Géolocalisation sous Android

Rapport Final

GIRAUD Eliot & JOUSSELIN Antoine

GIRAUD Eliot
JOUSSELIN Antoine

Sommaire

Contexte et définition du problème

Depuis Android 7, les données brutes de navigation sont accessibles via l'API 24 des appareils possédant ce système d'exploitation. Ces données peuvent ainsi être utilisées pour développer des algorithmes de navigation. Ces algorithmes permettront de développer des outils de localisation plus précis que ceux existants.

L'Institut français des sciences et technologies des transports, de l'aménagement et des réseaux (IFSTTAR), un acteur majeur dans l'intégration des nouvelles technologies liées à la navigation, travaille sur ce type de solutions. L'institut a ainsi créé l'application GeolocPVT dont le code est en open source sur GitLab. Le but de cette application est de calculer la position uniquement à partir des satellites. Elle n'a pas pour objectif de concurrencer les grandes applications telle que GoogleMaps qui utilisent une multitude de données pour calculer la position. A terme, les calculs effectués par l'application pourraient être intégré comme bibliothèque de ces applications pour la partie GNSS.

Cette application a été développée sur la base de téléphones possédant le système d'exploitation Android 9. Plus précisément, GeolocPVT fonctionne parfaitement sur le téléphone Xiaomi Mi 8. Or, des téléphones plus anciens (avec un autre système d'exploitation, une dimension d'écran différente) ou n'ayant pas accès à toutes les constellations GNSS et les fréquences associées seront destinés à utiliser cette application à l'avenir. Il faut donc que l'application puisse fonctionner sur tous types d'appareils ayant au minimum Android 7 comme système d'exploitation.

Ci-dessous quelques exemples des corrections à effectuer :

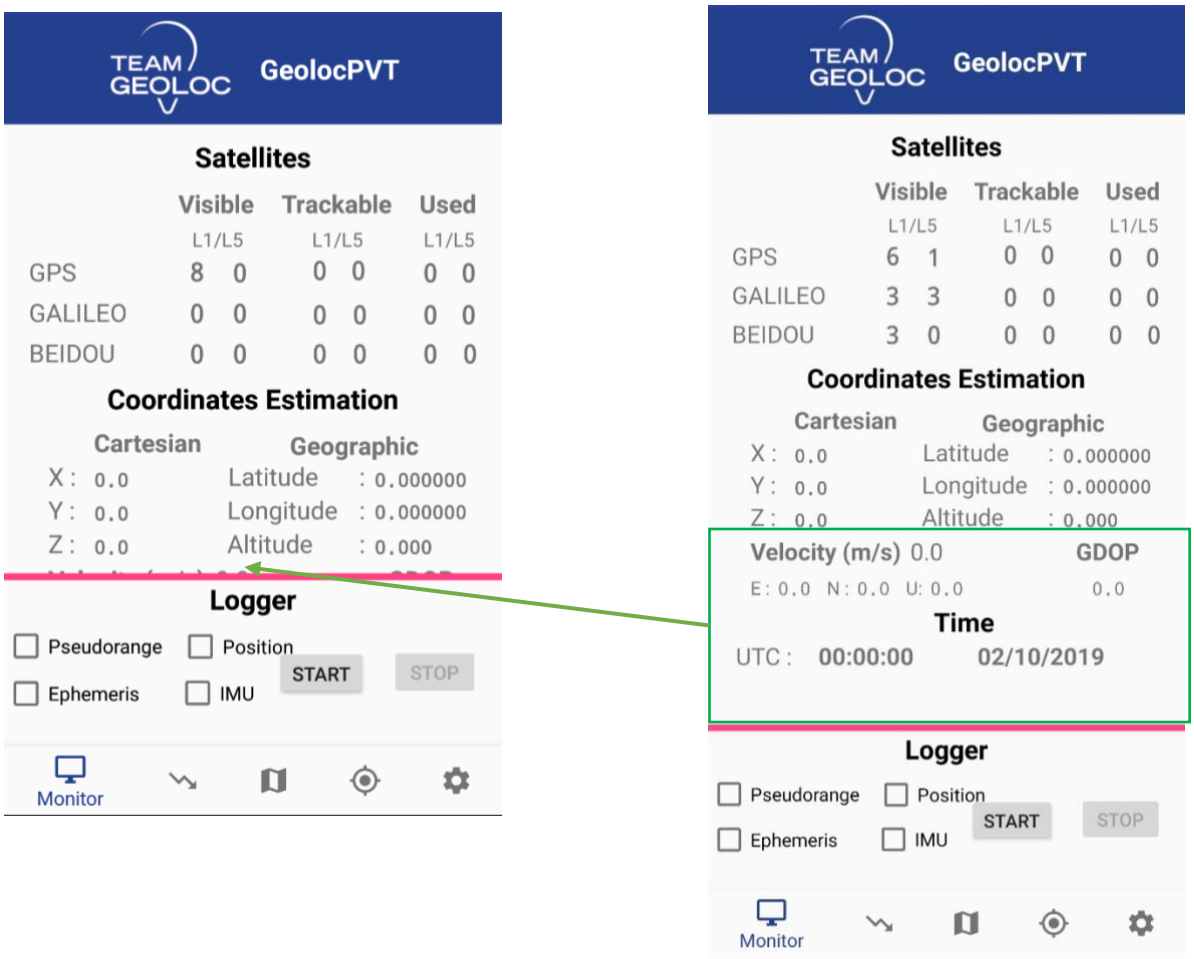

Figure 1 : Comparaison de l'affichage entre Huawei P10 (à gauche) et Huawei Mate 20 Pro (à droite)

GIRAUD Eliot
JOUSSELIN Antoine

Sur l'image ci-dessus, on remarque que l'affichage de l'application ne s'adapte pas à la taille de l'écran.

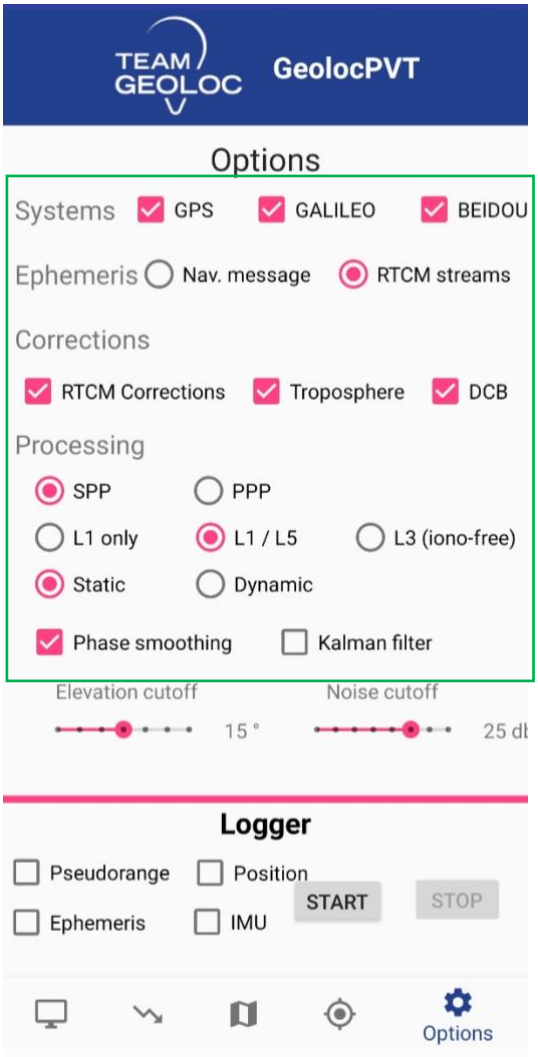

Figure 2 : Choix des différentes options possibles pour la localisation

Ici la zone concernée est la sélection de constellation et de fréquence, à partir de laquelle on souhaite calculer notre localisation. Or, sur l'image de droite qui concerne un Android 9 (Huawei Mate 20 Pro) on sait que toutes les constellations sont accessibles. Cependant, pour le téléphone de gauche un nombre restreint de constellations sont accessibles mais on peut tout de même toutes les sélectionner dans les options, idem pour les fréquences.

Objectifs

Pour rappel voici les différents objectifs fixés au début du projet :

- Adapter le code source pour le rendre fonctionnel jusqu'à la version Android 7 :
  - Faire le diagramme des classes
  - Adapter le fragment *Options* afin qu'il ne propose à l'utilisateur que les constellations, le mode d'obtention des éphémérides et les fréquences disponibles en fonction du téléphone utilisé
  - Écrire la Javadoc
- Rendre l'application active en arrière-plan :
  - Implémenter la méthode *onPause()*
  - Implémenter la méthode *onResume()*
- Adapter l'affichage selon les caractéristiques de l'appareil :
  - Adapter l'affichage selon la taille de l'écran (scroll ou responsive)

Périmètre

L'adaptation du code source pour un Android 7 n'affecte pas les utilisateurs d'un Android 9. Elle est bénéfique pour les utilisateurs d'un Android 7 ainsi que pour les personnes ayant un téléphone ne captant pas toutes les constellations GNSS ou ayant une taille d'écran différente du Xiaomi Mi 8. Au cours de projet, l'application n'a pas été directement impactée puisqu'une copie du code source, accessible à l'adresse : https://gitlab.com/TeamGEOLOC/geolocpvt, a été faite et nous avons travaillé dessus.

Description fonctionnelle

Développement de GeolocPVT pour Android 7

Fonction : Avoir une interface fonctionnelle de GeolocPVT sur Android 7	
Objectif	Rendre l'application utilisable en mode dégradé sur Android 7 et Android 8
Description	Modification du code source du fragment 'Options' permettant de ne pouvoir sélectionner que les constellations GNSS, mode d'obtention des éphémérides et fréquences possibles en fonction des puces (chipset) présentes dans le téléphone.
Contraintes / Règles de gestion	Contraintes liées à Android Studio et au téléphone
Niveau de priorité	Haute

Sous fonction :

- Affichage de l'application adapté à chaque écran

Fonctionnement de l'application en background

Fonction : Fonctionnement de l'application en background	
Objectif	Accéder aux coordonnées de localisation à chaque instant.
Description	L'application doit fonctionner en background pour que les algorithmes calculent la localisation en continue. Pour cela les fonctions onPause() et onResume() seront implémentées.
Contraintes / Règles de gestion	Contraintes liées à Android Studio
Niveau de priorité	Moyenne

GIRAUD Eliot
JOUSSELIN Antoine

Budget, Ressources et planning

Pour la réalisation de ce projet nous avons utilisé les ressources suivantes :

- GitLab : site sur lequel se trouve le code en open source et une copie du projet. Copie sur laquelle nous avons travaillé.
- Android Studio qui nous a permis de tester les modifications de code réalisées ainsi que de générer l'APK.

Aucun budget n'a été nécessaire pour ce projet.

Voici le planning qui a été suivi afin d'atteindre le maximum d'objectifs :

Dates \ Tâches	7 Oct – 13 Oct	14 Oct – 20 Oct	21 Oct – 27 Oct	28 Oct – 3 Nov	4 Nov – 10 Nov	11 Nov – 17 Nov	18 Nov – 24 Nov	25 Nov – 1er Dec	2 Dec – 8 Dec	9 Dec – 15 Dec	16 Dec – 20 Dec
Familiarisation avec le code source, l'application et Android Studio											
Modification du code pour Android 7 et autres appareils											
Implémentation des fonctions pour le fonctionnement de l'application en background											
Gestion de l'affichage sur différents écrans											
Affichage de la trace du déplacement en temps réel											Manque de temps
Ajout des derniers commentaires et génération de la javadoc											

Diagramme UML

Dans un premier temps, nous avons fait le diagramme d'activité du package 'App' du code de l'application. Ce diagramme représente l'ensemble des fragments actifs de l'application ainsi que leurs fonctionnalités et les liaisons entre eux. Notre travail a majoritairement porté sur le fragment Options. Plus particulièrement, sur le choix des systèmes, des fréquences et des éphémérides de fréquence.

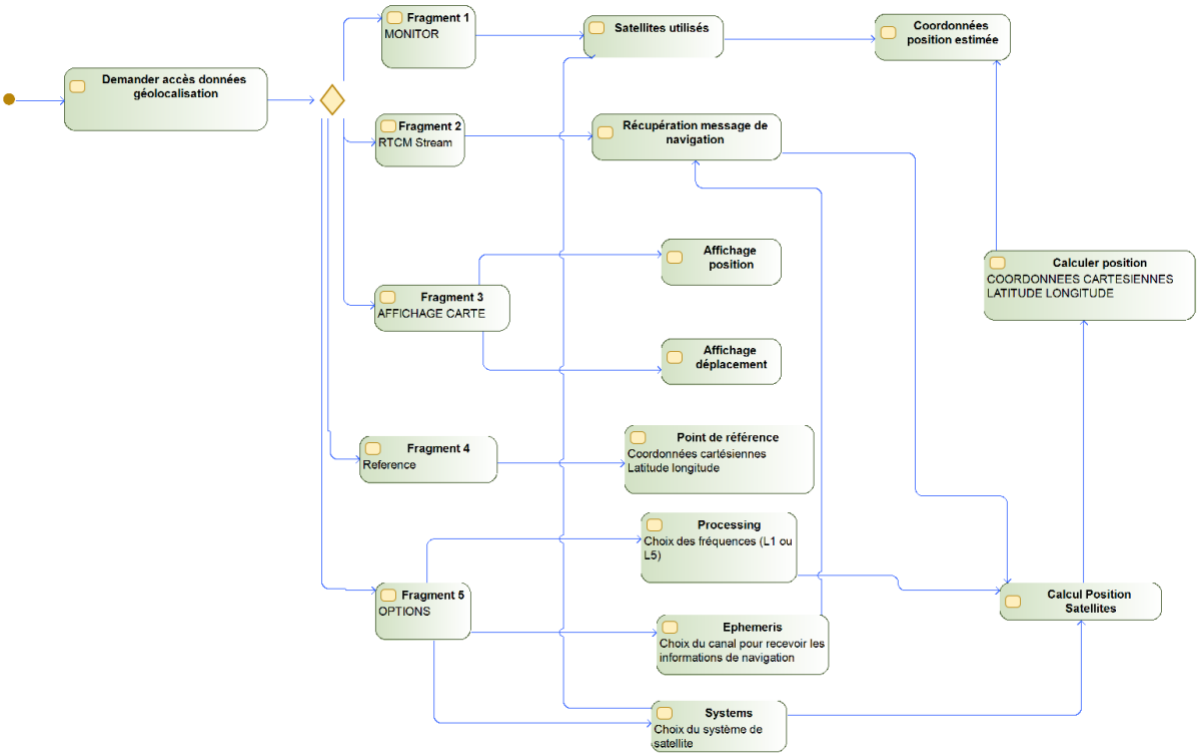

Figure 3 : Diagramme d'activité des modifications apportées à l'application

Dans un second temps, nous avons créé le diagramme de classes de l'ensemble de l'application. Ce diagramme a pour but d'aider à comprendre le fonctionnement de cette dernière ainsi que de se repérer plus facilement dans le code.

Analyse du code de l'application

Afin d'être le plus efficace et le plus rapide possible nous sommes partis d'une analyse globale avant de rentrer plus en précision dans les différentes parties de code.

Pour commencer, nous nous sommes renseignés sur le principe de développement d'une application Android qui est quelque chose que nous n'avions jamais fait au-par-avant. Cette étape s'est faite à l'aide de nombreux sites internet tels que OpenClassrooms ou le site des développeurs Android. Ces sites nous ont donné les principes de base pour qu'une application fonctionne ainsi que la compréhension de Android Studio, environnement de développement (IDE) le plus utilisé pour l'écriture du code d'une application Android.

Après cette étape générale nous sommes entrés plus en profondeur dans le code. Pour cela nous l'avons analysé package par package à l'aide de la Javadoc. Voici ce que nous en avons ressorti :

- **Package app :** ce package gère l'ensemble de l'affichage des données sur l'application ainsi que leur mise à jour en temps réel. Cette actualisation est gérée par les classes du dossier *fragments* qui héritent de la classe *Fragment*. A ces différentes classes est associée la classe *MainActivity*. Cette classe gère le fonctionnement global de l'application. C'est dans cette dernière que les méthodes *onCreate()*, *onStart()* et toutes celles qui concernent le fonctionnement de l'application, sont implémentées.
- **Package geoloclib :** ce package introduit les classes et méthodes qui analysent les données provenant des satellites des différentes constellations et contiennent les algorithmes de calcul de position. On peut citer l'exemple de la classe *SatellitePositionGNSS* qui permet de calculer la position du satellite pour en déduire celle du téléphone.
- **Package gogps :** ce dernier package est utilisé pour récupérer les éphémérides et les corrections d'éphéméride via les flux de données RTCM. Pour ces méthodes de calcul, une connexion internet est nécessaire.

Puis, à l'aide de la Javadoc nous avons pu déterminer les liens entre les différentes classes.

Enfin, en analysant le code à l'intérieur des différentes classes ainsi que les commentaires nous avons eu une vision globale du fonctionnement de l'application. Cette dernière s'appuie sur le package *android.location* de Java qui fournit de nombreux outils pour analyser les données envoyées par les satellites et reçues par le téléphone.

Amélioration de l'application GeolocPVT

Fragment Option

Comme cela a été mentionné précédemment dans le cahier des charges fonctionnel, l'objectif de cette partie est d'adapter l'affichage du fragment option de l'application en fonction des fonctionnalités présentes dans le téléphone.

Choix de la solution

Pour cette partie, le but était d'activer les différentes cases de *System* en fonction des constellations captées par le téléphone.

Cette partie a été assez délicate et périlleuse. En effet, notre première option consistait à s'introduire dans le chipset du téléphone via l'API pour trouver quelle constellation GNSS le téléphone était susceptible de détecter. Cependant, devant le manque d'information sur ce sujet et après de nombreuses recherches effectuées nous nous sommes tournés vers une autre solution.

L'autre solution, qui s'est finalement avérée pertinente, consiste à utiliser une partie du code déjà écrite. Dans le fragment *Monitor,* le nombre de satellites des différentes constellations (GPS, Galileo et Beidou) est affiché en temps réel. A partir de ces données, l'objectif est de détecter, lors du démarrage de l'application, le moment à partir duquel un satellite d'une des constellations est capté. Une fois ce dernier capté nous pouvons rendre active la checkbox de la constellation à laquelle il appartient.

Maintenant que nous avons expliqué la solution choisie nous allons décrire comment cette solution a été mise en place dans le code.

Dans un premier temps, nous avons trouvé l'endroit dans le code du fragment *Monitor* où est modifié l'affichage du nombre de satellites détectés. Nous avons créé trois booléens associés aux trois constellations GPS, Galileo et Beidou dont l'état initial est false. Ils sont regroupés dans un tableau appelé *tabConstel* dans le fragment *Monitor*. Ce tableau est utilisé dans la méthode *refreshData()*. Plus particulièrement, lorsque dans cette dernière, on réalise les tests pour savoir à quelle constellation appartient le satellite 'visible' (ligne 292). Ainsi, quand un satellite est détecté et que sa constellation est trouvée, le booléen associé à cette dernière devient vrai et le reste tout le temps que l'application est ouverte.

Puis, ce tableau est importé, à l'aide d'un getter, dans la méthode *refreshProcessingOptions()* du fragment *Options*. Dans cette méthode, à l'aide d'un if on teste si la constellation a été détectée. Si oui, alors on active la checkbox et inversement. On peut voir dans l'image ci-dessous un exemple avec le GPS :

```
179     private void refreshProcessingOptions()
180     {
181         // Check ticked systems*
182         //TODO Change Vector to Set
183         Vector<Integer> systemsEnabled = new Vector<>();
184         // import the tabConstellation and freqL5 from Monitor fragment in order
185         boolean constellation[] = MonitorFragment.getTabConstel();
186         boolean L5 = MonitorFragment.getFreqL5();
187         //test to know if GPS constellation is detected
188         if (constellation[0]) {
189             if (gpsCB.isChecked())
190             {
191                 systemsEnabled.add(GnssStatus.CONSTELLATION_GPS);
192             }
193             else
194             {
195                 if (systemsEnabled.contains(GnssStatus.CONSTELLATION_GPS)) {
196                     systemsEnabled.removeElement(GnssStatus.CONSTELLATION_GPS);
197                 }
198             }
199         }
200         else {
201             gpsCB.setEnabled(false);
202         }
```

Figure 4 : Exemple du test pour savoir si la constellation GPS est détectée

Dans l'image ci-dessus, on importe le tableau à la ligne 185, puis on effectue le test de détection à la ligne 188. Si ce test s'avère faux, alors on désactive la checkbox comme on peut le voir à la ligne 201 avec la méthode *setEnabled()* de la classe *android.widget.Checkbox*.

De plus, lors des différents essais réalisés nous nous sommes aperçus que les checkboxes étaient désactivées mais pas décochées. Cela peut être modifié en allant dans le fichier fragment_options.xml dans le dossier layout et en modifiant l'état par défaut des différentes cases.

Enfin, nous avons été confrontés à une erreur qui était que les booléens du tableau *tabConstel* que nous importions dans le fragment *Options* n'étaient pas statiques alors qu'on se situe dans un contexte statique. Sur ce dernier point nous n'avons pas réussi à comprendre pourquoi la méthode *refreshProcessingOptions()* était considérée comme statique. Ce problème a été résolu en définissant directement *tabConstel* comme statique. Cela a été rendu possible par le fait qu'il n'est pas propre à une instance de la classe *OptionsFragment* dans laquelle il est utilisé.

Choix de la fréquence de réception des éphémérides

Dans cette partie l'objectif était de désactiver la case *Nav Message* dans le cas où le téléphone n'était pas capable de recevoir les messages de navigation alors qu'ils peuvent tous les avoir via « RTCM stream » puisqu'il suffit d'une connexion internet.

Nous avons été confrontés à quelques difficultés qui nous ont empêché de réaliser cet objectif. Mais nous allons décrire une voie d'exploration possible.

Dans un premier temps, nous nous sommes aperçus que le bouton *Nav Message* n'était pas codé dans l'application. Néanmoins, ce dernier a bien été créé dans le fichier fragment_options.xml (ce qui explique qu'on le visualise dans le fragment *Options*). Il suffit donc de créer un attribut RadioButton

que l'on peut appeler *navMes* et de lui associer *Nav Message* du fichier fragment_options.xml. C'est ce qui a été fait.

Ensuite, afin de savoir si le téléphone est capable ou non de recevoir les messages de navigation, il existe une constante 'STATUS_NOT_SUPPORTED' dans la classe android.location.GnssNavigationMessage.Callback qui permet d'établir si le système est capable de recevoir les messages de navigation. Si le téléphone ne reçoit pas les messages, alors la constante vaut 0. En fonction de la valeur de cette constante on peut donc activer ou désactiver cette case.

Voici le lien vers la classe du package Android mentionnée ci-dessus :

https://developer.android.com/reference/android/location/GnssNavigationMessage.Callback

Ainsi, il suffit d'importer la classe et d'extraire la constante pour savoir si le téléphone est apte ou non.

Néanmoins, pour le moment, aucune méthode en rapport à ce RadioButton n'a été développée. Il faudra prévoir d'ajouter le test lorsque qu'une telle méthode sera prévue.

Choix de la fréquence

Ici, le but était de désactiver les boutons L1/L5 et L3 (iono-free) (c'est un mélange de fréquence L1 et L5) en fonction de si le téléphone est capable de capter la fréquence L5. En effet, tous les téléphones peuvent capter la fréquence L1 mais tous ne captent pas la fréquence L5.

L'activation des RadioButtons pour le choix des fréquences à utiliser dans le calcul de la position est basée sur le même principe que l'activation des checkboxes des constellations. Au même endroit, dans le code du fragment *Monitor*, se fait à la fois l'incrémentation du nombre de satellites captés et l'incrémentation du compteur qui permet de savoir si le message reçu par le satellite est émis à la fréquence L5 ou à la fréquence L1 : dans la méthode *refreshData().*

La seule différence repose sur le fait que si la fréquence L5 n'est pas captée, cela implique la désactivation des deux RadioButtons L1/L5 et L3 (iono-free).

Nous avons donc créé un autre attribut statique de type booléen nommé *freqL5* et initialisé à la valeur false. Dès lors qu'une fréquence L5 est détectée, quelle que soit la constellation GNSS (d'où le fait de l'avoir écrit dans toutes les constellations), la valeur du booléen passe à true et elle le reste.

En parallèle de cela, on importe cet attribut dans la méthode *refreshProcessingOptions()* de *OptionsFragment*, que l'on renomme L5, et on active les deux RadioButtons en fonction de la valeur du booléen, à l'aide d'une condition if qu'on peut voir dans l'image ci-dessous :

```
262     //Test to know if the phone can capture L5 frequency
263     if (!L5){
264         monoFreqRB.setEnabled(true);
265         dualFreqRB.setEnabled(false);
266         ionoFreeRB.setEnabled(false);
267     }
```

Figure 5 : Test pour savoir si la fréquence L5 est captée

De plus, lors des différents essais réalisés nous nous sommes aperçus que les *RadioButtons* sélectionnés par défaut ne sont pas les bons : celui coché par défaut est L1/L5 alors qu'il devrait être L1 qui est la fréquence captée par tous les téléphones. Nous avons résolu en modifiant le fichier fragment_options.xlm.

Implémentation des méthodes *onPause()* et *onResume()*

Pour que l'application continue de fonctionner en arrière-plan, il faut implémenter deux méthodes :

- *onPause()* : méthode qui définit le comportement de l'application lorsque celle-ci passe en arrière-plan.
- *onResume()* : méthode qui définit le comportement de l'application lorsque celle-ci passe de l'arrière-plan au premier plan.

Ces méthodes viennent en complément de la méthode *onCreate()* qui créée les différents objets nécessaires au fonctionnement de l'application lors de son démarrage. Elles font partie de ce que l'on appelle le concept de cycle de vie de l'activité, et doivent donc être écrites dans la classe *MainActivity*.

Malheureusement nous avons manqué de temps pour les implémenter, mais plus d'informations au sujet du cycle de vie de l'activité sont disponible ici :
https://developer.android.com/guide/components/activities/activity-lifecycle#java

Mise en place d'un scroll

Pour permettre l'affichage de l'application sur tout type d'écran, nous avons mis un place un scroll. Nous avons fait le choix de le mettre en place seulement sur les fragments *Options* et *Monitor*, car ce sont les seuls avec assez de contenu pour poser un problème.

Cette mise en place a été relativement simple : elle a consisté à encapsuler la balise *RelativeLayout* dans une balise *ScrollView* dans les fichiers fragment_monitor.xml et fragment_options.xml.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="wrap_content"
5      android:fillViewport="false">
6
7      <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
8          xmlns:app="http://schemas.android.com/apk/res-auto"
9          xmlns:tools="http://schemas.android.com/tools"
10         android:layout_width="match_parent"
11         android:layout_height="match_parent">
12
```

Figure 6 : Bout de code ayant permis l'ajout du scroll

Estimation des coûts du projet

Pour ce projet nous avons travaillé le nombre d'heures qui nous étaient allouées, soit 64h au global.

Évolutions potentielles à effectuer sur l'application

Tout d'abord, nous pensons que le travail effectué tout au long du projet peut être optimisé. Ainsi, voici les évolutions auxquelles nous avons pensé :

- A propos de la détection des différentes constellations par le téléphone, il pourrait être intéressant de créer une méthode permettant de savoir quelle constellation est captée lors de la première ouverture de l'application et garder ça en mémoire pour les prochaines fois ou l'application sera utilisée. Cela éviterait un léger temps de latence présent lors de la détection des constellations. Sur ce point-là nous pensons que cela peut se faire dans la classe *MainActivity*.
- Pour le choix des éphémérides, l'objectif sera de mettre en place la proposition que nous avons fait précédemment.
- Mettre à jour régulièrement le diagramme de classe dont les fichiers Modelio ont été transférés à la fin du projet.

De plus, durant notre travail tout au long du projet, nous avons repéré quelques points pouvant être améliorés sur l'application :

- Sur les différents téléphones utilisés, dans le fragment *Monitor* l'affichage des satellites *Visible* et *Trackable* est efficace. Cependant l'affichage du nombre de satellites *Used* ne semble pas fonctionner.
- Dans le même fragment, les *Coordinates Estimation* restent constamment à 0.
- Dans le fragment *Map*, la localisation ne fonctionne pas. Il semblerait qu'aucune action ne se passe lorsqu'on clique sur le bouton.

Conclusion

Nous sommes satisfaits du travail effectué car nos objectifs principaux ont été atteint. Néanmoins quelques points n'ont pas pu être finalisés. Nous pensons avoir pris un peu trop de temps lors de la compréhension de l'ensemble du code de l'application, qui n'a pas été facilitée par le manque de documentation. De même, nous avons passé trop de temps pour trouver un moyen d'accéder aux chipsets du téléphone ce qui a nui à la réalisation des méthodes *onPause()* et *onResume().*

C'était la première fois que nous avions l'occasion de travailler sur le développement d'une application mobile, et ce fut une expérience enrichissante. Nous avons d'une part découvert l'univers Android et l'IDE Android Studio, et d'autre part appris à travailler à plusieurs sur une même application grâce à git. De plus, il est intéressant de se confronter à du code que nous n'avons pas écrit car cela entraîne un effort de compréhension que l'on retrouvera probablement en entreprise.