

PROGRAMMATION MOBILE **ANDROID**

TD 3 **CAPTEURS**

OBJECTIFS



- Lire les informations en provenance des capteurs
- Travailler avec des permissions spéciales
- Géolocaliser le périphérique

EXERCICE 1.1 - NIVEAU DE CHARGE DE LA BATTERIE

Certaines applications ont besoin d'un niveau de charge suffisant pour activer des fonctionnalités. Par exemple, l'application appareil photo n'activera pas le flash si votre batterie est jugée trop faible.

Principe

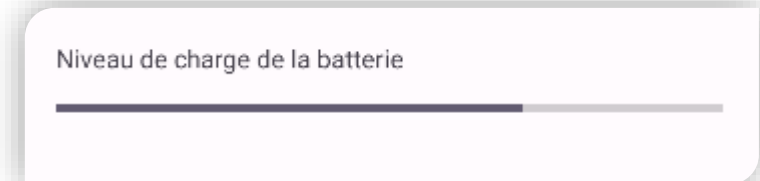
Un activité système scanne régulièrement le niveau de la batterie et transmet son niveau de charge à toutes les applications qui se sont abonnées à cette information (broadcasting)

Nouveau projet

- Créez un nouveau projet selon la méthode vue lors du TD 1.

Interface

- Reproduisez l'interface ci-contre à l'aide des composants suivants :
 - TextView pour le libellé
 - ProgressBar pour la barre indiquant le niveau de charge



EXERCICE 1.2 - ATTRIBUTS

Nouveaux attributs

- Ajoutez à **MainActivity** deux attributs **batteryLevel** et **batteryMaxLevel** de type **Int** initialisés à **0**.
- Ajoutez également un attribut **batteryReceiver** de type **BroadcastReceiver** comme indiqué ci-dessous.

```
private lateinit var batteryReceiver: BroadcastReceiver
```

Notes :

- **batteryMaxLevel** indique la valeur que prendra **batteryLevel** pour un niveau de charge de 100%
- **batteryLevel** est une valeur comprise entre **0** et **batteryMaxLevel** qui indique le niveau de charge actuel
- Dans la déclaration de **batteryReceiver**, **Lateinit** permet d'indiquer que l'attribut sera initialisé plus tard

EXERCICE 1.3 - BATTERYRECEIVER

Création du BatteryReceiver


- Ajoutez à **MainActivity** la méthode **initBatteryReceiver** ci-contre :
- Placez votre curseur à l'endroit désigné par la flèche verte et appuyez sur Ctrl+Espace. Redéfinissez la méthode **onReceive**.

onReceive est appelée chaque fois que l'activité système transmet des information sur le niveau de charge de la batterie.

Elle prend notamment un Intent en paramètre qui contient deux informations (extra) :

- **BatteryManager.EXTRA_LEVEL** qui indique le niveau de charge de la batterie
- **BatteryManager.EXTRA_SCALE** qui indique le niveau de charge maximal pour la batterie

- Modifiez le code de onReceive pour attribuer une valeur aux attributs batteryLevel et batteryMaxLevel de MainActivity

```
private fun initBatteryReceiver()
{
    batteryReceiver = object: BroadcastReceiver()
    {
        
    }
}
```

EXERCICE 1.4 - AFFICHAGE & ABONNEMENT

Affichage du niveau de charge

- Ajoutez à **MainActivity** la méthode **displayBatteryLevel** qui modifiera les attribut **progress** et **max** de la ProgressBar en fonction du niveau de charge de la batterie.
- Appelez la méthode **displayBatteryLevel** à l'endroit du code qui vous semble le plus judicieux.

Abonnement à l'activité système

- Ajoutez à **MainActivity** la méthode **registerBatteryReceiver** qui abonnera notre **batteryReceiver** aux Intent de type **ACTION_BATTERY_CHANGED** :

```
private fun registerBatteryReceiver()  
{  
    registerReceiver(batteryReceiver, IntentFilter(Intent.ACTION_BATTERY_CHANGED))  
}
```

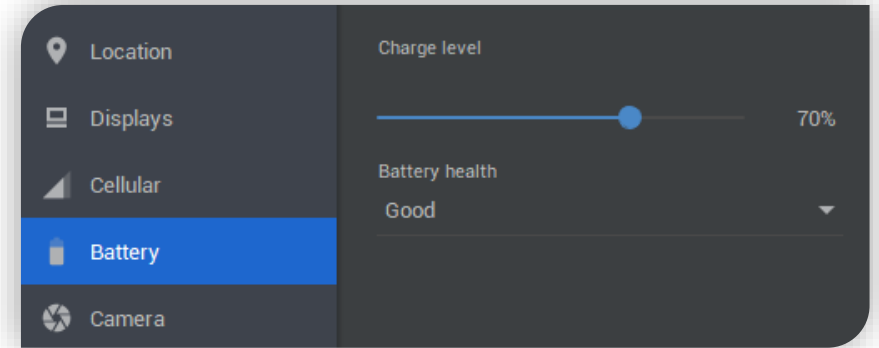
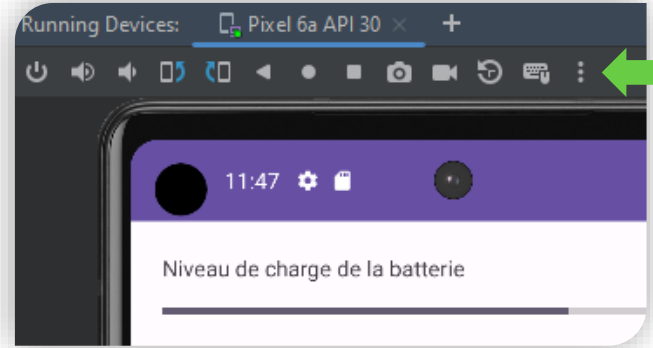
- Pensez à appeler **initBatteryReceiver** et **registerBatteryReceiver** dans la méthode **onCreate**.

EXERCICE 1.5 - TEST & AMÉLIORATIONS

Test

- Testez l'application.

Note : l'émulateur offre une fonctionnalité permettant de faire varier artificiellement le niveau de charge de la batterie :



Améliorations

Lorsque l'activité de votre application est en pause (pas en première position sur la pile d'activité), elle continue à recevoir les informations de charge de la batterie et à réagir à ces données.

- Ajoutez une méthode `unregisterBatteryReceiver` à `MainActivity`. Cette méthode appellera la fonction `unregisterReceiver`.
- Redéfinissez la méthode `onPause` de `MainActivity` et faites en sorte qu'elle appelle `unregisterBatteryReceiver`.
- Redéfinissez la méthode `onResume` de `MainActivity` et déplacez y l'appelle de la méthode `registerBatteryReceiver`.
- Testez de nouveau.

EXERCICE 2.1 - GPS

L'accès aux données de géolocalisation nécessite une demande de permission qui se déroule en deux temps :

- Tout d'abord, à l'installation, l'utilisateur est informé que l'application peut avoir besoin de sa localisation. S'il n'est pas d'accord avec cela, l'application ne s'installera pas. Cette première phase est gérée par le manifest que l'on verra un peu plus loin.
- Ensuite, lors de l'exécution de l'application, vous devrez demander au moins une fois à l'utilisateur la permission d'accéder à ses données de géolocalisation. Là encore, il peut refuser et votre application devra traiter ce cas.

Interface

- Complétez l'interface à l'aide des composants suivants :
 - TextView pour les libellés et les valeurs affichées
 - LinearLayout vertical et horizontal pour positionner le tout

| GPS | | |
|-----|--------------------|-------------------|
| | Latitude | Longitude |
| | 47.465223333333334 | 5.275158333333334 |

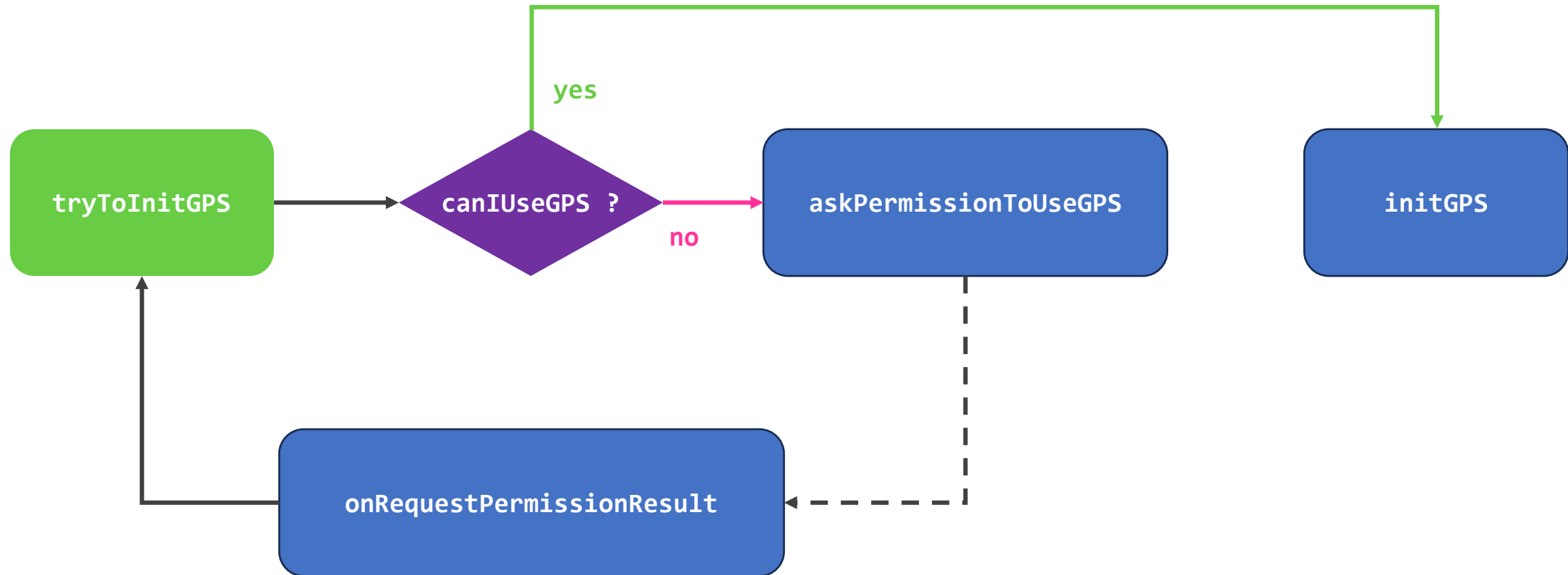
Manifest

- Ouvrez le fichier **manifests/AndroidManifest.xml**
- Ajoutez les demandes d'autorisation suivantes avant l'ouverture de la balise application

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

<application
```


EXERCICE 2.2 - ÉTAPE PAR ÉTAPE



EXERCICE 2.3 - AVONS NOUS LA PERMISSION ?

Tenter d'initialiser le GPS

- Ajoutez à la classe MainActivity une méthode `tryToInitGPS` qui appellera la méthode `canIUseGPS`. Si cette dernière renvoie `true`, `tryToInitGPS` appellera `initGPS`, sinon elle appellera `askPermissionToUseGPS`
- Appelez la méthode `tryToInitGPS` dans la méthode `onCreate` de `MainCativity`.

Pouvons nous utiliser le GPS ?

- Ajoutez à la classe `MainActivity` une méthode `canIUseGPS` qui demandera au système si l'application possède l'autorisation d'accéder à la géolocalisation du périphérique :

```
private fun canIUseGPS(): Boolean
{
    val permissionStatus = ContextCompat.checkSelfPermission(
        this,
        android.Manifest.permission.ACCESS_FINE_LOCATION
    )

    return permissionStatus == PackageManager.PERMISSION_GRANTED
}
```

EXERCICE 2.4 - DEMANDER ET RECEVOIR LA PERMISSION

Demander la permission

- Ajoutez à la classe **MainActivity** une méthode **askPermissionToUseGPS** qui demandera à l'utilisateur sa permission pour que l'application puisse accéder à la géolocalisation de son périphérique :
- Remplacez **gpsAccessRequestCode** par une valeur numérique de votre choix. Elle servira à identifier le résultat de la demande de permission

```
private fun askPermissionToUseGPS()
{
    ActivityCompat.requestPermissions(
        this,
        arrayOf(android.Manifest.permission.ACCESS_FINE_LOCATION),
        gpsAccessRequestCode
    )
}
```

Recevoir la permission

- En appuyant sur Ctrl+Espace redéfinissez la méthode **onRequestPermissionsResult** de **MainActivity**.
- Appelez **tryToUseGPS** si le paramètre **requestCode** est égal à la valeur (**gpsAccessRequestCode**) passée en paramètre à **requestPermission**

EXERCICE 2.5 - INITIALISATION DU GPS

Création du listener

- Ajoutez à la classe **MainActivity** une méthode **initGPS**.
- Dans cette méthode, instanciez un **LocationListener** qui sera chargé de réagir lorsque les coordonnées GPS du périphérique changeront :

```
val locationListener = object: LocationListener {  
    override fun onLocationChanged(location: Location) {  
  
    }  
}
```

Abonnement du listener aux événements du GPS

- Toujours dans la méthode **initGPS**, récupérez le service en charge du GPS et abonnez le **LocationListener** précédent aux événements de mise à jour de la position du périphérique :
- Enfin, modifiez le code de **onLocationChanged** pour afficher sur l'interface la latitude et la longitude fournies par le paramètre **location**

```
val locationManager = getSystemService(Context.LOCATION_SERVICE) as LocationManager  
  
locationManager.requestLocationUpdates(  
    LocationManager.GPS_PROVIDER,  
    1000,  
    10f,  
    locationListener  
)
```

*Note : l'appel de la méthode **requestLocationUpdates** nécessite d'avoir préalablement vérifié que l'application possède les permissions requises. Comme les test est effectué en dehors de **initGPS**, il sera nécessaire de placer un décorateur sur la fonction pour éviter d'avoir une erreur :*

```
@SuppressWarnings("MissingPermission")  
private fun initGPS()
```

EXERCICE 2.6 - TEST

Test avec l'émulateur

- Testez le bon fonctionnement de l'application.

Note : l'émulateur permet de tester artificiellement la géolocalisation :

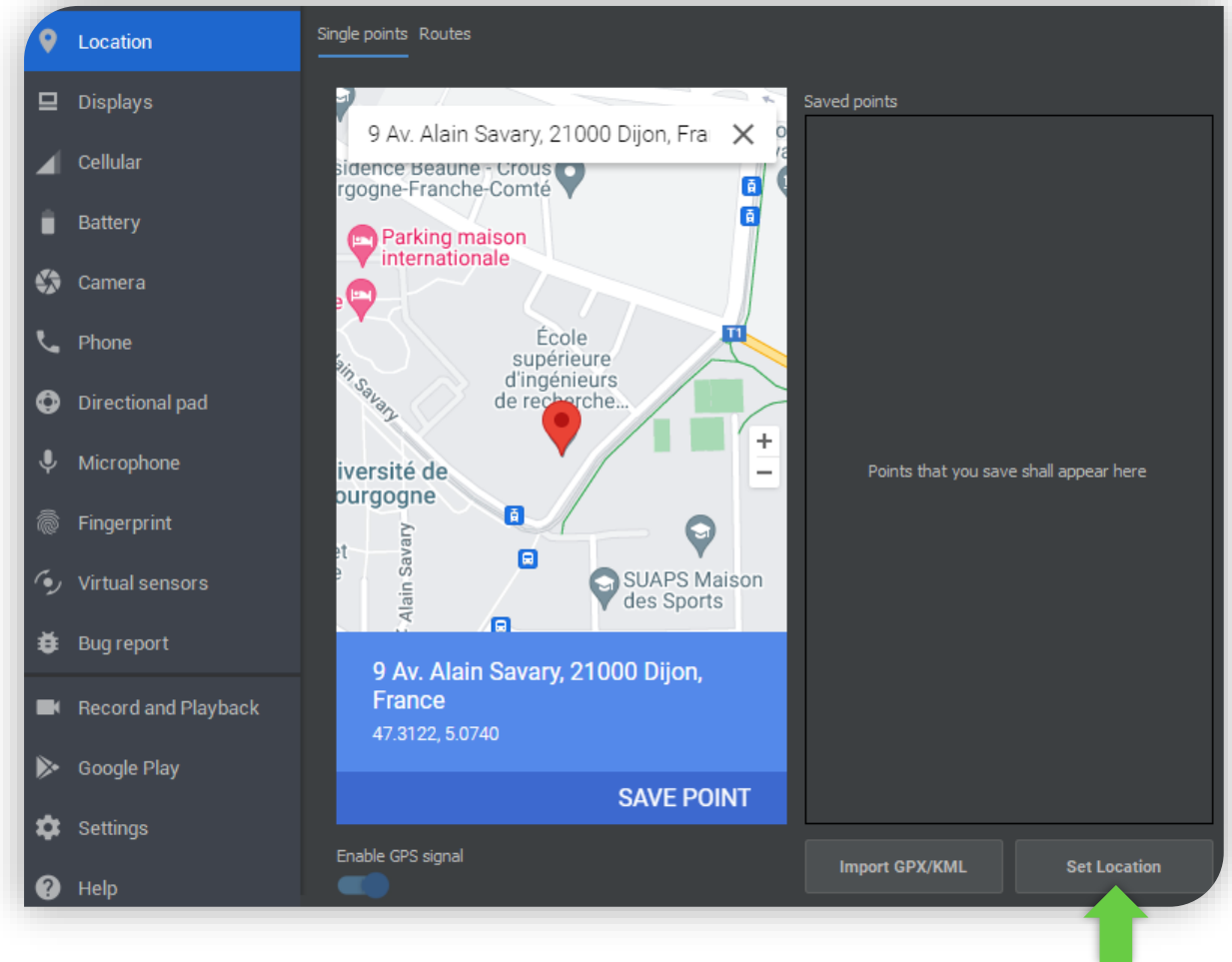
*Sélectionnez une adresse et pensez à cliquer sur le bouton **Set Location** pour définir les nouvelles coordonnées GPS.*

Test avec votre téléphone

- Testez le bon fonctionnement de l'application.

Note : le signal GPS ne passe pas à travers les murs des bâtiments. Placez vous vers une fenêtre pour que votre signal GPS soit actualisé.

Pensez également à activer la géolocalisation sur votre téléphone.



EXERCICE 3.1 - ACCÉLÉROMÈTRE

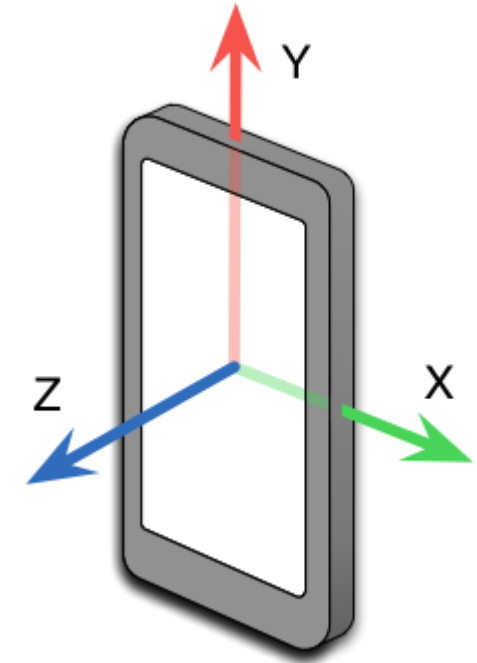
Les périphériques mobiles embarquent un accéléromètre qui mesure l'accélération linéaire le long des trois axes de l'appareil, comme représenté dans l'illustration ci-contre :

Principe

Chaque activité à la possibilité d'implémenter l'interface `SensorEventListener` pour ensuite s'abonner aux informations transmises par les capteurs du périphériques (accéléromètre, gyroscope et magnétomètre).

Interface

- Complétez l'interface à l'aide des composants suivants :
 - TextView pour les libellés et les valeurs affichées
 - LinearLayout vertical et horizontal pour positionner le tout



| Accéléromètre | | |
|---------------|------|----------|
| X | Y | Z |
| 0.0 | -0.0 | 9.809989 |

EXERCICE 3.2 - SENSOR MANAGER

SensorManager

- Ajoutez un attribut `sensorManager` de type `SensorManager` à la classe `MainActivity`.
- Ajoutez à `MainActivity` la méthode `initSensorManager` :

```
private fun initSensorManager()  
{  
    sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager  
}
```

- Appelez la méthode `initSensorManager` dans `onCreate`.

EXERCICE 3.3 - ABONNEMENT AU CAPTEUR

Abonnement

- Ajoutez à **MainActivity** la méthode **registerAccelerometerSensor** :

La méthode tente d'obtenir un accès à l'accéléromètre si ce dernier existe sur le périphérique.

Si c'est le cas, le **SensorManager** de notre activité va s'abonner au flux de données du capteur avec une fréquence d'échantillonnage définie par la constante **SENSOR_DELAY_UI**.

- Ajoutez la méthode **unregisterSensors** à **MainActivity**.
- Appelez **registerAccelerometerSensor** et **unregisterSensors** au bon endroit.

```
private fun registerAccelerometerSensor()
{
    val accelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)

    if(accelerometerSensor != null)
    {
        sensorManager.registerListener(
            this,
            accelerometerSensor,
            SensorManager.SENSOR_DELAY_UI
        )
    }
}
```

```
private fun unregisterSensors()
{
    sensorManager.unregisterListener(this)
}
```


EXERCICE 3.4 - RÉCUPÉRATION DES INFORMATIONS

Nouvel attribut

- Ajoutez à **MainActivity** un attribut **accelerometerValues** de type **FloatArray** initialisés à **FloatArray(3)**.

Interface **SensorEventListener**

- Modifiez l'activité **MainActivity** pour que celle-ci implémente l'interface **SensorEventListener** :

```
class MainActivity : AppCompatActivity(), SensorEventListener {
```

- Comme vu précédemment, implémentez les méthodes **onSensorChanged** et **onAccuracyChanged** requises par l'interface **SensorEventListener**

onSensorChanged reçoit un **SourceEvent** en paramètre qui possède deux attributs intéressants :

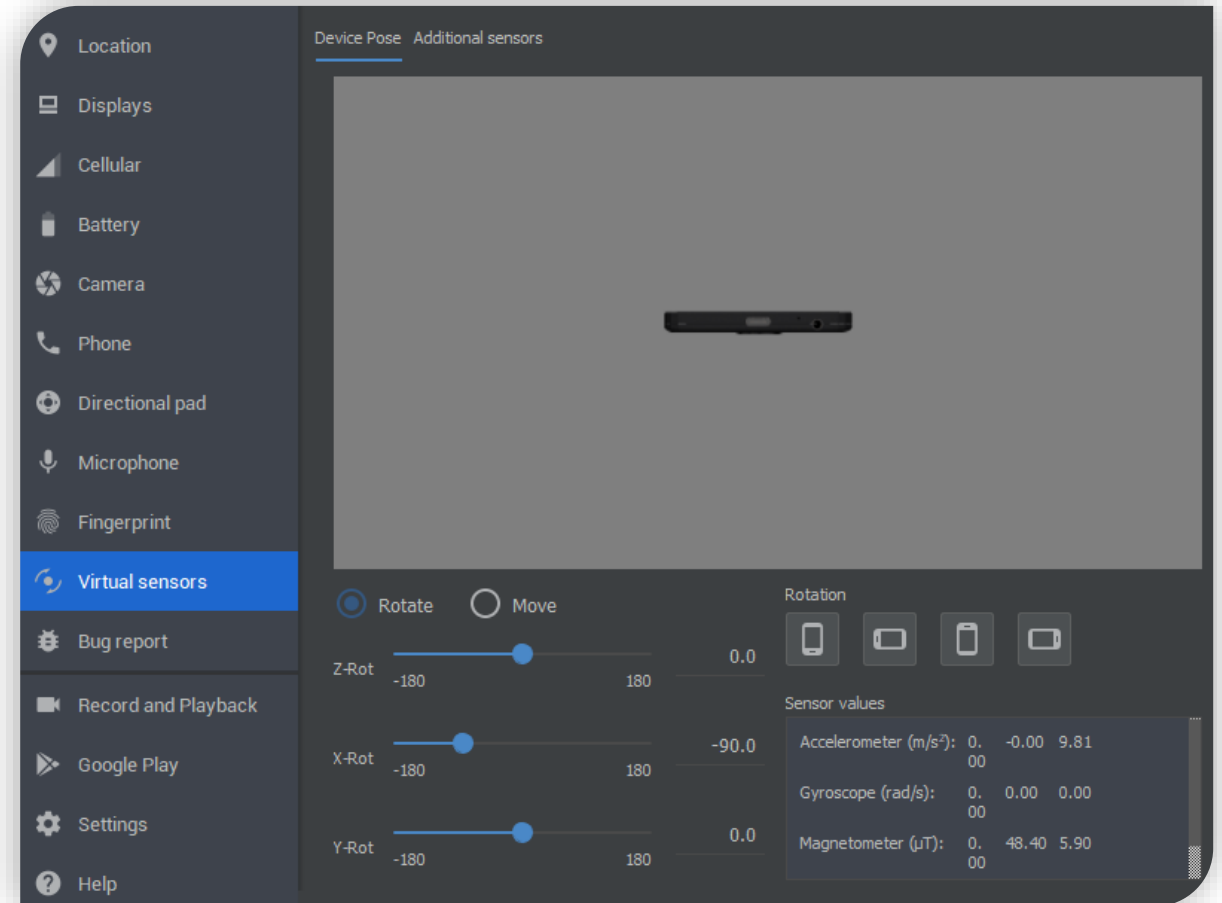
- **sensor** qui précise quel capteur transmet l'information
- **values** qui est un **FloatArray** contenant 3 valeurs correspondant aux accélérations mesurées sur les axes x, y et z

- Modifiez **onSensorChanged** pour stocker les valeurs reçues dans **accelerometervalues** si le capteur qui envoie les données est de type **Sensor.TYPE_ACCELEROMETER**
- La méthode **onAccuracyChanged** ne sera pas utile ici. Elle doit être implémentée mais laissez cette fonction vide.

EXERCICE 3.4 - AFFICHAGE DES INFORMATIONS

Affichage des données de l'accéléromètre

- Ajoutez à `MainActivity` une méthode `displayAccelerometerData` qui affiche les valeurs contenues dans `accelerometerValues` dans les TextView correspondants.
- Testez le bon fonctionnement. Si vous placez votre téléphone à plat à l'horizontal, vous devriez avoir des valeurs proches des suivantes :
 - X : 0
 - Y : 0
 - Z : 9,81
- Expliquez ces valeurs.



EXERCICE 3.5 - BONUS

Inclinaison gauche / droite

- Ajoutez une SeekBar à l'interface, comme présenté ci-contre :
- Faites progresser la SeekBar en fonction de l'inclinaison de l'écran : vers la gauche si l'écran est incliné à gauche et vers la droite si l'écran est incliné à droite.

