Localisation et Carte sous android

Plan

I. Introduction

Avec les progrès de la miniaturisation, la plupart — voire la quasi-totalité — des terminaux sont équipés de puces GPS. La géo localisation est ainsi devenue un élément du quotidien qu'on retrouve dans énormément d'applications. On peut penser aux applications de navigation aidée par GPS, mais aussi aux applications sportives qui suivent nos efforts et élaborent des statistiques, ou encore aux applications pour noter les restaurants et les situer. On trouve ainsi deux API qui sont liées au concept de localisation [1] :

- Une API qui permet de localiser l'appareil.
- Une API qui permet d'afficher des cartes.

II. La localisation

La position d'une personne est estimée à l'aide de ces informations sur son téléphone : [2]

- **GPS** : grâce aux satellites, sa position est détectée avec une marge d'erreur de quelques mètres.
- **Wi-Fi** : l'emplacement des réseaux Wi-Fi alentours aide Google Maps à lui situer.
- **Relais de téléphonie mobile** : la précision d'une connexion à un réseau mobile est de quelques kilomètres.
- **Autres informations** : sa position peut être améliorée par l'accéléromètre, la boussole, le gyroscope et le baromètre de son téléphone.

- **Les services de localisation Apple** sont utilisables sur iPhone et iPad si le GPS n'est pas disponible.

II.1. Permissions

Le GPS est la solution la plus efficace pour localiser un appareil, cependant il s'agit aussi de la plus coûteuse en batterie. Une autre solution courante est de se localiser à l'aide des points d'accès WiFi à proximité et de la distance mesurée avec les antennes relais du réseau mobile les plus proches (par triangulation) [1].

Tout d'abord, on devra demander la permission dans le Manifest pour utiliser les fonctionnalités de localisation. Si on veut utiliser la géolocalisation (par GPS, donc), utiliser ACCESS_FINE_LOCATION ; pour une localisation plus imprécise par WiFi et antennes relais, utiliser ACCESS_COARSE_LOCATION. Enfin, si on veut utiliser les deux types de localisation, on peut déclarer uniquement ACCESS_FINE_LOCATION, qui comprend toujours ACCESS_COARSE_LOCATION [1]:

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

Pour les versions d'android inférieures à l'API23 (Android 6.0 marshmallow), l'ajout de l'une de ces deux permissions est suffisant pour avoir la permission de localiser l'appareil. Cependant à partir de l'API23, en plus de l'ajout de l'une de ces deux permissions il faut demander une autorisation explicite de l'utilisateur pour localiser l'appareil. Ceci permet d'augmenter la sécurité : l'utilisateur connaît que l'application est en train de le localiser à chaque fois qu'elle utilise la localisation. La demande de cette autorisation explicite se fait en utilisant le code java.

Le code suivant permet de demander une autorisation explicite d'utiliser la localisation [8] :

```
private void askPermissionsAndShowMyLocation() {
  // With API> = 23, you have to ask the user for permission to view their location.
  if (Build.VERSION.SDK_INT >= 23) {
      int accessCoarsePermission
        = ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION);
      int accessFinePermission
        = ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION);
        if (accessCoarsePermission != PackageManager.PERMISSION_GRANTED
        || accessFinePermission != PackageManager.PERMISSION_GRANTED) {
    // The Permissions to ask user.
    String[] permissions = new String[]{Manifest.permission.ACCESS_COARSE_LOCATION,
      Manifest.permission.ACCESS_FINE_LOCATION};
    // Show a dialog asking the user to allow the above permissions.
    ActivityCompat.requestPermissions(this, permissions,
      REQUEST_ID_ACCESS_COURSE_FINE_LOCATION);
    return;
      }
  }
}
// When you have the request results.
@Override
public void onRequestPermissionsResult(int requestCode,
             String permissions[], int[] grantResults) {
  super.onRequestPermissionsResult(requestCode, permissions, grantResults);
  switch (requestCode) {
      case REQUEST_ID_ACCESS_COURSE_FINE_LOCATION:
    // Note: If request is cancelled, the result arrays are empty.
    // Permissions granted (read/write).
    if (grantResults.length > 1
      && grantResults[0] == PackageManager.PERMISSION_GRANTED
      && grantResults[1] == PackageManager.PERMISSION_GRANTED) {
        Toast.makeText(this, "Permission granted!", Toast.LENGTH_LONG).show();
        // Show current location on Map.
        this.showMyLocation();
    }
    // Cancelled or denied.
    else {
        Toast.makeText(this, "Permission denied!", Toast.LENGTH_LONG).show();
    }
    break;
  }
}
```

Le code suivant permet d'afficher le paramétrage du GPS (activation/désactivation) :

```
private void demanderActivationGPS() {
        Intent i = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
        startActivity(i);
}
```

II.2. Gestionnaire de localisation

On trouve tous les outils de localisation dans le package android.location.

LocationManager est une classe qui représente un gestionnaire de localisation, un gestionnaire de localisation est initialisé comme suit :

```
LocationManager locationManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
```

II.3. Fournisseurs de position

On aura ensuite besoin d'un fournisseur de position qui sera dans la capacité de déterminer la position actuelle. On a par un exemple un fournisseur pour le GPS et un autre pour les antennes relais. Ces fournisseurs dériveront de la classe abstraite LocationProvider. Il existe plusieurs méthodes pour récupérer les fournisseurs de position disponibles sur l'appareil. Pour récupérer le nom de tous les fournisseurs, il suffit de faire List<String> getAllProviders(). Le problème de cette méthode est qu'elle va récupérer tous les fournisseurs qui existent, même si l'application n'a pas le droit de les utiliser ou qu'ils sont désactivés par l'utilisateur [1].

Pour ne récupérer que le nom des fournisseurs qui sont réellement utilisables, on utilisera List<String> getProviders(boolean enabledOnly). Enfin, on peut obtenir un LocationProvider à partir de son nom avec LocationProvider getProvider(String name) :

```
ArrayList<LocationProvider> providers = new ArrayList<LocationProvider>();
ArrayList<String> names = locationManager.getProviders(true);
for(String name : names)
  providers.add(locationManager.getProvider(name));
```

Les noms des fournisseurs sont contenus dans des constantes, commeLocationManager.GPS_PROVIDER pour le GPS et LocationManager.NETWORK_PROVIDER pour la triangulation.

List<String> getAllProviders() permet de récupérer tous les noms des fournisseurs actifs et inactifs.

II.4. Obtenir des notifications du fournisseur

Pour obtenir la dernière position connue de l'appareil, utiliser Location getLastKnownLocation(String provider).

La dernière position connue n'est pas forcément la position actuelle de l'appareil. En effet, il faut demander à mettre à jour la position pour que celle-ci soit renouvelée dans le fournisseur. Si on veut faire en sorte que le fournisseur se mette à jour automatiquement à une certaine période ou tous les x mètres, on peut utiliser la méthode void requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener) avec [1] :

- **provider** le fournisseur de position.
- **minTime** la période entre deux mises à jour en millisecondes. Il faut mettre une valeur supérieure à 0, sinon le fournisseur ne sera pas mis à jour périodiquement. D'ailleurs, ne metter pas de valeur en dessous de 60 000 ms pour préserver la batterie.
- **minDistance** la période entre deux mises à jour en mètres. Tout comme pour minTime, il faut mettre une valeur supérieure à 0, sinon ce critère ne sera pas pris en compte. De plus, on privilégie quand même minTime parce qu'il consomme moins de batterie.
- **listener** est l'écouteur qui sera lancé dès que le fournisseur sera activé.

Ainsi, il faut qu'on utilisera l'interface LocationListener, dont la méthode void onLocationChanged(Location location) sera déclenchée à chaque mise à jour. Cette méthode de callback contient un objet de type Location duquel on peut extraire des informations sur l'emplacement donné. Par exemple, on peut récupérer la latitude avec double getLatitude() et la longitude avec double getLongitude() [1]:

```
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 60000, 1500, new
LocationListener() {
        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) {

        }
        @Override
        public void onProviderEnabled(String provider) {
        }
        @Override
        public void onProviderDisabled(String provider) {
        }
        @Override
        public void onLocationChanged(Location location) {
          Log.d("GPS", "Latitude " + location.getLatitude() + " et longitude " +
       location.getLongitude());
        }
});
```

II.5. Les alertes de proximité

Cette partie traite les alertes de proximité qui permet de nos informés quand on s'approche d'un endroit ou qu'on s'en éloigne. Cet endroit peut être symbolisé par un cercle dont on va préciser le centre et le rayon. Ainsi, si on entre dans ce cercle ou qu'on sort de ce cercle, l'alerte est lancée.

Le prototype de la méthode qui peut créer une alerte de proximité est void addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent intent) avec :

- La latitude et la longitude du centre du cercle.
- Le rayon d'effet est précisé dans radius en mètres.
- expiration permet de déclarer combien de temps cette alerte est valable. Tout nombre en dessous de 0 signifie qu'il n'y a pas d'expiration possible.
- Enfin, on donne aussi le PendingIntent qui sera lancé quand cette alerte est déclenchée, avec intent.

Cette fois, l'intent contiendra un booléen en extra, dont la clé est `KEY_PROXIMITY_ENTERING` et la valeur sera `true` si on entre dans la zone et `false` si on en sort.

```
     Intent intent = new Intent(this, AlertReceiver.class);
     PendingIntent pending = PendingIntent.getBroadcast(this, 0, intent,
     PendingIntent.FLAG_UPDATE_CURRENT);
// On ajoute une alerte de proximité si on s'approche ou on ss'éloigne du bâtiment de Simple IT
     locationManager.addProximityAlert(48.872808, 2.33517, 150, -1, pending);
```

On le recevra ensuite dans :

```
     public class AlertReceiver extends BroadcastReceiver {
       @Override
       public void onReceive(Context context, Intent intent) {
            // Vaudra true par défaut si on ne trouve pas l'extra booléen dont la clé est
            LocationManager.KEY_PROXIMITY_ENTERING
           bool entrer =
     booleanValue(intent.getBooleanExtra(LocationManager.KEY_PROXIMITY_ENTERING,
              true));
       }
     }
```

Enfin, il faut désactiver une alerte de proximité avec `void removeProximityAlert(PendingIntent intent)`.

III. Carte

L'affichage de la longitude et de la latitude sous forme de double engendre une difficulté de savoir la signification géographique de ces valeurs. Pour cela google propose une API Google Maps qui permet d'afficher les cartes.

III.1. Google Maps API

Avant toute chose, nous avons besoin d'une clé de l'API afin de pouvoir l'utiliser dans l'application. Pour cela, il faut se rendre sur Google Developer Console, une fois connecté avec votre compte Google, il faut créer un nouveau projet qui concernera l'application. Une fois le projet créé, il faut activer l'API qui nous intéresse, ici celui de **Google Maps Android API**. Par la suite, il faut vérifier que L'API est bien activée, sinon il faut l'activer. Enfin, il nous faut les identifiants afin de pouvoir les utiliser dans notre application. Il est possible selon l'application que vous crée de définir des restrictions tels que sur des sites Web, ou encore des adresses IP, afin d'éviter des utilisations abusives en autres [8].

III.2. Installation de Google Play Services

Avec le SDK Manager il faut installer le package Google Play Services. Dans le **build.gradle**, il faut aussi ajouter la dépendance de la dernière version de play-service de Google Maps [8].

```
implementation 'com.google.android.gms:play-services-maps:10.0.0'
```

III.3. Configurer Google Maps

Dans le fichier **AndroidManifest.xml**, on ajoute la clé obtenu depuis Google Developer Console dans la balise *<application>*.

```
<meta-data
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="#myKey" />
```

Note: Remplacer #myKey par votre clé.

Pour pouvoir afficher notre carte, il ne faut pas oublier d'ajouter les permissions et les features nécessaires dans la balise <manifest> avant la balise <application>.

```
<permission
        android:name="#myPackage.permission.MAPS_RECEIVE"
        android:protectionLevel="signature" />
<uses-permission android:name="#myPackage.permission.MAPS_RECEIVE" />
```

Note: Remplacer **#myPackage** par le nom de votre package.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"
/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<!-- current location -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-feature
               android:glEsVersion="0x00020000"
               android:required="true" />
```

ACCESS_NETWORK_STATE, donne l'état de la connexion.
INTERNET, autorise la connexion Internet.
READ_GSERVICES, lire les données de configuration des services Google.
WRITE_EXTERNAL_STORAGE, écrire des données sur la carte SD.
ACCESS_COARSE_LOCATION, obtenir votre position approximative. Cet emplacement est dérivé par des services de localisation utilisant des sources de localisation de réseau telles que des tours de cellules et Wi-Fi. Ces services de localisation doivent être activés et disponibles sur votre appareil pour que l'application puisse les utiliser.
ACCESS_FINE_LOCATION, obtenir un emplacement précis à l'aide du système de positionnement global (GPS) ou des sources de localisation de réseau telles que les tours de cellules et le Wi-Fi. Ces services de localisation doivent être activés et disponibles sur votre appareil pour que l'application puisse les utiliser.
glEsVersion, permet d'utiliser Google Map v2 sur l'appareil pour cette version d'OpenGL.

III.4. ActivityMain.java

```
package com.mapping;
//imports
public class MainActivity extends Activity {
  // Google Map
  private GoogleMap googleMap;
  public static final  float ISET_LONG=10.772176f;
  public static final  float ISET_LAT=34.756932f;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    try {
      initMap();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  private void initMap() {
    if (googleMap == null) {
         googleMap = ((MapFragment) getFragmentManager().findFragmentById(R.id.map)).getMap();
         if (googleMap == null) {
         Toast.makeText(getApplicationContext(),"Impossible de créer des maps", Toast.LENGTH_SHORT)
         .show();
         }
    }
         marquerIset();
         zoomIset();
  }
  private void zoomIset() {
         CameraPosition cameraPosition = new CameraPosition.Builder()
                  .target(new LatLng(ISET_LAT, ISET_LONG)).zoom(12).build();
         googleMap.animateCamera(CameraUpdateFactory
                  .newCameraPosition(cameraPosition));
  }
  private void marquerIset() {
         MarkerOptions marker = new MarkerOptions().position(new LatLng(ISET_LAT, ISET_LONG));
                  marker.title("Iset Sfax");
                  marker.snippet("DSI3");
                  googleMap.addMarker(marker);

  }
  @Override
  protected void onResume() {
         super.onResume();
         initilizeMap();
  }
}
```

III.5. Placer marqueur

```
double latitude = 34.756932f;
double longitude = 10.772176f;
MarkerOptions marker = new MarkerOptions().position(new LatLng(latitude, longitude)).title("Hello
Maps ");
googleMap.addMarker(marker);
```

III.6. Changer la couleur d'un marqueur

Par défaut les marqueurs sont rouges, pour changer la couleur d'un marquer, utiliser :

```
marker.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN));
```

III.7. Icône personnalisée pour un marqueur

Pour modifier l'icône d'un marqueur, utiliser :

```
double latitude = 17.385044;
double longitude = 78.486671;
```

```
MarkerOptions marker = new MarkerOptions().position(new LatLng(latitude, longitude))
                       .title("Hello Maps");
marker.icon(BitmapDescriptorFactory.fromResource(R.drawable.my_marker_icon)));
googleMap.addMarker(marker);
```

III.8. Déplacer la caméra à une location avec animation

```
CameraPosition cameraPosition = new CameraPosition.Builder().target(
                new LatLng(17.385044, 78.486671)).zoom(12).build();

googleMap.animateCamera(CameraUpdateFactory.newCameraPosition(cameraPosition));
```

III.9. Changer le type de la carte

Google propose quatre types de cartes : normal, hybride, satellite et terrain.

```
googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
googleMap.setMapType(GoogleMap.MAP_TYPE_NONE);
```

III.10. Visualiser la localisation courante

```
googleMap.setMyLocationEnabled(true); // false pour masquer
```

III.11. Boutons Zoom

```
googleMap.getUiSettings().setZoomControlsEnabled(true); // false pour masquer
```

III.12. Zoom sur les gestes

```
googleMap.getUiSettings().setZoomGesturesEnabled(true);  // false pour désactiver
```

III.13. Fonctionnalité Boussoule

```
googleMap.getUiSettings().setCompassEnabled(true); // false pour désactiver
```

III.14. Button ma localisation

```
googleMap.getUiSettings().setMyLocationButtonEnabled(true);    // false pour masquer
```

III.15. Rotation sur les gestes

```
googleMap.getUiSettings().setRotateGesturesEnabled(true);    // false pour désactiver
```

III.16. Evènement LongClick sur la carte

```
googleMap.setOnMapLongClickListener(new OnMapLongClickListener() {
        @Override
        public void onMapLongClick(LatLng arg0) {
                parametrer(arg0);
        }
});
```

Références

[1] https://openclassrooms.com/courses/creez-des-applications-pour-android/la-localisation-et-les-cartes

[2] https://support.google.com/gmm/answer/2839911?rd=1

[3] http://www.androidhive.info/2013/08/android-working-with-google-maps-v2/

[4] http://o7planning.org/en/10501/google-maps-android-api-tutorial

[5] https://developer.android.com/training/permissions/requesting#java

[6] https://www.frandroid.com/android/developpement/214541_devez-absolument-verifier-vos-permissions-dinstaller-application

[7] https://o7planning.org/en/10501/google-maps-android-api-tutorial

[8] https://www.supinfo.com/articles/single/3785-android-api-google-maps-v2