UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO E IMPLEMENTACIÓN DE UN AJEDREZ ONLINE PARA ANDROID

César Montaner Cardoso

Tutor: José Enrique López Patiño

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2018-19

Valencia, 5 de septiembre de 2019

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

El presente trabajo de fin de grado consiste en el desarrollo de un ajedrez online para Android en español e inglés mediante el entorno de desarrollo Android Studio utilizando el lenguaje de programación Java. La aplicación ha sido implementada en diversos dispositivos móviles, tanto virtuales como físicos, para lograr un diseño más consistente.

Las partidas consistirán en emparejar a dos usuarios de la aplicación que estén buscando el mismo modo de juego. Por ello, también se ha creado el servidor del juego mediante XAMPP. Éste estará formado por una base de datos y por ficheros PHP encargados de realizar consultas en lenguaje SQL a la base de datos y de responder a los usuarios.

Cada usuario podrá consultar su registro de partidas, las normas del juego, y seleccionar un modo de juego de entre tres: normal, rápido y relámpago, los mismos que en los estándares de competición. En el caso de que no disponga de conexión a Internet, podrá jugar en modo desconectado con otra persona en ese mismo dispositivo.

Resum

El present treball de fi de grau consisteix en el desenvolupament d'un escacs en línia per Android en espanyol i anglés mitjançant l'entorn de desenvolupament Android Studio utilitzant el llenguatge de programació Java. L'aplicació ha estat implementada en diversos dispositius mòbils, tant virtuals com físics, per aconseguir un disseny més consistent.

Les partides consistiran en aparellar a dos usuaris de l'aplicació que estiguin buscant el mateix mode de joc. Per això, també s'ha creat el servidor del joc mitjançant XAMPP. Aquest estarà format per una base de dades i per fitxers PHP encarregats de realitzar consultes en llenguatge SQL a la base de dades i de respondre als usuaris.

Cada usuari podrà consultar el seu registre de partides, les normes del joc, i seleccionar una manera de joc d'entre tres: normal, ràpid y llamp, els mateixos que en els estàndards de competició. En el cas que no disposi de connexió a Internet, podrà jugar en mode desconnectat amb una altra persona en aquest mateix dispositiu.

Abstract

This final degree project consists in the development of an online chess for Android in Spanish and English through the Android Studio development environment using the Java programming language. The application has been implemented in several mobile devices, both virtual and physical, to achieve a more consistent design.

The games consist of matching two application users who are looking for the same gameplay. Therefore, the game server has also been created using XAMPP. This will consist of a database and PHP files responsible for performing queries in SQL language to the database and responding to users.

Each user can check his game record, the rules of the game, and select a game mode from among three: normal, fast and blitz, the same as in the competition standards. If Internet connection is not available, the user can play in disconnected mode with another person on that same device.

Índice

1. Introducción

1.1 Motivación

Mi motivación para realizar este proyecto ha sido la de poner en práctica todos los conocimientos relacionados con bases de datos, programación en el lenguaje Java, Android, PHP y servidores que he ido aprendiendo durante los estudios de Grado, profundizando y aprendiendo más de éstos.

He decidido el caso de desarrollar un videojuego desde cero, partiendo de unas normas ampliamente conocidas, ya que ha supuesto una forma entretenida y eficaz de aprender más sobre el desarrollo de aplicaciones y de algoritmos de programación más enrevesados. En concreto, he elegido el clásico del ajedrez ya que es un juego muy antiguo, pero que hoy en día sigue teniendo un renombre considerable en el mundo de la competición y en el de los clásicos juegos de mesa.

1.2 Objetivos

El objetivo de este trabajo de fin de grado consiste en desarrollar desde cero una aplicación en Android en la que se pueda jugar al ajedrez desde cualquier ubicación con acceso a Internet. Esta aplicación podrá registrar usuarios nuevos, los cuales podrán jugar contra otros y poder ver su registro de partidas.

La aplicación debe de tener una estética simple y a la vez atractiva, dejando ver en algún lugar visible que se trata de un proyecto de la UPV y de la ETSIT. Esto último se indicará en el inicio de sesión y en el registro de usuarios, en la parte inferior de la pantalla del dispositivo mediante dos imágenes de los logos oficiales de estas entidades. También se incluirán las normas del juego en la aplicación para ayudar a los jugadores novatos.

Se deberá desarrollar también el motor de juego desde cero, que se encargará de validar los movimientos de los usuarios, regular los tiempos, indicar los movimientos válidos de una pieza seleccionada, recopilar la información del tablero a tiempo real y de indicar el fin de partida.

Se incluirán tres modos de juego que irán acorde a los estándares de competición para atraer a los jugadores más competitivos y que puedan jugar en su modalidad favorita. Estos modos varían en el tiempo de partida y en el incremento del tiempo de juego por cada turno [1].

También se desarrollará el servidor de juego, que incluirá la base de datos, con todos los registros de los usuarios y de las partidas que se están jugando, y los ficheros PHP que permitirán recopilar las peticiones de los usuarios para acceder a su cuenta y a las partidas existentes. Además, se deberá externalizar el servidor de la red privada del hogar para que pueda ser accesible desde fuera de ésta.

1.3 Software utilizado

La plataforma en la que se ha desarrollado la aplicación ha sido el IDE Android Studio. Este software es el oficial de Google y está en constante actualización, a diferencia de Eclipse, el entorno anterior de desarrollo de Android, que Google anunció su renuncia a éste y la migración de sus proyectos a Android Studio en 2015 [2].

El sistema operativo es Windows 10, que es compatible con Android Studio y con los otros programas utilizados en el proyecto. Los componentes del ordenador de desarrollo permiten simular varios dispositivos virtuales simultáneamente, lo cual ofrece una mayor de corrección de errores de diseño y de conexión.

El servidor se hará en XAMPP y el gestor de bases de datos será HeidiSQL, explicados ambos en el apartado 3.1. Por último, para la redacción de los ficheros PHP, se ha elegido Sublime como editor de texto plano, ya que proporciona ayudas al usuario que el editor de texto estándar no hace, como por ejemplo el bloc de notas.

2. Metodología

En cuanto a la realización del proyecto, es importante tener una estructura clara de los pasos a seguir para el correcto funcionamiento de éste, lo que se denomina el ciclo de vida de desarrollo software, que es la secuencia estructurada y definida de las etapas para desarrollar un producto software deseado.

Este ciclo de vida se divide en dos bloques que en última estancia se juntan. Por un lado, está el servidor de bases de datos y por el otro la aplicación. Estas dos se pueden desarrollar de forma independiente hasta cierta etapa de desarrollo, que será cuando se haga el código Java para la comunicación online.

Teniendo en cuenta eso, primero se desarrolló el diseño de la aplicación. Este paso consta de realizar la estructura de actividades que tendrá la aplicación, los elementos de cada actividad, ya sean imágenes, botones, campos de texto y otros tipos de componentes. También cuenta en este paso el desarrollar mínimamente el código de cada clase de Java para que las clases estén interconectadas correctamente. Por último, se tuvo en cuenta el emular la aplicación en distintos dispositivos virtuales y físicos para una correcta estética de ésta sin fallos en el texto o imágenes.

El siguiente paso fue el realizar desde cero el motor de juego de la aplicación. Este paso es el más complejo de la aplicación, ya que contiene la mayor concentración de algoritmos de todo el programa. También es el que más tiempo ha llevado para ser completado. Este motor de juego valida los movimientos del usuario e indica a qué casillas puede mover determinada pieza, además de gestionar los turnos, el tiempo de juego y determinar si la partida ha finalizado o no.

Una vez finalizado el desarrollo de la aplicación sin conexión, se creó la base de datos en el servidor de juego y se redactaron los ficheros PHP que se encargan de recibir las peticiones de los datos del cliente y hacer las consultas a la base de datos, además de generar una respuesta a cada petición. También se externalizó la dirección IP del servidor y se le asignó un nombre de dominio para poder acceder a él desde fuera de la red privada en la que se encuentra.

Por último, se amplió el código de las clases Java de la aplicación que requerían conexión a Internet, y se modificó el código del motor de juego para adecuarlo a la jugabilidad online, lo que conlleva el enviar y recibir continuamente el estado de la partida al servidor de juego.

En el siguiente diagrama se puede ver con una mayor claridad la dedicación temporal que ha supuesto cada tarea del proyecto, incluyendo la redacción de este documento.

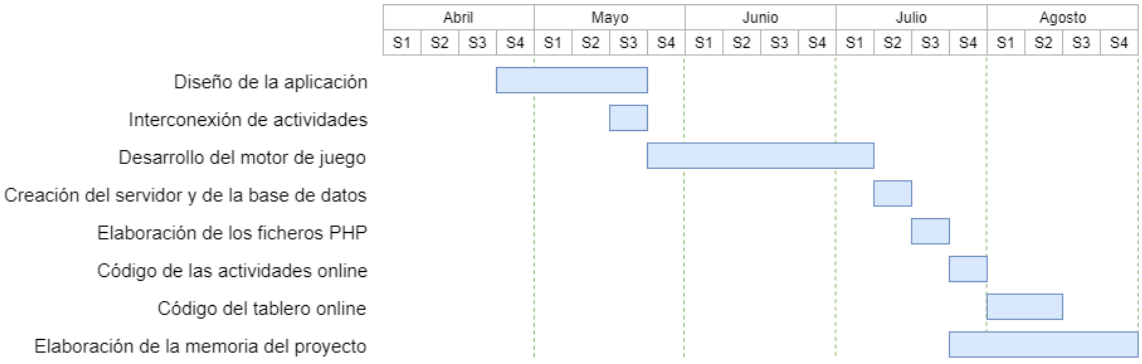

Diagrama 1. Diagrama de Gantt.

3. Creación del servidor

Uno de los dos componentes de este proyecto es el servidor de juego. Éste se encargará de almacenar toda la información de los usuarios y de las partidas existentes, además de conectarse con los usuarios que quieren acceder a su cuenta y al contenido online.

El esquema general del funcionamiento del servidor es el del Diagrama 2. Está estructurado en cómo el usuario accede a través de una dirección de dominio al servidor que corresponderá con la dirección IP pública que tenga éste en ese momento. La dirección será *chessgame-app.duckdns.org*, y se accederá a través del puerto 80 del router, como en cualquier servidor web. La dirección privada será la que tendrá el servidor asignada de forma estática dentro de la red local en la que se encuentra.

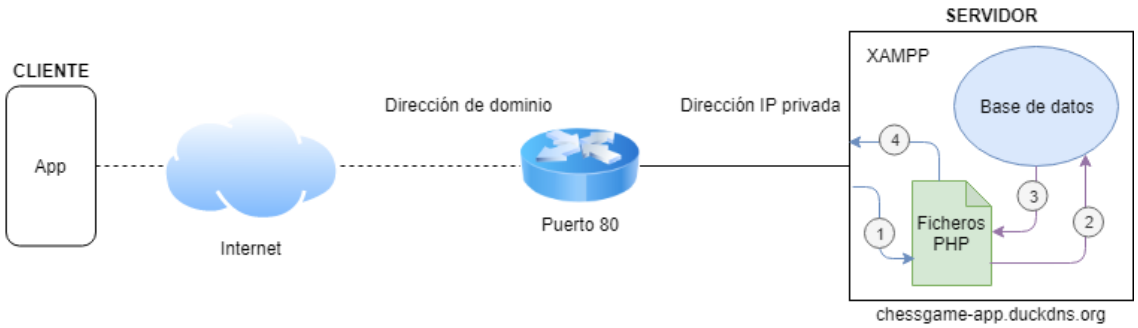

Diagrama 2. Estructura simplificada de la comunicación entre el cliente y el servidor.

Dentro del servidor se encuentran los dos elementos clave de este, que son la base de datos y los ficheros PHP. Primero, el usuario se conecta a una URL, en la cual viene indicado el fichero al que desea acceder. Esta conexión viene acompañada de uno o varios datos, dependiendo del tipo de consulta que haga. El fichero correspondiente se encarga de procesar estos datos y realiza consultas a la base de datos conectándose a través del puerto 3306. La base de datos contesta al fichero y éste procesa la respuesta, ya sea con un registro o con un mensaje de confirmación. Por último, el fichero procesa la respuesta y envía un mensaje de confirmación o de error al cliente.

3.1 Base de datos

La base de datos es uno de los principales componentes del proyecto y el eje de toda la funcionalidad online del mismo.

Una base de datos es un conjunto de información que está organizada de forma que resulte fácil de buscar y actualizar, la cual se aloja en un servidor de bases de datos, dando acceso a la misma.

Los servidores de bases de datos administran y organizan la base de datos tanto a nivel físico como a nivel lógico. A nivel físico, los servidores de bases de datos almacenan la información en ubicaciones específicas dentro del árbol de archivos y directorios del sistema de disco que utilizan. A nivel lógico, cada servidor organiza la información en función del tipo de base de datos de que se trate.

En nuestro caso, se empleará una base de datos relacional, que está organizada en tablas compuestas por filas y columnas, donde las filas son los registros y las columnas son los campos de cada registro.

Uno de los campos de una tabla, o una agrupación de campos, deben ser elegidos como clave. Las claves se emplean para localizar filas en una tabla e identificarlas inequívocamente. El valor del campo o campos elegidos como clave nunca podrá repetirse en el conjunto de registros de la tabla.

En las bases de datos se pueden establecer relaciones entre tablas. Esto se consigue relacionando un campo de cada tabla. Para ello, los dos campos deben tener obligatoriamente el mismo tipo de

valor. Estas relaciones permiten organizar los datos de una manera más estructurada sin perder información y reducir el tamaño de los datos que se almacenan físicamente en el sistema de disco. Pueden ser de tres tipos: de uno a varios (1:n), de uno a uno (1:1) y de varios a varios (n:n).

En este proyecto, solamente será necesario el uso de relaciones 1:n. Estas determinan una relación entre dos tablas, de manera que a cada ocurrencia de la primera entidad le pueden corresponder varias ocurrencias de la segunda, y a cada ocurrencia de la segunda no más de una de la primera.

A partir de las tablas, las relaciones y las consultas, se puede obtener información en formato no relacional. Las consultas son operaciones de lectura que implican la combinación de información ubicada en diferentes tablas a partir de las relaciones existentes entre ellas. El lenguaje SQL permite definir estas consultas, y otras operaciones, mediante una sintaxis estándar. La sintaxis de las consultas necesarias en este proyecto será explicada posteriormente, en el apartado 3.2.2.

3.1.1 Diseño de la Base de Datos

A la hora de representar la estructura de tablas que conformarán la base de datos, se pondrá en dos columnas por tabla. En la columna de la izquierda aparecerá el nombre del campo, y en la de la derecha el tipo de valor de este. Dicha estructura es la siguiente:

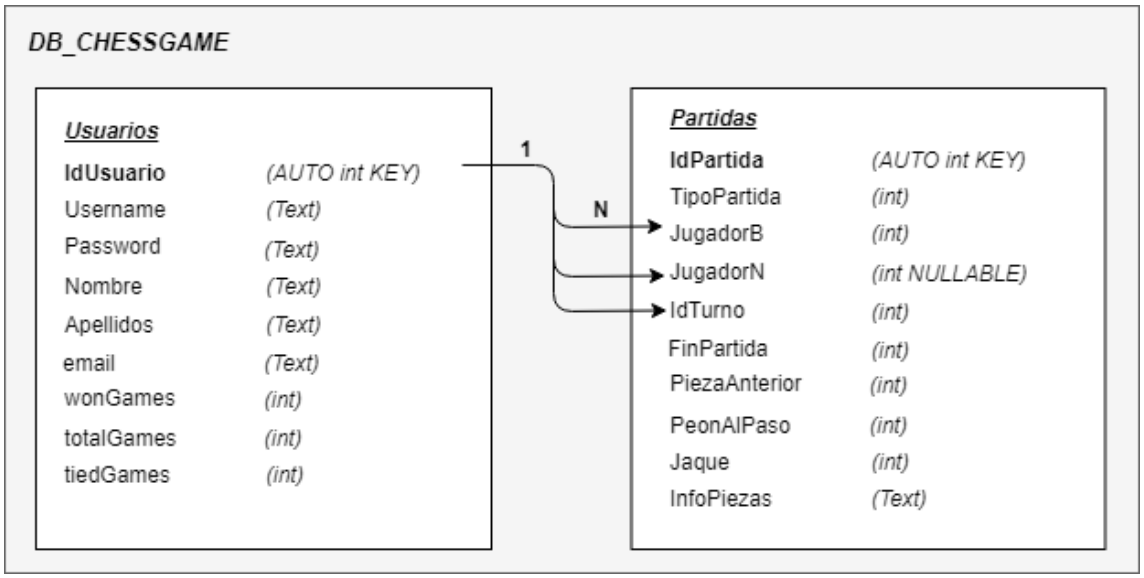

Figura 1. Estructura de la Base de datos.

Tal y como se puede apreciar, sólo será necesario emplear dos tablas. En la primera, se almacenará todo lo necesario de cada usuario. En la segunda, se hará lo mismo con las partidas del juego, pero con diferencias notables.

En la tabla de usuarios, el campo clave será *IdUsuario*, un identificativo generado automáticamente por el gestor de bases de datos. Este número, tal y como se ha explicado anteriormente, no se repetirá en ningún otro registro. Se elige este campo como clave en lugar del *Username* porque, a pesar de que este tampoco se puede repetir, es un campo de texto, y por ello es menos recomendable asignarlo como clave. Lo óptimo es que el campo clave sea un valor que tenga el menor tamaño posible, y un número entero tiene un tamaño menor que un texto en casi todos los casos.

El campo *Username* será el nombre de usuario, que se usará para hacer el inicio de sesión y aparecerá cuando juegue con otros jugadores. Al ser un campo de texto, se almacenará en un tipo de campo TEXT. Los campos Nombre y Apellidos también se almacenarán en este tipo de campo. Estos dos campos, al igual que los campos de información de partidas, sólo aparecerán en el perfil a modo de información personal.

El *Password* no es exactamente la contraseña que el usuario introduce al iniciar sesión. Esto se debe a que en la base de datos se almacenará únicamente la contraseña cifrada mediante el algoritmo SHA-256 de la contraseña introducida, para que ni el propio administrador de la base de datos pueda conocer las contraseñas de los usuarios y acceder con sus cuentas a la plataforma. El método de Java que genera este cifrado es explicado en el apartado 4.2.2.

En cuanto a la tabla de partidas, el campo clave será *IdPartida*, un entero que, al igual que *IdUsuario* en la tabla de Usuarios, será el identificativo de cada partida. Los campos de *JugadorB* y *JugadorN* son los integrantes de la partida. Estos están relacionados con el campo *IdUsuario* de la tabla usuarios, lo que conlleva que el tipo de campo debe de ser el mismo. Lo mismo sucede con el campo *IdTurno*. Este planteamiento de hacer el campo clave el ID de la partida y no los campos de jugadores ofrece la posibilidad de simplificar considerablemente las consultas a la base de datos, ya que sólo habrá que buscar por un campo y no por dos.

El campo *TipoPartida* indica la modalidad de juego de dicha partida. Éste será un entero que irá desde 1 hasta 3. Se indica con un entero el lugar de con un texto para optimizar el tamaño que ocupa cada registro.

FinPartida es un campo que en principio está vacío, y almacenará, al final de la partida, el ID del jugador que haya ganado. *IdTurno* almacenará al usuario que tenga que mover ficha. Los campos *PiezaAnterior*, *PeonAlPaso* y Jaque son campos de control que sirven para poder hacer la captura de peón al paso y los jaques.

Por último, el contenido del campo *InfoPartida* de la tabla Partidas es un objeto JSON en formato de texto con la información de la ubicación de las piezas en el tablero. El contenido inicial de este es el siguiente:

{"blancas":{"rey":"60","reina":"59","torres":"63#56","caballos":"62#57","alfiles":"61#58","peones":"48#49#50#51#52#53#54#55"},"negras":{"rey":"4","reina":"3","torres":"0#7","caballos":"1#6","alfiles":"2#5","peones":"8#9#10#11#12#13#14#15"}}

Como la cadena de texto resultante puede resultar confusa de leer, existe una herramienta que, al escribir dicha cadena de texto, te devuelve un esquema organizado del objeto o array JSON. Dicha herramienta se encuentra en el enlace http://jsonviewer.stack.hu/. El esquema resultante es el siguiente:

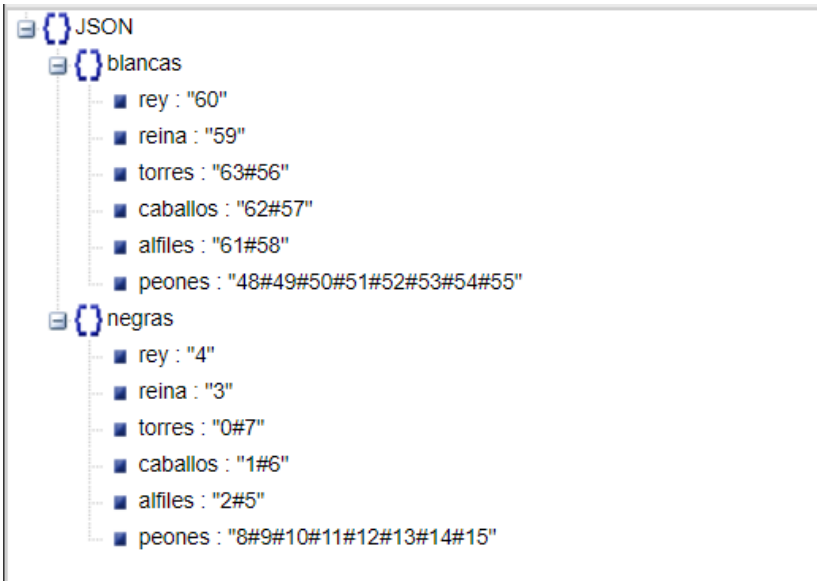

Figura 2. Esquema del objeto JSON almacenado en la Base de Datos.

La estructura del JSON consta de dos objetos con los mismos campos, pero valores distintos: “blancas” y “negras”. Estos hacen referencia a la posición de cada ficha dependiendo de si es

blanca o negra. Estos campos serán cadenas de texto formadas por números y el carácter "#". En la Figura 2 tienen valores asignados, que corresponden a la posición inicial de las piezas, que posteriormente irán variando a lo largo del transcurso de la partida.

Con esta distribución del objeto, se consigue una mayor organización y un menor número de columnas y de registros en la base de datos, así como la necesidad de contar con nuevas tablas. La forma de acceder al contenido de este objeto JSON se describirá en la clase Java correspondiente, donde también está explicada toda la forma de procesar estos datos para llevar a cabo el buen funcionamiento del juego.

3.1.2 Creación de la Base de Datos.

La base de datos deberá alojarse en el servidor. Para ello, emplearemos XAMPP, un entorno que, entre otras cosas, contiene la funcionalidad de iniciar un servidor MySQL, imprescindible para el acceso a dicha base de datos. Contiene también los módulos de Mercury, Tomcat y FileZila, pero no serán necesarios en este proyecto.

Para poder operar con el software de gestión de bases de datos, deberemos iniciar el servidor. En la Figura 3 se ve cómo está iniciado, y el puerto en el que será necesario conectarse para hacer las consultas pertinentes, que por defecto es el 3306.

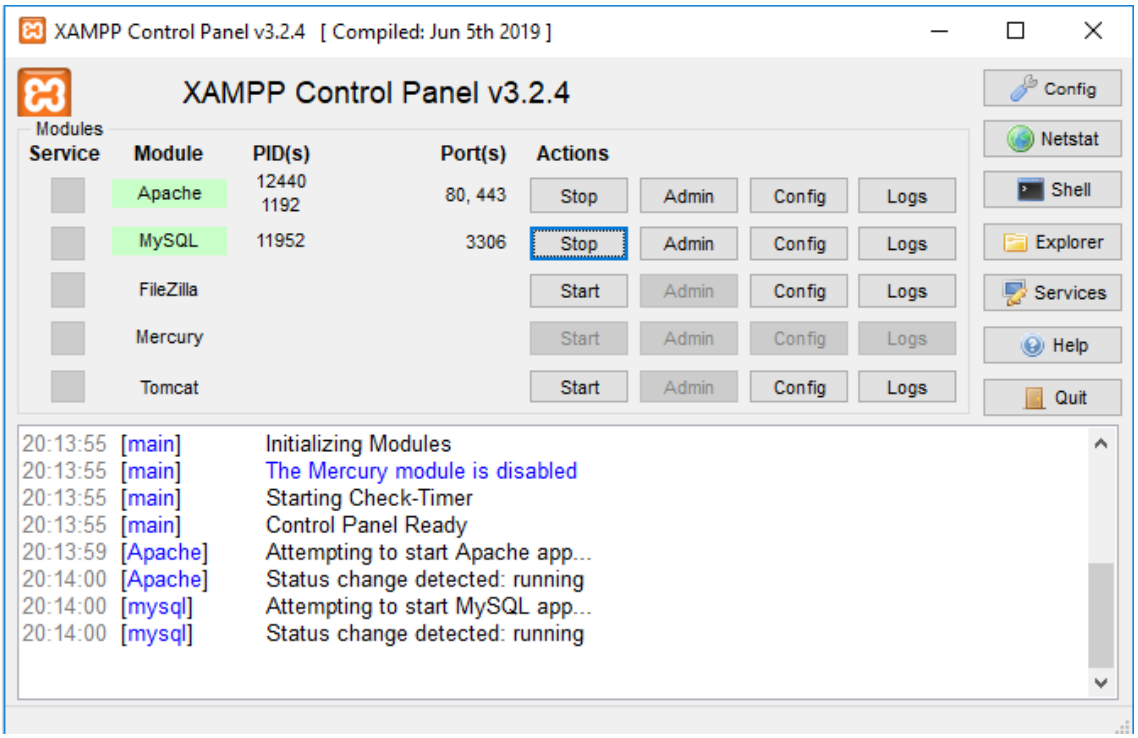

Figura 3. Panel de control de XAMPP

El siguiente paso será crear la base de datos. Se empleará el gestor de bases de datos HeidiSQL para hacer las capturas, pero se puede realizar con cualquier otro gestor, como por ejemplo PhpMyAdmin o MySQL. Una vez instalado y abierto, el programa muestra una ventana para la administración de sesiones del programa.

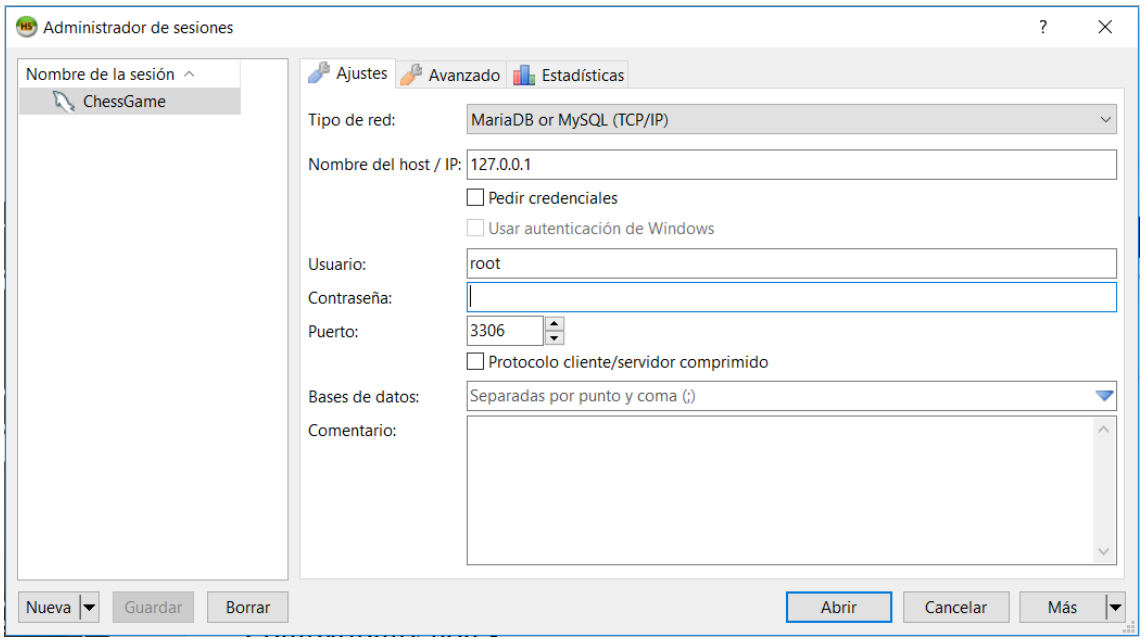

Figura 4. Administrador de sesiones de HeidiSQL.

En el tipo de red tendremos MySQL (TCP/IP). El nombre del host es la dirección del host local, ya que la base de datos estará alojada en el mismo ordenador en el que se está creando la base de datos. El usuario será el raíz (*root*), y no habrá contraseña. Por último, el puerto será el 3306, explicado anteriormente.

Finalizada la configuración anterior, ya se puede crear la base de datos, que en nuestro caso se llamará *db_chessgame*. Como se ha mencionado en la Figura 1, será necesario crear dos tablas: la de usuarios y la de partidas.

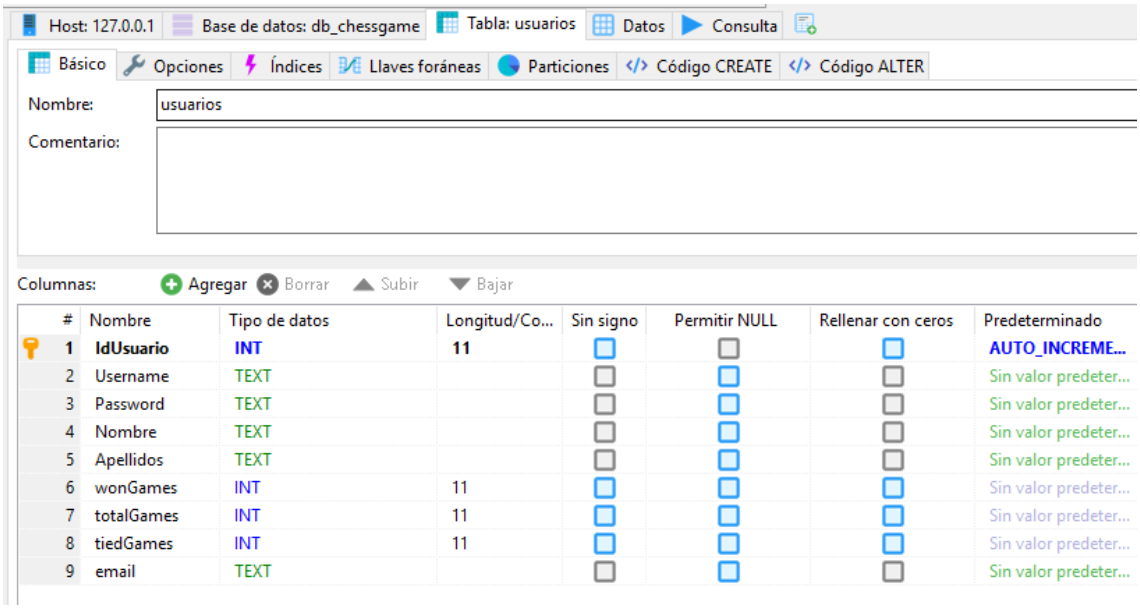

Figura 5. Tabla Usuarios.

En la tabla de usuarios, se tendrá que declarar el campo IdJugador como un INT y de valor predeterminado AUTO_INCREMENT. Este tipo de valor predeterminado exige que el campo sea clave y que sólo hay uno. Esa condición se cumple en esta tabla. El tipo de valor AUTO_INCREMENT puede ser configurado para que tenga un valor inicial o para elegir el incremento entre nuevos registros, pero en este caso no interesa asignar valores iniciales.

Para declarar el campo como llave tendremos que crear un índice en dicho campo del tipo PRIMARY KEY. El porqué de que este campo sea declarado clave y no el Username, a pesar de que este no se puede repetir, está explicado en el apartado 3.1.1. En el resto de los campos de la tabla no hay nada nuevo a destacar, tan sólo crear las columnas con los nombres y los tipos de campos que se ven en la Figura 4.

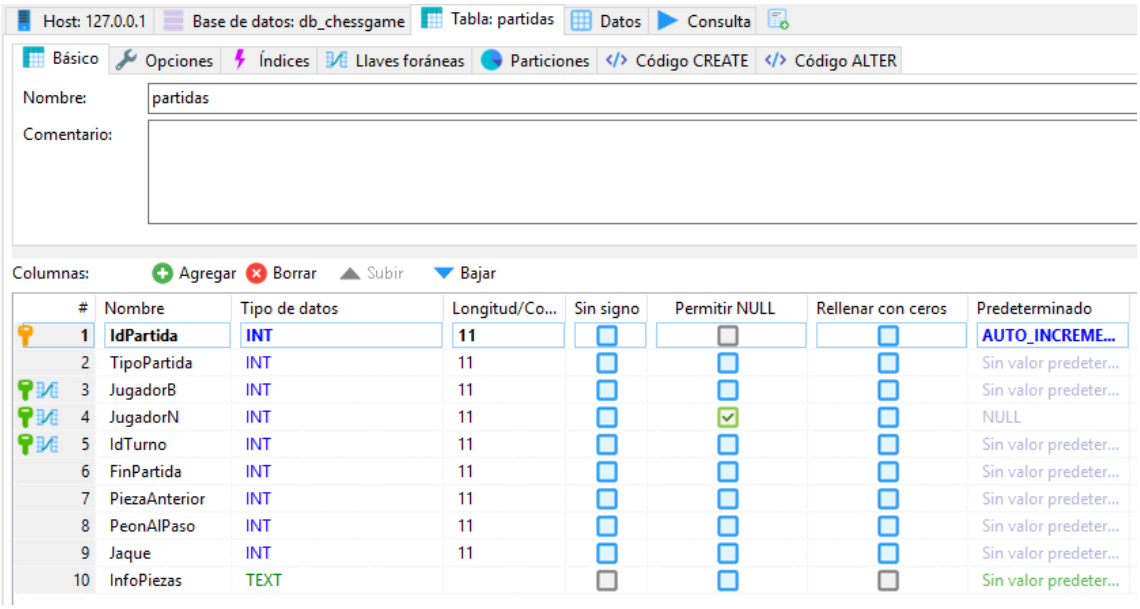

Figura 6. Tabla Partidas.

La creación de la tabla de las partidas es similar a la anterior, con la diferencia de que en esta tabla el campo JugadorN puede tener un valor nulo, lo cual habrá que indicarlo seleccionando la casilla "Permitir NULL" de la columna correspondiente.

Ahora queda añadir las llaves foráneas, que son las que relacionan las dos tablas. Siguiendo el esquema de la Figura 1, la tabla de usuarios es la que hace de 1 y la de partidas es la que hace de N, según se ha explicado en el apartado 3.1. Las llaves foráneas deben crearse en la tabla que hace de N, por lo que se indicarán en la tabla de partidas.

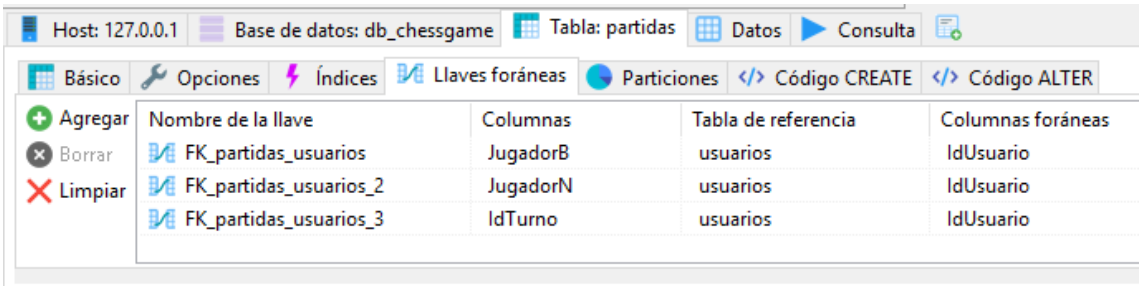

Figura 7. Llaves foráneas de la base de datos.

A la hora de crearlas tan solo hace falta indicar el campo que quieres relacionar, la tabla de referencia y la columna de dicha tabla. En este caso, habrá que crear 3 llaves, relacionando los campos JugadorB, JugadorN e IdTurno con la columna de IdUsuario.

3.2 Acceso al servidor

3.2.1 Acceso remoto mediante DDNS

Hasta ahora, el servidor creado en el anterior apartado sólo tenía accesibilidad desde la red local. Eso conlleva que no se puede acceder a éste desde fuera de dicha red. Para solucionar este problema, es necesario abrir el puerto 80 del router y que excluyan la dirección IP del servidor del CG-NAT (Carrier-Grade NAT). Esto lo gestiona el operador de Internet que se tenga contratado llamando al soporte técnico.

Una vez realizado este paso, ya se puede acceder al servidor desde el exterior introduciendo la dirección IP pública. No obstante, esta dirección es dinámica, con lo que varía a lo largo del tiempo de forma automática. Por eso no se puede poner esta dirección en las consultas al servidor en el código del programa, ya que sólo valdrán mientras esta dirección coincida con la del servidor.

La mejor forma de solucionar este problema es con un DNS dinámico (Dynamic Domain Name System). El DDNS es un servicio que permite la actualización en tiempo real de la información sobre nombres de dominio situada en un servidor de nombres. En este proyecto se ha utilizado Duck DNS, un gestor DDNS gratuito sencillo de utilizar.

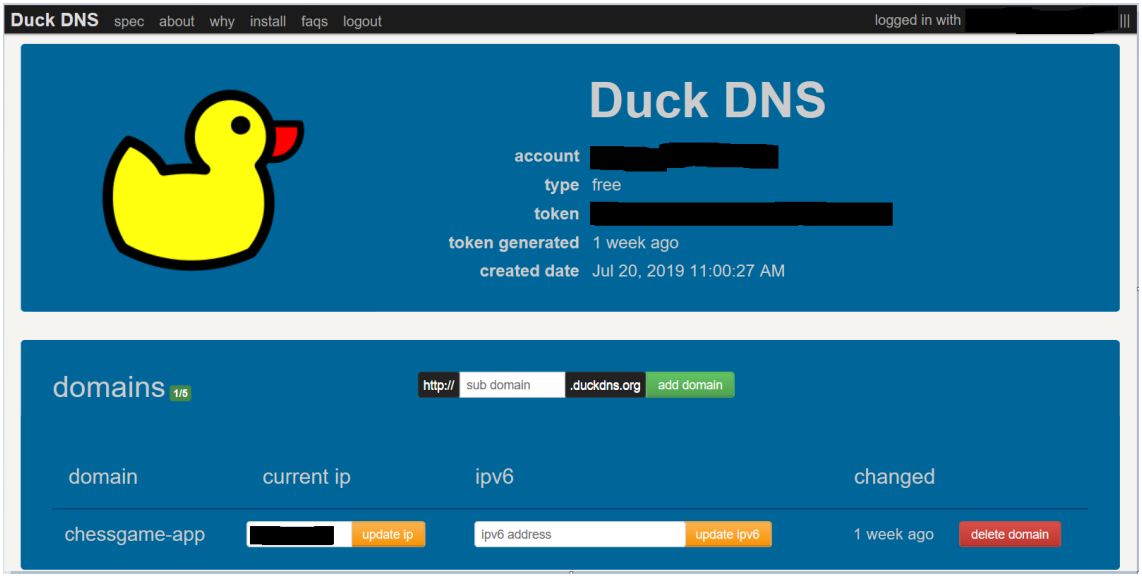

Figura 8. Datos del dominio creado en Duck DNS.

El nombre de dominio resultante para acceder a los ficheros del servidor será http://chessgame-app.duckdns.org. Si introducimos esa URL en el navegador, se podrá ver la página de inicio de XAMPP, lo cual indica que se produce conexión entre el cliente y el servidor de juego.

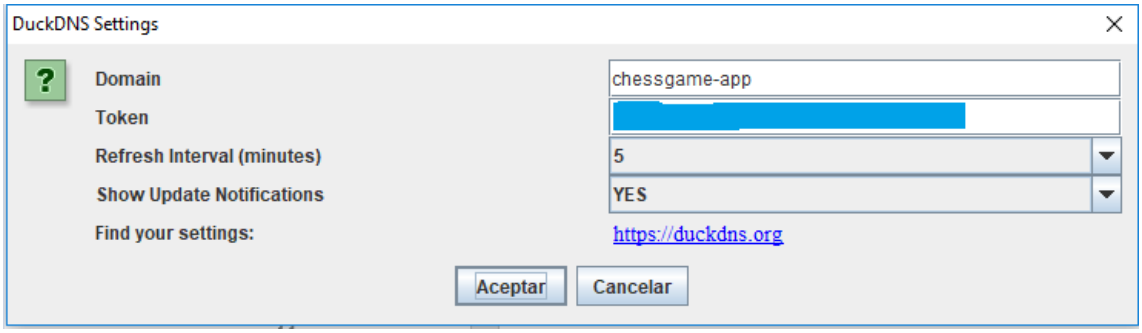

Figura 9. Datos del dominio creado en Duck DNS.

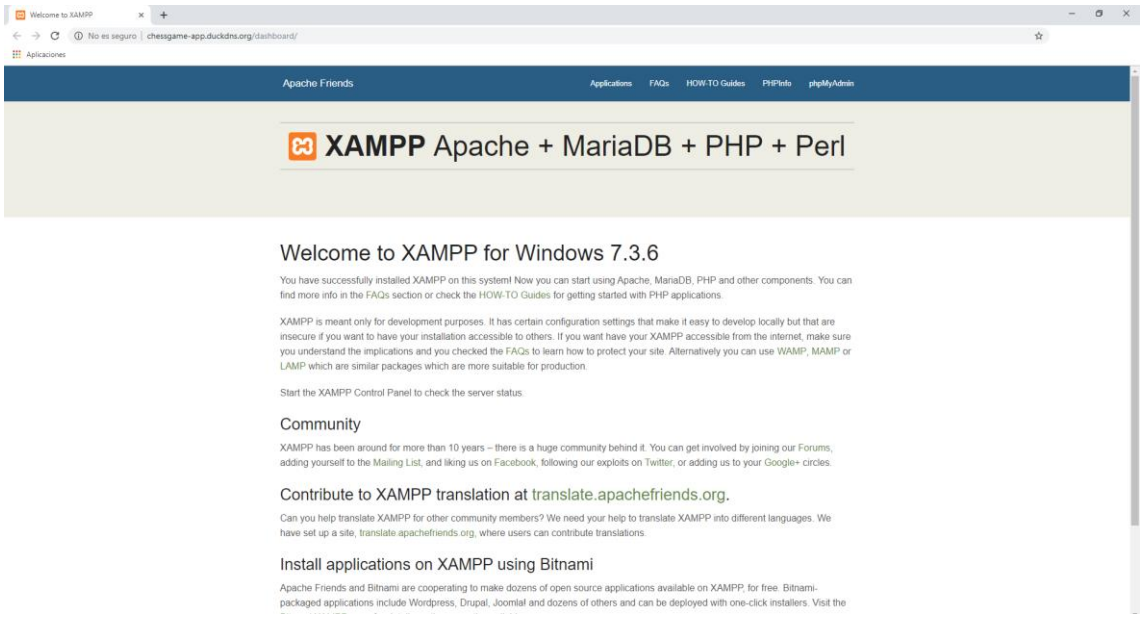

Figura 10. Página de inicio de XAMPP.

3.2.2 Ficheros PHP

El lenguaje PHP, acrónimo en inglés de preprocesador de hipertexto (Hypertext Preprocessor), es un lenguaje de programación del lado del servidor que ofrece una salida de texto con codificación UTF-8 compatible con documentos HTML. A pesar de que en este proyecto no se empleará HTML, ya que no se hará en una página web, sigue siendo útil para el envío de las respuestas del servidor en formato de texto plano.

Las consultas que se harán a la base de datos se realizarán en ficheros PHP, donde el usuario desde la aplicación tendrá que enviar ciertos datos en un formulario, los cuales serán obtenidos por los ficheros y podrán realizar las consultas SQL correspondientes. Estos ficheros se alojarán dentro de la carpeta *htdocs* de XAMPP. Concretamente, dentro de una carpeta previamente creada llamada *chess_server* dentro de htdocs.

Las funcionalidades online que debe tener la aplicación son las de inicio de sesión, registro, crear partida, buscar partida, obtener la información de las partidas, y finalizar la partida. Se llevarán a cabo en distintos ficheros a los que la aplicación accederá mediante peticiones de tipo *post* para que los valores introducidos no aparezcan en la URL.

En todos los ficheros se establecen 4 variables con el nombre de la base de datos y los datos necesarios para que puedan conectarse con la base de datos (usuario, contraseña y dirección IP). Después, se crea un objeto de la clase *mysqli* con los 4 parámetros nombrados previamente. Si se produce un error, responde al cliente con dicho error. El procesado de las respuestas de los ficheros se explicará en apartado 3.5.

El primero a analizar será el fichero que hace la función de iniciar sesión. Se crea una consulta SQL que obtiene todos los registros del nombre de usuario enviado al fichero. Si no hay ningún registro, responde diciendo que no hay un usuario con ese nombre. En caso contrario, obtiene el valor del campo contraseña. Si la contraseña es errónea, el servidor lo informa al cliente. Si los datos introducidos son correctos, el fichero devuelve un OK seguido de un objeto JSON con los datos relevantes del usuario para que pueda ver su perfil. De esta forma, no será necesario crear otro fichero para que el usuario pueda ver su perfil.

```php
<?php

    $host_db = "localhost";
    $user_db = "root";
    $pass_db = "";
    $db_name = "db_chessgame";

    $conexion = new mysqli($host_db, $user_db, $pass_db, $db_name);

    if ($conexion->connect_error) {
        die("La conexion falló: " . $conexion->connect_error);
    }

    $consulta = "SELECT * FROM usuarios WHERE Username = '" . $_POST['Username'] . "'";
    $result = mysqli_query( $conexion, $consulta ) or die (mysqli_error());

    if (mysqli_num_rows($result) == 1) {

        $row = mysqli_fetch_array($result);

        if($row["Password"] == $_POST['Password']){
            $product = array();
            $product["IdUsuario"] = $row["IdUsuario"];
            $product["Username"] = $row["Username"];
            $product["Nombre"] = $row["Nombre"];
            $product["Apellidos"] = $row["Apellidos"];
            $product["wonGames"] = $row["wonGames"];
            $product["totalGames"] = $row["totalGames"];
            $product["tiedGames"] = $row["tiedGames"];

            echo "OK##" . json_encode($product);
        } else {
            echo "Wrong Password";
        }

    } else {
        echo "No users found";
    }
    mysqli_close( $conexion );
?>
```

Figura 11. Fichero login.php

El segundo fichero es el de registro de usuarios en la aplicación. En este caso, se comprueba si el nombre de usuario introducido por el usuario existe ya o no en la base de datos. Si no existe, se inserta en la base de datos un nuevo registro mediante una consulta de tipo INSERT INTO. En este tipo de consulta se indica primero la tabla en la que se quiere introducir el nuevo registro, luego los nombres de los campos de la tabla y por último los valores. Estos valores son los introducidos por el usuario, a excepción del ID de usuario, que eso lo gestiona la propia base de datos, y la información de las partidas jugadas, que son todos a cero.

```php
<?php

    $host_db = "localhost";
    $user_db = "root";
    $pass_db = "";
    $db_name = "db_chessgame";

    $conexion = new mysqli($host_db, $user_db, $pass_db, $db_name);

    if ($conexion->connect_error) {
        die("La conexion falló: " . $conexion->connect_error);
    }

    $consulta = "SELECT * FROM usuarios WHERE username = '". $_POST['username'] ."'";

    $result = $conexion->query($consulta);

    $count = mysqli_num_rows($result);

    if ($count >= 1) {
        echo "El nombre de Usuario ya existe";
    } else{

        $query = "INSERT INTO usuarios(IdUsuario, Username, Password, Nombre, Apellidos, wonGames,
            totalGames, tiedGames, email) VALUES (null,'". $_POST['username'] ."','". $_POST['password']
            ."','". $_POST['name'] ."','". $_POST['surname'] ."',0,0,0,'". $_POST['email'] ."')";

        if ($conexion->query($query) === TRUE) {
            $consulta2 = "SELECT IdUsuario FROM usuarios WHERE username = '". $_POST['username'] ."'";
            $result = mysqli_query( $conexion, $consulta2 ) or die (mysqli_error());

            $row = mysqli_fetch_array($result);
            echo "OK#" . $row["IdUsuario"];
        }

        else {
            echo "Error al crear el usuario";
        }
    }
    mysqli_close($conexion);
?>
```

Figura 12. Fichero register.php

El siguiente fichero es el de crear partida, y está formado por dos partes, que se ejecutará una u otra dependiendo de si hay una partida creada con un jugador a la espera de entrar o no. Primero, se obtienen los datos del ID del jugador y del modo de juego seleccionado y se crea una variable con el texto del objeto JSON que describe las posiciones de las piezas, explicado en el apartado 3.1.1.

El servidor buscará una partida de esa modalidad donde solo haya un jugador en ella. Para ello, se analizará que el campo *JugadorN* sea nulo. Esto se ha hecho así ya que el color de las piezas que jugará cada jugador será aleatorio, es decir, que el jugador no lo escogerá. De esta forma, se equilibran los emparejamientos y le da un factor de aleatoriedad al juego.

Si hay algún un registro, se rellena el campo de *JugadorN* con la ID del jugador que ha buscado la partida mediante un UPDATE. En tal caso, el jugador blanco será el otro jugador, lo cual se obtendrá su ID para obtener el nombre de usuario de ese jugador para poder representarlo en el tablero. Finalmente, el servidor informa de que se ha creado la partida con éxito.

```php
<?php

    $host_db = "localhost";
    $user_db = "root";
    $pass_db = "";
    $db_name = "db_chessgame";

    $tipoPartida = $_POST['TipoPartida'];
    $idJugador = $_POST['IdJugador'];

    $jsonInic = '{"blancas":{"rey":"60","reina":"59","torres":"63#56","caballos":"62#57","alfiles":"61#58
    ","peones":"48#49#50#51#52#53#54#55"},"negras":{"rey":"4","reina":"3","torres":"0#7","caballos":"1#6"
    ,"alfiles":"2#5","peones":"8#9#10#11#12#13#14#15"}}';

    $conexion = new mysqli($host_db, $user_db, $pass_db, $db_name);

    if ($conexion->connect_error) {
        die("La conexion falló: " . $conexion->connect_error);
    }

    $consulta = "SELECT * FROM partidas WHERE JugadorN IS NULL AND TipoPartida = $tipoPartida AND
        JugadorB <> $idJugador";

    $result = $conexion->query($consulta);

    $count = mysqli_num_rows($result);

    if ($count > 0) {
        $row = mysqli_fetch_array($result);
        $idPartida = $row["IdPartida"];

        $query = "UPDATE partidas SET JugadorN = $idJugador WHERE IdPartida = " . $row["IdPartida"];

        if ($conexion->query($query) === TRUE) {
            $idOtro = $row["JugadorB"];

            $consulta2 = "SELECT Username FROM usuarios WHERE IdUsuario = $idOtro";

            $result2 = $conexion->query($consulta2);

            $row2 = mysqli_fetch_array($result2);
            $nickOtro = $row2["Username"];

            echo "Creada#" . $idPartida . "#" . $nickOtro;
        }
        else {
            echo "Error al crear partida";
        }

```

Figura 13. Primera parte del fichero crear_partida.php

En caso de no haber ningún registro, se crea uno nuevo mediante un INSERT INTO. Se rellenan todos los campos a excepción del ID de la partida y el JugadorN, que se rellenará con el ID del próximo jugador que busque partida. El servidor responde con un mensaje indicando que está a la espera de otro jugador junto con el ID de la partida que se acaba de crear.

```php
    } else{
        $query = "INSERT INTO partidas (IdPartida, TipoPartida, JugadorB, JugadorN, IdTurno, FinPartida,
            PiezaAnterior, PeonAlPaso, Jaque, PosReyB, PosReyN, InfoPiezas) VALUES (null,$tipoPartida,$
            idJugador,null,$idJugador,0,0,0,0,60,4,'$jsonInic')";

        $result = mysqli_query($conexion, $query) or die (mysqli_error());

        $consulta2 = "SELECT IdPartida FROM partidas WHERE JugadorB = $idJugador AND TipoPartida = $
            tipoPartida";
            $result2 = mysqli_query( $conexion, $consulta2 ) or die (mysqli_error());

            $row = mysqli_fetch_array($result2);
            echo "Esperando#" . $row["IdPartida"];

    }
    mysqli_close($conexion);
?>
```

Figura 14. Segunda parte del fichero crear_partida.php

Si se da el caso de que el jugador está esperando un jugador, será necesario un fichero adicional para comprobar cuándo se ha añadido el otro jugador en el registro de esa partida. Para ello, la

consulta que hará este fichero a la base de datos será buscar el registro donde el *JugadorN* sea nulo y que el campo *IdPartida* sea el mismo que el que se ha creado anteriormente. A continuación, se calcula el número de registros que hay con esas condiciones. Si hay un registro, es que aún no se ha introducido un segundo jugador en la partida, con lo cual el servidor responde al cliente con el mismo mensaje que en el fichero anterior. En cambio, si no hay ningún registro, significará que ya hay un segundo jugador, lo que conlleva a realizar una segunda consulta de tipo SELECT con un INNER JOIN entre las dos tablas para obtener el nombre del usuario cuyo ID ocupa el lugar de *JugadorN*. Finalmente, el fichero responde con un mensaje igual al anterior fichero en el caso de haberse creado la partida.

```php
<?php

    $host_db = "localhost";
    $user_db = "root";
    $pass_db = "";
    $db_name = "db_chessgame";

    $idPartida = $_POST['IdPartida'];

    $conexion = new mysqli($host_db, $user_db, $pass_db, $db_name);

    if ($conexion->connect_error) {
        die("La conexion falló: " . $conexion->connect_error);
    }

    $consulta = "SELECT * FROM partidas WHERE JugadorN IS NULL AND IdPartida = $idPartida";

    $result = mysqli_query( $conexion, $consulta ) or die (mysqli_error());

    $count = mysqli_num_rows($result);

    if ($count == 0) {

        $consulta2 = "SELECT usuarios.Username FROM usuarios INNER JOIN partidas ON usuarios.IdUsuario =
            partidas.JugadorN WHERE partidas.IdPartida = $idPartida";

        $result2 = mysqli_query( $conexion, $consulta2 ) or die (mysqli_error());

        $row = mysqli_fetch_array($result2);

        $nickOtro = $row["Username"];

        echo "Creada#$idPartida#$nickOtro";
    }
    else {
        echo "Esperando#$idPartida";
    }

    mysqli_close($conexion);
?>
```

Figura 15. Fichero buscar_jugador.php

Una vez creada y empezada la partida, será necesario obtener información de ésta cada cierto tiempo. Para ello será necesario un fichero que devuelva la información relevante de la partida. En estos datos se incluyen el jugador blanco y el jugador negro. Estos dos datos solo serán útiles al principio de la partida, para que el usuario sepa de qué color son sus fichas. El resto son el ID del jugador que le toca mover pieza, si ha finalizado la partida, si se ha producido jaque, si se puede realizar la captura de peón al paso y la posición de la pieza anterior. Por último, envía el JSON de la posición de todas las piezas de la partida.

El servidor envía al cliente un OK junto con la respuesta en formato JSON. Si la partida que ha buscado no existe, suceso que se puede dar cuando el otro jugador abandona, se lo comunica al cliente, lo cual significará que habrá ganado la partida.

```php
<?php

    $host_db = "localhost";
    $user_db = "root";
    $pass_db = "";
    $db_name = "db_chessgame";

    $conexion = new mysqli($host_db, $user_db, $pass_db, $db_name);

    if ($conexion->connect_error) {
        die("La conexion falló: " . $conexion->connect_error);
    }

    $consulta = "SELECT * FROM partidas WHERE IdPartida = ". $_POST['IdPartida'];

    $result = mysqli_query( $conexion, $consulta ) or die (mysqli_error());

    if (mysqli_num_rows($result) > 0) {
        $row = mysqli_fetch_array($result);

        $response = array();
        $response["JugadorB"] = $row["JugadorB"];
        $response["JugadorN"] = $row["JugadorN"];
        $response["IdTurno"] = $row["IdTurno"];
        $response["FinPartida"] = $row["FinPartida"];
        $response["PiezaAnterior"] = $row["PiezaAnterior"];
        $response["PeonAlPaso"] = $row["PeonAlPaso"];
        $response["Jaque"] = $row["Jaque"];
        $response["PosReyB"] = $row["PosReyB"];
        $response["PosReyN"] = $row["PosReyN"];
        $response["InfoPiezas"] = $row["InfoPiezas"];

        echo "OK##" . json_encode($response);
    }
    else{
        echo "PartidaNoExiste";
    }

    mysqli_close($conexion);
?>
```

Figura 16. Fichero info_partida.php

Cada vez que se realice un movimiento en la partida, el usuario deberá notificarlo al servidor para que se actualice en la base de datos y así poder informarlo al otro jugador. Esta función la realizará otro fichero, el cual obtendrá los datos relevantes de la partida por parte del jugador que ha realizado el movimiento. Como es una gran cantidad de datos, se almacenarán en variables para una mayor organización y simplicidad en la consulta correspondiente.

Primero buscará la partida por su ID, si no existe, lo notificará al usuario de la misma forma que el anterior fichero. En el caso de que la partida siga vigente, realizará una consulta de tipo UPDATE para actualizar ese registro con los datos recibidos anteriormente. Por último, el servidor informará del éxito en la actualización con un OK.

```php
<?php

    $host_db = "localhost";
    $user_db = "root";
    $pass_db = "";
    $db_name = "db_chessgame";

    $idPartida = $_POST['IdPartida'];
    $idOtro = $_POST['IdOtro'];
    $infoPartida = $_POST['InfoPartida'];
    $piezaAnterior = $_POST['PiezaAnterior'];
    $peonAlPaso = $_POST['PeonAlPaso'];
    $jaque = $_POST['Jaque'];
    $posReyB = $_POST['PosReyB'];
    $posReyN = $_POST['PosReyN'];

    $conexion = new mysqli($host_db, $user_db, $pass_db, $db_name);

    if ($conexion->connect_error) {
        die("La conexion falló: " . $conexion->connect_error);
    }

    $consulta = "SELECT * FROM partidas WHERE IdPartida = $idPartida";

    $result = mysqli_query( $conexion, $consulta ) or die (mysqli_error());

    if (mysqli_num_rows($result) > 0) {
        $query = "UPDATE partidas SET infopiezas = '$infoPartida', idturno = $idOtro, piezaAnterior = $
            piezaAnterior, peonAlPaso = $peonAlPaso, jaque = $jaque, posReyB = $posReyB, posReyN = $
            posReyN WHERE idpartida = $idPartida";

        if ($conexion->query($query) === TRUE) {
            echo "OK";
        }
        else {
            echo "Error";
        }
    }
    else{
        echo "PartidaNoExiste";
    }

    mysqli_close($conexion);
?>
```

Figura 17. Fichero actualizar_partida.php

Al finalizar la partida, el servidor borrará el registro de la base de datos para que no se acumulen infinitamente. Para ello, el jugador hará una petición a un fichero con el ID de la partida, el cual contiene una consulta de tipo DELETE, que borrará el registro de la condición dada. Además, se deberán actualizar las victorias y partidas de los usuarios que han intervenido en la partida.

En la Figura 18 se ve un fragmento del fichero que se encarga de esta acción. Consta principalmente de un *if-else* que comprueba si el que ha enviado el formulario al fichero ha ganado la partida, la ha empatado o la ha perdido, lo cual ejecutará una consulta de UPDATE u otra dependiendo del caso. Como este fichero será llamado por los dos jugadores, no hará falta actualizar a los dos usuarios en una misma consulta.

```php
    $idUsuario = $_POST['IdJugador'];
    $victoria = $_POST['Victoria'];
    $tablas = $_POST['Tablas'];

    if ($victoria == 1)  {
        $query = "UPDATE usuarios SET totalgames = totalgames + 1, wongames = wongames + 1 WHERE
            idUsuario = $idUsuario";

        if ($conexion->query($query) === TRUE) {
            echo "OK";
        }
        else {
            echo "Error";
        }
    }
```

Figura 18. Fragmento del fichero actualizar_usuarios.php

4. Desarrollo de la aplicación en Android Studio

Una vez creada la base de datos y el servidor, ya se puede empezar a desarrollar el proyecto en el software Android Studio. En este caso, se parte de que el software de desarrollo Android ya está instalado en el ordenador con sus librerías correspondientes, al igual que los anteriores programas utilizados.

Al abrir el programa, saldrá una ventana en la que se mostrarán por pantalla varias opciones. Al darle a *New Project*, aparecerá una ventana de creación en la que se tendrá que introducir los datos más relevantes del proyecto.

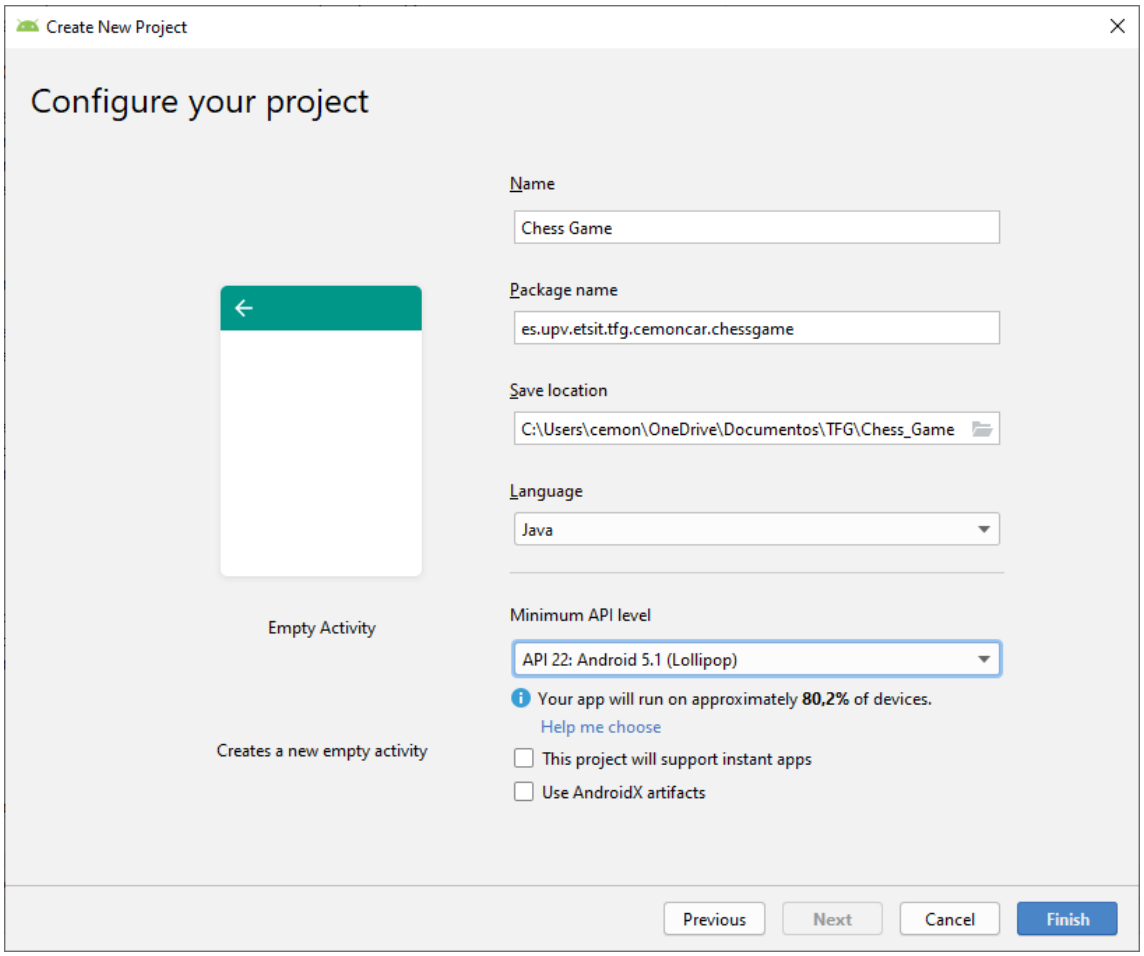

Figura 19. Ventana de creación del proyecto

El nombre del proyecto será el nombre de la aplicación. Este nombre se podrá cambiar posteriormente en el fichero strings.xml, que es donde se almacenan todas las cadenas de texto que se muestren en algún momento por pantalla.

El *Package* es el repositorio, dentro del repositorio seleccionado posteriormente, donde se guardará el proyecto. Dependiendo del proyecto que se desee hacer, no haría falta rellenar este campo, pero como éste será un proyecto que forma parte de un trabajo de fin de grado de la UPV, se puede formalizar rellenándolo en el orden inverso a la jerarquía que tendría este trabajo en Internet. El *package* resultante será *es.upv.etsit.tfg.cemoncar.chessgame*.

En el tipo de *Activity* se seleccionará *Empty Activity*, y de nombre el que aparece por defecto. Este campo por el momento no es relevante, ya que el proyecto contará con varios *Activities*, cada uno son un nombre descriptivo de su función. Todos ellos serán del tipo *empty*, es decir, que no contendrán nada y se tendrá que rellenar todo, que es lo que se desea en el apartado de diseño. Este *activity* inicial es el *main* de la aplicación, lo cual conlleva a que, al ejecutarla, empezará por

ésta, y con ello mostrará por pantalla el fichero XML que contiene. Esto se puede cambiar al *activity* que queramos en el apartado de Run de la barra superior del programa. Además, se deberá cambiar el fichero AndroidManifest.xml, y situar dentro del *activity* que se desee ejecutar en primer lugar los objetos de tipo Intent.

La API es la Interfaz de Programación de Aplicaciones, que es un conjunto de reglas en forma de código y especificaciones que las aplicaciones siguen para comunicarse entre ellas. Éstas sirven de interfaz entre programas diferentes de la misma manera en que la interfaz de usuario facilita la interacción entre la persona y el programa [3]. En este programa se usará como API mínima la 22, ya que hay funciones en el programa que se implementaron en esta API. Esto hará que la aplicación no pueda ser usada en todos los dispositivos hoy en día, pero sí en más de un 80% aproximadamente, tal y como se indica en la Figura 19.

Una vez creado el proyecto, hay que tener en cuenta la estructura que se desea seguir a lo largo del desarrollo de la aplicación. Esto conlleva a tener claro cuántas actividades, o *activities*, tendrá la aplicación y qué utilidad tendrá cada una. El esquema resultante de actividades de la aplicación y sus relaciones entre ellas es el del Diagrama 3. La explicación de cada una y el cómo se conectan entre ellas se explica a lo largo de este apartado.

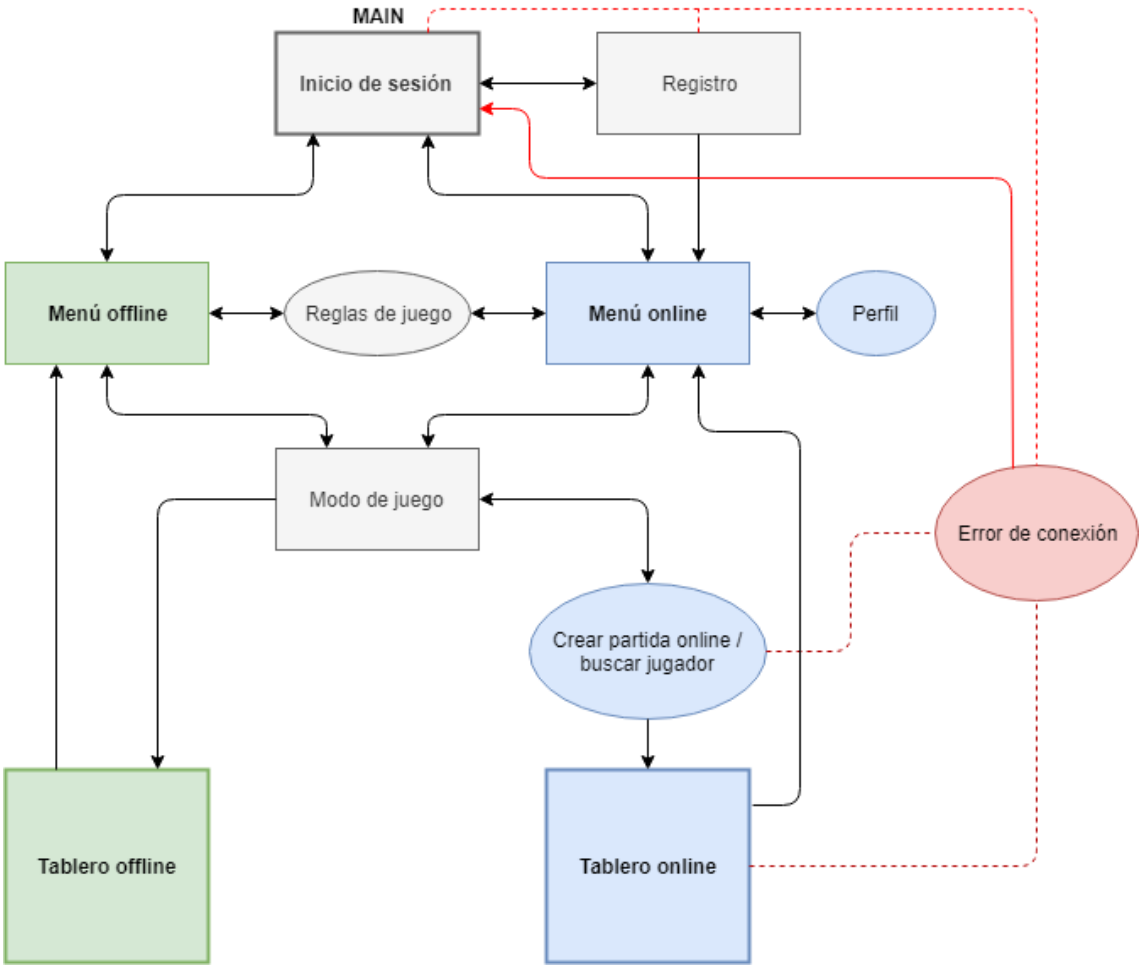

Diagrama 3. Esquema de actividades del proyecto

La aplicación comenzará con el inicio de sesión, donde el usuario podrá conectarse, registrarse o jugar en modo desconectado. Si decide jugar en modo desconectado, irá directamente al menú offline. En cambio, si usa una de las otras dos funcionalidades, el usuario pasará al menú online. En los dos menús se pueden consultar las reglas de juego y seleccionar un modo de juego, con la diferencia de que el menú online puede además llevar al usuario a su perfil o jugar contra otro jugador en línea. En cualquier caso, será llevado a la selección de modo de juego, donde tendrá

tres opciones de juego: normal, rápido y relámpago. Si se ha elegido jugar desconectado, se iniciará directamente el tablero offline. En el otro caso, el usuario deberá esperar en una actividad intermedia hasta que se cree la partida y encuentre un jugador. Una vez creada, empezará el tablero online. Si se produce un error de conexión en cualquiera de las actividades que la emplean, se iniciará automáticamente la actividad de error, donde el usuario podrá volver al inicio de sesión y empezar de nuevo.

El *activity main* será entonces el del inicio de sesión, ya que es del que parten todos los demás. Para lograrlo, hay que añadir en este fichero un intent-filter con una acción y una categoría: `action.MAIN` y `category.LAUNCHER`. También, para evitar problemas en la estética de la aplicación y simplificar los diseños, se ha indicado en todas las *activities* que la orientación de la pantalla sea siempre vertical mediante el atributo `screenOrientation="portrait"`.

Cada *activity* que tenga la aplicación ha de indicarse en el fichero AndroidManifest.xml para que el programa la reconozca. Sin embargo, el IDE empleado en este proyecto, Android Studio, lo hace automáticamente cada vez que se crea una nueva actividad mediante *New Activity*, por lo que no será necesario estar pendiente de hacer este paso.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="es.upv.etsit.MontanerCardosoCesar.tfg.chessgame">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Chess Game"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        android:usesCleartextTraffic="true">
        <activity
            android:name=".TableroOnline"
            android:theme="@style/AppThemeNoActionBar"
            android:screenOrientation="portrait" />
        <activity
            android:name=".BuscandoRival"
            android:screenOrientation="portrait" />
        <activity
            android:name=".Login"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Error"
            android:screenOrientation="portrait" />
```

Figura 20. Fragmento del fichero AndroidManifest.xml

Si una aplicación requiere conexión a internet es necesario indicárselo a la aplicación en este fichero. Como se da el caso en este proyecto, se ha escrito una instrucción que da permisos de conexión a internet:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Por último, habrá que indicar que el lenguaje de programación sea Java. En los demás campos no hace falta indicar nada, se dejarán en sus valores por defecto. Una vez finalizado, el proyecto se creará y se podrá comenzar a desarrollar el contenido de este.

4.1 Diseño de la aplicación

El diseño de esta aplicación de Android se basará en una serie de *layouts*, esquemas de diseño que indican cómo están distribuidos los elementos de ésta, interconectados entre sí. Está planteado de forma que el usuario pueda acceder a todas las variedades de pantallas en pocos pasos, a excepción del *layout* de Error, tal y como se ha visto en el Diagrama 3.

Todas ellas tienen en común varios aspectos. A excepción de los *layouts* de inicio de sesión y registro, cuentan con el mismo fondo, que es una fotografía personal, para evitar cualquier problema de copyright, con una edición en los colores con la finalidad de lograr una variación tonal menor, logrando que no resalten en exceso y se pueda apreciar mejor el contenido.

Los colores de la aplicación también se pueden compartir entre *layouts*, tanto colores RGB como colores RGBA. Esto se consigue gracias al fichero colors.xml, un fichero donde se almacenan todos los colores que se deseen emplear en la misma y así, en cada *layout*, únicamente es necesario referenciarlos con el valor del campo del fichero correspondiente. Un extracto del contenido del fichero es el siguiente:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <color name="colorPrimary">#008577</color>
4      <color name="colorPrimaryDark">#00574B</color>
5      <color name="colorAccent">#F4DCBF</color>
6      <color name="colorBlancoTablero">#F4DCBF</color>
7      <color name="colorNegroTablero">#A17349</color>
8      <color name="colorMarronArriba">#704025</color>
9      <color name="colorFondoTextos">#90704025</color>
10     <color name="colorFondoBlancoTextos">#BBFFE2C5</color>
11     <color name="colorMovimientoFicha">#902222FF</color>
```

Figura 21. Fragmento del fichero colors.xml.

Esta aplicación se ha planteado para al menos dos idiomas: El inglés y el español. La forma más sencilla de llevar a cabo una aplicación con más de un idioma es creando múltiples ficheros strings.xml, donde a cada uno le asignas un idioma. Siempre habrá uno que no tenga idioma asignado, que será el idioma por defecto. El idioma no se lo pregunta al usuario, sino que lo obtiene del sistema operativo de su dispositivo. Si el sistema operativo está en un idioma distinto a los que tiene la aplicación, se pondrá en el idioma por defecto.

En este caso, como el inglés es un idioma más universal que el español, el idioma por defecto será el inglés, y luego el español tendrá su propio fichero. Ambos ficheros deben contener los mismos campos con el mismo nombre, para que, a la hora de referenciarlos en un *layout* o en una clase Java, obtenga el valor del idioma correspondiente. En caso de que un texto no tenga traducción, se podrá indicar en el editor de traducción.

Translations Editor

Show All Keys | Show All Locales | ?

Key	Resource Folder	Untranslatable	Default Value	Spanish (es)
turnorival	app\src\main\res	☐	Rival turn	Turno del rival
JugandoContra	app\src\main\res	☐	Playing against:	Jugando contra:
partida_normal	app\src\main\res	☐	Normal Game	Partida Normal
partida_rapida	app\src\main\res	☐	Fast Game	Partida Rápida
reglas	app\src\main\res	☐	Chess rules	Reglas del ajedrez
reglasESP	app\src\main\res	☐	[...]	[...]
reglasPiezas	app\src\main\res	☐	Chess pieces	Piezas del ajedrez
reglasPiezasESP	app\src\main\res	☐	[...]	[...]
modalidad	app\src\main\res	☐	Normal game	Partida normal
partida_relampago	app\src\main\res	☐	Blitz Game	Partida Relámpago
ver_perfil	app\src\main\res	☐	See my account	Ver mi cuenta
partida_online	app\src\main\res	☐	Online game	Partida online
partida_ia	app\src\main\res	☐	Play against AI	Jugar contra la IA
nombre	app\src\main\res	☐	Name:	Nombre:
apellido	app\src\main\res	☐	Surname:	Apellidos:

Figura 22. Editor de traducciones.

Se clasificarán los *layouts* en dos tipos para este proyecto: los *layouts* fijos, que mostrarán un "pantallazo" con botones y textos y el usuario seleccionará uno, y los *layouts* dinámicos, que varían a lo largo del tiempo. Los únicos dinámicos son los referentes al tablero de juego, ya que varía el contenido en función del tiempo y de los movimientos realizados sin necesidad de actualizar la página.

4.1.1 Diseño de pantallas fijas

Los *layouts* estáticos son pantallazos con texto, imágenes, gráficas; enlaces a otros *layouts* con botones o enlaces y formularios hechos con botones, casillas, etc. Estos formularios pasarán sus valores a la clase Java relacionada con dicho *layout* para ser procesados posteriormente. Es posible que el contenido de un *layout* varíe en función de ciertos valores de variables en sus clases correspondientes, como por ejemplo los datos del usuario en el perfil.

El primer caso para analizar es el Login. Es un *layout* con un texto estático y dos campos de texto, uno de los cuales será un campo de tipo *password*, lo que hará que no se pueda ver el contenido cuando escribes. Cuenta con un botón que finaliza la actividad y el formulario, enviando los valores escritos a la clase Java Login.class, la cual llamará al fichero de inicio de sesión para localizar dicho usuario y comprobar su contraseña. Para que se produzca esto, habrá que rellenar el campo onClick del botón con un método que haga dicha acción. Los comandos que llevan a cabo esto se pueden ver en la Figura 24, explicados posteriormente en su clase Java correspondiente. En caso de error, se volverá a mostrar el inicio de sesión.

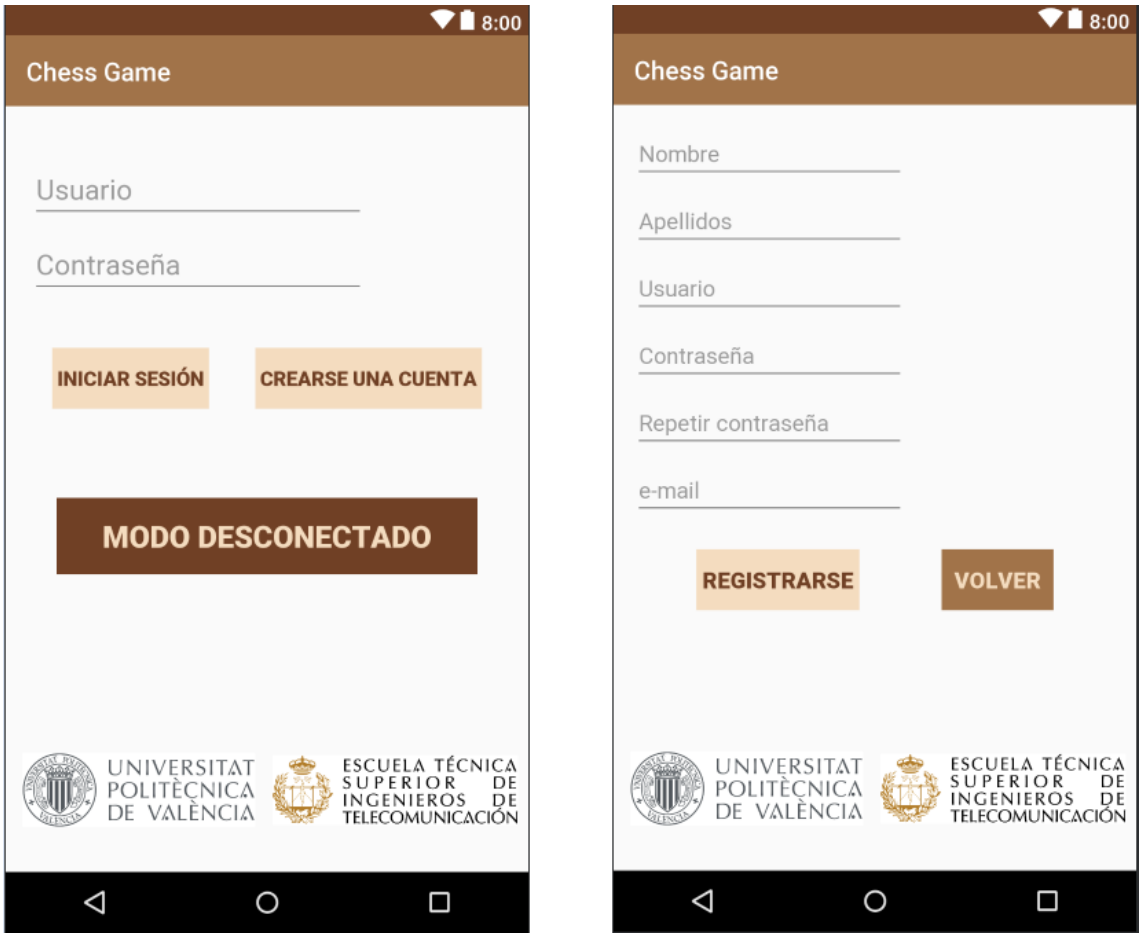

Figura 23. Layouts de las actividades de inicio de sesión y registro.

También contiene un enlace a el *layout* de registro para aquel que no tenga una cuenta creada. En el caso de no disponer de conexión o no querer conectarse, contiene otro botón con el que puedes acceder al modo desconectado, en el que sólo podrás jugar contra otro jugador con el mismo dispositivo.

```
public void CrearCuenta (View view){
    Intent intent = new Intent ( packageContext: this, Registro.class);
    startActivity(intent);
}
```

Figura 24. Ejemplo de los comandos de un método que cambia de Activity.

El *layout* del registro cuenta con una estructura similar al anterior, pero con más campos a rellenar. Lo único a destacar a diferencia del inicio de sesión es el campo de contraseña, que está duplicado. El contenido de estos campos deberá coincidir para poder procesar el formulario. El Nickname deberá de ser único también. En caso contrario, se mostrará un mensaje por pantalla indicándoselo al usuario. Si la conexión con el servidor resulta fallida, mostrará un nuevo *layout* que servirá a modo de información de desconexión. Este *layout* será explicado posteriormente.

El menú es el centro del esquema del proyecto mostrado en el Diagrama 3. En la parte superior cuenta con un botón para acceder al perfil del usuario y otro para cerrar sesión. En el centro, cuenta con dos botones similares que sirven para seleccionar la modalidad de juego dependiendo de si se desea jugar contra otro jugador online o en el mismo dispositivo. Estos botones pasarán a sus respectivas clases Java parámetros de tipo Bundle para indicar, entre otras cosas, que el jugador se encuentra en modo online.

El fondo de estos botones es una imagen sencilla en lugar de un color estático, haciendo que se vea similar a unas casillas de un tablero. La finalidad de esto es puramente estética. En la parte inferior cuenta con dos botones donde se explican las normas del ajedrez y los movimientos de las piezas. Cuenta también con un botón para volver al inicio de sesión.

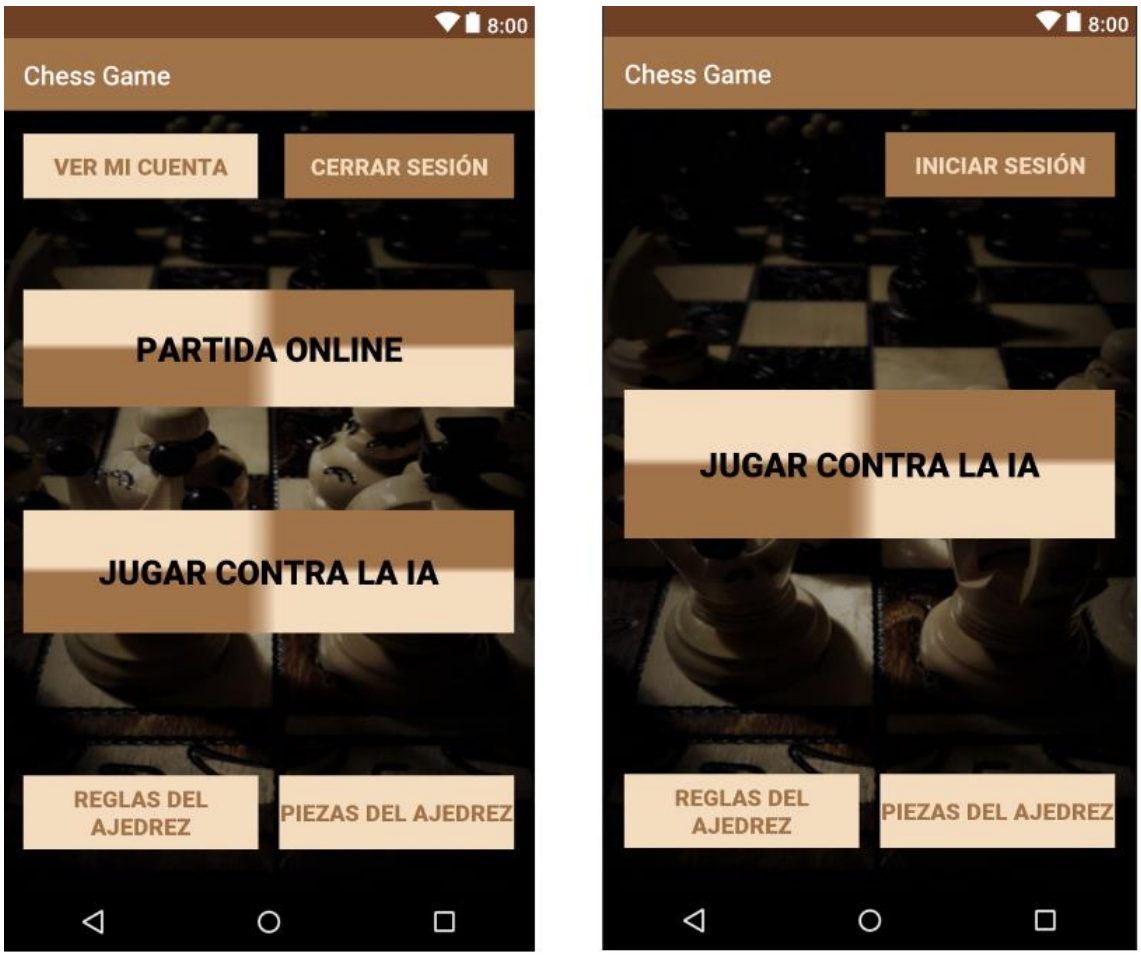

Figura 25. Layouts de las actividades menú online y menú offline.

En el caso de haber seleccionado el modo desconectado en el login, aparecerá otro menú similar al anterior, pero con menos contenido. No se podrá acceder al perfil del usuario, ya que las estadísticas se almacenan en la base de datos del servidor. En su lugar, aparecerá un texto que indique que el usuario está sin conexión y otro botón para volver al inicio de sesión. Tampoco se podrá jugar online, con lo que solo aparecerá un botón central para jugar contra otro jugador en el mismo dispositivo. Los botones en los que se pueden ver las reglas del ajedrez y los movimientos de las piezas son iguales que en el menú normal. Dichos botones también pasarán objetos Bundle a la clase que seleccione el usuario, pero en este caso indicando que el jugador está en modo offline.

El perfil del usuario se puede ver en un *layout* aparte, en el que se muestran los datos personales introducidos en el formulario de registro de jugador. También muestra el porcentaje de victorias, un dato proporcionado por la clase Java del perfil a partir de los valores obtenidos de la base de datos. Cuenta también con un botón superior que, dependiendo del Bundle obtenido que informa del estado de su conexión, volverá a un menú u otro.

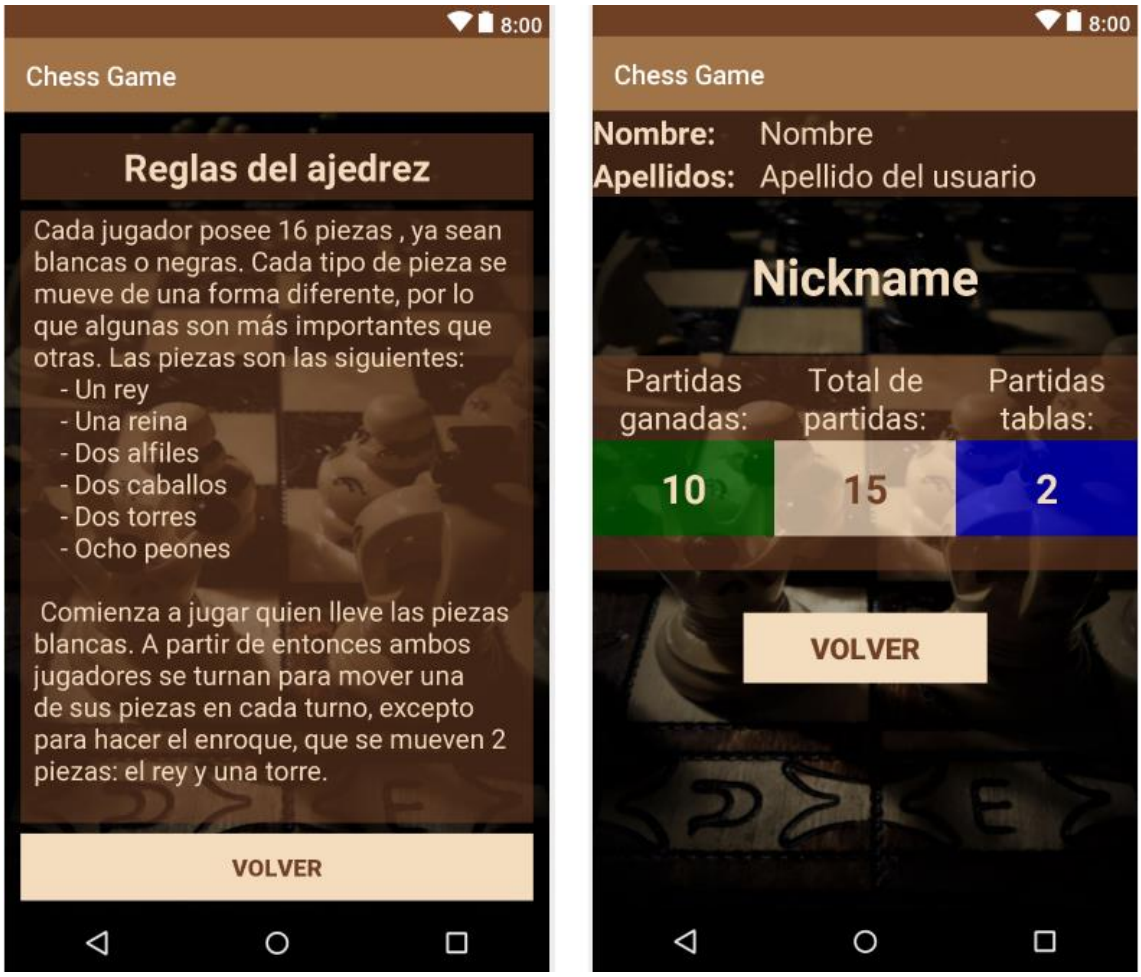

Figura 26. Layouts de las actividades de las reglas de juego y del perfil.

Los dos *layouts* donde se explican las reglas del ajedrez con prácticamente iguales, únicamente cambiando el contenido de los textos que se muestran por pantalla. Contiene un texto fijo en grande que sirve de título y luego un *Scroll view*, un elemento que sirve para poder desplazar su contenido verticalmente, muy útil cuando el texto que quieres mostrar en un *layout* es más grande que el propio *layout*. El contenido de dicho elemento es el string con las normal del ajedrez o con los movimientos de las piezas, dependiendo del elemento seleccionado previamente en el menú. En la parte inferior, hay un botón de volver que opera con las mismas condiciones que el botón que aparecía en el *layout* Perfil.

Los modos de juego se seleccionan en el *layout* de ModoDeJuego. Es el mismo para las dos opciones de juego de cara a la conexión: contra la IA o contra otro jugador. Cuenta con tres botones, donde cada uno es un modo de juego distinto (normal, rápido o relámpago). Cualquiera de estos 3 botones conduce al *layout* del tablero, pero con valores de tiempo distintos. Cuenta también con un botón superior para volver al menú.

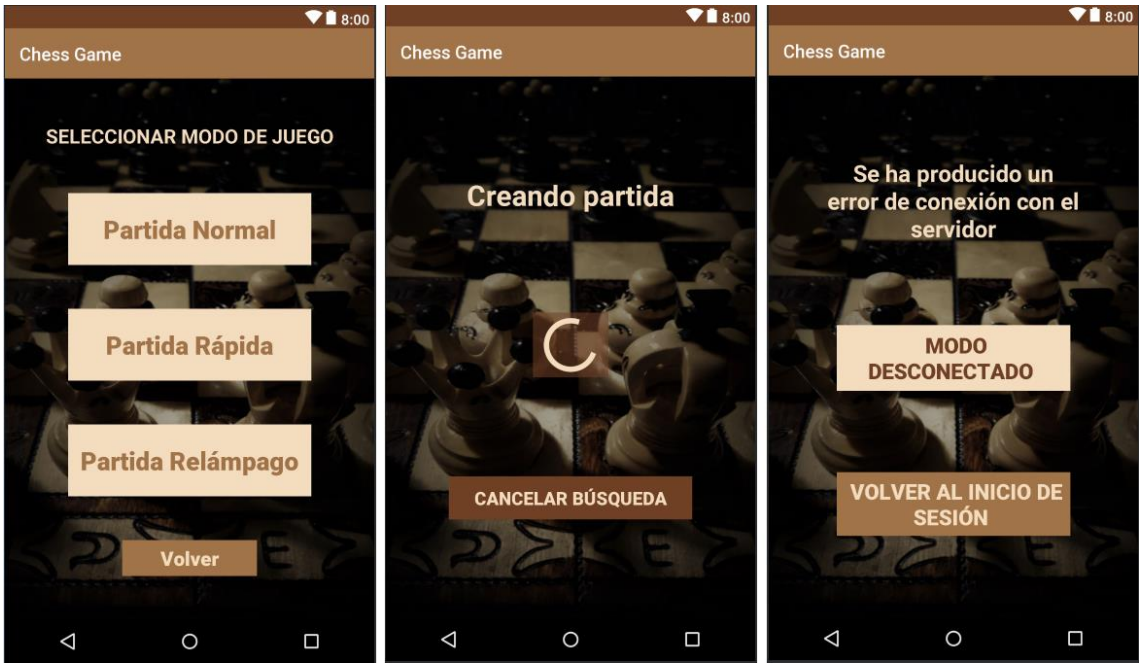

Figura 27. Layouts de las actividades de modo de juego, creando partida y error de conexión.

Una vez seleccionado el modo de juego, el programa primero buscará si hay una partida en espera en esa modalidad de juego. En caso contrario, se creará dicha partida esperando a que entre otro jugador. Mientras tanto, se mostrará un *layout* en el que aparecerá un texto informando de dicho suceso, con la posibilidad de cancelar la búsqueda y volver al menú. También hay una progress bar circular con finalidad estética y un botón de cancelar búsqueda.

Y el último *layout* estático es el de mensaje de error de conexión. Este *layout* contiene un breve mensaje a modo de text view en la zona superior que informa de que se ha perdido la conexión con el servidor. Este *layout* será accesible desde cualquier *activity* que se produzca después de haber hecho login. Contará también con dos botones, los cuales dan acceso al login o al menú en modo desconectado.

4.1.2 Diseño del tablero

El tablero es el *layout* con más elementos de toda la aplicación. Cuenta con más de 80 *views*, que es el máximo permitido en un *layout* por defecto, así que no se podría compilar. Para solucionar esto, hay que crear una variable de entorno ANDROID_LINT_MAX_VIEW_COUNT. El software comprobará si existe dicha variable en el sistema y, de ser así, su valor. Con esto se consigue variar este valor máximo, pudiendo aumentarlo o disminuirlo. En este caso, se tendrá que aumentar (se pondrá un valor de 150 para no ir ajustados con el contenido máximo del *layout*).

El tablero se dividirá en dos ficheros XML distintos, uno para cada tipo de conectividad. La distribución de los elementos del *layout* es la siguiente:

Figura 28. Layouts de las actividades del tablero online y offline.

En la parte superior, se ven dos textos fijos separados en dos *horizontal layouts* en el caso online. El segundo es el nickname del jugador contra el que estás jugando. En el caso de jugar offline, sólo hay un texto que pondrá "Player VS Player" en inglés, con su respectiva traducción al castellano. El nombre del rival se obtendrá en la clase Java de crear partida. Las dimensiones de cada sección de esta zona están acotadas por lo que se llaman *horizontal guidelines*. Su función es dar unas medidas proporcionales en función de la pantalla del dispositivo. El *layout* está planteado con estas guidelines debido a un problema que ocasiona el realizar el tablero con medidas en dps o dips, explicado posteriormente.

En la zona inferior, se muestra el tipo de partida que se está jugando, de quién es el turno y el tiempo restante de cada jugador. Este tiempo irá decreciendo conforme pasa el tiempo de cada uno, así que serán valores en continuo cambio, al igual que el turno del jugador. Por ello, habrá que ir cambiando el valor del String que representa dicho cronómetro por pantalla. La forma de lograr esto será explicada en el apartado con la clase Java correspondiente. También cuenta con un botón para rendirse, lo cual finalizará la partida y sumará al jugador rival una victoria en el caso de jugar online, volviendo al menú principal. Las dimensiones verticales de estos elementos también están acotadas por guidelines en proporción a la pantalla del dispositivo. En cuanto a las horizontales, los cronómetros y el botón de rendición cuentan con un ancho fijo, y los textos adyacentes cuentan con la anchura restante disponible. En la versión online hay unas barras de color adicionales que le indican al jugador de qué color son sus piezas para ayudar a evitar confusiones.

En el centro está el tablero. Está formado por 64 *Image Buttons*, que, tal y como indica el nombre, son botones que contienen una imagen en su interior. Esta imagen variará en función de lo que

suceda en la partida. Puede contener una pieza o un color transparente. Para lograrlo, hay que cambiar el valor del parámetro `app:srcCompat` del mismo botón. Se hará a través de Java, al mismo tiempo que se hacen las demás operaciones y cambios en el tablero.

Las dimensiones de cada casilla están hechas con guidelines horizontales y verticales, a modo de cuadrícula. En un principio, lo normal sería colocar los botones en una *Table Layout*, que consiste en una agrupación de *Table Rows* colocados verticalmente. Se colocarían 8 botones en cada fila en 8 columnas y ya estarían los 64 botones necesarios. Sin embargo, esto ocasiona problemas de dimensiones a la hora de colocar las piezas. Esto se debe a que las imágenes de las piezas tienen un tamaño definido, pero los colores no. Si se agrupan en una tabla, cada botón con una pieza ocupará lo necesario para representar la pieza con su tamaño en píxeles correspondiente, y los que tienen un color ocuparán el espacio restante para no sobrepasar el límite de la tabla, lo cual resultaría en un tablero deformado con casillas cuadradas y rectangulares. La solución planteada a este problema es la comentada anteriormente, puesto que se consigue una proporcionalidad acorde al tamaño de la pantalla y los botones no se deforman independientemente del contenido de estos.

4.2 Desarrollo del código Java

El siguiente apartado es el más extenso, el más complejo y el más importante del proyecto. Cada *activity* del proyecto tiene una clase java correspondiente, que representa el fichero XML asociado y, además, contiene todos los algoritmos y métodos necesarios para procesar la información recibida de cada uno. También hay una clase que no representa ningún *layout*, pero realiza ciertas operaciones necesarias para el correcto funcionamiento de éste, simplificando el código de las demás clases. A continuación, se analizará el contenido de todas las clases del proyecto y su funcionalidad.

4.2.1 ConexionPHP.class

La clase ConexionPHP es la única clase Java que no tiene un fichero XML asociado, con lo cual no pertenece a ninguna actividad. La utilidad de esta clase será la de conectarse a los ficheros PHP del servidor enviando unos valores y procesar la respuesta. La clase está formada por dos atributos, un constructor y dos métodos, donde el primero obtendrá y juntará los valores a enviar y el segundo los envía y procesa la respuesta.

Como esta clase tendrá funciones de entrada-salida y de conexión a la red, será necesario importar los paquetes de java que contienen estas funcionalidades. Android Studio indica qué paquete hay que importar cada vez que se escribe un método que no reconoce el código de la clase. Sin embargo, para simplificar, se incluirán todas las funciones de los paquetes java.io y java.net mediante las instrucciones `import java.io.*` para el paquete IO e `import java.net.*` para el paquete NET.

El constructor es la inicialización de la clase cuando es llamada desde otra. Este recibe como dato un String que será la URL del fichero al que accederá la clase que le ha llamado. Lo que hará con ese String será convertirlo en un objeto de la clase URL teniendo en cuenta la excepción MalformedURLException y almacenarlo en el atributo de la clase. También pondrá a nulo el atributo de datos.

El primer método, llamado *add*, recibirá dos Strings, que harán referencia al nombre de la propiedad y al valor de esta respectivamente. Primero comprobará que el atributo "datos" esté vacío, para que a la hora de poner los datos en la variable añada un "&" o no. Después, codificará tanto el nombre de la propiedad como el valor en codificación UTF-8, que es un formato de codificación de caracteres Unicode e ISO 10646 utilizando símbolos de longitud variable, el cual es capaz de representar cualquier carácter Unicode [4]. Por último, lo añadirá al atributo de datos de tipo String.

El segundo método realiza dos tareas: enviar el String de datos al servidor y procesar la respuesta recibida. La primera tarea se conecta con la URL indicada en el atributo de la clase y mediante el

método openConection de la clase URLConnection y escribe el texto de datos en un búfer de salida mediante el método write de la clase OutputStreamWriter. Por último, envía el contenido del búfer y lo cierra.

```
public String getRespueta() throws IOException {
    String respuesta = "";
    URLConnection con = url.openConnection();
    con.setDoOutput(true);
    OutputStreamWriter wr = new OutputStreamWriter(con.getOutputStream());
    wr.write(data);
    wr.close();

    BufferedReader rd = new BufferedReader(new InputStreamReader(con.getInputStream()));
    String linea;
    while ((linea = rd.readLine()) != null) {
        respuesta += linea;
    }
    return respuesta;
}
```

Figura 29. Método getRespuesta de la clase ConexionPHP.

La segunda tarea consiste en crear un buffer de entrada, un String llamado "línea", e ir almacenando en esta variable cada línea de la respuesta que ha recibido por parte del servidor que se encuentra dentro del buffer. Cuando ya no hay más líneas, finaliza el método devolviendo el resultado en una cadena de texto, donde ya se encargará de procesar la respuesta la clase que la ha llamado.

Para una mayor comprensión sobre los métodos y constructores de la Figura 29, consultar el Anexo I: Métodos y constructores de Java.

4.2.2 Login.class

La clase Login será la que ejecutará la primera actividad del proyecto. Esta actividad cuenta con tres botones y dos campos de texto. Los botones de crear cuenta y modo desconectado lo único que hacen es cambiar de una actividad a otra tal y como se indica en la Figura 24. El método onCreate, el método con el que cuentan todas las actividades e inicializa el *layout* de la actividad, declara los TextView de la actividad para poder usarlos en otros métodos y obtener lo que ha escrito el usuario en ellos.

El botón de iniciar sesión está dividido en dos partes. En una, se comprueba que el usuario haya rellenado los dos campos y se hace el cifrado de la contraseña. En la otra, se conecta con el servidor para validar los datos introducidos.

Lo primero que hace el método es obtener la cadena de texto introducida en cada campo mediante las funciones *text_view*.getText().toString(). Luego, compara las dos cadenas con "". Eso quiere decir que, si alguna de las dos cadenas de texto está vacía, muestra una Toast, un mensaje por pantalla, que le dice al usuario que es necesario rellenar los dos campos. En caso contrario, cifra la contraseña con el algoritmo de cifrado SHA-256.

```
public void IniciarSesion (View view){

    usuario = username.getText().toString();
    String contrasena = password.getText().toString();

    if(usuario.equals("") || contrasena.equals("")){
        Toast toast = Toast.makeText(getApplicationContext(), "It is necessary to fill in all the fields.", Toast.LENGTH_SHORT);
        toast.show();
    }
    else {
        contrasena256 = conversorSHA256(contrasena);
        new ObtenerUsuarios().execute();
    }
}
```

Figura 30. Método IniciarSesion de la clase Login.

Las siglas SHA significan algoritmo de "hash" seguro (Secure Hash Algorithm). SHA-256 pertenece al conjunto de funciones SHA-2, que incluye un significante número de cambios respecto al anterior grupo, SHA-1. Este grupo consiste en un conjunto de cuatro funciones hash, las cuales son de 224, 256, 384 o 512 bits.

Este algoritmo transforma la cadena de texto en un único valor de longitud fija llamado hash. También se puede aplicar a ficheros de texto, lo cual puede ser utilizado para la verificación de la integridad de copias de un dato original sin la necesidad de dar el mismo dato. Esta función es irreversible, lo que significa que un valor hash puede ser almacenado o distribuido sin problemas, ya que sólo se utiliza para fines de comparación.

Para emplear este algoritmo, se ha añadido un método que recibe un String con la contraseña y devuelve otro String con el hash de ésta. Esto se consigue creando un objeto de la clase MessageDigest y aplicándole el método getInstance("SHA-256"). Los métodos utilizados en la Figura 31 están explicados con mayor detalle en el Anexo I: Métodos y constructores de Java.

```
public String conversorSHA256(String password) {
    MessageDigest md;
    try {
        md = MessageDigest.getInstance("SHA-256");
    }
    catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        return null;
    }

    byte[] hash = md.digest(password.getBytes());
    StringBuffer sb = new StringBuffer();

    for(byte b : hash) {
        sb.append(String.format("%02x", b));
    }

    return sb.toString();
}
```

Figura 31. Método conversor256 de la clase Login.

Después de cifrar la contraseña, ejecuta una tarea asíncrona que se encargará de enviar los datos al servidor para validarlos. Esta acción se realiza en una tarea asíncrona y no en el hilo principal de ejecución del programa debido a que el servidor puede tardar en responder, y, mientras, el programa se quedaría bloqueado. Por eso, mientras la aplicación envía el formulario y espera respuesta, no se bloquea el dispositivo del usuario.

El método doInBackground de esta clase creará un objeto de la clase ConexionPHP con la URL del fichero que se encarga de hacer el inicio de sesión. A este objeto se le añadirán, mediante el método add de esta clase, los datos del usuario y contraseña, y enviará al método onPostExecute la respuesta del servidor.

```
class ObtenerUsuarios extends AsyncTask <Void, String, String> {

    @Override
    protected String doInBackground(Void[] uri) {
        try {
            ConexionPHP post = new ConexionPHP( url: "http://chessgame-app.duckdns.org/chess_server/login.php");
            post.add( propiedad: "Username", usuario);
            post.add( propiedad: "Password", contrasena256);

            return post.getRespueta();
        }
        catch (IOException e){
            return "ERROR";
        }
    }
```

Figura 32. Método doInBackground de la tarea asíncrona ObtenerUsuarios.

Este fichero, como se ha indicado en el apartado 3.2.2, en caso de éxito, devuelve un OK junto con un JSON de los datos del usuario separado por "##". Por eso, lo primero que tendrá que hacer es dividir la respuesta en partes mediante el método Split de la clase String, usando este mismo carácter. Después, analizará la primera parte de las dos. En el caso de recibir "No users found" o "Wrong Password", se le mostrará al usuario una toast con ese mismo mensaje, pero traduciéndolo al español en el caso de que el usuario tenga su dispositivo en este idioma.

```
@Override
protected void onPostExecute(String respuesta) {
    String [] partes = respuesta.split( regex: "##");
    if(respuesta.equals("No users found")){
        Toast toast = Toast.makeText(getApplicationContext(), "Username not exists", Toast.LENGTH_SHORT);
        toast.show();
    }
    else if (respuesta.equals("Wrong Password")){
        Toast toast = Toast.makeText(getApplicationContext(), "Wrong password", Toast.LENGTH_SHORT);
        toast.show();
    }
    else if(partes[0].equals("OK")){
        try {
            JSONObject json = new JSONObject(partes[1]);
            idUsuario = json.getInt( name: "IdUsuario");
            nombre = json.getString( name: "Nombre");
            apellidos = json.getString( name: "Apellidos");
            ganadas = json.getInt( name: "wonGames");
            totales = json.getInt( name: "totalGames");
            tablas = json.getInt( name: "tiedGames");

            Intent intent = new Intent( packageContext: Login.this, Menu.class);
            Bundle b = new Bundle();
            b.putInt("IdUsuario", idUsuario );
            b.putString("nombre", nombre);
            b.putString("apellidos", apellidos);
            b.putString("usuario", usuario);
            b.putInt("ganadas", ganadas);
            b.putInt("totales", totales);
            b.putInt("tablas", tablas);
            intent.putExtras( b );
            startActivity(intent);

        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
    else{
        Intent intent = new Intent( packageContext: Login.this, Error.class);
        startActivity(intent);
    }
} // onPostExecute
```

Figura 33. Método onPostExecute de la tarea asíncrona ObtenerUsuarios.

Si se recibe OK, significa que ambos datos son correctos, así que el usuario podrá tener acceso a la aplicación con su cuenta. Lo que tendrá que hacer el método es obtener el JSON con los datos del usuario que está en la segunda parte de la respuesta que había sido dividida previamente. Luego, obtendrá los datos de su nombre, apellidos, partidas jugadas y su ID mediante las funciones getInt o getString, dependiendo de si se trata de un número o un texto. Por último, creará un objeto de la clase Intent que enviará al usuario a la actividad del menú, creará un Bundle que contendrá los datos obtenidos previamente y los incluirá en el intent anterior. Todas estas operaciones deberán considerar la excepción JSONException, por lo que irán dentro de un try-catch. Si se recibe cualquier otra respuesta, redirigirá al usuario al *layout* de error de conexión.

4.2.3 Registro.class

La clase de registro es similar a la del inicio de sesión. Cuenta también con una tarea asíncrona que envía los datos al servidor y procesa la respuesta. El botón de volver cuenta con un método idéntico al de la Figura 24, con la diferencia de que este redirige al usuario al inicio de sesión.

El método de registrarse, aparte de contar ahora con más campos que analizar, contiene dos nuevas funciones que se encargan de validar el correo: *Pattern* y *Matcher*. Estas funciones no comprueban si el correo introducido existe o no, sino que analiza si la estructura de éste es la correcta. Es decir, que empiece con uno o más caracteres alfanuméricos, incluyendo guiones y guiones bajos, un "@" y una secuencia de caracteres de la A a la Z en minúsculas con al menos un punto entre medias. El código resultante es el siguiente:

```
Pattern patMail = Pattern.compile("([a-z0-9]+(\\.?[a-z0-9])*)+@(([a-z]+)\\.([a-z]+))+");
Matcher matMail = patMail.matcher(correo);
```

Este método también comprueba que todos los campos estén rellenados y cifra la contraseña con el mismo algoritmo, pero, además, verifica que las dos contraseñas sean idénticas y que el correo esté bien escrito mediante el método find de la clase Matcher. Si está todo correcto, ejecuta la tarea asíncrona de registrar usuario.

Esta tarea es prácticamente idéntica a la de obtener usuarios de la clase Login. Esta tendrá que conectarse al fichero register.php ubicado en el servidor, que es el encargado de comprobar si el nombre de usuario existe y de registrar al usuario en la base de datos. Si el nombre de usuario ya existe, el servidor lo notifica tal y como se indica en la explicación de este fichero. Si no está repetido, el servidor envía un OK junto con el JSON de los datos del usuario. La aplicación extrae los datos del JSON recibido y los envía a la actividad del menú en un Bundle de la misma forma que se hace en la clase Login.

4.2.4 Menu.class

Una vez iniciada la sesión o tras registrarse por primera vez en la aplicación, se accederá al menú principal. Lo primero que se hará es obtener los datos del usuario que ha recibido del servidor en alguna de las clases anteriores. De esto se encargará el método onCreate, que obtendrá estos parámetros del Bundle que contiene los "extras" de la actividad ejecutada previamente y los almacenará en variables declaradas al inicio de la clase.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_menu);

    Bundle bundle = getIntent().getExtras();
    idUsuario = bundle.getInt( key: "IdUsuario");
    wins = bundle.getInt( key: "ganadas");
    total = bundle.getInt( key: "totales");
    tied = bundle.getInt( key: "tablas");
    name = bundle.getString( key: "nombre");
    surname = bundle.getString( key: "apellidos");
    username = bundle.getString( key: "usuario");
}
```

Figura 34. Método onCreate de la clase Menu.

Esta actividad es la más centrada del esquema del Diagrama 3. Tiene acceso al perfil del usuario, a ver las reglas del ajedrez y de los movimientos de las piezas, a jugar tanto online como en modo desconectado en el mismo dispositivo, y al cierre de sesión. Cada una de estas acciones está acompañada de un botón con un método específico. En la siguiente figura se ve el caso concreto de jugar online.

```
//Metodo para ir a Jugar contra otro rival
public void JugarRival (View view){
    Intent intent = new Intent ( packageContext: this, ModoDeJuego.class);
    Bundle b = new Bundle();
    b.putBoolean("Offline", false );
    b.putBoolean("jugarOffline", false);
    b.putInt("IdUsuario", idUsuario);
    b.putString("nombre", name);
    b.putString("apellidos", surname);
    b.putString("usuario", username);
    b.putInt("ganadas", wins);
    b.putInt("totales", total);
    b.putInt("tablas", tied);
    intent.putExtras( b );
    startActivity(intent);
}
```

Figura 35. Método para jugar online de la clase Menu.

Este método primero generará un Intent que redirigirá al usuario a la clase correspondiente al botón que haya pulsado. Posteriormente se crea un Bundle con todos datos que ha recibido y los almacena en el Intent anterior. En el caso de que el jugador haya seleccionado cualquier modalidad de juego, se añadirá en el Bundle dos variables que indicarán si el jugador ha iniciado sesión y si ha decidido jugar online o no. Finalmente inicia la actividad que se ha indicado en el Intent.

4.2.5 MenuOffline.class

La clase del menú offline es similar a la anterior explicada, pero con una mayor sencillez. En este caso, como no se recibe ningún dato del inicio de sesión, no es necesario obtenerlos y enviarlos a las actividades adyacentes.

```
public void JugarAI (View view){
    Intent intent = new Intent ( packageContext: this, ModoDeJuego.class);
    Bundle b = new Bundle();
    b.putBoolean("Offline", true );
    b.putBoolean("jugarOffline", true);
    intent.putExtras( b );
    startActivity(intent);
}
```

Figura 36. Método para jugar offline de la clase MenuOffline.

Otro cambio a tener en cuenta es que se indicará a la clase de modo de juego que el usuario se encuentra en modo desconectado mediante un Bundle que irá dentro del Intent que accederá a esta clase.

4.2.6 Perfil.class

En esta clase se recogerán los datos recibidos por el menú, de la misma forma que lo hacía ésta al inicio de sesión, para poder representarlos por pantalla. De esto se encargará el método onCreate, ya que se debe de realizar en el momento en el que se inicia la actividad.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_perfil);

    Bundle bundle = getIntent().getExtras();
    if(bundle != null) {
        idUsuario = bundle.getInt( key: "IdUsuario");
        wins = bundle.getInt( key: "ganadas");
        total = bundle.getInt( key: "totales");
        tied = bundle.getInt( key: "tablas");
        name = bundle.getString( key: "nombre");
        surname = bundle.getString( key: "apellidos");
        username = bundle.getString( key: "usuario");

        usuario = findViewById(R.id.username);          usuario.setText(username);
        nombre = findViewById(R.id.nombre);             nombre.setText(name);
        apellidos = findViewById(R.id.apellidos);       apellidos.setText(surname);
        ganadas = findViewById(R.id.ganadas);           ganadas.setText(String.valueOf(wins));
        totales = findViewById(R.id.total_partidas);    totales.setText(String.valueOf(total));
        tablas = findViewById(R.id.tablas);             tablas.setText(String.valueOf(tied));
    }
}
```

Figura 37. Método onCreate de la clase Perfil.

Para poder representar cada parámetro, primero hay que declarar cada View mediante su ID. Los nombres de los elementos del *layout* son los que aparecen en la Figura 26. Por último, mediante la función setText, se introducen los datos en los TextView correspondientes.

```
public void Volver (View view){
    Intent intent = new Intent ( packageContext: this, Menu.class);
    Bundle b = new Bundle();
    b.putInt("IdUsuario", idUsuario);
    b.putString("nombre", name);
    b.putString("apellidos", surname);
    b.putString("usuario", username);
    b.putInt("ganadas", wins);
    b.putInt("totales", total);
    b.putInt("tablas", tied);
    intent.putExtras(b);
    startActivity(intent);
}
```

Figura 38. Método Volver de la clase Perfil.

El botón de volver es similar a los anteriores vistos. Consiste en declarar un objeto de la clase Intent que redirigirá al usuario al menú, en el cual se incluirá un Bundle con los mismos datos que ha recibido del menú.

4.2.7 ChessRules.class y ChessPieces.class

Estas dos clases son idénticas, cambiando únicamente el fichero XML que representan con la función setContentView.

```
public void Volver (View view){
    Bundle bundle = getIntent().getExtras();
    Boolean offline = bundle.getBoolean( key: "Offline");
    Intent intent;
    if(offline){
        intent = new Intent ( packageContext: this, MenuOffline.class);
    } else{
        idUsuario = bundle.getInt( key: "IdUsuario");
        wins = bundle.getInt( key: "ganadas");
        total = bundle.getInt( key: "totales");
        tied = bundle.getInt( key: "tablas");
        name = bundle.getString( key: "nombre");
        surname = bundle.getString( key: "apellidos");
        username = bundle.getString( key: "usuario");

        intent = new Intent ( packageContext: this, Menu.class);
        Bundle b = new Bundle();
        b.putInt("IdUsuario", idUsuario);
        b.putString("nombre", name);
        b.putString("apellidos", surname);
        b.putString("usuario", username);
        b.putInt("ganadas", wins);
        b.putInt("totales", total);
        b.putInt("tablas", tied);
        intent.putExtras(b);
    }
    startActivity(intent);
}
```

Figura 39. Método Volver de las clases de reglas del ajedrez.

El método volver es similar a los anteriores, pero con la diferencia de que puede ir a un menú u otro dependiendo de si se ha accedido desde el menú online u offline. Si se tratan de obtener los datos del usuario recibidos del servidor cuando se está en modo desconectado se producirá un error en la ejecución, ya que tratará de buscar un dato en el Bundle que no existe. Para ello, es importante obtener los datos sólo si se ha accedido a esta actividad desde el menú online.

4.2.8 ModoDeJuego.class

En la clase de selección de modo de juego, hay 3 botones en los cuales el usuario indica qué tipo de partida quiere jugar. La única diferencia entre estas 3 modalidades es el tiempo que dura la partida, además de que en una partida normal el tiempo de cada jugador se incrementa quince segundos tras cada tirada. Los tres botones tendrán el mismo método, que en primer lugar creará un objeto de la clase Bundle y, dependiendo de la ID de la View que se haya seleccionado, se añadirá un parámetro llamado “tiempo” con un valor u otro.

Después, se creará un Intent que dirigirá al usuario al tablero offline o a buscar rival dependiendo de si se ha seleccionado jugar offline o no, dato que se ha obtenido en el método onCreate del Bundle de la actividad anterior. También se tendrá en cuenta si se está jugando tras haber iniciado la sesión o no, así que el método lo comprueba de la misma forma que antes y, en caso de estar online, introduce todos los datos que ha recibido de la clase anterior en el Bundle. Por último, se añade el Bundle al Intent creado anteriormente y empieza la siguiente actividad.

```
public void modalidad (View view){
    Bundle b = new Bundle();
    switch(view.getId()){
        case R.id.normal: b.putInt("tiempo", 5400); break;
        case R.id.fast: b.putInt("tiempo", 3000); break;
        case R.id.blitz: b.putInt("tiempo", 600); break;
    }
    Intent intent;
    if(jugarOffline)
        intent = new Intent ( packageContext: this, Tablero.class);
    else
        intent = new Intent ( packageContext: this, BuscandoRival.class);

    if(offline)
        b.putBoolean("Offline", true);
    else {
        b.putBoolean("Offline", false);

        b.putInt("IdUsuario", idUsuario);
        b.putString("nombre", name);
        b.putString("apellidos", surname);
        b.putString("usuario", username);
        b.putInt("ganadas", wins);
        b.putInt("totales", total);
        b.putInt("tablas", tied);
    }
    intent.putExtras( b );
    startActivity(intent);
}
```

Figura 40. Método de selección de modalidad de juego de la clase ModoDeJuego.

Esta actividad también cuenta con un botón de volver. Este botón es idéntico al de las clases de las reglas de juego (Figura 39).

4.2.9 BuscandoRival.class

Esta clase cuenta con dos métodos y con dos tareas asíncronas. El primer método es el onCreate, el cual, aparte de recibir los parámetros del usuario del Bundle de la clase referente a la selección modalidad de juego, ejecuta una de las dos tareas asíncronas. Esto quiere decir que dicha tarea se ejecutará automáticamente al iniciar la actividad.

El método doInBackground, el que se ejecuta paralelo al hilo principal de ejecución, tiene dos formas de ejecutarse, que dependerá de si es la primera vez que se ejecuta la clase o no. En el primer caso, se conectará al fichero encargado de crear partidas mediante un objeto de la clase ConexionPHP, el cual tendrá dos parámetros: el ID del jugador y el tipo de partida seleccionado. Este fichero, a parte de los posibles errores que puedan surgir, respondía con dos tipos de respuestas: “Creada” y “Esperando”, junto con el ID de la partida recién creada separado por el carácter “#”.

```
class CrearPartida extends AsyncTask<Integer, String, String> {

    @Override
    protected String doInBackground(Integer[] datos) {
        String idUser = String.valueOf(datos[0]);
        try {
            if(primeraVez)
                Thread.sleep( millis: 2000);
            else
                Thread.sleep( millis: 500);
        }catch(Exception e){
            return "";
        }
        try {
            ConexionPHP post;
            if(cancelado){
                return "CANCELADO";
            }
            else if(primeraVez) {
                post = new ConexionPHP( url: "http://chessgame-app.duckdns.org/chess_server/crear_partida.php");
                post.add( propiedad: "IdJugador", idUser);
                post.add( propiedad: "TipoPartida", String.valueOf(tipoPartida));
            }
            else{
                post = new ConexionPHP( url: "http://chessgame-app.duckdns.org/chess_server/buscar_jugador.php");
                post.add( propiedad: "IdJugador", idUser);
                post.add( propiedad: "TipoPartida", String.valueOf(tipoPartida));
                post.add( propiedad: "IdPartida", String.valueOf(idPartida));
            }
            return post.getRespueta();
        }
        catch (IOException e){
            return "ERROR";
        }
    }
```

Figura 41. Método doInBackground de CrearPartida.

En el primer caso, el método onPostExecute generará un nuevo Intent redirigirá al usuario al *layout* del tablero. Posteriormente, se crea un objeto de la clase Bundle, el cual contendrá todos los parámetros pasados por la actividad anterior, además del nombre de usuario del otro jugador, dato que también lo proporciona el servidor. Por último, se añade el Bundle al Intent creado previamente e inicia la actividad del tablero online.

```
@Override
protected void onPostExecute(String respuesta) {
    String [] partes = respuesta.split( regex: "#");
    if(!cancelado) {
        if (partes[0].equals("Creada")) {
            idPartida = Integer.parseInt(partes[1]);
            Intent intent = new Intent( packageContext: BuscandoRival.this, TableroOnline.class);
            Bundle b = new Bundle();
            b.putInt("tiempo", tiempo);
            b.putInt("IdUsuario", idUsuario);
            b.putInt("IdPartida", idPartida);
            b.putString("nickOtro", partes[2]);
            b.putString("nombre", name);
            b.putString("apellidos", surname);
            b.putString("usuario", username);
            b.putInt("ganadas", wins);
            b.putInt("totales", total);
            b.putInt("tablas", tied);
            intent.putExtras(b);
            startActivity(intent);

        } else if (partes[0].equals("Esperando")) {
            idPartida = Integer.parseInt(partes[1]);
            if (primeraVez) {
                TextView buscandoOponente = findViewById(R.id.BuscandoOponente);
                buscandoOponente.setText("Looking for an opponent");
                primeraVez = false;
            }
            new CrearPartida().execute(idUsuario);
        } else {
            Intent intent = new Intent( packageContext: BuscandoRival.this, Error.class);
            startActivity(intent);
            Toast toast = Toast.makeText(getApplicationContext(), respuesta, Toast.LENGTH_SHORT);
            toast.show();
        }
    }
}
```

Figura 42. Método onPostExecute de CrearPartida.

En el segundo caso, el jugador quedará a la espera de un nuevo jugador. Para saber cuándo lo han emparejado, ejecutará de nuevo la misma tarea asíncrona, con la diferencia de que ahora hará la consulta a otro fichero, que será el encargado de informar al usuario si ya se ha creado la partida o sigue esperando. Esta consulta se hará en bucle cada medio segundo hasta que se cree. Mientras, se cambiará la frase que aparece en pantalla de "creando partida" por "buscando un oponente" para que el usuario tenga una mejor información de lo que está sucediendo. En caso de cualquier tipo de error, será redirigido al *layout* de error de conexión.

La segunda tarea asíncrona se ejecutará al darle al botón de cancelar búsqueda. Su función es conectarse al servidor de la misma forma que la anterior, con la diferencia de que se conectará al fichero encargado de borrar la partida. De esta forma, cuando busque partida otro usuario, no lo emparejarán con el que acaba de cancelarla. Después, volverá al menú principal. Al igual que en la otra actividad, en caso de error de conectividad, el usuario será redirigido al *layout* de error.

```
class BorrarPartida extends AsyncTask<Void, String, String> {

    @Override
    protected String doInBackground(Void[] nada) {
        try {
            ConexionPHP  post = new ConexionPHP( url: "http://chessgame-app.duckdns.org/chess_server/borrar_partida.php");
            post.add( propiedad: "IdPartida", String.valueOf(idPartida));
            return post.getRespueta();
        }
        catch (IOException e){
            return "ERROR";
        }
    }

    @Override
    protected void onPostExecute(String respuesta) {
        if(respuesta.equals("Partida borrada")){
            Intent intent = new Intent( packageContext: BuscandoRival.this, Menu.class);
            Bundle b = new Bundle();
            b.putInt("IdUsuario", idUsuario);
            b.putString("nombre", name);
            b.putString("apellidos", surname);
            b.putString("usuario", username);
            b.putInt("ganadas", wins);
            b.putInt("totales", total);
            b.putInt("tablas", tied);
            intent.putExtras( b );
            startActivity(intent);
        }
        else{
            Intent intent = new Intent( packageContext: BuscandoRival.this, Error.class);
            startActivity(intent);
            Toast toast = Toast.makeText(getApplicationContext(), respuesta, Toast.LENGTH_SHORT);
            toast.show();
        }
    }
} // BorrarPartida
```

Figura 43. Tarea asíncrona BorrarPartida.

4.2.10 Tablero.class

Las clases referentes al tablero de juego son las que más computación simultánea necesitan. Se tiene que comprobar si el jugador ha seleccionado una pieza o no, si luego la casilla que ha marcado es válida o no para mover la ficha, si la ficha es aliada, la posición de todas las piezas para que no se salten entre ellas, a excepción del caballo, que sí tiene permitido esta acción. Además de los cronómetros de cada jugador, pausando cada uno a su debido tiempo.

Si se integran todos estos procesos en el hilo principal de ejecución del programa, la interfaz gráfica se vería comprometida, sufriendo bloqueos por cada acción o, directamente, que la actividad no se inicie. Android cuenta con una forma de ejecutar todos estos cálculos simultáneamente sin que la interfaz del programa se vea comprometida. Dicha funcionalidad son los hilos o tareas asíncronas, las cuales se ejecutan aparte del hilo principal del programa, y éstas sobrescriben la interfaz gráfica de la forma deseada. Se almacenan en clases dentro de la clase principal o superclase, en este caso Tablero.class. Estos hilos se pueden ejecutar tanto en serie

como en paralelo. El cómo se han empleado este tipo de clases en el programa será explicado posteriormente.

Esta clase tiene una gran cantidad de variables que será necesario ubicarlas al principio de ésta para que puedan ser llamadas desde los distintos métodos que contiene. Contarán con un valor inicial, el cual muchas veces no hará falta indicarlo ya que se pondrá el valor por defecto. Resumiendo, estas variables hacen referencia a las casillas del tablero, la posición inicial de cada una, el tiempo restante de cada jugador, el turno de la partida, las casillas seleccionadas y la casilla de la pieza movida por el jugador anterior, si se puede hacer enroque, si se puede hacer la captura de peón al paso, si ha finalizado la partida o se ha producido un jaque, etc.

```
final String TAG = "TAG";

int casilla = 10, casillaAnterior = 0, casillaPiezaAnterior = 23;
int posReyB = 60, posReyN = 4;
boolean [] posPiezas = new boolean [64]; boolean [] posBlancas = new boolean [64]; boolean [] posNegras = new boolean [64];

String nombre;
boolean piezaBlanca;

boolean piezaSeleccionada, piezaMarcada, peonAlPaso, jaque;
boolean isPartidaNormal;
boolean movPeonN, movPeonB, movReyN, movReyB, movTorreN, movTorreB, movAlfilN, movAlfilB; //booleans necesarios para el enroque
boolean turnoBlancas = true;
int cPeonAlPaso;
boolean finPartida = false; boolean jaqueMate = false; boolean tablas = false;

Drawable pieza;

Drawable peon_b;    Drawable peon_n;
Drawable torre_b;   Drawable torre_n;
Drawable caballo_b; Drawable caballo_n;
Drawable alfil_b;   Drawable alfil_n;
Drawable reina_b;   Drawable reina_n;
Drawable rey_b;     Drawable rey_n;

Drawable colorTransparente, colorMovFicha;

int tiempoGlobal, tiempoBlancas, tiempoNegras;

CronometroColor cronoJugadores;
TextView cronometro1, cronometro2, modoJuego;
Button surrender;

int [] idButtons = new int [] {R.id.B1, R.id.B2, R.id.B3, R.id.B4, R.id.B5, R.id.B6, R.id.B7, R.id.B8, R.id.B9, R.id.B10, R.id.B11,
```

Figura 44. Variables comunes a toda la clase Tablero.

Lo siguiente a analizar son las clases asíncronas. En esta clase, sólo será necesario emplear un hilo asíncrono: el cronómetro. Cada jugador cuenta con un cronómetro, que irá descontando el tiempo restante a cada jugador cuando sea su turno. A pesar de que haya dos cronómetros aparentemente independientes, sólo será necesario emplear una clase asíncrona, la cual irá alternando el decremento de tiempo según el turno de la partida.

Las clases asíncronas pueden implementar hasta 5 métodos sobrescritos, que se ejecutarán de forma automática antes, durante y tras finalizar la ejecución. También hay otro que se ejecutará en el caso de que sea cancelada. Estos 4 son opcionales, de los cuales en esta clase se empleará principalmente el que actualiza el cronómetro durante la ejecución del hilo y el que finaliza la partida al acabarse el tiempo. El último método es obligatorio en este tipo de clase, doInBackground, el cual es el que realiza la ejecución en paralelo para no detener la interfaz gráfica.

```
@Override
protected Void doInBackground(Integer ... tiempo) {
    tiempoBlancas = tiempo[0];
    tiempoNegras = tiempo [0];
    while(tiempoBlancas >= 0 && tiempoNegras >= 0 && !isCancelled()) {
        try {
            Thread.sleep( millis: 1000);
            if(turnoBlancas) {
                tiempoBlancas--;
                publishProgress(tiempoBlancas);
            }else{
                tiempoNegras--;
                publishProgress(tiempoNegras);
            }
        } catch (Exception e) {
            publishProgress( ...values: 0);
            return null;
        }
        if (isCancelled()) {
            publishProgress( ...values: 0);
            return null;
        }
    }
    publishProgress( ...values: 0);
    return null;
}
```

Figura 45. Método doInBackground de la AsyncTask Cronometro.

Este método analizará las variables de la clase Tablero referentes al turno en cada segundo. Por ello, se emplea un bucle que se ejecutará mientras ningún cronómetro llegue a cero. Dependiendo de cada turno, descontará una variable u otra y enviará el valor del tiempo restante al método onProgressUpdate para representarlo por pantalla en su respectivo cronómetro. Estas variables de tiempo se inicializan en el método onCreate al momento de llamar a esta clase. El valor dependerá del tipo de partida que se haya elegido, por ello habrá que obtener dicha modalidad desde el Bundle de la clase anterior.

```
Bundle bundle = getIntent().getExtras();
tiempoGlobal = bundle.getInt( key: "tiempo");
isPartidaNormal = tiempoGlobal == 5400;
boolean isPartidaRapida = tiempoGlobal == 3000;

InicCronos(); InicPosiciones();

cronoJugadores = new CronometroColor();
cronoJugadores.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR,tiempoGlobal);
```

Figura 46. Llamada a la AsyncTask desde onCreate.

El método onCreate es el que se ejecuta al iniciar la clase. Se encarga de representar el fichero XML en el dispositivo, de inicializar los cronómetros y algunas variables relacionadas con las piezas, y de representar por pantalla qué tipo de partida se está jugando.

El método onClick es el que se encarga de analizar la casilla que se ha pulsado y qué hay en ella. Es el que usa casi todos los otros métodos que contiene la clase. Primero comprueba que sea el turno del jugador, ya que si no lo es no hace nada. Después obtiene la ID del botón que ha sido pulsado y la compara con el vector de IDs de los botones del tablero. Luego analiza el contenido de éste, ya sea una pieza y su color o una casilla vacía. A continuación, comprueba si se había seleccionado una pieza o no anteriormente. Si no se había seleccionado ninguna, esa pieza pasará a ser la seleccionada, lo que ejecutará los métodos encargados de comprobar los posibles movimientos. Si ya se había seleccionado una, comprueba la validez del movimiento. Si es válido,

se mueve la pieza y se cambia el turno. En caso contrario, la pieza ya no queda seleccionada y vuelve al estado inicial.

Al finalizar este método, comprueba si se ha producido un jaque, si se ha finalizado o no la partida y cambia el texto y los colores que indican el turno de la partida, además de actualizar las variables referentes a la posición de los reyes para poder comprobar correctamente el jaque en cada turno.

En esta clase, aparte de los dos métodos previamente analizados, hay 21 métodos relacionados entre sí para el correcto funcionamiento de ésta. Se encargan de inicializar los vectores, comparar una pieza con otra, de comprobar la validez de un movimiento, decir el nombre y el color de una pieza, si se ha realizado un jaque, si se ha finalizado la partida, si el jugador quiere rendirse, etc.

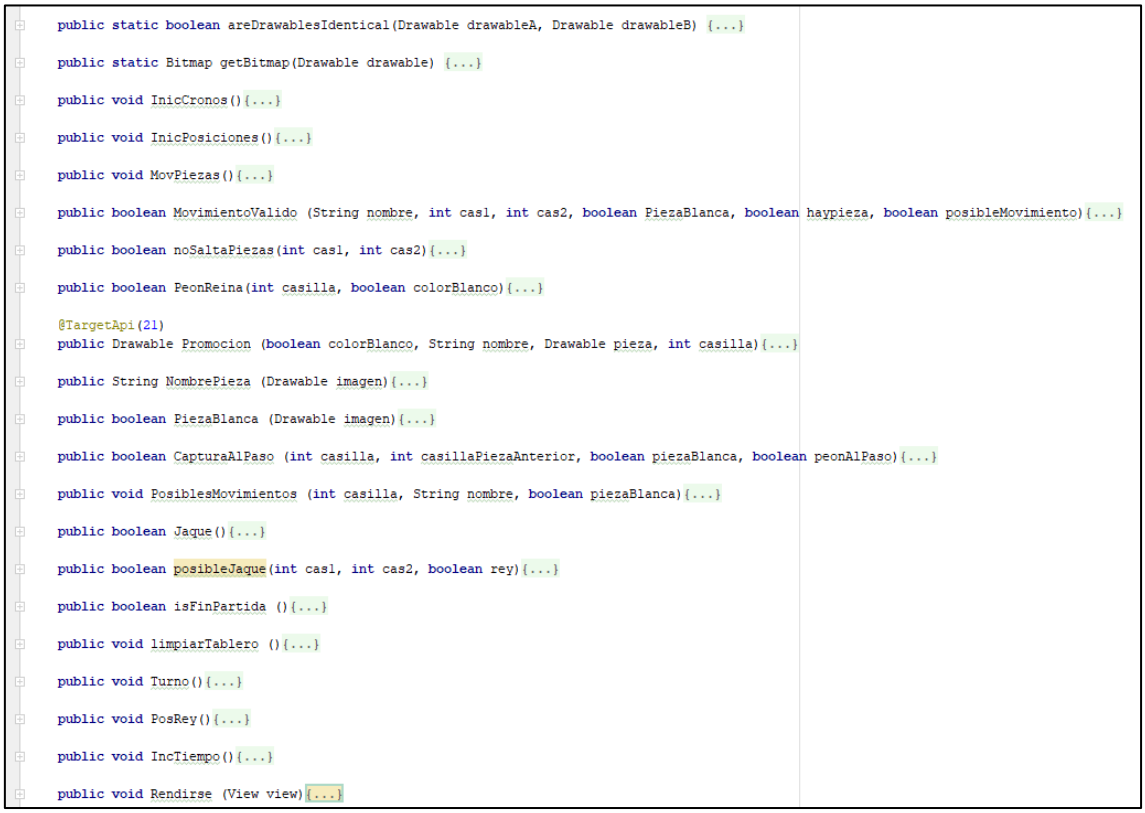

Figura 47. Métodos de la clase Tablero.

Esta clase tiene la funcionalidad de poder jugar dos personas al juego con el mismo dispositivo Android. Es el modo de juego al que se podrá jugar cuando no hay conexión a Internet.

4.2.11 TableroOnline.class

La clase Java referente al tablero de juego online contiene el mismo motor de juego que el tablero offline. Para hacer funcionar ésta, es necesario añadir nuevas tareas asíncronas y nuevas variables de control comunes a toda la clase para poder compartir la partida con otro jugador.

Las nuevas variables que se han añadido han sido el String con el nombre del jugador rival y cuatro variables de tipo boolean que controlan si es el turno del jugador, si el jugador lleva las piezas blancas o negras y si es la primera vez que recoge la información de la base de datos o no. Esta última por defecto será un *true*.

El principal cambio que se ha realizado en esta clase es el de añadir cuatro tareas asíncronas nuevas. La primera se encargará de recibir constantemente la información de la partida de la base de datos y representarla por pantalla. La segunda enviará el estado de la partida al servidor cada vez que se realice un movimiento, para que el otro jugador lo reciba y pueda mover. La última borrará la partida cuando ésta se acabe.

La tarea encargada de recibir la información de la partida se llama InfoPartida. El método doInBackground realiza una consulta al fichero info_partida.php enviándole el ID de la partida mediante un objeto de la clase ConexionPHP, al igual que en las anteriores tareas asíncronas de conexión. Esta tarea deberá estar activa durante todo el transcurso de la partida, así que se pondrá dentro de un bucle *do-while* que se ejecutará mientras la partida no finalice. De esta forma, la aplicación estaría enviando peticiones al servidor sobre el estado de la partida, lo cual puede saturar tanto al dispositivo como al servidor si hay numerosas partidas en curso. Para solucionar este problema, se añade a cada iteración del bucle un tiempo de espera de 500 milisegundos. El inconveniente de hacer este cambio es que el juego irá con un retardo de conexión mínimo de medio segundo. Como el ajedrez es un juego que no requiere una respuesta lo más rápida posible, es un problema que se puede asumir.

La respuesta recibida del servidor es analizada por el método onProgressUpdate, que la recibirá del método doInBackground. Las respuestas que puede recibir, según se ha visto en el fichero info_partida.php en el apartado 3.2.2, son "OK" seguido de un objeto JSON con los datos del registro de la partida, "PartidaNoExiste" o un error que se pueda de dar con la base de datos. Si se recibe un "PartidaNoExiste" significará que la partida ha sido borrada, por lo que este método se encargará de poner las variables de fin de partida a true. Si se ha recibido un error, enviará al usuario al *activity* de Error de la misma forma que se hace en las otras clases.

Si se recibe un "OK" se separará la respuesta total recibida en dos partes mediante los caracteres "##". Luego, se tendrá en cuenta solamente el JSON del segundo fragmento. Una vez obtenido, se tendrá en cuenta si es la primera vez que se obtiene la información de la partida o no. Si es la primera vez, se analizará de quién es el jugador que juega las blancas para saber quién mueve primero, establecer los turnos correctamente, y cambiar los dos espacios de colores del tablero referentes al color de las piezas del usuario.

```
JSONObject json = new JSONObject(partes[1]);
if(primeraVezInfo){
    int jugadorB = json.getInt( name: "JugadorB");
    turnoJugador = jugadorB == idUsuario;
    jugadorEsBlancas = jugadorB == idUsuario;
    if(jugadorEsBlancas) {
        idOtro = json.getInt( name: "JugadorN");
        ImageView color1 = findViewById(R.id.color1); color1.setImageDrawable(getDrawable(R.color.colorBlancoFicha));
        ImageView color2 = findViewById(R.id.color2); color2.setImageDrawable(getDrawable(R.color.colorBlancoFicha));
    }
    else {
        idOtro = jugadorB;
        ImageView color1 = findViewById(R.id.color1); color1.setImageDrawable(getDrawable(R.color.colorNegroFicha));
        ImageView color2 = findViewById(R.id.color2); color2.setImageDrawable(getDrawable(R.color.colorNegroFicha));
    }
    primeraVezInfo = false;
}
```

Figura 48. Primera vez que se obtiene la información de la base de datos.

Para el resto de las consultas sobre el estado de la partida, se comprobará si ha habido un cambio en la partida. Para ello, se analizará en primer lugar el turno de la partida. Esto se hace comprobando que el campo IdTurno de la tabla sea el mismo o no que el campo IdJugador. Si son iguales, le tocará al usuario, así que se cambiarán las variables de control y podrá mover pieza. En caso contrario, esas variables permanecerán en el mismo estado.

A continuación, se actualizarán las variables relacionadas con la captura de peón al paso y el jaque. Después se ejecutará un método de esa misma tarea asíncrona que se encargará de procesar el JSON que contiene la información de las piezas y representarlas. Este JSON es el que está explicado en la Figura 2.

```
if(!turnoJugador) {
    int idTurno = json.getInt( name: "IdTurno");
    if (idTurno == idUsuario) {
        turnoJugador = true; turnoBlancas = !turnoBlancas;
    }
    finPartida = json.getInt( name: "FinPartida") != 0;
    if(finPartida){
        if(idUsuario == json.getInt( name: "FinPartida"))
            ganador = username;
        else
            ganador = nickrival;
    }
    casillaPiezaAnterior = json.getInt( name: "PiezaAnterior");
    peonAlPaso = json.getInt( name: "PeonAlPaso") == 1;
    jaque = json.getInt( name: "Jaque") == 1;
    posReyB = json.getInt( name: "PosReyB");
    posReyN = json.getInt( name: "PosReyN");
    String infPieces = json.getString( name: "InfoPiezas");
    String JSON = infPieces.replace( target: "\\\"", replacement: "\"");
    JSONObject infoPiezas = new JSONObject(JSON);
    representarPiezas(infoPiezas);
}
Turno();
```

Figura 49. Procesado de la información recibida por el servidor.

Este método primero divide los valores de cada propiedad del objeto en vectores de enteros separándolos mediante el carácter "#". Después, rellena un vector de tipo String de tamaño 64, en el cual se irán poniendo los nombres de las piezas en la posición que se ha obtenido de cada una. Por último, se añaden las piezas al tablero mediante un *switch-case* con el vector anterior y la función setDrawable. Finalmente, el método onPostExecute ejecuta el método que analiza los turnos para que, en el caso de que haya cambiado el turno, se represente por pantalla y el usuario sea consciente de ello.

La siguiente tarea asíncrona para analizar es la encargada de actualizar la partida. Será llamada al final del método onClick tras realizar un movimiento. Lo primero que hará al ejecutarse será obtener el estado del tablero. De esto se encargará un método de esta tarea que devolverá un String con un JSON de las piezas con el mismo formato que estaba en la base de datos, para no perder la sinergia. Este String será almacenado en una variable común a la clase ActualizarPartida. El funcionamiento de este método consiste en generar un bucle que analiza el nombre y el color de cada pieza del tablero, y con un *switch-case* se almacena el valor de la casilla en una variable u otra, intercalando cada número con un "#". Si no hay ninguna pieza de un tipo, añade un "NO". Finalmente, almacena todas estas cadenas de texto con números en un String con el formato indicado anteriormente para devolverlo con el *return*.

El método doInBackground de esta tarea es similar a los anteriores. En este caso, se conecta con el fichero actualizar_partida.php y envía todos los datos requeridos por este fichero. Lo único especial a tener en cuenta es que hay que convertir los *true* o *false* de las variables que indican la captura de peón al paso y el jaque en unos o ceros, ya que el tipo de contenido de estas columnas es un número entero. El método finaliza devolviendo la respuesta del servidor.

El método onPostExecute analiza la respuesta recibida de este fichero. Si es un OK, significa que todo ha ido bien, así que la partida sigue sin problemas. Si recibe un "PartidaNoExiste" significará que la partida ha sido borrada porque el contrincante se ha rendido, por lo que el método finalizará la partida y contará una victoria al usuario. Si recibe cualquier otra respuesta, significaría que se ha producido un error, con lo que conducirá al usuario al *activity* Error.

Cuando acaba la partida, es necesario actualizar los registros de los jugadores con relación al número de partidas jugadas. Para ello, será necesario el uso de otra clase asíncrona que le envíe al fichero actualizar_usuarios.php la ID del usuario e indicarle si ha ganado, ha empatado o ha

perdido. Esta clase recibirá estos valores en forma de un boolean cuando sea llamada, que será justo antes de ejecutar la tarea de borrar la partida. El método onPostExecute se encargará únicamente de comprobar que no se haya producido ningún error en la actualización.

```
class ActualizarUsuarios extends AsyncTask<Boolean, String, String> {

    @Override
    protected String doInBackground(Boolean[] valores) {
        boolean victorias = valores[0];
        boolean tablas = valores [1];

        try {
            ConexionPHP post = new ConexionPHP( url: "http://chessgame-app.duckdns.org/chess_server/actualizar_usuarios.php");
            post.add( propiedad: "IdUsuario", String.valueOf(idUsuario));
            if(victorias)
                post.add( propiedad: "Victoria",  valor: "1");
            else
                post.add( propiedad: "Victoria",  valor: "0");
            if(tablas)
                post.add( propiedad: "Tablas",  valor: "1");
            else
                post.add( propiedad: "Tablas",  valor: "0");

            return post.getRespueta();

        } catch (IOException e) {
            return "ERROR";
        }
    }
```

Figura 50. Método doInBackground de ActualizarUsuarios.

La última tarea asíncrona añadida a esta clase es la encargada de borrar la partida de la base de datos. Ésta se conecta con el fichero borrar_partida.php enviándole el dato del ID de la partida y analiza la respuesta que recibe. Si se recibe el mensaje “Partida borrada”, volverá al menú principal con los datos recibidos de la actividad anterior mediante un Bundle. Si se recibe otro mensaje es porque se ha producido un error, con lo que el usuario será redirigido a la actividad de Error.

4.2.12 Error.class

La última clase por analizar es Error.class. Sólo se accederá a esta actividad cuando se produzca un error de conexión con el servidor y no se reciba respuesta de éste, ya sea porque el usuario se ha quedado sin conexión a internet o porque el servidor ha quedado inaccesible. Es la clase más sencilla de todas, ya que cuenta con dos métodos, además del onCreate, los cuales representan a un botón cada uno. Cada método creará un Intent que redirigirá al usuario a la clase que haya seleccionado, ya sea el inicio de sesión o el modo desconectado.

```
public class Error extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_error);
    }

    public void ModoOffline (View view){
        Intent intent = new Intent ( packageContext: this, MenuOffline.class);
        startActivity(intent);
    }

    public void VolverLogin (View view) {
        Intent intent = new Intent( packageContext: this, Login.class);
        startActivity(intent);
    }
}
```

Figura 51. Métodos de la clase Error.

5. Conclusiones y propuesta de trabajo futuro

5.1 Conclusiones

En líneas generales se puede decir que el objetivo principal del trabajo se ha cumplido, que es el hacer un ajedrez bilingüe en línea desde cero junto al servidor de juego. Para la realización del proyecto se ha necesitado tener conocimientos previos sobre programación en Java y en gestión de bases de datos. Sin embargo, no ha sido necesario a la hora de programar en Android o PHP, ya que se ha usado mayormente Java y los ficheros PHP, que cuentan con funciones simples y repetidas en casi todos los ficheros.

Lo más costoso de desarrollar ha sido el motor de juego, ya que el ajedrez es de los juegos clásicos con más normas y restricciones de cara a la programación. También ha supuesto varios problemas el que el juego coloree los posibles movimientos antes de mover una pieza, ya que alteraba todos los métodos que hacían funcionar al juego y se ha requerido hacer varios cambios en el código.

Se ha cumplido el hacer un diseño de actividades sencillo pero atractivo y con detalles originales, dentro de lo que un ajedrez puede ofrecer, y adaptable a distintos tamaños y proporciones de la pantalla de un dispositivo móvil. También se ha simplificado la estructura de la base de datos todo lo posible para una mayor sencillez en las consultas SQL y, con ello, una mayor rapidez de procesado y respuesta del servidor.

5.2 Propuesta de trabajo futuro

Una de las posibles ampliaciones para tener en cuenta es integrar una inteligencia artificial con varios niveles de dificultad para que el usuario pueda jugar solo y aprender más, ya que no siempre podrá jugar con alguien si está sin conexión.

El diseño actual de la aplicación está adaptado únicamente a dispositivos móviles. Esto conlleva que si se ejecuta en una Tablet no se verá de una forma atractiva. Así que el hacer un fichero adicional para cada *layout* adaptado a Tablets ampliaría las opciones de jugabilidad y las oportunidades de mercado.

También sería algo más ventajoso para la aplicación el disponer de un mejor servidor con una mayor accesibilidad. Pero para lograrlo, habría que tener en cuenta los gastos de gestión y almacenamiento, ya sea porque los datos se almacenen en la nube o porque se compre equipo adicional para seguir almacenándolos en el servidor actual.

Si la aplicación llegase a tener un gran número de usuarios, sería interesante comparar las estadísticas de cada jugador con las de los demás, donde se vean los jugadores con más victorias y los que tengan un porcentaje mayor de victorias. Esto se conseguiría creando actividades adicionales encargadas de recopilar estos datos.

Tal y como está planteada la aplicación, no se puede recibir ninguna clase de ingresos de ella a no ser que se introduzca como aplicación de pago. Si se hace esto, el número de descargas disminuirá considerablemente, ya que las aplicaciones de pago son menos descargadas que las gratuitas generalmente. Si se desea comercializar en un futuro, sería una opción interesante introducir anuncios o pagos en el juego para obtener un beneficio de éste.

Por último, sería cómodo para el usuario vincular la cuenta que tenga en la aplicación con su cuenta de Google Play, para que no tenga que iniciar sesión cada vez que entra y su cuenta se quede almacenada en una base de datos más confiable.

6. Bibliografía

[1] Reglas del ajedrez. https://feda.org/feda2k16/reglamentacion-feda-y-leyes-del-ajedrez/

[2] "An update on Eclipse Android Developer Tools" Anuncio oficial del cierre de soporte de Android a Eclipse ADT, Junio 2015: https://android-developers.googleblog.com/2015/06/an-update-on-eclipse-android-developer.html

[3] "¿Qué es una API y para qué sirve?" Publicación digital de TICbeat, 12 de Julio de 2014: https://www.ticbeat.com/tecnologias/que-es-una-api-para-que-sirve/

[4] "UTF-8, a transformation format of ISO 10646" RFC 3629: https://tools.ietf.org/html/rfc3629

[5] Documentación de la clase MessageDigest de Java en Oracle: https://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html

[6] Documentación de la clase StringBuffer de Java en Oracle: https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuffer.html

[7] Documentación de la clase String de Java en Oracle: https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

[8] Documentación de la clase URLConnection de Java en Oracle: https://docs.oracle.com/javase/7/docs/api/java/net/URLConnection.html

[9] Documentación de la clase URL de Java en Oracle: https://docs.oracle.com/javase/7/docs/api/java/net/URL.html

[10] Documentación de la clase OutputStreamWriter de Java en Oracle: https://docs.oracle.com/javase/7/docs/api/java/io/OutputStreamWriter.html

[11] Documentación de la clase InputStreamReader de Java en Oracle: https://docs.oracle.com/javase/7/docs/api/java/io/InputStreamReader.html

[12] Documentación de la clase BufferedReader de Java en Oracle: https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html

[13] Guía oficial de Android: https://developer.android.com/reference

[14] Guía estructurada de PHP: https://www.php.net/manual/es/index.php

[15] SQL en PHP: https://aprende-web.net/progra/sql/sql_1.php

[16] Foro de consultas sobre programación: https://stackoverflow.com

Anexo I. Métodos y constructores de Java

1. Class MessageDiggest [5]:

```
java.lang.Object
      java.security.MessageDigestSpi
            java.security.MessageDigest
```

Modificador y tipo	Método	Descripción
`byte[]`	`digest(byte[] input)`	Realiza una actualización final en el resumen utilizando la matriz de bytes especificada, luego completa el cálculo del resumen.
`String`	`getInstance()`	Devuelve un objeto MessageDigest que implementa el algoritmo de resumen especificado.

Tabla 1. Métodos de la clase MessageDigest utilizados.

2. Class StringBuffer [6]:

```
java.lang.Object
      java.lang.StringBuffer
```

Modificador y tipo	Método	Descripción
`StringBuffer`	`append(String str)`	Agrega la cadena especificada a esta secuencia de caracteres.

Tabla 2. Método append de la clase StringBuffer.

3. Class String [7]:

```
java.lang.Object
      java.lang.String
```

Modificador y tipo	Método	Descripción
`static String`	`format(String format, Object... args)`	Devuelve una cadena con formato utilizando la cadena y los argumentos con el formato especificado.

Tabla 3. Método format de la clase String.

4. Class URLConnection [8]:

```
java.lang.Object
      java.net.URLConnection
```

Modificador y tipo	Método	Descripción
`void`	`setDoOutput(boolean dooutput)`	Establece el valor del campo doOutput para esta URLConnection en el valor especificado.
`OutputStream`	`getOutputStream()`	Devuelve una secuencia de salida que escribe en esta conexión.
`InputStream`	`getInputStream()`	Devuelve una secuencia de entrada que se lee desde esta conexión abierta.

Tabla 4. Métodos de la clase URLConnection utilizados.

5. Class URL [9]:

```
java.lang.Object
      java.net.URL
```

Modificador y tipo	Método	Descripción
URLConnection	openConnection()	Devuelve una instancia de URLConnection que representa una conexión con el objeto remoto al que hace referencia la URL.

Tabla 5. Método openConnection de la clase URL.

6. Class OutputStreamWriter [10]:

```
java.lang.Object
      java.io.Writer
            java.io.OutputStreamWriter
```

Constructor	Descripción
OutputStreamWriter (OutputStream out)	Crea un OutputStreamWriter que usa la codificación de caracteres predeterminada.

Tabla 6. Constructor utilizado de la clase OutputStreamWriter.

Modificador y tipo	Método	Descripción
void	write(String str, int off, int len)	Escribe una porción de un String
OutputStream	close()	Cierrra el flujo de salida, enviándolo primero.

Tabla 7. Métodos de la clase OutputStreamWriter utilizados.

7. Class InputStreamReader [11]:

```
java.lang.Object
      java.io.Reader
            java.io.InputStreamReader
```

Constructor	Descripción
InputStreamReader(InputStream in)	Crea un InputStreamReader que usa el juego de caracteres predeterminado.

Tabla 8. Constructor utilizado de la clase InputStreamReader.

8. Class BufferedReader [12]:

```
java.lang.Object
      java.io.Reader
            java.io.BufferedReader
```

Constructor	Descripción
BufferedReader(Reader in)	Crea un búfer de un flujo de caracteres de entrada que utiliza un tamaño predeterminado.

Tabla 9. Constructor utilizado de la clase BufferedReader.

Modificador y tipo	Método	Descripción
String	readLine()	Lee una línea de texto

Tabla 10. Métodos de la clase BufferedReader utilizados.