5-5-2021

Aplicación Android para conectar con el proyecto del Pokémon.

Grado Superior en Desarrollo de Aplicaciones Multiplataforma.

Álvaro Carbajo Alcalde
IES COMERCIO

Índice

Metodologías ágiles.

¿Qué es un sprint?

El núcleo central de la metodología de trabajo “scrum” es el “sprint”. Se trata de un mini proyecto de no más de un mes (ciclos de ejecución muy cortos -entre una y cuatro semanas), cuyo objetivo es conseguir un incremento de valor en el producto que estamos construyendo. Todo ‘sprint’ cuenta con una definición y una planificación que ayudará a lograr las metas marcadas.

Sprint 1 (22/3/21 – 12/4/21):

Realización de un programa en Android que permita comunicarme con el proyecto de Windows, así como añadir los cambios necesarios al proyecto Windows para posibilitar la comunicación. Además, realización de los métodos necesarios para escribir los XML.

Sprint 2 (12/4/21 – 26/4/21):

Realización de la base de datos de Android y las vistas necesarias para el correcto funcionamiento del proyecto. Realización de los cambios necesarios para transformar el proyecto Windows en una especie de servidor.

Sprint 3 (26/4/21 – 10/5/21):

Programar el Android para mostrar en las vistas los datos que llegan del PC. Crear la vista de la Pokédex en el dispositivo móvil.

Sprint 4 (10/5/21 – 24/5/21):

Realización de las pruebas pertinentes del funcionamiento del proyecto en su conjunto. Redactar la memoria del proyecto.

Introducción.

Este trabajo se basa en realizar una aplicación para dispositivos Android que permita comunicaciones con el proyecto del módulo de Desarrollo de Interfaces, realizado durante el curso de Desarrollo de Aplicaciones Multiplataforma (DAM) en el curso académico 2020/21.

El proyecto del módulo se basa en el popular videojuego RPG (role-playing game o, en español, juego de rol) japonés ¨Pokémon¨. Los Pokémon son una clase de criaturas inspiradas en animales reales, insectos, objetos, plantas o criaturas mitológicas. Los jugadores toman el papel Entrenadores Pokémon y tienen que entrenar un equipo de Pokémon para competir contra otros entrenadores. El objetivo final del juego es llegar a ganar la Liga Pokémon y convertirse en el campeón regional. La temática de coleccionar, entrenar y luchar está presente en casi todas las versiones de la franquicia Pokémon, incluidos los videojuegos, las series de anime y películas. Estos videojuegos tienen un sistema de combate basado en turnos, en el que el jugador controla a una serie de criaturas con las cuales puede combatir realizando distintos movimientos.

El sistema de combate consiste en batallas de seis contra seis, en las que participan dos entrenadores. Durante el combate se pueden realizar varias acciones como usar un movimiento de los cuatro que tiene cada Pokémon, usar un objeto, cambiar de Pokémon, huir del combate...

El combate lo gana el jugador que logre eliminar a los seis Pokémon de su adversario.

Durante el módulo de Desarrollo de Interfaces se realizó una aplicación para Windows utilizando formularios de Windows Forms. El proyecto se basa en replicar los combates de los videojuegos para crear un juego similar a estos en el que poder realizar combates contra una IA simple (que elige ataques de forma aleatoria).

La aplicación que he realizado en este Módulo profesional de Proyecto es un controlador remoto que permita jugar al proyecto Pokémon desarrollado en D.I. Lo he programado para dispositivos Android utilizando Java y Android Studio. Además, he tenido que modificar el proyecto del Pokémon para que permita el juego multijugador a través de redes LAN. (Red de Área Local).

La aplicación permitirá conectarse a la partida en curso para controlar al rival y poder jugar PVP (Jugador contra Jugador). Así como poder visualizar la Pokédex (una enciclopedia que los entrenadores Pokémon llevan consigo para registrar las fichas de todas las diversas especies Pokémon vistas y capturadas durante su viaje como entrenadores).

La comunicación entre el dispositivo Android y el ordenador se realizará mediante sockets TCP (protocolo de red importante que permite que dos anfitriones o hosts se conecten e intercambien flujos de datos). Utilizando el protocolo IP (protocolo que tiene la función de establecer las comunicaciones entre todos los dispositivos que tratan de relacionarse entre sí en internet.). Y se enviarán archivos XML que contendrán la información necesaria para el correcto funcionamiento de la aplicación, como los datos del combate o las habilidades que tiene cada Pokémon.

Objetivos.

- Enviar y recibir mensajes por TCP entre dos aplicaciones, una en un ordenador y otra en un dispositivo Android.
- Enviar y recibir ficheros XML a través de la conexión TCP.
- Crear una base de datos SQLite en el dispositivo Android que almacene los datos de los Pokémon.
- Rellenar dicha base de datos de forma automática partiendo de la ya existente en el ordenador.
- Programar una forma para poder visualizar dichos datos de forma cómoda para el usuario.
- Programar una forma de desplazarse por dichos datos mediante gestos.
- Adaptar el proyecto de Desarrollo de Interfaces para poder jugar multijugador entre distintos dispositivos.
- Establecer una conexión automática entre el móvil y el ordenador, transformando el juego de multijugador a un solo jugador en tiempo de ejecución.
- Crear una interfaz de usuario en el teléfono móvil que permita ser modificada en tiempo de ejecución de forma que adapte su contenido los datos recibidos.

Fases del proyecto.

-Diseño.

El proyecto se compone de dos programas distintos:

- El desarrollado para Desarrollo de Interfaces es un programa de ordenador, para Windows, que utiliza Windows Forms. Está escrito en C# y utilicé Visual Studio 2019. Para guardar los datos de los Pokémon, ataques, entrenadores… utiliza Microsoft Access (un gestor de datos que utiliza los conceptos de bases de datos relacionales y pueden manejarse por medio de consultas e informes. Está adaptado para recopilar datos de otras utilidades como Excel, SharePoint y otros programas de Microsoft). Se utilizó este programa para almacenar datos porque es muy compatible con Visual Studio. Para probar la aplicación he usado mi ordenador personal, un PC de sobremesa, Windows 10, con un I7 9700K y 16GB de RAM.

- La parte cliente, está desarrollada en Java, para móviles Android. Utilicé para programarlo Android Studio. Al ser un servidor que recibe la información del servidor no requiere de una base de datos para poder realizar combates, sin embargo, para poder realizar la Pokédex es necesario una base de datos. La base de datos del proyecto Android usa SQLite y las herramientas que proporciona Android para realizar consultas. Para probar la aplicación usé un Xiaomi Mi A2 Lite.

Diagramas de Casos de Uso:

El programa Servidor, como fue realizado como proyecto para la asignatura de Desarrollo de Interfaces tiene más acciones que el usuario puede realizar, sin embargo, en este diagrama me centro en las acciones que han tenido que ser modificadas para poder desarrollar el apartado multijugador del proyecto. Estas acciones son las que tienen que ver con el combate, tales como realizar movimientos y cambiar de Pokémon.

Para el cliente sí que se muestran todas las acciones que el usuario puede realizar, se dividen en dos, la parte de los combates y la parte de la visualización de la Pokédex.

Diagrama del Servidor (Ordenador):

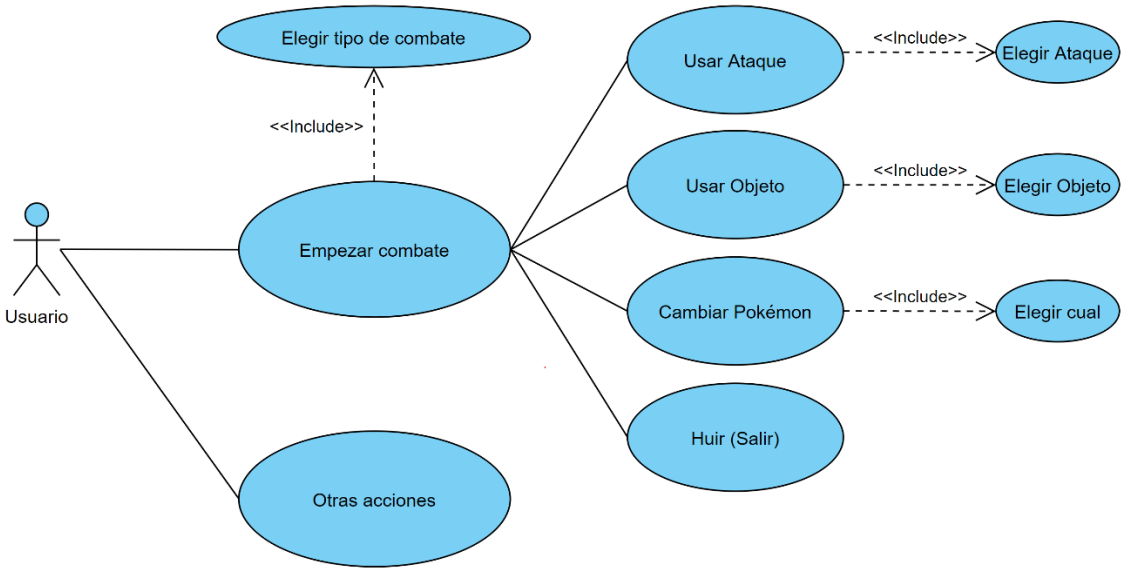
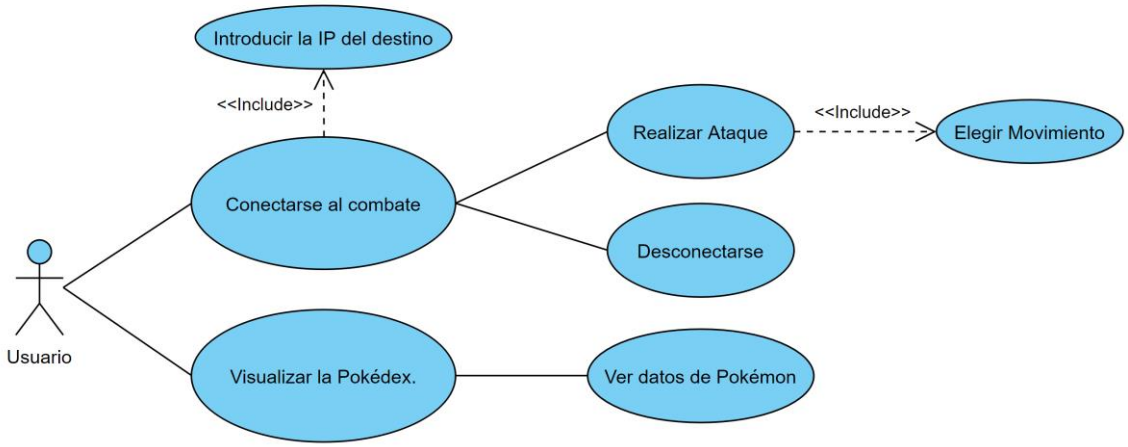


Diagrama del Cliente (Android):



Diseño de la aplicación móvil:

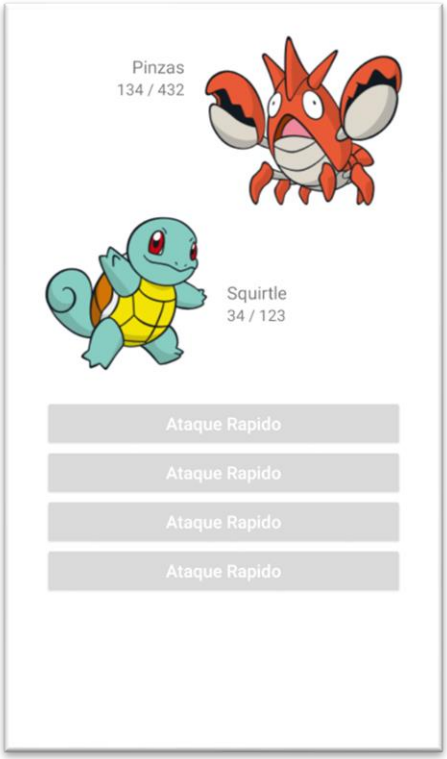
Para la aplicación Android he creado cuatro actividades, esto es, una ventana que utiliza Android para introducir en ella las interfaces de usuario, las actividades se dividen en su parte visual (un XML) y en su parte lógica. Estas cuatro actividades son: La ¨main¨ o principal donde el usuario podrá elegir entre las acciones de visualizar Pokédex y la de conectarse al combate, la actividad del combate donde se podrán realizar movimientos y jugar cuando se encuentre conectado con el ordenador y las dos de la Pokédex (la primera siendo una lista de todos los Pokémon y la segunda que se abre cuando un Pokémon de la lista es seleccionado y contiene todos los datos de un Pokémon en concreto).

*** Actividad del combate:**

Durante el combate, esta actividad muestra los datos de los dos Pokémon que se encuentran en el combate (el nuestro y el del rival). Muestra las imágenes que representan a las criaturas junto con sus nombres y la vida restante que tienen. En la parte inferior se encuentran los botones que tienen los datos del ataque que se puede realizar.

Los botones de los movimientos durante el combate tendrán, además del nombre del movimiento, los PP restantes (veces que se puede usar dicho movimiento) y una pequeña imagen vectorial en el lado izquierdo del botón que indica el tipo que tiene el movimiento.

Tras realizar un movimiento los botones desaparecerán (se harán invisibles) para dejar paso a un Toast (notificación en pantalla de Android) que mostrara un mensaje sobre lo acontecido durante el turno.

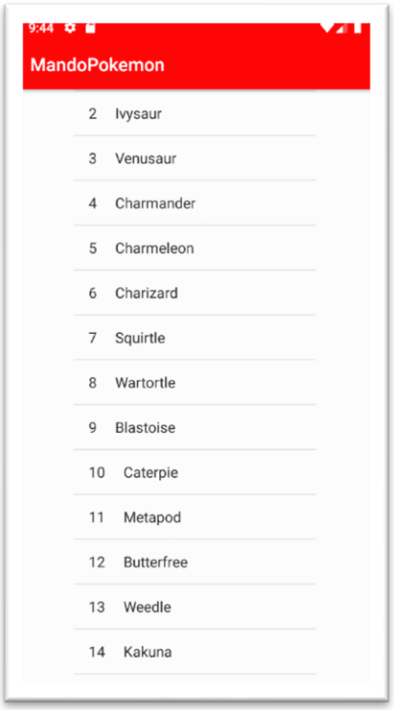


Layout del combate.

*** Actividad lista Pokédex.**

La actividad de la lista de Pokémon contiene un List View que se carga al leer todos los datos de la tabla Pokémon, en ella se muestran el id correspondiente al número de la Pokédex y el nombre del Pokémon.

Al hacer clic en uno de los elementos del List View se lanzará la actividad de visualizar un Pokémon concreto y se cargarán los datos necesarios en el Layout de dicha actividad.



Captura de pantalla de la lista Pokédex.

*** Actividad de visualizar Pokémon.**

Al seleccionar un elemento de la lista podemos ver las características del Pokémon que son: su nombre y categoría en la parte de arriba, la imagen en el centro, su tipo o tipos, peso, altura y una descripción.

Las características que se muestran del Pokémon están almacenadas en el sistema SQLite que incluye la aplicación.

Podremos cambiar de Pokémon arrastrando el dedo de derecha a izquierda, o viceversa, para avanzar por la Pokédex e ir viendo todos los Pokémon, que son 493 en total e incluye todos los Pokémon existentes hasta la cuarta generación de videojuegos.

Las imágenes son cargadas desde el sistema de recursos de Android.

Layout de la Pokédex.

El modelo de la aplicación Android se basa en dos clases principales:

Pokémon: que contiene los datos del Pokémon que esta luchando como su vida, tipos… Un Pokémon tiene cuatro movimientos.

Movimiento: contiene los datos del movimiento como los puntos de poder restantes, totales…

PokedexPokemon es una clase auxiliar que sirve para contener los datos de los Pokémon de la Pokédex, tales como categoría, descripción, peso, altura…

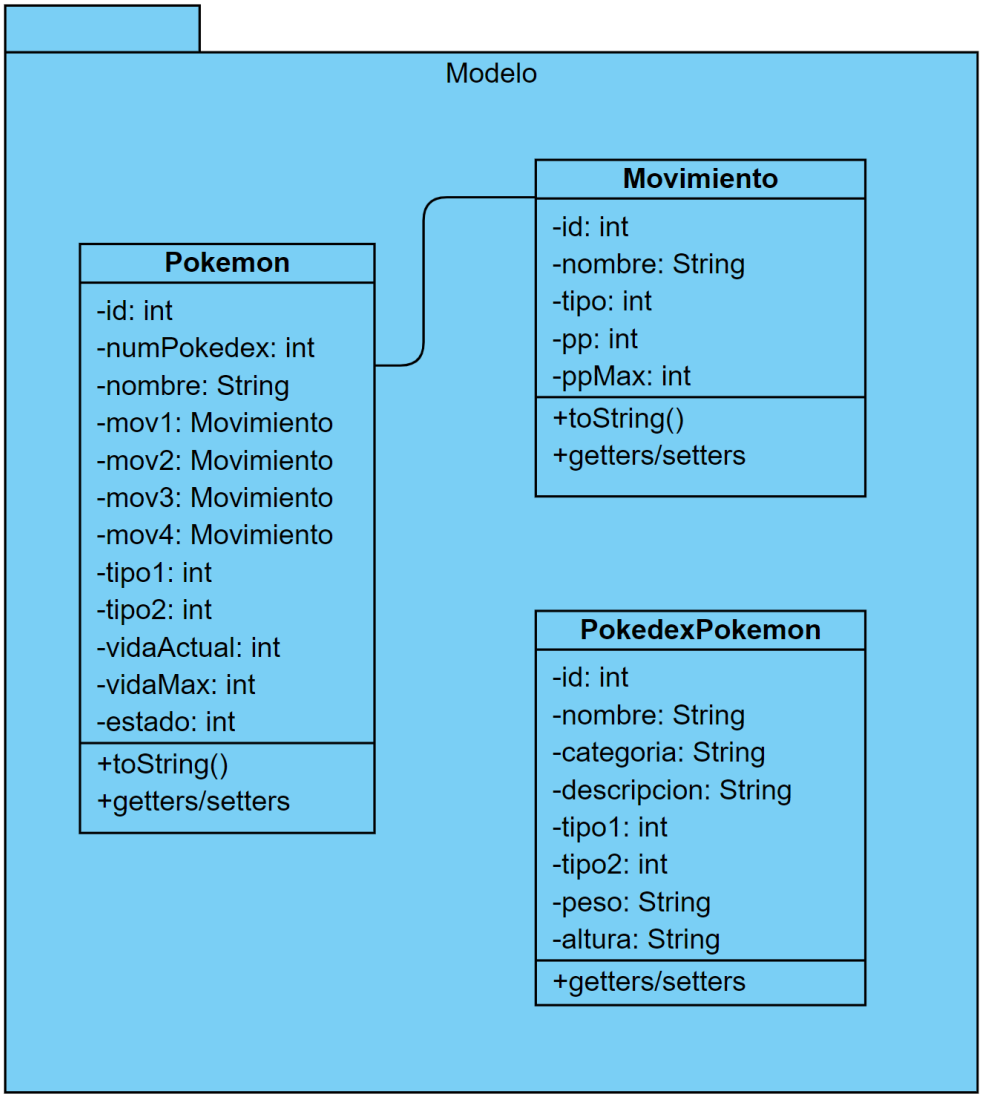


Diagrama de clases de Android.

-Implementación Android.

Manifest:

En el manifest son necesarios los permisos de almacenamiento para guardar los XML recibidos y los de internet, que permiten abrir los puertos TCP.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_INTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_INTERNAL_STORAGE" />
```

Base de datos de Android:

La base de datos de Android es un SQLite con una única tabla, la tabla Pokémon que contiene los datos de los Pokémon que se mostraran en la Pokédex.

Utilizo la clase "ConexionSQLiteHelper" que extiende de SQLiteOpenHelper. Esta clase permite la creación y actualización de la base de datos de forma muy sencilla.

Para las tablas creo variables para los nombres de los campos de forma que sea fácil cambiar los nombres de los campos cambiando únicamente los valores de estas variables.

Diagrama tabla Pokémon.

Variables de los campos:

```
//TABLA POKEMON
public static final String NOMBRE_TABLA_POKEMON = "POKEMON";
public static final String CAMPO_POKEMON_ID = "ID_POKEMON";
public static final String CAMPO_POKEMON_NOMBRE = "NOMBRE";
public static final String CAMPO_POKEMON_TIPO_1 = "TIPO_1";
public static final String CAMPO_POKEMON_TIPO_2 = "TIPO_2";
public static final String CAMPO_POKEMON_DESCRIPCION = "DESCRIPCION";
public static final String CAMPO_POKEMON_PESO = "PESO";
public static final String CAMPO_POKEMON_ALTURA = "ALTURA";
public static final String CAMPO_POKEMON_CATEGORIA = "CATEGORIA";
```

```
//Crear tablas
public static final String CREAR_TABLA_POKEMON = String.format(
        "CREATE TABLE %s(" +
                "%s INTEGER PRIMARY KEY, " +
                "%s TEXT NOT NULL, " +
                "%s INTEGER NOT NULL, " +
                "%s INTEGER, %s TEXT, " +
                "%s TEXT, %s TEXT, " +
                "%s TEXT" +
                ")",
        NOMBRE_TABLA_POKEMON, CAMPO_POKEMON_ID, CAMPO_POKEMON_NOMBRE,
        CAMPO_POKEMON_TIPO_1, CAMPO_POKEMON_TIPO_2,
        CAMPO_POKEMON_DESCRIPCION, CAMPO_POKEMON_CATEGORIA,
        CAMPO_POKEMON_PESO, CAMPO_POKEMON_ALTURA);
```

El resto de la clase tiene los métodos de la clase SQLiteOpenHelper: ¨onCreate¨ que se ejecuta al crear la base de datos y ¨onUpgrade¨ que se ejecuta cuando se realizan modificaciones a la tabla, se basa en borrarla y volver a crearla.

```
//Drop table
public static final String DROP_TABLA_POKEMON = String.format("DROP TABLE IF EXISTS %s", NOMBRE_TABLA_POKEMON);

public ConexionSQLiteHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
    super(context, name, factory, version);
}

@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(CREAR_TABLA_POKEMON);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL(DROP_TABLA_POKEMON);
    onCreate(db);
}
```

Para introducir los datos he usado los mismos datos que tenía en la base de datos Access del proyecto de Desarrollo de Interfaces. Para poder exportarlos programé un método en C# que lee la base de datos y escribe un XML, luego puse el XML resultante en Android Studio y leí mediante un programa en Java los datos y fui insertándolos en la base de datos.

```
public static void CrearXMLPokedex()
{
    try
    {
        //Creo el archivo para escribir el XML
        StreamWriter file = new StreamWriter(RUTA_FICHERO_XML_POKEMON, append: false);

        //Abro la conexion a la base de datos de ACCESS con un singleton
        OleDbConnection con = ConexionAccess.GetConexion();
        con.Open();

        //Hago la consulta
        OleDbCommand selectPokemon = new OleDbCommand(CONSULTA_POKEMON, con);
        OleDbDataReader reader = selectPokemon.ExecuteReader();

        //Leo la consulta
        file.WriteLine($"{CABECERA_XML}\n");
        file.WriteLine("<Pokemons>\n");
        while (reader.Read())
        {
            file.WriteLine("\t<Pokemon>");

            file.WriteLine($"\t\t<IdPok>{reader.GetInt32(0)}</IdPok>");
            file.WriteLine($"\t\t<Nombre>{reader.GetString(1)}</Nombre>");
            file.WriteLine($"\t\t<Categoria>{reader.GetString(2)}</Categoria>");
            file.WriteLine($"\t\t<Tipo1>{reader.GetInt32(3)}</Tipo1>");
            file.WriteLine($"\t\t<Tipo2>{reader.GetInt32(4)}</Tipo2>");
            file.WriteLine($"\t\t<Descripcion>{reader.GetString(5)}</Descripcion>");
            file.WriteLine($"\t\t<Peso>{reader.GetString(6)}</Peso>");
            file.WriteLine($"\t\t<Altura>{reader.GetString(7)}</Altura>");

            file.WriteLine("\t</Pokemon>\n");
        }
        file.WriteLine("</Pokemons>");
        file.Close();

        con.Close();
    }
    catch (Exception e)
    {
        Console.WriteLine("ERROR al crear XML de pokedex: " + e.Message);
    }
}
```

Método para escribir el XML de la Pokédex.

```
private static readonly string CONSULTA_POKEMON = "SELECT * FROM POKEMON ORDER BY ID";

private static readonly string CABECERA_XML = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";
```

Consulta y cabecera del XML.

```java
/**
 * Lee los datos del xml para insertarlos en la base de datos
 */
public void leerDatosXML() {
    PokedexPokemon pokemon = new PokedexPokemon();
    try {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        // Parseamos el documento y lo almacenamos en un objeto Document
        Document doc = builder.parse(getResources().openRawResource(R.raw.pokedex));

        // Obtenemos el elemento raiz del documento, pokemons
        Element raiz = doc.getDocumentElement();

        // Obtenemos todos los elementos llamados pokemon, que cuelgan de la raiz
        NodeList items = raiz.getElementsByTagName("Pokemon");

        // Recorremos todos los elementos obtenidos
        for (int i = 0; i < items.getLength(); i++) {
            Node nodoPokemon = items.item(i);

            // Recorremos todos los hijos que tenga el nodo pokemon
            for (int j = 0; j < nodoPokemon.getChildNodes().getLength(); j++) {
                Node nodoActual = nodoPokemon.getChildNodes().item(j);

                // Compruebo si es un elemento
                if (nodoActual.getNodeType() == Node.ELEMENT_NODE) {
                    switch (nodoActual.getNodeName().toLowerCase()) {
                        case "idpok":
                            pokemon.setId(Integer.parseInt(nodoActual.getChildNodes().item( index: 0).getNodeValue()));
                            break;
                        case "nombre":
                            pokemon.setNombre(nodoActual.getChildNodes().item( index: 0).getNodeValue());
                            break;
                        case "tipo1":
                            pokemon.setTipo1(Integer.parseInt(nodoActual.getChildNodes().item( index: 0).getNodeValue()));
                            break;
                        case "tipo2":
                            pokemon.setTipo2(Integer.parseInt(nodoActual.getChildNodes().item( index: 0).getNodeValue()));
                            break;
                        case "categoria":
                            pokemon.setCategoria(nodoActual.getChildNodes().item( index: 0).getNodeValue());
                            break;
                        case "descripcion":
                            pokemon.setDescripcion(nodoActual.getChildNodes().item( index: 0).getNodeValue());
                            break;
                        case "peso":
                            pokemon.setPeso(nodoActual.getChildNodes().item( index: 0).getNodeValue());
                            break;
                        case "altura":
                            pokemon.setAltura(nodoActual.getChildNodes().item( index: 0).getNodeValue());
                            break;
                    }
                }
            }
            //Insertamos esos datos
            arrayListVista.add(new VistaLista(pokemon.getId(), pokemon.getNombre()));
            listaPokemon.setAdapter(adaptador);
            insertarDatos(pokemon);
        }
```

Método que lee el XML para insertar los datos en la base de datos.

```java
/**
 * Inserta datos en la base de datos.
 *
 * @param pokemon Datos a insertar
 */
public void insertarDatos(PokedexPokemon pokemon) {
    try {
        SQLiteDatabase db = conn.getWritableDatabase();
        ContentValues values = new ContentValues();

        //Insertamos valores
        values.put(CAMPO_POKEMON_ID, pokemon.getId());
        values.put(CAMPO_POKEMON_NOMBRE, pokemon.getNombre());
        values.put(CAMPO_POKEMON_CATEGORIA, pokemon.getCategoria());
        values.put(CAMPO_POKEMON_DESCRIPCION, pokemon.getDescripcion());
        values.put(CAMPO_POKEMON_ALTURA, pokemon.getAltura());
        values.put(CAMPO_POKEMON_PESO, pokemon.getPeso());
        values.put(CAMPO_POKEMON_TIPO_1, pokemon.getTipo1());
        values.put(CAMPO_POKEMON_TIPO_2, pokemon.getTipo2());

        db.insert(NOMBRE_TABLA_POKEMON, CAMPO_POKEMON_ID, values);
        db.close();
    } catch (Exception e) {
        System.out.println("Error insert: " + e.getMessage());
    }
}
```

Método que inserta los datos.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Pokemons>
 - <Pokemon>
      <IdPok>1</IdPok>
      <Nombre>Bulbasaur</Nombre>
      <Categoria>Semilla</Categoria>
      <Tipo1>12</Tipo1>
      <Tipo2>17</Tipo2>
      <Descripcion>Una rara semilla fue
          Pokémon.</Descripcion>
      <Peso>6,9 kg</Peso>
      <Altura>0,7 m</Altura>
  </Pokemon>
 - <Pokemon>
      <IdPok>2</IdPok>
      <Nombre>Ivysaur</Nombre>
      <Categoria>Semilla</Categoria>
      <Tipo1>12</Tipo1>
      <Tipo2>17</Tipo2>
      <Descripcion>Puede aumentar su f
          lomo.</Descripcion>
      <Peso>13 kg</Peso>
      <Altura>1 m</Altura>
  </Pokemon>
```

Estructura del XML resultante.

Visualización de los datos:

Una vez cargados los datos, el usuario puede visualizarlos mediante la Pokédex. La actividad de la lista, carga todos los datos en un List View mediante el uso de un adaptador. Al seleccionar un elemento de dicha lista el usuario accederá a la siguiente actividad: ver los datos del Pokémon seleccionado.

```
//Click listener de la lista
listaPokemon.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Intent intent = new Intent(view.getContext(), PokedexActivity.class);
        Bundle bundle = new Bundle();
        bundle.putInt("id", position + 1);
        intent.putExtras(bundle);
        startActivityForResult(intent,  requestCode: 0);
    }
});
```

Código que cambia entre una actividad y otra.

La actividad Pokédex recibe el id del Pokémon y consulta a la base de datos para poder mostrar las características en la vista.

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

    //Bundle
    Bundle entrega = this.getIntent().getExtras();
    pokemonActual =  entrega.getInt( key: "id");

    initComponents();

    //Detector de gestos
    gestureDetector = new GestureDetectorCompat( context: this, new GestureListener());

    setDatosPokemon(pokemonActual);
}
```

Código onCreate de la actividad Pokédex.

```java
/**
 * Cambia los datos de la vista para ajustarlos al pokeomon
 *
 * @param id id del pokemon para realizar la consulta
 */
public void setDatosPokemon(int id) {
    //Realizamos consulta
    SQLiteDatabase db = conn.getReadableDatabase();
    String consulta = String.format("SELECT * FROM %s WHERE %s=%d", NOMBRE_TABLA_POKEMON, CAMPO_POKEMON_ID, id);
    Cursor c = db.rawQuery(consulta,  selectionArgs: null);

    if (c.moveToFirst()) {
        //Establecemos los datos
        txtViewNombre.setText(c.getString(c.getColumnIndex(CAMPO_POKEMON_NOMBRE)));
        txtViewDescripcion.setText(c.getString(c.getColumnIndex(CAMPO_POKEMON_DESCRIPCION)));
        ivPokemon.setImageResource(UtilImagenes.getIdRecursoPokemon(c.getInt(c.getColumnIndex(CAMPO_POKEMON_ID))));
        ivTipo1.setImageResource(UtilImagenes.getIdRecursoTipo(c.getInt(c.getColumnIndex(CAMPO_POKEMON_TIPO_1))));
        txtViewAltura.setText("Altura: " + c.getString(c.getColumnIndex(CAMPO_POKEMON_ALTURA)));
        txtViewPeso.setText("Peso: " + c.getString(c.getColumnIndex(CAMPO_POKEMON_PESO)));
        txtViewCategoria.setText("Pokemon " + c.getString(c.getColumnIndex(CAMPO_POKEMON_CATEGORIA)));

        //Si solo hay un tipo invisibilizamos el otro
        int tipo2 = UtilImagenes.getIdRecursoTipo(c.getInt(c.getColumnIndex(CAMPO_POKEMON_TIPO_2)));
        if (tipo2 != Integer.MAX_VALUE) {
            ivTipo2.setImageResource(tipo2);
            ivTipo2.setVisibility(View.VISIBLE);
        } else {
            ivTipo2.setVisibility(View.GONE);
        }
    }

    db.close();
}
```

Método que realiza la consulta para cambiar los datos de la vista.

Moverse por la Pokédex al hacer gestos con el dedo.

Para que al deslizar el dedo a través de la pantalla avancemos o retrocedamos por la Pokédex he programado la clase GestureListener, esta clase extiende de SimpleOnGestureListener.

La clase SimpleOnGestureListener permite leer los gestos que realiza el usuario en la pantalla del teléfono, tales como toques, pulsaciones o deslizar el dedo por la pantalla.

El método onFling se lanza cuando se desliza el dedo por la pantalla. De izquierda a derecha o viceversa, el método nos devuelve una velocidad en el eje X. Si la velocidad es negativa supone que el dedo del usuario se ha desplazado de derecha a izquierda. Con este dato de la velocidad podemos pasarle al método de insertar datos el id que corresponde al Pokémon siguiente o al anterior según corresponda.

```
private class GestureListener extends GestureDetector.SimpleOnGestureListener {

    @Override
    //Cuando se hace un gesto se llama a este metodo
    public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
        //Comprobamos hacia que lado se hizo el gesto.
        if (velocityX < 0) {
            pokemonActual++;
            if (pokemonActual > 493) pokemonActual = 1;
        } else {
            pokemonActual--;
            if (pokemonActual < 1) pokemonActual = 493;
        }
        setDatosPokemon(pokemonActual);
        return super.onFling(e1, e2, velocityX, velocityX);
    }

}

@Override
public boolean onTouchEvent(MotionEvent event) {
    gestureDetector.onTouchEvent(event);
    return super.onTouchEvent(event);
}
```

Clase GestureListener.

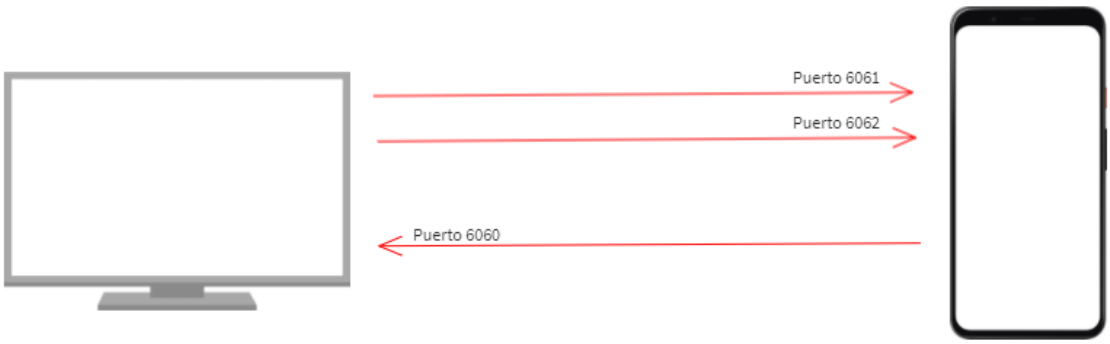
Implementación de la conexión por TCP, Servidores.

Para la comunicación entre ambas aplicaciones he usado TCP, este sistema me asegura que no se va a perder información ya que van a enviarse archivos entre ambos. He creado dos clientes (uno en cada dispositivo) y tres servidores:

Puerto 6060, Ordenador: Esta destinado a recibir las acciones que el usuario del Android realice, así como el inicio de la conexión.

Puerto 6061, Android: Es el puerto destinado a recibir el fichero XML que contiene la información de los Pokémon que se encuentran en combate.

Puerto 6062, Android: Es el puerto destinado a recibir mensajes, no ficheros, como los datos de lo que ha ocurrido durante el turno.



```java
@Override
public void run() {
    try {
        server = new ServerSocket(Configuracion.PUERTO_SERVIDOR);
        while (true) {
            //Aceptar conexiones
            connection = server.accept();

            //Buffer de 1024 bytes
            receivedData = new byte[1024];
            bufferedInputStream = new BufferedInputStream(connection.getInputStream());
            dataInputStream = new DataInputStream(connection.getInputStream());

            //Para guardar fichero recibido
            bufferedOutputStream = new BufferedOutputStream(new FileOutputStream(new Fil

            //Leer fichero recibido
            while ((in = bufferedInputStream.read(receivedData)) != -1) {
                bufferedOutputStream.write(receivedData,  off: 0, in);
            }

            bufferedOutputStream.close();
            dataInputStream.close();

            //Se realiza tras recibir el fichero:
            pokemonsLuchando = leerPokemonsDeXML();

            //Necesario para que no se produzca excepcion de cambiar vistas desde hilo
            runOnUiThread(() -> {
                    //Cambio botones
                    setDatosBotones();
                    setDatosPokemons();
                    txtViewEspera.setVisibility(View.GONE);
                    layoutCombate.setVisibility(View.VISIBLE);
            });

        }
    } catch (IOException e) {
        if (e.getMessage() != "Socket closed")
            System.out.println("ERROR SERVIDOR TCP: " + e.getMessage());
    }
}
```

Hilo del servidor con el puerto 6061, Android.

Cuando llegan mensajes que no son ficheros, el servidor de Android con el puerto 6062 crea el Toast que muestra el mensaje por pantalla.

```
//Trato el contenido
if (mensajeFinal.contains("END")) {
    linearBotonesMovimientos.setVisibility(View.VISIBLE);
} else {
    linearBotonesMovimientos.setVisibility(View.GONE);
    Toast.makeText(getApplicationContext(), mensajeFinal, Toast.LENGTH_LONG).show();
}
```

Código, dentro del hilo del servidor de mensajes, que crea el Toast.

Cuando el mensaje llega al móvil, este se guarda los datos de los Pokémon en un array de Pokémons. Luego, cambia los botones y el resto de la vista en tiempo de ejecución para adecuarlos a los datos recibidos.

```
/**
 * Cambia la vista para colocar los datos de los pokemon
 */
private void setDatosPokemons() {
    tvPokFront.setText(pokemonsLuchando[POKEMON_FRONT].getNombre());
    tvPokBack.setText(pokemonsLuchando[POKEMON_BACK].getNombre());
    ivPokFront.setImageResource(UtilImagenes.getIdRecursoPokemon(pokemonsLuchando[POKEMON_FRONT].getNumPokedex()));
    ivPokBack.setImageResource(UtilImagenes.getIdRecursoPokemon(pokemonsLuchando[POKEMON_BACK].getNumPokedex()));
    tvPokBackVida.setText(pokemonsLuchando[POKEMON_BACK].getVidaActual() + " / " + pokemonsLuchando[POKEMON_BACK].getVidaMax());
    tvPokFrontVida.setText(pokemonsLuchando[POKEMON_FRONT].getVidaActual() + " / " + pokemonsLuchando[POKEMON_FRONT].getVidaMax());
}
```

Código que cambia los datos de la vista en tiempo de ejecución.

```
/**
 * Setea los datos de los 4 botones.
 */
private void setDatosBotones() {
    setDatosBoton(btnMov1, pokemonsLuchando[POKEMON_BACK].getMov1());
    setDatosBoton(btnMov2, pokemonsLuchando[POKEMON_BACK].getMov2());
    setDatosBoton(btnMov3, pokemonsLuchando[POKEMON_BACK].getMov3());
    setDatosBoton(btnMov4, pokemonsLuchando[POKEMON_BACK].getMov4());
}

/** Cambia el boton para ajustarse a los datos del movimiento. ...*/
private void setDatosBoton(Button button, Movimiento movimiento) {
    //Texto del boton
    button.setText(movimiento.getNombre() + "    " + movimiento.getPp() + " / " + movimiento.getPpMax());

    //Icono del boton
    Drawable myDrawable;
    switch (movimiento.getTipo()) {...}
    myDrawable.setBounds( left: 0,  top: 0,  right: 110,  bottom: 110);
    button.setCompoundDrawables(myDrawable,  top: null,  right: null,  bottom: null);
}
```

Código que modifica los botones.

-Implementación Windows.

El servidor de la aplicación de ordenador se lanza siempre que se inicia un juego. Si este recibe datos, automáticamente, cambia el formulario del combate para transformarlo en multijugador, de igual forma si el usuario del Android sale de la aplicación, recibirá un mensaje para que el combate deje de ser multijugador.

Al recibir el mensaje del ataque realizado por el usuario del Android, selecciona dentro del formulario del combate dicha opción para que pueda dar comienzo el turno, el cual, se quedará bloqueado esperando a que ambos jugadores hayan elegido movimiento.

```
public static void Servidor(Form_Combate form_Combate, TcpListener server)
{
    try
    {
        server.Start();

        //Buffer
        byte[] bytes = new byte[256];
        string data = null;

        while (true)
        {
            TcpClient client = server.AcceptTcpClient();
            data = null;
            NetworkStream stream = client.GetStream();
            int i;

            //Recibir datos
            while ((i = stream.Read(bytes, 0, bytes.Length)) != 0)
            {
                data = Encoding.UTF8.GetString(bytes, 0, i);
                data = data.ToUpper();

                byte[] msg = Encoding.UTF8.GetBytes(data);

                //Respuesta.
                stream.Write(msg, 0, msg.Length);

                //Seleccionamos el tipo de accion dependiendo del dato recibido
                form_Combate.multiplayer = true;
                int ataque = 0;

                //Ataque
                if (int.TryParse(data.Substring(0, 1), out ataque))
                    form_Combate.ataqueSeleccionadoMultiplayer = ataque;

                //Salir multijugador
                if (data.ToUpper().Contains("STOP"))
                    form_Combate.multiplayer = false;
            }

            //Cerrar cliente
            client.Close();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("Excepcion Servidor: {0}", e);
    }
}
```

Servidor del ordenador.

Implementación de la conexión por TCP, Clientes.

El cliente del ordenador creará primero un XML con los datos de los dos Pokémon en combate para luego enviarlos al Android.

```
public static void CrearXMLDatosCombate(Pokemon pokUsuario, Pokemon pokRival)
{
    //Creo el archivo para escribir el XML append false para sobreescribir el archivo si ya existe
    StreamWriter file = new StreamWriter(RUTA_FICHERO_DATOS_POKEMON, append: false);
    file.WriteLine($"{CABECERA_XML}\n");

    file.WriteLine("\n<Pokemons>");

    CrearXMLDatosPokemon(pokUsuario, file);
    CrearXMLDatosPokemon(pokRival, file);

    file.WriteLine("\n</Pokemons>");
    file.Close();
}

2 referencias
private static void CrearXMLDatosPokemon(Pokemon pokemon, StreamWriter file)
{
    //Escribo el XML
    file.WriteLine("\n\t<Pokemon>\n");

    //Datos del pokemon
    file.WriteLine($"\t\t\t<IdPok>{pokemon.numAlmacenamiento}</IdPok>");
    file.WriteLine($"\t\t\t<NumPokedex>{pokemon.fkPokedex}</NumPokedex>");
    file.WriteLine($"\t\t\t<Nombre>{pokemon.nombre}</Nombre>");
    file.WriteLine($"\t\t\t<VidaMax>{pokemon.vidaMax}</VidaMax>");
    file.WriteLine($"\t\t\t<VidaAct>{pokemon.vidaActual}</VidaAct>\n");
    file.WriteLine($"\t\t\t<Estado>{(int)pokemon.estadisticasActuales.estadoActual}</Estado>\n");

    //Movimientos
    EscribirXMLAtaque(file, pokemon.mov1, 1);
    EscribirXMLAtaque(file, pokemon.mov2, 2);
    EscribirXMLAtaque(file, pokemon.mov3, 3);
    EscribirXMLAtaque(file, pokemon.mov4, 4);

    //Tipos del pokemon
    file.WriteLine($"\t\t\t<Tipo1>{(int)pokemon.tipo1}</Tipo1>");
    file.WriteLine($"\t\t\t<Tipo2>{(int)pokemon.tipo2}</Tipo2>");

    file.WriteLine("\n\t</Pokemon>");
}

4 referencias
private static void EscribirXMLAtaque(StreamWriter file, Ataque ataque, int numMov)
{
    if (ataque != null)
    {
        file.WriteLine($"\t\t\t<Mov{numMov}>");
        file.WriteLine($"\t\t\t\t<MovId>{(int)ataque.ataqueID}</MovId>");
        file.WriteLine($"\t\t\t\t<MovNombre>{ataque.nombre}</MovNombre>");
        file.WriteLine($"\t\t\t\t<MovTipo>{(int)ataque.tipo}</MovTipo>");
        file.WriteLine($"\t\t\t\t<PP>{(int)ataque.ppActuales}</PP>");
        file.WriteLine($"\t\t\t\t<PPmax>{(int)ataque.ppMax}</PPmax>");
        file.WriteLine($"\t\t\t</Mov{numMov}>\n");
    }
}
```

Código que escribe el fichero XML con los datos del combate.

```
<?xml version="1.0" encoding="UTF-8"?>
<Pokemons>
  - <Pokemon>
        <IdPok>784</IdPok>
        <NumPokedex>248</NumPokedex>
        <Nombre>Tyranitar</Nombre>
        <VidaMax>183</VidaMax>
        <VidaAct>183</VidaAct>
        <Estado>0</Estado>
      - <Mov1>
            <MovId>148</MovId>
            <MovNombre>Terremoto</MovNombre>
            <MovTipo>16</MovTipo>
            <PP>9</PP>
            <PPmax>10</PPmax>
        </Mov1>
      - <Mov2>
            <MovId>306</MovId>
            <MovNombre>Filo del Abismo</MovNombre>
            <MovTipo>16</MovTipo>
            <PP>10</PP>
            <PPmax>10</PPmax>
        </Mov2>
      - <Mov3>
            <MovId>202</MovId>
            <MovNombre>Triturar</MovNombre>
            <MovTipo>15</MovTipo>
            <PP>14</PP>
            <PPmax>15</PPmax>
        </Mov3>
      - <Mov4>
            <MovId>172</MovId>
            <MovNombre>Enfado</MovNombre>
            <MovTipo>4</MovTipo>
            <PP>10</PP>
            <PPmax>10</PPmax>
        </Mov4>
        <Tipo1>14</Tipo1>
        <Tipo2>15</Tipo2>
    </Pokemon>
  - <Pokemon>
        <IdPok>835</IdPok>
        <NumPokedex>171</NumPokedex>
        <Nombre>Lanturn</Nombre>
        <VidaMax>208</VidaMax>
        <VidaAct>208</VidaAct>
        <Estado>0</Estado>
      - <Mov1>
            <MovId>264</MovId>
            <MovNombre>Rayo Carga</MovNombre>
            <MovTipo>5</MovTipo>
            <PP>10</PP>
            <PPmax>10</PPmax>
        </Mov1>
      - <Mov2>
            <MovId>173</MovId>
            <MovNombre>Chispa</MovNombre>
            <MovTipo>5</MovTipo>
            <PP>20</PP>
            <PPmax>20</PPmax>
        </Mov2>
      - <Mov3>
            <MovId>7</MovId>
            <MovNombre>Refugio</MovNombre>
            <MovTipo>2</MovTipo>
            <PP>40</PP>
            <PPmax>40</PPmax>
        </Mov3>
      - <Mov4>
            <MovId>8</MovId>
            <MovNombre>Surf</MovNombre>
            <MovTipo>2</MovTipo>
            <PP>15</PP>
            <PPmax>15</PPmax>
        </Mov4>
        <Tipo1>2</Tipo1>
        <Tipo2>5</Tipo2>
    </Pokemon>
</Pokemons>
```

XML Resultante.

El XML con los datos se envía al móvil mediante el siguiente código, de esta forma el servidor Android lo trata para mostrar los datos.

```
public static void EnviarArchivoTCP(string rutaArchivo)
{
    try
    {
        byte[] sendingBuffer;

        //Abrimos el cliente
        TcpClient client = new TcpClient(IP_MOVIL, PUERTO_ENVIAR);
        NetworkStream netStream = client.GetStream();

        //Abrimkos el archivo a enviar
        FileStream fileStream = new FileStream(rutaArchivo, FileMode.Open, FileAccess.Read);

        //Vamos leyeendo el archivo para enviarlo
        int numPaquetes = Convert.ToInt32(Math.Ceiling(Convert.ToDouble(fileStream.Length) / Convert.ToDouble(TAM_BUFFER)));
        int tamagnoTotal = (int)fileStream.Length, tamPaquete;
        for (int i = 0; i < numPaquetes; i++)
        {
            if (tamagnoTotal > TAM_BUFFER)
            {
                tamPaquete = TAM_BUFFER;
                tamagnoTotal -= tamPaquete;
            }
            else
                tamPaquete = tamagnoTotal;

            sendingBuffer = new byte[tamPaquete];
            fileStream.Read(sendingBuffer, 0, tamPaquete);

            //Enviamos datos leidos
            netStream.Write(sendingBuffer, 0, sendingBuffer.Length);
        }

        //Cerramos todo
        fileStream.Close();
        netStream.Close();
        client.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine("ERROR TCP: " + ex.Message);
    }
}
```

Este código envía el fichero XML siempre que es necesario actualizar la información del dispositivo móvil, esto es, cuando se conecta al combate, cuando acaba un turno, cuando un jugador debe cambiar de Pokémon y al comenzar el combate.

```
1 referencia
private void EnviarDatosMultijugador()
{
    //Creamos los datos para enviar al otro jugador
    XML.CrearXMLDatosCombate(pokemonBack, pokemonFront);
    TCP.EnviarArchivoTCP(XML.RUTA_FICHERO_DATOS_POKEMON);
}
```

Código llamado para enviar los datos.

Para los mensajes normales (no ficheros) que muestran lo que ha ocurrido durante el turno mediante el Toast en Android. Se usa este código:

```
2 referencias
public static void EnviarMensajeTCP(string mensaje)
{
    try
    {
        byte[] sendingBuffer;

        //Abrimos el cliente
        TcpClient client = new TcpClient(IP_MOVIL, PUERTO_ENVIAR_MENSAJE);
        NetworkStream netStream = client.GetStream();

        //Enviamos el mensaje
        sendingBuffer = Encoding.UTF8.GetBytes(mensaje);
        netStream.Write(sendingBuffer, 0, sendingBuffer.Length);

        netStream.Close();
        client.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine("ERROR TCP: " + ex.Message);
    }
}
```

Se envían estos mensajes cuando acaba un subturno, esto es, cada una de las veces que ocurre algo en un turno.

Pruebas.

Para la realización de las pruebas de la aplicación he usado:

- Para probar la parte de Windows (servidor) he usado mi ordenador personal, un PC de sobremesa, Windows 10, con un I7 9700K y 16GB de RAM.
- Para probar el cliente (Android) usé un Xiaomi Mi A2 Lite.

Para las pruebas de funcionamiento necesito comprobar:

1- Conexiones entre Android y Windows funcionan correctamente, tanto los servidores como los clientes.
2- Los datos de los Pokémon en combate se escriben correctamente en los XML pertinentes.
3- Los datos del XML generado se leen correctamente por el Android.
4- Los datos de los Pokémon corresponden con la parte grafica mostrada.
5- En la base de datos de Android se cargan correctamente, de forma ordenada por el numero de la Pokédex.
6- Los datos de la base de datos se cargan correctamente en las actividades, pudiendo visualizar los datos del Pokémon correspondiente y cambiando entre los Pokémon al hacer gestos sobre la pantalla.
7- Los mensajes del móvil se leen correctamente por la aplicación de Windows y hacen las modificaciones pertinentes (poner el juego en multijugador y quitarlo cuando el móvil se desconecta).
8- Los ataques seleccionados en el dispositivo móvil se realizan en la aplicación Windows.
9- Los mensajes del Windows se reciben y muestran correctamente en el Android.

Todas las pruebas mencionadas, fueron comprobadas con éxito durante la realización de 3 combates completos, utilizando Pokémon con distintos movimientos y tipos.

He necesitado hacer también pruebas especificas para el funcionamiento de la aplicación durante el combate:

1- El dispositivo Windows envía los datos del combate en el momento en el que el Android realiza la conexión.

2- El dispositivo Windows envía los datos del combate en el momento en el que se debilita uno de los Pokémon de los jugadores.
3- El dispositivo Windows envía los datos del combate en el momento en el que se cambia uno de los Pokémon de los jugadores.

Los XML generados por el proyecto de Windows fueron validados en xmlvalidation.com, pasando correctamente las comprobaciones para tres casos distintos tomando en cuenta distintos Pokémon con distintas habilidades.

Los datos de entrada del usuario como la introducción de la IP del Servidor en la aplicación móvil (que es el único dato que pedimos al usuario). Es validado mediante un método con un patrón que comprueba que el dato introducido sea una IPv4 coherente, es decir, del tipo ([1-255].[1-255].[1-255].[1-255]).

```
/**
 * Es ip
 *
 * @param ip ip introducida por el usuario
 * @return true si es una ip valida, false en caso contrario
 */
private boolean esIP(String ip) {
    Pattern pattern = Pattern.compile("^(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.){3}([01]?\\d\\d?|2[0-4]\\d|25[0-5])$");
    return pattern.matcher(ip).matches();
}
```

Código que comprueba si la IP introducida es coherente.

-Bibliografía.

Video comunicación TCP entre Android y NetBeans:

https://www.youtube.com/watch?v=7QNJvxXCYOY&ab_channel=ProgrammingExperts

Comunicación por TCP C#:

https://www.luisllamas.es/comunicacion-tcp-con-c/

Pagina Comunicación TCP archivos C#:

https://www.codeproject.com/Articles/32633/Sending-Files-using-TCP

Transferir archivos Android:

https://www.programacion.com.py/escritorio/java-escritorio/transferencia-de-fichero-por-socket-en-java

Leer XML en Android:

https://josehernandez.es/2012/07/30/leer-xml-android.html

Acceder a archivos en Android:

https://developer.android.com/training/data-storage/app-specific?hl=es-419

Resolver error al acceder a archivos:

https://es.stackoverflow.com/questions/236130/android-studio-open-failed-erofs-read-only-file-system

Resolver error cambiar vistas desde hilo en Android:

https://stackoverflow.com/questions/5161951/android-only-the-original-thread-that-created-a-view-hierarchy-can-touch-its-vi

Resolver error icono botón en Android:

https://stackoverflow.com/questions/6590838/calling-setcompounddrawables-doesnt-display-the-compound-drawable

Navegación con gestos en Android:

https://www.youtube.com/watch?v=grxRWLEDf7c&ab_channel=Stevdza-San

Wikipedia Pokémon:

https://es.wikipedia.org/wiki/Pok%C3%A9mon

Definición dirección IP:

https://blog.orange.es/consejos-y-trucos/que-es-direccion-ip-y-que-tienes-que-saber-sobre-la-tuya/