33

Accès aux services de localisation

Le GPS est une fonctionnalité très appréciée des terminaux mobiles actuels car il permet de vous indiquer votre emplacement géographique à tout moment. Bien que l'utilisation la plus fréquente du GPS soit la cartographie et l'orientation, connaître votre emplacement vous ouvre de nombreux autres horizons. Vous pouvez, par exemple, mettre en place une application de chat dynamique où vos contacts sont classés selon leurs emplacements géographiques, afin de choisir ceux qui sont les plus près de vous. Vous pouvez également "géotaguer" automatiquement les articles que vous postez sur Twitter ou d'autres services similaires.

Cependant, le GPS n'est pas le seul moyen d'identifier un emplacement géographique :

- L'équivalent européen de GPS, Galileo, est encore en cours de mise au point.
- La triangulation permet de déterminer votre position en fonction de la force du signal des antennes relais proches de vous.
- La proximité des "hotspots" Wifi, dont les positions géographiques sont connues.

Les terminaux Android peuvent utiliser un ou plusieurs de ces services. En tant que développeur, vous pouvez demander au terminal de vous indiquer votre emplacement, ainsi que des détails sur les fournisseurs disponibles. Vous pouvez même simuler votre localisation avec l'émulateur pour tester les applications qui ont besoin de cette fonctionnalité.

Fournisseurs de localisation : ils savent où vous vous cachez

Les terminaux Android peuvent utiliser plusieurs moyens pour déterminer votre emplacement géographique. Certains ont une meilleure précision que d'autres ; certains sont gratuits, tandis que d'autres sont payants ; certains peuvent vous donner des informations supplémentaires, comme votre altitude par rapport au niveau de la mer ou votre vitesse courante.

Android a donc abstrait tout cela en un ensemble d'objets `LocationProvider`. Votre environnement utilisera zéro ou plusieurs instances de `LocationProvider`, une par service de localisation disponible sur le terminal. Ces fournisseurs ne connaissent pas seulement votre emplacement mais possèdent également leurs propres caractéristiques – précision, prix, etc.

Vous aurez donc besoin d'un `LocationManager` contenant l'ensemble des `LocationProvider` pour savoir quel est le `LocationProvider` qui convient à votre cas particulier. Votre application devra également disposer de la permission `ACCESS_LOCATION` ; sinon les différentes API de localisation échoueront à cause d'une violation de sécurité. Selon les fournisseurs de localisation que vous voulez utiliser, vous pourrez également avoir besoin d'autres permissions, comme `ACCESS_COARSE_LOCATION` ou `ACCESS_FINE_LOCATION`.

Se trouver soi-même

L'opération la plus évidente d'un fournisseur de localisation consiste à trouver votre emplacement actuel. Pour ce faire, vous avez besoin d'un `LocationManager`, que vous obtiendrez par un appel à `getSystemService(LOCATION_SERVICE)` à partir de votre activité ou service, en transtypant le résultat pour obtenir un `LocationManager`.

L'étape suivante consiste à obtenir le nom du `LocationProvider` que vous voulez utiliser. Pour ce faire, deux possibilités s'offrent à vous :

- demander à l'utilisateur de choisir un fournisseur ;
- trouver le fournisseur qui convient le mieux en fonction d'un ensemble de critères.

Si vous choisissez la première approche, un appel à la méthode `getProviders()` du `LocationManager` vous donnera une liste de fournisseurs que vous pouvez présenter à l'utilisateur pour qu'il fasse son choix.

Vous pouvez également créer et initialiser un objet `Criteria`, en précisant ce que vous attendez d'un `LocationProvider`. Par exemple :

- `setAltitudeRequired()` pour indiquer si vous avez besoin ou non de connaître votre altitude ;
- `setAccuracy()` pour fixer un niveau de précision minimal de la position, en mètres ;
- `setCostAllowed()` pour indiquer si le fournisseur doit être gratuit ou non (c'est-à-dire s'il peut impliquer un paiement de la part de l'utilisateur du terminal).

Lorsque l'objet `Criteria` a été rempli, appelez la méthode `getBestProvider()` de votre `LocationManager` et Android passera les critères en revue pour vous donner la meilleure réponse. Tous ces critères peuvent ne pas être vérifiés – à part celui concernant le prix, ils peuvent tous être ignorés si rien ne correspond.

Pour effectuer des tests, vous pouvez également indiquer directement dans votre code le nom d'un `LocationProvider` (`gps`, par exemple).

Lorsque vous connaissez le nom du `LocationProvider`, vous pouvez appeler `getLastKnownPosition()` pour trouver votre dernière position. Notez, cependant, que cette "dernière position" peut être obsolète (si, par exemple, le téléphone était éteint) ou valoir `null` si aucune position n'a encore été enregistrée pour ce fournisseur. En revanche, `getLastKnownPosition()` est gratuite et ne consomme pas de ressource car le fournisseur n'a pas besoin d'être activé pour connaître cette valeur.

Ces méthodes renvoient un objet `Location` qui vous indiquera la latitude et la longitude du terminal en degrés – des valeurs `double` de Java. Si le fournisseur donne d'autres informations, vous pouvez les récupérer à l'aide des méthodes suivantes :

- `hasAltitude()` indique s'il y a une valeur pour l'altitude et `getAltitude()` renvoie l'altitude en mètres.
- `hasBearing()` indique s'il y a une information d'orientation (une valeur de compas) et `getBearing()` renvoie cette valeur en degrés par rapport au vrai nord.
- `hasSpeed()` indique si la vitesse est connue et `getSpeed()` la renvoie en mètres par seconde.

Ceci dit, une approche plus fréquente pour obtenir l'objet `Location` à partir d'un `LocationProvider` consiste à s'enregistrer pour les modifications de la position du terminal, comme expliqué dans la section suivante.

Se déplacer

Tous les fournisseurs de localisation ne répondent pas immédiatement. GPS, par exemple, nécessite l'activation d'un signal et la réception des satellites (c'est ce que l'on appelle un "fix GPS") avant de pouvoir connaître sa position. C'est la raison pour laquelle Android ne fournit pas de méthode `getMeMyCurrentLocationNow()`. Ceci combiné avec le fait que les utilisateurs puissent vouloir que leurs mouvements soient pris en compte dans l'application, vous comprendrez pourquoi il est préférable d'enregistrer les modifications de la position et les utiliser pour connaître la position courante.

Les applications `Weather` et `WeatherPlus` montrent comment enregistrer ces mises à jour – en appelant la méthode `requestLocationUpdates()` de l'objet `LocationManager`. Cette méthode prend quatre paramètres :

1. Le nom du fournisseur de localisation que vous souhaitez utiliser.
2. Le temps, en millisecondes, qui doit s'écouler avant que l'on puisse obtenir une mise à jour de la position.
3. Le déplacement minimal du terminal en mètres pour que l'on puisse obtenir une mise à jour de la position.
4. Un `LocationListener` qui sera prévenu des événements liés à la localisation, comme le montre le code suivant :

```
LocationListener onLocationChange=new LocationListener() {
  public void onLocationChanged(Location location) {
    updateForecast(location);
  }
  public void onProviderDisabled(String provider) {
    // Exigée par l'interface, mais inutilisée
  }
  public void onProviderEnabled(String provider) {
    // Exigée par l'interface, mais inutilisée
  }
  public void onStatusChanged(String provider, int status,
                              Bundle extras) {
    // Exigée par l'interface, mais inutilisée
  }
};
```

Ici, nous appelons simplement `updateForecast()` en lui passant l'objet `Location` fourni à l'appel de la méthode de rappel `onLocationChanged()`. Comme on l'a vu au Chapitre 30, l'implémentation d'`updateForecast()` construit une page web contenant les prévisions météorologiques pour l'emplacement courant et envoie un message de diffusion afin que l'activité sache qu'une mise à jour est disponible.

Lorsque l'on n'a plus besoin des mises à jour, on appelle `removeUpdates()` avec le `LocationListener` que l'on avait enregistré.

Est-on déjà arrivé ? Est-on déjà arrivé ? Est-on déjà arrivé ?

Parfois, on veut savoir non pas où l'on se trouve ni même où l'on va, mais si l'on est là où l'on voulait aller. Il pourrait s'agir d'une destination finale ou d'une étape dans un ensemble de directions pour pouvoir indiquer le virage suivant, par exemple.

Dans ce but, `LocationManager` fournit la méthode `addProximityAlert()`, qui enregistre un `PendingIntent` qui sera déclenché lorsque le terminal se trouvera à une certaine distance d'un emplacement donné. La méthode `addProximityAlert()` attend les paramètres suivants :

- La latitude et la longitude de la position qui nous intéresse.
- Un rayon précisant la proximité avec la position pour que l'intention soit levée.
- Une durée d'enregistrement en millisecondes – passée cette période, l'enregistrement expirera automatiquement. Une valeur de –1 indique que l'enregistrement sera maintenu jusqu'à ce que vous le supprimiez manuellement *via* un appel à `removeProximityAlert()`.
- Le `PendingIntent` qu'il faudra lever lorsque le terminal se trouve dans la "zone de tir" définie par la position et le rayon.

Notez qu'il n'est pas garanti que vous receviez une intention s'il y a eu une interruption dans les services de localisation ou si le terminal n'est pas dans la zone cible pendant le temps où l'alerte de proximité est active. Si la position, par exemple, est trop proche du but et que le rayon est un peu trop réduit, le terminal peut ne faire que longer le bord de la zone cible ou y passer si rapidement que sa position ne sera pas enregistrée pendant qu'il est dans la zone.

Il vous appartient de faire en sorte qu'une activité ou un récepteur d'intention réponde à l'intention que vous avez enregistrée pour l'alerte de proximité. C'est également à vous de déterminer ce qui doit se passer lorsque cette intention arrive : configurer une notification (faire vibrer le terminal, par exemple), enregistrer l'information dans un fournisseur de contenu, poster un message sur un site web, etc. Notez que vous recevrez l'intention à chaque fois que la position est enregistrée et que vous êtes dans la zone ciblée – pas simplement lorsque vous y entrez. Par conséquent, vous la recevrez plusieurs fois – le nombre d'occurrences dépend de la taille de la zone et de la vitesse de déplacement du terminal.

Tester... Tester...

L'émulateur d'Android ne permet pas d'obtenir un "fix GPS", de trianguler votre position à partir des antennes relais ni de déduire votre position à partir des signaux Wifi voisins. Si vous voulez simuler un terminal qui se déplace, il faut donc trouver un moyen de fournir à l'émulateur des données de localisation simulées.

Pour une raison inconnue, ce domaine a subi des changements importants au cours de l'évolution d'Android. À une époque, il était possible de fournir des données de localisation simulées à une application, ce qui était très pratique pour les tests et les démonstrations mais, malheureusement, cette possibilité a disparu à partir d'Android 1.0.

Ceci dit, DDMS (*Dalvik Debug Monitor Service*) permet de fournir ce type de données. Il s'agit d'un programme externe, séparé de l'émulateur, qui peut fournir à ce dernier des points d'emplacements ou des routes complètes, dans différents formats. DDMS est décrit en détail au Chapitre 37.

34

Cartographie avec `MapView` et `MapActivity`

Google Maps est l'un des services les plus connus de Google – après le moteur de recherche, bien entendu. Avec lui, vous pouvez tout trouver, de la pizzeria la plus proche au trajet menant de Toulouse à Paris en passant par les vues détaillées des rues (*Street View*) et les images satellites.

Android intègre Google Maps : cette activité de cartographie est directement disponible à partir du menu principal mais, surtout, les développeurs ont à leur disposition les classes `MapView` et `MapActivity` pour intégrer des cartes géographiques dans leurs applications. Grâce à elles, ils peuvent non seulement contrôler le niveau du zoom, permettre aux utilisateurs de faire défiler la carte, mais également utiliser les services de localisation pour marquer l'emplacement du terminal et indiquer son déplacement.

Heureusement, cette intégration est assez simple et vous pouvez exploiter toute sa puissance si vous le souhaitez.

Termes d'utilisation

Google Maps, notamment lorsqu'il est intégré dans des applications tierces, nécessite le respect d'un assez grand nombre de termes juridiques. Parmi ceux-ci se trouvent des clauses que vous trouverez peut-être insupportables. Si vous décidez d'utiliser Google Maps, prenez soin de bien lire tous ces termes afin d'être sûr que l'utilisation que vous comptez en faire ne les viole pas. Nous vous conseillons fortement de demander l'avis d'un conseiller juridique en cas de doute.

En outre, ne délaissez pas les autres possibilités de cartographie qui reposent sur d'autres sources de données géographiques, comme OpenStreetMap[1].

Empilements

À partir d'Android 1.5, Google Maps ne fait plus partie du SDK à proprement parler mais a été déplacé dans les API supplémentaires de Google, qui sont des extensions au SDK de base. Ce système d'extension fournit des points d'entrée aux autres sous-systèmes qui peuvent se trouver sur certains terminaux mais pas sur d'autres.

En réalité, Google Maps ne fait pas partie du projet open-source Android, et il existera nécessairement des terminaux qui n'en disposeront pas à cause des problèmes de licence. Dans l'ensemble, le fait que Google Maps soit une extension n'affectera pas votre développement habituel à condition de ne pas oublier les points suivants :

- Vous devrez créer votre projet pour qu'il utilise la cible 3 (`-t 3`), afin d'être sûr que les API de Google Maps sont disponibles.
- Pour tester l'intégration de Google Maps, vous aurez également besoin d'un AVD qui utilise la cible 3 (`-t 3`).
- Inversement, pour tester votre application dans un environnement Android 1.5 sans Google Maps, vous devrez créer un AVD qui utilise la cible 2 (`-t 2`).

Les composants essentiels

Pour insérer une carte géographique dans une application, le plus simple consiste à créer une sous-classe de `MapActivity`. Comme `ListActivity`, qui enveloppe une partie des détails cachés derrière une activité dominée par une `ListView`, `MapActivity` gère une partie de la configuration d'une activité dominée par une `MapView`.

1. http://www.openstreetmap.org/.

Dans le fichier de layout de la sous-classe de MapActivity, vous devez ajouter un élément qui, actuellement, s'appelle com.google.android.maps.MapView. Il s'agit ici d'un nom totalement développé qui ajoute le nom de paquetage complet au nom de la classe (cette notation est nécessaire car MapView ne se trouve pas dans l'espace de noms com.google.android.widget). Vous pouvez donner la valeur que vous souhaitez à l'attribut android:id du widget MapView et gérer tous les détails lui permettant de s'afficher correctement à côté des autres widgets.

Vous devez cependant préciser les attributs suivants :

- android:apiKey. Dans une version de l'application en production, cet attribut doit contenir une clé de l'API Google Maps (voir plus loin).
- android:clickable = "true". Si vous voulez que les utilisateurs puissent cliquer sur la carte et la faire défiler.

Voici, par exemple, le contenu du fichier layout principal de l'application Maps/NooYawk :

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <com.google.android.maps.MapView android:id="@+id/map"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:apiKey="<VOTRE_CLÉ_API>"
    android:clickable="true" />
  <LinearLayout android:id="@+id/zoom"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true" />
</RelativeLayout>
```

Nous présenterons ces mystérieux LinearLayout zoom et apiKey plus bas dans ce chapitre.

Vous devez également ajouter deux informations supplémentaires à votre fichier AndroidManifest.xml :

- Les permissions INTERNET et ACCESS_COARSE_LOCATION.
- Dans l'élément <application>, ajoutez un élément <uses-library> avec l'attribut android:name = "com.google.android.maps" pour indiquer que vous utilisez l'une des API facultatives d'Android.

Voici le fichier AndroidManifest.xml du projet NooYawk :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.maps">
  <uses-permission android:name="android.permission.INTERNET" />
```

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

    <application android:label="@string/app_name">
      <uses-library android:name="com.google.android.maps" />
      <activity android:name=".NooYawk" android:label="@string/app_name">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
    </application>
  </manifest>
```

Avec la sous-classe de `MapActivity`, c'est à peu près tout ce dont vous avez besoin pour débuter. Si vous ne faites rien d'autre, que vous compiliez ce projet et que vous l'installiez dans l'émulateur, vous obtiendrez une belle carte du monde. Notez, cependant, que `MapActivity` est une classe abstraite et que vous devez donc implémenter la méthode `isRouteDisplayed()` pour préciser si vous fournissez ou non une gestion des itinéraires.

En théorie, l'utilisateur doit pouvoir faire défiler la carte en utilisant le pad directionnel. Cependant, ce n'est pas très pratique lorsque l'on a le monde entier dans sa main...

Une carte du monde n'étant pas très utile en elle-même, nous devons lui ajouter quelques fonctionnalités.

Testez votre contrôle

Pour trouver votre widget `MapView`, il suffit, comme d'habitude, d'appeler la méthode `findViewById()`. Le widget lui-même fournit la méthode `getMapController()`. Entre le `MapView` et le `MapController`, vous disposez d'un bon nombre de possibilités pour déterminer ce qu'affiche la carte et la façon dont elle se comporte ; les sections suivantes présentent le zoom et le centrage, qui sont sûrement celles que vous utiliserez le plus.

Zoom

La carte du monde avec laquelle vous démarrez est plutôt vaste. Sur un téléphone, on préfère généralement consulter une carte ayant une étendue plus réduite – quelques pâtés de maisons, par exemple.

Vous pouvez contrôler directement le niveau du zoom grâce à la méthode `setZoom()` de `MapController`. Celle-ci attend un paramètre entier représentant le niveau du zoom, où `1` représente la vue du monde entier et `21`, le plus fort grossissement que vous pouvez obtenir. Chaque niveau double la résolution effective par rapport au niveau précédent : au niveau `1`, l'équateur fait 256 pixels de large et il en fait 268 435 456 au niveau `21`. L'écran du téléphone n'ayant sûrement pas autant de pixels ; l'utilisateur ne verra donc qu'une petite partie de la carte centrée sur un endroit du globe. Le niveau `16` montrera plusieurs

pâtés de maisons dans chaque dimension et constitue généralement un bon point de départ pour vos essais.

Si vous souhaitez que les utilisateurs aient le droit de changer le niveau du zoom, utilisez l'appel `setBuiltInZoomControls(true)` : il pourra alors utiliser les contrôles de zoom qui se trouvent en bas de la carte.

Centrage

Généralement, quel que soit le niveau du zoom, vous voudrez contrôler ce qui est affiché sur la carte : la position courante de l'utilisateur ou un emplacement sauvegardé avec d'autres données de votre activité, par exemple. Pour changer la position de la carte, appelez la méthode `setCenter()` de `MapController`.

Cette méthode prend un objet `GeoPoint` en paramètre. Un `GeoPoint` représente un emplacement exprimé par une latitude et une longitude. En réalité, il les stocke sous la forme d'entiers en multipliant leurs vraies valeurs par `1E6`, ce qui permet d'économiser un peu de mémoire par rapport à des `float` ou à des `double` et d'accélérer un peu la conversion d'un `GeoPoint` en position sur la carte. En revanche, vous ne devez pas oublier ce facteur de `1E6`.

Terrain accidenté

Tout comme sur votre ordinateur de bureau, vous pouvez afficher les images satellites avec Google Maps et Android.

`MapView` offre la méthode `toggleSatellite()`, qui, comme son nom l'indique, permet d'activer ou de désactiver la vue satellite de la surface présentée sur la carte. Vous pouvez faire en sorte de laisser à l'utilisateur le soin de faire ce choix à partir d'un menu ou, comme dans `NooYawk`, *via* des touches :

```
@Override
 public boolean onKeyDown(int keyCode, KeyEvent event) {
   if (keyCode == KeyEvent.KEYCODE_S) {
     map.setSatellite(!map.isSatellite());
     return(true);
   }
   else if (keyCode == KeyEvent.KEYCODE_Z) {
     map.displayZoomControls(true);
     return(true);
   }

   return(super.onKeyDown(keyCode, event));
 }
```

Couches sur couches

Si vous avez déjà utilisé la version complète de Google Maps, vous avez sûrement déjà vu que l'on pouvait déposer des choses sur la carte elle-même : les "repères", par exemple, qui indiquent les emplacements des points d'intérêt proches de la position que vous avez demandée. En termes de carte – et également pour la plupart des éditeurs graphiques sérieux –, ces repères sont placés sur une couche distincte de celle de la carte elle-même et ce que vous voyez au final est la superposition de ces deux couches.

Android permet de créer de telles couches, afin de marquer les cartes en fonction des choix de l'utilisateur et des besoins de votre application. `NooYawk`, par exemple, utilise une couche pour montrer les emplacements des immeubles sélectionnés dans Manhattan.

Classes `Overlay`

Toute couche ajoutée à votre carte doit être implémentée comme une sous-classe d'`Overlay`. Si vous voulez simplement ajouter des repères, vous pouvez utiliser la sous-classe `ItemizedOverlay`, qui vous simplifiera la tâche.

Pour attacher une couche à votre carte, il suffit d'appeler la méthode `getOverlays()` de votre objet `MapView` et d'ajouter votre instance d'`Overlay` avec `add()` :

```
marker.setBounds(0, 0, marker.getIntrinsicWidth(),
                       marker.getIntrinsicHeight());
map.getOverlays().add(new SitesOverlay(marker));
```

Nous expliquerons un peu plus loin le rôle de `marker`.

Affichage d'`ItemizedOverlay`

Comme son nom l'indique, `ItemizedOverlay` permet de fournir une liste de points d'intérêt (des instances d'`OverlayItem`) pour les afficher sur la carte. La couche gère ensuite l'essentiel du dessin pour vous, mais vous devez toutefois effectuer les opérations suivantes :

- Dérivez votre sous-classe (`SitesOverlay`, dans notre exemple) d'`ItemizedOverlay<OverlayItem>`.
- Dans le constructeur, mettez en place la liste des instances `OverlayItem` et appelez `populate()` lorsqu'elles sont prêtes à être utilisées par la couche.
- Implémentez `size()` pour qu'elle renvoie le nombre d'éléments qui devront être gérés par la couche.
- Redéfinissez `createItem()` pour qu'elle renvoie l'instance `OverlayItem` correspondant à l'indice qui lui est passé en paramètre.

- Lors de l'instanciation de la sous-classe d'`ItemizedOverlay`, fournissez-lui un objet `Drawable` représentant l'icône par défaut de chaque élément (une épinglette, par exemple).

Le `marker` que l'on passe au constructeur de `NooYawk` est le `Drawable` utilisé en dernier recours – il affiche une épinglette.

Vous pouvez également redéfinir `draw()` pour mieux gérer l'ombre de vos marqueurs. Bien que la carte fournisse une ombre, il peut être utile de l'aider un peu en lui indiquant où se trouve le bas de l'icône, afin qu'elle puisse en tenir compte pour l'ombrage.

Voici, par exemple, le code de la classe `SitesOverlay` :

```
private class SitesOverlay extends ItemizedOverlay<OverlayItem> {
  private List<OverlayItem> items=new ArrayList<OverlayItem>();
  private Drawable marker=null;
  public SitesOverlay(Drawable marker) {
    super(marker);
    this.marker=marker;
    items.add(new OverlayItem(getPoint(40.748963847316034,
                                       -73.96807193756104),
                              "UN", "Nations Unies"));
    items.add(new OverlayItem(getPoint(40.76866299974387,
                                       -73.98268461227417),
                              "Lincoln Center",
                              "La maison du Jazz"));
    items.add(new OverlayItem(getPoint(40.765136435316755,
                                       -73.97989511489868),
                              "Carnegie Hall",
            "Entraînez-vous avant d'y jouer !"));
    items.add(new OverlayItem(getPoint(40.70686417491799,
                                       -74.01572942733765),
                              "The Downtown Club",
                "Le lieu d'origine du trophée Heisman"));

    populate();
  }
  @Override
  protected OverlayItem createItem(int i) {
    return(items.get(i));
  }
  @Override
  public void draw(Canvas canvas, MapView mapView,
                   boolean shadow) {
    super.draw(canvas, mapView, shadow);
    boundCenterBottom(marker);
  }
  @Override
  protected boolean onTap(int i) {
    Toast.makeText(NooYawk.this,
                   items.get(i).getSnippet(),
```

```
                    Toast.LENGTH_SHORT).show();
      return(true);
    }
    @Override
    public int size() {
      return(items.size());
    }
  }
```

Gestion de l'écran tactile

Une sous-classe d'`Overlay` peut également implémenter `onTap()`pour être prévenue lorsque l'utilisateur touche la carte afin que la couche ajuste ce qu'elle affiche. Dans Google Maps, cliquer sur une épinglette fait surgir une bulle d'information consacrée à l'emplacement marqué, par exemple : grâce à `onTap()`, vous pouvez obtenir le même résultat avec Android.

La méthode `onTap()` d'`ItemizedOverlay` prend en paramètre l'indice de l'objet `OverlayItem` sur lequel on a cliqué. C'est ensuite à vous de traiter cet événement.

Dans le cas de la classe `SitesOverlay` que nous venons de présenter, le code d'`onTap()` est le suivant :

```
@Override
protected boolean onTap(int i) {
  Toast.makeText(NooYawk.this,
                 items.get(i).getSnippet(),
                 Toast.LENGTH_SHORT).show();
  return(true);
}
```

Ici, on lève simplement un `Toast` contenant le texte associé à l'`OverlayItem` et l'on renvoie `true` pour indiquer que l'on a géré le toucher de cet objet.

Moi et `MyLocationOverlay`

Android dispose d'une couche intégrée permettant de gérer deux scénarios classiques :

- l'affichage de votre position sur la carte, en fonction du GPS ou d'un autre fournisseur de localisation ;
- l'affichage de la direction vers laquelle vous vous dirigez, en fonction de la boussole intégrée lorsqu'elle est disponible.

Il vous suffit pour cela de créer une instance de `MyLocationOverlay`, de l'ajouter à la liste des couches de votre `MapView` et d'activer et de désactiver ces fonctionnalités aux moments opportuns.

La notion de "moments opportuns" est liée à l'économie de la batterie. Comme il n'y a aucune raison de mettre à jour des emplacements ou des directions lorsque l'activité est en pause, il est conseillé d'activer ces fonctionnalités dans `onResume()` et de les désactiver dans `onPause()`.

Pour que `NooYawk` affiche une boussole dans `MyLocationOverlay`, par exemple, nous devons d'abord créer la couche et l'ajouter à la liste des couches :

```
me=new MyLocationOverlay(this, map);
map.getOverlays().add(me);
```

Puis nous activons et désactivons cette boussole lorsque cela est nécessaire :

```
@Override
public void onResume() {
  super.onResume();
  me.enableCompass();
}
@Override
public void onPause() {
  super.onPause();
  me.disableCompass();
}
```

La clé de tout

Si vous compilez le projet `NooYawk` et que vous l'installiez dans votre émulateur, vous verrez sûrement un écran montrant une grille et deux épinglettes, mais pas de carte.

La raison en est que la clé de l'API dans le code source n'est pas valide pour votre machine de développement. Vous devez donc produire votre propre clé pour l'utiliser avec votre application.

Le site web d'Android[1] donne toutes les instructions nécessaires pour produire ces clés, que ce soit pour le développement ou pour la production. Pour rester brefs, nous nous intéresserons ici au cas particulier de l'exécution de `NooYawk` dans votre émulateur. Vous devez effectuer les étapes suivantes :

1. Allez sur la page d'inscription pour la clé de l'API et lisez les termes d'utilisation.
2. Relisez ces termes et soyez absolument sûr que vous les approuvez.
3. Recherchez la signature MD5 du certificat utilisé pour signer vos applications en mode debug (voir ci-après).

1. http://code.google.com/android/toolbox/apis/mapkey.html.

4. Sur la page d'inscription pour la clé de l'API, collez cette signature MD5 et envoyez le formulaire.
5. Sur la page de réponse, copiez la clé de l'API et collez-la dans la valeur de l'attribut `android:apiKey` du layout de votre `MapView`.

La partie la plus compliquée consiste à trouver la signature MD5 du certificat utilisé pour signer vos applications en mode debug... et une bonne partie de cette complexité consiste à comprendre le concept.

Toutes les applications Android sont signées à l'aide d'une signature numérique produite à partir d'un certificat. Vous recevez automatiquement un certificat de débogage lorsque vous installez le SDK et il faut suivre un autre processus pour créer un certificat autosigné utilisable avec vos applications en production. Ce processus nécessite d'utiliser les outils `keytool` et `jarsigner` de Java. Pour obtenir votre clé d'API, vous n'avez besoin que de `keytool`.

Si vous utilisez OS X ou Linux, faites la commande suivante pour obtenir la signature MD5 de votre certificat de débogage :

```
keytool -list -alias androiddebugkey -keystore ~/.android/debug.keystore
 -storepass android -keypass android
```

Sur les autres plates-formes de développement, vous devrez remplacer la valeur de `-keystore` par l'emplacement sur votre machine et votre compte utilisateur :

- Windows XP : `C:\Documents et Settings\<utilisateur>\Local Settings\ApplicationData\Android\debug.keystore`.
- Windows Vista : `C:\Users\<utilisateur>\AppData\Local\Android\debug.keystore` (où *<utilisateur>* est le nom de votre compte).

La seconde ligne du résultat qui s'affiche contient votre signature MD5, qui est une suite de paires de chiffres hexadécimaux séparées par des caractères deux-points.