Proyecto Fin de Grado
Ingeniería de las Tecnologías Industriales

Aplicación móvil para el control remoto de una máquina lanzapelotas de tenis

Autor: Miguel Perozo Torrents

Tutores: Manuel Gil Ortega Linares y Manuel Garrido Satué

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Proyecto Fin de Grado
Ingeniería de las Tecnologías Industriales

Aplicación móvil para el control remoto de una máquina lanzapelotas de tenis

Autor:
Miguel Perozo Torrents

Tutor:
Manuel Gil Ortega Linares
Profesor titular:
Manuel Garrido Satué

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Proyecto Fin de Grado: Aplicación móvil para el control remoto de una máquina lanzapelotas de tenis

Autor: Miguel Perozo Torrents

Tutores: Manuel Gil Ortega Linares y
Manuel Garrido Satué

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Quiero agradecer con este trabajo a todo el profesorado y cuerpo docente que ha fomentado mi conocimiento y curiosidad durante mi estancia en la ETSI. Especialmente a todos aquellos que disfrutan de su trabajo y son conscientes de la importante labor que desempeñan de cara al futuro, ayundando así a no solo mejorar los conocimientos del alumnado, sino a inicidir en conceptos tan importantes como son la constancia, la responsabilidad o el trabajo en equipo. Por otro lado, quiero dejar constancia de la gran ayuda y labor que Manuel Garrido Satué ha realizado como tutor, sus aportaciones y esfuerzos han sido imprescindibles para la ejecución de este trabajo de fin de grado, complementado estos valores, con una total entrega que ha facilitado enormemente el camino del aprendizaje.

Por último, también quiero agradecer con este trabajo a mi familia, por el apoyo recibido incondicionalmente durante toda mi vida, dedicando su tiempo y esfuerzo en formarme como persona, involucrando todo lo que ha estado al alcance para sacar lo mejor de mí.

Miguel Perozo Torrents

Sevilla, 2022

Resumen

El objetivo de este trabajo viene dado por la ausencia que existe de un control remoto para las máquinas lanza pelotas de tenis. Actualmente son pocas las facilidades que nos ofrecen estos dispositivos a la hora de poder controlarlos desde el otro lado de la pista. Si quisiéramos programar una secuencia de lanzamientos y estuviéramos solos, tendríamos que programarlas manualmente, dar la vuelta a la pista y ponernos a jugar tras haber perdido los dos o tres primeros golpes.

Además, las pocas máquinas en el mercado que existen con una aplicación real para el control remoto de las mismas, poseen unas Apps muy limitadas con funciones muy concretas. El manejo remoto queda muy reducido en comparación al manejo manual. Por eso, el objetivo de este trabajo consiste en implementar una App móvil que resuelva estos conflictos, ideando un buen control remoto de la máquina lanza pelotas y además proporcionando una mayor capacidad de maniobra desde el otro lado de la pista sin tener la necesidad de estar al lado de la máquina continuamente.

El trabajo, por lo tanto, ha consistido en crear una forma de asegurar una buena comunicación entre la aplicación diseñada (cliente) y el servidor al que se conecta (máquina lanza pelotas), para así ser capaces de ejecutar las mismas opciones que si lo hiciéramos manualmente.

El entorno de trabajo que hemos utilizado es Visual Studio 2019, el cual es una plataforma que nos permite trabajar con distintos lenguajes de programación. El entorno visual de la aplicación se ha diseñado en XML y la parte funcional en C#.

Además del diseño de la aplicación, también se ha diseñado un servidor que iría incluido en la parte de la máquina lanza pelotas ejecutándose dentro de una Raspberry Pi4. Esta implementación es necesaria, ya que a la hora de establecer una comunicación y desarrollar la App en cuestión necesitamos conectarnos con la propia máquina en sí, para así entender, corroborar y cerciorarnos del buen funcionamiento de la App.

El protocolo de comunicación empleado es el TCP/IP, el cual permite conectarse a una red WiFi para llevar a cabo la comunicación entre cliente y servidor. La comunicación es asíncrona y es el propio usuario quien posee el control íntegro de la App durante todo el proceso de ejecución. La aplicación cuenta con 4 interfaces distintas, una inicial para la conexión con el servidor y otras 3 para el control remoto de la máquina.

El resultado de esta implementación es una App robusta y dinámica, capaz de comunicarse con el servidor enviándoles distintas peticiones y ejecutando diversos modos de juegos rápidos, además permite practicar un golpe específico en una zona concreta del campo y crear, actualizar y borrar sets de lanzamientos, para luego ser ejecutados en la máquina lanza pelotas como entrenamientos. Dichos pueden ser implementados tanto conectados al servidor como sin conexión, además una vez registrados quedan almacenados en la memoria de la App para poder ser reutilizados en futuros entrenamientos.

Por lo que el contenido de este trabajo consiste en desglosar el funcionamiento de la App implementada y en cómo dichas funciones son capaces de ser interpretadas por la máquina lanza pelotas de manera remota.

Abstract

The aim of this project to provide a ball throwing machine phone application in order to have the best dynamic control of it on the tennis court. Nowadays, this items have a control interface in their structures, but requieres a manual control. The point of this work is based on create a WIFI link of comunnication between the app and the machine, making easier for the user to use the machine from the other side of the tennis court.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

A^*	Conjugado
c.t.p.	En casi todos los puntos
c.q.d.	Como queríamos demostrar
■	Como queríamos demostrar
e.o.c.	En cualquier otro caso
e	número e
IRe	Parte real
IIm	Parte imaginaria
sen	Función seno
tg	Función tangente
arctg	Función arco tangente
sen	Función seno
$\sin^x y$	Función seno de x elevado a y
$\cos^x y$	Función coseno de x elevado a y
Sa	Función sampling
sgn	Función signo
rect	Función rectángulo
Sinc	Función sinc
∂y ∂x	Derivada parcial de y respecto
$x_\circ$	Notación de grado, x grados.
Pr(A)	Probabilidad del suceso A
SNR	Signal-to-noise ratio
MSE	Minimum square error
:	Tal que
<	Menor o igual
>	Mayor o igual
\	Backslash
⇔	Si y sólo si

1 INTRODUCCIÓN

La tecnología en estos últimos años ha alcanzado un punto clave en nuestras vidas, la sociedad está orientada a vivir en ella a través de un "click". Estos avances han repercutido notablemente en todo tipo de sectores: industrial, deportivo, comercial…, todos con el mismo objetivo, permitir que el usuario tenga una mejor experiencia aportándole una mayor comodidad o eficacia a la hora de realizar una tarea.

Es indudable que nuestro teléfono móvil se ha convertido en una herramienta indispensable para nuestro día a día. Los beneficios de estar interconectados, ya sea con otro usuario o con otros dispositivos, son mayores que las desventajas. El intercambio de datos, la opción de buscar información al instante o la ejecución de ordenes remotamente es un sinfín de las posibilidades que tenemos a nuestro alcance a través de un dispositivo electrónico conectado a la red.

Si centramos nuestro foco de atención en el sector deportivo observamos que las tecnologías se van incorporando a un ritmo vertiginoso. El ojo de halcón en el tenis, las imágenes paisajísticas mostradas por drones en las carreras de ciclismo, el video arbitraje o la recopilación de datos de rendimiento, médicos o estratégicos son algunos de ellos.

1.1. Objetivos

La idea de este trabajo se centra en otorgar una mayor comodidad a un aficionado de tenis que disponga de una maquina lanza pelotas, a través del desarrollo de una app que controle el funcionamiento del sistema.

Este objetivo viene motivado por el hecho de que la mayoría de estos dispositivos solamente cuentan con una interfaz de usuario implementada en la propia máquina, no suelen contar con un sistema que los controlen remotamente desde el otro lado de la pista.

El poder repetir golpes específicos, crear sets de entrenamientos particulares o modificar los parámetros de la máquina son unas de las varias funcionalidades de la que se podrán acceder a través de dicha aplicación.

En algunas máquinas dicha aplicación está ausente por lo que, si un particular pretender entrenar por sí solo debe poner en marcha la máquina, cruzar al otro lado de la pista y tras esto ponerse a jugar, habiendo perdido las primeras bolas que se hayan lanzado.

Por otro lado, otras marcas disponen de unas apps muy limitadas, cuya funcionalidad reside en una navegación rápida sobre la interfaz del sistema, sin poder crear nuevas propiedades o repetir secuencias de golpeos distintos, solamente navegar sobre las pistas de entrenamiento ya creadas.

Por lo tanto, el objetivo de este trabajo no es solamente crear una app de la que podamos darle al *play* para comenzar a jugar, sino de disponer de una app de la que podamos interactuar completamente sin limitaciones.

1.2. Funcionamiento y funcionalidad

El funcionamiento de una máquina lanza pelotas, en general, es similar entre los diferentes modelos que existen en el mercado. Si analizamos la siguiente figura podemos observar lo siguiente:

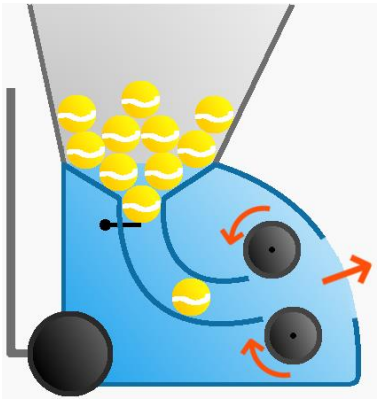

Figura 1.1: Funcionamiento de una máquina lanza pelotas de tenis

Una máquina lanza pelotas posee una tolva que se llena de pelotas de tenis para ser propulsadas. Estas pelotas caen hasta el fondo de la tolva hasta toparse con una barrera que les impide pasar al embudo que comunica la tolva con los rodillos.

Cuando la barrera se abre, la pelota cae al embudo y llega a dichos rodillos de aceleración. Estos rodillos están propulsados por dos motores que giran en direcciones opuestas. Recogen la pelota por medio de la rotación, la aprietan ligeramente y luego la catapultan fuera del dispositivo hacia la cancha de tenis.

Dentro de este dispositivo encontramos parámetros importantes que cambian según el modelo con el que estemos trabajando, como son, la capacidad de la tolva, la velocidad de lanzamiento, el efecto, el intervalo de tiempo de ejecución y la elevación del lanzamiento.

Por un lado, una tolva posee la capacidad de englobar entre 120-130 pelotas de tenis. Cuanto más bolas quepan en la tolva, menos interrupciones habrá entre lanzamientos y más eficiente será el entrenamiento.

La velocidad de lanzamiento también resulta ser un parámetro esencial, los principiantes prefieren jugar con pelotas lentas, mientras que los avanzados prefieren darles mayor velocidad a sus golpes. La velocidad en este tipo de dispositivos oscila entre los 30 km/h y los 130 km/h. Por lo tanto, la velocidad de lanzamiento también posee un gran espectro.

El efecto que se puede impregnar en un lanzamiento resulta ser uno de los parámetros principales por la que una máquina lanza pelotas se diferencian de otras. Este parámetro aporta una gran versatilidad en las posibilidades del entrenamiento. La máquina deja que ambos rodillos de aceleración funcionen a diferentes velocidades para así impregnar dicho efecto. Si la máquina gira el rodillo superior más rápido que el inferior, crea un *topspin*. Si el rodillo inferior gira más rápido, crea una *rebanada*.

Por otro lado, el tiempo de ejecución es un parámetro que define el tiempo que transcurre entre que la máquina espera lanzar una bola y la siguiente. Este parámetro suele oscilar entre los 2 segundos y los 13 segundos. Para los principiantes resulta excelente contar con un tiempo suficiente para preparar la posición del golpeo y mejorar la técnica. A medida que se va aumentando el nivel, se puede ir disminuyendo este intervalo de ejecución.

Por último, la elevación es un parámetro del dispositivo que permite adaptar al jugador a distintas situaciones del partido. Poder cambiar este parámetro ofrece prácticar golpes más profundos o recepciones más bajas a ras de la red. Este parámetro se maneja en grados y suele oscilar entre los 0º y 80º, donde cada aumento de grado se traduce en una petición de mayor profundidad del lanzamiento enviado por el dispositivo de la máquina lanza pelotas.

Además de los parámetros de la propia máquina lanza pelotas, encontramos dentro del dispositivo diferentes modos de juego. Una máquina lanza pelotas de tenis posee programado por lo general un modo de juego de juego principal o modo *home* que engloba una serie de modos "rápidos" como son: el modo *random*, el modo *2-line*, el modo *sweep*, el modo *depth,* y el modo *drop*. Además de esto, cuentan contro dos modos de juego adicionales. Por un lado, el modo de juego *manual*, que permite al usuario enviar repetidamente un lanzamiento concreto dentro de la cancha de tenis. Y, por otro lado, el modo de juego *drills*, que permite al usuario crear secuencias de lanzamientos en diferentes puntos de la pista.

Por un lado, el modo *home* usa los parámetros que tiene la máquina por defecto en ese momento y puede ejecutar, el modo *random*, para enviar lanzamientos de manera aleatoria por toda la pista, el modo *2-line*, que se encarga de enviar lanzamientos lejanos al fondo de la pista, el modo *sweep*, que envia bolas de un lado al otro de la cancha de tenis, el modo *depth*, que se encarga de enviar lanzamiento desde el fondo de la pista hasta la red o viceversa o el modo *drop*, que se encarga de enviar bolas cerca de la red.

El modo *manual* se encarga de perimitirle al usuario generar un golpe específico en una posición concreta de la pista. En este modo es el cliente el que se encarga de asignarle los parámetros del lanzamiento en cuestión, dichos parámetros son: el número de repeticiones del lanzamiento, posición a la que se desea lanzar, velocidad de disparo, efecto de la bola, intervalo de tiempo de ejecución y elevación de lanzamiento. La siguiente imagen ilustra un golpe concreto en una zona concreta de la pista.

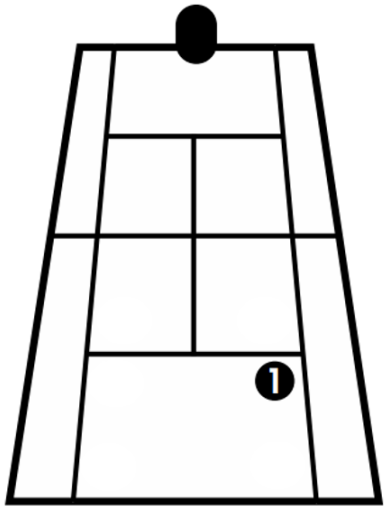

Figura 1.2: Ejemplo de modo *manual*

Por último, las máquinas lanza pelotas de tenis también poseen un modo de juego denominado *drills*, con el que el usuario puede crear *sets* de entrenamientos para crear secuencias de golpes y experimentar distintas situaciones. Este modo de juego ofrece una gran libertad al tenista y ayuda a mejorar la técnica en diferentes situaciones del partido. Es un modo de juego editable que permite al usuario crear nuevas secuencias de golpes continuamente. La siguiente imagen ilustra una distribución de hasta 6 golpes distintos por la pista preprogramados por el usuario.

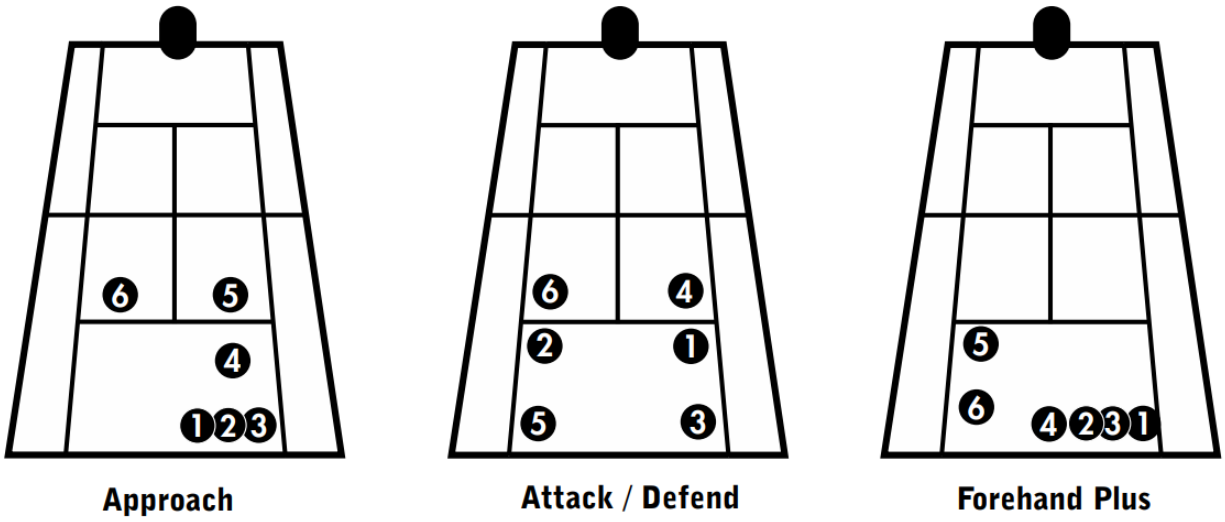

Figura 1.3: Ejemplo de modo *drills*

Una vez desglosado el funcionamiento estándar de una máquina lanza pelotas de tenis, es necesario analizar cómo se maneja y se implementa las funcionalidades de la misma. Todas las máquinas lanza pelotas de tenis cuentan con una interfaz de usuario (IU) que proporcionan un control manual del mecanismo. La siguiente figura recopila dicha interfaz.

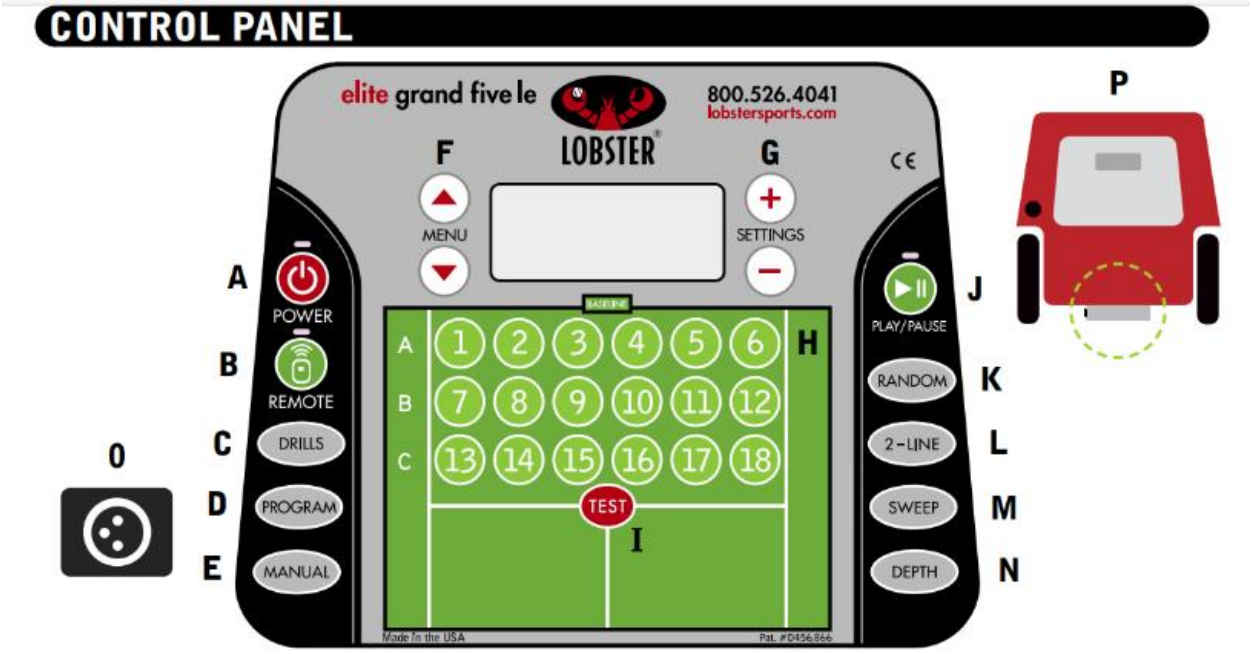

Figura 1.4: Panel de control de máquina Lobster Elite Grand Five LE Portable Ball Machine

Como podemos observar, esta herramienta incrustada en la parte trasera del dispositivo, posee la capacidad de manejar tanto los parámetros como los modos de juego explicados anteriormente. Esta figura 3.1, recopila la interfaz que el trabajo de la aplicación de móvil de control remoto ha intentado de reproducir en el dispositivo móvil.

Por un lado, la parte derecha de la interfaz posee los distintos modos de juego rápidos, lo que sería el modo *Home* anteriormente explicado (se puede evidenciar que no está incluido el modo *drop*, este nuevo modo es una mejora propuesta en este trabajo). En la parte izquierda poseemos la opción de oscilar entre el modo *manual* y *drills*, además de encender, apagar y conectarnos remotamente a la

máquina.

En la parte superior se presenta una botonera denominada *menú* y *settings* con los que aumentaremos los parámetros intrínsecos de la máquina, es decir, la velocidad, efecto, intervalo de tiempo de disparo y elevación. Y, por último, la parte central que contiene una distribución de la pista de tenis con una división numérica de lo que sería cada posición del campo rival visto desde el lado de la máquina lanza pelotas. Esta división permite enviar lanzamientos a dichas posiciones de la cancha rival. Además, se incorpora un botón de *test* para calibrar la máquina.

1.3. Limitaciones

Una vez que disponemos de los métodos de programación necesarios para llevar a cabo nuestra implementación, el siguiente problema que temenos que afrontar es el de los rangos físicos y dinámicos de la propia máquina Lanza pelotas.

Como cualquier mecanismo objeto de estudio, estos rangos de funcionamento definen los parámetros de configuración del dispositivo, restringiendo así directamente las capacidades que podemos aplicar en dicha aplicación para su propio control remoto.

Por lo que, a la hora de trabajar con una velocidad de disparo, un efecto de lanzamiento, un tiempo de ejecución y una elevación de lanzamiento tendremos uno espectro de valores que respetar al controlar la máquina lanza pelotas. Estos parámetros se han escogido de datos estándar de funcionamiento dentro del rango marcado por estos dispositivos en el mercado. El rango de los parámetros de la máquina se recopila en la siguiente tabla.

Parámetros	Mínimo	Máximo
Speed (km/h)	30	130
Spin (nivel)	2	9
Feed (s)	2	13
Elev (º)	0	50

Tabla 1.1: Rangos de los parámetros de la máquina lanza pelotas

1.4. Alcance del proyecto

Como se ha comentado anteriormente, este trabajo pretende reproducir la interfaz de una máquina lanza pelotas de tenis en una aplicación de móvil para que pueda controlarse remotamente. Por tanto, el contexto en el que se va a situar el trabajo en cuestión se puede reflejar en la siguiente imagen.

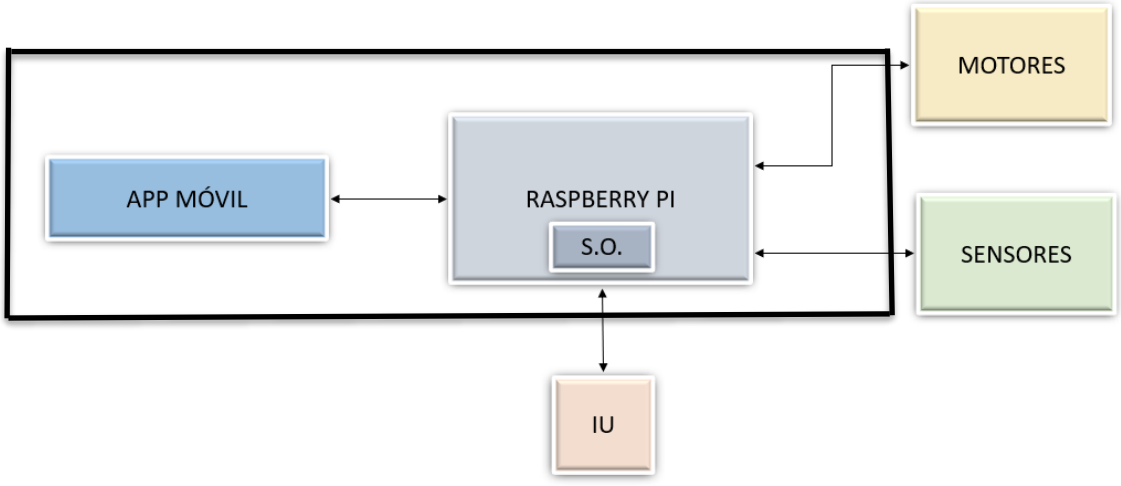

Figura 1.5: Diagrama de bloques de trabajo

El foco principal de este trabajo se centra en la parte de la app móvil identificada como cliente. Además, también se ha desarrollado la programación del servidor al que se conectaría dicha aplicación móvil y que controla la máquina lanza pelotas. El uso de ambas implementaciones permite conocer cómo es la sincronización y comunicación entre el cliente y servidor.

La máquina lanza pelotas (lado del servidor) contará con una Raspberry Pi4 a la que se le habrá cargado el programa del funcionamiento de la máquina. Dicho código responderá tanto a la propia interfaz de usuario como a la app móvil, para así modificar los controles y opciones que se hayan previsto llevar a cabo. Por lo que la parte de los motores del dispositivo, sensores e IU no serán objeto de estudio de este trabajo.

La manera en la que se ha implementado la aplicación de control remoto sigue el patrón que rige el funcionamiento y la funcionalidad de una máquina lanza pelotas de tenis estándar explicado anteriormente. La app contiene 4 interfaces diferentes, la primera trata de establecer la conexión con el dispositivo (*Mainpage*), la segunda engloba la interfaz de *Home* vista anteriormente, la tercera contiene la interfaz de *Manual* y la cuarta la interfaz de *Drills*. La distribución de interfaces dentro de la aplicación móvil se muestra en la siguiente figura.

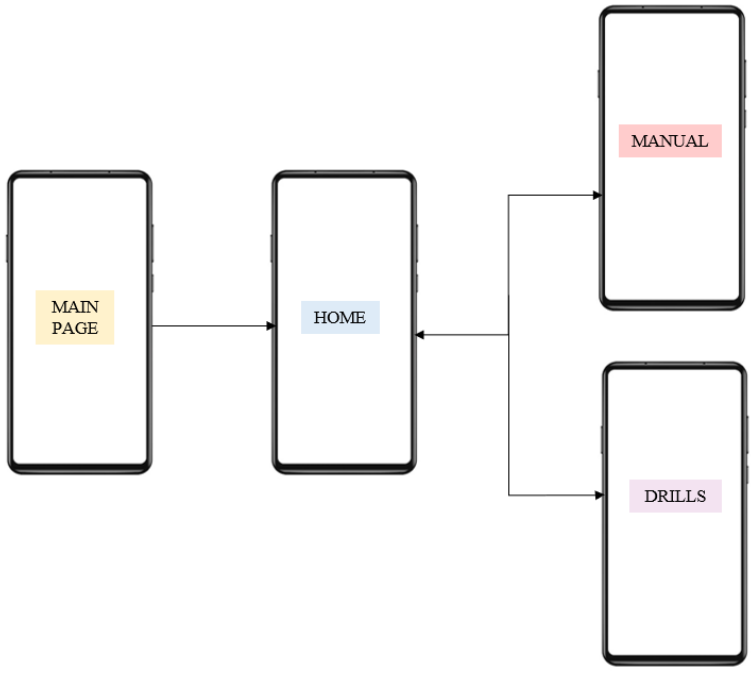

Figura 1.6: Distribución de interfaces

Dentro de cada interfaz poseemos la botonera correspondiente para acceder a los modos rápidos de juego o al cambio de los parámetros de la máquina.

El protocolo de comunicación entre la aplicación móvil de control remoto y la máquina lanza pelotas empleado es el TCP/IP, el cual permite conectarse a una red WiFi para llevar a cabo la comunicación. Dicha comunicación es asíncrona y es el propio usuario quien posee el control íntegro de la app durante todo el proceso de ejecución.

Además, la aplicación de control remoto cuenta con una gestión de errores que permite notificar al usuario cualquier problema durante la ejecución de la máquina lanza pelotas. La gestión de errores engloba notificaciones de problemas de comunicación, notificaciones de ausencia de datos y avisos de peticiones inválidas. El siguiente ejemplo ilustra una notificación de ausencia de datos en un campo obligatorio de la aplicación móvil.

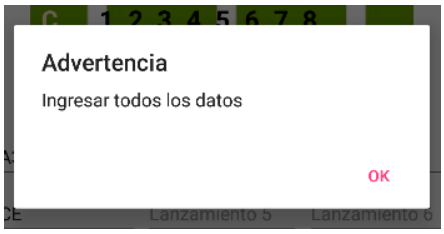

Figura 1.7: Ejemplo de gestion de errores

1.5. Herramientas

La implementación de esta aplicación vendrá codificada en la plataforma *Xamarin* que utiliza el lenguaje *XAML* a nivel de interfaz gráfica y *C#* a nivel de la lógica de la aplicación. La aplicación de desarrollo en la que se ha realizado la implementación es *Visual Studio 2022*.

1.5.1. Visual Studio

La plataforma de *Visual Studio* trata de un entorno de trabajo que posibilita la utilización de múltiples lenguajes de programación para el desarrollo de plataformas y aplicaciones para *Windows* y *macOS*, tales como *C++, C#, Java, Python,* etc.

Dicho entorno posee distintas versiones desde su creación en 1997. Para la ejecución de este trabajo se ha utilizado la versión *Visual Studio 2022*, la cual ofrece actualizaciones y mejoras de versiones más antiguas en "frameworks" y "severs" compatibles. *Visual Studio* permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cuaquier entorno compatible con la plataforma *.NET*.

Esta fácil escalibilidad e interoperabilidad del entorno posibilita una comunicación entre PC y dispositivos móviles, dispositivos embebidos o videoconsolas de una manera más sencilla.

1.5.2. Microsoft .NET

Esta implementación es un *Framework* (conjunto de herramientas) que tiene como objetivo solventar múltiples conflictos de comunicación entre códigos y permite ser empleada por la mayoría de las aplicaciones de *Windows*.

Esta plataforma se encarga de habilitar la posibilidad de crear aplicaciones y sistemas que sean independientes de la arquitectura física y del sistema operativo empleado en el dispositivo.

Esta fácil escalabilidad entre diferentes S.O. repercute en un ahorro del tiempo dedicado a la programación del código propiamente dicha. Además, *.NET* dispone de funciones ya diseñadas que se adaptan a las necesidades del programador y ayudan a mejorar la implementación que se desea alcanzar.

Actualmente, *.Net* es una de las plataformas para el desarrollo de software de mayor calado en el ámbito profesional.

1.5.3. Xamarin

Xamarin es un entorno de código abierto basada en .NET Framework. Este software posee la capacidad de compartir hasta un 90% de su código entre distintas plataformas utilizando un único código de lenguaje de progración, C#.

Este lenguaje cuenta con *Xamarin.iOS* y *Xamarin.Android* como clientes para implementaciones en las distintas aplicaciones que se quieran llevar a cabo. El objetivo principal es optimizar el tiempo de

programación, ya que para programar una app en *iOS* se necesita tener conocimientos previos de *Objetive-C* y para crear aplicaciones en *Android* se necesita poseer conocimientos de *Java*. Y gracias a este entorno, podemos crear aplicaciones a través del lenguaje *C#* que trabajen correctamente tanto en un sistema operativo como en otro en un mismo proyecto.

El siguiente diagrama ilustra la relación entre *Visual Studio* o *Xamarin Studio projects*, las librerías de *Xamarin* y las *APIs* de la plataforma. La tercera columna representa la comunicación de cualquiera de los sistemas incorporados al framework *.NET* de *Windows*.

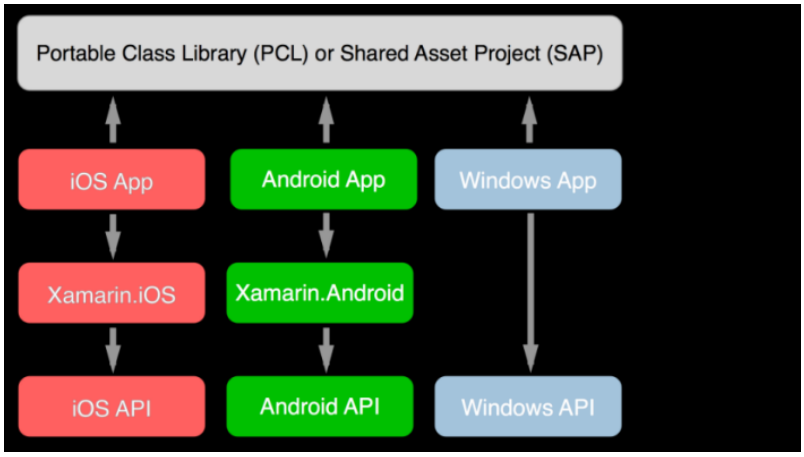

Figura 1.8: S.O. compatibles en Xamarin

1.5.4. SQLite

Dentro de la app que se va a desarrollar necesitamos una manera de organizar y almacenar la información que vamos generando, ya sea en tiempo real o de un día para otro. La solución a este problema reside en la biblioteca *SQLite*.

Esta base de datos trabaja como un sistema propio e independiente. Actualmente se encuentra bastante extendida para aplicaciones y servidores con poca cantidad de archivos. La gran ventaja de este sistema reside en que la base de datos y la app se ejecutan en paralelo, prescindiendo de la necesidad de un servidor externo.

Esta ausencia del servidor no ocurre con otros sistemas parecidos, por contra, la información que es capaz de guardar *SQLite* está mucho más limitada, pero para la app particular que se ha desarrollado es la mejor opción.

Dentro de las características principales encontramos que la base de datos *SQLite* se encuentra en un solo archivo, cuenta con una alta interoperabilidad de lenguajes de programación, lo que la hace muy versátil, además el poco contenido que se maneja dentro del sistema hace que sea rápida y de fácil acceso.

Esta herramienta será empleada para la creación y almacenamiento de nuevos sets de lanzamientos proporcionados por el usuario desde el modo *drills*. De manera que podamos guardar en una lista la secuencia de golpes que queramos ejecutar desde la máquina lanza pelotas.

1.5.5. XML

La manera en la que se comparten los datos tanto desde la máquina lanza pelotas como desde la aplicación remota están implementados en cadenas de textos con formato *XML*. Estos mensajes están encapsulados en un grupo con un tipo de mensaje general que luego se desglosa con la información particular que nos interese.

Este metalenguaje brinda la posibiliad de definir lenguajes de marcado con el propósito de compartir grandes volúmenes de información de forma sencilla, segura y fiable. La estructura que presenta *XML* se puede entender fácilmente y permite diferenciar las partes de un documento. La siguiente ilustración muestra la estructura de un documento *XML*.

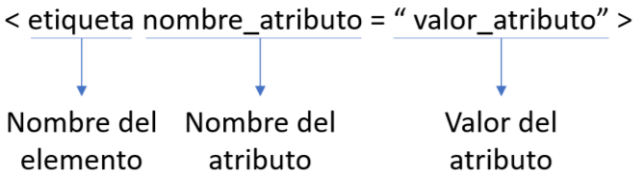

Figura 1.9: Elementos de un documento *XML*

Como podemos observar, este formato es ideal para presentar datos, como los parámetros de la máquina lanza pelotas o los modos de juego. El lenguaje de marcado recopila toda la información que se necesite de una manera fácil y legible, separando el contenido de la presentación. Su sencillo lenguaje permite que una persona con nulos conocimientos sobre la implementación, al leer el archivo *XML* pueda entender los datos que se están presentado. Además, dichos documentos *XML* presentan un rápido procesamiento.

Además, una implementación con *XML* sigue unas estrictas reglas de composición. Esto hace sencillo su análisis ya que siempre el proceso a realizar en este tipo de archivos está muy definido, creando así una jerarquía de etiquetas.

Por último, la herramienta de lenguaje *XML* añade una gran escalibilidad al código fuente. Podemos añadir nuevas etiquetas al documento, modificarlo y actualizarlo, sin necesidad de tocar el código ya creado.

2 ESTRUCTURA DE UNA APP PARA MÓVIL

El objetivo de este apartado consiste en describir la estructura empleada en la app de la máquina lanza pelotas. Para ello, es necesario desglosar el ciclo de vida de una app de forma general, para así, poder profundizar en aspectos más concretos de nuestra app particular.

2.1. Ciclo de vida

La anatomía de una aplicación móvil implementada en *Xamarin* responde a 3 etapas principales: una etapa de creación, una de implementación y otra de finalización. Dentro de estas etapas se puede acceder a subetapas secundarias volviendo una y otra vez a las principales, siempre y cuando no se haya alcanzado el final del proceso y la app no haya finalizado su ejecución.

Este diagrama responde al comportamiento de una app hecha para *Android*, es necesario señalar que la aplicación que se ha desarrollado también es compatible con *iOS*, siendo el ciclo de vida muy parecido al de *Android*. La diferencia entre un sistema operativo y otro reside en diferencias muy particulares dentro de la propia programación, pero a rasgos generales podemos englobar ambos sistemas dentro del mismo ciclo de vida de un mismo proyecto, de ahí que la solución implementada en *Visual Studio* se haga paralelamente tanto para *Android* como para *iOS*.

Los tres métodos que se proporcionan por defecto al crear un proyecto nuevo para una app son: *OnStart(), OnSleep() y OnResume()*. Son funciones protegidas y pueden contener o no código según los intereses del propio programador para controlar el ciclo de vida. En el caso particular del presente trabajo, el comportamiento de la app se ha controlado a través de funciones externas por lo no se ha incluido código en dichas funciones. Lo que es de obligatorio cumplimiento es inicializar la página en el constructor de la App que vayamos a utilizar como "página principal" o *MainPage*, para así darle un sentido lógico a nuestra aplicación y poder generar un espacio donde crear contenido.

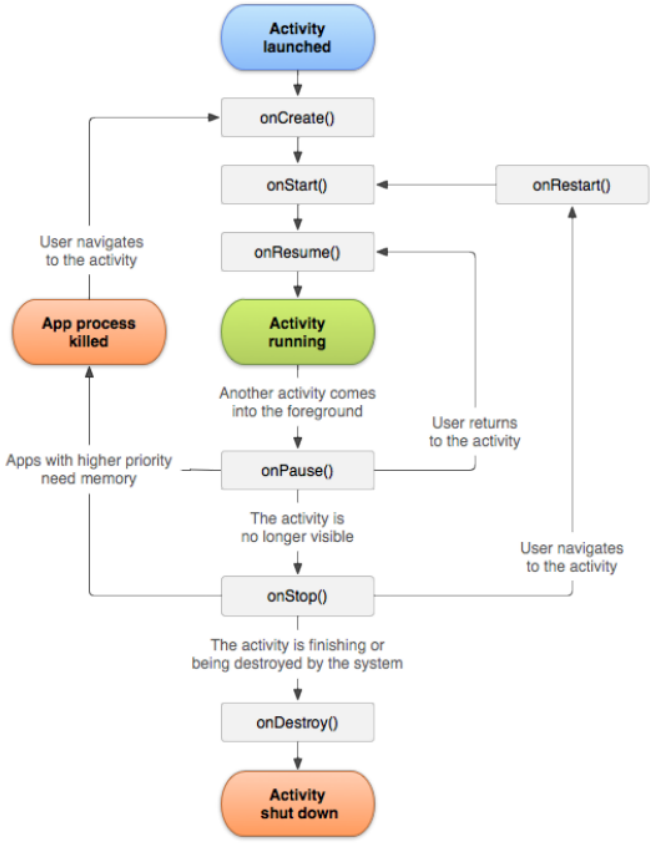

Figura 2.1: Ciclo de vida de una App

2.2. Anatomía de una aplicación

Una vez que conocemos el funcionamiento de la app a rasgos generales, es necesario controlar la familia de objetos que podemos incluir al generarla, desde lo más general como puede ser una vista de contenido, hasta lo más particular como puede ser la creación de un botón o una etiqueta.

Inicialmente, la creación de un app en *Visual Studio* ofrece la posibilidad de partir desde tres puntos de salida distintos. Se puede empezar la aplicación desde lo que se conoce como "Blank App" que no es más que una aplicación vacía ligada a una versión compartible de *Android* e *IOs*, sin ninguna implementación dada (Aplicación vacía). Esta versión ayuda a crear tu aplicación desde cero proporcionándole así libertad creativa al programador desde el primer momento.

La segunda opción que se otorga es un inicio de App con "control flotante", esta ayuda resulta interesante para el programador que tenga la idea desde el inicio de disponer de una barra deslizante vertical o panel en su aplicación. Y, por último, la tercera posibilidad que nos encontramos en el entorno de *Visual Studio* es la de "Con pestañas", que no es más que un inicio de la aplicación con secciones que se dividen en distintas pestañas que pueden ser programadas posteriormente por el usuario de una manera ya ordenada y preestablecida.

En el caso de la aplicación de la máquina lanza pelotas que se está explicando, se ha partido desde un modelo en blanco de manera que se ha ido incluyendo las necesidades a medida que iban surgiendo.

Una vez que elegido el modelo de partida entran en acción la programación orientada a objetos en la que se basa *Visual Studio* y por tanto *Xamarin*. Una aplicación de *Xamarin.Forms* puede estar constituida por una o mas *Pages* (Páginas). Una *Page* no es más que un área de trabajo que ocupa

toda la pantalla del teléfono móvil donde se incluyen los gráficos dinámicos y estáticos de programación, como etiquetas o botones. Normalmente existe una página principal *(MainPage)* que sería la página padre y a partir de esta, se generan todas las demás páginas hijos. Para crear a una página en *Xamarin.Forms* utilizamos el término *ContentPage*, una palabra reservada que genera una página nueva.

Una vez creado nuestro espacio de trabajo en la aplicación, entra en acción lo que se conoce como *Layout*, que no es más que distintos tipos de diseños en la que se basa nuestra página que acabamos de generar. Esta opción es una manera de ayudar a alcanzar la intención de programador de una forma más eficaz. Por ejemplo, si la intención del programador es utilizar muchos objetos de forma que no quepan en una única pagina estática, se utilizará el *Layout*: *ScrollView*, la cual, otorga a la página generada anteriormente la capacidad de deslizarse hacia arriba y hacia abajo. Si, por el contrario, no tenemos una idea fija de lo que queremos implementar el *Layout* que utilizaríamos sería un *ContentView*, el cual es el más genérico y establece un marco de trabajo similar a la dimensión de la pantalla. Las distintas opciones de *Layout* que tenemos en *Xamarin.Forms* son las siguientes:

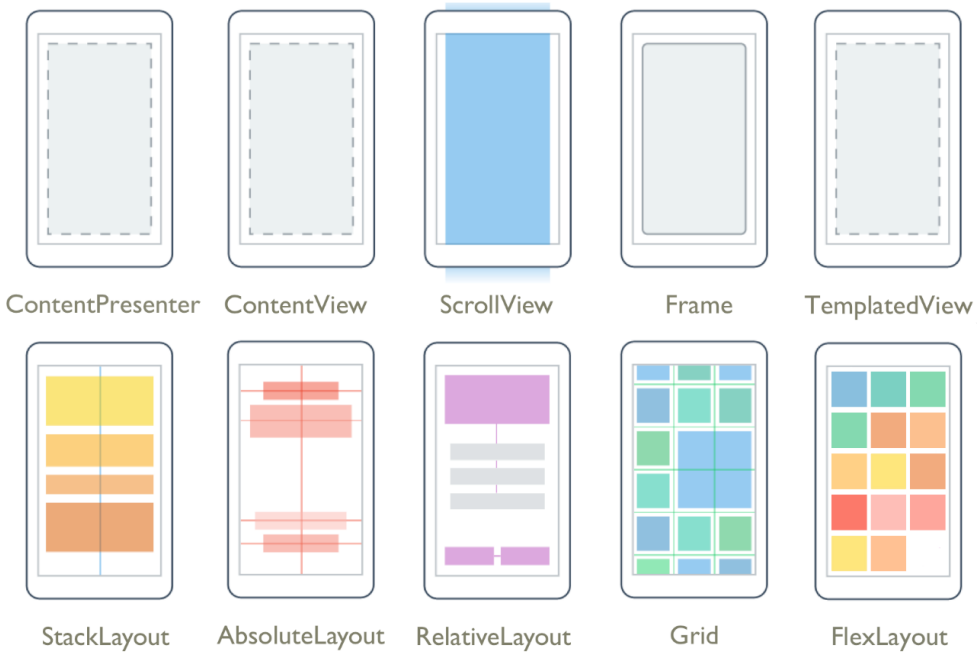

Figura 2.2: Tipos de *Layout* en *Xamarin.Forms*

Tras esto, el siguiente escalón de objetos se conoce como vista *View*, esta vista es una manera de encapsular diferentes objetos dinámicos o estáticos que tienen una misma función dentro de un *Layout*. Por ejemplo, si tenemos que crear un *Slider* (barra de deslizamiento) y una etiqueta numérica que varíe con dicho *Slider* aumentando o disminuyendo su valor, ambas acciones se podrían agrupar dentro de una vista en la página de trabajo. Esta manera de agrupar familias de objetos ayuda a tener controlado las distintas partes en la que se divide la página en cuestión.

Por tanto, recopilando todo lo anterior podemos resumirlo de manera que, entendemos que una aplicación puede partir desde un diseño en blanco o eliminar un diseño preestablecido. El inicio de la aplicación puede incluir una o varias paginas donde trabajar, tras esto se incluye en cada página los *Layouts* correspondientes según los intereses del programador y una vez que tenemos la idea fija de lo que se quiere implementar podemos agrupar los distintos objetos de programación en vistas, encapsulándolos así en un mismo marco de programación.

2.3. Objetos dentro de una aplicación

Como se ha nombrado anteriormente los objetos que se pueden incluir dentro de las paginas pueden tener una función estática, los cuales, son invariantes ante el tiempo o pueden tener una función dinámica, los cuales, podemos cambiar de funcionalidad o valor durante la ejecución de la aplicación. A continuación, se va a desarrollar y explicar los objetos empleados en la aplicación de la máquina lanza pelotas.

- "Label" (Etiqueta)
  Las etiquetas se utilizan para mostrar texto tanto de una como varías líneas, dentro de una aplicación pueden adoptar distintas formas, colores o tamaños. Estos ajusten varían con las propiedades intrínsecas del objeto Label en cuestión.

  Dentro de la aplicación desarrollada, las etiquetas ofrecen la capacidad describir la acción que se va a ejecutar al pulsar un botón, para distinguir entre una u otra, mostrar contenido numérico dentro de una lista o dar sentido y una ubicación a la página que nos encontramos a través de títulos, índices o apartados.

- "Button" (Botón)
  El botón resulta ser el elemento arquetípico cuando se definen los objetos que se pueden implementar a la hora de realizar una aplicación. La función de un botón consiste en habilitar una acción o proceso tras ser pulsado. Es decir, cuando se activa, el aspecto o la función de una parte de la aplicación cambia.

  La manera de implementar un botón en Xamarin.Forms puede ser tanto en xaml como en csharp y en ambas formas es necesario asignarle un evento "Clicked". Este método no es más que un manejador que guarda y genera un botón con el nombre que nosotros le hayamos otorgado anteriormente.

  La función asociada al evento tiene que recibir como argumentos dos parámetros. El primer argumento hace referencia al objeto que activa el evento del botón y el segundo nos proporciona más información acerca de dicho evento, este último argumento se utiliza sobre todo en la aplicación para detectar e imprimir posibles comportamientos anómalos durante la ejecución del botón.

  Los botones pueden ser síncronos o asíncronos, la diferencia entre una forma y otra es la siguiente: un botón síncrono espera al código anterior para ejecutarse, mientras que, si se trata de un botón asíncrono, la función que se lleva a cabo al pulsar el botón no espera a las instrucciones anteriores para ser ejecutadas, las procesa tras ser pulsado. Como el comportamiento de la aplicación que se ha desarrollado es completamente asíncrono ya que responde a una demanda aleatoria de peticiones según el cliente, los botones que se han diseñado son todos asíncronos.

- "Slider" (Deslizador de barra)
  Se trata de una barra que el usuario puede manipular durante la ejecución de la aplicación para seleccionar un valor en concreto dentro de un rango de valores. Este rango de valores trabaja desde un valor mínimo hasta un valor máximo definido previamente por el programador.

  Al igual que las etiquetas y los botones, los deslizadores tienen propiedades intrínsecas que permiten cambiarlos de forma o tamaño. La manera de generar un deslizador es parecida a la de un botón, en este caso utilizamos el evento *Valued_Changed*, el cual nos permite generar el deslizador con el nombre que nosotros le hayamos dado previamente. Recibe dos argumentos, por un lado, el argumento que permite manipular dicho deslizador y lo activa, y otro argumento que nos da información acerca de dicha activación. Este último, al igual que

con el botón, se utiliza para detectar e imprimir posibles comportamientos anómalos durante la ejecución del deslizador de barra.

El uso que se le da a este objeto dentro de la aplicación consiste principalmente en ligar una etiqueta con un deslizador, para así ver el dígito que se está seleccionando al manipular la barra. El deslizador permite manejar un amplio rango de valores de una manera precisa y fácil de tratar.

- "Entry" (Cajas de texto)
  Estas cajas de texto se utilizan para escribir caracteres por teclado en una sola línea dentro de la aplicación. Al igual que los objetos anteriores tiene propiedades que les permite cambiar de forma o darle un diseño particular.

  La ventaja principal de este objeto es la de permitir rellenar a modo de formulario cualquier sección que nosotros posteriormente queramos guardar o tratar. Por ejemplo, si queremos crear un set de lanzamientos a puntos específicos de la pista de tenis, se introducirán una serie de parámetros o códigos dentro de estas cajas de texto denominadas *entry*.

Como se ha visto, estos son los principales objetos relacionados con la parte gráfica de la aplicación que se han utilizado para el desarrollo de la aplicación de control remoto de la máquina lanzapelotas, ahora bien, dentro de dichos objetos, como ya se ha comentado anteriormente, hay una multitud de propiedades que permiten modificar dichos elementos a gusto del programador. Entre las más usadas destacan:

Etiquetas:

- "Padding": Separación entre el "layout" y el objeto que se introduce
- "Margin": Separación entre la página y el "layout"
- "TextColor": Color del texto
- "FontAttributes": Forma del texto, normal, negrita o cursiva
- "FontSize": Tamaño del texto
- "IsVisible": Controla el estado visible o invisible de un objeto
- "Text": Texto

Botón:

- "CornerRadius": Curvatura de las esquinas
- "BackgroundColor": Color del fondo
- "BorderWidth": Tamaño de la línea de borde
- "Clicked": Evento que genera el botón
- "Text": Texto

Deslizadores:

- "Ceiling": Redondeo hacia arriba de los dígitos
- "Valued_Changed": Evento que genera el deslizador
- "Margin": Separación entre la página y el "layout"

Cajas de texto

- "Placeholder": Texto introductorio que indica qué hay que escribir
- "Margin": Separación entre la página y el "layout"

Tras esto, podemos recapitular todo lo comentado anteriormente volviendo a la idea del inicio, es necesario conocer de manera clara y coherente el ciclo de vida de una aplicación. Paralelamente, el uso de los objetos de programación que incorporamos en el ciclo de vida ayuda a alcanzar nuestra meta, dándole un sentido lógico a nuestra idea principal. La siguiente imagen nos ayuda a comprender todo lo expuesto hasta ahora.

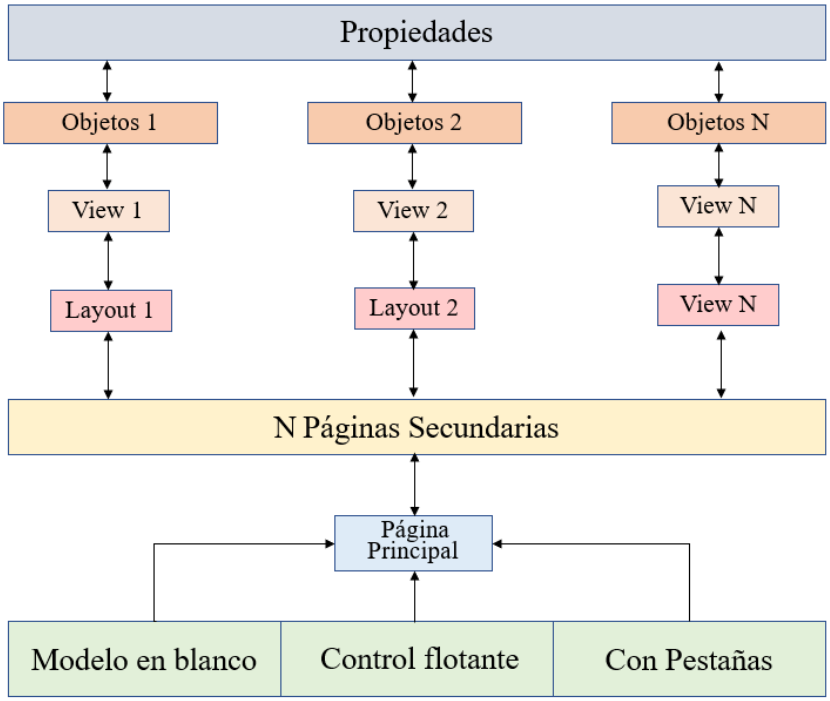

Figura 2.3: Esquema de una aplicación móvil con N paginas

2.4. Organización de una App móvil

Como hemos visto anteriormente, la manera que tenemos de organizar los objetos que creamos en una app viene precedida del tipo de *layout* que elijamos previamente. Sin embargo, en algunos casos no tenemos un modelo fijo de lo que queremos implementar o simplemente necesitamos una combinación de ambos, por lo que necesitamos darle nuestro toque personal a la página que estamos tratando.

Para dichas modificaciones dentro de los *layout* entran en juego lo que se conoce como *StackLayout*. Esta clase trabaja como una *View* dentro de una página, permitiéndonos agrupar objetos e información dentro de un mismo marco de trabajo que sea de interés.

El fundamental objetivo de esta clase *StackLayout* reside en apilar los objetos gráficos uno debajo de otros o unos al lado de otros, estableciendo así una organización coherente de la aplicación en cuestión. Todas las páginas que se han diseñado para la app de la máquina lanzapelotas vienen precedidas de un *StackLayout* de estas características.

Una vez que tenemos nuestro marco de trabajo bien definido para para trabajar queda el último paso, definir las filas y columnas. Este paso no es obligatorio, aunque es bastante recomendable, ya que te permite conocer de antemano como quedarían tus objetos dentro de una matriz, ayudando así a conocer las dimensiones y distribuciones de los objetos.

La matriz virtual se genera con la clase *Grid*, esta clase se trata de otro tipo de vista que se puede utilizar al mismo nivel que el *StackLayout*, es decir, como vista primaria o como vista secundaria, para así dividir en un matriz el espacio generado previamente por el *StackLayout*.

La clase *Grid* posee propiedades para definir filas, columnas, jugar con la alineación de ambas y crear matrices de distintos tipos y tamaños. El uso final de este tipo de vista es tener una aplicación ordenada en el espacio de la pantalla, pudiendo colocar cada objeto dentro de un marco de una o varias celdas de la matriz virtual. Un ejemplo del uso previo para la ayuda de la creación de una app con matrices sería el siguiente:

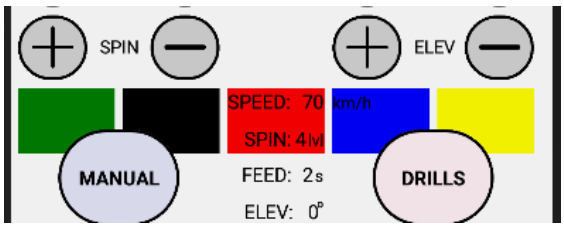

Figura 2.4: Muestra de Grid en App móvil

Como podemos observar en el extracto del simulador del teléfono móvil que alberga *Visual Studio*, la clase *Grid* nos ayuda a dividir el espacio de trabajo como se comentaba anteriormente. Los colores ayudan a evidenciar dicho suceso.

La gran parte de la propuesta como aplicación móvil para la máquina lanzapelotas sigue este patrón: utilizar como vista principal un *StackLayout* y dividir la zona en una matriz virtual con la clase *Grid* y a partir de ahí, coloca los objetos estaticos o dinamicos necesarios para completar dicha implementación.

Ya para finalizar, como útlima vista que se encarga de organizar el espacio de una aplicación y de la que también se hace uso en la aplicación de la máquina, es la vista *ListView*. Esta clase se encarga, como su propio nombre indica, de preparar una vista para presentar una serie de datos.

La colección de datos en un *ListView* facilita la recopilacion de información de una misma clase o grupo. Una manera sencilla y dinamica de organizar la informacion para listas grandes es utilizar la combinacion *ListView+Grid*, creando así una matriz de listas de informacion. Un ejemplo de esto mismo sería lo siguiente:

Figura 2.5: Uso de ListView

3 DESCRIPCIÓN DE LA APLICACIÓN CLIENTE

Ahora que conocemos cómo se implementa una aplicación desde Xamarin.Forms, este capítulo tratará de resolver las motivaciones que han llevado a realizar este proyecto.

Como se comentaba en la introducción de este trabajo, el objetivo principal es poder controlar remotamente el dispositivo desde el otro lado de la pista, sin la necesidad de estar físicamente al lado de la máquina y convertirlo en un entrenamiento más dinámico y eficiente.

Además, este trabajo pretende mejorar en algunos aspectos las funcionalidades establecidas hoy en día a nivel comercial en este sector. Aunque rencientemente, gracias al desarrollo de las tecnologías de comunicación, algunos fabricantes de máquinas lanzapelotas incorporan aplicaciones para dispositivos móviles en sus productos, el funcionamiento de las mismas es muy limitado, por tanto la implementacion que se explicará a continuación tratará de mejorar esto, extrapolando lo que es la interfaz de control manual de la máquina a la aplicación móvil del usuario, pudiéndose manejar completamente como si realmente estuvieramos allí.

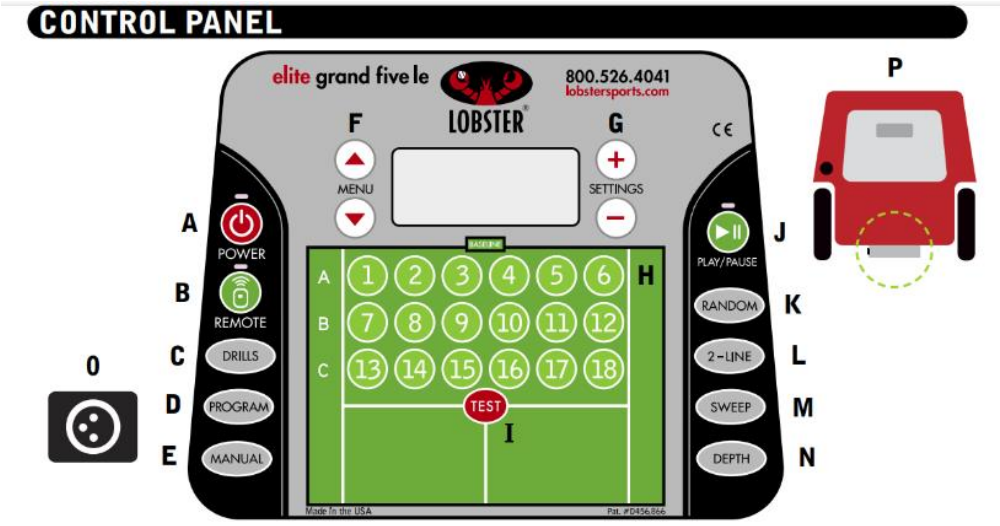

Figura 3.1: Panel de control de una máquina lanza pelotas

3.1. Interfaces de la Aplicación

La app que se ha desarrollado posee un total de 4 páginas implementadas, 1 principal (*MainPage*) y otras secundarias o de servicio (*ChildPages*).

En la página principal se espera a establecer la conexión TCP/IP con el servidor. En esta página es necesario introducir los parámetros necesarios para poder realizar esta conexión: dirección IP del

servidor y puerto de conexión, es decir, está constituida para establecer un puente de conexión via TCP/IP entre móvil y máquina, con el objetivo de poder intercambiar mensajes correctamente.

Tras haber completado dicha conexión, se accede a las interfaces de la aplicación propiamente dicha, desglosadas en una pagina llamada *Home*, otra página denomida *Manual* y una última llamada *Drills.*

El esquema de páginas que constituyen la aplicación es el siguiente:

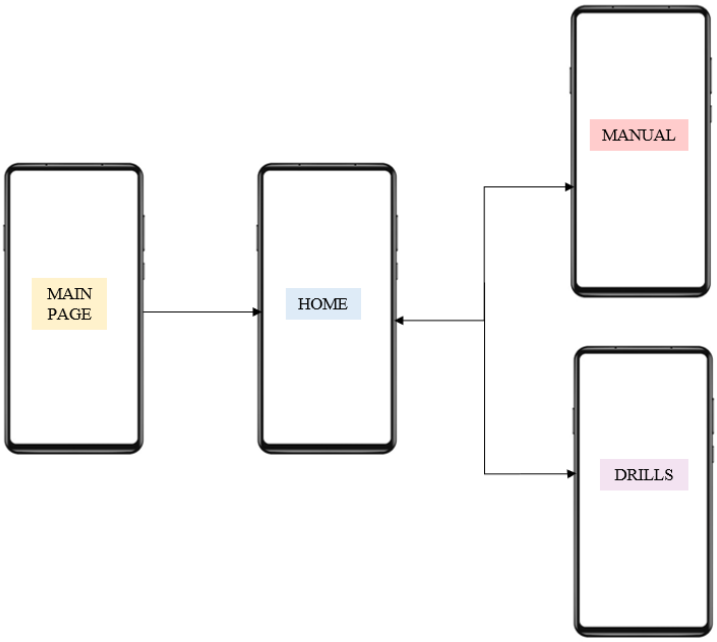

Figura 3.2: Distribución de Páginas de la App móvil

En esta figura de distribución de páginas, podemos observar cómo una vez que establecemos la conexión entre el cliente y el servidor (transición entre *Mainpage* y *Home*), podemos circular libremente entre las páginas *Home, Manual* y *Drills*, de manera que se manejen completamente a gusto del cliente las propiedades y acciones de la máquina, permitiéndose volver una y otra vez a la página que se desee. Esta libre circulación entre todas las páginas de la app será así hasta que muera la aplicación.

A continuación, se va a explicar con detalle la funcionalidad de cada página (*Home, Manual y Drills*), es decir, el objetivo y las acciones que se pueden realizar con la máquina en cada una de ellas. Por último, se explicará la página principal (*Mainpage*), dicha hoja se ha dejado para el final, a pesar de ser la primera que se implementa, debido a que alberga una serie de peculiaridades distintas a las demás y cobra más sentido una vez resumidas las demás páginas de la aplicación.

3.2. Interfaz de Home

Esta interfaz es la que se genera tras haber completado correctamente la conexión entre cliente y servidor. Resulta ser un menú principal que contiene un panel de control donde poder manejar los parámetros de la máquina, consultarlos y poder activar modos de juego concretos de una forma sencilla.

A esta página "semi-principal" se puede acceder en cualquier momento una vez iniciada la aplicación y es el puente que se encarga de enlazar tanto la página de modo *Manual*, como de modo *Drills*, cabe destacar que dicha página *Home*, es el nexo entre ambas, ya que no se puede acceder desde *Manual* hasta *Drills* y viceversa. Esto se ha hecho para no sobrecargar el código llevando la misma información de un lado a otro si no se va a utilizar.

Iniciando el simulador donde hemos implementado nuestra aplicación móvil, vemos la forma que tendría dicha página *Home* en una pantalla real de teléfono. El esquema que posee es el siguiente:

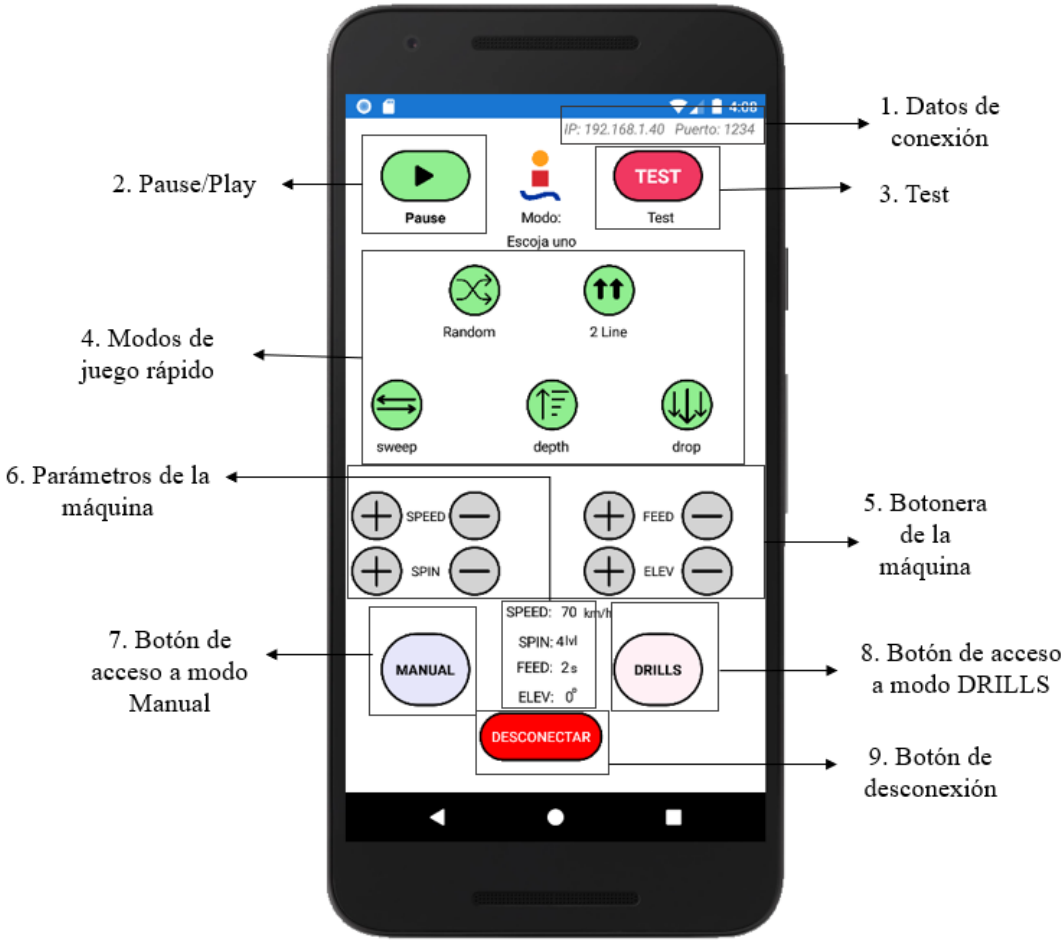

Figura 3.3: Interfaz de página HOME

A continuación, se va a desgranar por partes los aspectos y funciones que se muestran en la figura 3.3, con el objetivo de entender al completo qué se implementa en la interfaz de *Home*.

3.2.1. Datos de conexión

Los primero que vemos en la interfaz, concretamente en la esquina superior derecha, son los datos de IP y puerto al que el cliente, es decir, el teléfono móvil, se ha conectado. Esta información es una forma sencilla de corroborar que la conexión entre cliente/servidor la hemos hecho correctamente y nos hemos conectado al punto de acceso de la máquina lanza pelotas. Además, si hay varias máquinas en la pista de tenis, permite conocer a cuál de las máquinas nos hemos conectado.

3.2.2. Pause/Play

Este botón de *Pause/Play* viene por defecto en *Pause* y como su nombre indica, no es más que un botón que implementa una función que manda un mensaje de pausa o inicio a la máquina lanza pelotas.

Realmente esa parte de la interfaz está constituida por dos botones que se solapan, uno para el *Play* y otro para el *Pause*, la clave para el manejo eficiente se incorpora en la propiedad "IsVisible", dicha propiedad permite poner visible a un botón e invisible a otro, para que así pueda funcionar correctamente el que debe y viceversa.

La transición de *Pause* a *Play* se dará cuando el usuario haya cargado un modo de juego rápido o haya enviado un set de lanzamientos que quiera ejecutar, la manera de seleccionar estas funciones las veremos más adelante.

La transición de *Play* a *Pause* se dará durante la ejecución de lanzamientos por parte de la máquina lanzapelotas, es decir, se recurrirá a esta transición cuando el usuario quiera detener el dispositivo.

3.2.3. Test

Este botón permite calibrar la máquina. Envía un mensaje que consiste en una petición de calibración y es la máquina la que se encarga desde el otro lado, de procesar dicho mensaje y empezar a ejecutar el modo de calibración ya integrado en la máquina.

3.2.4. Modos juego rápido

Esta interfaz se encarga de seleccionar modos de juego de rápido para ejecutar en la máquina. Una vez recibido la petición del modo de juego seleccionado, la máquina se encarga de procesar el mensaje y ajustar los motores para llevar a cabo dicha función. Este paso de recepción de mensaje por parte de la máquina y procesamiento se lleva a cabo en todos los modos rápidos, pero particularizando el mensaje para cada modo seleccionado. La botonera encargada de elegir un modo rápido u otro dentro de la interfaz *Home* se muestra con mayor detalle en la siguiente figura:

Figura 3.4: Botonera de juegos rápidos en la interfaz de *Home*

Hay 5 botones y por lo tanto 5 modos de juego rápido:

- Modo *Random*: Es un modo de lanzamiento de pelotas aleatorio, las pelotas se lanzan por toda la pista a puntos aleatorios. Una vez seleccionado, la máquina espera a que el usuario pulse el botón de "Play" para comenzar. Ejecutar el modo Random implica mandar un mensaje a la máquina con esta petición junto con los parámetros de la máquina.

- Modo *2Line*: Es un modo de lanzamientos de pelotas al fondo de la pista, la máquina lanza pelotas manda bolas de la mitad del campo contrincante hacia adelante, este modo rápido tiene como objetivo ensayar los golpes lejanos.
- Modo *Sweep*: Es modo de lanzamientos de pelotas a lo ancho de la pista, la máquina lanza pelotas manda bolas a los laterales de pista del campo contrincante, este modo rápido refuerza los cambios de sentido en el intercambio de golpeos.
- Modo *Depth*: Es un modo de lanzamientos de pelotas que va desde el fondo de la pista contrincante hasta la red y viceversa, este modo rápido combina golpes largos en el fondo de la pista con golpes cortos desde la red.
- Modo *Drop*: Es un modo de lanzamientos de pelotas que se encarga de hacer dejadas en el campo contrario, el objetivo es mejorar la recepción en corto y mejorar el remate cerca de la red.

Cabe destacar que cada vez que seleccionamos un modo de juego distinto, vemos dicha selección de modo en la pantalla del modo *Home*. Inicialmente, cuando se ejecuta la aplicación la etiqueta que hace referencia a dicha función muestra un mensaje que escribe "Modo: Escoja uno", como se puede ver en la figura 3.3. Tras seleccionar un modo en concreto, aparece el nombre ahí reflejado para poder tener un seguimiento del último modo de juego que hemos seleccionado.

3.2.5. Botonera de la máquina

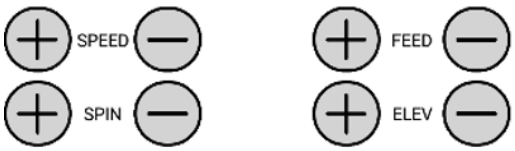

Figura 3.5. Botonera de los parámetros de la máquina lanza pelotas

Esta botonera está formada por una selección de 4 pares de botones que incluyen un "+" y un "-" para parámetro de la máquina. La funcionalidad de esta botonera es reducir o aumentar los parámetros de la máquina lanza pelotas a gusto del cliente. Los parámetros ajustados con esta botonera afectan a los modos rápidos de juego.

3.2.6. Parámetros de la máquina

La máquina lanza pelotas cuenta con 4 parámetros modificables durante su uso, los cuales son:

- *Speed*: La velocidad de lanzamiento enviadas por la máquina hacia el otro lado de la pista.
- *Spin*: El efecto que posee la bola al salir de la máquina lanza pelotas.
- *Feed*: El intervalo de tiempo entre un lanzamiento proporcionado por la máquina y el siguiente.
- *Elev*: La elevación con la que queremos que la máquina nos mande la bola hacia el otro lado de la pista.

Como toda máquina real, el funcionamiento que posee es limitado, no podemos enviar una bola hacia el otro lado de la pista con una velocidad exagerada o con un tiempo incoherente, por lo que hay unos límites en la capacidad de ajuste de dichos parámetros. Esta limitación a su vez permite preservar la seguridad y el buen funcionamiento de la máquina.

Dichos límites de la máquina se han ajustado lo más parecido a lo que se encuentra hoy en día en este tipo de dispositivos. Nuestra máquina en cuestión posee un uso de dichos parámetros que oscila entre los siguientes intervalos:

Parámetros	Mínimo	Máximo
Speed (km/h)	30	130
*Spin (nivel)	2	9
Feed (s)	2	13
Elev (º)	0	50

Tabla 3.1. Parámetros de la máquina

*El spin posee niveles porque es la manera de indicar si queremos una ejecución del lanzamiento cortado o liftado, dependiendo del nivel dicha ejecución será más plano o con mayor efecto.

Si optamos por salir de estos límites rápidamente saltará un mensaje en la aplicación móvil avisando de que la petición de aumento o disminución del parámetro no se ha ejecutado, recalcando que dicha petición se encuentra fuera de los rangos permitidos en la máquina. Un ejemplo de este mensaje es el siguiente:

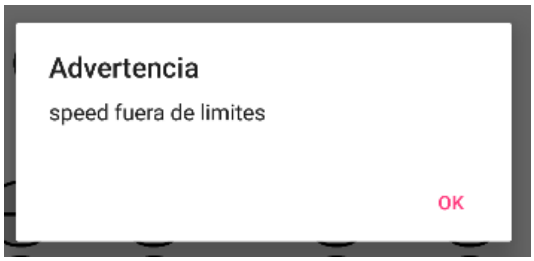

Figura 3.6: Mensaje de fuera de límites

Vemos en la figura 3.6, el mensaje que salta en la interfaz de *Home* una vez que el cliente pretende establecer unos limites fuera de los rangos preestablecidos por defecto en la velocidad de lanzamiento.

Cabe destacar que una vez que iniciamos la aplicación los parámetros con lo que comienza a ejecutarse la máquina se encuentran en la mitad de su rango disponible, como se ve a continuación:

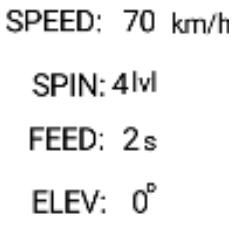

Figura 3.7: Display de los parámetros iniciales en la página HOME

3.2.7. Botón de acceso a modo Manual

Este botón es el que habilita la función que permite acceder a la página de modo MANUAL.

3.2.8. Botón de acceso a modo Drills

Al igual que con el botón de acceso a modo Manual, este botón permite el acceso al modo DRILLS.

3.2.9. Botón de desconexión

El botón de desconexión sirve para cerrar la conexión TCP/IP de los sockets que establecen la comunicación entre el móvil y la máquina. Al pulsar este botón nos aseguramos acabar con dicha conexión de una forma segura. Esta conexión entre móvil y máquina es la encargada del envío y recepción de datos continuamente.

3.3. Interfaz de Manual

La interfaz de *Manual* tiene como objetivo ensayar un golpe en concreto un número elevado de veces en una posición determinada de la pista. El número de veces que puede ejecutarse los lanzamientos a dicha posición viene limitado por la capacidad de la tolva de la propia máquina lanza pelotas, normalmente posee una capacidad de albergar hasta un número de 150 bolas dentro del dispositivo.

La secuencia de funcionamiento básicamente se resume en especificar la posición de la pista a donde el usuario quiere mandar los lanzamientos, programar los parámetros de dicho lanzamiento con la velocidad, el nivel de efecto, el intervalo de tiempo y elevación que se quiera dar y mandar el mensaje al servidor con dicha petición. Tras esto se volvería al menú de *Home* y se ejecutaría el botón de *Play* para comenzar a jugar.

La interfaz de *Manual* posee la siguiente distribución dentro de una pantalla de teléfono móvil real:

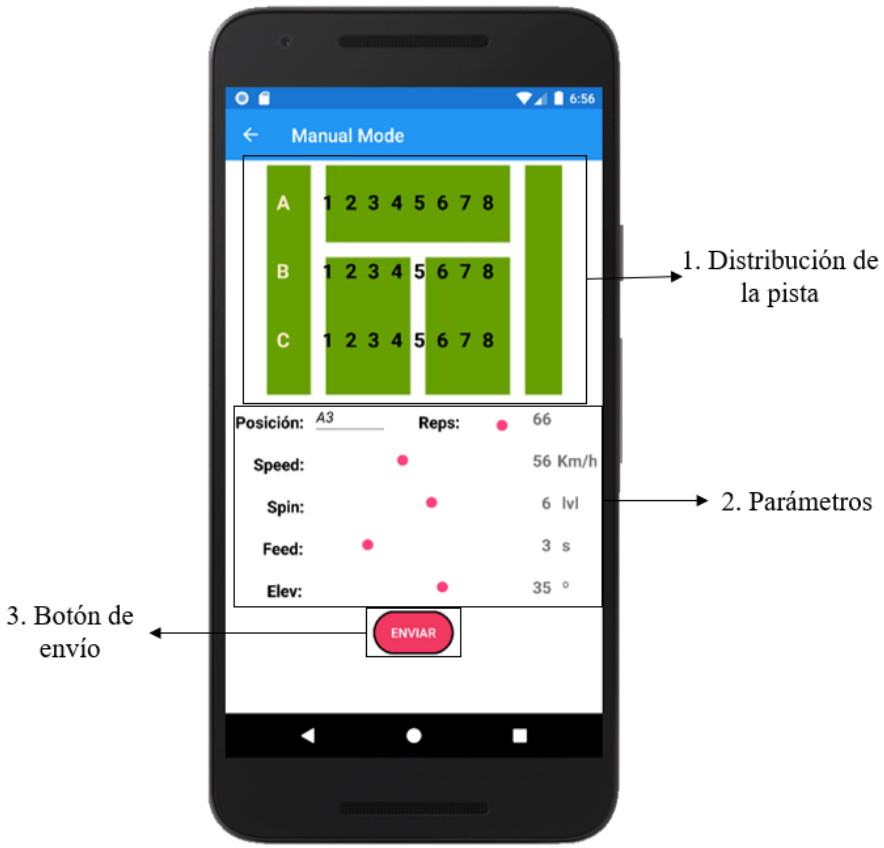

Figura 3.8: Interfaz de Manual

3.3.1. Distribución de la pista

Como podemos observar en la figura 3.8, en la parte superior de la interfaz se muestra un esquema de un campo de una pista de tenis, el lado opuesto de la pista donde se colocaría la máquina lanza pelotas. Dicha superficie representa el campo hábil donde la máquina es capaz

de lanzar pelotas. Los puntos donde la máquina puede lanzar las pelotas vienen precedidos por unos números del 1-8 repetidos en 3 filas A-B.

Esta distribución en forma de matriz es la manera visual que tiene el usuario de identificar el lugar en concreto donde quiere practicar el golpe dentro de la pista de tenis.

Tras identificar el lugar de la pista donde quiere practicar dicho golpeo pasaría a la programación de los parámetros para darle unas especificaciones concretas a dicho lanzamiento.

La configuración de dichos parámetros viene precedida de un *Entry*, que no es más que una casilla donde podemos escribir la posición del lanzamiento que previamente habíamos pensado ejecutar analizando la distribución de la pista, p.e "A3" o "C4", y unos *Sliders*, que funcionan como una barra deslizante donde poder ajustar los parámetros a gusto del cliente dentro del rango permitido para la máquina.

3.3.2. Parámetros

Estos parámetros son los datos modificables que el usuario envía a la máquina para especificar un golpe en concreto con unas características particulares. Dichos parámetros son los mismos que para la interfaz de *Home*. Se puede modificar dentro de esta configuración: la posición del lanzamiento que el usuario pretende recibir, el número de lanzamientos que quiere enviar desde la máquina, la velocidad de lanzamiento de las pelotas, el nivel del efecto del lanzamiento, la frecuencia con la que usuario quiere recibir los lanzamientos y la elevación con la que el usuario quiere recibir la pelota.

Dentro de los rangos de configuración de los parámetros nos encontramos las posibilidades de ejecutar lanzamientos en total de 24 posiciones distintas, con una cantidad de repeticiones de dichos lanzamientos de hasta 150 veces, con una velocidad de lanzamiento entre 30-130 km/h, con un efecto de spin de 9 niveles, dentro de una frecuencia de lanzamientos entre 2 y 13 segundos y una elevación de la recepción de la pelota que oscila entre los 0 y 50 grados. Esto parámetros son modificables en cualquier momento y para cualquier petición de lanzamiento en modo *Manual*.

3.3.3. Botón de envío

Dicho botón posee la funcionalidad de enviar los datos al servidor, es decir, de enviar a la máquina lanza pelotas las características del lanzamiento en concreto que hemos programado. Una vez enviado dicho mensaje, la función de dicho botón incorpora una especie de espera de confirmación, para cerciorarnos de que el envío que acabamos de mandar a la máquina ha llegado satisfactoriamente. La respuesta de la máquina tras el envío lo imprime la aplicación en forma de mensaje y posee la siguiente forma:

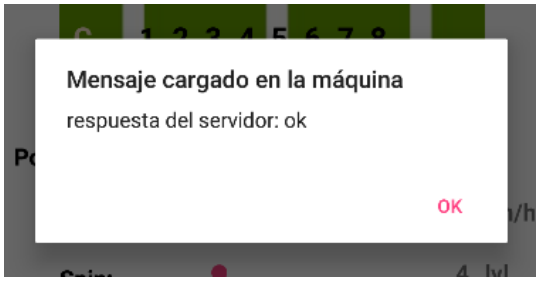

Figura 3.9: Mensaje de validación de envío

3.4. Interfaz de Drill

Como habíamos anunciado al principio del trabajo, uno de los objetivos de esta implementación entre cliente y servidor residía en poder implementar aspectos nuevos del manejo remoto de una máquina lanza pelotas, este nuevo añadido lo encontramos en el modo *Drills*.

La interfaz de modo *Drills* tiene como objetivo principal crear una secuencia de lanzamientos ideados por el cliente a diferentes puntos de la pista de tenis. Esta opción tiene la capacidad de ser configurada (off-line) tanto antes como durante la conexión entre cliente y servidor. Es decir, no hace falta establecer una conexión entre cliente y servidor para acceder a dicha interfaz.

Este modo en concreto cuenta con unas características distintas a las vistas anteriormente. Por un lado, tenemos establecido un *Layout* modo *Scrollview*, que no es más que una manera de asignar una página "deslizante" a lo largo de la pantalla durante la ejecución de la aplicación, esto será especialmente útil cuando se apile en una lista las secuencias de lanzamientos que hayamos programado.

Por otro lado, tenemos una interfaz dual. Esto implica que dependiendo de lo que estemos ejecutando en el modo *Drills* se verá en dicha página una pantalla u otra. Esta dualidad se ha implementado haciendo visible o invisible aquellos elementos de la pantalla que eran necesarios añadir o eliminar en cada momento de la operación. Esta propiedad ayuda a reciclar fácilmente la misma página con distintos elementos y no ramificar la aplicación con demasiadas páginas.

Pero lo significativo de este modo reside en la base de datos empleada denominada *SQlite*. Este sistema proporciona una base de datos capaz de almacenar los sets de lanzamientos creados durante el modo *Drills*, con el objetivo de recuperarlos de una vez para otra cada iniciemos la aplicación. Esta gran ventaja permite ejecutar los sets que hayamos programado tras apagar y reiniciar la máquina sin necesidad de crearlos cada vez que queramos empezar a jugar.

La manera que tiene la biblioteca *SQlite* de operar es con una base de datos que solo se encuentra en un archivo dentro del pc, es decir, funciona enteramente en memoria, por tanto, es muy rápida, no posee la necesidad de depender de un servidor donde almacenar la información, pero la cantidad de datos que puede almacenar está bastante limitada en comparación a otros métodos que sí disponen de un servidor. Sin embargo, para el uso que le vamos a dar de guardar sets de lanzamiento dentro de una app funciona correctamente pudiendo alcanzar las premisas que queríamos cumplir.

La gran ventaja de este modo reside en combinar la memoria de la que dispone con el poder de creación de nuevos sets, de ahí que podamos acceder al modo *Drills* desde la interfaz *Mainpage*, ya que tiene más sentido poder modificar y crear nuevos sets de lanzamientos antes de empezar el propio entrenamiento, para así, poder ejecutarlos nada más conectarse con la máquina y no tener que crearlos durante la sesión. Sin embargo, también poseemos la capacidad de acceder al modo *Drills* durante la conexión a través de la interfaz de Home. Esta implementación permite al usuario crear nuevos sets de lanzamientos durante su tiempo libre de una manera cómoda y sencilla.

El acceso a dicho modo Drills se puede ver en la siguiente ilustración que engloba tanto una página como otra:

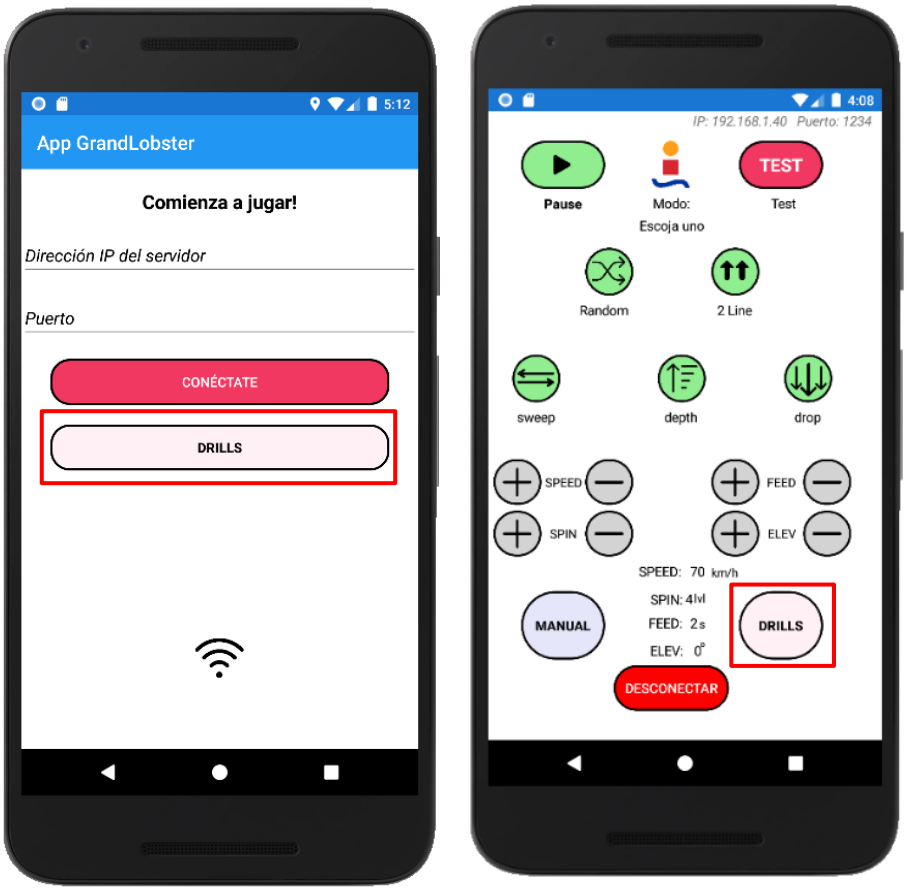

Figura 3.10: Acceso a modo *Drills* desde la Interfaz de *Mainpage* y de *Home*

Como podemos observar podemos acceder a la interfaz de modo *Drills* desde las dos páginas que se comentaban, una vez que accedemos a esta interfaz obtenemos la siguiente pantalla en el teléfono móvil.

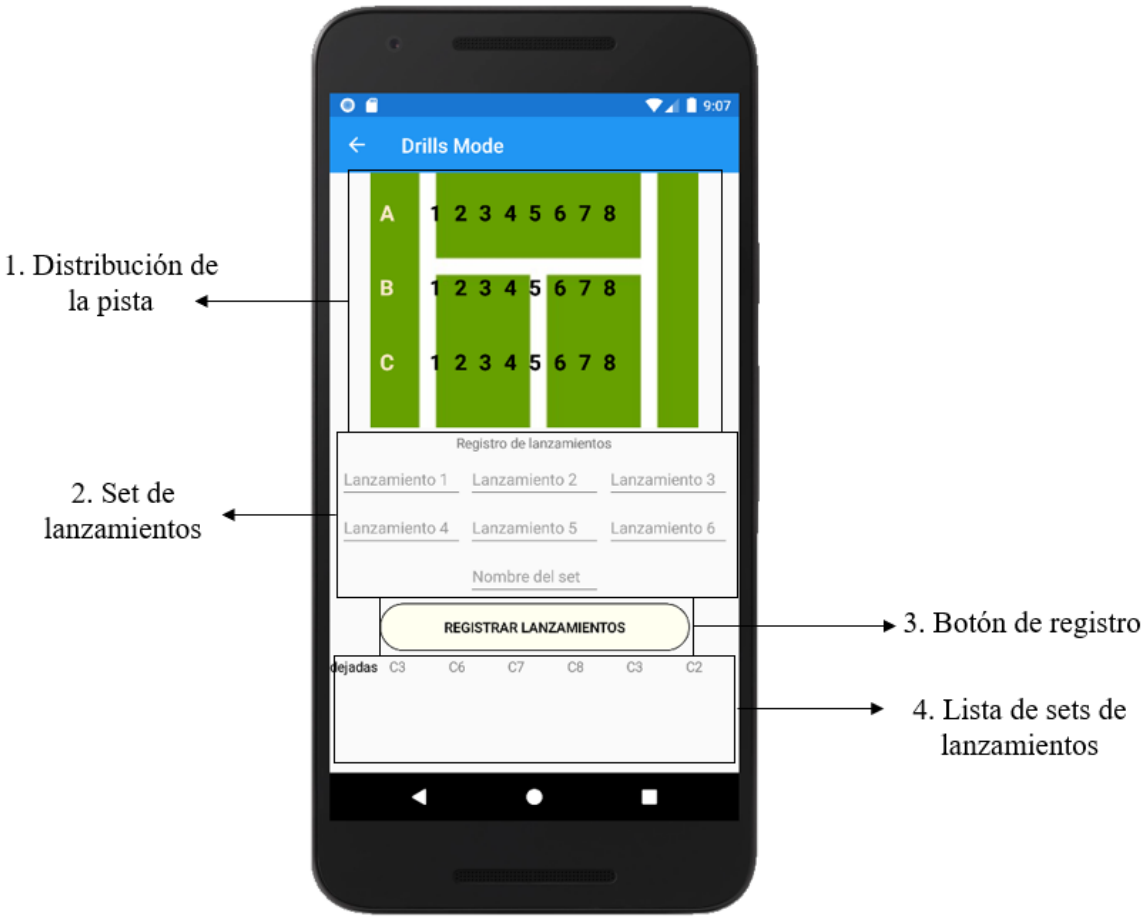

Figura 3.11: Script inicial del modo *Drills*

3.4.1. Distribución de la pista

Inicialmente podemos observar que la imagen de la pista de tenis en la parte superior de la interfaz es la misma que para la distribución en el modo manual. Tenemos una matriz de 3 filas (A-C), y 8 columnas (1-8). El objetivo es el mismo que antes, poder localizar los lanzamientos que queremos ejecutar de una manera rápida y visual.

3.4.2. Set de lanzamientos

Esta parte de la interfaz es la que se encarga de recoger las localizaciones que queremos almacenar en el set de lanzamientos, se hace a través de un *Entry* que recoge en una cadena de caracteres las posiciones de la matriz que se distribuyen alrededor de la matriz que se ha ideado anteriormente.

Se puede implementar hasta una secuencia de 6 golpeos consecutivos, aunque esta posibilidad no es obligatoria, podemos indicarle a la máquina que no queremos introducir un golpe escribiendo un guion "-" en el *Entry* correspondiente. También cabe destacar que el último *Entry* hace referencia al nombre del set que queremos asignarle a ese conjunto de golpes, dicha ID es una manera de identificar distintas secuencias de lanzamientos de manera rápida sin la necesidad de identificarlos por las posiciones que ocupan.

3.4.3. Botón de registro

Este botón implementa la opción de registrar el set de lanzamientos dentro de la memoria generada por la base de datos *SQlite*. Para ello valida que los datos metidos en los *Entrys* son los correctos, comprueba que no hay ninguno vacío y los almacena dentro de la memoria.

3.4.4. Lista de sets de lanzamientos

Esta lista refleja los sets de lanzamientos creados por el usuario. Es una manera de almacenarlos y poder operar con dichos sets explícitamente. Como podemos observar en la figura 3.12, la lista posee un set llamado "dejadas" que contiene una secuencia de 6 golpeos alrededor de la pista contrincante.

Figura 3.12: Lista de sets de lanzamientos

Si optamos por seleccionar uno de los sets creados y almacenados en la lista, accedemos inmediatamente a otra configuración de la pantalla. Esta configuración nueva es la ya mencionada anteriormente en referencia a la dualidad de la pantalla. Tras seleccionar un set de lanzamientos en concreto generamos una pantalla nueva con esta configuración dentro de una pantalla de teléfono móvil:

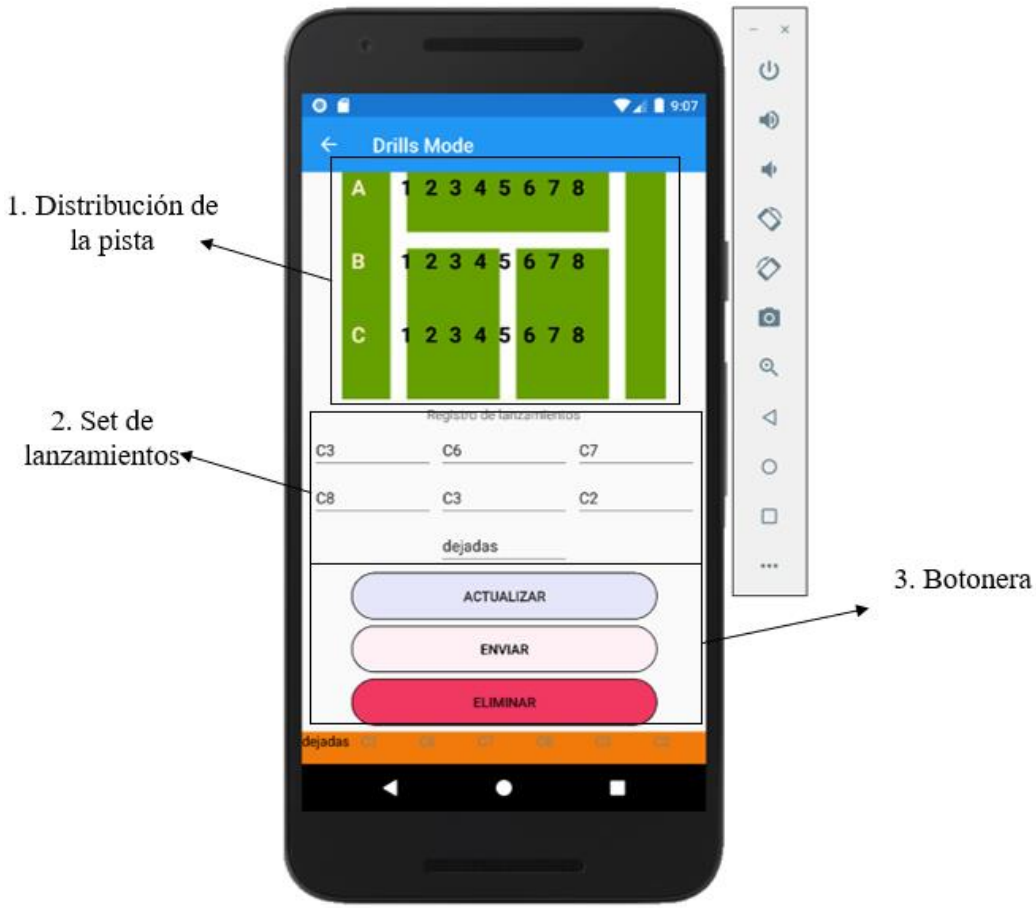

Figura 3.13: Script del modo Drills tras seleccionar un set de lanzamientos

3.4.5. Interfaz Drills secundaria

Como podemos observar en la figura 3.13, se genera una nueva interfaz dentro de la interfaz de *Drills* para ejecutar nuevas funciones dentro del set de lanzamientos seleccionado.

Por un lado, observamos como la interfaz superior que conforma la matriz de posiciones se mantiene, cumpliendo la misma función que ya se ha explicado.

Posee la misma configuración que la interfaz anterior, pero el objetivo es diferente. La intención de los *entrys* anteriores era registrar una secuencia de lanzamientos nueva, sin embargo, una vez que accedemos a esta interfaz a través de una secuencia ya preseleccionada, la intención de los *entrys* es de actualizar el valor de la posición en el caso de que el cliente quiera cambiar una parte de la secuencia de los lanzamientos.

Esta parte de la interfaz es la parte principal de la nueva pantalla creada. Tiene como objetivo llevar a cabo las funciones de actualizar, enviar y eliminar el set de lanzamientos previamente seleccionado.
Si queremos modificar la posición de un set de lanzamiento ya creado, basta con pulsar el botón de actualizar y la función que contiene dicho botón se encarga de actualizar el set de lanzamientos con los nuevos valores dentro de la memoria generada por la biblioteca *SQlite*.

Sin embargo, si queremos optar por enviar dicha secuencia de lanzamientos a la máquina, ejecutamos el botón de enviar, transmitiendo así un mensaje con las posiciones de dicha secuencia de lanzamientos.

Por último, tenemos la opción de borrar de la lista una secuencia de golpes con el botón de eliminar. Esta implementación se encarga de borrar permanentemente los datos del set seleccionado.

Estos tres botones siempre imprimen un mensaje por pantalla como los vistos anteriormente, para así cerciorarnos de que la opción que hayamos escogido se ha llevado a cabo correctamente. El botón de actualizar y eliminar imprime un mensaje de "éxito" o "advertencia" directamente tras ejecutar dicha operación y el botón de enviar imprime un mensaje que contiene la respuesta del servidor tras esperar dicha respuesta, al igual que se hacía en modo *Manual*.

3.5. Interfaz de MainPage

La interfaz de *MainPage* o página principal es la primera que se visualiza nada más iniciar la aplicación. El objetivo principal es establecer la conexión entre el cliente y el servidor de manera correcta a través de la IP generada por el servidor y el puerto que haya habilitado.

A pesar de ser la primera página que se genera tras iniciar la aplicación tenía más sentido explicarla tras haber descrito los distintos modos, ya que también posee la opción de acceder al modo *Drills* desde el inicio, como se ha comentado anteriormente.

Esta interfaz posee 2 *Entrys*, una para la IP a la que nos queremos conectar y otra para establecer el puerto que queremos usar y establecer como puente de comunicación entre ambos dispositivos. Una vez rellenados ambos campos, disponemos de un botón que contiene la etiqueta "conéctate", el cual dispone de las funciones necesarias para realizar la conexión entre el móvil y la máquina lanza pelotas.

Al pulsar el botón encargado de realizar dicha conexión, se imprime un mensaje en pantalla, indicando si la conexión se ha podido establecer o ha ocurrido algún problema en la comunicación. De esta manera nos cercioramos de lo que hemos hecho al haber procesado dichos datos con las funciones del botón de conexión.

La segunda funcionalidad de la interfaz la posee el botón de *Drills*, que realiza la misma función que el botón del modo *Home*, permite acceder a dicho modo sin la necesidad de estar conectado a la máquina, para así crear nuevos campos de lanzamientos en nuestro tiempo libre. Permite crear, borrar y actualizar nuevos sets de lanzamientos.

La pantalla del *Mainpage* tiene la siguiente visualización en un móvil real:

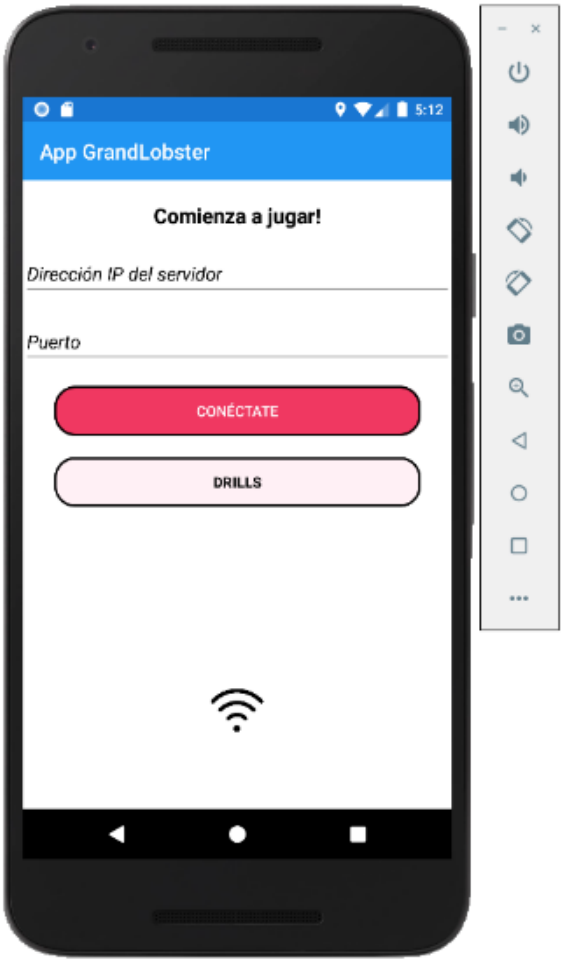

Figura 3.14: Script de MainPage

3.6. Resumen de las interfaces de la aplicación cliente

Hasta ahora hemos visto las 4 posibles pantallas que se pueden mostrar dentro de la ejecución de la aplicación móvil que se ha diseñado. Recopilando dicha información podemos concluir que tenemos lo siguiente:

- Una interfaz denominada *Mainpage* que se encarga de conectar el dispositivo móvil con la máquina lanza pelotas y de acceder al modo Drills sin necesidad de conexión previa.
- Una interfaz llamada modo *Home*, que se encarga de controlar los parámetros de la máquina lanza pelotas, de ejecutar modos de juego rápidos y de conceder el acceso al modo Manual y Modo Drills una vez que tenemos una conexión entre cliente y servidor.
- Una interfaz de modo *Manual*, que se encarga de ejecutar lanzamientos a un lugar en concreto de la pista unas veces determinadas por el usuario, programado con los parámetros de lanzamientos que el cliente haya establecido.
- Por último, un modo *Drills*, que se encarga de elaborar una secuencia de lanzamientos dentro de los márgenes de la pista para así crear distintos sets de entrenamientos. Tiene la capacidad de modificarlos, borrarlos o crear nuevos durante y tras la conexión. Posee memoria y por tanto no se borran a no ser que se lo indique el usuario.

4 APLICACIÓN SERVIDOR

En este capítulo se exponen los aspectos más importantes que se llevan a cabo en la aplicación servidor que se ejecuta en la Raspberry Pi4 integrada en la máquina lanzapelotas. Como se comentaba al inicio de este trabajo, para entender de forma coherente el uso que se le da a la aplicación es necesario comprender qué ocurre en este capítulo, ya que cliente y servidor están vinculados constantemente durante la ejecución de ambos sistemas.

Lo primero que cabe destacar es que el dispositivo que se ha usado para ejecutar el servidor que controlará en última instancia los actuadores de la máquina es una Raspberry pi4, la cual iría incorporada en el interior de la propia estructura de la máquina. Este dispositivo ejecuta un sistema operativo Raspian (Linux) al que se le ha instalado el framework .NET Core, en su runtime, de forma que se puedan ejecutar aplicaciones de consola desarrolladas en Visual Studio (Windows) programadas en el lenguaje C#.

4.1. Raspberry Pi4

Una *Raspberry Pi* es una serie de ordenadores de placa reducida, ordenadores de placa única u ordenadores de placa simple (SBC) de bajo costo con el objetivo de poner en manos de las personas de todo el mundo el uso de la informática y la creación digital.

La *Raspberry Pi4* cuenta con un procesador *ARM Cortex-A72*, una frecuencia de reloj de 1,5GHz, una memoria desde 1GB a 4GB dependiendo del modelo, conectividad *Bluetooth* 5.0, *Wi-fi* y *Gigabit Ethernet*. Además, cuenta con 40 pines, 2 micro *HDMI*, 4 *USB* y un *Micro SD Conector de audio Jack USB-C* (alimentación).

Figura 4.1: Raspberry Pi4

El uso de una *Raspberry Pi4* en dispositivos robóticos o de control como una máquina lanza pelotas nos aporta una gran flexibilidad y adaptabilidad debido a su reducido tamaño. Los pines de entrada/salida nos permite conectarnos a otros dispositivos electrónicos como LEDs o sensores. El sistema operativo que tiene cargado permite desarrollar y ejecutar el código tal y como lo haríamos

en un pc tradicional. Además, el uso de la Raspberry en implementaciones de autómatas está muy extendida por lo que disponemos de una gran cantidad de información para ejecutar nuestros intereses.

Por esta serie de motivos, la *Raspberry Pi* resulta ser una buena herramienta para el control y manejo de la máquina lanza pelotas en cuestión.

4.2. Descripción del programa Servidor

Dentro de la *Raspberry Pi4* ejecutamos un único código fuente realizado en *C#* para generar el servidor propio al que conectarnos desde la aplicación cliente. Dentro de la implementación, se incorporan varias fases.

Por un lado, disponemos una función llamada *SetupServer*, que solo se ejecuta una vez al iniciarse el programa. Es la encargada de crear un servidor TCP en la red en la que está conectada la *Raspberry Pi4* habilitando el puerto *1234* para la única conexión entrante que espera (el cliente desarrollado para dispositivos móviles). El puerto ha sido seleccionado arbitrariamente dentro de los posibles que quedaban libres para ser escogidos.

Una vez que establecemos dicha conexión entre el cliente y servidor, se empieza a ejecutar la fase secundaria del servidor, también asíncrona. Se trata de una función o *callback* que se ejecuta cada vez que se recibe un mensaje TCP. La función en cuestión encargada de procesar los datos entrantes desde la aplicación cliente se denomina *ReceiveCallback*.

Una vez que iniciamos la ejecución y entramos dentro de esta función *ReceiveCallback* se vuelve a llamar una y otra vez asímisma, para permanecer recibiendo información permanentemente por parte del cliente.

4.3. Comportamiento asincrónico del Servidor

La lógica de este servidor consiste en recoger los datos que le envía el cliente y ejecutar una serie de órdenes en función de dichos datos. Tenemos todas las ordenes posibles dentro de un mismo bloque *if, elseif, else,* a los cuales entra si una parte del mensaje corresponde con la condición impuesta dentro de dichas sentencias.

El funcionamiento básico de dicho procesamiento consiste en desglosar el mensaje que se recibe desde el cliente y actuar en base a ello, almacenando la respuesta y enviando un acuse de recibo al cliente.

El objetivo último de este procesamiento de mensajes consistiría en activar y desactivar los actuadores de la máquina lanzapelotas para llevar a cabo la instrucción especificada desde la aplicación de control remoto. Como ya se ha explicado, esta parte del control de actuadores de la máquina no forma parte del presente trabajo.

El siguiente ejemplo muestra el funcionamiento de la aplicación Servidor:

La primera fase, es el establecimiento de la conexión TCP entre la aplicación Cliente (control remoto) y la aplicación Servidor (máquina lanza pelotas). El servidor inicia una escucha y el cliente intenta establecer la conexión dada a la ip del servidor y puerto de escucha.

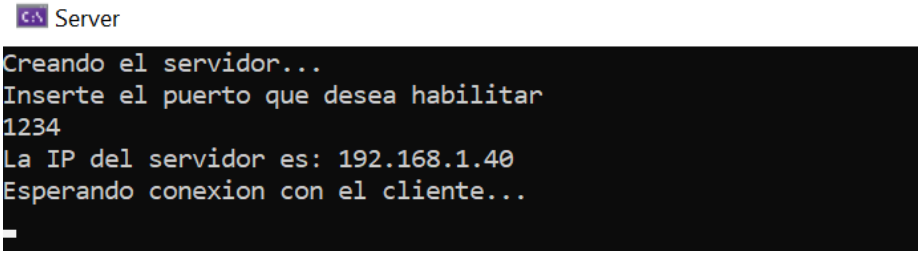

Figura 4.2: Inicio del servidor

La figura anterior muestra como el servidor generado demanda el puerto 1234 como accesible y que la IP a introducir desde la aplicación cliente es 192.168.1.40.

Una vez establecida la conexión correctamente con estos parámetros, comienza la segunda fase del programa. En este instante se llama a la función *ReceiveCallback*, la cual es el corazón del propio código fuente del servidor. Es la encargada de procesar la información que le llega y de invocar funciones adicionales para enviar respuestas al cliente. Todo esto se verá con más detalle en el capitulo de comunicación entre ambos sistemas.

Una vez obtenida una conexión correcta, la actualización de datos dentro del servidor se da respetando el mismo esquema que para aplicación. Los datos que se almacenan de la máquina se desgranan del mensaje *xml* que enviamos desde la aplicación cliente, para así desglosar su contenido de una manera más sencilla.

El servidor recoge dicho mensaje y lo procesa para llevar a cabo la petición. Se ha impreso por pantalla los parámetros que recibe el servidor con objeto de ver en todo momento lo que recibe la máquina lanza pelotas desde la aplicación cliente. A modo de ejemplo, la siguiente figura muestra una petición para aumentar la velocidad de disparo (*speed*) y otra petición para aumentar el intervalo de ejecución (*feed*) en la máquina lanza pelotas.

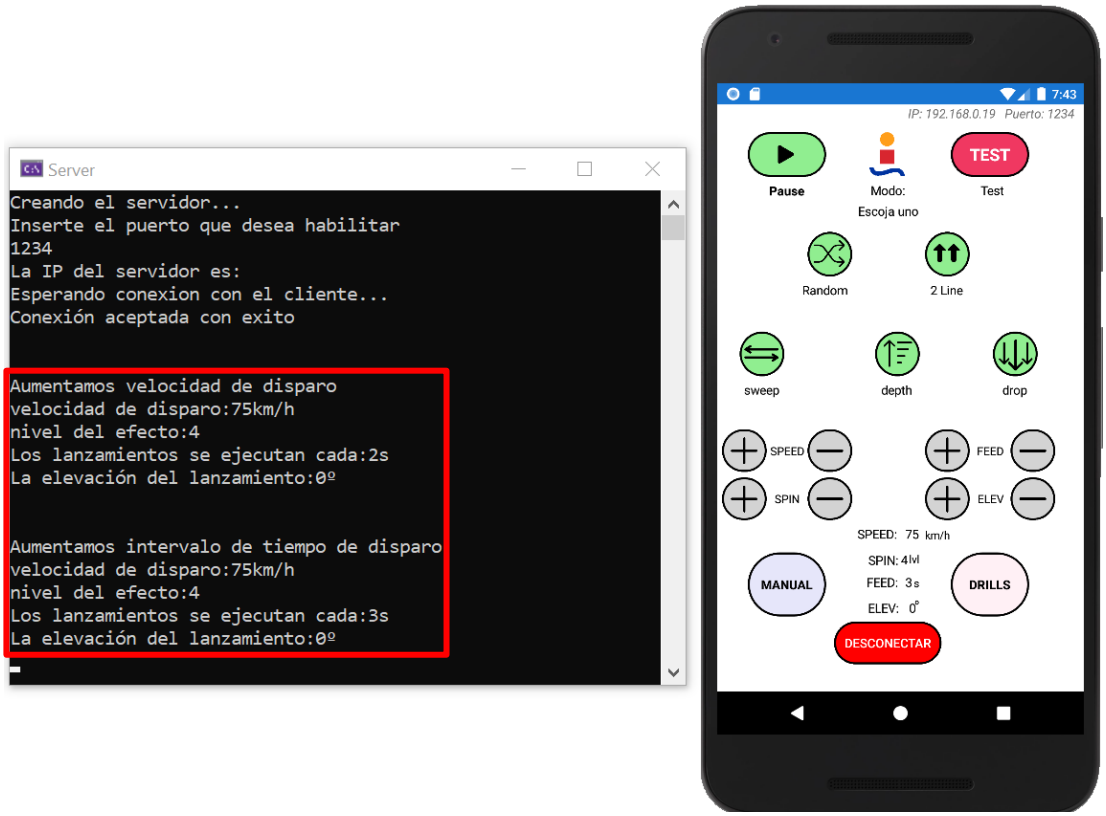

Figura 4.3: Ejemplo funcionamiento del Servidor I

4.4. Datos enviados desde la aplicación cliente

Una vez explicado cómo la información es recogida por el servidor desde la aplicación cliente, la manera que tenemos de operar para recibir otros tipos de peticiones ejecuta el mismo algoritmo.

Por un lado, si se opta por elegir un modo de juego rápido de la interfaz *Home*, el servidor identifica ese modo y lo ejecuta con los parámetros que contiene la máquina en ese instante. El siguiente ejemplo muestra cómo el servidor interpreta una nueva selección de modo rápido *random* con sus parámetros asociados.

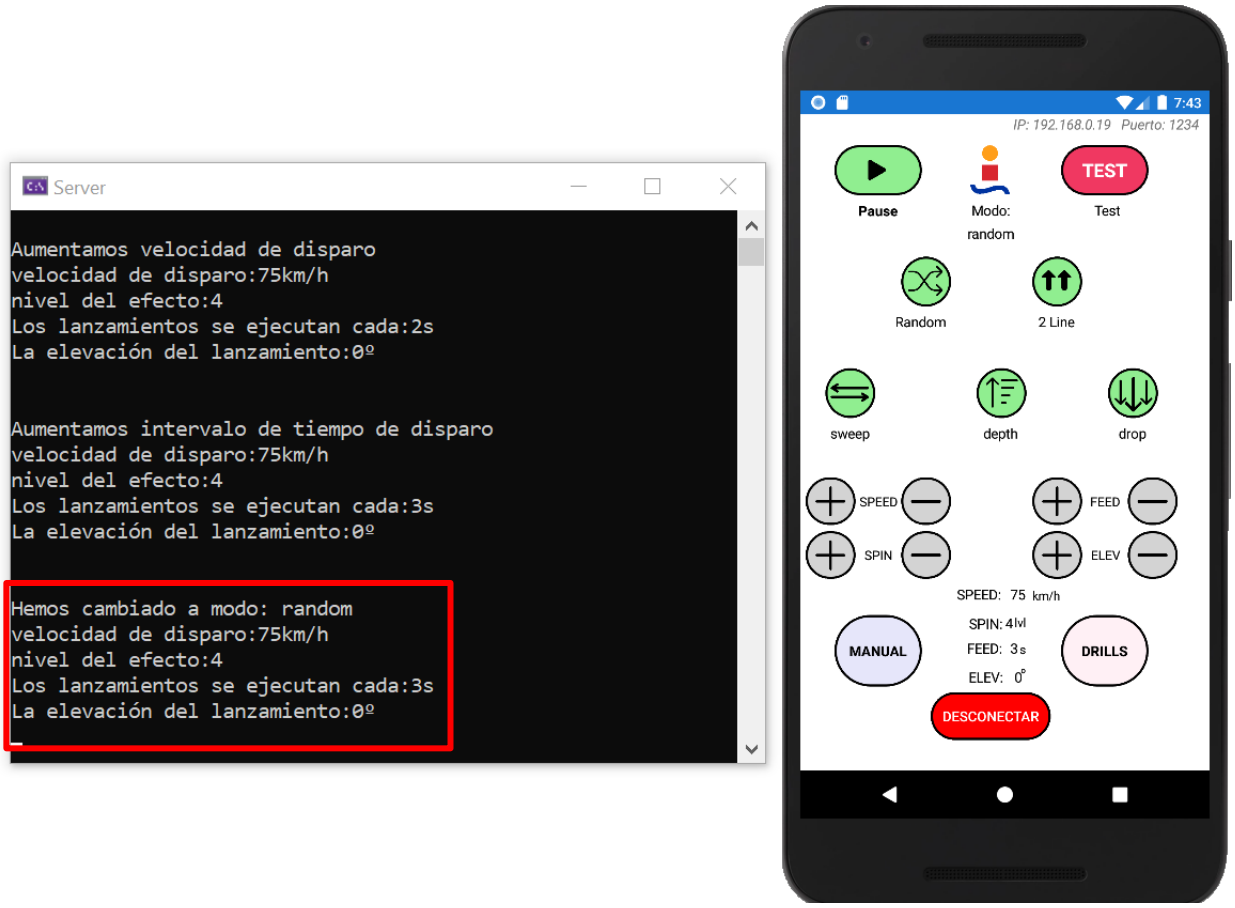

Figura 4.4: Ejemplo de funcionamiento del Servidor II

Por otro lado, la acción para activar y parar la máquina lanza pelotas se lleva a cabo mediante el botón de *play/pause* de arriba a la izquierda de la interfaz *Home*. Cuando el usuario pulsa dicho botón, el servidor está programado para identificar el modo de juego último que se ha establecido y ejecutarlo con los parámetros que contiene la máquina lanza pelotas en ese instante. El reflejo de este funcionamiento lo vemos en la siguiente imagen pulsando el botón de *play* tras previamente haber seleccionado el modo de juego rápido *random*.

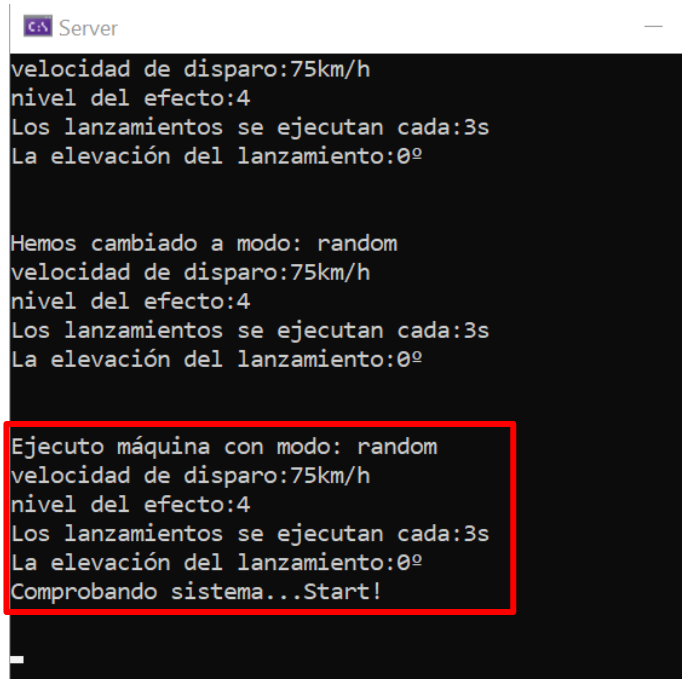

Figura 4.5: Ejemplo de funcionamiento del Servidor III

Los modos *Manual* y *Drills* funcionan en el servidor de manera distinta, ejecutan los parámetros que el usuario introduce en el momento de la petición, ignorando así los parámetros que poseía inicialmente dentro de la máquina.

Por un lado, el modo *Manual* cuenta con una mayor cantidad de datos que procesar, como son la posición a la que queremos efectuar el lanzamiento, el número de repeticiones y los parámetros íntrisecos de la máquina lanzapelotas (*speed, spin, feed* y *elev*).

El ejemplo siguiente muestra cómo el servidor recoge una petición desde el modo *Manual*, donde se solicita un envío de lanzamientos a la posición A4, para un número de 46 repeticiones, con una velocidad de disparo de 35 km/h, un efecto de nivel 3, un intervalo de tiempo entre un disparo y otro de 6 segundos y una elevación de 19º del envío.

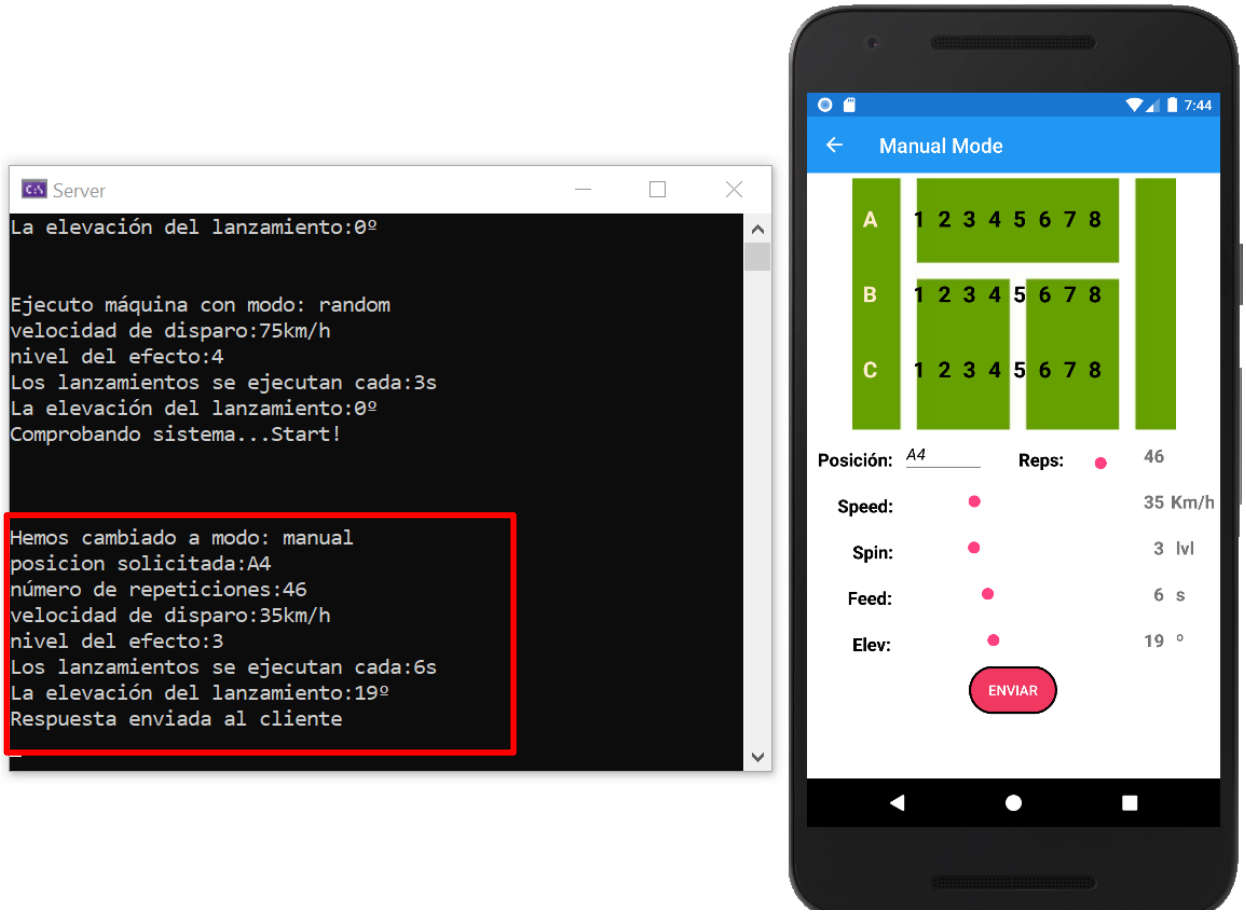

Figura 4.6: Ejemplo de funcionamiento del Servidor IV

Por último el modo *Drills* también funciona de una manera diferente al resto. Los datos que le llegan no son especificaciones de las propiedades de la máquina, sino lanzamientos a posiciones concretas. Por tanto, lo que se recoge en el mensaje enviado desde la aplicación cliente es el propio set de lanzamientos de las posiciones demandas por el cliente, las cuales se ejecutan con los parámetros ya almacenados previamente en el servidor que contiene la máquina lanza pelotas en ese instante. El ejemplo siguiente muestra cómo el servidor recoge los datos desde la interfaz de *Drills*

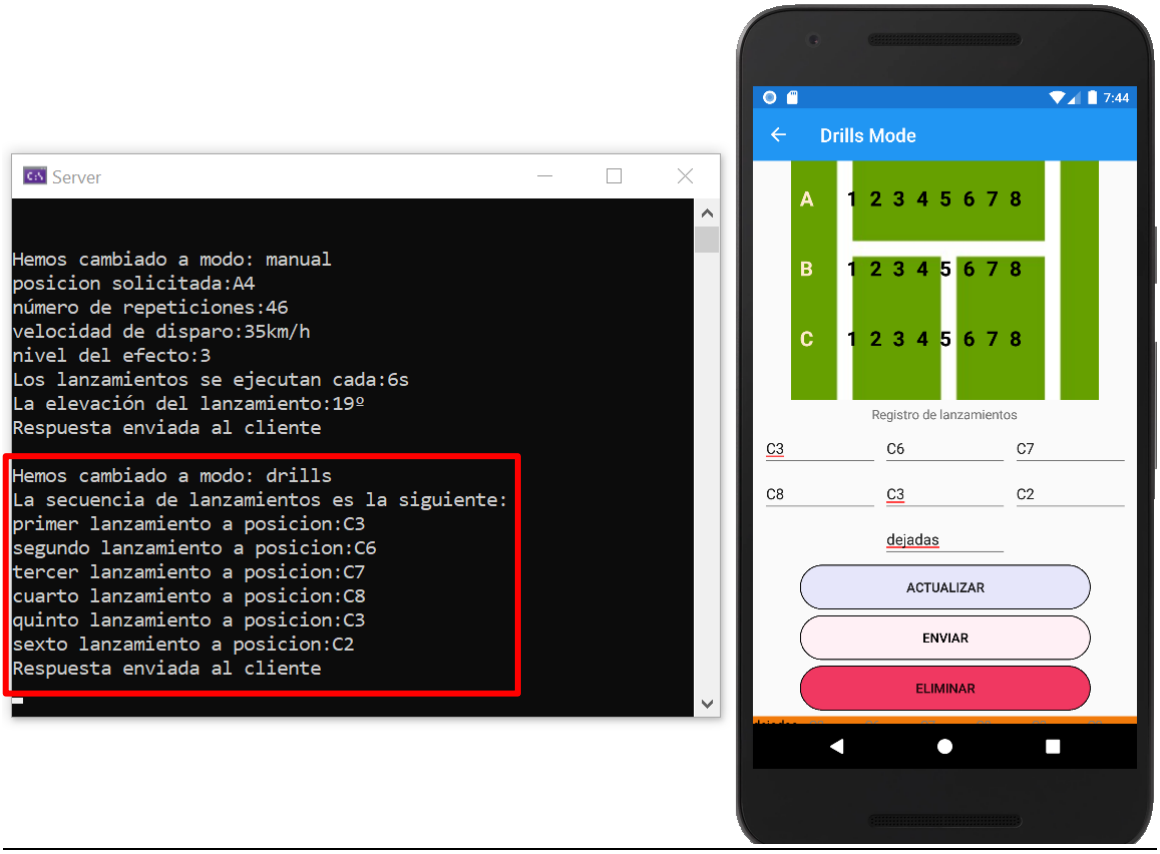

Figura 4.7: Ejemplo de funcionamiento del Servidor V

Como se observa, todas estas peticiones se realizan de forma asíncrona, el servidor está constantemente esperando peticiones del cliente, ya sean de distintos modos de juego o de cambios de los parámetros intrínsecos de la máquina. Esta serie de peticiones se pueden enviar de manera indefinida durante la ejecución de la aplicación remota, proporcionando así una aplicación dinámica con la que nos podemos comunicar con el servidor.

Por último, cabe añadir que la función *ReceiveCallback* posee subfunciones para enviar los mensajes de confirmación a la parte del cliente, esto son funciones auxiliares que entran en acción en caso específicos de confirmación como hemos visto anteriormente y que se ejecutan una vez que el mensaje se ha procesado y almacenado correctamente, en el caso de que no sea así se devuelve un mensaje de error a la aplicación para notificar al usuario que la petición no ha podido ser llevada a cabo.

Recapitulando todo lo anterior, podríamos encapsular la implementación dentro de un marco de dos fases, una para establecer la comunicación y otra para el procesamiento de los mensajes que se intercambian entre cliente y servidor, todo esto asu vez, trabajando de manera asincrónica.

5 COMUNICACIÓN

Este tema trata de abarcar el protocolo de comunicación utilizado, describiendo el contenido de los mensajes que se intercambian Cliente y Servidor para que la máquina lanzapelotas ejecute las instrucciones requeridas por el usuario, así como el protocolo de gestión de flujo de peticiones y respuestas.

El puente de comunicación se realiza utilizando la herramienta conocida como "sockets", los cuales establecen una comunicación TCP/IP (Protocolo de control de transmisión/Protocolo de Internet) que utiliza un conjunto de reglas estandarizadas que permiten a los dispositivos electrónicos conectarse a una red o entorno como Internet.

Una vez implementada dicha conexión, se desglosará la manera en la que se comparten los datos tanto de la máquina como de la app. Los mensajes están codificados en cadenas de texto con formato *XML*, los cuales encapsulan en un grupo un tipo de mensaje general para así luego desglosar la información particular que nos interese.

5.1. XML

La manera en la que se comparten los datos tanto desde la máquina lanza pelotas como desde la aplicación remota están implementados en cadenas de textos con formato *XML*. Estos mensajes están encapsulados en un grupo con un tipo de mensaje general que luego se desglosa con la información particular que nos interese.

Este metalenguaje brinda la posibiliad de definir lenguajes de marcado con el propósito de compartir grandes volúmenes de información de forma sencilla, segura y fiable. La estructura que presenta *XML* se puede entender fácilmente y permite diferenciar las partes de un documento. La siguiente ilustración muestra la estructura de un documento *XML*.

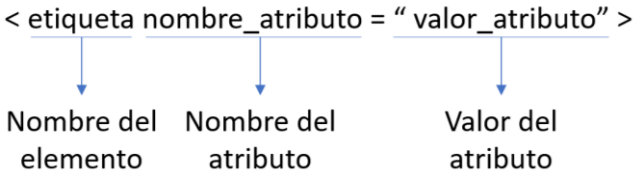

Figura 5.1: Elementos de un documento *XML*

Como podemos observar, este formato es ideal para presentar datos, como los parámetros de la máquina lanza pelotas o los modos de juego. El lenguaje de marcado recopila toda la información que se necesite de una manera sencilla y legible, separando el contenido de la presentación. Su

sencillo lenguaje permite que una persona con nulos conocimientos sobre la implementación, al leer el archivo *XML* pueda entender los datos que se están presentado. Además, dichos documentos *XML* presentan un rápido procesamiento.

Además, una implementación con *XML* sigue unas estrictas reglas de composición. Esto hace sencillo su análisis ya que siempre el proceso a realizar en este tipo de archivos está muy definido, creando así una jerarquía de etiquetas.

Por último, la herramienta de lenguaje *XML* añade una gran escalibilidad al código fuente. Podemos añadir nuevas etiquetas al documento, modificarlo y actualizarlo, sin necesidad de tocar el código ya creado.

5.2. Funciones del flujo de comunicación

El puente de comunicación se ejecuta de manera paralela y asincrónica tanto en la parte del cliente como en la parte del servidor. Por un lado, el servidor crea un socket de comunicación utilizando la red en la que está conectado a través de un puerto (configurable) del sistema que esté libre. El proceso se lleva a cabo dentro de una función denominada SetupServer.

Dentro del código del servidor se utiliza las clases de la biblioteca System.Net.Sockets, la cual dispone de una clase en concreto denominada *IPendpoint*, que permite crear una dirección de memoria con la IP y el puerto que el servidor haya establecido previamente.

Tras esto se ejecuta una función llamada *socket*, también de dicha biblioteca, que vincula a una variable tipo *socket* con el *socket* propiamente dicho, especificando el tipo que se quiere, en este caso orientado a conexión.

Una vez que se tiene de la dirección de memoria y la variable que funciona como representante del *socket*, se ejecuta una función denominada *bind*, que no es más que una función encargada de asociar a la variable del *socket*, la dirección de memoria. Tras esto la función *SetupServer* llama a otra función encargada de aceptar conexiones denominada *BeginAccept* y acaba su labor de creación.

En este punto del proceso, tenemos el servidor esperando una conexión. La siguiente figura muestra la aplicación Servidor esperando la conexión del cliente. Podemos observar en la siguiente figura, como el servidor hace uso de la herramienta de los *sockets*, habilitando así, la conexión para el puerto *1234*. Al final, se queda esperando aceptar alguna conexión externa con dichas especificaciones. Tras esto, se ejecutaría la aplicación de control remoto de la máquina, y se introduciría la IP del servidor y el puerto al que nos queremos conectar. Si la conexión es exitosa se daría lo siguiente:

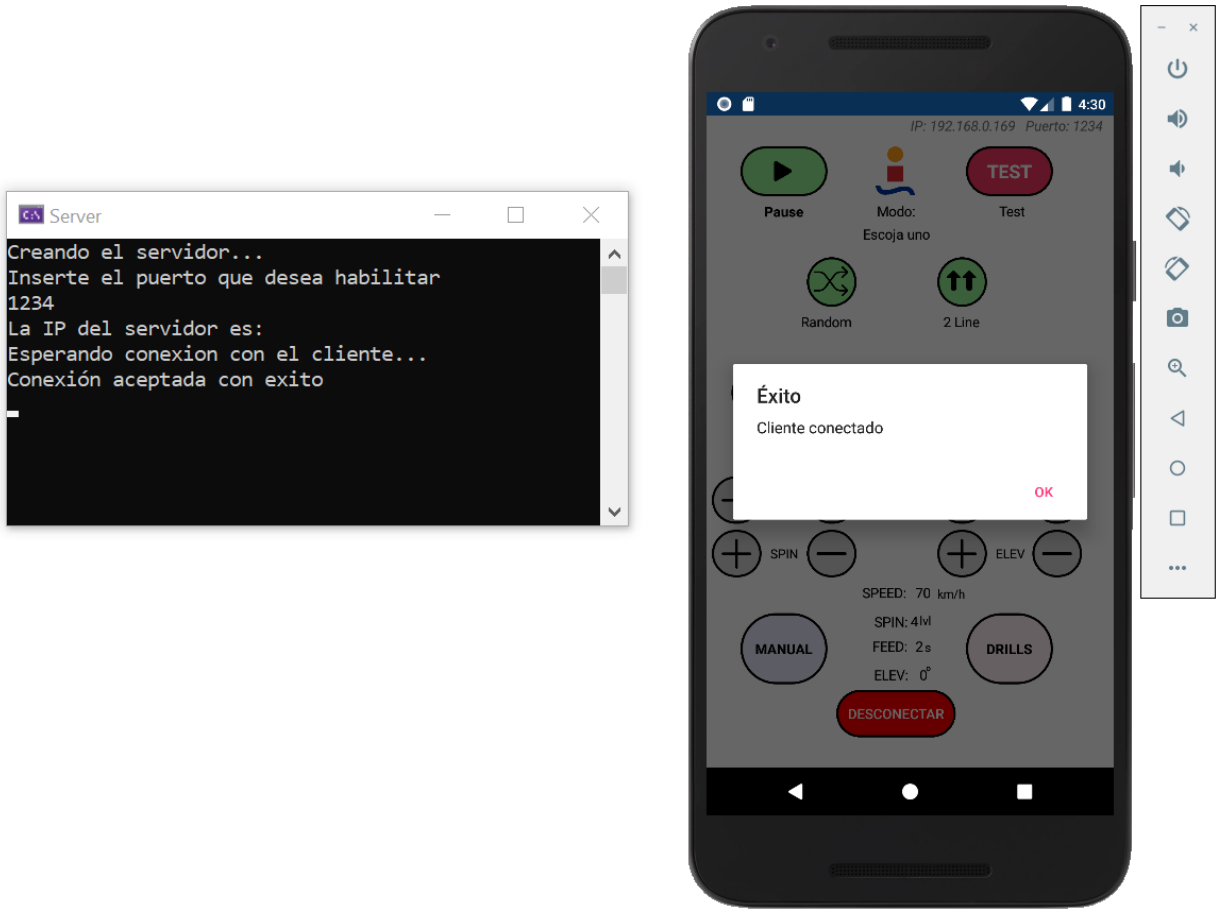

Figura 5.2: Conexión exitosa cliente-servidor

Como podemos observar en el lado del servidor, tras haber introducido la IP y el puerto correcto en la página MAINPAGE, salta un mensaje de "Conexión aceptada con éxito" en el terminal y una notificación con la cabecera de "Éxito" en el lado del cliente.

Esta conexión se debe a un doble funcionamiento, por un lado, disponemos de una función denominada *BeginConnect*, que se encarga de utilizar la IP y el puerto introducido en los *entrys* por el usuario en la interfaz de *Mainpage*, intentando así conectarse a un servidor que esté en escucha.

Por otro lado, el propio servidor que estaba en escucha en la máquina lanza pelotas, evidencia dicha solicitud de conexión por parte del cliente, llama a una función denominada *AcceptCallback* y termina aceptando la conexión para establecer un puente de comunicación entre el *socket* del cliente y el del servidor.

Una vez que tenemos el *socket* tanto en el lado del servidor como en el lado del cliente conectados, entra de nuevo en acción la parte del servidor.

Tras haber aceptado la conexión del cliente, comienza a ejecutarse en el servidor una función denominada *ReceiveCallback*.

Esta función se llama una y otra vez así misma tras haber captado un mensaje por parte del cliente, para crear un mecanismo asincrónico de llamadas y captación de mensajes desde la aplicación cliente.

El cliente no dispone de ninguna función general como ReceiveCallback, pero dispone de una función denominada *Send*, la cual, solo entra en acción cuando el cliente decide ejecutar alguna propiedad de la máquina, proporcionando así también un uso asincrónico de la aplicación.

Como el intercambio de información es dinámico y fluido, es decir, ocurre bidireccionalmente desde el lado del cliente al servidor y viceversa, dicha función *Send* está preparada para ser usada en el intercambio de mensajes de ambos códigos.

La función *Send* no es más que una función que se encarga de procesar el mensaje que queremos enviar (código ASCII), para luego llamar a otra subfunción denominada *SendCallback*, también asíncrona, que se encarga de mandar por el *socket* el mensaje ya procesado.

Para finalizar el flujo de mensajes, la parte del cliente tiene también una función denominada *Receive*, la cual entra en acción en algunos casos de intercambio de información importante al servidor. Mediante esta función se espera la recepción de un mensaje de asentimiento por parte del servidor (como respuesta a una petición del cliente). La aplicación imprime un "ok" enviado por el servidor si el mensaje ha llegado correctamente o imprime "error" si se produjera cualquier fallo durante el envío.

Además, durante la ejecución del proceso, tenemos la posibilidad de interrumpir y finalizar la comunicación entre ambos dispositivos a través de la función *Shutdown,* la cual, se encarga de cerrar el socket correspondiente en la parte del cliente y enviar un mensaje de fin de conexión al servidor de una manera ordenada.

Si optamos por recopilar en un diagrama la acción que se desarrolla dentro de la comunicación entre ambos sistemas tendríamos lo siguiente:

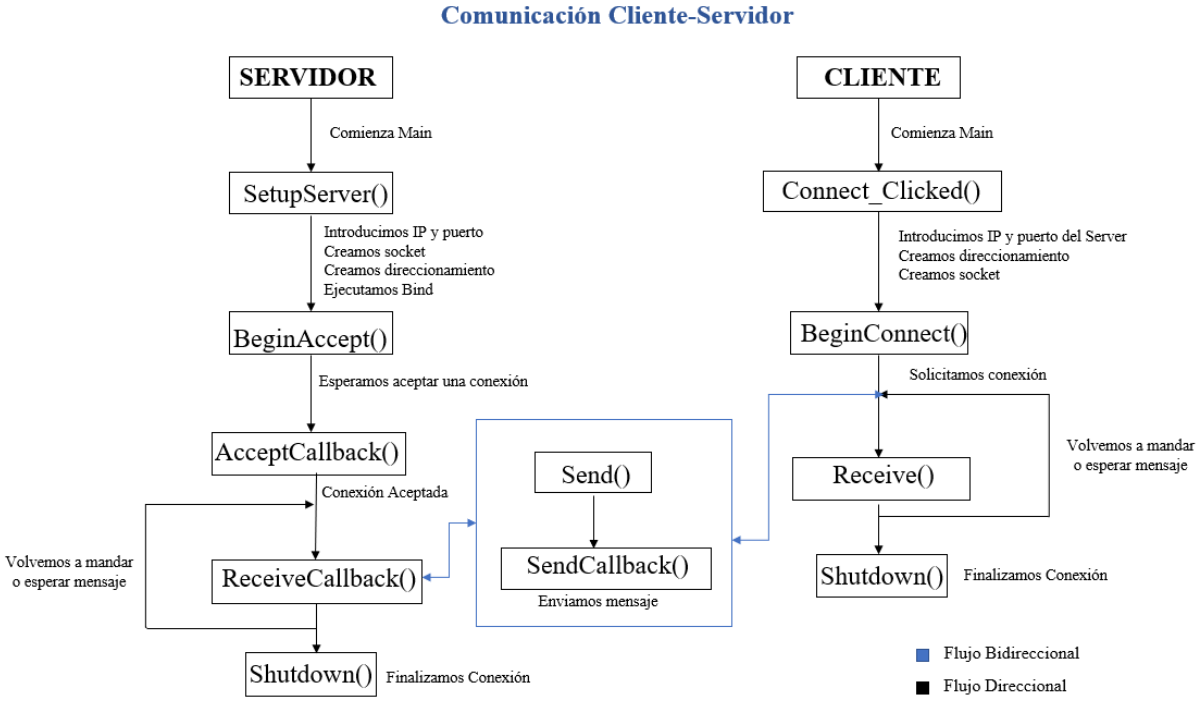

Figura 5.3: Diagrama de Fujo de comunicación

5.3. Descripción de los mensajes intercambiados

En el apartado anterior hemos descrito cómo es el proceso de comunicación entre el servidor y el cliente. En esta nueva sección se va a describir el formato, la estructura y el contenido de los mensajes que intercambian cliente y servidor utilizando como se ha comentado anteriormente el lenguaje de etiquetas *XML*.

Para ello, como se anunció al comienzo del capítulo, se ha hecho uso del lenguaje *XML*, que es un lenguaje de marcado similar a *HTML*. Sus siglas significan *Extensible Markup Language* (Lenguaje de Marcado Extensible) y es una especificación de *W3C* como lenguaje de marcado de propósito general. Esto significa que, a diferencia de otros lenguajes de marcado, XML no está predefinido, por lo que el usuario debe definir sus propias etiquetas. El propósito principal del lenguaje es compartir datos a través de diferentes sistemas, como Internet. Por tanto, cada mensaje consiste en un documento XML que contendrá la información pertinente. Se ha elegido esta forma de encapsular los mensajes porque ofrece modularidad, posibilidad de añadir nuevas funcionalidades sin cambiar las ya existetes, y también porque es un lenguaje estándar ampliamente utilizado.

El lenguaje XML posee una jerarquía de nodos (etiquetas) donde partimos de un documento padre que se puede ramificar en diferentes elementos hijos y a su vez, estos elementos se pueden diversificar en más elementos que en última instancia contienen las propiedades y atributos de los nodos.

Para crear los mensajes encargados de las peticiones enviadas desde la aplicación cliente se ha programado distintas funciones de creación de documentos *XML,* una para la interfaz *Home* denominada *XmlDocHome*, otra para la interfaz *Manual* denominada *XmlDocManual* y una última para la interfaz de *Drills* llamada *XmlDocDrills*.

Una vez creado el documento *XML* se codifica en una cadena de caracteres para poder ser enviados por el *socket* de comunicación. Una vez que el servidor recibe dicho mensaje, lo carga de nuevo en un documento nuevo *XML,* para así trabajar con el formato original y aprovechar las ventajas que esta herramienta nos ofrece.

Por un lado, la función *XmlDocHome* tiene la siguiente estructura.

```
<msg tipo=”modo”>
    <modo>Random<modo>
     <parametros>
          <speed>55</speed>
          <spin>6</spin>
          <feed>3</feed>
          <elev>30</elev>
     </parametros>
</msg>
```

Figura 5.4: mensaje *XML* desde la interfaz de Home I

Podemos observar en la figura anterior que el mensaje *XML* de la interfaz de *Home* se divide en dos hijos principales, un primer nodo denominado *modo*, en el cual se introduce la petición que se ha seleccionado, en este caso el modo de juego rápido *random* y un segundo nodo denominado *parámetros* que contiene los parámetros intrínsecos de la máquina lanza pelotas (*speed,spin, feed y elev*).

La manera de actualizar los parámetros de la máquina también se realiza desde la interfaz de *Home* como se comentó desde el inicio en el que se decribió dicha interfaz, por lo que el mensaje *XML* que

se genera respeta este mismo formato antes visto, lo que cambia es el contenido de la etiqueta *modo* como veremos a continuación para un ejemplo de una peticón de aumento de la velocidad de disparo.

```
<msg tipo=”modo”>
    <modo>mas_speed<modo>
    <parametros>
        <speed>55</speed>
        <spin>6</spin>
        <feed>3</feed>
        <elev>30</elev>
    </parametros>
</msg>
```

Figura 5.5: mensaje *XML* desde la interfaz de Home II

Observamos como el contenido de la etiqueta *nodo* posee un *mas_speed*, este dato es interpretado posteriormente por el sevidor para aumentar así la velocidad de disparo.

Por otro lado, la interfaz de *Manual* dispone de la función *XmlDocManual*, la cual presenta la siguiente estructura.

```
<msg tipo=”modo”>
    <modo>mas_speed<modo>
    <parametros>
        <pos>A4<pos>
        <rep>40<rep>
        <speed>55</speed>
        <spin>6</spin>
        <feed>3</feed>
        <elev>30</elev>
    </parametros>
</msg>
```

Figura 5.6: mensaje *XML* desde la interfaz de Manual

La estructura que presenta el documento generado desde la interfaz *Manual* es idéntica al del modo *Home* salvo porque contiene dos elementos más dentro de la etiqueta *parámetros,* uno denominado *pos* para almacenar la posición a la que queremos efectuar el lanzamiento y otro denominado *rep* para almacenar el número de repeticiones.

Por último, la interfaz *Drills* dispone de la función *XmlDocDrills*, la cual presenta la siguiente estructura.

```
<msg tipo=”modo”>
  <parametros>
    <lanzamiento>A3</ lanzamiento>
    < lanzamiento>A4</ lanzamiento>
    < lanzamiento>A5</ lanzamiento>
    < lanzamiento>A3</ lanzamiento>
    < lanzamiento>A4</ lanzamiento>
    < lanzamiento>A3</ lanzamiento>
  </parametros>
</msg>
```

Figura 5.7: mensaje *XML* desde la interfaz de Drills

El siguiente diagrama de flujo recoge el procedimiento que se ejecuta tanto por el lado del cliente como por el lado del servidor una vez que cargamos un mensaje *XML* desde el modo *Manual,* hasta que dicho mensaje es procesado por Servidor para llevar a cabo la lectura y emisión de los parámetros del mensaje.

Si optamos por recopilar en un diagrama la acción que se desarrolla dentro de la comunicación entre ambos sistemas tendríamos lo siguiente:

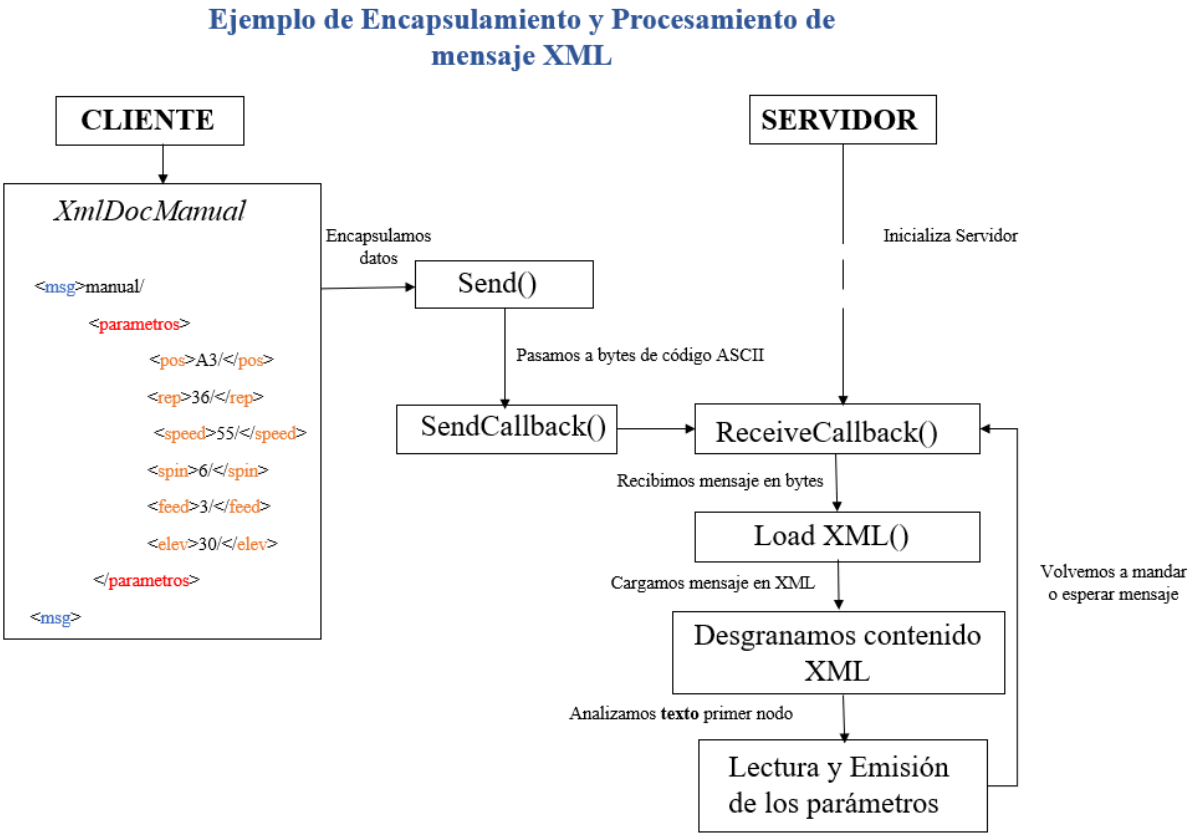

Figura 5.8: Ejemplo de Encapsulamiento y Procesamiento de mensaje *XML*

Como apunte adicional cabe destacar que el servidor posee un envío de mensaje cuyo objetivo es evidenciar al cliente de que el envío del mensaje que le acaba de llegar ha sido satisfactorio, es decir, queremos un asentimiento por parte del servidor. El esquema del mensaje es el siguiente:

```
<msg tipo=”asentimiento”>
    <parametros>
        <valor>NO OK</valor>
        < texto>Fallo en motor 1</texto>
    </parametros>
</msg>
```

Figura 5.9: Ejemplo de asentimiento por el Servidor

El documento *XML* creado desde el lado del servidor posee también dos nodos, uno cuya etiqueta se denomina *valor* que contiene un “OK” o “NO OK” si el asentimiento es satisfactorio o no y un segundo nodo que contiene la información de dicho fallo en caso de haberlo.

Tras enviar el mensaje de asentimiento, el cliente recibe dicho mensaje a través de la función *Receive*, la cual es una función síncrona que se queda esperando a recibir dicho mensaje tras enviar el mensaje principal desde el lado cliente. Una vez recibido, imprime dicho contenido enviado por el servidor en pantalla a través de una notificación.

6 GESTIÓN DE ERRORES

Una parte importante de la implementación de la aplicación que se ha diseñado, reside en detectar posibles errores que podrían surgir durante la ejecución de la aplicación cliente. La detección y gestión de posibles errores proporciona rosbustez a la aplicación.

6.1. Error de comunicación

El método por excelencia para la notificación de errores al usuario se centra en el uso de los *DisplayAlert*. Esta función se incorpora dentro de los callbacks asociados a los botones de la aplicación.

El funcionamiento de los botones, como se ha explicado en capítulo 2, se divide en un *try-catch*, donde el *try* ejecuta la función que realiza el botón propiamente dicho y en el *catch* se introduce la función *DisplayAlert*. La función del objeto *DisplayAlert* es mostrar un mensaje de advertencia al usuario. En la aplicación de control remoto de la máquina, el contenido de estos mensajes consiste en una breve descripción del aviso o error que se haya producido.

La manera que tendremos de avisar al usuario de cualquier fallo dentro de esta implementación asincrónica, utiliza la clase *await* junto al método *DisplayAlert* que se ha explicado. Por lo tanto, la manera que tendremos de trabajar a la hora de exponer posibles errores será salir del *catch* y llamar a dicha función para así mostrar el error a través de un mensaje en la app.

El fallo que cubre la mayoría de los objetos *DisplayAlert* proporcionan un mensaje de error o de aviso acerca de un posible fallo producido durante la conexión entre cliente y servidor. Todos los botones que envían comandos al servidor implementan una comprobación de que no se ha perdido la conexión con el servidor. En caso de que en el momento de enviar un mensaje al servidor se detecte que la conexión se ha interrumpido, se muestra al usuario un mensaje de error advirtiendo de esta de esta situación.

Un ejemplo de dicho uso lo podemos ver en la primera página de inicio, dentro del *Mainpage*, cuando intentamos conectarnos al servidor de la máquina lanza pelotas e introducimos datos incorrectos dentro de los *entrys* de la aplicación. Si optamos por introducir una IP o un puerto distinto al que nos demanda el servidor obtenemos lo siguiente.

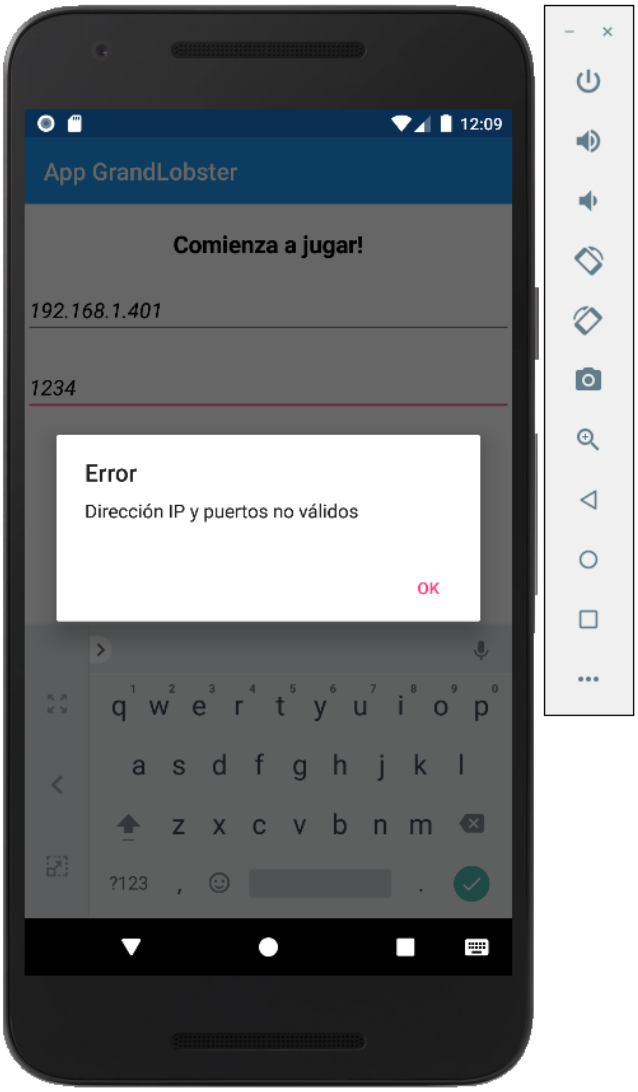

Figura 6.1: Ejemplo de error de conexión entre sockets

La figura 6.1 muestra un fallo de la conexión entre cliente y servidor, esta misma detección de fallo de comunicación saltaría en cualquiera de los botones implementados en la app si la comunicación fallara, de manera que se avisara así al usuario de que la comunicación se ha cortado.

6.2. Rango de los parámetros

Como cualquier dispositivo mecánico, la máquina lanza pelotas tiene unos rangos de valores con los que puede cambiar su funcionamiento. La detección de errores en este caso consiste en avisar al usuario de que está demandándole a la máquina unos parámetros incoherentes o no disponibles para su buen funcionamiento.

Si recordamos dichos parámetros de la máquina, tenemos la velocidad de disparo, que oscila entre 30-130 km/h, el efecto de disparo, que oscila entre 9 niveles diferentes, el intervalo de tiempo entre un disparo y otro, que oscila entre 2 y 13 segundos, y la elevación del disparo que oscila entre los 0º y 50º.

Por lo tanto, si el usuario pretende establecer uno de estos parámetros fuera de los límites anteriores,

se mostrará un aviso de que la operación que está intentando llevar a cabo no es posible.

Esta implementación es una manera de asegurarnos de que se cumplen los parámetros previstos para la máquina y no llevar al límite el funcionamiento de la misma durante su ejecución.

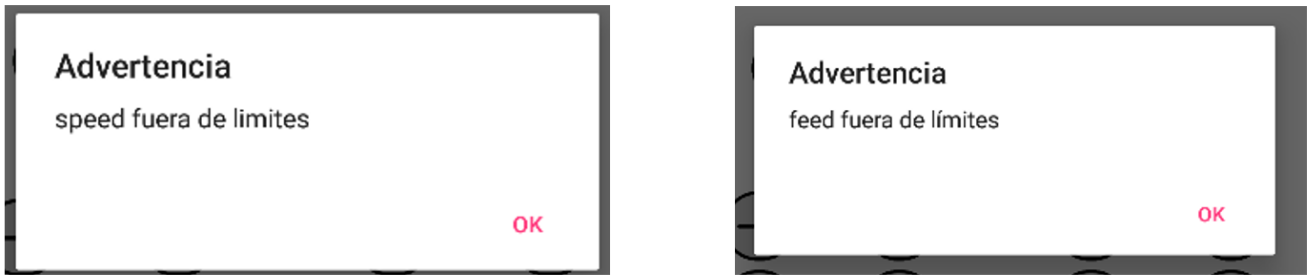

Figura 6.2: Ejemplo de aviso de fuera de límites

La imagen 6.2 muestra un ejemplo del mensaje de advertencia que se muestra si el usuario selecciona una velocidad de disparo mayor a 130 km/h recogiendo por pantalla "speed fuera de límites", al igual que si optamos por reducir el intervalo de disparo a 1", obtenemos un mensaje de "feed fuera de límites". De esta manera limitamos los parámetros intrínsecos de la máquina lanza pelotas.

6.4. Posiciones

Como se ha visto en la aplicación cliente, tenemos interfaces como la de modo *Manual* o modo *Drills* en la que el usuario es el encargado de seleccionar a qué posición concreta de la pista quiere que se realice un lanzamiento.

Esto quiere decir que es el propio usuario quien, a través de los *entrys*, introduce la posición de la pista a la que quiere que la máquina efectúe su próximo lanzamiento. Por tanto, hay que asegurarse de que esa petición de lanzamiento sea una solicitud coherente y que el valor introducido en el *enty* sea una posición válida dentro de la pista.

En el caso concreto de la interfaz *Manual* al introducir un valor incorrecto en el campo correspondiente a la posición y pulsar el botón de envío del comando, se mostrará un mensaje de advertencia indicando que la petición no se puede llevar a cabo teniéndose que elaborar una nueva petición de envío.

La siguiente figura muestra dicho caso:

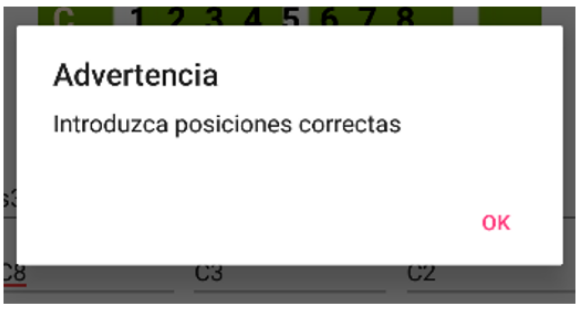

Figura 6.3: Ejemplo de posición inválida

En la interfaz de *Drills*, también se muestran estos mensajes de advertencia si se introduce información errónea en los campos de texto asociados a las distintas posiciones de la pista de tenis.

la interfaz de *Drills* como se dedica exclusivamente a registrar sets de lanzamientos para que luego puedan ser ejecutados por la máquina lanza pelotas, y por tanto posee una implementación un poco más restrictiva a la hora de poder registrar un set correctamente.

A la hora de crear un nuevo nuevo, existen de 6 campos o *entrys* donde introducir las posiciones de los lazamientos que queremos ejecutar, para poder registrar dicho set o actualizar sus valores, estos campos deben estar obligatoriamente rellenos con los caracteres de posiciones de la pista o con un guión "-" en el caso que queramos dejar un entry vacío.

De esta manera, nos aseguramos de que el mensaje que enviamos al servidor siempre contiene 6 posiciones válidas de la matriz de lanzamientos o guiones en el caso de que queramos ejecutar menos de 6 lanzamientos dentro del set creado.

Si optamos por representar dicho aviso obtenemos lo siguiente:

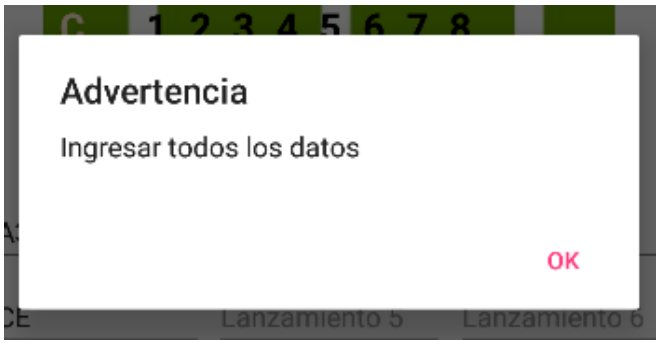

Figura 6.4: Ejemplo de ausencia de datos

La figura 6.4. nos muestra como la aplicación avisa al usuario para que introduzca todas las posiciones para poder registrar un set de lanzamientos en concreto. Este mismo método se incorpora en el botón de actualizar para sets ya creados, asegurando así que, tanto en la creación como en la actualización del set, se incorporan datos coherentes con las posiciones de la matriz que representa la pista de tenis.

6.5. Funciones auxiliares del algoritmo

Este tipo de funciones son las que se utilizan para asegurarnos de un buen comportamiento del sistema y darle robustez a nuestra aplicación. Se encargan de la gestión de espacio en memoria, cierre ordenado de la conexión y manejo de varibles globales. Estas subfunciones se ejecutan por abajo durante el proceso y permiten cerciorarnos de dicho buen comportamiento de la misma.

Por un lado, tenemos la función denominada *Limpiarcontroles*, la cual, se encarga de eliminar los datos de los *entrys* que acaban de ser utilizados para así poder machacar dichos caracteres y poder volver a sobre escribirlos sin problemas durante la creación o actualización de un set de lanzamiento en la interfaz de *Drills*.

Por otro lado, la aplicación de control remoto también dispone de otra subfunción encargada de vaciar los mensajes que acabamos enviamos al servidor, para así poder reutilizar dichas variables en otra petición y liberar espacio en memoria.

Estos mensajes *XML* generados para enviar peticiones al servidor son eliminados tras haber recibido el mensaje de asentimiento correspondiente a dicha petición.

Por último, se lleva a cabo un cierre ordenado de la conexión del *socket* cliente cuando ejecutamos el botón de desconexión. Este *socket* se queda recibiendo datos durante un breve intervalo de tiempo hasta que deja de recibir datos o evidenciar que la conexión del socket del lado del servidor también ha sido cerrada. De esta manera evitamos tener cierres violentos de desconexión entre *sockets*.

7 TRABAJO FUTURO

Este tema trata de recopilar una serie de mejoras a implementar en el trabajo en cuestión para un uso más eficiente de la aplicación móvil de control remoto. Las ideas que se muestran a continuación son unas propuestas para una ampliación futura de dicho trabajo.

7.1. Ampliación de gestión de errores

Se han propuesto dos ideas principales para la mejora de gestión de errores.

Por un lado, el servidor podría detectar la pérdida de conexión con el *socket* cliente y volver a iniciar la escucha para aceptar nuevas conexiones. Actualmente el servidor se incia únicamente una vez y tras haber completado la conexión dicho servidor muere cuando terminamos de ejecutar la aplicación móvil de control remoto. Este problema impide que podamos reconectarnos una vez cerrada la conexión. Además, el hecho de que el servidor deje de escuchar peticiones tras haber aceptado la primera conexión, limita que otros teléfonos moviles puedan conectarse simultáneamente a la máquina lanza pelotas privando el acceso desde distintos teléfonos móviles.

Por otro lado, la aplicación móvil de control remoto podría incluir *timouts* y esperas, los cuales saltarían tras un cierto tiempo o evento. Esta implementación ayudaría a la aplicación cliente a no dejarla nunca colgada esperando un mensaje de asentimiento por parte del servidor o en un punto crítico de la ejecución del programa. Normalmente estos temporizadores están asociados con la llegada de una señal que se desenmascara y hace saltar a un manejador (*handler*). Esta implementación tiene un uso muy frecuente en procesos multihilos.

7.2. Bluetooth

Este trabajo parte de la base de que disponemos conexión a internet por toda la cancha de tenis. Sin embargo, esto resulta ser una limitación si nuestra máquina estuviera destinada a ser utilizada en unas instalaciones que no cuentan con dicha conexión, traduciéndose así en pérdidas de clientes.

Por lo que una propuesta adicional para este trabajo sería implementar una conexión Bluetooth, para así resolver este problema. Esta solución es factible ya que el mecanismo cuenta con una Raspberry Pi4 que contiene un módulo Bluetooth integrado.

A diferencia de una Red de Area Local (LAN) inalámbrica, el bluetooth no necesita ninguna configuración para establecer la conexión y llevar a cabo una transferencia de archivos. Como consecuencia, este protocolo de comunicación resulta ser muy popular para la comunicación a corta distancia.

La conexión Bluetooth puede establecerse de forma sencilla sin la interferencia de dispotivos no identificados. A menos que el dispositivo esté ya emparejado con tu equipo, el protocolo Bluetooth permite aceptar o rechazar la conexión y la transferencia de archivos. Esto evita que se transfieran datos innecesarios o desconocidos. Además, tener acceso a un dispositivo Bluetooth no cuesta dinero, ya que todo lo que se necesita es que sendos dispositivos cuenten con un módulo Bluetooth.

Sin embargo, la comunicación Bluetooth cuenta con ciertas limitaciones, la distancia es uno de ellas. Una cancha de tenis cuenta con un largo de 23.77 metros, y la mayoría de los módulos Bluetooth tienen un alcance de entre 10-20 metros, por lo que habría problemas de conectividad desde el fondo de la pista.

Otra limitación es el problema de interferencias, los dispositivos Bluetooth trabajan con la banda de radio de 2,4 GHz, que es la misma frecuencia usada por muchos dispositivos inalámbricos. Por lo que, si existieran muchos dispositivos en la misma zona utilizando el mismo tramo de ancho de banda, podrían generarse problemas de red globales. Sin embargo, a pesar de estos inconvenientes, resultaría de gran interés poder implementar este protocolo entre la máquina lanza pelotas y la aplicación de control remoto para así disponer de una alternativa a la conexión Wifi.

7.3. Ampliación del modo Drills

La interfaz de Drills como se ha comentado desde el inicio de este trabajo es una novedad que pocas aplicaciones de control remoto de máquinas lanza pelotas poseen. El hecho de poder programar nuevos sets de entrenamientos para luego poder ejecutarlos en la máquina lanza pelotas es una gran ventaja. Sin embargo, la aplicación cliente presentada no posee la capacidad de poder ejecutar diferentes lanzamientos con diferentes parámetros durante una misma secuencia de lanzamientos.

Esta limitación podría ser corregida creando una interfaz más amplia que pueda englobar tanto las posiciones de los disparos a los que queremos acceder, como los parámetros de cada uno de ellos, recogiendo así, la velocidad de disparo, el efecto que queremos imprimir, el tiempo entre un disparo y otro y la elevación del lanzamiento, generando de esta manera una cohesión entre el modo *Manual* y *Drills* implementados actualmente en la aplicación cliente.

BLIBLIOGRAFÍA

[1] Charles Petzold, *Creating Mobile Apps with Xamarin.Forms*

[2] *Grand Five le owner's manual*

[3] Dan Hermes, *Building Xamarin.Forms Mobile Apps Using XAML*

[4] Patrice Clement, *Python y Raspberry Pi*

ÍNDICE DE CONCEPTOS

GLOSARIO