TP2 : Les capteurs
Programmation mobile

Mohamad Satea Almallouhi - Tony Nguyen
M1 Génie Logiciel
Faculté des Sciences
Université de Montpellier.

5 mars 2024

Résumé

Nous avons réalisé une application Android en Java afin de démontrer l'utilisation des capteurs intégré.

Table des matières

Faire une vidéo, le rapport avec des screenshot des résultats et du code et enfin un read.md(instruction). En plus, pour le bonus, faire une belle application, des tests unitaires, utiliser Kotlin, faire le rapport en Latex.

Introduction

Dans ce TP, nous allons l'utilisation des capteurs intégré dans nos smartphones.

Nous allons voir comment manipuler les différents types de capteur comme le GPS, la boussole, le gyroscope, etc ...

Pour réaliser ce tp, nous choisisons de créer une unique application avec un écran d'acceuil *(ChoseApplication.java)* qui mène aux différentes activités correspondant à chaque exercice.

Les sections du rapport suit les exercices.

Démonstration

En ligne sur Youtube, à l'adresse URL https://youtu.be/nQUkpSUjJlY une démonstration vidéo de notre travail.

1 Liste des capteurs *(Index)*

Lorsque l'on déploie une application sur un smartphone, nous ne pouvons pas être sûr des capteurs que l'on aura a notre disposition. Dans cette section, nous allons regarder tout les capteurs disponible sur un appareil donné.

Tout d'abord, il est nécessaire d'accèder au sensorManager par lequel les informations ainsi que l'accès aux capteurs passent.

```
setContentView(R.layout.activity_list);
//Récupere une instance du service de gestion des capteurs
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
listSensor();

ListView listView = findViewById(R.id.sensorListView);
ArrayAdapter<String> adapter = new ArrayAdapter<>(
        context: this,
        android.R.layout.simple_list_item_1,
        displayArray);
listView.setAdapter(adapter);
```

Ensuite, une fois qu'on a le sensorManager, nous pouvons faire appel à la fonction **getSensorList(Sensor.TYPE_ALL)** qui nous renvoie un tableau de tous les capteurs.

Pour accéder individuellement aux informations de chaque capteur représenté par un objet de type Sensor, nous invoquons les méthodes **getName(), getType() et getVersion()**.

```
private void listSensor() {
    //Lister les capteurs :
    sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
    displayArray = new String[sensors.size()];
    int i=0;
    for (Sensor sensor : sensors) {
        displayArray[i]="";
        displayArray[i] += "Name: " + sensor.getName() +
                "\r\n"+"\tType: " +
                getType(sensor.getType()) +
                "\r\n"+"\tVersion: " +
                sensor.getVersion();
        i++;
    }
```

Enfin, pour afficher à l'écran le résultat, nous utilisons une Vue de type ListView que nous avons au préalable déclarer dans le layout.

```xml
<LinearLayout xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".Index">

    <ListView
        android:id="@+id/sensorListView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

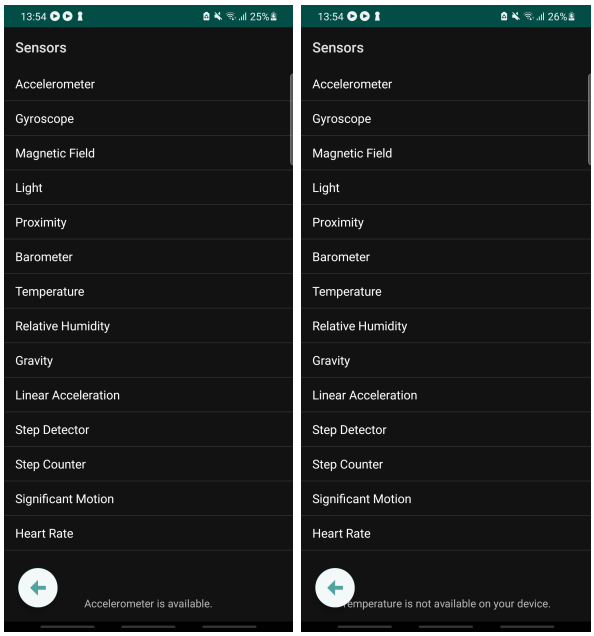
Dans ce nouvel exercice, nous allons affiché la disponibilité *(en bas de l'écran)* de un capteur en particulier.

Pour cela, nous réutilisons un ListView où chaque élément correspondra à un type de capteur. Puis nous redéfinissons la méthode **onItemClick()** pour modifier le comporter "cliquer" sur chaque item de la vue. La fonction nous donne en argument quelle item a été cliquer dessus sous forme d'un numéro d'index. On récupère le capteur qui correspond à cet index puis regarde si ce capteur est dispobile ou pas. C'est la fonction **getDefaultSensor()** qui nous inque qu'un capteur est indisponible quand il renvoie null.

```java
sensorListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    SateaMali
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        String selectedSensor = sensorTypes[position];
        checkSensorAvailability(selectedSensor);
    }
});
```

```java
private void checkSensorAvailability(String sensorName) {
    int sensorType = sensorTypeMap.get(sensorName);
    Sensor sensor = sensorManager.getDefaultSensor(sensorType);
    if (sensor != null) {
        sensorStatusTextView.setText(sensorName + " is available.");
    } else {
        sensorStatusTextView.setText(sensorName + " is not available on your device.");
    }
}
```

2 Détection de la disponibilité des capteurs *(SensorAvailibility)*



3 Accéléromètre *(Accelerometer)*

Nous avons fait une petite application qui change la couleur en fond en fonction de l'accélération du téléphone.

Nous commençons par récupérer le sensorManager et le capteur accéléromètre. Afin de récupérer les données du capteur, il nous suffit de s'enregistrer (avec **registerListener()**)auprès du service comme écouteurs d'événements et implémenter les méthodes de callback **onSensorChanged() et onAccuracyChanged()**.

Quand le capteur détecte un changement, la méthode **onSensorChanged()** est appelé. Si une accélération suffisament grande est détecté, on change la couleur du fond d'écran en rouge ou noir avec **view.setBackgroundColor()**.

```java
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
sensor_Acc = sensorManager.getDefaultSensor(TYPE_ACCELEROMETER);

sensorManager.registerListener( listener: this, sensor_Acc, SensorManager.SENSOR_DELAY_UI);
```

```java
@Override
public void onSensorChanged(SensorEvent event) {
    // Check if the sensor changement is from the accelerometer
    if (event.sensor.getType()== TYPE_ACCELEROMETER){ onAccelerometerChanged(event); }
}
1 usage   ± SateaMall
private void onAccelerometerChanged(SensorEvent event) {
    // Get accelometer values
    float[] values = event.values;
    float x = values[0];
    float y = values[1];
    float z = values[2];

    float magnitude = (float) Math.sqrt(x * x + y * y + z * z) / SensorManager.GRAVITY_EARTH;

    if (magnitude < THRESHOLD_MEDIUM) {
        // Low values, set background to green
        view.setBackgroundColor(Color.GREEN);
    } else if (magnitude < THRESHOLD_HIGH) {
        // Medium values, set background to black
        view.setBackgroundColor(Color.BLACK);
    } else {
        // High values, set background to red
        view.setBackgroundColor(Color.RED);
    }

}
```

4 Direction *(Direction)*

De façon très similaire à la section précédente, nous récupérons les capteurs qui nous intéresse et nous les écoutons.

De même on implémente la méthode onSensorChanged afin d'implémenter le comportement que l'on souhaite. Lors d'un mouvment nous affichons dans quelle direction le smartphone se déplace.

Nous avons déclarer une balise TextView dans le layout correspondant à cette activité. On le récupère dans le code avec **directionTextView = findViewById(R.id.directionTextView);** puis on change le texte qu'il affiche avec du code : **directionTextView.setText("Left");**

5 Secouer un appareil *(ShakeDevice)*

Cette activité *(ShakeDevice)* nous permet de faire basculer le flash entre allumé/éteint quand on second le smartphone.

Nous procédons comme dans les exercices précédent. On récupère le sensorManager. On récupère le capteur. On s'enregistre auprès de lui. On implémente les méthodes de callback. Si le smartphone est suffisament secoué, on fait basculer l'état du flash comme sur la capture d'écran suivante :

```java
private void toggleFlashlight() {
    isFlashlightOn = !isFlashlightOn;
    try {
        cameraManager.setTorchMode(cameraId, isFlashlightOn);
    } catch (CameraAccessException e) {
        e.printStackTrace();
    }
}
```

6 Proximité *(Proximity)*

Nous affichons une image indiquant si l'objet est proche ou loin.

On récupère le sensorManager, le capteur et on s'enregistre auprès de lui encore une fois. On implémente les méthodes de callback. On déclare une <**ImageView**> dans le layout *(activity_proximity.xml)* qu'on mainupulera ensuite en java avec la mathode **setImageRessource()**.

```java
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_PROXIMITY) {
        if (event.values[0] <2.0f) {
            // Object is near
            imageView.setImageResource(R.drawable.near_image); // Replace with your 'near' image
        } else {
            // Object is far
            imageView.setImageResource(R.drawable.far_image); // Replace with your 'far' image
        }
    }
}
```

7 Géolocalisation *(Geolocation)*

Dans de nombreuse application, il est souhaitable de connaitre la position de l'utilisateur. Pour ce faire, nous allons manipuler le gps.

8 Extra

8.1 Icone

Nous avons choisit de modifier l'îcone de notre application. Nous allons maintenant expliqué comment nous cela a été fait.

8.2 Retour

Nous avons placer un boutton de retour à l'écran d'acceuil.

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/my_button"
    android:contentDescription="Home"
    app:srcCompat="@drawable/sharp_arrow_circle_left_24"
    app:maxImageSize="67dp"
    app:tint="#EEFFFFFF"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginStart="20dp"
    android:layout_marginBottom="20dp"
    />
```

Dans le layout, nous ajoutons un boutton flottant (*FloatingActionButton*) dans le coin inférieur gauche de la vue

```
myButton = findViewById(R.id.my_button);
SateaMall
myButton.setOnClickListener(new View.OnClickListener() {
    SateaMall
    @Override
    public void onClick(View view) {
        Intent intent = new Intent( packageContext: ShakeDevice.this, ChoseApplication.class);
        startActivity(intent);
    }
});
```