UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE GRADO

DESARROLLO DE UN MÓDULO DE CONEXIÓN ENTRE DISPOSITIVOS ANDROID PARA LA COMUNICACIÓN EN TIEMPO REAL EN VIDEOJUEGOS MULTIJUGADOR

Javier Abella Taboada

MAYO 2014

RESUMEN

El objetivo principal de este proyecto es dotar a un videojuego desarrollado para Android de la posibilidad de conectar a varios usuarios en un mismo lugar para que puedan jugar entre ellos en tiempo real.

Para llevarlo a cabo, se ha desarrollado un sistema completo que permite la comunicación entre dispositivos Android a través de la red. Básicamente, este sistema se puede dividir en tres partes: un cliente, un servidor y un protocolo de comunicación.

El cliente se encuentra dentro de la aplicación. Su papel principal es el de enviar las peticiones que el usuario realice al servidor y mostrar la información que el servidor envíe al dispositivo.

El servidor actúa como punto de encuentro entre los usuarios que deseen jugar una partida contra otros usuarios y responderá a todas las peticiones que envíen los clientes que estén conectados. Además, gestiona todas las partidas que se estén jugando.

El protocolo de comunicación indica cuáles son los mensajes que deben intercambiarse entre cliente y servidor para la transmisión de información a través de la red.

Google proporciona una plataforma denominada *Google Play Services* que permite que el programador de juegos multijugador se concentre exclusivamente en la parte del cliente, pues los servidores y su funcionalidad son suministrados por Google. En este proyecto se ha prescindido de los servidores de Google para, precisamente, tener la oportunidad de entender con profundidad los desafíos planteados por los juegos multijugador.

En este sentido, se ha desarrollado un sistema de predicción que mantiene sincronizados a los jugadores y al servidor durante el curso de una partida. Este sistema rebaja el número de paquetes de información que se intercambian.

Todo el sistema funciona bajo los protocolos TCP y UDP. El primero se emplea para el intercambio de información durante la navegación en la aplicación (conexión, desconexión, crear una partida, unirse a una partida…) ya que para este caso es necesario el establecimiento de la conexión con el servidor. Mientras, el segundo se emplea durante el desarrollo de las partidas dado que es necesario que la comunicación sea rápida.

PALABRAS CLAVE

Google, Android, Aplicación, Multijugador, Tiempo real, Redes, Protocolos de comunicación, Cliente, Servidor, TCP, UDP.

SUMMARY

The main objective of this project is to equip an Android videogame with the ability of connecting multiple remote users so they can play with each other in real time.

To accomplish this, a complete system that enables communication between Android devices over the network has been developed. Basically, this system can be divided into three parts: a client, a server and a communication protocol.

The client is in the application. Its main role is to send the user's requests to the server and display the information the server sends to the device.

The server acts as a meeting point for users who want to play against each others and respond to all requests sent by connected clients. It manages all the games that are being played.

The communication protocol indicates which messages should be exchanged between the client and the server for the transmission of information through the network.

Google provides a system called *Google Play Services*. This allows the mulplayer game developer focus exclusively on the client side because the servers and their functionality are provided by Google. This project has been dispensed with Google servers to have the opportunity to understand the challenges posed by the multiplayer games.

In this respect, our platform makes use of a prediction system that keeps synchronized players and server during a game. This system reduces the number of information packets that are exchanged.

The entire system operates under the TCP and UDP protocols. The first is used for the exchange of information during the navigation through the application (connection, disconnection, creation of a game, joining a game ...). This is because it's necessary to establish the connection to the server. Meanwhile, the second is used for the development of games to secure a quick communication.

KEY WORDS

Google, Android, Application, Multiplayer, Real-time, Network, Communication protocol, Client, Server, TCP, UDP.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

- ***Juegos multijugador en tiempo real**:* Se definen así a los juegos que permiten jugar a varios jugadores a la vez en la misma partida de forma remota.

- ***Cliente:*** Es una aplicación o sistema informático que consume servicios de forma remota ofrecidos por otro sistema, el servidor.

- ***Servidor:*** Sistema informático que ofrece una serie de servicios para que sean empleados por otros sistemas, los clientes.

- ***Socket:*** Es un medio de conexión de dos vías que permite la comunicación entre dos sistemas a través de la red.

- ***Protocolo de comunicación:*** Es un conjunto de normas y reglas que permiten que dos o más sistemas se comuniquen entre ellos para transmitir información.

- ***TCP (Transmission Control Protocol):*** Es un protocolo de comunicación que permite el envío de flujos de datos a través de la red.

- ***UDP (User Datagram Protocol):*** Es un protocolo de comunicación basado en el intercambio de datagramas a través de la red sin que sea necesario establecer una conexión.

- ***Paquete de información:*** Cada una de las unidades en las que se divide la información que se va a enviar a través de la red.

- ***Android:*** Sistema operativo desarrollado por Google para funcionar en dispositivos móviles o *smartphones*.

- ***Aplicación móvil:*** Es un programa informático diseñado para ser ejecutado en dispositivos móviles o s*martphones.*

- ***Actividad:*** Es un componente de una aplicación Android que proporciona una pantalla con la que el usuario puede interaccionar para llevar a cabo una tarea, como marcar un número de teléfono, enviar un mail o ver un mapa. Cada actividad muestra una pantalla en la que se dibuja la interfaz de usuario.

- ***Servicio:*** Es un componente de una aplicación Android que se emplea para realizar tareas de larga duración que no necesitan interacción con el usuario o para proporcionar funcionalidad a otras aplicaciones.

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

La revolución en el mundo de la telefonía móvil es un hecho que está cambiando el mundo de la tecnología por completo. Todos los sectores, tanto tecnológicos como no, están tratando de aprovecharse de esta revolución y no quedarse atrás.

Con este nuevo mercado, que se abrió hace ya unos años, el sector de las empresas de videojuegos ha encontrado una nueva plataforma donde poder realizar sus proyectos. Una plataforma a la que tienen acceso un número de usuarios mucho mayor que a las plataformas clásicas, como PCs y consolas, lo que amplía el mercado de forma exponencial. Es por ello que este tipo de empresas han aprovechado para diseñar juegos, o bien, adaptar sus títulos para estas nuevas plataformas.

Dentro del mundo de los videojuegos, actualmente la funcionalidad de poder jugar online en tiempo real con otros jugadores está muy demandada. Siendo estos juegos los que ocupan desde hace varios años el top de los juegos más vendidos en el mundo.

En el mundo de los dispositivos Android se pueden encontrar una alta gama de aplicaciones que permiten explotar esto mismo. Estos juegos suelen encontrarse en las webs más influyentes del panorama actual y la mayoría de ellos están en la página principal de la *Play Store* de Android.

Por ello, el proyecto pretende hacerse un hueco en ese espacio del multijugador online y tratar de entender un poco más cómo funciona el mundo de las comunicaciones a través de la red. En este proyecto, se desarrollará un módulo de conexión para la comunicación entre dispositivos Android que permitirá dotar de la opción de jugar en línea en una aplicación previamente desarrollada.

1.2. OBJETIVOS

El objetivo de este proyecto es el desarrollo de un protocolo de comunicación entre dispositivos Android para dotar de la opción de jugar en tiempo real contra otros jugadores a una aplicación previamente desarrollada. De tal forma que cuando el usuario quiera jugar una partida, la aplicación le conecte con otro jugador de forma remota y pueda jugar contra él.

Para llevarlo a cabo será necesario:

- Diseñar el protocolo de comunicación, esto incluye los diferentes mensajes que se enviarán entre cliente/servidor y la interpretación que hará cada una de las partes de los mensajes recibidos.

- Desarrollar un servidor que atenderá las peticiones de cada uno de los jugadores que se encuentren conectados al servidor. Además, gestionará el correcto funcionamiento de las partidas que se estén desarrollando.

- Desarrollar un sistema de predicción que permita mantener la sincronización entre los diferentes dispositivos que se encuentren en la partida y el servidor.

- Desarrollar un módulo de comunicación usando el protocolo TCP para el envío/recepción de información crítica tanto del cliente como del servidor.

- Desarrollar un módulo de comunicación usando el protocolo UDP para el correcto desarrollo de cada una de las partidas.

- Desarrollar una base de datos que permita almacenar información relevante del usuario, como su nombre de usuario y contraseña.

- Desarrollar una aplicación que sirva como ejemplo del correcto funcionamiento del protocolo.

Además de todo esto, el desarrollo de la partida tiene que ser lo suficientemente fluido y que no haya demasiados retrasos entre el servidor y el cliente y no se produzcan saltos inesperados en las posiciones de los personajes en la pantalla. Para ello, el diseño del protocolo es muy importante tratando de equilibrar la tasa de envío de mensajes de tal forma que la comunicación solo se produzca en momentos claves de la partida.

1.3. PUNTO DE PARTIDA DEL PROYECTO

Este proyecto nace con la idea de dotar a un videojuego desarrollado para dispositivos Android de la posibilidad de poder jugar en modo multijugador online en tiempo real.

Por ello, en las fases previas al desarrollo del módulo de conexión propiamente dicho, se desarrolló una aplicación basada en enfrentar a dos jugadores en una misma pantalla. El objetivo del juego es derribar al oponente haciéndole caer de las plataformas presentes en la pantalla o agotando su vida golpeándole.

Para ello, se desarrollaron dos módulos:

- El módulo que contiene toda la lógica de la aplicación y el controlador del juego
- El módulo gráfico se hizo a través de la API *BlueTreeReality* desarrollada por Jorge Femenía del Rey en su trabajo: *Desarrollo de un motor gráfico sobre OpenGL para desarrollo de videojuegos 2D en dispositivos Android.* Esta API permite implementar todo el aspecto gráfico del videojuego de manera sencilla haciendo uso del motor gráfico OpenGL.

En el apartado *3.4.Diseño del videojuego* se puede encontrar una explicación más detallada sobre cómo está diseñado el juego y cómo funciona.

Una vez completados ambos módulos se comenzó a implementar el módulo de conexión que se explica en este documento.

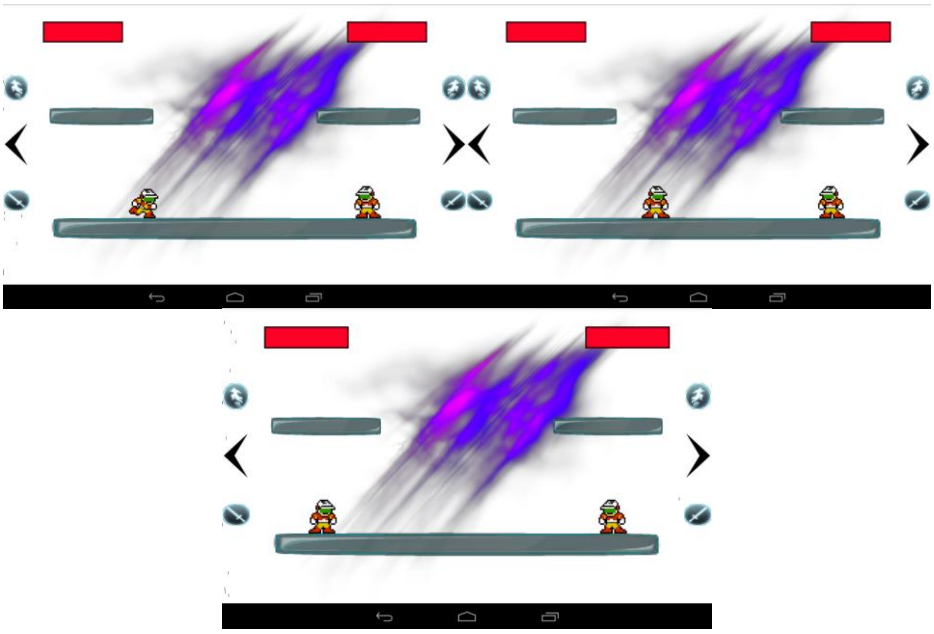

FIGURA 1: IMÁGENES EXTRAÍDAS DEL VIDEOJUEGO

1.4. ORGANIZACIÓN DE LA MEMORIA

Este documento se encuentra dividido en cinco partes:

- **Parte 1:** En esta parte, se da un enfoque genérico de qué es lo que se pretende con este proyecto y porqué se ha decido realizarlo. En segundo lugar, se explican cuáles son los objetivos que se pretenden alcanzar. Y, en tercer lugar, se explica de dónde parte este proyecto.

- **Parte 2:** En esta parte, se explica cuáles han sido las tecnologías que se han empleado para este proyecto y porqué se han elegido. Además, se da un enfoque de cómo se encuentra el mercado en el que se encontraría este proyecto.

- **Parte 3:** Esta parte explica cómo es el diseño tanto del servidor como del cliente. En segundo lugar, se muestran los mensajes que se emplean tanto para el protocolo TCP como para el de UDP, además de dar una explicación de cómo funcionan ambos. En tercer lugar, se explica cómo se mantiene la sincronización de los clientes con el servidor mediante el denominado *Sistema de predicción*. Y, por último, se muestra el diseño de la aplicación y su funcionamiento.

- **Parte 4:** En esta parte, se muestra el funcionamiento completo del sistema, cómo interaccionan cliente y servidor, cómo responden tanto el cliente como el servidor ante los mensajes recibidos…

- **Parte 5:** En esta última parte, se hace un resumen sobre lo que se ha realizado en este proyecto y cuál es trabajo a realizar en el futuro.

2. ESTADO DEL ARTE

2.1. SISTEMA OPERATIVO ANDROID

Android es un sistema operativo basado en Linux diseñado para ser usado en dispositivos móviles. Este sistema permite el uso de gráficos tanto 2D como 3D usando una variante para sistemas empotrados de la librería OpenGL llamada OpenGL ES.

Es el sistema operativo para dispositivos móviles más vendido del mundo, superando con creces a su gran competidor iOS. Tal y como se puede observar en el siguiente gráfico extraído del informe de IDC que muestra la cuota de mercado de los grandes fabricantes de smartphones y tablets durante el tercer trimestre del año 2013:

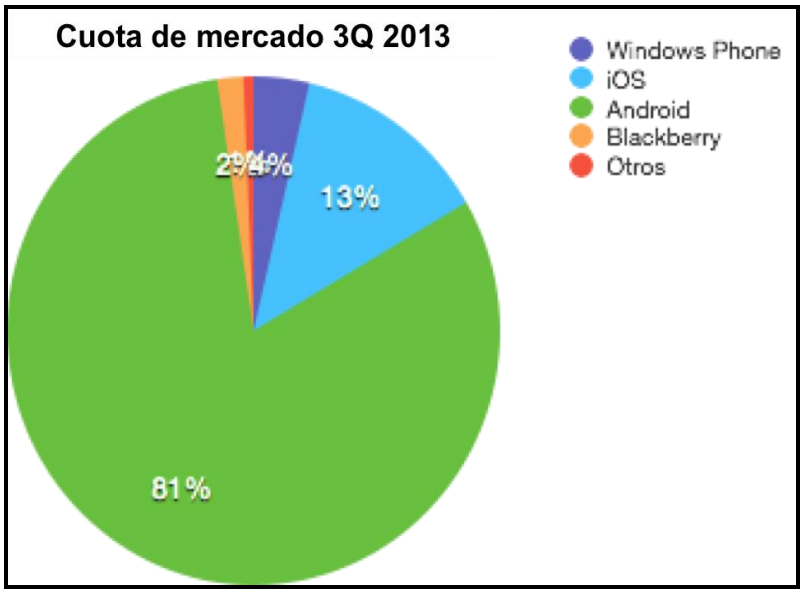

FIGURA 2: GRAFICO CUOTA DE MERCADO DE SMARTPHONES 3Q 2013

Se ha optado por emplear este sistema, dadas las facilidades que ofrece Google para desarrollar en Android. Ofrece un kit de desarrollo o SDK completo y gratuito, una completa documentación que puede ser consultada fácilmente y sus foros de consulta son muy útiles e interesantes. Además, el lenguaje que se emplea es bien conocido por todo el mundo como es Java. También se emplean otros lenguajes como XML.

2.1.2. GOOGLE PLAY SERVICES

Google permite dotar a un juego desarrollado en Android de la posibilidad de jugar online con otros jugadores mediante la plataforma *Google Play Services.*

Empleando este sistema, el desarrollador sólo tiene que encargarse de la parte del cliente ya que la parte del servidor ya está desarrollada por Google. Para poder tener acceso a esta posibilidad el desarrollador tiene que estar registrado en el sistema que tiene Google como desarrollador de aplicaciones Android y haber registrado ya el juego pagando la cuota de 25 USD (Mayo 2014). La siguiente figura muestra lo que ofrece Google y lo que tiene que implementar el desarrollador:

FIGURA 3: SERVICIOS OFRECIDOS POR GOOGLE PLAY SERVICES

La funcionalidad de estos servicios se obtiene a través de la API *GoogleApiClient.* Las funcionalidades que ofrece esta API, así como la relación que existe entre ellas se pueden ver en la siguiente figura:

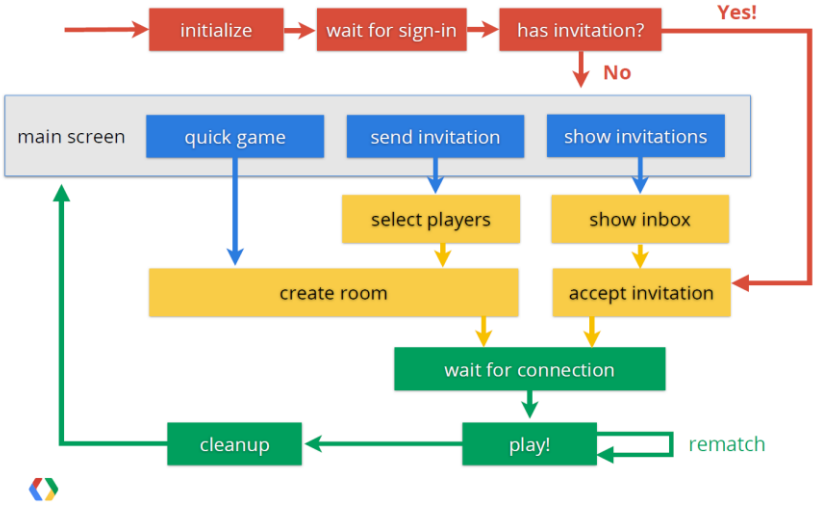

FIGURA 4: FUNCIONALIDAD DE GOOGLEAPICLIENT

El objetivo de este proyecto es desarrollar un sistema similar y entender en profundidad cómo funcionan estos sistemas y las dificultades que entrañan. Por ello, parte de la idea de este proyecto está basada en este sistema.

2.2. ARQUITECTURAS DE RED

Básicamente se han considerado dos tipos de arquitecturas de red: cliente/servidor y P2P.

La arquitectura cliente/servidor responde al siguiente esquema:

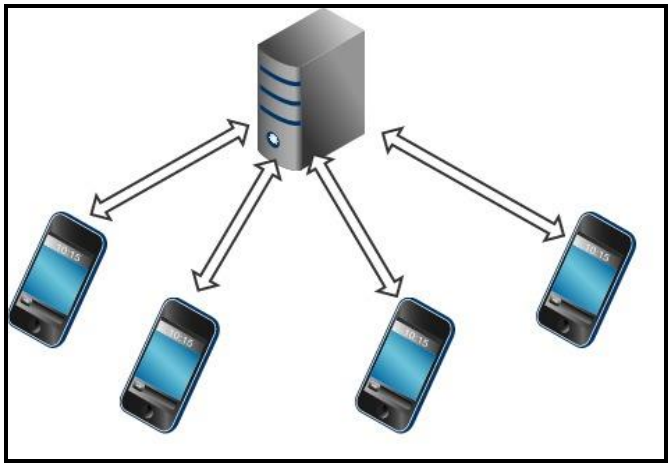

FIGURA 5: ARQUITECTURA CLIENTE/SERVIDOR

En esta arquitectura hay dos elementos principales. Por un lado, está el servidor que es el encargado de satisfacer las peticiones que realiza el otro elemento principal, el cliente. El servidor debe ser capaz de atender varias peticiones al mismo tiempo, ya que varios clientes estarán realizándolas.

En el servidor se centralizan lo diferentes recursos y aplicaciones con que se cuenta y los pone a disposición de los clientes cuando éstos lo solicitan. Esto quiere decir que en el servidor se concentran todas las gestiones.

Como ventajas de esta arquitectura se tienen: centralización del control, escalabilidad, fácil mantenimiento, seguridad. Como desventajas se tienen: la congestión del tráfico puede colapsar el servidor y hacer caer todo el sistema y la necesidad de un software y hardware potentes.

Por otro lado, se tiene la arquitectura P2P (Peer to Peer) que responde al siguiente esquema:

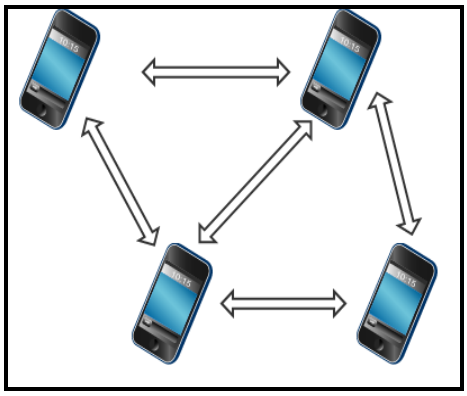

FIGURA 6: ARQUITECTURA P2P

En este tipo de arquitecturas desaparecen las figuras del cliente y del servidor, estando ahora formada por una serie de nodos que actúan de igual a igual, es decir, actúan como clientes y servidor respecto a los demás nodos de la red. Estas redes permiten el intercambio directo de información entre dispositivos.

Las redes *peer-to-peer* aprovechan, administran y optimizan el uso del ancho de banda de los demás usuarios de la red por medio de la conectividad entre los mismos, y obtienen así más rendimiento en las conexiones y transferencias que con algunos métodos centralizados convencionales, donde un número relativamente pequeño de servidores provee el total del ancho de banda y recursos compartidos para un servicio o aplicación.

Para este proyecto se ha optado por la arquitectura cliente/servidor ya que es totalmente necesario tener un servidor porque debe existir una figura que actúe como administrador de todas la partidas que se estén llevando a cabo, actuando como árbitro y asegurándose de que se está ejecutando la partida de manera correcta y que ambos jugadores están recibiendo la misma información sobre la misma. Además, también es necesaria la existencia de éste para poder almacenar toda la información de cada uno de los clientes y atender cada una de las peticiones que soliciten.

2.3. PROTOCOLOS DE COMUNICACIÓN

2.3.1. TCP

El protocolo TCP es un protocolo de comunicación a través de la red realizando conexiones entre dispositivos que permite el envío de flujos de datos. Este protocolo garantiza que la información transmitida será entregada en el destino, sin errores y en el mismo orden en que se enviaron.

Está pensado para poder enviar grandes cantidades de información de forma fiable, liberando al programador de la dificultad de gestionar la fiabilidad de la conexión (retransmisiones, pérdida de paquetes, orden en el que llegan los paquetes, duplicados de paquetes...) que gestiona el propio protocolo. Pero la complejidad de la gestión de la fiabilidad tiene un coste en eficiencia, ya que para llevar a cabo las gestiones anteriores se tiene que añadir bastante información a los paquetes enviados.

Las características principales de este protocolo son:

- *Orientado a conexión*: Antes de comenzar el envío de información los dos extremos deben pactar el establecimiento de una conexión.
- *Full Duplex*: Cada extremo tiene dos enlaces lógicos, uno de salida y otro de entrada.
- *Fiable*: Se garantiza la entrega de todos los paquetes de información, en orden y sin duplicidades.
- *Flujo de bytes*: Para TCP los datos que se envían por los enlaces lógicos de entrada y salida se consideran un flujo continuo de bytes.
- *Control de flujo en ambos extremos*.
- *Segmentación de datos de aplicación*.
- *Transmisión uno a uno.*

Este protocolo será empleado para el envío de información básica para la conexión del cliente con el servidor. Se usará para los siguientes casos:

- *Conexión del cliente*: envío del *username* y *password* al servidor, envío de la información básica del jugador almacenada en el servidor.
- *Navegación*: solicitud de nueva partida, solicitud de unirse a una partida en juego, salir de la aplicación.
- *Información del estado del servidor*: número de jugadores, número de partidas en juego.
- *Finalización de partida*: puntuación obtenida por el jugador.

2.3.2. UDP

El protocolo UDP es un protocolo de comunicación del nivel de transporte basado en el intercambio de datagramas. Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incluye la suficiente información de direccionamiento en su cabecera. Tampoco tiene confirmación ni control de flujo, por lo que los paquetes pueden adelantarse unos a otros y tampoco hay confirmación de llegada de información.

Este protocolo se emplea cuando la cantidad de información de cada paquete es relativamente pequeña pero se necesita una gran velocidad de transmisión aunque se puedan perder algunos paquetes de información, cosa que TCP no garantiza por completo.

Las características básicas de este protocolo son:

- *No está orientado a conexión.*
- *No es fiable.*
- *No hay control de flujo.*
- *Se garantiza velocidad de transmisión.*

Este protocolo se usará en las partidas que se estén jugando. Para ello, se enviarán al servidor todos los movimientos y gestos que esté realizando el jugador, el servidor tratará la información recibida y enviará esos movimientos al otro jugador. Es necesario emplear este protocolo ya que es necesario que los movimientos se envíen lo más rápido posible dada la gran cantidad de información que se puede generar en una partida. Además, evita el control de flujo que ralentizaría el juego y reduciría su fluidez.

2.4. VIDEOJUEGOS EN ANDROID

Un videojuego multijugador es aquel que permite jugar a varios jugadores de manera simultánea a través de la red. Actualmente, los juegos que más triunfan son los que están desarrollados para plataformas de consola/PC. Se encuentran títulos como:

- Sagas de Battlefield o Call of Duty: Ejemplos de videojuegos FPS que permiten jugar en línea que al fin y al cabo es uno de sus mayores atractivo

FIGURA 7: IMAGEN EXTRAIDA DE BATTLEFIELD 4

- LOL, la saga de World of Warcraft: Son ejemplos de juegos MMO en los que pueden participar cientos de jugadores online al mismo tiempo en la misma partida

FIGURA 8: IMAGEN WORLD OF WARCRAFT

Dentro del mundo de los dispositivos móviles, se puede encontrar una buena oferta de juegos que permiten jugar online contra otros jugadores. Se pueden encontrar muchas categorías de juegos, entre ellas:

- *Deportes*: NBA 2K13, Virtua Tennis o Let's Golf 3. Estos juegos permiten jugar tanto por la misma WiFi, usando datos o bien por Bluetooth.

FIGURA 9: IMAGEN VIRTUA TENNIS 3

- *Bélicos*: Call of Duty Blackops Zombies, N.O.V.A. 3 o Critical Strike. Estos juegos se pueden jugar online bajo la misma WiFi y el último juego también permite hacerlo por Bluetooth.

FIGURA 10: IMAGEN CRITICAL STRIKE

- *Clásicos*: Metal Slug (todas sus versiones) permite jugar a dos jugadores mediante Bluetooth. Worms 2 que deja jugar bajo la misma LAN. O Sonic 4 Episode II que permite jugar de ambas formas:

FIGURA 11: IMAGEN SONIC 4 EPISODE II

- *Coches*: como Asphalt 7 o Mini Motor Racing,
- *Otros*: juegos de mesa, billar, juegos de cartas….

3. DISEÑO Y DESARROLLO

3.1. ARQUITECTURA CLIENTE/SERVIDOR

Para este proyecto se ha empleado la arquitectura cliente/servidor. Esta arquitectura simplifica mucho el diseño de la aplicación ya que sólo es necesario el desarrollo de dos partes, el cliente y el servidor.

Además, el sistema se encuentra totalmente centralizado en el control de los recursos, datos y accesos. El mantenimiento es más simple ya que una actualización se aplica al servidor y de manera automática los clientes obtienen los beneficios de ésta. Y, por último, el servidor centraliza gran parte de la lógica de las partidas, lo que suma en seguridad ya que se tiene un control absoluto de lo que ocurre en cada una de ellas.

3.1.1. DISEÑO DEL SERVIDOR

El servidor se encarga básicamente tanto de atender las peticiones básicas del usuario (conexión, desconexión, intercambio de información básica...) como de la parte que concierne a la lógica del juego. De esta forma, si el usuario modifica el estado normal de la partida, el servidor detecta que la información recibida no es la correcta y puede actuar en consecuencia.

La información que recibe el servidor se engloba en dos grupos fundamentales:

- Por un lado, atiende peticiones generales del cliente: nuevas conexiones, altas de usuarios, nuevas partidas, unirse a partidas....
- Por otro lado, gestiona toda la información referente a lo que ocurre en cada una de las partidas que se estén jugando.

El primer grupo funciona bajo el protocolo TCP (*3.2.1 Protocolo TCP*), mientras que el segundo grupo de información emplea el protocolo UDP (*3.2.2. Protocolo UDP*).

El servidor está compuesto por dos partes fundamentales:

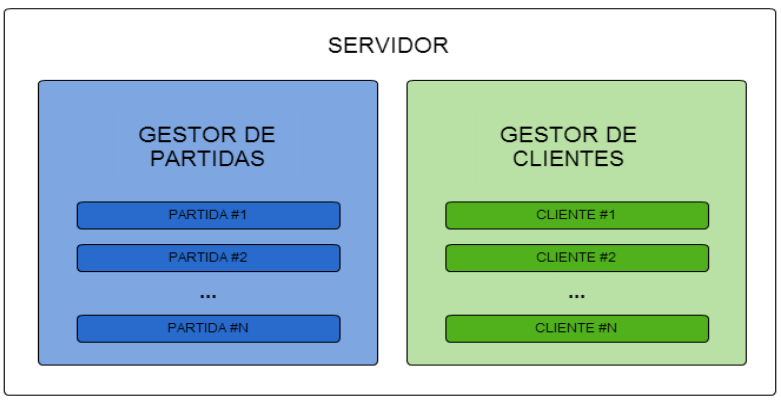

FIGURA 12: ARQUITECTURA DEL SERVIDOR

El gestor de clientes se encarga de almacenar y mantener cada uno de los clientes que entran en la aplicación. Dentro de él se encuentran cada uno de los hilos correspondientes a cada cliente que está conectado.

Por su parte, el gestor de partidas es el encargado de mantener las partidas que se estén jugando. Como en el caso anterior, el gestor de partidas contiene cada uno de los hilos correspondientes a cada partida. Dado que el protocolo UDP no es orientado a conexión, el gestor debe ser el encargado de distribuir la información que le llega a la partida a la que corresponde la información recibida.

El diagrama de clases correspondiente a la arquitectura del servidor es el siguiente:

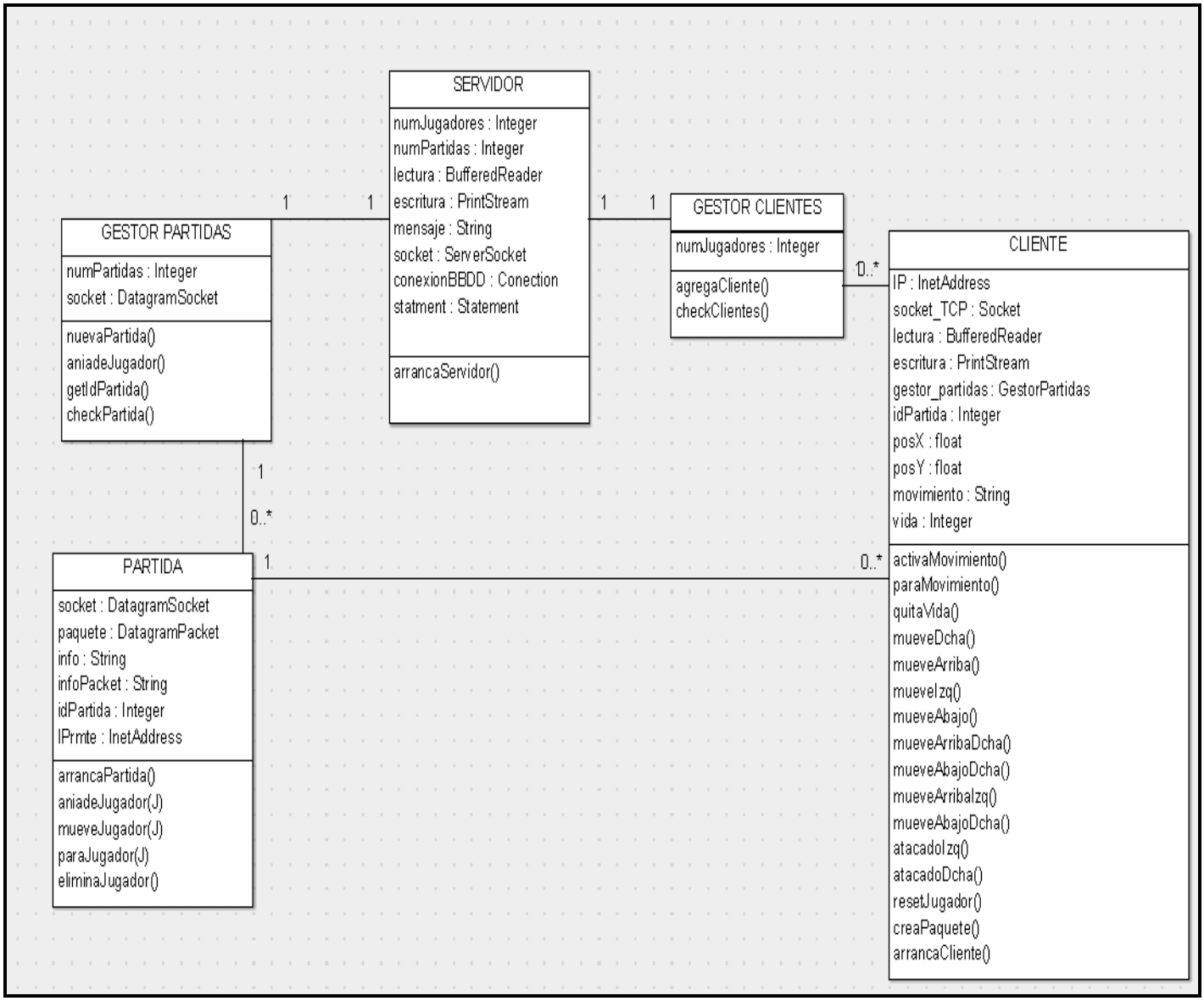

FIGURA 13: DIAGRAMA DE CLASES SERVIDOR

Dentro de la clase servidor se encuentran los siguientes atributos:

- `numJugadores`: almacena el número de jugadores que hay conectados.
- `numPartidas`: almacena el número de partidas que hay activas.
- `socket`: el serverSocket que crea el servidor para aceptar las conexiones de los clientes.
- `lectura`: es el canal por el que el servidor lee la información que le llega a través del `ServerSocket`.
- `escritura`: es el canal por el que el servidor escribe la información que necesita el cliente.
- `mensaje`: string en el que se almacena el mensaje que se envía por el canal de escritura y donde se almacena el mensaje que llega por el canal de lectura.
- `conexionBBDD`: permite la conexión con la base de datos que almacena la información de los clientes.
- `statment`: permite realizar consultas sobre la base de datos.

El método arrancaServidor inicia toda la configuración del servidor: crea los gestores de clientes y de partidas, crea la conexión con la base de datos, crea el socket del servidor a través de las siguientes sentencias:

```
socket = new ServerSocket(PORT);

gestorPartidas = new Gestor_partidas();
gestorClientes = new Gestor_clientes();

Class.forName("com.mysql.jdbc.Driver");
Connection conexion =
      DriverManager.getConnection("jdbc:mysql://localhost/tfg_game",
username, password);

Statement statment = (Statement) conexion.createStatement();
```

Por último, se mantiene a la espera de que algún cliente solicite conectarse:

```
while (true) {
      System.out.println("Esperando nueva conexion....");
      socket_cliente = socket.accept();
      //RESTO DE CODIGO//
}
```

Los siguientes apartados detallan el funcionamiento del gestor de clientes y el de partidas.

3.1.1.1. GESTOR DE CLIENTES

La función principal de este módulo es la añadir nuevos clientes y eliminar clientes que se hayan desconectado.

Para añadir nuevos clientes, el servidor se encuentra en un bucle infinito escuchando para atender nuevas conexiones. Cuando llega una nueva petición de conexión, el servidor crea un nuevo hilo con el objeto `Cliente` en que almacena la información más relevante del cliente y lo añade al gestor de clientes que lo almacenará y comprobará el estado de la conexión cada 1000 ms. Si cuando se realiza esa comprobación el hilo ya no está ejecutándose, se elimina ese hilo del gestor de clientes.

Dentro de este gestor se encuentran cada uno de los clientes que hay conectados al servidor. Sus estructuras son:

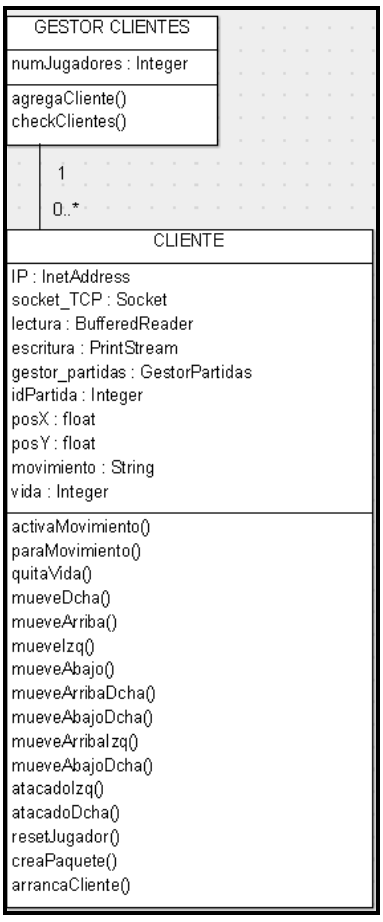

FIGURA 14: DIAGRAMA DEL GESTOR CLIENTES

El gestor contiene los siguientes métodos:

- `agregaCliente`*:* añade un nuevo cliente a su lista de clientes.
- `checkCliente`*:* comprueba qué clientes siguen activos. En caso de que alguno no lo esté, se elimina del array de clientes.

En el objeto Cliente se definen los siguientes métodos:

- `activaMovimiento`: activa el *`Timer`* que modifica las coordenadas de las posiciones.
- `paraMovimiento`: detiene el *`Timer`* que modifica las coordenadas de las posiciones.
- `quitaVida`: resta vida al jugador.
- `mueveDcha`: incrementa la coordenada X del jugador.*
- `mueveIzq`: disminuye la coordenada X del jugador.*
- `mueveAbajo`: disminuye la coordenada Y del jugador.*
- `mueveAbajo`: disminuye la coordenada Y del jugador.*
- `mueveArribaDcha`: incrementa X e Y del jugador.*
- `mueveAbajoDcha`: incrementa X y disminuye Y del jugador.*
- `mueveArribaIzq`: disminuye X y aumenta Y del jugador*
- `mueveAbajoIzq`: disminuye X e Y del jugador*
- `atacadoIzq`: mueve al jugador hacia la derecha 0.5f y disminuye su vida.
- `atacadoDcha`: mueve al jugador hacia la izquierda 0.5f y disminuye su vida.
- `resetJugador`: resetea la vida del jugador.
- `creaPaquete`: crea el `DatagramPacket` con la información que tiene que recibir el cliente incluyendo la IP del jugador y el puerto de escucha
- `arrancaCliente`: se mantiene a la espera de atender las peticiones del cliente.

*Todos los aumentos y disminuciones son de 0.05f.

Este objeto es el que atiende las peticiones del cliente, las gestiona y envía una respuesta al cliente. Para ello, el cliente se encuentra ejecutándose en un hilo a parte en un bucle infinito que seguirá ejecutándose hasta que se reciba una petición de desconexión, momento en el que se saldrá del bucle y se eliminará de la lista de clientes del gestor. Todo este intercambio de información se realiza mediante sockets usando el protocolo TCP, tal y como está explicado en el apartado *3.2. Protocolo de comunicación*.

En este objeto también se trata el estado del jugador dentro de la partida. Cuando un jugador está jugando, el servidor debe mantener continuamente el estado de éste. Para ello, cada una de las acciones que lleva a cabo el usuario las recibe el servidor, las trata el gestor de partidas y éste envía la información al objeto cliente correspondiente al jugador, actualizando su información. Está información hace que se mantenga actualizada la posición del jugador en pantalla y su estado de vida.

3.1.1.2. GESTOR DE PARTIDAS

El gestor de partidas es el módulo encargado de mantener las partidas que haya activas en el servidor. Básicamente, controla si las partidas que hay almacenadas en él están o no activas. Si no lo están las elimina del servidor dejando espacio para una nueva partida.

Si llega una petición de una nueva partida, el gestor de partidas se encarga de crear un nuevo hilo Partida, añadirle un nuevo jugador y añadirlo al array de partidas. En el caso de que se reciba una petición para unirse a una partida ya creada, el gestor añade ese nuevo jugador a la partida e informa a ambos jugadores de que la partida va a empezar. Este gestor tiene la siguiente estructura:

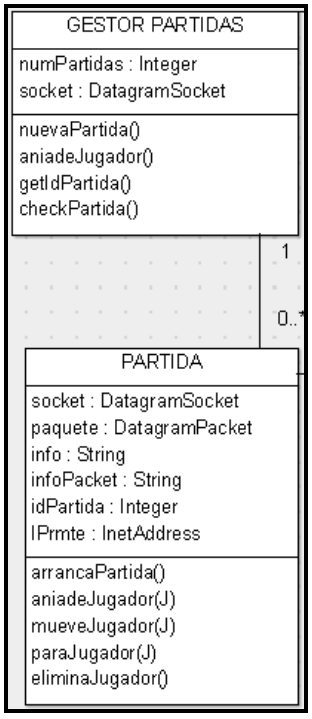

FIGURA 15: DIAGRAMA DEL GESTOR PARTIDAS

Dentro del gestor de partidas se encuentran los siguientes métodos:

- `nuevaPartida`*:* crea una nueva partida.
- `aniadeJugador`*:* añade un nuevo jugador a una partida en curso.
- `getIdPartida`*:* devuelve el ID de la partida solicitada.
- `checkPartidas`*:* comprueba que cada una de las partidas están activas.

Los métodos de `Partida` son los siguientes:

- `arrancaPartida`*:* inicia una nueva partida. Indica a los jugadores que la partida ha arrancado e inicia el bucle de escucha esperando información de los clientes.
- `aniadeJugador`*:* añade un nuevo jugador a la partida

- `mueveJugador`*:* indica al objeto Cliente correspondiente al jugador el tipo de movimiento que se está realizando.
- `paraJugador`*:* indica al objeto Cliente correspondiente que detenga el movimiento del jugador.
- `elminaJugador`*:* elimina un jugador de la partida.

Este objeto es el encargado de gestionar la partida que dos clientes estén jugando en ese momento. Para ello, este objeto se encuentra en un bucle infinito que está escuchando toda la información que los clientes están enviando al servidor.

Cuando esta información llega al servidor, éste se encarga de enviarla a los demás clientes de forma que el jugador oponente pueda ejecutar los movimientos que el otro jugador ha realizado. A su vez, el objeto `Partida` trata esa información para mantener el estado de la partida en el servidor. De esta forma, el servidor actúa como árbitro en la partida, de tal manera que no se pueda alterar de forma intencionada el estado de la misma y no pueda tener ninguna ventaja un jugador.

Cuando un jugador desea abandonar la partida, se recibe una petición para salir de la partida y el servidor elimina a los dos jugadores de la partida, indicando al otro jugador que su oponente ha abandonado y, por tanto, se acaba la partida.

La comunicación cliente/servidor se lleva a cabo mediante sockets usando el protocolo UDP, tal y como está explicado en el apartado *3.2. Protocolo de comunicación*.

3.1.2. DISEÑO DEL CLIENTE

El módulo del cliente se encuentra en la aplicación dentro del dispositivo móvil y se encarga de mantener la comunicación con el servidor. Para ello, el módulo está compuesto por los siguientes objetos:

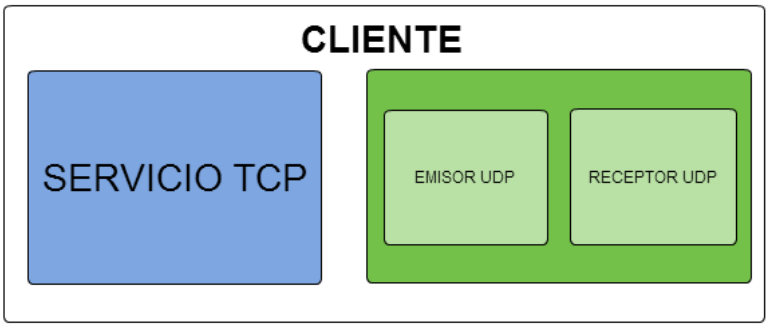

FIGURA 16: ARQUITECTURA DEL CLIENTE

3.1.2.1. SERVICIO TCP

Este servicio se encarga de establecer y mantener la comunicación con el servidor mediante el uso de un socket y bajo el protocolo TCP. Se hace uso de éste cuando el cliente solicita conectarse al servidor, ingresar en una partida, iniciar una nueva partida… (Ver apartado 3.2 Protocolo de comunicación).

La comunicación TCP se lleva a cabo mediante la clase `Service` de Android que permite realizar tareas en segundo plano mientras se esté ejecutando la aplicación. También se podría hacer mediante tareas asíncronas de Android, pero éstas no permiten la ejecución continua en segundo plano durante la ejecución de la aplicación ya que las tareas asíncronas ligadas al ciclo de vida de la actividad que las invoca.

Para poder hacer uso de este servicio, cuando el usuario pulsa el botón de entrar, la aplicación lanza el servicio:

```
servicio = new Intent(MainActivity.this, ServiceTCP.class);
startService(servicio);
```

Una vez hecho esto, el servicio crea un socket de conexión con el servidor usando la siguiente sentencia:

```
socket_TCP = new Socket(IP_SERVER, PORT);
```

A continuación, envía toda la información solicitada por el servidor a través de los canales de lectura y escritura, de manera similar a cómo lo hace el servidor.

Las actividades que necesiten comunicarse con el servidor hacen un `bind` al servicio cuando arrancan y un `unbind` cuando finalizan. De esta forma pueden usar toda la funcionalidad incluida en el servicio.

```
servicio = new Intent(this, ServiceTCP.class);
bindService(servicio, mConnection, Context.BIND_AUTO_CREATE);

unbindService(mConnection);
```

`mConnection` es de tipo ServiceConnection y se encarga de mantener la conexión al servicio.

El servicio contiene la siguiente estructura:

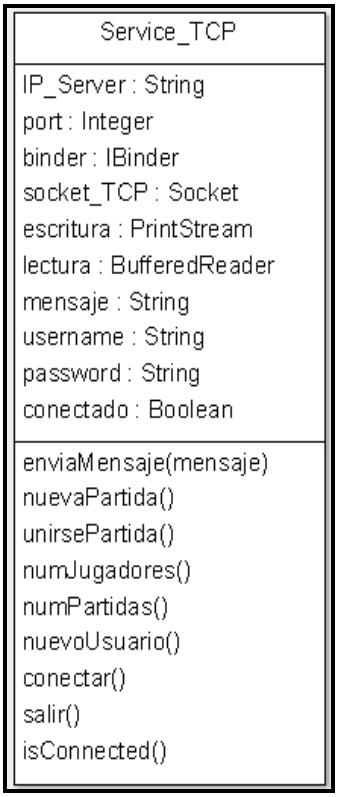

FIGURA 17: DIAGRAMA DEL SERVICIO TCP

Los métodos de `Service_TCP` son los siguientes:

- `isConnected`: Comprueba si el socket está conectado al servidor.
- `enviaMensaje`: Envía los diferentes mensajes al servidor.
- `numJugadores`: Envía la solicitud para pedir el número de jugadores online.
- `numPartdas`: Envía la solicitud para pedir el número de partidas en curso y sus ids.
- `nuevaPartida`: Solicita el inicio de una nueva partida.
- `unirsePartida`: Solicita unirse a una partida en curso.
- `salir`: Avisa al servidor de que va a salir de la aplicación. El servidor le elimina de su lista de clientes.
- `conectar`: Establece la conexión con el servidor a través del socket de conexión.

En la siguiente imagen se puede ver el ciclo de vida de un servicio en Android:

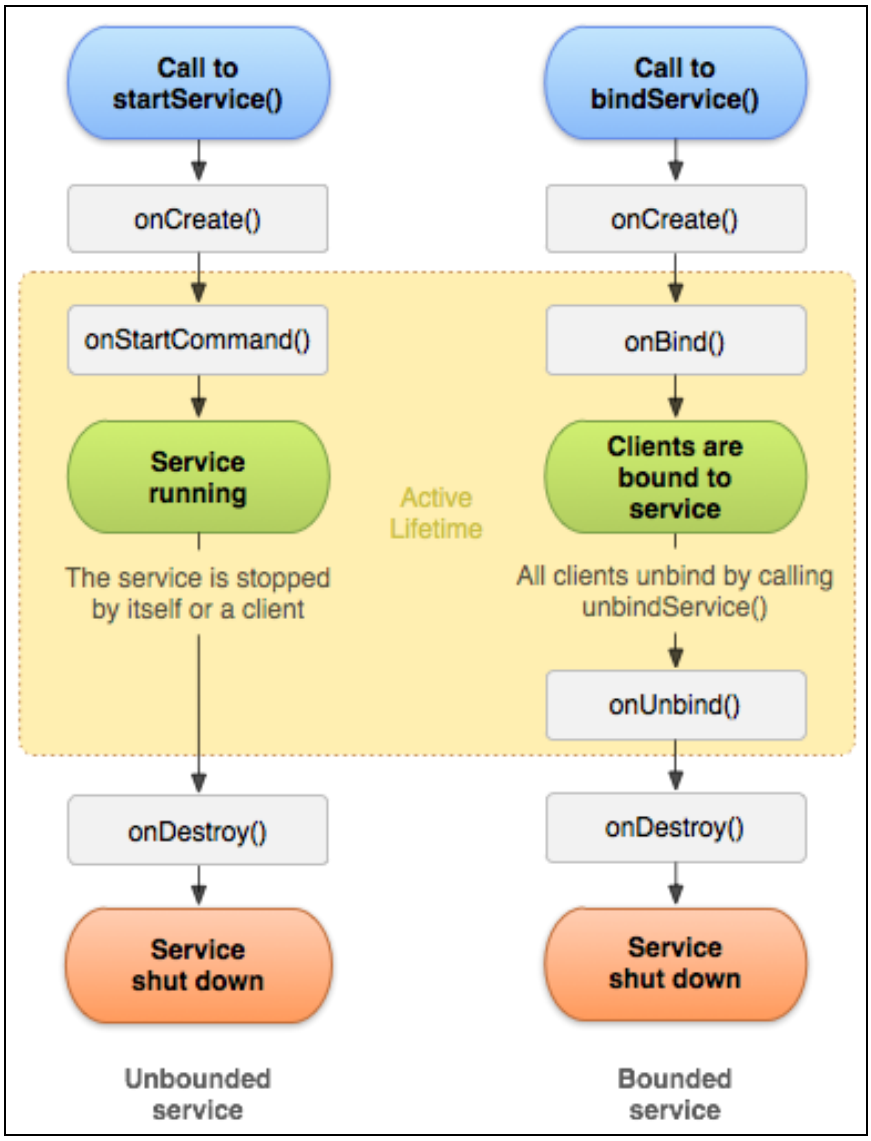

FIGURA 18: CICLO DE VIDA DE UN SERVICIO

3.1.2.2. RECEPTOR UDP

La clase `ReceptorUDP` tiene la siguiente estructura:

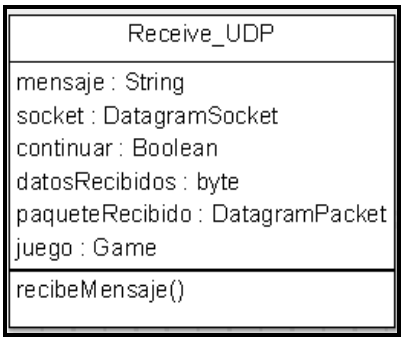

FIGURA 19: DIAGRAMA RECEPTOR UDP

Esta clase se encarga de recibir los paquetes que se envían desde el servidor, tratarlos y redirigirlos al juego. Estos mensajes hacen referencia a todo lo que puede pasar en la partida, incluyendo todos los movimientos que realiza el jugador contrario y todo lo relacionado con la lógica del juego (disminución de la vida, finalización de la partida…). Los diferentes mensajes que se reciben están explicados en el apartado 3.2. Protocolo de comunicación.

Esta clase implementa la interfaz `Runnable` ya que se trata de un hilo que se ejecutará a parte del hilo principal. De esta forma, se permite que el juego siga funcionando sin que estas tareas ocupen el hilo principal. Se encuentra ejecutándose en un bucle escuchando por el socket UDP hasta que o bien, el jugador salga de la partida intencionadamente, o bien la partida finalice cuando uno de los jugadores haya agotado su vida.

3.1.2.3. EMISOR UDP

El emisor de UDP está implementado en la clase `senderUDP` y contiene la siguiente estructura:

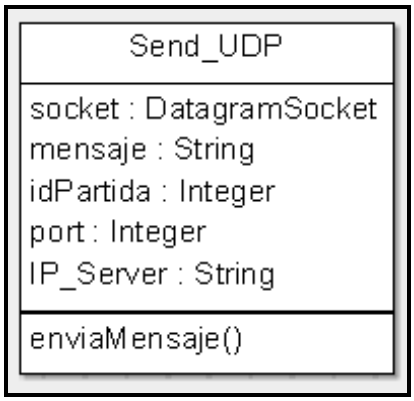

FIGURA 20: DIAGRAMA DEL EMISOR UDP

Esta clase es la encargada en enviar paquetes con la información de cada uno de los movimientos que realiza el jugador. Esta información se envía a través de un socket conectado al servidor mediante el protocolo UDP.

Para enviar paquetes, cada vez que se crea uno desde la partida, se llama al método `enviaMensaje` de la clase. Este método hace una llamada a una `AsyncTask` que es la que realmente envía el paquete al servidor a través del socket. Hay que emplear esta clase ya que Android no permite realizar comunicaciones a través de la red en el hilo principal de la aplicación, sólo es posible realizarlo o con un hilo a parte del hilo principal o, como en este caso, usando una tarea asíncrona. Esta tarea funciona de la siguiente manera:

```
public void enviaMensaje (String mensaje){
            this.mensaje = mensaje;
            Log.d("ENVIAR", "SENDED:" + mensaje);
            new Send().execute();
}

class Send extends AsyncTask<Void, Void, Void>{

      byte [] enviarDatos;

      @Override
      protected Void doInBackground(Void... params) {
          InetAddress DireccionIP = null;

          try {

            DireccionIP = InetAddress.getByName(IP_SERVER);
            mensaje = mensaje + ";idPartida:" + idPartida;
            enviarDatos = mensaje.getBytes();

            DatagramPacket enviarPaquete = new
DatagramPacket(enviarDatos,   enviarDatos.length, DireccionIP,
PUERTO_SERVER);

            socket.send(enviarPaquete);

          } catch (IOException e){
            e.printStackTrace();
          }
            return null;
      }
}
```

Todos los paquetes que se envían al servidor están detallados en el apartado 3.2. Protocolo de comunicación.

3.2. PROTOCOLO DE COMUNICACIÓN

3.2.1. PROTOCOLO TCP

A continuación se muestran los mensajes que se emplean para transmitir información cliente/servidor con su respuesta. Estos mensajes se envían usando el protocolo TCP ya que se trata de mensajes que contienen información crítica para la correcta configuración del juego.

Como se podrá ver, ninguno de estos mensajes incluye información sobre el remitente. Esto se debe a que se trata de un protocolo en el que se establece conexión cliente/servidor. Es decir, cuando un cliente quiere conectarse a la aplicación se crea un *Socket* en el lado del cliente al que es necesario pasarle como argumentos la dirección IP y el puerto del servidor en el que se ha creado un `ServerSocket` al arrancarlo. Cuando el servidor acepta una nueva conexión, se recoge el socket que se genera con el método `accept` de la clase `ServerSocket`. De esta forma, dado que se trata de un protocolo orientado a conexión, se mantiene la conexión cliente/servidor y el servidor puede seguir aceptando conexiones mientras que el objeto `cliente` (ver apartado *3.1 Diseño del servidor*) puede atender las peticiones que recibe del usuario. Por tanto, no es necesario incluir información sobre la IP del remitente en ningún sitio ya que no es posible que se envíe información a un cliente que no le corresponda.

Petición de conexión

Este mensaje lo recibe el cliente cuando solicita la conexión al juego y el servidor le responde que puede realizar la conexión.

```
OK CONEXION
```

Nombre de usuario y contraseña

Este mensaje se envía después de recibir el primer mensaje de que la conexión al servidor es correcta.

```
LOGIN; Nombre de usuario; Contraseña
```

El cliente puede recibir dos respuestas a este mensaje:

- Si el nombre de usuario y la contraseña son correctos el cliente recibe:

```
OK LOGIN
```

- En caso de que no haya sido correcto:

```
ERR LOGIN
```

Alta en el juego

Cuando un usuario se quiere dar de alta en el servidor envía una solicitud que puede recibir dos respuestas dependiendo de si el nombre de usuario está o no ocupado.

Cliente:

```
NEW LOGIN; Nombre de usuario; Contraseña
```

Servidor:

- Si el alta ha sido correcta

```
OK NEW LOGIN
```

- Si no ha sido correcta

```
ERR NEW LOGIN
```

Número de jugadores

Este mensaje lo envía el cliente cada vez que vuelve a la actividad principal de la aplicación. El servidor responde con el número de jugadores que hay online en ese momento.

Cliente:

```
NUM JUGADORES
```

Servidor:

```
NUM JUGADORES; Número de jugadores
```

Número de partidas

Cuando el usuario desea consultar qué partidas están ya en juego envía el siguiente mensaje solicitando el número de partidas activas y el id de cada una de estas.

```
NUM PARTIDAS
```

A lo que el servidor responde con:

```
NUM PARTIDAS; Número de partidas; id Partida 1; …; id Partida N
```

Nueva partida

Si el usuario quiere lanzar una nueva partida envía el siguiente mensaje:

```
NUEVA PARTIDA
```

El servidor crea la partida y envía el siguiente mensaje:

```
OK NUEVA PARTIDA; id Partida
```

Unirse a una nueva partida

Este mensaje se envía cuando el usuario quiere unirse a una partida que está ya creada.

```
JOIN PARTIDA
```

El servidor responde con el siguiente mensaje:

```
OK JOIN PARTIDA; id Partida
```

Empezar partida

Cuando la partida está ya preparada para comenzar el servidor envía el siguiente mensaje a los clientes que están conectados para indicarles que la partida ya ha comenzado:

```
RUN PARTIDA
```

Salir del juego

Cuando un usuario quiere salir de la aplicación, se avisa al servidor de que va a abandonar para que éste lo elimine de su lista de clientes.

```
SALIR JUEGO
```

3.2.2. PROTOCOLO UDP

Este tipo de mensajes se envían en el desarrollo de las partidas. Para ello, se hace uso del protocolo UDP ya que es necesario que el juego sea fluido y no se produzcan interrupciones que entorpezcan el desarrollo de la partida, tal y como se ha explicado en apartados anteriores.

Para establecer la comunicación, cuando arranca el servidor se crea un `DatagramSocket` asociado al puerto 5556 que servirá para recibir y enviar información a los clientes. Esto se hace de esta forma ya que no es eficiente ni práctico abrir un puerto por cada partida que se crea. Además podría darse el caso de que el número de partidas superara al número de puertos disponibles (65535).

En la parte del cliente, cuando se accede a una partida, la aplicación crea otro `DatagramSocket también` asociado al puerto 5556 que, como en el caso anterior, enviará y recibirá información al servidor.

En este caso, dado que se trata de un protocolo en el que no se establece conexión se hace necesario incluir en el paquete el destinatario. Para ello, se emplea un `DatagramPacket` cuyo constructor pide que se incluya tanto la dirección IP del destinatario como el puerto por el que está escuchando.

En todos los mensajes de este protocolo se incluye el id de la partida desde la que se ha enviado. Esto es totalmente necesario ya que los mensajes que se envían al mismo puerto por tanto, es necesario diferenciar de alguna forma desde qué partida se ha enviado ese mensaje, así que esa información se incluye en el mismo mensaje. Cuando este mensaje llega al servidor, todos los hilos de las partidas detectan que ha llegado un nuevo mensaje, comprueban que el id la partida incluido en el mensaje corresponde a su id. En caso de que coincidan, tratan el mensaje, si no es así, simplemente ignora el mensaje y se queda a la espera de uno nuevo.

A continuación, se muestran los mensajes que se emplean en este protocolo:

Abandonar partida:

Este mensaje se envía cuando uno de los usuarios sale de la partida antes de que haya finzalizado y al otro usuario se le indica que ha habido un abandono:

SALIR PARTIDA	ABANDONO

Movimientos:

Estos mensajes son los que se mandan cuando un jugador realiza un movimiento. El servidor se encarga de reenviar esta información al otro cliente tal y como está explicado en el apartado *3.1.1.2 Gestor de partidas*.

Movimento; id partida

Donde *movimiento* puede ser:

MOVIMIENTO	OBSERVACIÓN
`derecha`	El jugador se está moviendo hacia la derecha.
`izquierda`	El jugador se está moviendo hacia la izquierda.
`arribaDcha`	El jugador está saltando arriba a la derecha.
`arribaIzq`	El jugador está saltando arriba a la izquierda.
`pegarD`	El jugador ataca hacia la derecha.
`pegarI`	El jugador ataca hacia la izquierda.
`abajoDcha`	El jugador cae hacia la derecha.
`abajoIzq`	El jugador cae hacia la izquierda.
`abajo`	El jugador cae recto.
`para`	El jugador se ha detenido.

TABLA 1: MOVIMIENTOS DE UN JUGADOR

Resta de vida:

Este mensaje lo recibe el jugador cuando el otro jugador le ataca:

```
quitarVida; cantidad de vida; id partida
```

Correción de posición

Este mensaje es el que envía el servidor a los usuarios sobre la posición que tiene el servidor de los jugadores en la partida (ver apartado *3.2. Sistema de predicción*)

```
FIX POS;x;y;jugador
```

Donde *x* e *y* son las coordenadas del jugador, y *jugador* es el jugador del que hay que corregir la posición.

Fin de la partida

Ocurre cuando el servidor detecta que la partida ha finalizado.

```
FIN PARTIDA; ganador
```

3.3. SISTEMA DE PREDICCIÓN

Cuando se está diseñando un juego en el que se puede jugar online con otros jugadores con una arquitectura cliente/servidor surge el problema de saber cómo hacer llegar los movimientos de un jugador al dispositivo del jugador contrario.

Por un lado, se podría enviar al servidor continuamente la posición que está ocupando el jugador a medida que se mueve y que éste las reenviara al resto de jugadores para que las actualizaran en su correspondiente dispositivo. De esta forma, el resto de dispositivos dibujan al jugador contrario en la posición que recibe del servidor. Esto hace que el sistema sea mucho más simple ya que el servidor actuaría simplemente como intermediario entre un jugador y otro en una partida. Sin embargo, si la cantidad de paquetes que se pierden en la red es muy elevada, no se vería el juego de manera fluida, sino que se mostraría al jugador oponente a saltos en la pantalla en lugar de ser un movimiento fluido, que es lo que se pretende. También, el nivel de saturación de paquetes que llegan al servidor es muy alto lo que puede provocar mayores pérdidas de paquetes en la red.

Por otro lado, se puede enviar al servidor los movimientos que el jugador esté realizando en cada momento (derecha, izquierda, arriba, abajo…) y que éste los reenvíe al otro jugador, en lugar de enviar la posición que ocupa el jugador cada vez que éste se mueve. De esta forma, el otro jugador puede mover al jugador contrario sólo con saber el movimiento que está realizando y sin necesitar que el servidor le envíe continuamente la posición del contrario. El servidor por su parte actualiza la posiciones de ambos jugadores de la misma forma que lo hace cada dispositivo. Cuando uno de los jugadores se detiene, cambia de movimiento o cada cierto tiempo, en este caso, 500ms, el servidor manda a los dos jugadores las posiciones que tiene él almacenadas. Una vez que los jugadores han recibido esta información, comprueban si las posiciones que ellos tienen de los jugadores son las mismas que las que han recibido. Si no lo son, corrigen las posiciones a las que han recibido. Estas correcciones son las que provocan que en juegos multijugador se produzcan ciertos saltos en pantalla.

Este segundo caso es el que se conoce como sistema de predicción ya que la aplicación prevé en qué posición debe estar el jugador contrario en función de los movimientos que está recibiendo, sin necesitar continuamente una confirmación de las posiciones. Con este sistema, se mantiene una sincronización en ambos dispositivos porque al recibir la misma información de la misma fuente, el servidor, en caso de que alguno de los paquetes se perdiera en cualquiera de las fases del envío, provocaría que no se actualizaran las posiciones del jugador o en el servidor o en el otro dispositivo causando una desincronización en ambos dispositivos y sería el servidor el que actuaría como corrector para indicar qué posición del jugador es la correcta. Este es el sistema que se ha aplicado para este proyecto.

A continuación, se puede ver un ejemplo de cómo funciona:

La posición inicial cuando comienza la partida es igual tanto en los clientes como en el servidor:

FIGURA 21 : ESTADO INICIAL PARTIDA

A continuación, el jugador empieza a moverse hacia la derecha, por tanto, se manda un mensaje `derecha` (ver apartado 3.3.2):

- En caso de que no hubiera pérdida de paquetes se empezarían a actualizar tanto la posición en el servidor como en los dos clientes. Se muestra solo una imagen ya que en el otro cliente ocurriría lo mismo:

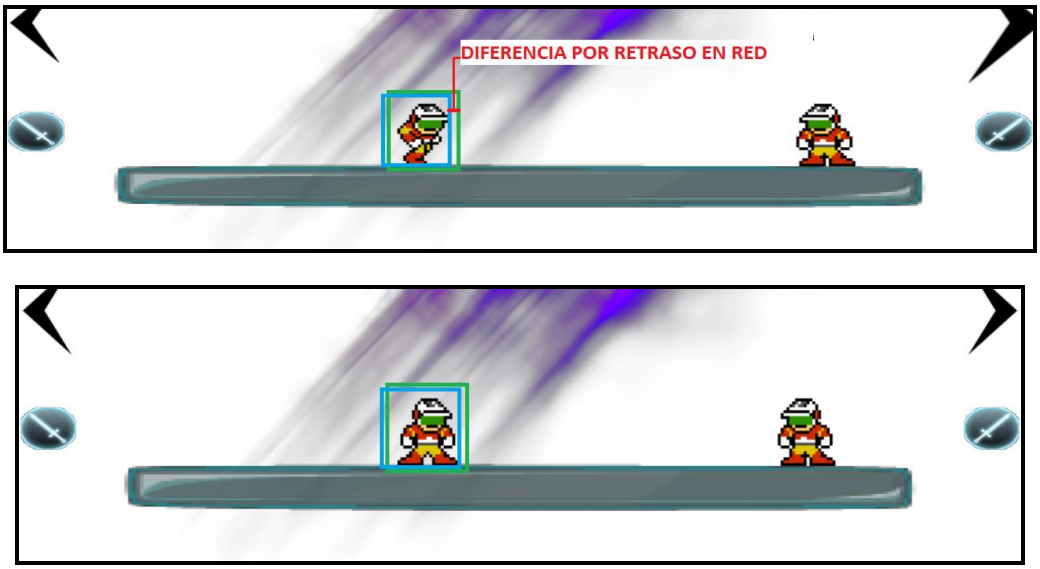

FIGURA 22: CLIENTE Y SERVIDOR COORDINADOS
(COLOR VERDE= CLIENTE; COLOR AZUL= SERVIDOR)

El jugador se detiene y, a continuación, se revisa la coincidencia de la posición del servidor con la de los clientes. Dado que la diferencia no es muy grande, la corrección es apenas visible por el usuario.

- En caso de que el paquete que contiene el movimiento que ha hecho el jugador se pierda y no llegue al servidor (como consecuencia tampoco llegaría al jugador contrario):

  En la pantalla del jugador que se mueve ocurre lo siguiente:

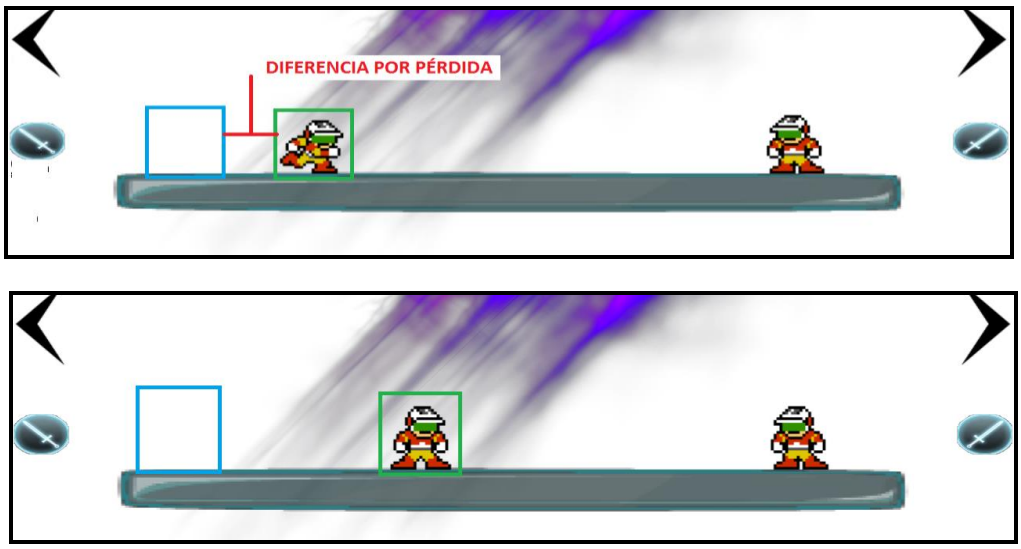

FIGURA 23: CLIENTE Y SERVIDOR DESFASADOS POR PERDIDA EN EL PRIMER ENVÍO (COLOR VERDE= CLIENTE; COLOR AZUL= SERVIDOR)

  En la pantalla del jugador contrario no se ha detectado el movimiento del contrario, por tanto, se mantiene en el estado de la figura 18.

  En este momento, el servidor manda a los clientes la posición que tiene almacenada de ambos. En el caso del jugador contrario, no hay que hacer ninguna corrección. Sin embargo, en la pantalla del jugador que ha realizado el movimiento se provoca una corrección que hace que el jugador se retrase a la posición que tenía antes de iniciar el movimiento.

- En caso de que el paquete que contiene el movimiento que ha hecho el jugador llegue al servidor pero no llegue al jugador contrario:

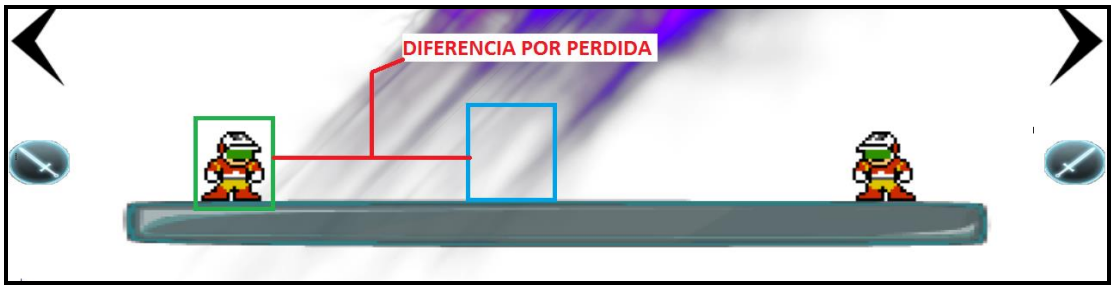

FIGURA 24: CLIENTE Y SERVIDOR DESFASADOS POR PERDIDA EN EL SEGUNDO ENVÍO (COLOR VERDE= CLIENTE; COLOR AZUL= SERVIDOR)

En el momento en el que el jugador se detiene, la diferencia entre la posición del servidor y la que tiene el jugador contrario es muy grande, dado que se ha perdido la información con el movimiento. Mientras, en el jugador que ha realizado el movimiento, el estado de la partida es el de la figura 19B. Por ello, se produce una corrección en el jugador contrario que da como resultado el siguiente estado:

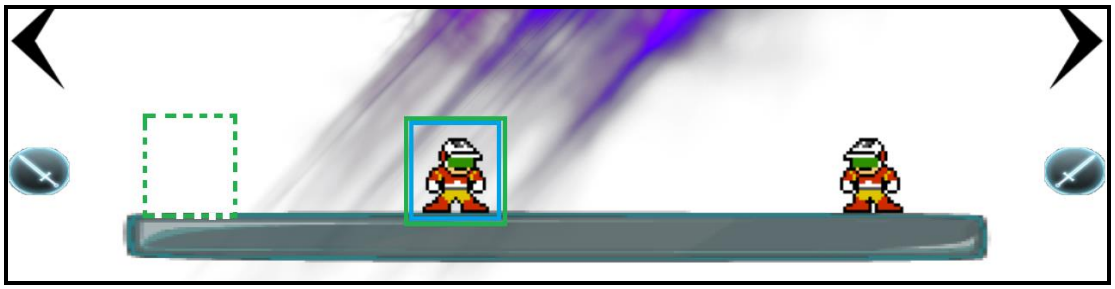

FIGURA 25: CORRECCIÓN EN EL DISPOSITIVO DEL JUGADOR CONTRARIO (COLOR VERDE= CLIENTE; COLOR AZUL= SERVIDOR)

En la imagen se puede ver que la posición del jugador contrario se ha corregido a la enviada por el servidor, dejando así el mismo estado de la pantalla en ambos dispositivos.

En todos estos casos, hay que recordar que gracias al sistema de predicción ha sido necesario enviar únicamente tres paquetes de información: uno para el movimiento a la derecha, uno para cuando el jugador se detiene y otro con las posiciones del servidor. Si este sistema no se hubiera implementado hubiese sido necesario enviar la posición del jugador cada vez que se moviera.

3.4. DISEÑO Y DESARROLLO DEL VIDEOJUEGO

Este juego entra dentro de los juegos de plataforma en combinación con juegos de lucha. Básicamente consiste en agotar la vida del otro jugador, o bien, golpeándole y quitándole vida, o bien, haciendo que caiga de alguna de las plataformas.

Dentro de la aplicación se han desarrollado cuatro actividades. Se trata de actividades con un diseño muy simple que permiten al usuario navegar de manera sencilla a través de la misma.

La navegación entre ellas es la siguiente:

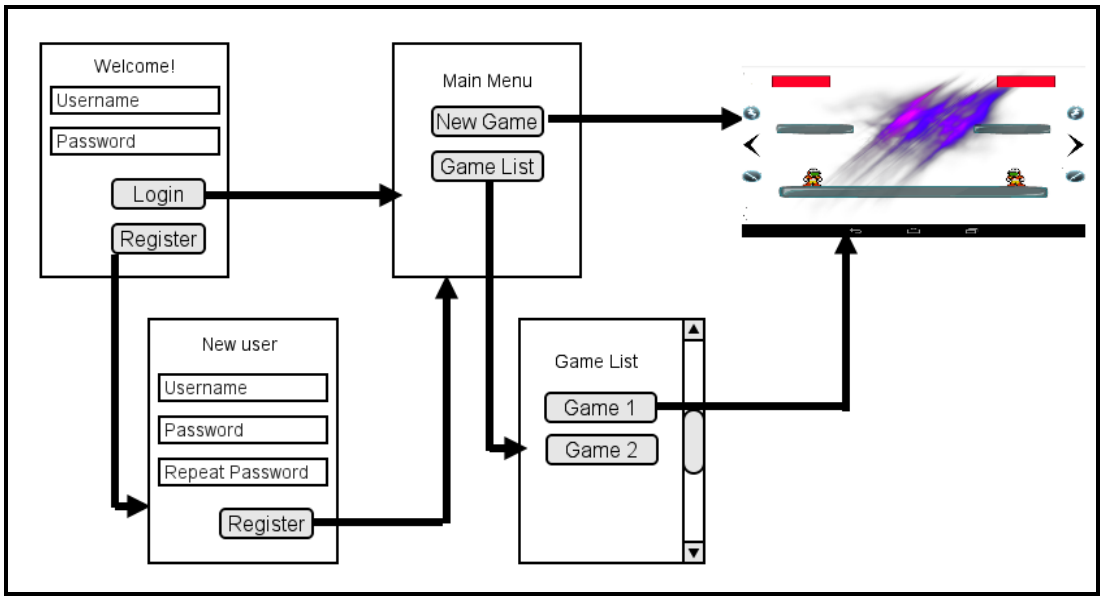

FIGURA 26: NAVEGACION ENTRE ACTIVIDADES

3.4.1. ACTIVIDAD INCIAL

Esta actividad es la primera que se muestra cuando el usuario arranca la aplicación. En ella, se le solicita al usuario que introduzca sus datos de registro (nombre de usuario y contraseña). Cuando el usuario inserta sus datos se envía la información al servidor para que lo compruebe y éste le devuelve el resultado de esta consulta. Si es correcto se le manda al usuario a la actividad `menu_principal`. Si no lo es se le pide que introduzca otra vez sus datos.

También se le da la opción al usuario de que se pueda registrar si no lo estaba. Esta opción le lleva al usuario a la actividad `nuevo_usuario`.

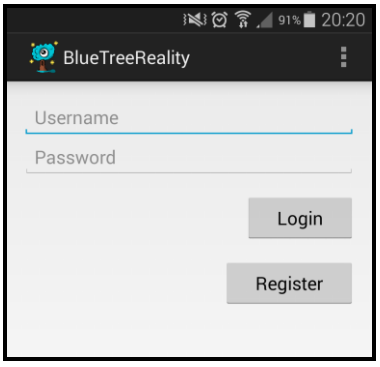

FIGURA 27: ACTIVIDAD INICIAL

3.4.2. ACTIVIDAD NUEVO USUARIO

Esta actividad permite el registro de nuevos usuarios al sistema. Para ello, se le solicita al usuario un nombre de usuario y que inserte dos veces la contraseña (para evitar errores) que quiere emplear.

Una vez introducidos, se le envía al servidor la información del nuevo registro y éste le responde informando de si ha sido correcto o no. Si es correcto se le manda al usuario a la actividad `menu_principal`. Si no lo es, se le pide que vuelva a introducir nuevos datos.

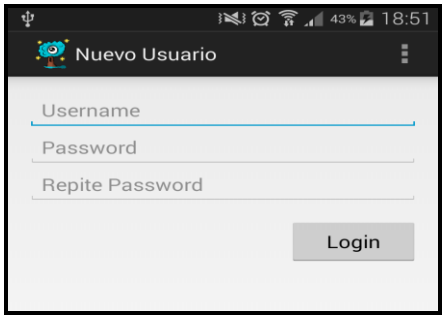

FIGURA 28: ACTIVIDAD NUEVO USUARIO

3.4.3. ACTIVIDAD MENU PRINCIPAL

Esta actividad es el menú principal de la aplicación. En ella, se le muestran dos opciones al usuario. Por un lado, el usuario puede comenzar una nueva partida y por otro, puede consultar la lista de partidas que hay en juego ahora mismo. La primera opción le lleva directamente al juego en sí, mostrándose una imagen de espera hasta que se conecte un nuevo jugador. La segunda le lleva a la actividad `lista_partidas`.

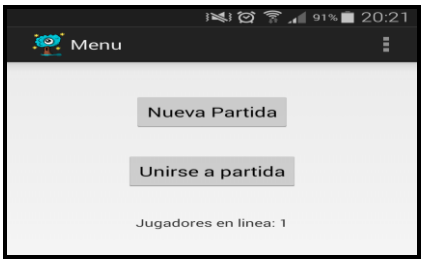

FIGURA 29: ACTIVIDAD MENU PRINCIPAL

3.4.4. ACTIVIDAD LISTA PARTIDAS

En esta actividad se muestran a modo de botones las partidas que hay creadas en el servidor que están a la espera de que se una un jugador para poder comenzar. Cuando el jugador elige la partida que quiere jugar, se inicia la actividad `partida`.

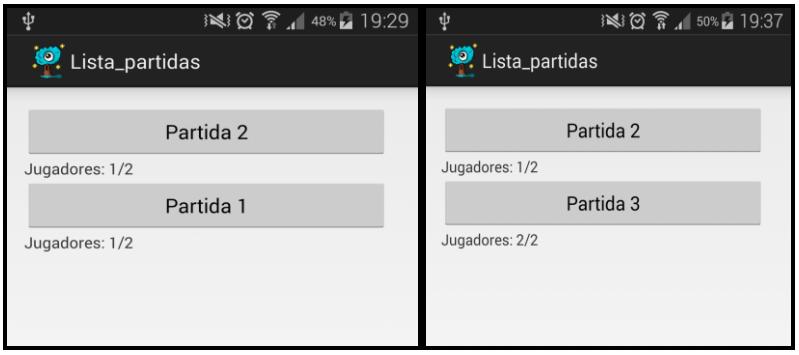

FIGURA 30: ACTIVIDAD LISTA PARTIDAS

3.4.5. ACTIVIDAD PARTIDA

Esta es la actividad principal de la aplicación. Se encarga tanto de enviar y recibir paquetes por UDP como de cargar y mantener todo el aspecto gráfico del juego

Cuando arranca esta actividad, la primera función que realiza es la de crear el socket asociándolo a un puerto del dispositivo que es que se empleará para el intercambio de información con el servidor. Una vez hecho esto, crea los objetos `sender_UDP` y `receiver_UDP` (apartado *3.1.2 Diseño del cliente*) a los que les pasa el socket previamente creado.

También se encarga de configurar la pantalla para poder visualizar una pantalla OpenGL en su versión 2.0 haciendo uso del objeto OpenGLConfig proporcionado por la API BTR [1]. Gracias a este objeto, se cargan todos los elementos gráficos del juego.

Para el juego en sí, se han desarrollado tres clases:

- **`GameResources`**

En ella se cargan y se sitúan en pantalla todos los elementos visuales que componen el juego: los personajes, las plataformas, las barras de vida, los botones del controlador, los diferentes rótulos que se muestran y el fondo de la pantalla.

- **`Player`**

Esta clase es la que contiene toda la información de los jugadores que están participando en la partida. En ella, se mantienen actualizadas las posiciones del jugador que se modifican a través de la clase `Game` cada vez que el usuario pulsa un botón. Además también se mantiene un estado del movimiento que está realizando el jugador que puede ser: saltando, derecha, izquierda, cayendo o atacando.

- **`Game`**

Esta clase es la más importante pues controla todo el videojuego. Por un lado, se encarga de capturar todos los gestos del usuario mediante el método `onTouchEvent` que se activa cuando el usuario realiza un gesto en la pantalla y por otro lado se encarga de comprobar todas las colisiones que puede tener un jugador, bien con las plataformas, o bien, con el jugador contrario.

A continuación, se describe la implementación del método `onTocuhEvent`:

Cuando se crea la clase `Game` se crean una serie de círculos invisibles para el usuario que se sitúan encima de cada uno de los botones de la pantalla. Además se crea otro círculo que se sitúa en mitad de la pantalla. Este último se irá moviendo a las posiciones en las que el usuario haya tocado la pantalla. En el momento en el que el usuario toque la pantalla se llamará a esta función que capturará el movimiento. De esta forma, si este

último círculo se solapa con alguno de los botones en pantalla se sabrá que el usuario quiere pulsar ese botón y por tanto se efectuará el movimiento correspondiente a ese botón. Esto se puede ver con mayor claridad en las siguientes imágenes:

La primera imagen corresponde al estado en el que el usuario no ha pulsado la pantalla En ese momento, el círculo de color verde es el que corresponde a los movimientos del usuario y se encuentra situado en mitad de la pantalla. Mientras que los que corresponden a los botones están situados encima de cada uno de ellos

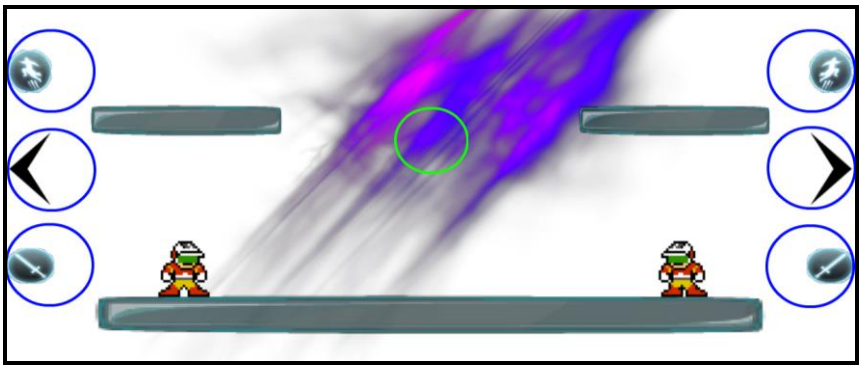

FIGURA 31: IMAGEN A DEL CONTROLADOR

La segunda imagen corresponde al momento en el que el usuario pulsa en alguno de los botones. En este caso, el usuario pulsa sobre el botón para mover al personaje hacia la derecha. Esto provoca que el círculo del usuario se mueva donde éste ha pulsado, haciendo que se solape con el círculo correspondiente al botón. De esta forma, el usuario puede mover el personaje por la pantalla. Lo mismo ocurre para el resto de movimientos:

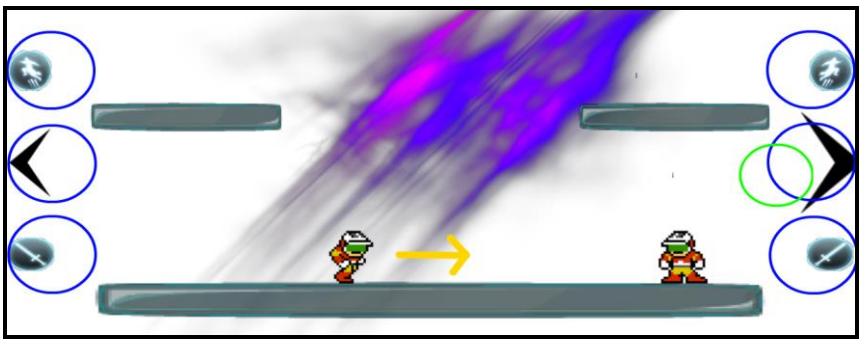

FIGURA 32: IMAGEN B DEL CONTROLADOR

Cuando se detecta que un botón ha sido pulsado, el movimiento que se está realizando se envía, a través del objeto `senderUDP`, al servidor para que éste realice el trabajo descrito en el apartado 3.1.1.

Una vez se ha capturado el movimiento, se le pasa este movimiento al objeto `Player` para actualizar las variables correspondientes al movimiento realizado.

Sesenta veces por segundo, la aplicación hace una llamada de forma automática al método `updateLogic` que básicamente se encarga de actualizar toda la partida en función del movimiento que se haya detectado después de que el usuario haya pulsado un botón. Para ello, el método comprueba qué movimiento está realizando el jugador mediante consultas al objeto `Player` y se pueden dar los siguientes casos:

- El jugador está saltando, ya sea hacia la izquierda o hacia la derecha. Para este caso, se hace necesario comprobar si el jugador, en la posición que va a ocupar tras la actualización, va a colisionar con algún objeto de la pantalla. Si es así es necesario cambiarle el movimiento al estado "cayendo". La detección de colisiones se realiza de la siguiente manera:

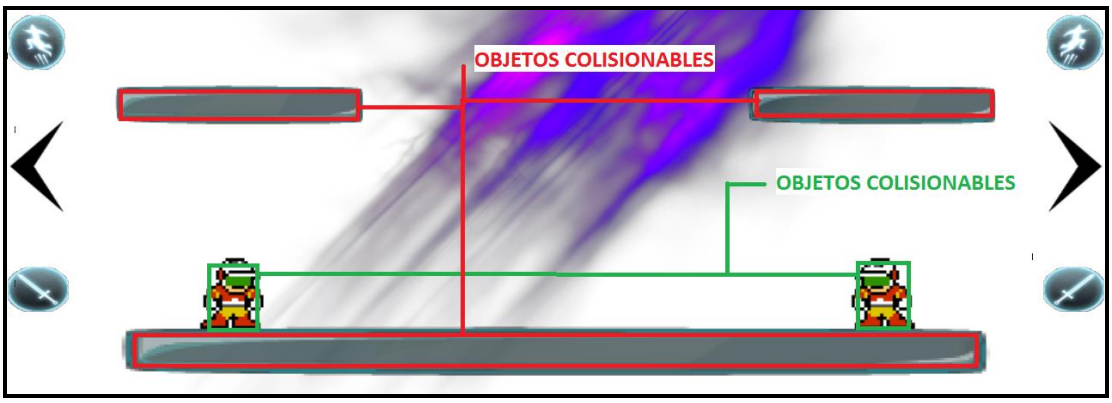

FIGURA 33: OBJETOS COLISIONABLES

  En la imagen se puede ver que en las plataformas y en ambos jugadores hay rectángulos (invisibles para el usuario) que tienen la misma forma y tamaño que el objeto que ocupan. Estos rectángulos se emplean para detectar las colisiones de tal forma que si estos rectángulos se solapan indicará que se ha producido una colisión entre ambos. Este método es el mismo que se emplea para detectar los botones que el usuario ha pulsado.

  Así, si el jugador está saltando, mientras está realizando el movimiento se comprueba si el jugador va a colisionar con alguna de las plataformas superiores. En el caso de estar saltando solo puede colisionar por la parte inferior de las plataformas. Cuando se detecta una colisión, se cambia el movimiento del jugador al de *cayendo* y se envía este cambio de movimiento al servidor para que actualice su información y para que la reenvíe al jugador contrario (**"abajo"**, ver Tabla 1: Movimientos de un jugador).

- El jugador está cayendo después de un salto o bien después de una colisión con una plataforma superior. Este caso es muy similar al anterior, la única diferencia es que la colisión, en este caso, solo se puede dar con la parte superior de la plataforma, es decir, que el único movimiento que puede seguir a una caída es que el jugador se detenga cuando se detecte la colisión con alguna de las plataformas. En ese

momento, se manda al servidor la información de que el jugador se ha detenido (“parar”, ver Tabla 1: Movimientos de un jugador), que enviará al otro usuario.

Debido al retraso en la red, ocurre el siguiente problema: mientras el paquete que contiene la orden de detener el movimiento llega al servidor, éste sigue actualizando las posiciones del jugador. Esto significa que para cuando llega el paquete, la posición del jugador en el servidor se encontraría dentro de la plataforma y no sobre ella. Entonces, cuando el servidor mande las posiciones corregidas al cliente, se mostrará al jugador dentro de la plataforma, como se puede ver en la figura 33, impidiendo el desarrollo del juego.

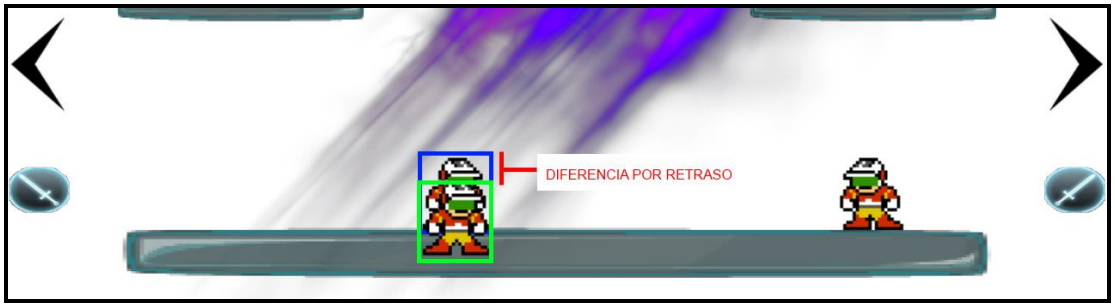

FIGURA 34: CORRECIONES EN EL SALTO

Para solucionar este problema, en el servidor, cuando llega un mensaje de “parar” y el estado del jugador que tiene almacenado es “cayendo”, se detiene el movimiento y se hace una pequeña corrección a la posición que debe tener en relación a la plataforma donde haya caído. A continuación, se manda la posición que tiene almacenada, siguiendo el protocolo explicado en el apartado del *3.3. Sistema de Predicción*.

Mientras no se produzca una colisión el jugador seguirá cayendo. Cuando el jugador quede por debajo de la plataforma inferior, significará que el jugador ha perdido la partida y se enviará el mensaje al servidor informando al respecto, que significará el final de la partida.

- El jugador está moviéndose ya sea hacia la izquierda o a hacia la derecha. En este caso, se comprueba continuamente que tiene una plataforma debajo, si no la tiene el jugador comienza a caer siguiendo el proceso anterior. También se comprueba si al moverse colisiona con el jugador contrario. Si es así, se le impide al jugador continuar y se envía un mensaje de parada al servidor.

- El jugador está atacando. Para este caso, se han añadido dos colisiones más a cada jugador. Se trata de dos círculos puestos a ambos lados del jugador que señalan su perímetro de ataque:

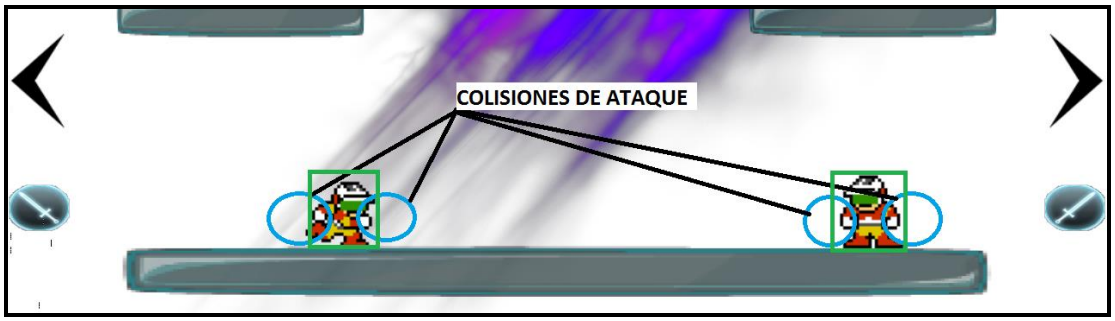

FIGURA 35 COLISIONES DE ATAQUE

De esta forma, cuando un jugador realiza un ataque se comprueba si su perímetro de ataque colisiona con el jugador contrario. En caso de ser así, el jugador contrario recibiría el ataque y se enviaría al servidor la información correspondiente al ataque. Como consecuencia, el servidor actualizará la información de la partida y la enviará al jugador contrario. Cuando se recibe un ataque, las barras rojas que hay situadas en la parte superior de la pantalla y que indican la vida de los jugadores (Figura 1), se van reduciendo hasta que no quede vida y se dé por finalizada la partida.

Por otro lado, se hace necesario el desarrollo de un método que recoja los mensajes que el servidor envía a través del socket UDP. Este método se denomina `ejecutaAccion` y es llamado por el objeto `ReceiveUDP` cada vez que se recibe un mensaje del servidor durante la partida. Los mensajes que se pueden recibir están explicados en el apartado *3.3.2. Protocolo UDP*.

3.5. BASE DE DATOS

Para poder almacenar y registrar nuevos usuarios en la aplicación se ha desarrollado una base de datos que se describe a continuación. Esta base de datos está formada por una única tabla:

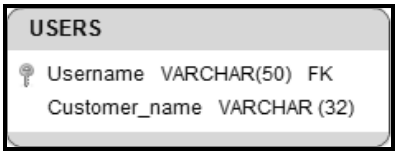

FIGURA 36: TABLA USUARIOS

En ella, se guarda el nombre de usuario que servirá como clave principal y la contraseña del usuario. La base de datos está desarrollada sobre MySQL que es uno de los gestores de bases de datos más empleados en el mundo.

Para poder entrar a la aplicación, el usuario debe introducir sus datos, en caso de que esté registrado, o realizar un nuevo registro. En caso de estar registrado, se comprueba primero que el nombre de usuario introducido existe y, en segundo lugar, que la contraseña es correcta. Si no se cumple alguno de estos requisitos, se manda un mensaje de error al dispositivo. Si el usuario es nuevo, se comprueba que el nombre que ha introducido no existe en la base de datos y, por tanto, puede realizar un nuevo registro.

Una vez se ha realizado el registro, se manda al usuario a la actividad del menú principal.

4. FUNCIONAMIENTO DE LA APLICACIÓN

En este apartado se explica de manera más gráfica el funcionamiento completo de la aplicación, explicando cuándo se conectan cliente y servidor (tanto en TCP como en UDP) y cómo se comunican entre ellos, qué mensajes se envían y cuáles son las respuestas. A continuación, se muestra un diagrama que representa de forma gráfica el funcionamiento, de manera resumida, de la aplicación:

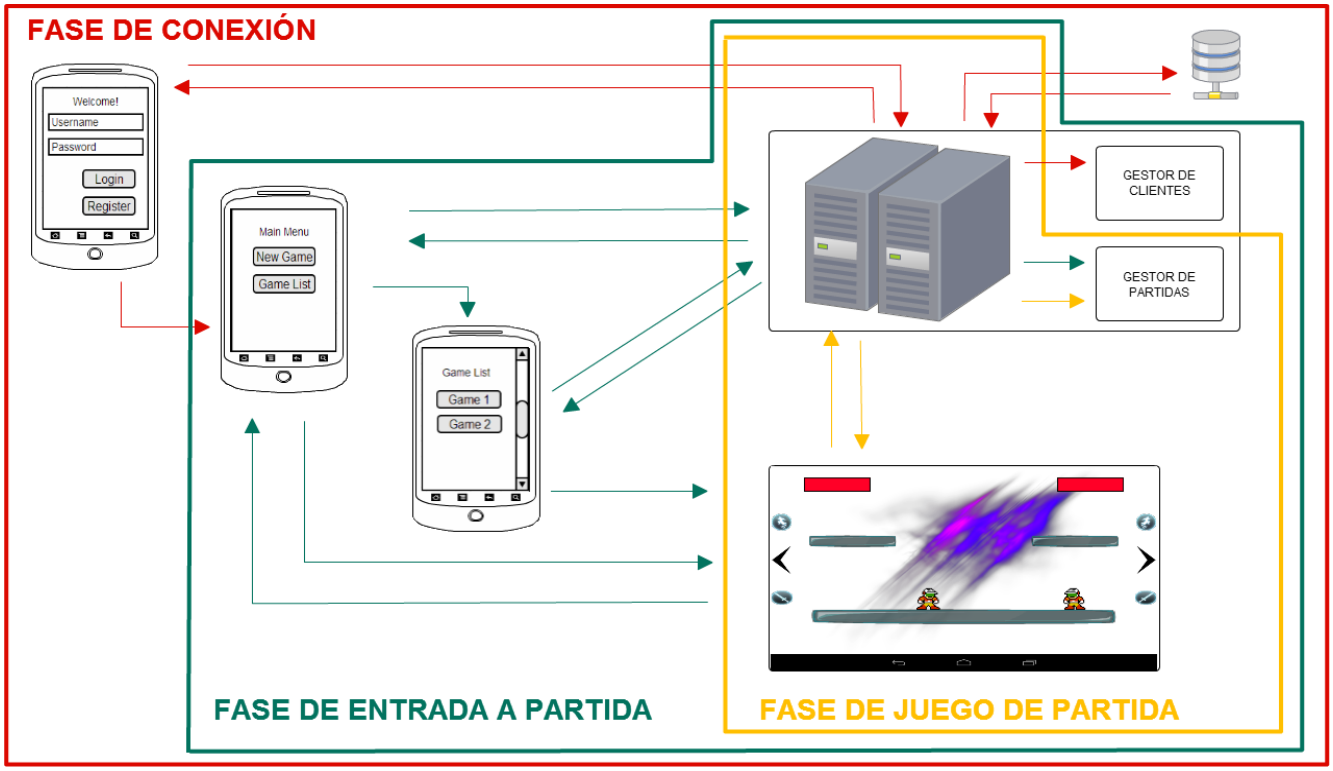

FIGURA 37: DIAGRAMA DEL FUNCIONAMIENTO GENERAL DE LA APLICACIÓN

Los apartados que siguen explican de forma detallada cada una de las fases mostradas en el diagrama anterior.

4.1. ARRANQUE DEL SERVIDOR

Durante esta fase, se carga todo el servidor. Por un lado se crean tanto el gestor de partidas como el de clientes. En ambos también comienzan a funcionar sus correspondientes `Timer` para chequear qué clientes y qué partidas siguen activos. Por otra parte, el servidor crea el `Socket` de conexión de tipo `ServerSocket` en el puerto 4446 que servirá para recibir las peticiones de conexión de los clientes. Y, por último, se realiza la conexión a la base de datos que almacena todos los datos de los usuarios. Una vez cargado todo esto, el servidor entra en un bucle infinito y se mantiene a la espera de nuevas conexiones gracias a la llamada al método `Accept` de la clase `Socket` que es bloqueante.

4.2. FASE DE CONEXIÓN

En esta fase el usuario introduce sus datos de registro y se lo comunica al servidor, siguiendo el siguiente esquema:

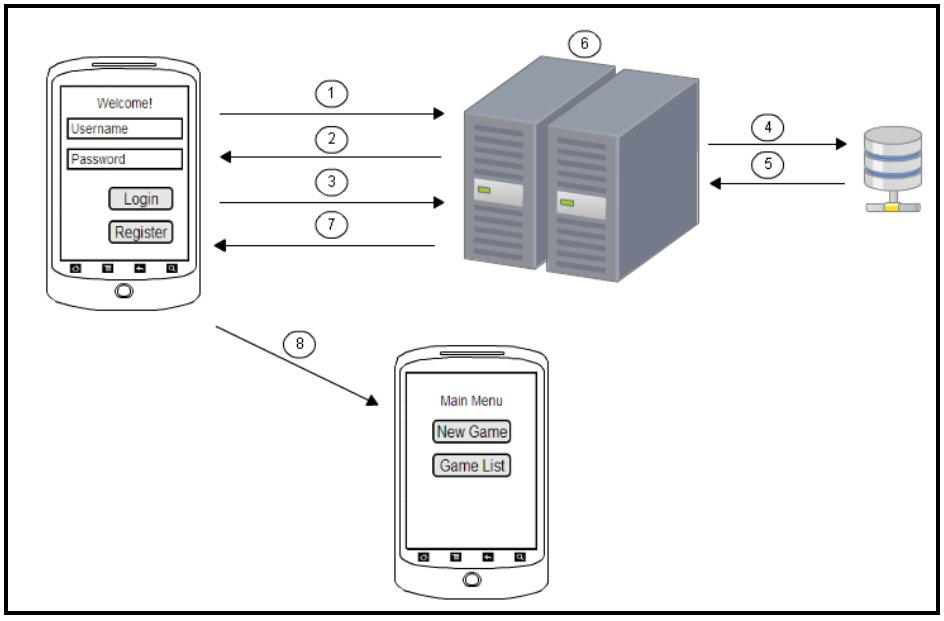

FIGURA 38: DIAGRAMA DE LA FASE DE CONEXION

1. Cuando el usuario introduce sus datos y pulsa sobre el botón de Login, la aplicación arranca el servicio `SERVICE_TCP` y éste envía una solicitud de conexión con el servidor.

2. El servidor acepta la conexión (`Socket.accept`) y le manda un mensaje de confirmación al cliente. También recoge el `socket` del cliente para comunicarse a través de él.

3. El cliente envía sus datos de usuario (`LOGIN; Nombre de usuario; Contraseña` (apartado *3.2.1. Protocolo TCP*)) a través del `socket`.

4. El servidor realiza una consulta a la base de datos pidiendo los registros de la tabla usuarios que coincidan en username y contraseña con los datos enviados.

5. El resultado de la consulta se envía al servidor

6. El servidor chequea que existe un solo registro en el resultado de la consulta.

7. Si los datos del usuario son correctos, el servidor manda un `OK_LOGIN` al cliente indicando que puede entrar o `ERR_LOGIN` en caso de que los datos no fueran correctos. En este momento el servidor crea un nuevo hilo para atender las peticiones del cliente y lo añade al gestor de clientes.

8. Si el registro ha sido correcto, la aplicación mostrará la actividad `MENU_PRINCIPAL`.

También se puede dar el caso de que el usuario quiera realizar un nuevo registro. En ese caso, en el paso 3, se manda el mensaje: `NEW_LOGIN; *Nombre de usuario*; *Contraseña*.` El servidor, en el paso 4, realiza una consulta pidiendo la cantidad de usuarios que coinciden con el nombre de usuario recibido. Si no hay ninguno, se registra al nuevo usuario y se le envía `OK_NEW_LOGIN.`

4.3. ENTRANDO A UNA PARTIDA

En esta fase se solicita al servidor entrar a una partida ya sea creando una nueva o bien uniéndose a una que está ya creada. Para ello, se sigue el siguiente esquema:

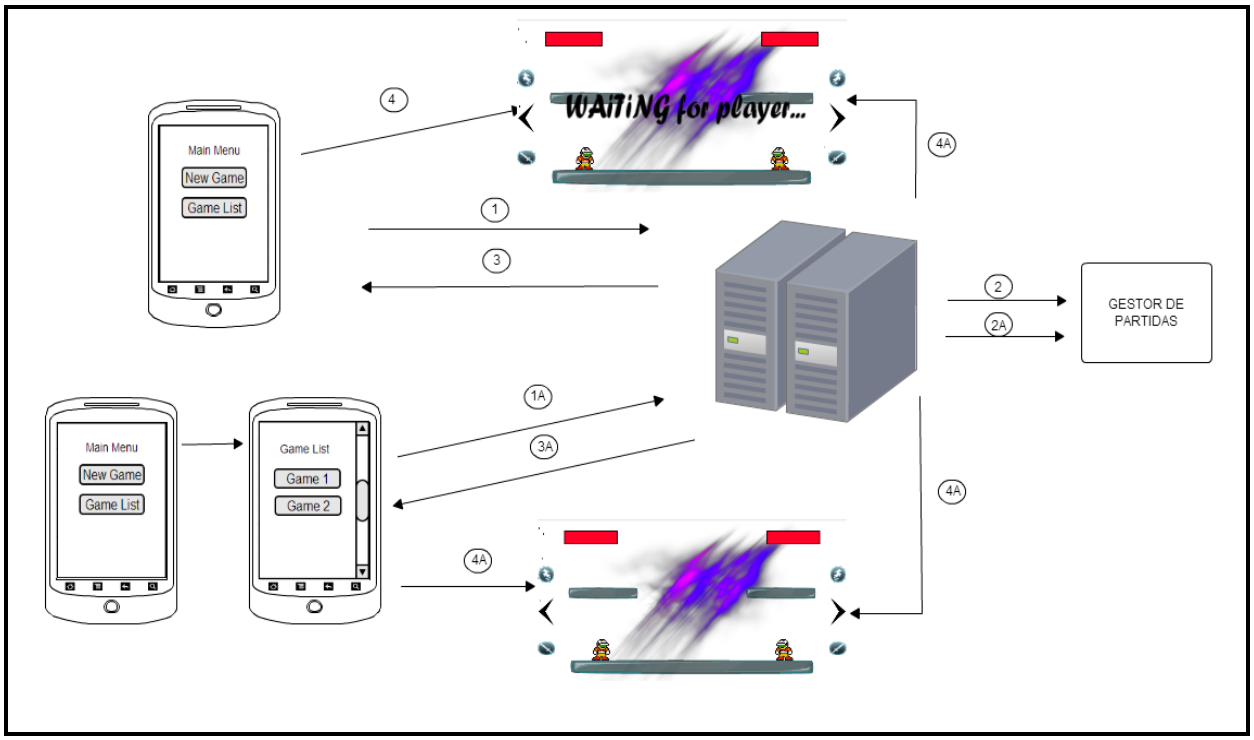

FIGURA 39: DIAGRAMA DE ENTRADA PARTIDA

1. El usuario quiere crear una nueva partida. Cuando pulsa sobre el botón "New Game", se llama al servicio `Service_TCP` para que envíe el mensaje de nueva partida al servidor (`NUEVA_PARTIDA`, apartado 3.2.1 Protocolo TCP). El servicio lo envía a través del `Socket` que se ha creado en la conexión.

2. El mensaje llega al hilo del cliente correspondiente del servidor (que es donde se había recogido el socket en la fase de conexión). Éste recibe el mensaje y se comunica con el gestor de partidas para crear una nueva partida, método `iniciaNuevaPartida` del gestor de partidas. Este método devuelve el ID de la partida creada que se almacena dentro del hilo para saber en qué partida está jugando el usuario.

3. Una vez creada la partida y recogido el ID de la misma, se envía el mensaje `OK_NUEVA_PARTIDA; id_partida` al cliente. Este mensaje es recibido por el `Service_TCP` y almacena el ID recibido en las preferencias de la aplicación de tal forma que puedan acceder a este valor el resto de actividades.

4. Cuando el servicio ha introducido el valor, la actividad `Menu_Principal` comprueba que hay un valor válido para el ID (distinto de -1) y, si lo es, lanza la

actividad `Partida`. Dado que hay un único jugador en la partida, se muestra un rótulo que dice "Waiting for player" y se inhabilita el controlador del juego hasta que entre un nuevo jugador.

Por otra parte, otro jugador desea entrar a una partida que está ya creada:

1A. El usuario quiere unirse a una nueva partida. Para ello, selecciona una de las partidas que haya activas en la actividad `Lista_Partidas`. Una vez seleccionada, el servicio manda el mensaje `JOIN_PARTIDA;idPartida` al servidor.

2A. El mensaje llega al hilo del cliente correspondiente del servidor (que es donde se había recogido el socket en la fase de conexión). Éste recibe el mensaje y se comunica con el gestor de partidas para unirse a la partida indicada, método `aniadeJugador` del gestor de partidas.

3A. Una vez se ha unido el jugador a la partida, se envía el mensaje `OK_JOIN_PARTIDA;idPartida` al cliente. Este mensaje es recibido por el `Service_TCP` que almacena el ID recibido en las preferencias de la aplicación de tal forma que puedan acceder a este valor el resto de actividades.

4A. Cuando el servicio ha introducido el valor, la actividad `Menu_Principal` comprueba que hay un valor válido para el ID (distinto de -1) y, si lo es, lanza la actividad `PARTIDA`. Al mismo tiempo, el servidor manda el mensaje `RUN_GAME` a ambos jugadores para que pueda comenzar la partida.

4.4. JUGANDO UNA PARTIDA

Durante la partida según los movimientos que se están realizando se van enviando los diferentes mensajes que se pueden ver en la tabla 1 *Movimientos de un jugador*. Cuando uno de los jugadores alcanza el objetivo final, se envía el mensaje `FIN_PARTIDA;ganador` indicando el jugador que ha ganado. Cuando esto ocurre se muestran los siguientes mensajes en función del resultado de la partida:

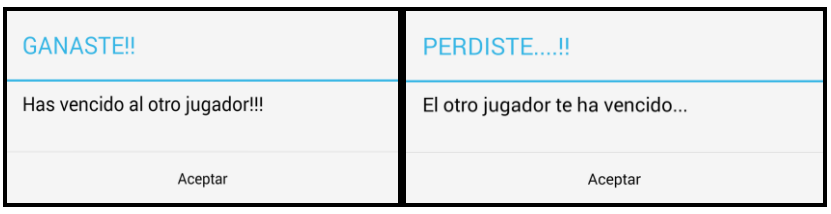

FIGURA 40: MENSAJES DE VICTORIA Y DERROTA

Si uno de los jugadores abandona la partida, se envía al servidor el mensaje `SALIR_PARTIDA`. Cuando el servidor lo recibe, se encarga de eliminar a ese jugador de la partida y envía al otro usuario el mensaje `ABANDONO` que indica que el otro usuario ha abandonado la partida. Finalmente, se muestra el mensaje de victoria:

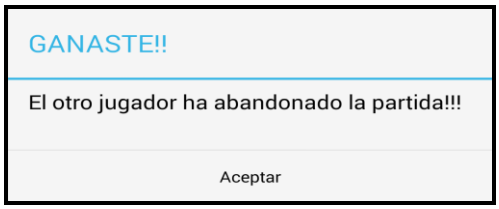

FIGURA 41: MENSAJE DE ABANDONO

A continuación, en ambos casos, se devuelve al usuario al menú principal. Mientras, el gestor de partidas comprueba si el hilo de la partida sigue activo a través del `TIMER` con el método `ISALIVE`. Dado que no lo está, la partida es eliminada de la lista de partidas, liberando el espacio para poder crear una nueva.

4.5. SALIENDO DE LA APLICACIÓN

Cuando el usuario quiere cerrar la aplicación se le muestra un mensaje de confirmación indicando si quiere abandonarla. Cuando el usuario confirma que quiere salir, se manda el mensaje `SALIR` al hilo correspondiente del cliente en el servidor. Cuando lo recibe, envía al cliente `OK_SALIR`, se cierra la conexión con el cliente con el método `close` de la clase `Socket` y el hilo sale del bucle en el que está, terminando su ejecución. El gestor de clientes comprueba que el hilo ya no se está ejecutando y lo elimina de la lista de clientes.

Mientras, en el cliente, una vez se ha recibido el mensaje `OK_SALIR`, se cierra el socket de conexión con el servidor y se detiene el servicio `Service_TCP`. Por último, se envía al usuario a la actividad de `Login`.

5. CONCLUSIONES Y TRABAJO FUTURO

En este proyecto se ha desarrollado una plataforma para dotar a los videojuegos de una funcionalidad muy demandada actualmente: jugar online contra otros jugadores en tiempo real en dispositivos Android.

Para llevarlo a cabo, se ha diseñado un sistema cliente/servidor con las siguientes características:

- El cliente realiza dos tareas básicas. Por un lado, recibe las peticiones que realiza el usuario, las envía al servidor y actúa en función de la respuesta recibida. Por otro lado, también gestiona la partida que está jugando el usuario, enviando los movimientos que realiza el jugador y recibiendo la información que envía el servidor sobre el otro jugador. La primera tarea funciona bajo el protocolo TCP y la segunda tarea bajo el protocolo UDP.

- El servidor se encarga de gestionar las peticiones de los usuarios que están conectados y de gestionar las partidas que se están jugando. Para ello, se han diseñado tanto un gestor de clientes como de partidas. Igual que en el caso anterior, el gestor de clientes funciona bajo el protocolo TCP y el segundo gestor bajo el protocolo UDP.

También se ha creado un protocolo que permite el intercambio de información entre el servidor y los clientes. Y, para mantener la sincronización entre cliente y servidor, se ha diseñado un sistema que permite que ambos se mantengan sincronizados.

Como trabajo para el futuro se han estudiado las siguientes líneas:

- Mejorar el diseño de los menús creándolos con la API *BlueTreeReality* de manera similar a cómo está hecho el videojuego.

- Añadir nueva funcionalidad a las partidas, como la utilización de objetos, nuevos ataques, nuevos personajes…

- Permitir que jueguen más de dos jugadores a la vez en la misma partida.

- Acumular más datos de los jugadores (victorias, derrotas…) para poder generar estadísticas.

- Con los datos anteriores, crear un sistema de autoasignación de un rival de un nivel similar al del usuario.

- Desarrollar un sistema de seguridad en la transmisión de paquetes de información aplicando un sistema de codificación.

Gracias a este trabajo, se ha conseguido entender cómo funcionan este tipo de videojuegos y cómo se solventan muchos de los problemas que surgen cuando se diseñan debido, fundamentalmente, a la necesidad mantener la sincronización entre los jugadores. En general, ha sido un trabajo muy satisfactorio en el que el aprendizaje ha sido muy alto.

BIBLIOGRAFÍA

[1] Femenía del Rey, Jorge. **Desarrollo de un motor gráfico sobre OpenGL para desarrollo de videojuegos 2D en dispositivos Android** [TFG]; Madrid; Universidad Autónoma de Madrid; Mayo de 2014.

Fledler, **Glenn. Game Networking**. [Artículo de Internet]; Octubre de 2008; Disponible en: http://gafferongames.com/networking-for-game-programmers/

Valve. **Source Multiplayer Networking**. [Artículo de Internet]; Última actualización: Enero 2014; Disponible en: https://developer.valvesoftware.com/wiki/source_multiplayer_networking

EpicGames. **Unreal Networking Architecture**. [Artículo de Internet]; 2012; Disponible en: http://udn.epicgames.com/Three/NetworkingOverview.html#The%20update%20loop

Departamento de lenguajes y sistemas informáticos de la Universidad de Alicante. **Sockets en Java**. [Artículo]; Disponible en: http://www.dlsi.ua.es/asignaturas/sid/JSockets.pdf

Google. **Mobile Multiplayer Made Manageable**. [Video]; Conferencia I/O 2013 de Google; EEUU; Mayo 2013; Disponible en: https://developers.google.com/events/io/sessions/325345389

Google. **Developing a Real-Time Multiplayer Game in Android** [Artículo de Internet]; Última actualización: Mayo de 2014; Disponible en: https://developers.google.com/games/services/android/realtimeMultiplayer

Google. **Services**[Artículo de Internet]; Disponible en: http://developer.android.com/guide/components/services.html

"*sebastiancipolat"*. **Sockets en Android** [Artículo de Internet]; Agosto de 2012; Disponible en: http://androideity.com/2012/08/05/sockets-en-android/

Diaz, F. **Sockets en Java** [Artículo]; Universidad de Segovia; Disponible en: www.infor.uva.es/~fdiaz/sd/doc/java.net.pdf