Escuela Politécnica Superior

Desarrollo de un videojuego en Unity controlado mediante un smartwatch

Máster Universitario en Desarrollo de Software para Dispositivos Móviles

Trabajo Fin de Grado

Autor:

Juan Pomares Bernabeu

Tutor/es:

Miguel Ángel Lozano Ortega

Septiembre 2016

Universitat d'Alacant
Universidad de Alicante

Agradecimientos

Me gustaría agradecer a diversas personas la colaboración que me han ofrecido a la hora de realizar este proyecto, ya que seguramente sin su ayuda, el resultado final no hubiese sido tan bueno.

En primer lugar, a mi tutor, Miguel Ángel, por sus consejos y su sabiduría demostrada a lo largo de estos meses, donde me ha orientado bastante bien durante el desarrollo de todo el proyecto.

Por último, agradecer a mi familia y amigos, que siempre han estado ahí apoyándome y ayudándome en todo lo que han podido.

Citas

"Todos somos muy ignorantes. Lo que ocurre es que no todos ignoramos lo mismo."

Albert Einstein.

"Medir el progreso del desarrollo de software por las líneas de código es como medir el progreso de la construcción de un avión por su peso."

Bill Gates.

"Si la depuración es el proceso de eliminar errores, entonces la programación debe ser el proceso de introducirlos".

Edsger W. Dijkstra.

"Cuando se está depurando, el programador novato introduce código correctivo; el experto, elimina el código defectuoso".

Richard Pattis.

Justificación y objetivos

El proyecto consiste en desarrollar un videojuego compuesto de diversos minijuegos que se controlen usando un smartwatch. Además de la realización de este proyecto se estudiarán los usos que se dan a los relojes inteligentes, así como la existencia de proyectos parecidos.

El motor de videojuegos que se usará será **Unity**, ya que es un motor gratuito muy potente, donde se pueden crear videojuegos de forma sencilla. Se desarrollará un plugin para este motor, para poder usar el reloj como controlador de juegos. Antes de crear el videojuego, se realizará una aplicación auxiliar para poder ver el funcionamiento de este plugin.

Por último, otro objetivo de este proyecto, es la publicación del juego final en el **Play Store**. De esta forma, la aplicación estará disponible en la tienda, para que pueda ser probada.

En un primer momento, la idea del proyecto era un poco distinta. Consistía en realizar un escenario virtual en **Unity** y utilizar **Google CardBoard**, para darle más inmersión al usuario. Como método de control, para poder desplazarse por el escenario, se usarían los movimientos o gestos que se realizarán en un smartwatch.

Después de una primera búsqueda no se encontró ningún lpugin que se pudiera usar para este proyecto, la solución sería crear uno propio. Así que se decidió realziar la creación del mismo y una serie de minijuegos para poder usarlo; y realizar el escenario virtual como proyecto futuro.

Índice de contenidos

Índice de abreviaturas

Las abreviaturas que se usan en diversas partes de la memoria son las siguientes:

- **APK:** Android aPplication pacKage (aplicación empaquetada de Android).
- **API:** Application Programming Interface (interfaz de programación de aplicaciones).
- **HDMI:** High Definition Multimedia Interface (interfaz multimedia de alta definición).
-
- **IDE:** Integrated Development Environment (entorno de desarrollo integrado).
- **GPS:** Global Positioning System (sistema de posicionamiento global).
- **MAME:** Multiple Arcade Machine Emulator (emulador de múltiples máquinas recreativas).
- **SNES:** Super Nintendo Entertainment System (sistema de entretenimiento de Nintendo.
- **XML:** eXtensible Markup Languaje (lenguaje de marcas extensible).

Índice de figuras

Índice de tablas

Cuerpo del documento

1 Introducción

Durante los últimos años se ha observado un gran cambio en el sector tecnológico. Ahora, si nos centramos en la telefonía móvil, se puede apreciar que el avance ha sido aún mayor. Los móviles han reducido sus tamaños, aumentado su capacidad de cómputo y posibilidades.

Cuando se diseñó el primer teléfono móvil, éste fue pensado, únicamente, para realizar y recibir llamadas. Actualmente, esta es una de las funciones que, paradójicamente, menos se usa. Esto es debido en gran parte a las aplicaciones instaladas que permiten realizar gran cantidad de acciones. Existen desde aplicaciones simples de comunicación instantánea, que hacen que el uso de los SMS sea escaso en la actualidad, hasta aplicaciones más complejas que, por ejemplo, permiten tener un control total de la casa mediante domótica.

Figura 1 - Teléfono móvil con varias aplicaciones

Los teléfonos móviles actuales, debido a su potencia y prestaciones, permiten la instalación de diversas aplicaciones y un tipo de ellas son los videojuegos. En la tienda de aplicaciones las más descargadas y que más beneficios aportan, son sin lugar a dudas, los videojuegos. Como se puede comprobar los videojuegos en los móviles son muy importantes.

Recientemente, están saliendo unos dispositivos llamados wearables que se conectan al teléfono móvil y añaden ciertas funcionalidades. Existen desde dispositivos simples como pulseras cuantificadoras que únicamente miden la cantidad de pasos y calorías,

hasta otros más complejos como los smartwatches o las **Google Glass**. Debido a la conexión constante de estos dispositivos con el teléfono, las posibilidades de uso que ofrecen son elevadas.

Figura 2 – Distintos wearables

El propósito de este proyecto consiste en crear un videojuego con Unity para dispositivos móviles cuyo método de control sea el wearable. Realmente el proyecto no consiste únicamente en la realización del videojuego, más bien consiste en crear una librería/plugin para Unity que permita este tipo de control. Para poder probar este añadido, se realizarán diversos minijuegos con diferentes controles. Además, de la realización de los minijuegos y la librería, se estudiarán los posibles usos de los relojes inteligentes y la existencia de algún plugin o proyecto similar.

2 Objetivos

El objetivo principal de este proyecto consiste en realizar un videojuego en **Unity** para móviles que sea controlado con el smartwatch. Este videojuego estará compuesto de diversos minijuegos, cada uno de ellos se controlará de forma distinta, bien sea usando diferentes interfaces táctiles o diferentes controles en el smartwatch.

Antes de empezar a desarrollar el videojuego se estudiará si existen aplicaciones parecidas o plugins existentes para realizar esto en **Unity**. Este estudio servirá para tener una idea sobre cómo está el mercado y si hay alguna forma de mejorar lo actual, en caso de que hubiera algo creado.

El plugin o librería que se desea desarrollar para **Unity** debe realizar una comunicación entre el móvil y el reloj. Básicamente debe enviar mensajes al móvil cuando se produzcan cambios en el reloj y el móvil debe poder 'decirle' al reloj que entre/salga de la aplicación, cambie de interfaz, etc.

Debido a que previamente no se han realizado plugins para **Unity** ni **Android Wear** (smartwatch), primero se crearán unos proyectos básicos para probar el funcionamiento.

Después de crear las diferentes pruebas, se creará el proyecto que servirá como antecesor a los juegos. En esta aplicación se podrá comprobar de una forma, más o menos, gráfica todos los datos que se reciben desde el reloj. De esta forma se puede comprobar que datos son los que recibirán los juegos de **Unity** y probar que todo funcione correctamente.

Por último, cuando ya se haya creado esta prueba/aplicación, es momento de empezar con la creación de los diversos minijuegos en **Unity**. La aplicación con los minijuegos, se puede considerar como la versión final del proyecto y será subido al **Play Store**.

3 Marco teórico

En primer lugar, a modo de introducción, se realizará un estudio previo, sobre el cambio desde los primeros smartwatches hasta los actuales. En él se intentará determinar cuáles han sido y son los principales usos de estos dispositivos, poniendo algunos ejemplos de aplicaciones.

Por otro lado, se realizará un estudio sobre proyectos y plugins para Unity que se centren en los 'smartwatches' como método de control. Ya que el objetivo principal de este proyecto consiste en realizar un juego controlado mediante wearable, es importante saber si se ha realizado algún proyecto similar y la existencia de plugins que realicen cosas parecidas.

3.1 Smartwatches y sus principales usos

3.1.1 Historia de los smartwatches

Aunque parezca mentira, los smartwatches, llevan bastante tiempo entre nosotros. El primero de ellos fue el **Pulsar NL-C01** de **Seiko** en 1982. Apenas permitía almacenar 24 dígitos de información, pero puede ser considerado smartwatch. Después de este 'reloj inteligente' fueron añadiéndoles diferentes mejoras y funcionalidades. En el año 2003, la empresa **Fossil** lanzó al mercado **Wrist PDA**, un reloj con funciones de PDA, pantalla LCD y más funciones que el anterior. Pero el gran salto tecnológico sucedió en 2010, cuando **Sony** lanzó el **LiveView**, con pantalla táctil OLED y conexión Bluetooth con nuestro smartphone **Android**.

Seiko RC1000 | **Fossil Wrist PDA** | **Sony LiveView**

Figura 3 - Primeros smartwatches

Después del lanzamiento del **Sony LiveView**, varias marcas han lanzado sus propios relojes con diversos sistemas operativos: **PebbleOS**(2012), **Tizen**(2013), **Android Wear**(2014) o **watchOS**(2015). De todos estos sistemas operativos para relojes, el más abierto a crear aplicaciones, mejor documentado y más compatibilidad con diversos sistemas operativos (**Android** e **iOS**) es **Android Wear**. Sin embargo, un smartwatch con este sistema operativo en iOS, sólo se puede usar para notificaciones y calendario, no permite la instalación de nuevas apps; pero ya el algo más que **WathOs** que solo permite sincronización con teléfonos con **iOS**.

Pebble Watch | **Samsung Gear S** | **Motorola Moto 360** | **Apple Watch**

Figura 4 - Diferentes smartwatches y sistemas operativos

De la creación de los smartwatches no se encargan únicamente los fabricantes de teléfonos móviles, los fabricantes de relojes tradicionales también han creado los suyos propios. Todos ellos poseen Android Wear como sistema operativo, debido a las ventajas comentadas anteriormente. Algunos ejemplos podían ser: **Tag Heuer Connected**, **Fossil Q Founder**, **Casio Smart Outdoor Watch** o **Michael Kors Access**. La principal diferenciación que aportan estos relojes podría ser la exlusividad y personalización que ofrecen los fabricantes; ya que ofertan varias correas oficiales para sus modelos. El precio de estos relojes es un poco superior a los otros smartwatches; a diferencia del smartwatch de **Tag Heuer** cuyo precio recomendado es, un poco más elevado, 1350€ (1500$)[1].

Figura 5 - Smartwatches marcas convencionales

[1] Dato obtenido de la página oficial del reloj (http://www.tagheuerconnected.com/es/)

3.1.2 Principales usos

Como se ha podido comprobar los primeros smartwatches no tenían muchas funcionalidades y su uso era bastante reducido. Con la llegada de los últimos smartwatches, la utilidad de éstos se ha visto multiplicada. Además, debido a sus sensores y conectividad con el teléfono móvil, las posibilidades que ofrecen son casi infinitas. Ahora se verán en detalle algunas de ellas, evidentemente, no se comentarán todas, ya que el documento sería demasiado extenso.

El principal uso de todos los relojes es poder ver la hora, con los smartwatches no iba a ser diferente. Cuando tienen que mostrar la hora, estos dispositivos tienen una ventaja con respecto a los tradicionales, la esfera se puede cambiar fácilmente. A estas esferas se les llama 'watchfaces'. Para cambiarla es suficiente con seleccionar alguna que viene por defecto desde el reloj o descargar alguna de la tienda de aplicaciones. Existen multitud de watchfaces, unos muestran la hora como un reloj digital, otras como uno analógico, algunos permiten al usuario seleccionar la imagen de fondo o mostrar datos climatológicos, incluso existen watchfaces que imitan las esferas de relojes convencionales como **Rolex**, **Omega** o **Casio**, entre otros.

Figura 6 - Diferentes watchfaces y smartwatches

Una vez visto el uso más simple que se le puede dar al smartwatch, ver la hora, vamos a ver uno un poco más novedoso. Debido a la conexión constante que existe entre el reloj y el teléfono, mediante **Bluetooth**, se pueden visualizar las notificaciones también en el reloj.

Cuando una notificación llega al móvil, ésta automáticamente se puede ver, también, en el reloj. Además, desde el reloj, puedes eliminarla o realizar ciertas acciones con ella. Algunas de ellas te permiten abrir la aplicación en el teléfono.

Por ejemplo, la aplicación de mensajería instantánea **Whatsapp**, permite algunas acciones muy interesantes. Desde el mismo smartwatch, se puede leer todos los mensajes recibidos en una conversación, así como contestar mediante el envío de emoticonos, notas de voz o texto trascrito. Por otro lado, si el reloj dispone de altavoces, se pueden contestar las llamadas a través de él.

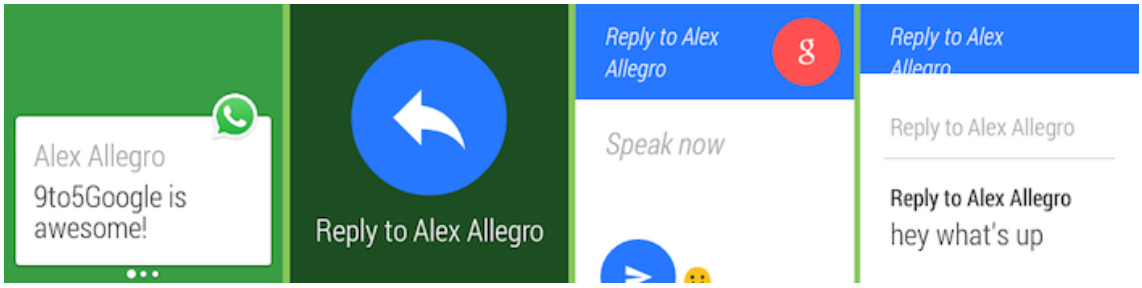


Figura 7 - Acciones de Whatsapp en un Android Wear

Como se ha podido comprobar, otra posibilidad que poseen los smartwatches es poder controlar el teléfono móvil. Debido a que el reloj, normalmente, es más accesible que el teléfono, este tipo de funcionalidades son muy útiles. Los reproductores de música, por ejemplo, **Spotify** o **Play Music,** permiten, entre otras acciones, controlar el volumen, pausar/reproducir la canción actual o seleccionar una lista de reproducción o buscar una canción.

Figura 8 - Spotify en Android Wear

Como se puede comprobar, muchas aplicaciones para smartphone han sacado versiones reducidas para smartwatch. En ellas, permiten realizar algunas de las funcionalidades principales que se hacían desde el teléfono, pero ahora desde el reloj. Otro ejemplo podría ser la aplicación de reconocimiento de canciones **Shazam**. Esta aplicación, básicamente, escucha un trozo de canción, lo compara con su base de datos, y muestra el nombre y autor de la canción. Realizar esta acción desde el reloj es mucho más sencillo y rápido que desde el teléfono.

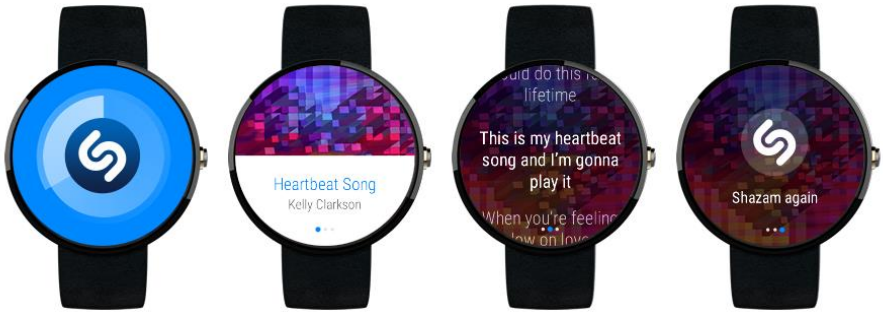


Figura 9 - Shazam en Android Wear

Los smartphones con **Android** se pueden manejar mediante comandos de voz, los smartwatches no iban a ser menos. En los móviles esto se debe configurar, para permitir que esté siempre esperando el comando 'Ok Google', y que funcione también con la pantalla apagada. Pero algunos terminales presentan ciertos problemas con esta funcionalidad y para poder usarlo es necesario presionar un botón. Estos problemas con el reloj no ocurren y es muy cómodo realizar estas búsquedas o acciones sin necesidad de sacar el smartphone del bolsillo. Por ejemplo, se pueden realizar consultas rápidas, enviar mensajes de texto o poner alarmas, usando comandos tan sencillos como estos: 'Ok Google, tiempo en Elche', 'Ok Google, enviar Whatsapp a mamá: llegaré en cinco minutos' o 'Ok Google, despiértame en media hora'.

Figura 10 - LG G Watch usando comandos de voz

Debido a los sensores que algunos de estos wearables disponen, como frecuencia cardíaca, movimiento o GPS, otro uso podría estar relacionado con la salud y la actividad física. Este es el caso de **Runtastic**, esta aplicación, permite establecer una ruta para salir a correr y va guiando al usuario a seguirla, mientras va mostrando datos como velocidad media, tiempo transcurrido, calorías quemadas, ... Desde la aplicación de reloj todos estos datos se pueden comprobar de forma más cómoda. Aplicaciones de este estilo existen varias como por **Google Fit**, **Endomondo** o **RunKeeper**.

Figura 11 - Runtastic en Android Wear

Por último, cabe destacar que los relojes inteligentes también son usados para jugar. La mayoría de estos juegos existentes son minijuegos. Esto es debido en gran parte a sus reducidas dimensiones, ya que, al no contar con una pantalla de mayor tamaño, los controles deben ser sencillos. Aún con esta desventaja existen bastantes juegos publicados en la tienda.

2048 - Android Wear **Math it!** **Wear Droid** **Wear Rider**

Figura 12 - Juegos en Android Wear

3.1.3 Conclusiones

Como se ha podido comprobar las posibilidades que ofrecen estos relojes son inmensas. Se pueden realizar multitud de acciones, desde contestar mensajes o realizar búsquedas mediante comandos de voz, de forma rápida, hasta jugar a algunos juegos, todo ello directamente desde el reloj, sin necesidad de usar el teléfono.

Como ya se ha comentado anteriormente, el proyecto consiste en utilizar el smartwatch como controlador para jugar en el móvil, algo así como si de un mando se tratara. Este uso parece que es bastante novedoso, así que hay que investigar a ver si hay algo parecido; bien sea como plugin/librería para poder usarlo o algún proyecto o aplicación para reloj o teléfono.

3.2 Proyectos similares

3.2.1 El Smartphone como controlador

El objetivo principal del proyecto consiste en realizar un juego o una serie de minijuegos para el móvil y que estos sean controlados mediante un smartwatch. Aunque a la hora de buscar proyectos similares, no es necesario ser tan específico. Para que un proyecto sea similar a este, únicamente, son necesarios dos dispositivos, uno que haga de pantalla y otro que haga de controlador/mando.

Los juegos existentes para **Chromecast**, son un buen ejemplo. Este aparato se conecta a cualquier televisor/monitor por HDMI y desde un teléfono móvil o tablet se envía contenido a la pantalla. Puede usarse para ver películas, visualizar lo mismo que en el dispositivo (mirroring) o jugar a algún videojuego. En el caso de los juegos, la televisión, a través del **Chromecast** sería la pantalla y el teléfono móvil, el mando. Por ejemplo, en el juego **Angry Birds Friends**, el móvil funciona como controlador táctil y se usa para indicar hacia donde se realizará el lanzamiento. Pero ya que el mando se trata de un teléfono móvil, los controles no tienen por qué ser únicamente toques en la pantalla. Este es el caso del **Motion Tennis Cast**, cuyo método de control es el movimiento, para ello se usa los datos de los sensores del teléfono.

Figura 13 - Angry Birds Friends

Figura 14 - Motion Tennis Cast

Otro buen ejemplo que haga algo parecido sería el caso de **AirConsole**[2]**.** En este caso como pantalla se usa el monitor de un ordenador y como controlador un teléfono móvil. Para poder usarlo, únicamente es necesario conexión a internet en los dos dispositivos y acceder desde un navegador web. Para el móvil, también se puede descargar una aplicación, pero esto no es necesario. Existen otras aplicaciones que realizan lo mismo, es decir el móvil como controlador para juegos de ordenador, pero está es la más sencilla y puede funcionar sin necesidad de instalarse aplicaciones.

[2] Página oficial de AirConsole (www.airconsole.com)

La conexión entre ambos dispositivos se realiza, de forma sencilla, a través de un código de configuración. Para jugar, al igual que en los anteriores ejemplos, se emplean tanto controles virtuales como datos del giroscopio del teléfono.

Figura 15 – AirConsole con dos controladores

BT Controller es otra aplicación que sirve como ejemplo; en este caso un móvil o tablet hace de controlador y el otro de pantalla. Este controlador sirve para varios emuladores, un dispositivo ejecuta el emulador y el juego, mientras que el otro únicamente se conecta con el primero y lo controla. Esta aplicación es compatible con los emuladores de **MAME**, **PlayStation 1**, **Nintendo 64** y **SNES**. Esta conexión se realiza a través de Wifi o Bluetooth. Además de los 'mandos' básicos que trae la aplicación por defecto, el usuario puede crear y mapear todos los que desee; usando para ello una imagen de fondo y colocando los botones por toda la pantalla. En este caso los datos de movimiento y giroscopio del terminal no son usados.

Figura 16 - BT Controller

Una vez comentado estos ejemplos donde se usa un dispositivo con **Android** como controlador, es hora de buscar proyectos que usen el smartwatch como controlador. Al buscar algo tan específico, los resultados que aparecen son bastante reducidos. De hecho, solo hay dos proyectos que cumplan estos requisitos.

Uno de ellos es **PrestoAndroidPlugin**, un plugin para **Unity**. El otro es un proyecto que mezcla una aplicación en **Unity** usando **CardBoard** con un smartwatch como controlador.

3.2.2 PrestoAndroidPlugin

Figura 17 - Logo de PrestoAndroidPlugin

PrestoAndroidPlugin es un plugin gratuito desarrollado por la empresa **Presto** que se encuentra de forma gratuita en la tienda de plugins de **Unity**. Este plugin permite que los objetos de Unity sean controlados mediante movimientos o gestos como lanzamientos o chasqueo de los dedos. Para detectar estos movimientos y gestos, se usa un smartwatch con **Android Wear** y luego se comunica con el proyecto de **Unity**.

Este plugin facilita bastante al desarrollador incluir los relojes inteligentes como controladores en sus proyectos. Para hacerlo funcionar en su proyecto debe seguir una serie de sencillos pasos que explican en su página web. El primero de ellos es añadir el plugin al proyecto, es decir, realizar una búsqueda en **Unity Asset Store** e importar el plugin al proyecto. A parte de esto, también es necesario añadir el 'bundle identifier' al proyecto desde los ajustes. Una vez realizado esto, ya estará el plugin añadido correctamente. Ahora solo queda definir los diferentes manejadores de eventos deseados a los diferentes 'gameobjects'.

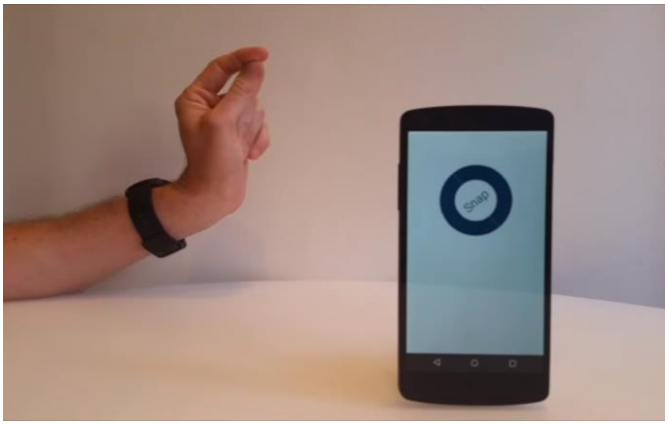


Figura 18 - Proyecto de ejemplo de PrestoAndroidPlugin

Para que este plugin funcione es necesario que el terminal tenga instalada la aplicación '**Presto Unity Controller'**. Esta aplicación es gratuita y se encuentra en el **Play Store**. El funcionamiento del plugin es sencillo. En primer lugar, la aplicación que es necesaria para su uso, es la encargada de comunicarse con el smartwatch. Básicamente, es un servicio que se está ejecutando en segundo plano. Cuando se reconoce algún gesto o movimiento, se comunica con el proyecto en **Unity** para que estos cambios tengan efecto en el proyecto.

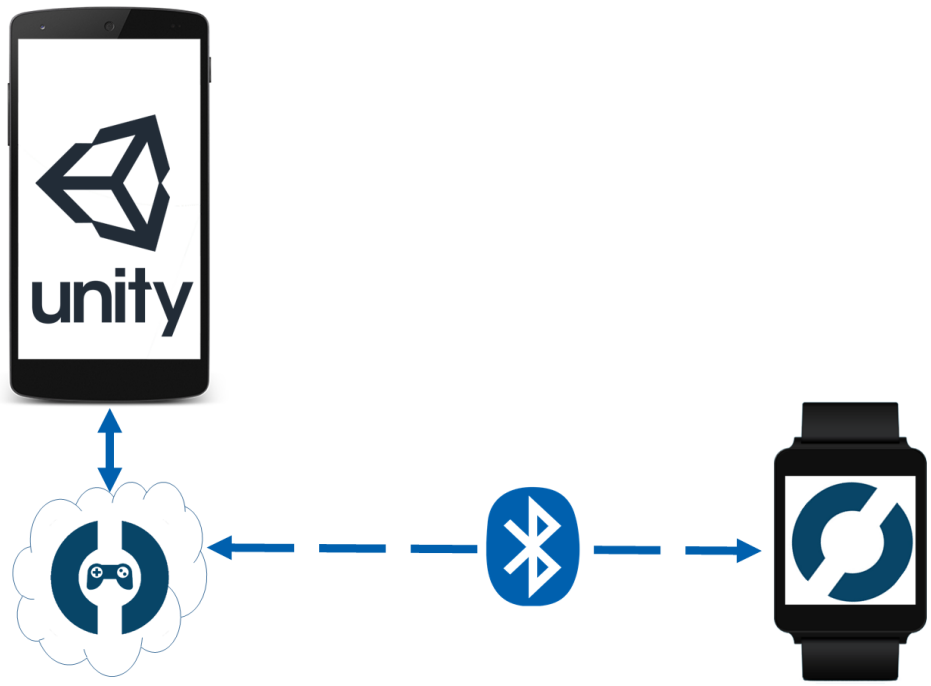


Figura 19 - Funcionamiento sencillo de PrestoAndroidPlugin

Un inconveniente que presenta este plugin es que, al tener un servicio en segundo plano y que está calculando si se produce algún gesto, presenta bastante latencia. Esto en juegos que se requiere una respuesta inmediata es un gran impedimento. Por ejemplo, si el proyecto se tratará de un escenario virtual haciendo uso de gafas de realidad virtual, este retraso podría ser permisible.

Otro inconveniente a destacar es la necesidad de instalarse otra aplicación. Cuando creas un proyecto con Unity usando este plugin, para poder usarlo, debes instalar la aplicación comentada anteriormente. Esto es una gran desventaja ya que cuando subas tu aplicación a la tienda, para que la puedan disfrutar los usuarios, deben bajarse la otra aplicación también.

3.2.3 Proyecto parecido

Damian Mehers, ha creado un proyecto parecido al que se desea implementar. Este proyecto consiste en una prueba usando un reloj **Android Wear** como controlador, con **Google CardBoard**, usando como motor de videojuegos **Unity**. Básicamente, este proyecto consiste en usar los datos del giroscopio del reloj para mover un objeto del motor **Unity**. Pero todo esto, se trata únicamente de una prueba donde el autor ha documentado toda la creación en su blog[3].

El autor explica desde la creación de la librería usando **Android Studio**, hasta la integración del plugin en **Unity**. A la hora de crear la librería, exporta el proyecto como ficheros aar; ya que **Unity** soporta estos ficheros como librerías precompiladas de Java para crear plugins nativos para **Android**.

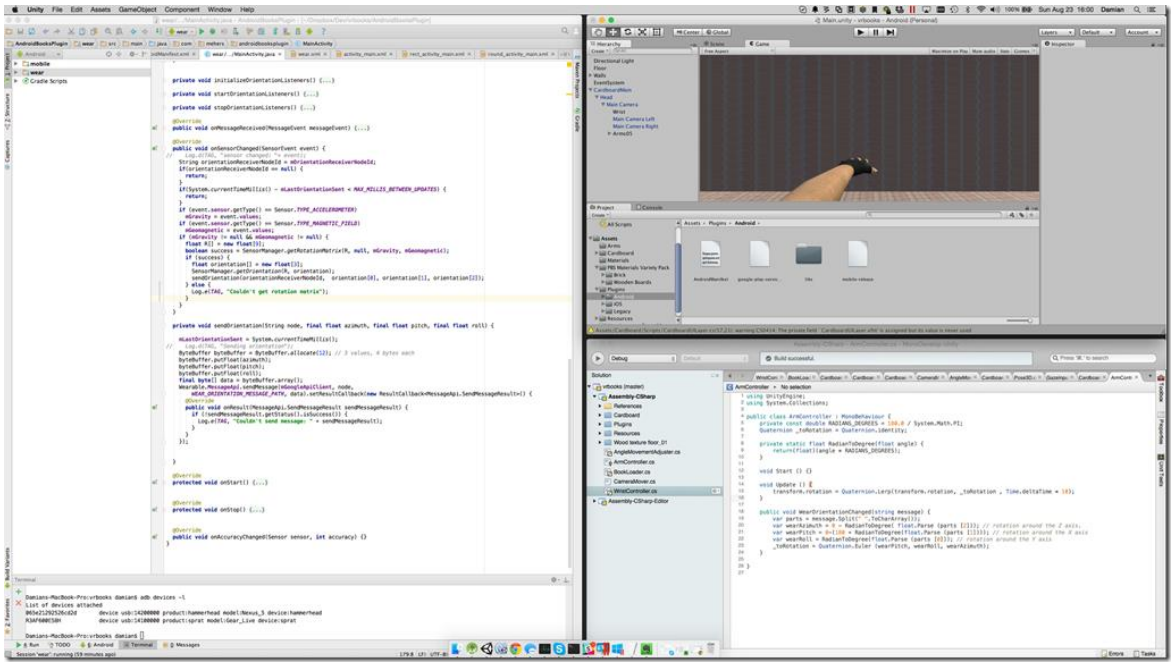


Figura 20 - Espacio de trabajo del proyecto similar

En la entrada de este blog, se comenta la mayoría del código de este proyecto. Al principio se explica cómo es la creación de los plugins nativos de Unity. Más adelante, explica el código del reloj; es decir, la obtención de los datos del reloj y la comunicación con el teléfono. Después, comenta como conectar el teléfono con la aplicación de Unity. Por último, en un proyecto de Unity, se usan estos datos para modificar la rotación de un cubo y luego este objeto se cambia por un modelo en 3D de un brazo.

[3] Entrada en el blog de Damien Mehers donde habla del proyecto realizado: http://damianblog.com/2015/08/23/android-wear-arm-in-cardboard-vr/

El proyecto, como comenta el autor, posee un poco de retraso entre que el reloj envía los datos y se ven reflejados en el proyecto. Este retraso comentado, así como el proyecto se puede comprobar en un vídeo de **Youtube**[4] subido por el propio autor. Además, añade que, por culpa de este retraso, para videojuegos, este plugin puede no ser muy útil, pero para otras experiencias puede funcionar correctamente.

Figura 21 - Proyecto similar de Unity en funcionamiento

Al tratarse de una prueba, no se comenta como se firmaría y empaquetaría la aplicación usando **Unity**. Pero es una buena referencia para entender como es el proceso de creación de plugins, así como la comunicación entre ambos dispositivos y proyectos.

[4] La imagen que aparece es la miniatura del vídeo de **Youtube**, donde se puede ver el proyecto en funcionamiento: https://www.youtube.com/watch?v=8ywn21ValCA

3.2.4 Conclusiones

Como se puede comprobar el proyecto que se pretende crear no es pionero, ya que existen algunos ejemplos bastante parecidos. Aunque son muy similares, existe una gran diferencia entre estos y la librería que se desea crear. Estos ejemplos son muy concretos, es decir, no se pueden usar en muchos ejemplos. La librería que se desea crear, se pretende que tenga varias funcionalidades para que se pueda adaptar a diferentes juegos o experiencias distintas.

Por otro lado, el primer plugin analizado tiene un grave problema, para poder utilizarlo es necesario descargarse una aplicación del **Play Store**. El otro ejemplo, al tratarse de una prueba no se ha llegado a completar el proyecto hasta firmarlo y subirlo a la tienda. Es posible que **Unity** presente problemas al realizar estas acciones, por eso ninguno de estos proyectos lo realiza. Además, ambos proyectos presentan retrasos entre que el reloj realiza una acción hasta que esta se ve reflejada en el reloj. Cuando se desarrolle el proyecto, se estudiarán estos problemas para intentar solucionarlos.

Como se puede apreciar, ambos proyectos presentan aspectos buenos y malos. El segundo proyecto ofrece el código del mismo que será de gran ayuda a la hora de desarrollar el propio proyecto.

4 Metodología

En este apartado, se detallan las técnicas y las herramientas empleadas para llevar a cabo el proyecto.

4.1 Flujo de trabajo

En primer lugar, como se ha comentado anteriormente, antes de empezar la implementación, se realizarán un par de análisis previos. El primero de ellos consiste en estudiar los principales usos que actualmente tiene un smartwatch. El otro análisis consiste en estudiar los plugins y/o proyectos que conecten **Unity** con el smartwatch. De estos análisis se obtendrán unas conclusiones que serán tomadas en cuenta cuando se empiece a desarrollar el plugin.

Cuando se hayan realizado los análisis previos, se empezará creando pequeños proyectos y plugins de prueba. Primero se crearán pruebas para probar la conexión y comunicación entre el smartwatch con el dispositivo móvil. Una vez realizadas estas pruebas se realizarán plugins sencillos en **Unity**, para ver cómo es el funcionamiento de estos. Finalmente, cuando estas pruebas estén realizadas, se creará el plugin que conecte **Unity** con el smartwatch.

Una vez realizado el plugin, se creará el proyecto final. Se irán desarrollando los mini-juegos por separado, para que sea más sencillo su prueba y depuración. Finalmente, cuando estén creados se juntarán en una aplicación final.

4.2 Herramientas utilizadas

Figura 22 - Logos de las herramientas utilizadas

Para la realización de este proyecto se utilizan las siguientes herramientas/programas:

- **Unity 3D:** este es el motor de videojuegos que se va a utilizar para realizar el proyecto. Se creará un plugin para este motor y después el juego principal.
- **Android Studio:** este IDE se usa para la creación de aplicaciones para Android. Concretamente para este proyecto se usará para la creación y compilación del plugin para el motor **Unity**.
- **Adobe Photoshop:** este programa se emplea para crear y/o modificar las diversas imágenes/texturas que se usarán en el proyecto.
- **Notepad++:** este editor de texto gratuito se usará para cambiar ficheros de configuración de forma sencilla. Este editor es muy ligero y rápido, además es bastante útil ya que remarca en diversos colores algunas etiquetas de los archivos utilizados.
- **Microsoft Office Word:** este procesador de textos se utiliza para crear la documentación/memoria del proyecto.
- **Google Chrome:** este navegador web se usará principalmente para la búsqueda de información en las diversas páginas web.
- **Dropbox:** este programa se emplea para tener una copia tanto de la memoria como de los archivos subidos en la nube y poder llevar un control de versiones.
- **GitHub**: al final del proyecto se ha empezado a usar este servicio, ya que permite llevar un mejor control de versiones.

4.3 Dispositivos utilizados

Para realizar todas las pruebas de las diferentes versiones de las aplicaciones, durante el desarrollo del proyecto, se dispone de dos dispositivos, un teléfono móvil y un reloj inteligente.

El proyecto se va a realizar para sispositivos móviles que posean como sistema operativo Android. El **Motorola Moto G 4G**, es el smartphone elegido para realizar las pruebas; su sistema operativo es **Android 5.1** (Lollipop), las características de este terminal son suficientes para poder realizar los proyectos deseados.

Figura 23 - Motorola Moto G 4G

El smartwatch que se empleará es el **LG G Watch R**, posee una forma redondeada con una pantalla P-OLED de 1.3 pulgadas. El sistema operativo que tiene instalado es la última versión de **Android Wear** (6.01). Además, este reloj cuenta con diversos sensores como medidor de pasos, frecuencia cardíaca, giroscopio y acelerómetro.

Figura 24 - LG G Watch R

5 Desarrollo del proyecto

5.1 Introducción

En primer lugar, se empezará creando un par de plugins nativos para **Unity**. De esta forma se podrá ver cómo es la creación y funcionamientos de los mismos. Primero, se creará un primer proyecto para ver la configuración del proyecto, la creación de la librería **Android** y la ejecución en **Unity**. Después continuando este proyecto, se modificará para probar la comunicación bidireccional entre una clase propia de **Java** y un objeto de **Unity**.

A parte de los proyectos de prueba realizados para **Unity**, se realizarán proyectos para **Android Wear,** usando el IDE **Android Studio**. El primero de ellos, consiste en crear un proyecto básico y conseguir instalarlo tanto en el teléfono como en el reloj. Después de realizar este proyecto se creará otro para probar la comunicación entre ambos dispositivos.

Una vez finalizados estos proyectos de prueba, se empezará a crear el plugin para **Unity**. Primero se creará una aplicación auxiliar, que se utilizará para probar de una forma, más o menos, gráfica la librería creada. Una vez creada esta aplicación, se probará como es la publicación de aplicaciones para smartwatches en el **Play Store**.

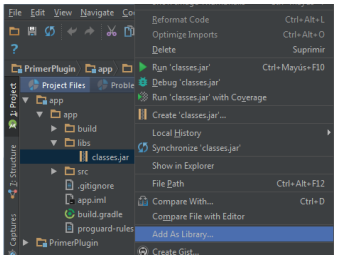
Por último, cuando esta aplicación esté creada y subida, se realizarán los cambios necesarios para convertirlo en plugin para **Unity**. Además, se crearán diversas minijuegos, a modo de prueba, para comprobar el correcto funcionamiento del plugin. Este videojuego, igual que la otra aplicación, será subido a la tienda de aplicaciones.

5.2 Creación de las pruebas de los plugins de Unity

5.2.1 Creando el primer plugin de Unity

Para crear un plugin nativo de Unity para Android, lo primero que se debe hacer es crear un proyecto de **Android** usando Android Studio. Al crearlo usando este IDE, se generan todas las carpetas y recursos necesarios para las aplicaciones de **Android**. Con el proyecto creado, se deben realizar una serie de cambios.

Lo primero que se debe hacer es incluir las clases de Unity en el proyecto. Para ello, se debe buscar la librería precompilada de Unity para Android. Este archivo, llamado 'classes.jar', se encuentra ubicado en **Windows** dentro de este directorio "ProgramFiles/Unity/Editor/Data" y en **MAC** "/Applications/Unity". Dentro del directorio en la subcarpeta "PlaybackEngines/AndroidPlayer/Variations/mono/Release/Classes/".



"Una vez localizado este archivo, se debe copiar al directorio 'app/libs' del proyecto de **Android**. Una vez copiado, para añadirlo al proyecto desde **Android Studio** se debe seleccionar este archivo y seleccionar la opción de añadir como librería al proyecto.

Figura 25 - Añadiendo la librería al proyecto de Android Studio

A la hora de crear un plugin para **Unity**, se pueden crear dos tipos de clases Java. Las clases normales, las cuales se crean/instancian desde Unity. Las clases que heredan de la actividad principal de **Unity**. Para este caso, se va a utilizar una clase que herede de la actividad principal de Unity. De esta forma no es necesaria la creación de una instancia desde **Unity**, ya que cuando se cree la aplicación, esto se hará de forma automática.

En la clase 'MainActivity' del proyecto se deben hacer una serie de cambios. El primero de ellos consiste en modificar la actividad de la que hereda. Por defecto aparece 'AppCompatActivity', se debe cambiar por 'UnityPlayerActivity'. Además de este cambio, se debe eliminar/comentar la línea donde se establece la vista (setContentView). Para probar el correcto funcionamiento del plugin, se crea un mensaje (Toast) en pantalla.

```
...
public class MainActivity extends UnityPlayerActivity
{
  @Override
  protected void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
```

```
    //setContentView(R.layout.activity_main);
    createToast("Parece que funciona :) ");
  }

  private void createToast(final String txt)
  {
    MainActivity.this.runOnUiThread(new Runnable()
    {
      @Override
      public void run()
      {
        Toast.makeText(MainActivity.this, txt, Toast.LENGTH_LONG).show();
      }
    });
  }
}
```

Figura 26 - MainActivity del primer plugin para Unity

Una vez realizados los cambios necesarios en el código, se debe especificar que el proyecto no se trata de una aplicación sino de una librería. **Unity** permite que los plugins creados estén como librerías precompiladas (jar) o librería de **Android** (aar). Ya que, para este proyecto, solo son necesarias las clases **Java** y ningún otro recurso de **Android**, se creará una librería jar. Para ello, se deben realizar modificaciones en el fichero 'build.gradle' del módulo app:

- Cambiar la primera línea "`apply plugin: 'com.android.application'`" por: "`apply plugin: 'com.android.library'`". De esta forma se indica que lo que se va a crear es una librería y no una aplicación.
- Eliminar los atributos de "`applicationId`", "`versionCode`" y "`versionName`" de la configuración por defecto. Estos atributos se deben especificar cuando se crean aplicaciones.
- Añadir dos tareas de **Graddle** para que se genere la librería jar, una para exportar y otra para eliminar la última versión generada.

```
apply plugin: 'com.android.library' //'com.android.application'

android
{
    compileSdkVersion 24
    buildToolsVersion "24.0.0"

    defaultConfig {
        //applicationId "juanpomares.tfm.mastermoviles.primerplugin"
        minSdkVersion 21
        targetSdkVersion 24
        //versionCode 1
        //versionName "1.0"
    }
```

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.1.1'
}

//task to delete the old jar
task deleteOldJar(type: Delete) { delete 'release/AndroidPlugin.jar' }

//task to export contents as jar
task exportJar(type: Copy) {
    from('build/intermediates/bundles/release/')
    into('release/')
    include('classes.jar')
    rename('classes.jar', 'AndroidPlugin.jar')
    ///Rename the jar

}

exportJar.dependsOn(deleteOldJar, build)
```

Figura 27 - Fichero build.gradle modificado para generar la librería

Una vez realizadas las modificaciones necesarias en este fichero, ya se puede generar la librería jar. Al ejecutar la 'task' exportjar, la librería se generará y se creará el fichero 'AndroidPlugin.jar' en la carpeta 'app/release'.

Añadir este plugin creado a Unity es bastante sencillo. Primero se debe especificar el 'bundle identifier' desde los ajustes para **Android**. Este es el identificador de la aplicación y debe usarse el mismo nombre de paquete que la librería creada. Además, la versión de la **API** de **Android**, que aparece abajo, debe ser igual o superior a la está especificada e librería.

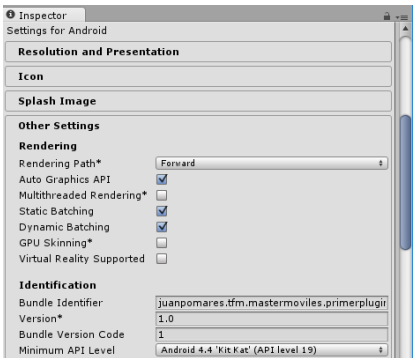


Figura 28 - Especificando el identificador de aplicación en Unity

Dentro de la carpeta 'Assets' del proyecto, se debe crear una carpeta llamada 'Plugins' y dentro de esta, otra llamada 'Android'. La librería anteriormente creada (archivo jar) se debe copiar dentro de la carpeta Android. También, se debe crear un fichero

AndroidManifest.xml, donde se indica que la actividad principal es la clase de la librería creada.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="juanpomares.tfm.mastermoviles.primerplugin">
    <uses-sdk android:minSdkVersion="19" />
    <application android:label="@string/app_name">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Figura 29 - Manifest empleado en el primer plugin creado para Unity

Con todos estos cambios, ya se puede generar la aplicación y probar, en un teléfono móvil o emulador, si funciona todo de forma correcta.

Figura 30 - Probando el primer plugin nativo creado para Unity

5.2.2 Estudiando la comunicación entre Android y Unity

Una vez creado y configurado el primer plugin, se puede usar el código del mismo como base para la siguiente prueba. En esta prueba, se pretende ver como es la comunicación entre el código de la librería creada con **Unity**. En el anterior plugin, el método de creación del mensaje (Toast) se llamaba implícitamente al crearse la aplicación; en es este nuevo plugin, esa llamada y otras se realizarán explícitamente desde **Unity**.

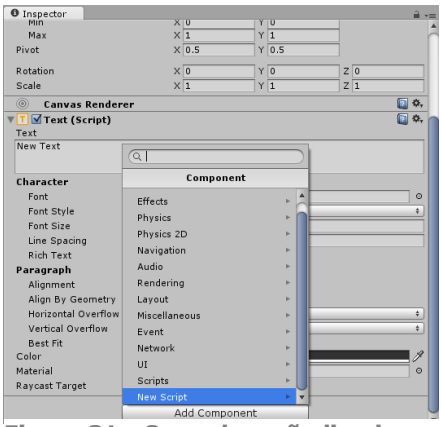


Figura 31 - Creando y añadiendo un script a un objeto en Unity

En primer lugar, debemos construir una escena simple en Unity. Para mostrar los resultados de las diferentes llamadas que se realicen, a modo de log, se utilizará un TextView de **Unity**. A este texto creado se le debe añadir un componente script. Para ello se debe seleccionar añadir componente y luego nuevo script. Este script será el encargado de invocar a los métodos de la librería.

Antes de crear el código de este script, se creará el código del fichero de la librería. En la clase principal se añadirán una serie de métodos simples que luego serán llamados desde el nuevo script creado.

```
...
public class MainActivity extends UnityPlayerActivity
{
  public static Context myActivity;
  public String myString="Test String";

  private float[] myValues={0, 2.5f,-3};

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    //setContentView(R.layout.activity_main);
    myActivity=this;
  }

  public String getMyString(){return myString;}

  public void ChangeMyValues(float dif)
  {
    for (int i=0; i<myValues.length; i++) myValues[i]+=dif;
  }
```

```
  public float[] getMyValues(){ return myValues; }
  public void TestSendMessage(String idObjt, String method){
    NotifyObject(idObjt, method, "Sended message from Android To Unity");
  }

  private void NotifyObject(String idObject, String method, String message){
    UnityPlayer.UnitySendMessage(idObject, method, message);
  }
}
```

Figura 32 - MainActivity del nuevo plugin de Unity

Como se puede comprobar es la misma clase que en el primer plugin creado, pero con modificaciones. Se han añadido varias variables, las cuales se obtendrán y modificarán desde el script de **Unity**. Además de las variables, se han creado algunos métodos simples; algunos de estos métodos y variables son públicos para que desde **Unity** se pueda acceder a ellos. El método 'NotifyObject', es el encargado de enviar un mensaje a un objeto de **Unity**. En este caso, notifica al objeto con el nombre pasado como parámetro al método elegido un mensaje predefinido. Como se puede comprobar, realizar la comunicación es muy sencillo; sin embargo, es la única forma de comunicarse y sólo admite enviar cadenas de texto (String). Si se deseará enviar un número o un vector de números, éstos deben convertirse a cadena.

Con el código modificado de la librería, ya es posible la creación del script de **Unity**; porque ya se conocen los métodos que la componen. Desde el script se van a invocar estos métodos creados, así como obtener y modificar algunas variables.

```
public class TextScript : MonoBehaviour
{
   public Text mytext;
   #if UNITY_ANDROID && !UNITY_EDITOR
   AndroidJavaObject currentActivity;
   #endif

   void Start (){
      mytext.text = "";
      appendText ("Begin method start of TextScipt");
      #if UNITY_ANDROID && !UNITY_EDITOR

      AndroidJavaClass AndroidClas;//unityAndroidClass;
   AndroidClas=new AndroidJavaClass("juanpomares.tfm.mastermoviles.primerplugin.MainActivity");

      if(AndroidClas!=null){
         currentActivity= AndroidClas.GetStatic<AndroidJavaObject> ("myActivity");

         if(currentActivity!=null) {
             string mystr=currentActivity.Get<string>("myString");
             appendText ("My String: '"+mystr+"'");
             appendText ("Changing my String...");

             currentActivity.Set<string>("myString", "..Unity changed String");
             mystr=currentActivity.Call<string>("getMyString");
```

```
            appendText ("My String changed: '"+mystr+"'");
            string vals=getStringFromArray(currentActivity.Call<float[]>("getMyValues"));

            appendText ("My values: "+vals);
            appendText ("Changing my values adding 5.1...");
            currentActivity.Call("ChangeMyValues", 5.1f);
            vals=getStringFromArray(currentActivity.Call<float[]>("getMyValues"));
            appendText ("My changed values: "+vals);

            Invoke("invokedmethod", 5);
        }
    }
    #endif
    appendText ("End method start of TextScipt\n");
  }

  string getStringFromArray(float[] values){
    string returning="[";
    for(int it=0; it<values.Length; it++)
       returning+=values[it].ToString()+" ";
    returning+="]";

    return returning;
  }

  void invokedmethod(){
    #if UNITY_ANDROID && !UNITY_EDITOR
    appendText ("Calling Android to notify Unity....");
       if(currentActivity!=null)
           currentActivity.Call("TestSendMessage", transform.name, "appendText");
    #endif
  }

  void appendText(string str) {
    string theTime = System.DateTime.Now.ToString("hh:mm:ss: ");
    mytext.text+=theTime+str+"\n";
  }
}
```

Figura 33 - Script asignado al texto de Unity

Esta clase, 'TextScript', que hereda de 'MonoBehaviour', es el nuevo script que se ha añadido al texto de la escena. Como variables de clase, posee el texto, que se usará a modo de log y una referencia al 'MainActivity', para poder invocar sus métodos. La variable del texto, se debe establecer desde el editor; la referencia se inicializan el método 'Start'. Para inicializarla, primero se instancia la clase y después se accede a la referencia estática de la actividad principal. Además, en este método, se hacen las diversas llamadas a los métodos creados anteriormente en la librería. Como se puede comprobar, se pueden obtener y modificar algunas variables, directamente accediendo a los métodos 'Get' y 'Set' o accediendo a métodos propios.

El método auxiliar 'appendText' es el encargado de añadir el mensaje pasado por parámetro al texto; también se añade la hora a la cual este texto se ha introducido. El otro método auxiliar es el encargado de transformar el array de números en una cadena de texto.

Por último, el método 'invokemethod', se encarga de llamar al método creado en la librería que se encargaba de pasarle un mensaje a un objeto de **Unity**. En este caso, se le indica que el identificador es el objeto que tenga este script (usando transform.name). A parte de esto, el método encargado de recibir este mensaje es el que se encarga de añadir texto en pantalla ('appendText'). De esta forma, cuando se reciba el mensaje, se verá instantáneamente en la pantalla. Desde el método 'Start', no se llama a este método, se llama al 'Invoke'. Este método invocará el método una vez haya transcurrido un tiempo especificado, en este caso, cinco segundos.

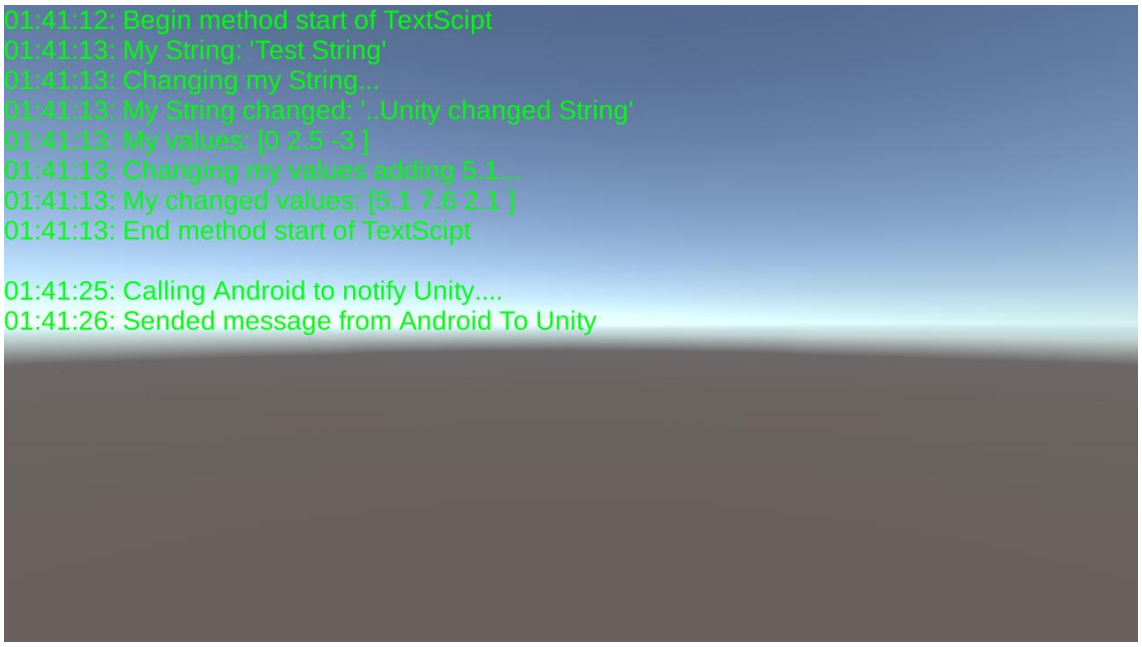


Figura 34 – Probando el plugin de Unity de la comunicación bidireccional

Como se puede comprobar la creación de plugins nativos para Unity, así como la comunicación bidireccional entre Unity y la librería creada son bastante sencillos.

5.3 Creación de los proyectos iniciales de Android Wear

5.3.1 Creando y empaquetando la primera aplicación en Android Wear

En primer lugar, se creará nuestro primer proyecto para **Android Wear**. No solo se realizará la creación de esta aplicación, se verá cómo se empaquetan de las aplicaciones para **Android Wear**. Una vez se haya completado, se creará un proyecto para ver cómo realizar una comunicación bidireccional, entre el smartwatch y el Smartphone, usando 'MessageAPI'.

Para crear este primer proyecto desde **Android Studio**, se debe seleccionar nuevo proyecto. Como en las aplicaciones normales de **Android**, se debe establecer el nombre de la aplicación, así como el nombre del paquete. Una vez seleccionado esto, se debe establecer para que dispositivos se creará la aplicación, en esta ocasión, se deben seleccionar 'Phone and tablet' y 'Wear'. Al seleccionar esto, ya se nos crea toda la estructura de directorios y recursos propios de **Android** de las dos aplicaciones.

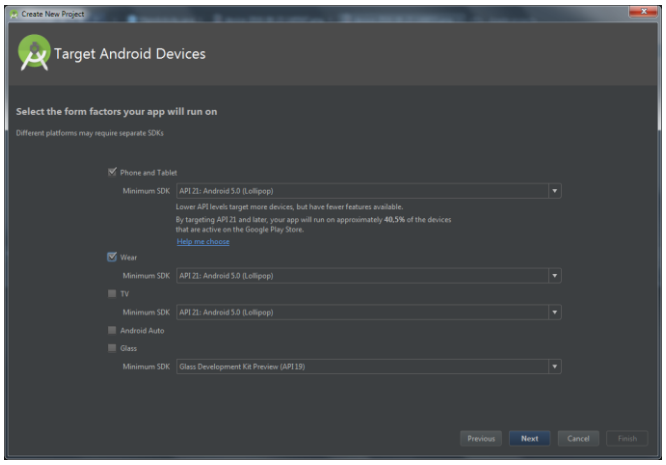


Figura 35 - Seleccionando las plataformas en Android Studio

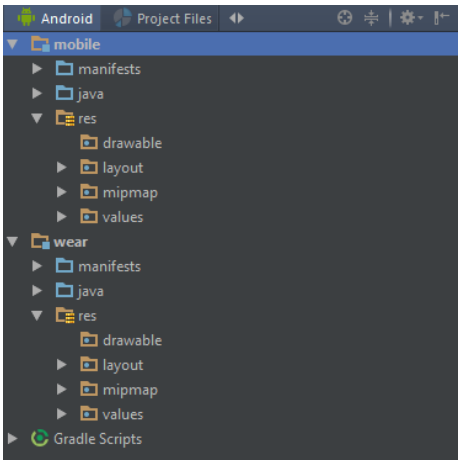


Figura 36 - Estructura de módulos y directorios generados mediante Android Studio

Como se puede comprobar, **Android Studio** trata las aplicaciones como dos módulos por separado. De esta forma, es más fácil de probar, ya que se puede únicamente ejecutar la aplicación que se desee. Cuando se desee ejecutar o firmar la aplicación, se debe selecciona un módulo de ambos (wear o mobile).

Para conseguir empaquetar las aplicaciones en un mismo fichero, se deben seguir una serie de pasos. Cuando se sube una aplicación al **Play Store**, se debe adjuntar solamente un fichero APK; por eso es muy importante realizar este empaquetado.

El primer paso, consiste es compilar y firmar la aplicación/módulo del reloj. Para ello, se hace como si de una aplicación normal de Android se tratara; primero, se selecciona 'Build' → 'Generate Signed APK…' y después se indica que se desea el módulo wear.

Una vez hecho esto, se debe introducir la clave y el usuario con el que se desea firmar esta aplicación. Se puede usar una almacenada en el 'keystore' o crear una nueva. En estos almacenes de claves (keystore), se permite guardar diversos alias con los que firmar las aplicaciones.

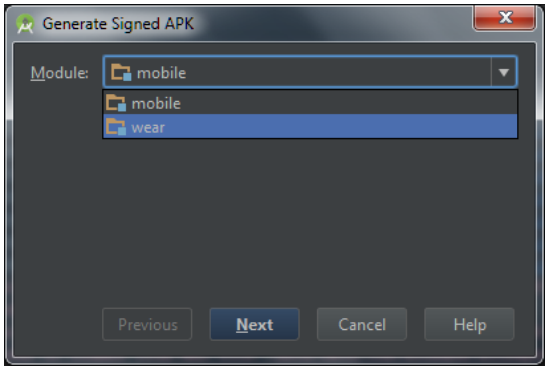


Figura 37 - Seleccionando el módulo deseado para firmar la aplicación

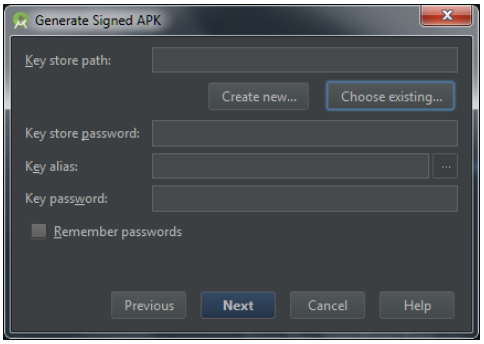


Figura 38 - Seleccionando las claves para firmar la aplicación

Una vez compilada y firmada la aplicación del reloj, debemos añadirla a la aplicación del teléfono. Esto se puede realizar a través de Gradle o una forma un poco más manual. En esta ocasión se usará la segunda opción; ya que ofrece más versatilidad. Para ello se deben crear dos directorios 'raw' y 'xml' en la carpeta 'res' del módulo 'mobile'. En la carpeta 'raw', se debe copiar la aplicación del reloj firmada y en el directorio 'xml', un archivo llamado 'wearable_app_desc.xml'. En este fichero se especifica el paquete, la versión y el recurso de la aplicación del smartwatch. La aplicación del Smartphone, debe tener estos valores iguales; si esto no se cumple, la aplicación no se instalará en el reloj.

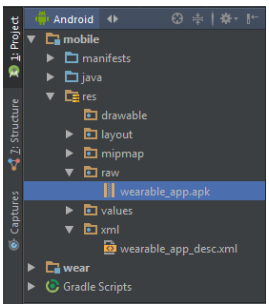


Figura 39 - Nuevos ficheros y directorios añadidos al proyecto

```
<wearableApp
package="juanpomares.tfm.mastermoviles.firstsmartwatchapp">
   <versionCode>1</versionCode>
   <versionName>1.0</versionName>
   <rawPathResId>wearable_app</rawPathResId>
</wearableApp>
```

Figura 40- Fichero smartwatch_app_desc.xml

Con estos ficheros y directorios creados, es necesario hacer referencia a ellos desde el Manifest de la aplicación. Únicamente es necesario añadir un metadato dentro de la aplicación. Como nombre del metadato se puede usar el paquete de la aplicación y como recurso una referencia al fichero XML creado.

```
 <?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="juanpomares.tfm.mastermoviles.firstsmartwatchapp">

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

     <meta-data android:name="juanpomares.tfm.mastermoviles.firstsmartwatchapp"
            android:resource="@xml/smartwatch_app_desc"/>

        <activity android:name=".MainActivity">
...
```

Figura 41 - Manifest de la aplicación con el metadata añadido

Con estos cambios ya se puede firmar la aplicación y probarla. Es posible que a la hora de generarla, aparezca un problema relacionado con el tamaño máximo de referencias en algunos ficheros con extensión '.dex'. Para solucionarlo, en el fichero 'build.gradle' del módulo 'mobile', el que se ha intentado generar, se debe añadir: "multidexEnabled true". De esta forma, a la hora de compilar, estos ficheros se dividen en varias partes para no exceder el tamaño máximo permitido.

Figura 42 - Error relacionado con el tamaño máximo de los ficheros DEX

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 24
    buildToolsVersion "24.0.0"
    defaultConfig {
        // Enabling multidex support
        multiDexEnabled true

        applicationId 'juanpomares.tfm.mastermoviles.firstsmartwatchapp'
        minSdkVersion 21
...
```

Figura 43 - Build.gradle con la solución al error del tamaño de los ficheros DEX

Una vez solucionado este error, ya se puede generar y firmar sin problema la aplicación. Para probar si funciona se debe instalar este fichero APK en el smartphone. Una vez instalado en el reloj, aparecerá una notificación en el smartwatch indicando que se ha instalado una nueva aplicación.

```
C:\Users\Usuario>cd AppData\Local\Android\sdk\platform-tools

C:\Users\Usuario\AppData\Local\Android\sdk\platform-tools>adb install "C:\Users\Usuario\FirstSmartwatchApp\mobile-release.apk"
[100%] /data/local/tmp/mobile-release.apk
        pkg: /data/local/tmp/mobile-release.apk
Success

C:\Users\Usuario\AppData\Local\Android\sdk\platform-tools>
```

Figura 44 - Instalando la nueva aplicación a través de la consola usando ADB

Figura 45 - Aplicación instalada en el reloj

Ahora que ya están instaladas las aplicaciones en ambos dispositivos, se pueden abrir para ver que funcionan. Este proyecto de ejemplo instala dos aplicaciones simples, que al abrirlas muestran un mensaje de texto. En el caso del Smartphone, muestra 'Hello World!', mientras que cuando el dispositivo es un en el reloj con pantalla redonda 'Hello Round World!' y si se trata de un reloj cuadrado: 'Hello Square World!'.

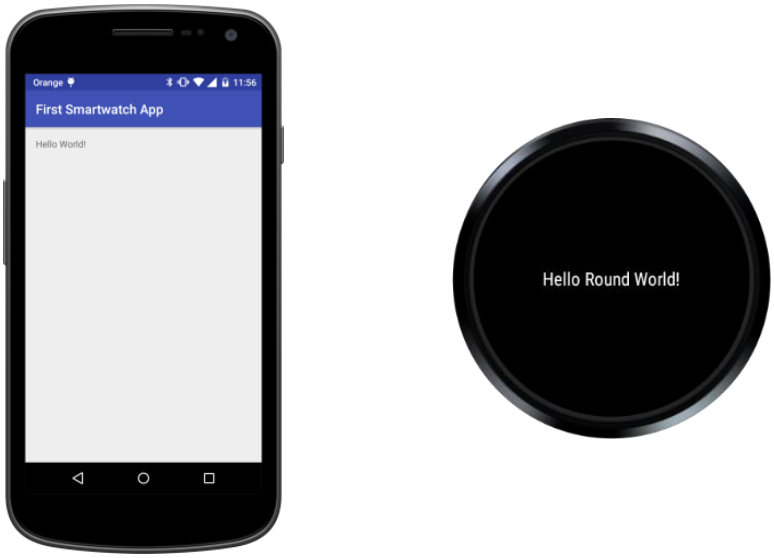


Figura 46 - Probando la aplicación creada en el teléfono y reloj

5.3.2 Creando comunicación bidireccional entre reloj y smartphone

En este proyecto, se pretende ver como se realiza una comunicación entre ambos dispositivos. Existen dos formas para conseguir esta comunicación, usando 'DataAPI' o 'MessageAPI'. La primera forma no es instantánea, ya que está pensada para transmitir gran cantidad de datos; mientas que la segunda forma sí, ya que se limita únicamente a comunicar un mensaje. Por ejemplo, si la aplicación deseada, se encarga de intercambiar muchos datos entre ambos dispositivos y no se requiere una respuesta inmediata; es mejor utilizar 'DataAPI'. Para este ejemplo, ya que solo se pretende el paso de mensajes se utiliza 'MessageAPI'.

Tanto si se desea usar una forma o la otra, lo primero que se debe realizar es crear un cliente de **Google**, añadir los callbacks de conexión, fallo y añadir 'WearableAPI'. Una vez se haya creado, se debe llamar al método para realizar la conexión. Si se conecta o existe algún error, se llamará a los métodos que se hayan definido en los callbacks. Esta inicialización se puede crear en un método propio y llamarlo cuando se cree la aplicación (método 'onCreate'). Todo esto debe ir tanto en la aplicación del teléfono como la del reloj.

```
private void initGoogleApiClient() {
    createConnectionCallbacks();
    createConnectionFailed();

    mApiClient = new GoogleApiClient.Builder( this )
      .addApi( Wearable.API )
      .addConnectionCallbacks(mConnectionCallbacks)
      .addOnConnectionFailedListener(mConnectionFailedListener)
      .build();
    mApiClient.connect();
}
```

Figura 47 - Iniciando el cliente de Google añadiendo WearableAPI

Cuando se produce no se puede conectar de forma correcta el 'GoogleApiClient', se muestra un mensaje indicando que ha ocurrido un error. Si, por el contrario, se ha conectado de forma satisfactoria, se inicializa y asigna el listener para recibir los mensajes. Por último, se obtiene el dispositivo al que se le enviarán los mensajes más adelante.

```
private void createConnectionFailed() {
    mConnectionFailedListener=new GoogleApiClient.OnConnectionFailedListener(){
        @Override
        public void onConnectionFailed(@NonNull ConnectionResult connectionResult){
          Toast.makeText(MainActivity.this, "Ha ocurrido un error :0", Toast.LENGTH_LONG).show();
        }
    };
}
```

Figura 48 - Iniciando el listener del fallo de conexión

```
 private void createConnectionCallbacks()
{
     mConnectionCallbacks=new GoogleApiClient.ConnectionCallbacks()
     {
        @Override
        public void onConnected(Bundle bundle)
        {
            initMessageListener();
            Wearable.MessageApi.addListener(mApiClient, mMessageListener);
           (new GetSendDevice()).start(); //Get the node of the device
        }

        @Override
        public void onConnectionSuspended(int i) {Log.d("onConnectionSuspended", "Error: "+i);}
     };
 }
```

Figura 49 - Iniciando los callbacks de conexión

Para poder recibir los mensajes que envíe el otro dispositivo, se debe crear un 'MessageListener'. Se debe implementar el método 'onMesaggeReceived', este método recibe un 'MessageEvent' con datos interesantes sobre el mensaje recibido. De este método se puede obtener, el identificador del dispositivo que lo ha enviado, el 'path' del mensaje, así como los bytes del mensaje. Si se sabe que siempre se enviarán cadenas de texto, se puede convertir estos bytes de forma sencilla a 'strings'. En este ejemplo, cuando un mensaje se reciba se mostrará por pantalla.

```
  private void initMessageListener()
  {
       mMessageListener=new MessageApi.MessageListener()
       {
           @Override
           public void onMessageReceived(MessageEvent messageEvent)
           {
               String path = messageEvent.getPath();
               byte[] data = messageEvent.getData();
               String sData=new String(data);
               Log.d("onMessageReceived", "p: " + path + " d: " + sData);
           }
       };
   }
```

Figura 50 - Creando el listener para recibir mensajes

Para obtener el dispositivo al que se le enviarán los datos, se debe hacer en un hilo paralelo al principal. Debido a que la llamada para obtener los dispositivos conectados, se realiza de forma asíncrona, no se puede realizar en el hilo principal. Cuando se obtienen los dispositivos conectados, si hubiera más de un dispositivo conectado, se debería dar la posibilidad al usuario de elegir con cuál desea establecer la conexión. Sin embargo, ya que esto es un ejemplo simple, se seleccionará por defecto el primero. Si no existiera ningún dispositivo conectado, se cerraría la aplicación y se mostraría un mensaje por consola.

```
private class GetSendDevice extends Thread
{
  public void run()
  {
   NodeApi.GetConnectedNodesResult nodes = Wearable.NodeApi.getConnectedNodes( mApiClient ).await();
    List<Node> list=nodes.getNodes();

    if(list.size()>0) mSendDevice=list.get(0);

    if (mSendDevice == null)
    {
        MainActivity.this.finish();
        Log.d("GetSendDevice", "No device connected");
    }
  }
}
```

Figura 51 - Obteniendo el dispositivo con el que realizar la comunicación

Para el envío de mensajes, se ha creado una clase propia para realizarlo. En este caso, como se desea comprobar el estado del mensaje, es decir si ha llegado correctamente a su destino, no se puede hacer a través del hilo principal. Cuando se crea la clase, se inicializan sus variables, el 'path' y los bytes a enviar, en este caso son cadenas de texto. Cuando se ejecuta, se envía el mensaje a través del 'MessageAPI' y se comprueba si ha llegado correctamente a su destino. Si ha habido algún problema durante el envío del mismo, se muestra un mensaje por la consola.

```
  private void sendMessage(String path, String data)
  {
     (new SendMessage_Thread(path,data)).start();
  }

  private class SendMessage_Thread extends Thread
  {
     protected String Path;
     protected ByteBuffer Buff;
     protected PendingResult<MessageApi.SendMessageResult> PendingResultSend;

     SendMessage_Thread(String pth, String buffer)
     {
        Path=pth;
        Buff=ByteBuffer.allocate(buffer.length()); Buff.put(buffer.getBytes());
     }

     public void run()
     {
        Log.d("MessageSend", "P: "+Path+" "+new String(Buff.array()));
        PendingResultSend=Wearable.MessageApi.sendMessage(mApiClient, mSendDevice.getId(), Path,
        Buff.array());

        if(PendingResultSend!=null)
        {
           MessageApi.SendMessageResult result = PendingResultSend.await();
           if (!result.getStatus().isSuccess())
                 Log.e("sendMessage", "ERROR: failed to send Message: " + result.getStatus());
        }else
           Log.d("sendMessage", "No smartwatch connected");
     }
  }
```

Figura 52 - Clase empleada para el envío del mensaje

Algo muy importante, que no se debe olvida es desconectar el cliente de **Google** cuando se cierra la aplicación. Para ello, en el método 'onDestroy' se debe comprobar si está conectado y si es así, desconectarlo llamando al método especifico de la clase. Además de realizar esta desconexión, se debe también quitar el listener creado para recibir los mensajes creados.

```
@Override
protected void onDestroy()
{
    DisconnectWear();
    super.onDestroy ();
}

private void DisconnectWear()
{
    if(mApiClient.isConnected())
    {
        if (mMessageListener!=null)
            Wearable.MessageApi.removeListener(mApiClient, mMessageListener);

        mApiClient.disconnect();
    }
}
```

Figura 53 - Desconectando el cliente y eliminado los listeners

Todo lo anteriormente comentado, sería el código básico para lograr la conexión, el envío y la recepción de mensajes entre dispositivos. Para probarlo, en la interfaz del teléfono se han añadido dos entradas de texto (para indicar el 'path' y el contenido del mensaje) y un botón para realizar este envío. Además, posee un texto para mostrar el último mensaje recibido. La aplicación del reloj es más simple, ya que solo muestra el mensaje recibido por el teléfono. Cuando este texto se pulsa, se envía un mensaje al teléfono con la hora a la que ha sido pulsado. De esta forma tan simple se puede comprobar si todo funciona correctamente.

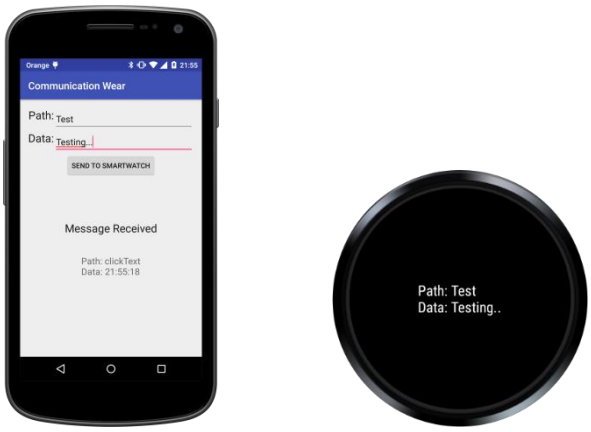


Figura 54 - Probando las aplicaciones creadas para la comunicación bidireccional

5.4 Creación de la aplicación principal para Android y Android Wear

5.4.1 Especificaciones de la aplicación

Antes de empezar con la creación de la librería/aplicación, es importante saber cuáles son los requisitos de esta. En primer lugar, se creará la aplicación que realice la comunicación entre ambos dispositivos. Después, esta aplicación, con algunos cambios, se convertirá en plugin para **Unity**.

En primer lugar, es importante conseguir empaquetar ambas aplicaciones, Smartphone y smartwatch, en un mismo fichero. Se debe conseguir esto, porque cuando se publican las aplicaciones en el **Play Store**, hay que subir únicamente un fichero. Por lo tanto, un primer requisito es subir la aplicación a la tienda y para ello, antes, se debe conseguir que se realice este empaquetado correctamente. En los proyectos previos esto se ha conseguido, así que seguramente no exista ningún problema en conseguirlo. Además, **Play Store** permite distribuir las aplicaciones para **Android Wear**, se intentará que la aplicación desarrollada obtenga esta distinción.

Otra cosa muy importante son las diversas interfaces y métodos de control que debe tener el reloj. Ya que se trata de utilizar el smartwatch como controlador, es necesario, que este disponga de diversas alternativas. Como mínimo se deberían implementar las siguientes:

- **Vista con texto**. Esta vista contendrá un texto cuyo contenido será pasado por el teléfono. Cuando el usuario presione este texto, se le notificará al teléfono el evento ocurrido.
- **Joystick virtual**. Esta vista estará compuesta por un círculo que seguirá la pulsación del usuario. Cada vez que el usuario cambie la posición de la pulsación, el teléfono recibirá la nueva posición de este.
- **Vista con botones**. Esta vista contendrá una serie de botones que el usuario podrá pulsar. Se podrá configurar para que aparezcan diferentes botones: dos verticales, dos horizontales y/o un botón central.
- **Datos de orientación**. Se podrá activar/desactivar que se obtengan los datos de orientación en cualquiera de estas vistas.

Por último, otra funcionalidad que debe cumplir es la apertura y cierre de la aplicación del reloj desde el teléfono. La aplicación del reloj es un complemento para la aplicación del teléfono; por lo tanto, cuando la aplicación del teléfono se abra/cierre la del reloj debería realizar lo mismo.

5.4.2 Creando la aplicación para el Smartphone

En primer lugar, para realizar este proyecto se usarán los proyectos previos como base. Lo único que se debe hacer para poder empezar es juntar ambos proyectos en uno para poder empezar. Al realizar esto, se consigue un proyecto que tenga hecha la comunicación bidireccional y el empaquetado de las aplicaciones.

Ya que en esta aplicación el envío de datos es más completo que en la anterior aplicación, se debe establecer un 'protocolo' de comunicación. Cuando se envía un mensaje, se debe establecer el 'path' y los datos; pues se usará el 'path' para indicar el tipo de mensaje recibido. En la siguiente tabla se pueden apreciar la estructura y significado de los mensajes que se intercambiarán ambos dispositivos.

<table>
<tr><th>Path</th><th>Explicación</th><th>Emisor</th></tr>
<tr><td>/MyWearController-Start_activity</td><td rowspan="2">Cuando se abre o cierra la aplicación en el teléfono, este mensaje se envía al reloj para notificarle este cambio. Con este mensaje no se envían datos.</td><td rowspan="5">Todos estos mensajes se envían desde el teléfono.</td></tr>
<tr><td>/MyWearController-Stop_activity</td></tr>
<tr><td>/change_layout</td><td>Con este mensaje se le indica que el reloj debe cambiar la vista. El dato de este mensaje es la nueva vista que debe establecerse.</td></tr>
<tr><td>/activate_sensors</td><td rowspan="2">Se usa para indicar si se debe activar o desactivar algún sensor en el reloj. Con el dato enviado, se indica el sensor deseado. En este caso, este dato, será siempre orientación, ya que solo se hace uso de este sensor.</td></tr>
<tr><td>/deactivate_sensors</td></tr>
<tr><td>buttonPress</td><td rowspan="2">El reloj envía este mensaje cuando un botón de la interfaz se ha apretado o soltado. Con el dato del mensaje, se indica que botón ha sido.</td><td rowspan="4">Todos estos mensajes se envían desde el reloj.</td></tr>
<tr><td>buttonRelease</td></tr>
<tr><td>connectedWear</td><td rowspan="2">Cuando el reloj abre o cierra la aplicación notifica usando este mensaje al teléfono. Con este mensaje no se envían datos.</td></tr>
<tr><td>disconnectedWear</td></tr>
</table>

orientationValues	Cuando los datos de los sensores de orientación o la posición del joystick cambian, se envía este mensaje. Con los datos enviados, son los nuevos valores separados por un delimitador (‘#’).	Se envía desde el reloj
joystickValues		

Tabla 1 – Estructura de los mensajes enviados

Uno de los requisitos, anteriormente comentados, es la apertura y cierre de la aplicación del reloj a la vez que lo haga el teléfono. Para lograr esto, cuando se abra o cierre la aplicación, se envía un mensaje al reloj.

```
@Override
public void onConnected(@Nullable Bundle bundle)
{
    initMessageListener();
    Wearable.MessageApi.addListener(mApiClient, mMessageListener);
    sendMessageChecking(PublicConstants.START_ACTIVITY, "");
}

@Override
protected void onDestroy()
{
    if(mConnectedWearable==1)
        sendMessageUnchecked(PublicConstants.STOP_ACTIVITY, "");

    DisconnectWear();
    super.onDestroy ();
}
```

Figura 55 - Envío de mensajes cuando se abra y cierre la aplicación del teléfono

Como se puede comprobar cuando se envía el mensaje, en el método onConnected, se invoca un método distinto al empleado cuando se desconecta. Básicamente un método emplea la comprobación de la recepción del mensaje (‘sendMessageChecking’), como se hizo en el primer ejemplo y el otro método (‘sendMessageUnchecked’), más rápido, no la realiza. Además, si se envía utilizando el primer método, si la aplicación del reloj no estuviera abierta, se envía un mensaje para abrirla y cuando esté abierta enviarle el mensaje que no se ha recibido.

Todas las cadenas de texto empleadas como ‘path’ en los mensajes, están almacenadas en una clase pública y estática llamada PublicConstants. Cuando se recibe un mensaje se comprueba cuál es su ‘path’ para realizar una acción u otra.

```
  @Override
   public void onMessageReceived(MessageEvent messageEvent)
   {
       String path = messageEvent.getPath();
       byte[] Bdata = messageEvent.getData();
       String data=new String(Bdata);

       switch(path)
       {
           case PublicConstants.BUTTON_PRESS:
               mButtonPressed=data;
               RefreshValues();
               break;

           case PublicConstants.JOYSTICK_VALUES:
               String[] _piecs=data.split("#");
               if(_piecs.length>1)
                   for(int i=0; i<2; i++)
                       mJoystickValues[i]=Float.parseFloat(_piecs[i]);

               RefreshValues();
               break;

           case PublicConstants.DISCONNECTION:
               mConnectedWearable = -1;
               mConnectedWearTView.setText("FALSE");
               break;
...
```

Figura 56 - Recibiendo los mensajes en el teléfono

Como se puede comprobar, cuando se recibe un mensaje, se guardan los valores recibidos en variables propias de la clase. Luego, se llama al método 'refreshValues', que se encarga de actualizar la interfaz con los datos recibidos.

En la imagen, se puede apreciar el layout del teléfono, donde se ven los datos que se muestran. Mediante el spinner se selecciona las vistas deseada del reloj. Cuando se esté ejecutando, según la vista, se muestran algunos datos. Por ejemplo, si se selecciona una vista con botones, solo sería visible la información del botón pulsado; la posición del joystick, así como el texto a enviar, se ocultarían. Con el giroscopio ocurre lo mismo, los datos de la rotación, solo se mostrarían cuando estos estén activos.

Figura 57 - Vista con datos de la aplicación del teléfono

5.4.3 Creando la aplicación para el smartwatch

5.4.3.1 Creando las diferentes vistas de la aplicación

En primer lugar, se han creado las diversas vistas disponibles del reloj. Las vistas con botones han sido las primeras enser creadas. Al principio, para crearlas, se usaron los botones nativos de Android. Pero esto presentaba algunos inconvenientes. Dentro de los botones, únicamente se puede poner texto. Esto es u pequeño problema, ya que estas vistas quedarían mejor con dibujos de flechas que con caracteres. Pero el principal problema son las pulsaciones. Por defecto, **Android** en los botones, solo tiene definidos los eventos de pulsación simple y pulsación larga; para cuando se deja de presionar, no tiene nada. Esto es un grave problema, ya que es necesario saber tanto cuando una tecla se pulsa y cuando deja de ser pulsada

Figura 58 - Diferentes layouts usando los botones nativos de Android

Para solucionar estos problemas, se ha creado una vista propia. En ella desde el método 'onDraw', de forma sencilla, se pueden dibujar diversas líneas para hacer las deseadas flechas. Al realizar la vista propia, el tema de eventos de pulsaciones también está solucionado. Ya que cuando se presiona o no la pantalla, desde el método 'ontouchEvent', se puede gestionar sin demasiadas complicaciones.

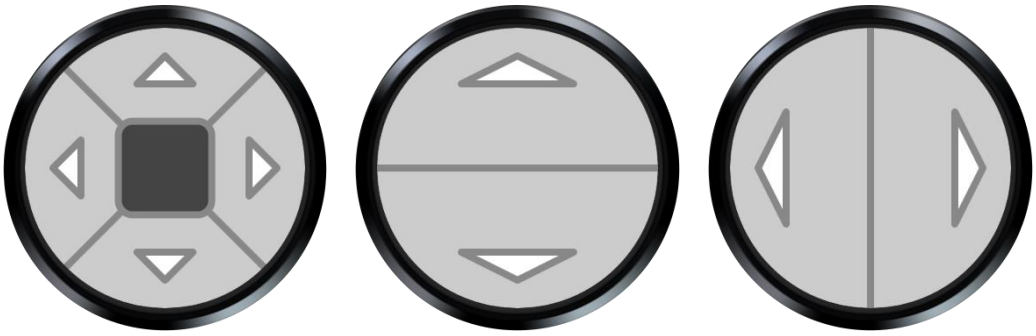


Figura 59 - Diferentes layouts usando la vista creada

Otra vista creada, es una vista simple con texto. En esta vista cuando se crea, se establece el texto deseado; además el evento de la pulsación del mismo se tiene en cuenta, para notificarlo al teléfono. Por último, de las vistas especificadas, solo falta el joystick virtual. En ella, se dibuja un par de círculos que hacen de palanca virtual. Cuando se mueve el dedo, estos siguen este movimiento y cuando se deja de pulsar la pantalla, estos vuelven a su posición inicial. Para implementar esta vista, se ha creado na vista propia, ya que con las que ofrecía Android era más complejo realizarlo.

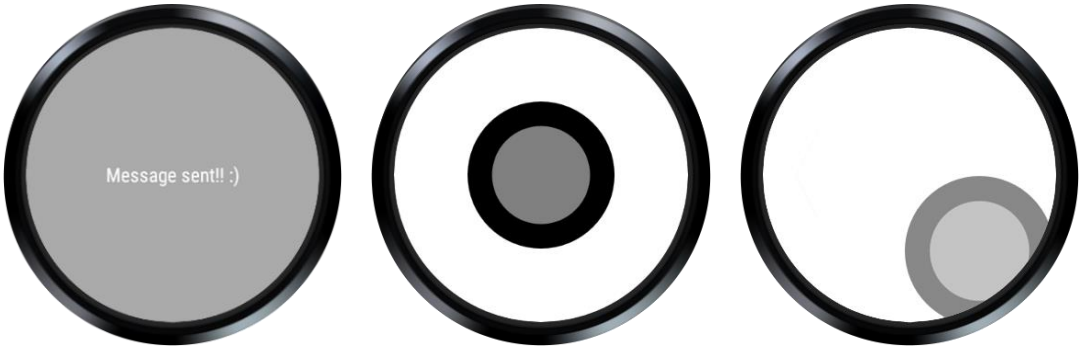


Figura 60 - Vista con texto y vista de joystick virtual

En estas vistas propias, el pincel usado para dibujar los elementos, se ha tenido que indicar que se active el 'antialiasing'. Cuando no estaba activado, se podían observar las líneas poco definidas. Además, también se ha activado el redondeado de las líneas, para evitar las puntas tan pronunciadas de las flechas.

```
private Paint CreatePaintWithAntialias(){
   Paint _paint=new Paint();
   _paint.setDither(true);
   _paint.setStrokeJoin(Paint.Join.ROUND);
   _paint.setStrokeCap(Paint.Cap.ROUND);
   _paint.setAntiAlias(true);
   return _paint;
}

@Override
protected void onDraw(Canvas canvas) {
   Paint paint = CreatePaintWithAntialias();
   Paint paint2=CreatePaintWithAntialias();
   ...
```

Figura 61 - Creando el pincel con las opciones necesarias para activar el suavizado

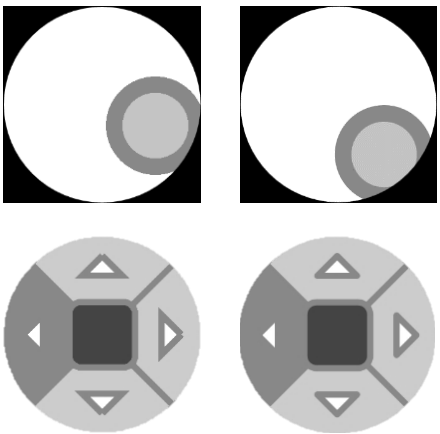


Figura 62 - Diferencias entre usar el pincel normal y el pincel creado

5.4.3.2 Obteniendo los datos de orientación

Para obtener los datos de la orientación del reloj se debe usar un 'SensorEventListener' y registrar este listener para recibir los cambios ocurridos en los sensores de movimiento y sensor magnético. Cuando se reciben los valores de ambos sensores, se calcula la orientación usando un método del 'sensorManager'. Una vez obtenidos estos datos, se envían al teléfono; pero no se envían siempre cuando se calculan, para no sobrecargar el canal de envío. Este envío de datos se realiza cada 125 milisegundos.

```
protected void onCreate(Bundle savedInstanceState){
. . .
    mOrientationListener=new SensorEventListener() {
         @Override
         public void onSensorChanged(SensorEvent event) {
             switch(event.sensor.getType()) {
               case Sensor.TYPE_ACCELEROMETER: mGravityValues =event.values; break;
               case Sensor.TYPE_MAGNETIC_FIELD: mGeoMagneticValues =event.values; break;
            }

            if(System.currentTimeMillis()-mLastOrientationSent< 125) return;

            if(mGeoMagneticValues !=null && mGravityValues !=null)
                  if(SensorManager.getRotationMatrix(R, null, mGravityValues, mGeoMagneticValues)){
                        SensorManager.getOrientation(R, orientation);
                        sendMessageChecking(PublicConstants.ORIENTATION_VALUES,
                              orientation[0]+"#"+orientation[1]+"#"+orientation[2]);
                        mLastOrientationSent=System.currentTimeMillis();
                  }
         }
     };

     List<Sensor> sensorsAcelerometer = mSensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
     List<Sensor> sensorsMagnetic = mSensorManager.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
     if (sensorsAcelerometer.size() > 0 && sensorsMagnetic.size()>0) {
          mSensorManager.registerListener(mOrientationListener, sensorsAcelerometer.get(0), delay);
          mSensorManager.registerListener(mOrientationListener, sensorsMagnetic.get(0), delay);
     }
```

Figura 63 - Obteniendo los datos de la orientación del reloj

5.4.3.3 Abriendo la aplicación cuando el teléfono lo haga

Para conseguir que ambas aplicaciones (teléfono y reloj) se abran y cierren a la vez se ha creado un servicio en segundo plano. Este servicio, se encuentra 'escuchando' y cuando reciba el mensaje apropiado del teléfono realiza la acción necesaria. Este servicio se debe declarar en el manifest de la aplicación.

```
<service android:name=".MyWearableListenerService">
 <intent-filter>
  <action android:name="com.google.android.gms.wearable.MESSAGE_RECEIVED" />
    <data android:scheme="wear" android:host="*" android:path="/MyWearController-Start_activity"/>
    <data android:scheme="wear" android:host="*" android:path="/MyWearController-Stop_activity"/>
  <!--<action android:name="com.google.android.gms.wearable.BIND_LISTENER" />-->
 </intent-filter>
</service>
```

Figura 64 - Declarando el servicio en el manifest

Para declarar el servicio en el manifest, se debe indicar que acción lleva asociada. En principio, se especificó la acción: 'com.google.android.gms.wearable.BIND_LISTENER'. Pero esta opción no era la mejor, ya que el servicio se crea y se destruye cuando se enchufa y apaga el dispositivo. De esta forma, está siempre ejecutándose en segundo plano y todos los eventos que le ocurran al dispositivo, pasan por este servicio, tanto si los tiene definidos como sino. La forma más óptima es usando la acción 'com.google.android.gms.wearable.MESSAGE_RECEIVED'; de esta forma, sólo se ejecuta cuando recibe un mensaje. Además, por medio de los atributos 'data', se le puede especificar el 'path' del mensaje, así como el destinatario y el esquema. De esta forma, solo se ejecutará el servicio cuando reciba los mensajes establecidos.

```
public class MyWearableListenerService extends WearableListenerService
{
    @Override
    public void onMessageReceived(MessageEvent messageEvent){
        String path = messageEvent.getPath();
        byte[] data = messageEvent.getData();
        String sData = new String(data);

        if(path.equals(PublicConstants.STOP_ACTIVITY)){
            AppSharedPreferences.setAppOpen(getApplicationContext(), false);
            super.onMessageReceived(messageEvent);
        }else{
            if (!AppSharedPreferences.getAppOpen(getApplicationContext()) &&
                path.equals(PublicConstants.START_ACTIVITY)))
            {
                Intent intent = new Intent(this, MainActivity.class);
                intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent);
            } else
                super.onMessageReceived(messageEvent);
        }
    }
}
```

Figura 65 - Servicio para recibir los mensajes en segundo plano

En el servicio, se comprueba que mensaje ha recibido, para realizar una acción u otra. Cuando se recibe el mensaje para abrir la aplicación, se comprueba si no está abierta ya. Para ello se usa una clase auxiliar que guarda, en el almacén de claves 'SharedPreferences', el estado de la aplicación (abierta o cerrada). Si no se realizara esta comprobación, podría dar problemas al crear varias veces la misma aplicación y realizar varias veces la inicialización de cliente de Google.

```
public class AppSharedPreferences
{
   private static final String mIdSP="AppSP", mIdVar="AppOpened";
   public static boolean getAppOpen(Context context)
   {
      SharedPreferences prefs =context.getSharedPreferences(mIdSP, Context.MODE_PRIVATE);
      return prefs.getBoolean(mIdVar, false);
   }
```

```
   public static void setAppOpen(Context context, boolean open)
   {
      SharedPreferences prefs =context.getSharedPreferences(mIdSP, Context.MODE_PRIVATE);

      SharedPreferences.Editor editor = prefs.edit();
      editor.putBoolean(mIdVar, open);
      editor.commit();
   }
}
```

Figura 66 - AppSharedPreferences de la aplicación

Como se puede comprobar, esta clase es muy simple; únicamente posee dos métodos. Uno de ellos se usa para guardar este dato y el otro para poder recuperarlo. Para realizar ambas acciones, primer se obtiene el SharedPrefences con un identificador establecido y en modo privado. Una vez conseguido el almacén, se guarda o se obtiene el booleano asociado al identificador 'AppOpened'.

5.4.3.4 Enviando los datos de los eventos al reloj

En el reloj, cuando se produce un evento se envían los datos. Cuando se presiona un botón o se deja de hacerlo, se envía un mensaje indicando el evento ocurrido. Con el joystick este envío es un poco diferente. Cada vez que se modifica la posición del mismo, se llama al método 'onJoystickPositionChange'.

```
   @Override
   public void onButtonPress(ButtonName pressed) {
       sendMessageChecking(PublicConstants.BUTTON_PRESS, pressed.name());
   }

   @Override
   public void onButtonHold(ButtonName released) {
       sendMessageChecking(PublicConstants.BUTTON_RELEASE, released.name());
   }

   @Override
   public void onJoystickPositionChange(float newX, float newY){
     if(newX!=mJoystickNX && newY!=mJoystickNY){
         mJoystickNX=newX;
         mJoystickNY=newY;
     }else
         return;

     long actualTime=System.currentTimeMillis();

     if(Math.abs(newX)+Math.abs(newY)>0.1)
         if(actualTime- mLastPositionJoystickSend_Time < 375){return;}

     sendJoystickPosition(actualTime);
   }

   private void sendJoystickPosition(long time) {
       mLastPositionJoystickSend_Time =time;
       sendMessageUnchecked(PublicConstants.JOYSTICK_VALUES, mJoystickNX+"#"+mJoystickNY);
       mJoystickSendNX=mJoystickNX; mJoystickSendNY=mJoystickNY;
   }
```

Figura 67 - Enviando datos al teléfono cuando se producen eventos

Los datos de la posición del joystick no se envían cada vez que se obtiene, ya que este comportamiento no sería óptimo. Para ello este envío se realiza cuando la nueva posición está muy cerca de la inicial (0,0) o cada 375 milisegundos.

```
class SendJoystickNewPosition_Task extends TimerTask
{
   public void run(){
       if (mJoystickSendNX != mJoystickNX && mJoystickSendNY != mJoystickNY)
       {
           long actualTime = System.currentTimeMillis();
           if (actualTime - mLastPositionJoystickSend_Time < 500) return;
           sendJoystickPosition(actualTime);
        }
    }
}
```

Figura 68 - Timer empleado para el envío de los valores del joystick

Si los datos se enviarán únicamente cuando se cumplen una de las dos condiciones anteriormente comentadas, se perderían datos. Para ello hay un 'timer', que cada 500 milisegundos, está comprobando si hay datos sin enviar. De esta forma no se pierden datos, lo único es que algunos de ellos se enviarían con un pequeño retraso.

5.4.3.5 Eliminando el 'Swipe To Dismiss' de la aplicación

Las aplicaciones en Android Wear, disponen de dos formas para salir de ellas. A través del botón lateral o realizando un deslizamiento hacia la derecha. Para esta aplicación en concreto, solamente se desea que se salga de la aplicación cuando el teléfono le notifique para que cierre la aplicación y cuando se presione el botón lateral. Para eliminar el comportamiento al realizar el desplazamiento (Swipe to dismiss), se debe asignar un tema a la aplicación/actividad que indique que no se desea este comportamiento.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="AppTheme" parent="@android:style/Theme.DeviceDefault">
    <item name="android:windowSwipeToDismiss">false</item>
  </style>
</resources>
```

Figura 69 - Tema para evitar el 'Swipe To Dismiss'

Para cumplir con los patrones de diseño de Android Wear, si se elimina este desplazamiento para salir de la aplicación, se debe añadir otro método para poder salir de la aplicación. Este método consiste en realizar una pulsación larga en pantalla; al cabo de unos segundos aparecerá un botón rojo para poder salir de la aplicación. A este método se le conoce como 'LongPress'.

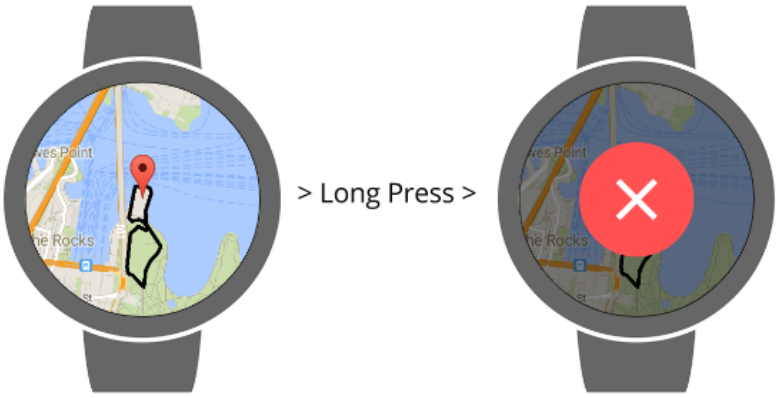


Figura 70 - Ejemplo de LongPress en aplicación de Maps

Para activar este método se debe añadir una vista al layout ('DismissOverlayView') y desde la clase mostrarla cuando sea necesario. Para esta aplicación del smartwatch que se está desarrollando, esto se activará únicamente en la primera vista, que indica que está esperando la conexión con el teléfono. En las demás interfaces se desea asignar otro comportamiento a las pulsaciones largas en pantalla.

```
  ...
  // Obtain the DismissOverlayView element
   mDismissOverlay = (DismissOverlayView) findViewById(R.id.dismiss_overlay);
   mDismissOverlay.setIntroText("Long Press to dismiss");
   mDismissOverlay.showIntroIfNecessary();

   // Configure a gesture detector
   mDetector = new GestureDetector(this, new GestureDetector.SimpleOnGestureListener() {
      public void onLongPress(MotionEvent ev) { mDismissOverlay.show(); }
     });
}

// Capture long presses
@Override
public boolean onTouchEvent(MotionEvent ev) {
    if(mActualView=="None") return mDetector.onTouchEvent(ev) || super.onTouchEvent(ev);
    else                    return super.onTouchEvent(ev);
 }
```

Figura 71 - Activando el DismissOverlayView en la actividad principal

5.4.4 *Subida de la aplicación al Play Store*

Una vez probada y finalizada la aplicación, es momento de subirla al **Play Store**. Cuando se sube una aplicación se puede elegir en que versión subirla, alfa, beta o en producción. Esta aplicación se va a subir en versión beta y abierta. Para poder acceder a ella, es necesario usar un código que habilita al usuario como tester de esta aplicación.

Para subir una aplicación lo primero es disponer de una cuenta de desarrollador. Con esta cuenta, desde la consola de desarrolladores del **Play Store**, se debe seleccionar el nombre de la aplicación y subir el fichero APK. Más adelante, se pueden subir diferentes versiones del APK para ir corrigiendo pequeños errores o ir añadiendo nuevas funcionalidades. Accediendo al apartado de APK, se puede llevar un control de las versiones subidas, así como añadir o eliminar una versión.

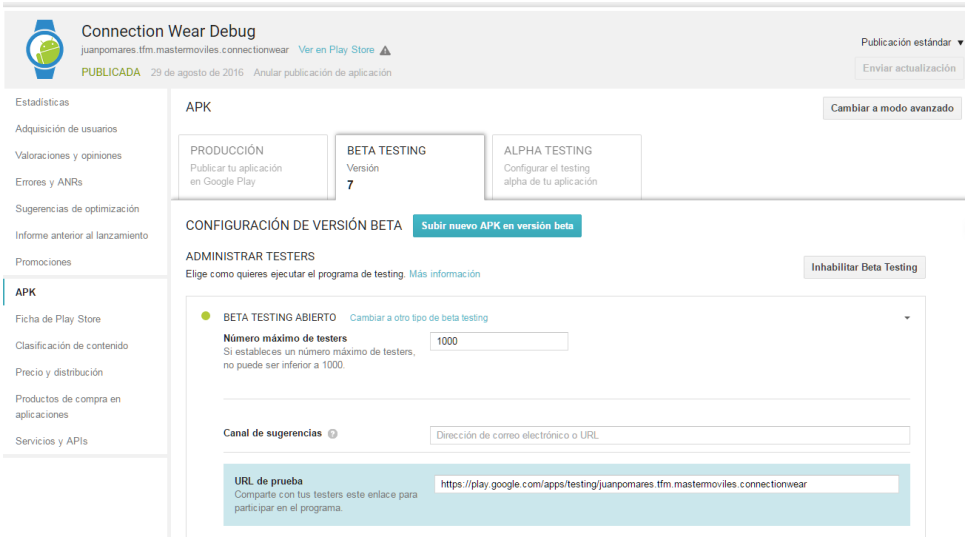
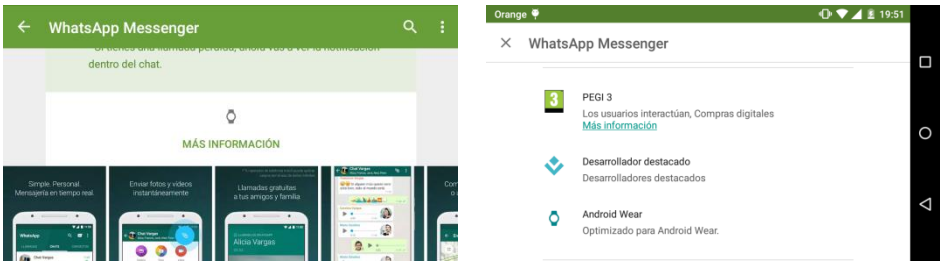


Figura 72 - Subiendo el APK de la aplicación 'Connection Wear'

Después de subir el fichero de la aplicación (APK), se puede crear la ficha de la misma. En la ficha se debe especificar el nombre de la aplicación, una descripción corta y otra descripción completa. Esta ficha se puede traducir a varios idiomas. A parte de las descripciones, es necesario seleccionar el icono de la aplicación en alta resolución, una imagen destacada y algunas capturas de la aplicación. Cuando se suben las capturas se pueden subir separadas según la plataforma a la que se vayan a distribuir, en este caso, se han subido capturas para la aplicación de **Android** y la de **Android Wear**.

Figura 73 - Creando la ficha de la aplicación 'Connection Wear'

Cuando se sube una aplicación que se instala en un dispositivo **Android Wear**, se puede especificar que se distribuya para **Android Wear**. Para ello se debe seleccionar en distribución esta opción. Si después de realizar la revisión, aprueban esta distribución, en el **Play Store** aparecerá el logo de **Android Wear** en la ficha de la aplicación.



airconFigura 74 - Distribución para Android Wear en la aplicación 'Whatsapp Messenger'

Como la aplicación desarrollada se instala también en un smartwatch, se ha seleccionado la opción de distribución para Android Wear. Desgraciadamente, no la han aprobado porque no cumplía con algunos patrones de diseño.

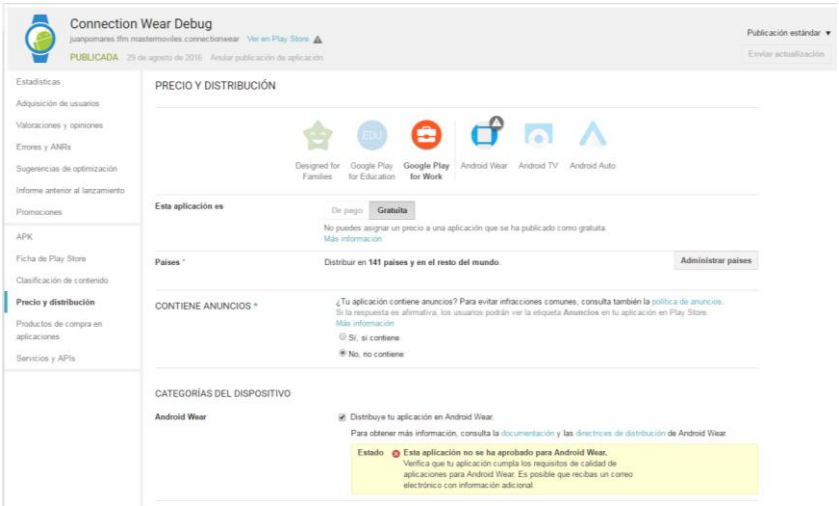


Figura 75 - Distribución para Android Wear denegada para 'Connection Wear'

5.4.5 *Resultado final de la aplicación de Android Wear*

Con la aplicación finalizada y publicada, se puede valorar los resultados de esta aplicación realizada. En primer lugar, considero que las especificaciones establecidas para esta aplicación se han cumplido perfectamente. Es cierto que la apertura de forma simultánea en ambos dispositivos, no se ha logrado correctamente, ya que en ciertos dispositivos no funciona como debería.

En lo referente a la publicación de la aplicación en **Play Store**, se ha realizado correctamente. El único problema, como se ha comentado anteriormente, ha sido la denegación de la distribución para **Android Wear**. Esta distinción habría sido muy positiva para la aplicación, ya que podría aparecer como aplicación destacada para Android Wear.

De todas formas, la aplicación únicamente presenta estos dos inconvenientes, pero estos no son demasiado relevantes. A parte de esto, se ha podido comprobar que el retraso existente en la comunicación de ambos dispositivos es casi imperceptible. Por todo lo comentado, considero que los resultados de la aplicación son bastante buenos.

En la siguiente tabla, se muestran enlaces de interés sobre la aplicación. El primero de ellos es el enlace al **Play Store**, donde se puede descargar y probarla. El segundo enlace, es un vídeo de **Youtube**, donde se ve la aplicación en funcionamiento. El último enlace es el proyecto subido a **Github**, donde se puede ver el código y los recursos de todo el proyecto.

Google play	https://play.google.com/store/apps/details?id=juanpomares.tfm.mastermoviles.connectionwear https://play.google.com/apps/testing/juanpomares.tfm.mastermoviles.connectionwear
	https://www.youtube.com/watch?v=2hGCf_Uh_c4
	https://github.com/juanpomares/Mobile-projects/tree/master/Connection Wear Project(Android)

Tabla 2 - Enlaces de la aplicación 'Connection Wear'

5.5 Creación del proyecto final para Unity

5.5.1 Cambios realizados en la aplicación para convertirlo en plugin

Lo primero, antes de empezar con el proyecto de Unity, se debe convertir la aplicación anterior en plugin para este motor. Para ello, primero hay que seguir los pasos especificados en el apartado: "Creando el primer plugin de Unity". A parte de esos se han realizado algunos cambios más.

El primero de ellos, es modificar de la clase 'PublicConstants', los comandos de 'Start_activity' y 'Stop_activity'. De esta forma, esta aplicación y la anterior (Connection Wear), poseen comandos diferentes entre ellas para abrir y cerrarlas.

```
public static final String START_ACTIVITY = "/MiUnityWearController-Start_activity";
public static final String STOP_ACTIVITY = "/MiUnityWearController-Stop_activity";
```

Figura 76 - Nuevos comandos para abrir y cerrar la aplicación

Cuando se produce un evento en la aplicación del reloj, como la pulsación de un botón o cambio en los valores de orientación, se notifica a los objetos de **Unity**. Para ello existen en la clase unos mapas donde se almacenan el id del objeto y el método al que se le notificará este evento. Se podría decir que los objetos de **Unity** se suscriben y des suscriben a los diferentes eventos que ocurren en el reloj.

```
protected LinkedHashMap<String, String> mJoystickListeners=null;
protected LinkedHashMap<String, String> mButtonPressListeners=null;
protected LinkedHashMap<String, String> mButtonReleaseListeners=null;
protected LinkedHashMap<String, String> mOrientationListeners=null;
protected LinkedHashMap<String, String> mConnectionListeners=null;
protected LinkedHashMap<String, String> mDisconnectionListeners=null;
. . .
public void addConnectionListener(String object, String function)
{ addListener(mConnectionListeners, object, function);}
public void addGDisconnectionListener(String object, String function)
{  addListener(mDisconnectionListeners, object, function);}
. . .
public void removeJoystickListener(String object)
{  removeListener(mJoystickListeners, object);}
public void removeButtonPressListener(String object)
{  removeListener(mButtonPressListeners, object);}
. . .
private void addListener(LinkedHashMap<String, String> map, String obj, String func)
{  map.put(obj, func);  }

private void removeListener(LinkedHashMap<String, String> map, String object)
{  map.remove(object);  }
```

Figura 77 - Definiendo los listeners usados para notificar los eventos

Los eventos a los que se pueden suscribir son los siguientes: botón pulsado y soltado, conexión y desconexión del reloj, nuevos datos de orientación y cambio en la posición del joystick, botón pulsado y soltado. Cuando se produce un evento de los mencionados, se recorre el mapa en concreto para notificar a los objetos interesados.

Para ello, se llama al método que realiza el envío de mensaje del plugin a Unity, usando el id de ese objeto, el método de notificación y los datos del evento.

```
private void notifyListener(LinkedHashMap<String, String> _map, String data)
{
      Iterator<String> _objects=_map.keySet().iterator();
      while(_objects.hasNext())
      {
          String _object=_objects.next();
          String _method=_map.get(_object);

          NotifyObject(_object, _method, data);
      }
  }

  private void NotifyObject(String idObject, String method, String message){
      UnityPlayer.UnitySendMessage(idObject, method, message);
  }
```

Figura 78 - Notificando los objetos de Unity suscritos a los eventos

La aplicación de smartwatch no necesita que se realicen cambios en ella; únicamente el cambio de los comandos para abrir y cerrarla. Con todo esto ya podemos exportar la librería y comenzar con la creación del proyecto de **Unity**.

5.5.2 Creación de los menús

En primer lugar, se ha empezado con la creación de los menús. Para ello, en la escena, se han añadido diversos botones para poder desplazarte y se le ha asignado un script al canvas de la misma. Mediante este script, se gestionan los eventos del reloj para poder desplazarse por estos menús. En el proyecto existen dos escenas con menús, el menú principal y el menú de selección de nivel.

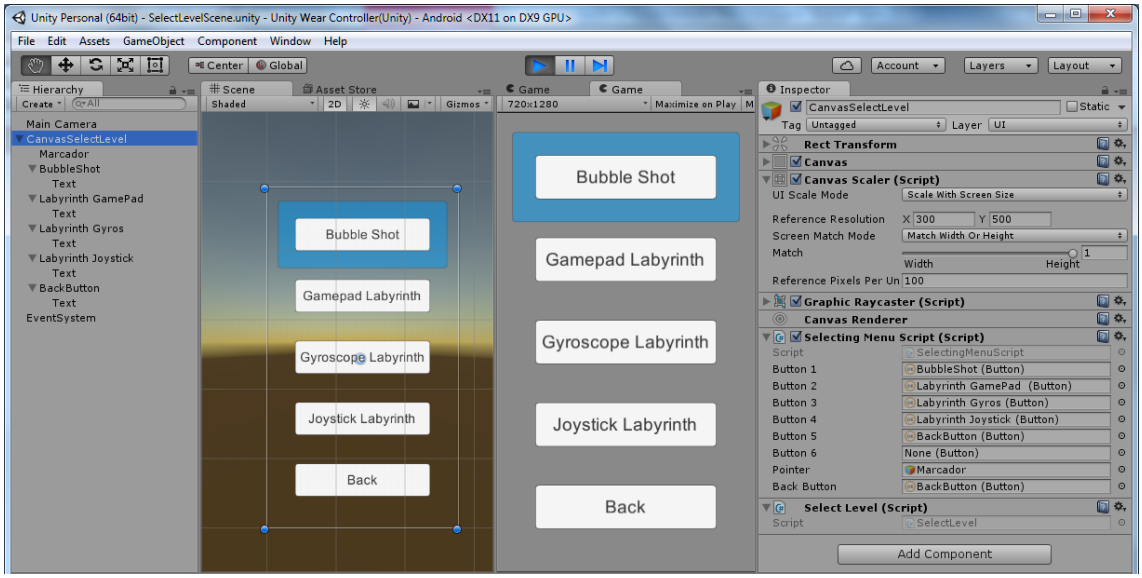


Figura 79 - Escena con el menú principal del proyecto en Unity

En este script, en el método 'Start', primero, se obtiene la referencia a la actividad actual, para poder acceder a sus métodos. Una vez obtenido se cambia la interfaz del reloj para poner la interfaz que posee dos botones y un botón central. De esta forma, el usuario puede desplazarse de forma cómoda por los menús. En este método de inicialización, aparte de cambiar la interfaz, este objeto se suscribe a los eventos de pulsación de botones. Para ello, se llama al método 'addButtonPressListener', como parámetros, se le indica el nombre del objeto y el método deseado de este script. Por último, cuando se destruye este objeto (método 'onDestroy'), se llama al método para 'des suscribirse' de estos eventos.

```
public class SelectingMenuScript : MonoBehaviour
{
  ...

 void Start ()
 {
  inicializeButtons();
  if (mButtonsCount>0)
  {
  #if UNITY_ANDROID && !UNITY_EDITOR
  AndClass=new AndroidJavaClass("juanpomares.tfm.matermoviles.unitywearcontroller.MainActivity");
  if(AndClass!=null)
    currentActivity= AndClass.GetStatic<AndroidJavaObject> ("mContextActivity");

  if(currentActivity!=null)
  {
    currentActivity.Call("addButtonPressListener", transform.name, "onButtonPressed");
    currentActivity.Call("setWearInterface", "HorizontalPadWithButton");
  }
  #endif
  setButtonSelected (0);
 }
}

void OnDestroy()
{
  if (mButtonsCount>0) {
  #if UNITY_ANDROID && !UNITY_EDITOR
  if(currentActivity!=null){currentActivity.Call("removeButtonPressListener", transform.name);}
  #endif
}}

public void onButtonPressed(string button)
{
  switch (button)
  {
    case "Up":      setButtonSelected (-1);   break;
    case "Down":    setButtonSelected (+1);   break;
    case "Center":  if(mButtonSelected!=-1){ mButtons[mButtonSelected].onClick.Invoke();}  break;
  }
}
```

Figura 80 - Script usado para los menús de Unity

En el método 'onButtonPressed', se gestiona que acción realizar cuando se presiona un botón en el reloj. Si el botón presionado es el botón de arriba o abajo, se modifica el botón seleccionado de la interfaz; mientras que, si es el botón central, se llama al método 'onClick' de ese botón.

```
public class SelectLevel : MonoBehaviour
{
  public void ClickBack(){ SceneManager.LoadScene ("MainScene");}
  public void ClickBubbleShotButton(){SceneManager.LoadScene ("Bubbleshot");}
  public void ClickGyrosLabyrinthButton(){ SceneManager.LoadScene ("GyroscopeLabyrinth");}
  public void ClickPadLabyrinthButton(){ SceneManager.LoadScene ("PadLabyrinth");}
  public void ClickJoysLabyrinthButton(){SceneManager.LoadScene ("JoystickLabyrinth");}
}
```

Figura 81 - Script usado para el cambio de escenas en Unity

En el script 'SelectLevel', están definidos los métodos que serán invocados cuando se presionen los botones de la interfaz. En ellos, se llama al cargar escena del gestor de escenas, de esta forma se cambia de escena en **Unity** y se carga un minijuego.

5.5.3 Creación de los diversos minijuegos

Se han creado un par de minijuegos para poder probar el plugin creado. En ambos juegos para salir de ellos, es necesario apretar la pantalla del teléfono.

El primero de estos minijuegos es uno parecido al 'Bubble Shot'. En este juego, el usuario dispara pelotas de diferentes colores y cuando se juntan cuatro del mismo color, todas ellas desaparecen. El objetivo final de este juego consiste en eliminar todas ellas.

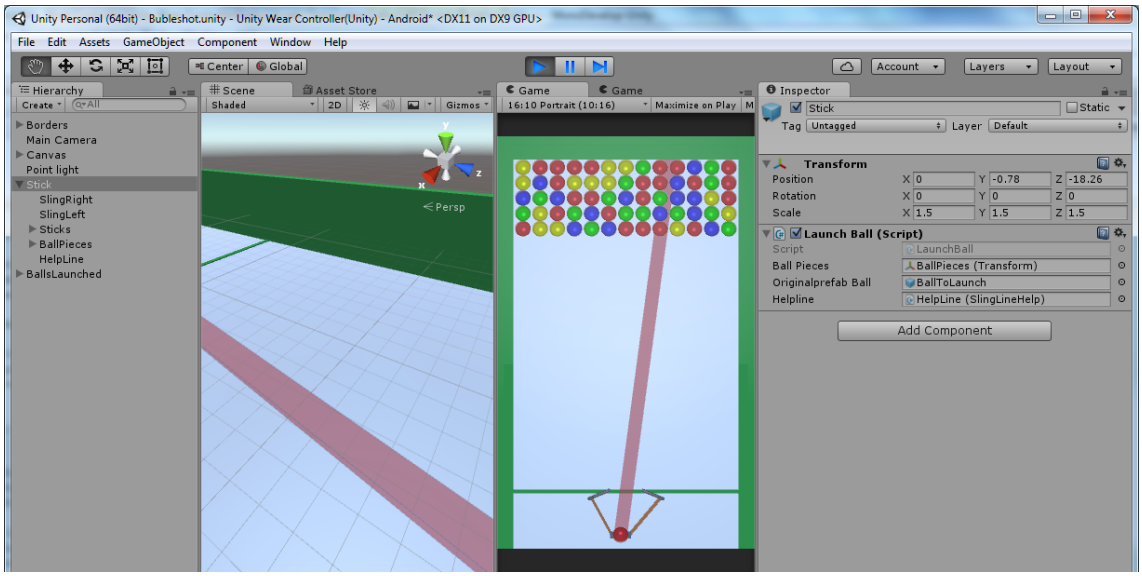


Figura 82 - Bubbleshot creado en Unity

En este juego, el script 'LaunchBallScript' es el encargado de lanzar y dirigir las bolas. Cuando se inicia este script, se realizan las inicializaciones necesarias; entre ellas, se añaden los listeners para cuando un presione o suelte un botón y se cambia la interfaz, para establecer la que posee dos botones verticales con un botón central. Cuando el objeto se destruya, se de-suscribirá de estos listeners usando el método de 'removeListener'.

Cuando el botón central es presionado, se lanza una pelota llamando al método 'ThrowBall'. Cada vez que se llama al método 'Update', se comprueba que botón es pulsado, por si se debería girar la pelota actual.

```
public class LaunchBallScript : MonoBehaviour
{
   void Start () {
      ...
         if(currentActivity!=null)
         {
             currentActivity.Call("addButtonPressListener", transform.name, "onButtonPressed");
             currentActivity.Call("addButtonReleaseListener", transform.name, "onButtonReleased")
             currentActivity.Call("setWearInterface", "VerticalPadWithButton");
         }
      }#endif
   }

   void OnDestroy()
   {
      #if UNITY_ANDROID && !UNITY_EDITOR
      if(currentActivity!=null) { currentActivity.Call("removeListeners", transform.name); }
      #endif
   }

   public void onButtonPressed(string newButton)
   {
      if (newButton == "Center") ThrowBall ();
      buttonpressed = newButton;
   }

   public void onButtonReleased(string releasedButton)
   {
      if (releasedButton == buttonpressed) buttonpressed = "";
   }

   void Update ()
   {
      . . .
            if(buttonpressed=="Left")
               changeDesp (despX+mov, despZ);
            else if(buttonpressed=="Right")
               changeDesp (despX-mov, despZ);
    . . .
```

Figura 83 - Script usado para controlar el lanzamiento de bolas

El otro minijuego creado consiste en un laberinto. Normalmente, en los juegos de laberinto, el objetivo es llegar a la salida del mismo evitando los distintos obstáculos. En este caso, el objetivo es recoger las diferentes monedas que están repartidas por todo el escenario. Para jugar a este minijuego, se han utilizado tres tipos de controles. En una prueba se usan los sensores del giroscopio para mover el escenario. En las otras pruebas, el escenario está estático y lo que se mueve es la pelota. Para mover esta pelota se usará el joystick virtual o los cuatro botones de dirección.

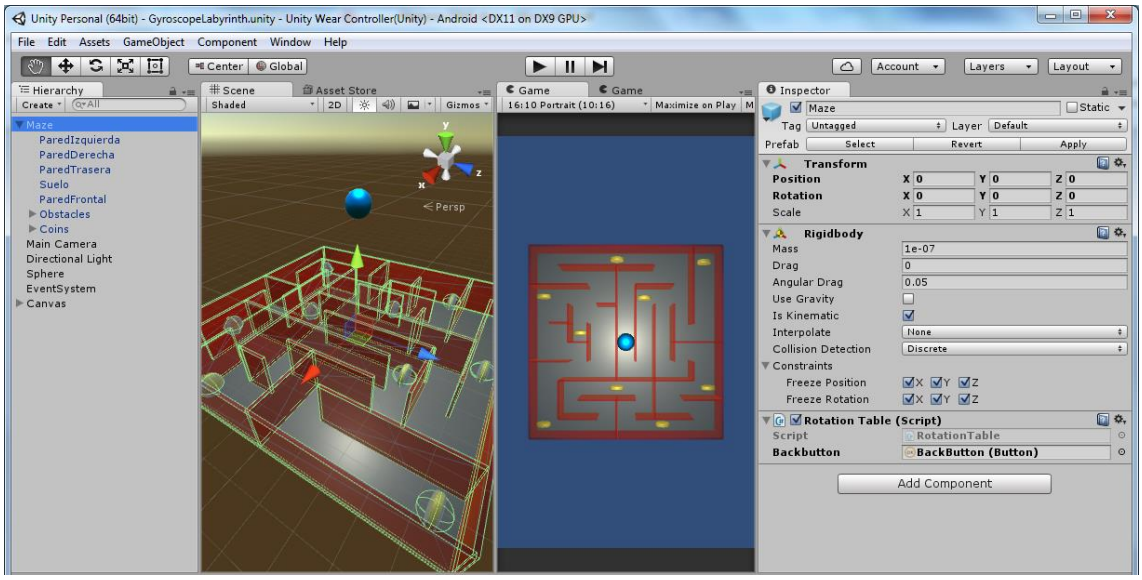


Figura 84 - Escena del laberinto en Unity

Según el script asignado, se usará un método de control u otro. Para mover el escenario con los datos del giroscopio, es necesario usar el script 'RotationTable' . Este script se le añade al tablero y se encarga de modificar la rotación del mismo.

En primer lugar, en este plugin, se activan los sensores de orientación, se añade el listener y se cambia la interfaz del reloj. En este caso, la interfaz del reloj es la vista de texto indicando las instrucciones del laberinto. Cuando el objeto se destruye, se desactivan los sensores y se elimina el listener.

En el método 'getNewRotation', se obtiene la rotación actual del sensor y después en el método 'Update', se va actualizando la rotación del objeto. Por último, para volver a la pantalla de selección de minijuego, se puede presionar tanto la pantalla del teléfono como la del reloj.

```
public class RotationTable : MonoBehaviour
{
   ...
   void Start () {
      . . .
      if(currentActivity!=null) {
         currentActivity.Call("ActiveOrientationSensors");
         currentActivity.Call("addButtonPressListener", transform.name, "onButtonPressed");
        currentActivity.Call("setWearInterface", "Rotate the watch.\n Tap the screen to exit.");
      }
      #endif
   }

   void OnDestroy()
   {
      #if UNITY_ANDROID && !UNITY_EDITOR
         if(currentActivity!=null){
            currentActivity.Call("DeActiveOrientationSensors");
            currentActivity.Call("removeButtonPressListener", transform.name);
         }
      #endif
   }
```

```
   public void getNewRotation()
   {
      #if UNITY_ANDROID && !UNITY_EDITOR
         float[] parts = currentActivity.Call<float[]> ("getOrientationValues");

         mCurrentX = Math.Clamp(parts [1]*RADIANS_DEGREE, -mMaxDegree, mMaxDegree);
         mCurrentZ = Math.Clamp(-parts [2]*RADIANS_DEGREE, -mMaxDegree, mMaxDegree);

         mNewRotation = Quaternion.Euler (mCurrentX, mCurrentY , mCurrentZ);
      #endif
   }

   void Update ()
   {
      getNewRotation ();
      transform.rotation = Quaternion.Lerp (transform.rotation, mNewRotation, Time.deltaTime*10);
   }

   public void onButtonPressed(string buttonpressed)
   {
      if (backbutton)
         backbutton.onClick.Invoke ();
   }
}
```

Figura 85 - Script usado en el laberinto para la rotación del tablero

En las otras pruebas, el script no se añade al escenario, se añade a la pelota. Para ambos métodos de control, se ha usado el mismo script (BallMovementScript); a través de una variable booleana, se especifica el método de control deseado. En el método 'Start', según esta variable se establece una interfaz en el reloj u otra. Cuando se destruye el objeto, no es necesario eliminar remover ningún listener, ya que no se había añadido ninguno al comienzo del script. Por último, en el método de actualización (Update), se le aplica una fuerza al objeto según los valores del joystick o el botón pulsado. Ya que en cada iteración, se le añade una fuerza a la pelota, en el método 'FixedUpdate', se limita la velocidad de la misma para que no sobrepase el límite.

```
public class BallMovementScript : MonoBehaviour
{
   public bool isJoystick;
   ...

   void Start ()
   {
     ...
      if(currentActivity!=null)
      {
         if(isJoystick) currentActivity.Call("setWearInterface", "Joystick");
         else currentActivity.Call("setWearInterface", "Pad");
      }
      #endif
   }

   void FixedUpdate()
   {
      if (mRigidBody.velocity.magnitude > maxSpeed)
         mRigidBody.velocity = mRigidBody.velocity.normalized *maxSpeed;
   }
```

```
void Update ()
{
   #if UNITY_ANDROID && !UNITY_EDITOR
      if(currentActivity!=null) {
         if(isJoystick){
            float[] _jvalues=currentActivity.Get<float[]>("mJoystickValues");
            mForceZ= _jvalues[1]*1.09f;
            mForceX= _jvalues[0]*1.09f;
         }else{
            mForceZ=0; mForceX=0;
            string buttonPressed=currentActivity.Get<string>("mButtonPressed");

            if (buttonPressed=="Up")          mForceZ= 1;
            else if (buttonPressed=="Down")  mForceZ= -1;
            else if (buttonPressed=="Left")  mForceX= -1;
            else if (buttonPressed=="Right") mForceX = 1;
         }
      }
   #endif
   mRigidBody.AddForce(mMaxSpeed*new Vector3(mForceX, 0, mForceZ));
}
```

Figura 86 - Script usado en el juego del laberinto para el control de la bola

Como se puede comprobar la integración del plugin como método de control en los minijuegos ha resultado bastante sencillo. Además, se puede crear una escena en Unity y según el script empleado, de forma sencilla, usar un método de control u otro, sin hacer muchos cambios en el código.

5.5.4 Empaquetado y firmado de la aplicación

Con el proyecto finalizado, podemos empaquetar y firmar la aplicación para poder subirla al **Play Store**. Para firmar la aplicación, igual que ocurre en **Android Studio**, se debe aportar el alias y la contraseña para firmar. Firmando la aplicación desde **Unity**, se puede seleccionar el mismo almacén (keystore) usado en **Android Studio** o crear uno nuevo. Para introducir estos datos, se debe seguir esta secuencia: 'Edit'→'Build Settings…' → 'Player Settings…' → 'Publishing Settings'. Desde esta pestaña ya se puede escribir el alias y contraseña para realizar el firmado.

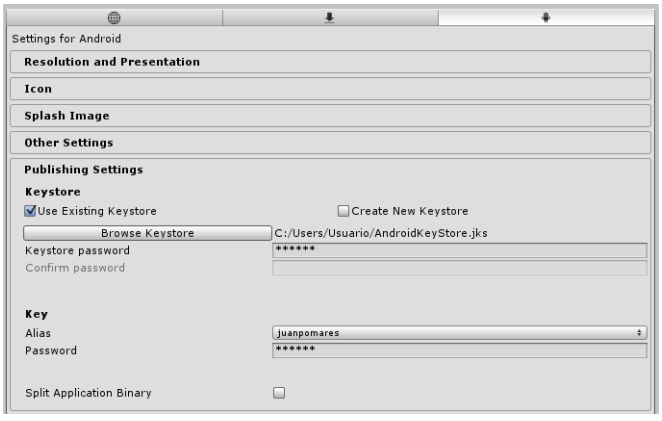
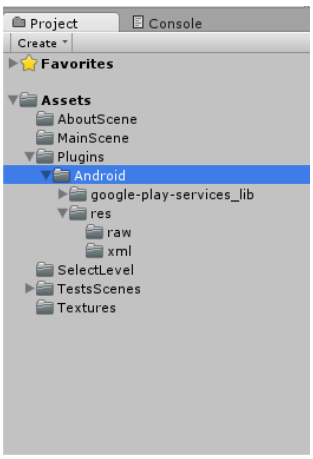


Figura 87 - Firmando la aplicación en Unity

Como el plugin creado hace uso del 'GoogleApiClient', en la carpeta específica de 'Android' en 'Plugins', se debe añadir el proyecto "Play_services_lib" de **Google**. Si esta librería no se añade, al crear la aplicación dará un error porque no encuentra las clases a las que se hacen referencia en el plugin nativo creado.




```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="juanpomares.tfm.matermoviles.unitywearcontroller">

    <application
          android:label="@string/app_name"
           android:icon="@drawable/app_icon">

        <meta-data android:name="com.google.android.wearable.beta.app"
            android:resource="@xml/wearable_app_desc"/>

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Figura 88 - Estructura de los Assets del proyecto de Unity

Figura 89 - Manifest del proyecto de Unity

Dentro de la carpeta 'Android', se debe copiar la librería generada(jar), así como el 'AndroidManifest.xml'. Además de estos ficheros, se deben crear las carpetas 'raw' y 'xml'. Dentro de estos directorios creados, se debe copiar la aplicación firmada del reloj y la descripción del recurso, igual que está explicado en: "Creando y empaquetando la primera aplicación para Android Wear". Por último, en el manifest, se debe incluir también el metadata indicando que posee una aplicación para **Android Wear**. Con todo esto realizado, ya está la aplicación preparada para poder subirla al **Play Store**.

5.5.5 Subida al Play Store del videojuego final

Para subir esta aplicación al **Play Store**, se deben seguir los mismos pasos que con la anterior aplicación. Primero se debe crear y rellenar la ficha, aportando la información necesaria y las capturas, así como subir la aplicación en formato APK. Con todo esto después de pasar la revisión que se realizan a las aplicaciones subidas, la aplicación estaría subida.

Esta aplicación, a la hora de pasar la revisión, tuvo un problema y no se pudo publicar la primar vez. El principal problema era que tanto en el logo como en el nombre escogido aparecía **Unity**. Debido a que no es una aplicación oficial de **Unity** y podía generar confusión, no se publicó. En la consola, aparecía un mensaje indicando que si se enviaba una actualización solucionando este problema, se publicaría la aplicación.

Figura 90 – 'Unity Wear Controller' rechazada en el Play Store

Debido a este problema se ha realizado un cambio en el nombre de la aplicación por uno más genérico. Anteriormente, se llamaba 'Unity Wear Controller', ahora se llama 'Game Wear Controller'. El logo también se ha visto modificado, se ha reemplazado e logotipo el de Unity por un mando de videojuegos. Con estos cambios realizados, la aplicación se ha conseguido publicar.

Figura 91 – Cambios realizados en el logo y nombre de la aplicación

Al igual que la anterior aplicación, se ha seleccionado la distribución para **Android Wear**. En esta ocasión, igual que con la otra aplicación, esta distribución también ha sido denegada. Ya que ambas aplicaciones, comparten la misma aplicación para reloj, era esperable que también fuera rechazada.

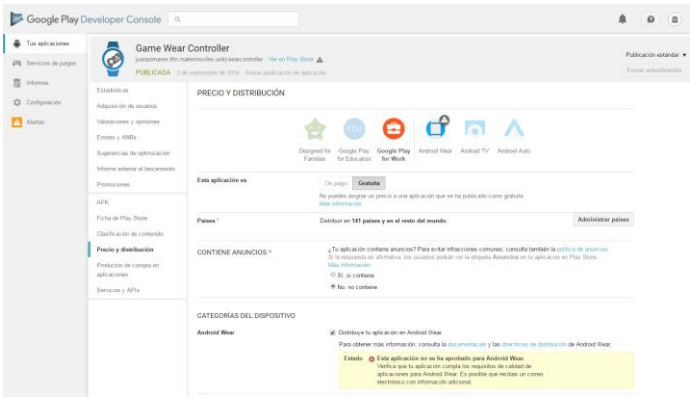


Figura 92 - Distribución para Android Wear denegada para 'Game Wear Controller'

5.5.6 *Resultado final del proyecto de Unity*

Con el proyecto finalizado y la aplicación publicada, se puede valorar los resultados obtenidos. En primer lugar, el proyecto consistía en crear un plugin para **Unity** y realizar un proyecto con diversos minijuegos para poder probarlo.

En primer lugar, con la creación del plugin, se ha podido ver como se crean e integran plugins nativos en Unity. Igual que ocurre con la otra aplicación, la apertura de forma simultánea de la aplicación en ambos dispositivos, no se ha logrado correctamente, ya que en ciertos dispositivos no funciona como debería.

En lo referente al proyecto de **Unity**, se han creado diversos minijuegos donde se puede ver el plugin creado en funcionamiento. Además, se puede apreciar lo sencillo que es integrarlo en los proyectos y que el retraso que puede existir en la comunicación apenas es perceptible, por lo que este plugin se puede usar para crear videojuegos sin ningún problema.

La publicación de la aplicación en **Play Store**, se ha realizado correctamente. Al principio la aplicación fue rechazada, pero tras realizar una serie de cambios, ya está publicada. Igual que ocurre con la aplicación anterior, tampoco han aceptado la distribución para **Android Wear**. Pero a pesar de todo ello, considero que los resultados obtenidos son bastante buenos.

En la siguiente tabla, se muestran enlaces de interés sobre el proyecto creado. El primero de ellos es el enlace al **Play Store**, donde se puede descargar y probar la aplicación. El segundo enlace, es un vídeo de **Youtube**, donde se ve el videojuego en funcionamiento. El último enlace es el proyecto subido a **Github**, donde se puede ver el código y los recursos de todo el proyecto; en ese proyecto hay dos carpetas, una para donde se ve el proyecto de **Android Studio** y el otro el proyecto de **Unity.**

Google play	https://play.google.com/store/apps/details?id=juanpomares.tfm.matermoviles.unitywearcontroller https://play.google.com/apps/testing/juanpomares.tfm.matermoviles.unitywearcontroller
	https://www.youtube.com/watch?v=hwc8hd744KE
	https://github.com/juanpomares/Mobile-projects/tree/master/Unity Wear Controller

Tabla 3 - Enlaces de la aplicación 'Game Wear Controller'

6 Conclusiones

Antes de empezar con la valoración de los resultados obtenidos, es importante, recordar cuáles eran los principales objetivos de este. En primer lugar, se debía crear una aplicación para teléfono y reloj, que después se modificaría para convertirlo en plugin. Después de la creación de esta aplicación, se debía crear el plugin para Unity así como unos minijuegos donde se pudiera ver en funcionamiento. Otro objetivo es la publicación de ambas aplicaciones en la tienda de aplicaciones.

En primer lugar, considero que los resultados de la aplicación creada, han sido correctos. Con la creación de esta aplicación se puede comprobar de una forma gráfica el funcionamiento interno del plugin, así como la latencia casi inexistente que presenta la comunicación. Además, con la subida a la tienda de la misma, se ha podido comprobar como es el proceso. Con esta aplicación solo ha habido dos problemas que no se han podido solucionar, la apertura de la aplicación del reloj a la vez que el teléfono, en algunos terminales y la desaprobación de la distribución para **Android Wear** en la tienda. Aún con estos problemas, el resultado ha sido muy bueno.

Ahora vamos a centrarnos en los resultados obtenidos con el plugin y los minijuegos creados en **Unity**. El plugin creado cumple con las diferentes especificaciones planteadas, ya que ha quedado un plugin que funciona correctamente y además es reutilizable para diversos proyectos. En cuanto a los minijuegos desarrollados, demuestran la versatilidad y facilidad de uso del plugin creado. Es cierto, que se podrían haber creado minijuegos más completos, sin embargo, el objetivo de ellos era probar el plugin, lo que se realiza de forma satisfactoria.

Un aspecto muy importante es que el código de este proyecto se encuentra disponible en un repositorio público. Como trabajo futuro, una de las posibles tareas a realizar es la creación de una documentación para el uso del plugin de **Unity**. Como se ha comentado, el código del plugin se encuentra subido a **Github**, una buena idea sería explicar la instalación y uso del mismo. Otra tarea futura podría ser la publicación del plugin en la tienda de plugins de **Unity**, así como la adición de nuevos controles en el mismo. Por último, otro proyecto futuro sería la creación de un videojuego, que haga uso de este plugin, más completo que los creados como prueba.

En resumen, el desarrollo de este trabajo de fin de máster ha sido muy satisfactorio, el proyecto obtenido cumple con los objetivos planteados inicialmente. Además, gracias a la realización de este proyecto se han obtenido algunos conocimientos nuevos que, seguramente, serán de utilidad en un futuro.

Bibliografía y referencias

- **[Blog] Smartwatches Through the Years | IT Business Blog** (http://www.itbusiness.ca/blog/smartwatches-through-the-years-the-new-idea-that-isnt-so-new/)
- **[Web] Before Apple Watch: the timely history of the smartwatch| TechRadar** (http://www.techradar.com/news/wearables/before-iwatch-the-timely-history-of-the-smartwatch-1176685)
- **[Web] TAG Heuer Connected - The Smartwatch** (http://www.tagheuerconnected.com/)

- **[Plugin de Unity] Android Presto Plugin** (https://www.assetstore.unity3d.com/en/#!/content/45191)
- **[Aplicación Android] Android Presto Plugin** (https://play.google.com/store/apps/details?id=watch.presto.prestounitycontroller)
- **[Blog] Using Android Wear to control Google Cardboard Unity VR | Damian Mehers' Blog** **(**http://damianblog.com/2015/08/23/android-wear-arm-in-cardboard-vr)

- **[Web] Packaging Wearable Apps – Android Developers** (https://developer.android.com/training/wearables/apps/packaging.html)
- **[Web] A guide to the Android Wear Message API - Tutorial - Binpress** (https://www.binpress.com/tutorial/a-guide-to-the-android-wear-message-api/152)

- **[Web] Unity – Scripting documentation** (https://docs.unity3d.com/ScriptReference/)
- **[Blog] Create an Android Plugin for Unity using Android Studio | The Game Contriver** (http://www.thegamecontriver.com/2015/04/android-plugin-unity-android-studio.html)