications for Cloud-Based Gaming*. 2016.

Jowen Mei. The original paper on scrum (oopsa 1995), 2020. URL <https://www.linkedin.com/pulse/original-paper-scrum-oopsa-1995-jowen-meि/?articleId=666958521588883712>.

David Sanmartín. El futuro y los retos del “gaming” para teléfonos móviles, 2019. URL [https://retina.elpais.com/retina/2019/08/27/innovacion/1566898640\\_613309.html](https://retina.elpais.com/retina/2019/08/27/innovacion/1566898640_613309.html).

Schwaber and Sutherland. Scrum development process, 1995. URL [http://agilix.nl/resources/scrum\\_OOPSLA\\_95.pdf](http://agilix.nl/resources/scrum_OOPSLA_95.pdf).

Jimmy Stamp. Fact of fiction? the legend of the qwerty keyboard, 2013. URL <https://www.smithsonianmag.com/arts-culture/fact-of-fiction-the-legend-of-the-qwerty-keyboard-49863249/>.

TEO. Teo pc games impact index, 2020. URL <https://esportsobserver.com/q3-2020-impact-index/>.