Uso de sensores en Android – Aplicación SMAC (**Sm**art **Ac**tivities)

Escrito por: Pablo García López

Índice

1.- Introducción

En este documento vamos a explicar como utilizar los distintos sensores que tienen los dispositivos Android. Se va a tratar como utilizar el acelerómetro, el GPS, como obtener el tiempo meteorológico, el sensor de proximidad y el multitouch.

Esto se ha llevado a cabo mediante la implementación de la aplicación SMAC, una aplicación que busca ayudar en la realización de rutas de senderismo, rutas en bicicleta o a caballo. Ofrece distintas funcionalidades como puede ser:

- La obtención del tiempo meteorológico en los próximos 5 días de la zona a la que vamos a acudir.
- La visualización previa y una vez estemos en la ruta del mapa con los distintos marcadores a seguir.
- La planificación de descansos: Durante la ruta en cuestión podemos sacudir el teléfono de un lado a otro y se activará una alarma que nos avisará un tiempo después para continuar la ruta.
- Llamada de emergencia: Si tapamos la parte superior del teléfono durante al menos 5 segundos se realizará una llamada al teléfono de emergencias o al teléfono que indiquemos a la aplicación, de modo que si surge cualquier imprevisto durante el trascurso del camino podamos obtener ayuda rápidamente si fuera necesario.
- Vuelta a casa: Si durante la sesión el usuario desea dar por finalizada la ruta que esta llevando a cabo, basta con tocar la pantalla con al menos 3 dedos, lo cual actualizará el mapa de ruta invirtiéndolo y por tanto guiando al usuario de nuevo al punto de inicio de la ruta, para que pueda volver a casa si así lo desea.

Este proyecto ha sido llevado a cabo utilizando Android Studio y el lenguaje de programación Java, por lo que esta memoria va a ser descrita en base a estos requisitos.

2.- Previsión meteorológica

En este apartado vamos a tratar como obtener la previsión meteorológica, que es utilizado en la planificación de actividades para ofrecer al usuario en la propia aplicación una información que consideramos muy importante a la hora de preparar una ruta. La aplicación proporciona al usuario información relativa al tiempo en el día actual y en los próximos 4 días, incluyendo temperatura máxima, mínima, velocidad del viento y tiempo esperado (sol, lluvia, tormenta, etc.). Esto se lleva a cabo mediante una llamada a la API OpenWeatherMap, que nos ofrece una gran cantidad de información relativa al estado meteorológico de un lugar, de la cual obtenemos los apartados que nos interesan para nuestra aplicación.

Para comenzar, debemos registrarnos en la web de OpenWeatherMap (https://openweathermap.org) para obtener una key con la que poder utilizar la API. Debemos implementar las siguientes clases Java para llevar a cabo la funcionalidad descrita anteriormente:

2.1.- Clase Weather

Clase auxiliar utilizada para recoger el tipo de dato Weather, que contiene la temperatura máxima, temperatura mínima, velocidad del viento y el icono asociado al tiempo. Implementa los getter y setter de todas estas variables.

Este tipo de dato se utiliza en JSONWeatherParser para, de los datos del tiempo obtenidos como objeto JSON, extraer los necesarios e introducirlos en el dato Weather, y en MainActivity, para poder mostrar los datos asociados al tiempo de cada día en la pantalla principal de nuestra aplicación.

2.2.- Clase WeatherHttpClient

Esta clase se utiliza para obtener los datos del tiempo de la API OpenWeatherMap.

Contiene solo un atributo, BASE_URL que refleja la URL a la cual se va a realizar la conexión. En esta URL están también los parámetros de la solicitud, como puede ser la latitud y longitud del sitio a buscar, las unidades o la key de la API.

Esta clase contiene solamente un método, getWeatherData que como su nombre indica se utiliza para obtener los datos meteorológicos de la API. Primero crea dos variables, una del tipo HttpURLConnection y otra del tipo InputStream. A continuación, parametriza la conexión HTTP añadiéndole la URL (obtenida de BASE_URL) y que va a realizar una solicitud GET. Una vez realizada la conexión, se leen los datos de la misma mediante un StringBuffer. Al finalizar, se cierra el InputStream y se desconecta de la HTTPURLConnection, devolviendo el buffer como un String. Si se produce algún error en este método, devuelve null.

2.3.- Clase JSONWeatherParser

Clase utilizada para, una vez tenemos los datos del tiempo como un String, crea un JSONObject asociado a los datos y los manipula para extraer los datos utilizados en nuestra aplicación y devolverlos como un Array de 5 elementos de tipo Weather.

Esta clase no tiene ningún atributo, pues solo se utiliza para convertir los datos y no necesita almacenarlos, simplemente una vez los obtiene los devuelve a MainActivity (desde donde se llama a esta clase). Respecto a los métodos, podemos destacar:

- getObject, getString, getFloat y getInt: Se utilizan como funciones auxiliares para, dentro de esta misma clase, manipular el JSONObject y poder extraer los distintos tipos de datos de él.

- getWeather: Utilidad principal de la clase. Recibe como parámetros un String con los datos del tiempo obtenidos de la API y devuelve un array de tipo Weather de 5 elementos. Su funcionamiento comienza con la creación del array de weather y del JSONObject asociado a los datos. A continuación, crea un JSONArray del objeto anterior, tomando solo la parte del objeto (array) etiquetado como “daily” (esto ya es un array con el tiempo de todos los días, eliminando los datos que no vamos a utilizar). Entra en un bucle for para, de cada día extraer del JSONArray los datos (temperatura, viendo y código del icono) utilizando una sucesión de funciones implementadas en los JSONObject y JSONArray que permiten manipular estos datos

y obtener solo los valores requeridos.

Con estas clases implementadas, podemos obtener el tiempo con la siguiente función:

```
protected Weather[] doInBackground(String... params) {
    Weather weather[] = new Weather[5];
    String data = ((new WeatherHttpClient()).getWeatherData());
    try {
        weather = JSONWeatherParser.getWeather(data);
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return weather;

}
```

Como podemos ver, creamos un array de 5 elementos de tipo Weather, creamos un String con la información que nos devuelve la llamada a getWeatherData() de WeatherHttpClient y de ese String, seleccionamos las partes que nos interesan mediante la función getWeather() de la clase JSONWeatherParser. Teniendo esto ya podemos mostrar los datos en la aplicación, como podemos ver en la siguiente imagen:

3.- GPS – Google maps

Para comenzar a utilizar los servicios de Google maps en nuestra aplicación, debemos obtener una clave de la API e introducirla en el archivo gradle.properties del proyecto en Android Studio, en la propiedad GOOGLE_MAPS_API_KEY. Una vez hecho esto, en el método onCreate() de la actividad en la que queremos el mapa debemos obtener un controlador para el fragmento de mapa y registrar esto para la devolución de la llamada al mapa cuando este esté disponible, con las siguientes dos líneas de código:

```
SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
        .findFragmentById(R.id.map);
mapFragment.getMapAsync( onMapReadyCallback: this);
```

En la misma actividad en la que hemos hecho esto, debemos sobrecargar la función onMapReady() de la siguiente manera:

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;

    //Sacar las posiciones iniciales y finales del objeto Ruta
    LatLng posIni = ruta.getWaypoints().get(0);
    //LatLng posFin = ruta.getWaypoints().get(ruta.getWaypoints().size() - 1);
    Polyline line = mMap.addPolyline(new PolylineOptions()
            .width(5)
            .color(Color.RED));
    //Pintar un marker para cada waypoint
    for(int i=0; i<ruta.getWaypoints().size(); i++)
    {
        mMap.addMarker(new MarkerOptions().position(ruta.getWaypoints().get(i)).title("Marker "+ i));

    }
    line.setPoints(ruta.getWaypoints());

    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(posIni,  v: 12));

}
```

Esta recibe un parámetro de tipo GoogleMap con el mapa ya listo, el cual debemos modificar para adecuarlo a nuestros requisitos. Pintamos una línea (para el recorrido) con la función addPolyline(), pintamos un marcador para cada punto de la ruta con la función addMarker() y finalmente situamos la cámara en el punto de inicio de la ruta con la función moveCamera(), pasándole como parámetros la latitud y longitud del primer marker y un zoom de 12.

4.- Acelerómetro y sensor de proximidad

En este apartado vamos a ver como utilizar el sensor de proximidad y el acelerómetro en una aplicación Android. Como el uso de todos ellos es similar, vamos a explicar como se utiliza el acelerómetro (ya que el resto son análogos) y posteriormente explicaremos la utilización concreta de cada uno de ellos en la aplicación.

Por claridad en la implementación, el uso del acelerómetro se ha recogido en una clase Accelerometer, la cual podemos observar en la siguiente imagen:

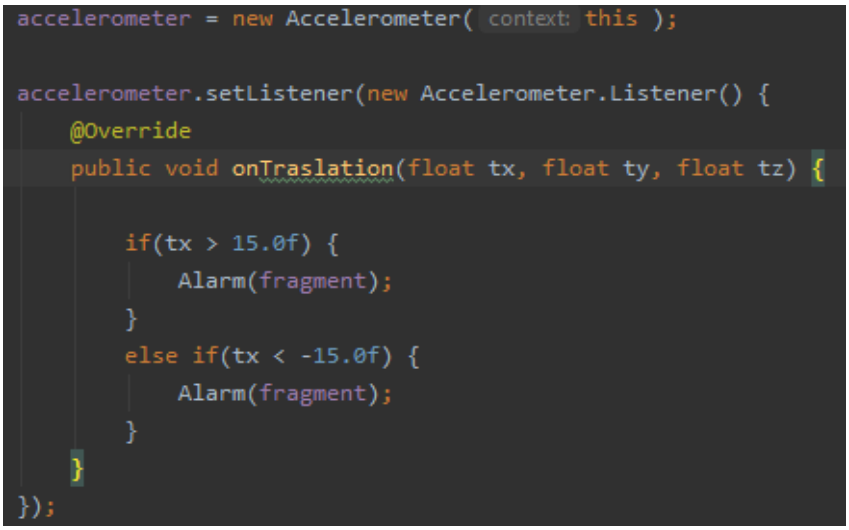
```
public class Accelerometer {

    public interface Listener{
        void onTraslation(float tx, float ty, float tz);
    }

    private Listener listener;

    public void setListener(Listener l) { listener = l; }

    private SensorManager sensorManager;
    private Sensor sensor;
    private SensorEventListener sensorEventListener;

    Accelerometer(Context context){
        sensorManager = (SensorManager) context.getSystemService(context.SENSOR_SERVICE);
        sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        sensorEventListener = new SensorEventListener() {
            @Override
            public void onSensorChanged(SensorEvent event) {
                if(listener != null){
                    listener.onTraslation(event.values[0], event.values[1],event.values[2]);
                }
                else
                {
                    System.out.println(1);


                }
            }

            @Override
            public void onAccuracyChanged(Sensor sensor, int accuracy) {


            }
        };
    }
    public void register(){
        sensorManager.registerListener(sensorEventListener,sensor,SensorManager.SENSOR_DELAY_NORMAL);
    }
    public void unregister() { sensorManager.unregisterListener(sensorEventListener); }
}
```

De esta clase destacamos:

- SensorManager: Crea una instancia del servicio del sensor del sistema, que podemos usar para acceder a los sensores del sistema.

- Sensor: Crea una instancia de un sensor específico. Se llama al método getDefaultSensor() con el tipo de sensor TYPE_ACCELEROMETER para obtener una referencia al sensor de tipo acelerómetro por defecto del sistema.

- SensorEventListener: Interfaz que implementa la función onSensorChanged(), que es llamada cada vez que el sensor capta un valor nuevo (un objeto del tipo SensorEvent). A su vez llama a la función onTraslation() (de la interfaz Listener), que recibe como parámetros tx, ty y tz, los valores de aceleración que registra el sensor en cada uno de los ejes. Este método será sobrecargado cuando queramos utilizar la clase Accelerometer, dando la funcionalidad concreta que queramos realizar. En nuestro caso, se ha realizado de la siguiente manera:



```
accelerometer = new Accelerometer( context: this );

accelerometer.setListener(new Accelerometer.Listener() {
    @Override
    public void onTraslation(float tx, float ty, float tz) {

        if(tx > 15.0f) {
            Alarm(fragment);
        }
        else if(tx < -15.0f) {
            Alarm(fragment);
        }
    }
});
```

Se crea una nueva instancia de la clase Accelerometer y se llama a su función setListener() para fijar un nuevo Listener que sobrecargue la función onTranslation(). Esta implementación comprueba que tx, es decir, la aceleración que registra el sensor en el eje x (movimiento del teléfono lateralmente) sea mayor que 15.0f o menor que -15.0f, en cuyo caso llama a la función Alarm() (función que establece una alarma dentro de una hora). Los valores 15.0f y -15.0f determinan la sensibilidad a la que va a trabajar el sensor, así que tocando estos valores podemos hacer que el evento sea recogido con movimientos más o menos bruscos (por ejemplo, si aumentamos el valor a 20, será necesario un movimiento de mayor intensidad para que el sensor lo reconozca).

El sensor de proximidad se implementa y utiliza de una manera similar, aunque respecto a su utilización podemos destacar una pequeña diferencia. En muchos teléfonos, este sensor no trabaja como el de proximidad por ejemplo que mide un desplazamiento, si no que solo puede tomar dos valores, si el objeto está más cerca que cierto umbral, se determina que el objeto esta cerca y si esta más alejado que el umbral, se determina como lejos. Por esto, no podemos comparar los datos proporcionados por el sensor con un cierto valor, si no que debemos comprobar que el valor tomado sea inferior al máximo rango del sensor, en cuyo caso el sensor se activará y llamará a la función EmergencyCall(), como podemos observar a continuación:

```
@Override
public void onSensorChanged(SensorEvent event) {
    if(event.values[0] < sensor.getMaximumRange()){
        EmergencyCall(fragment);
    }
}
```

5.- MultiTouch

Para utilizar la característica Multitouch del teléfono, debemos sobrecargar la función onTouchEvent() como podemos ver en la siguiente imagen:

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    if(event.getPointerCount()>2 && !multione){
        multione = true;
        ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, requestCode: 1);
        if(ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED){
            locManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
            loc = locManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
        }
        LatLng inicio = new LatLng(loc.getLatitude(), loc.getLongitude());
        LatLng fin = ruta.getWaypoints().get(0);
        ArrayList<LatLng> nuevaR = new ArrayList<~>();
        nuevaR.add(inicio); nuevaR.add(fin);
        ruta.setWaypoints(nuevaR);
        iniciarLocalizacion();

    }
    return true;
}
```

Esta función tiene un parámetro del tipo MotionEvent, que contiene las características del evento concreto que ha provocado que la función se active. Por esto, al principio de nuestra función llamamos a la función getPointerCount() de MotionEvent, que nos devuelve el número de puntos en los que se ha tocado la pantalla y si este es mayor de 2, realiza su funcionalidad, obligando así a que esta función solo tenga en cuenta las pulsaciones de pantalla con al menos 3 dedos.

La acción a realizar ha sido deshacer la ruta anterior y crear una ruta nueva desde nuestra posición al lugar de inicio, es decir, dar por concluida nuestra ruta y volver.

Para realizarlo en primer lugar comprobamos que tenemos los permisos necesarios, y tras

esto, guardamos en una variable nuestra última localización mediante la función getLastKnownLocation. Ahora simplemente creamos las variables inicio y fin, siendo inicio la localización que acabamos de guardar, y fin el WayPoint 0 de nuestra ruta.

6.- Conclusiones

En este documento hemos podido observar como trabajar con distintos sensores que llevan hoy en día una gran parte de los teléfonos Android del mercado. Como hemos visto, trabajar con ellos no es demasiado complicado y son muy parecidos de utilizar todos ellos entre si (tanto los comentados en este documento como otros sensores que llevan los teléfonos, como pueden ser sensor de luminosidad, de temperatura, de gravedad, de campo eléctrico...). Por esta razón, se puede ampliar la funcionalidad de muchas de las aplicaciones del mercado de manera relativamente sencilla y proporcionando al usuario una gran mejora en su comodidad, ya que con gestos tan sencillos como mover el teléfono, rotarlo o tapar su parte frontal se pueden realizar una gran cantidad de acciones que simplifiquen la interfaz de las mismas.

Además, como podemos observar en nuestra aplicación, dando uso a estos sensores podemos ayudar a nuestros usuarios en situaciones en las que no puedan utilizar de manera normal su teléfono, como puede ser en mitad de una ruta de senderismo que es más cómodo tocar el teléfono con 3 dedos que navegar por una serie de menús o que simplemente con tapar el teléfono se produzca una llamada de emergencia, ya que si se ha producido algún accidente creemos que es más útil esto que salir de la aplicación, acceder al teléfono y marcar un número al que llamar.

En general, los sensores son una parte muy útil de los dispositivos móviles que utilizan para realizar distintos controles internos, pero que si los programadores comenzaran a utilizarlos más dentro de sus aplicaciones, se podría mejorar en gran medida la comodidad de la gente que quiera utilizarlos, sin ello perjudicar a la gente que no esté interesada en esto, sin suponer además una gran carga extra de trabajo.