DMA - LAB2 - Positionnement en intérieur – Les iBeacons

Authors: Yanis Ouadahi, Rachel Tranchida, Eva Ray

1. Lister les balises à portée

Choix d'implémentation

Mise en place de la détection des beacons

Pour détecter les balises à proximité, nous avons utilisé la bibliothèque Android Beacon Library. La 1ère étape consiste à configurer le `BeaconManager` en ajoutant un `BeaconParser` pour chaque type de balise que nous souhaitons détecter. Dans notre cas, nous avons utilisé le format iBeacon qui est le plus courant. Le `BeaconParser` est configuré pour extraire les informations de l'annonce Bluetooth et les convertir en objets `Beacon` que nous pouvons utiliser dans notre code.

```
val beaconManager = BeaconManager.getInstanceForApplication(this)
beaconManager.beaconParsers.add(
    BeaconParser().setBeaconLayout("m:2-3=0215,i:4-19,i:20-21,i:22-23,p:24-24")
)
```

Ensuite, nous avons créé une `BeaconRegion` pour définir les critères de détection des balises. Ici, nous avons utilisé un `BeaconRegion` générique qui correspond à toutes les balises iBeacon. Nous avons ensuite ajouté un `Observer` pour écouter les balises détectées et démarré le monitoring et le scanning. Le `BeaconManager` va alors commencer à recevoir les annonces Bluetooth des balises à proximité.

```
val region = BeaconRegion("test", BeaconParser(), null, null, null)
beaconManager.getRegionViewModel(region).rangedBeacons.observe(this,
rangingObserver)
beaconManager.startRangingBeacons(region)
beaconManager.startMonitoring(region)
```

Enfin, nous avons créé un `Observer` pour écouter les balises détectées. Cet observer est responsable de mettre à jour la liste des balises à proximité dans le ViewModel de l'application. Il appelle la méthode `updateNearbyBeacons` du ViewModel pour mettre à jour la liste des balises détectées. C'est la méthode du ViewModel qui va ensuite mettre à jour l'interface utilisateur en passant par les LiveData `_nearbyBeacons` et `_closestBeacon`.

```
val rangingObserver = Observer<Collection<Beacon>> { beacons ->
    beaconsViewModel.updateNearbyBeacons(beacons.toList())
}
```

Mise en place d'un cache

Afin de permettre de lisser les annonces, nous avons mis en place un cache qui garde une map de tous les beacons détectés récemment. Nous avons décidé de laisser une période de grâce de 10 secondes à tous les beacons qui ne sont plus détectés. Cela veut dire que si un beacon n'est plus détecté, il apparait tout de même dans la liste des beacons proche dans l'application. Si ce beacon n'est pas détecté à nouveau dans les 10 secondes, il est effacé du cache. Pour mettre cela en place, nous avons dû ajouter un attribut `lastSeen` à la classe `PersistentBeacon` qui représente la dernière fois que le beacon a été détecté. Nous avons également modifié l'id du beacon pour qu'il soit unique et qu'il puisse être utilisé comme clé dans la map. Pour cela, l'id du beacon est maintenant une concaténation de son uuid, de son major et de son mineur. Pour représenter le cache, nous avons créé une classe `BeaconCache` qui contient une map dont la clé est l'id du beacon et la valeur est un objet `PersistentBeacon`. Cette classe contient une méthode `updateCache` qui met à jour le cache en fonction des beacons détectés et s'occupe de gérer la période de grâce. Le ViewModel de l'application contient une instance de cette classe et l'utilise pour mettre à jour la liste des beacons à afficher dans l'application.

Questions

> 1.1.1 Est-ce que toutes les balises à proximité sont présentes dans toutes les annonces de la librairie ? Que faut-il mettre en place pour permettre de « lisser » les annonces et ne pas perdre momentanément certaines balises ?

Non, toutes les balises à proximité ne sont pas nécessairement présentes dans chaque annonce de la librairie. Selon la documentation officielle de Android Beacon Library, les annonces Bluetooth sont reçues de manière intermittente et peuvent être manquées en raison de plusieurs facteurs:

- **Nature intermittente des signaux BLE** : Les balises transmettent typiquement leurs signaux à des intervalles de 100ms à 1s.
- **Scanning cyclique** : Le smartphone ne scanne pas en continu mais par cycles, ce qui implique que certaines annonces peuvent être manquées.
- **Interférences et obstacles** : Les obstacles physiques et interférences radio peuvent bloquer certaines transmissions.
- **Rotation des appareils** : L'orientation du téléphone peut affecter la réception du signal.

Pour "lisser" les annonces et éviter de perdre momentanément certaines balises, il faut mettre en place un mécanisme de persistance temporaire. Il est possible de faire cela en maintenant un cache. Voici une approche possible:

1. Ajouter un attribut `lastSeen` à la classe `PersistentBeacon` pour stocker le dernier moment où la balise a été détectée.
2. Garder une map de tous les beacons détectés récemment, où la clé est l'id du beacon (uuid + major + mineur) et la valeur est un objet `PersistentBeacon`.
3. À chaque nouvelle annonce, mettre à jour le cache avec les balises détectées en mettant à jour leur attribut `lastSeen`.
4. Conserver les balises non détectées dans le cache pendant une période de grâce (par exemple 5-10 secondes)
5. Supprimer les balises qui n'ont pas été détectées pendant plus longtemps que cette période de grâce.

Cette approche permet d'avoir une vue plus stable des balises à proximité, même si certaines ne sont pas détectées à chaque cycle d'annonce.

> 1.1.2 Nous souhaitons effectuer un positionnement en arrière-plan, à quel moment faut-il démarrer et éteindre le monitoring des balises ? Sans le mettre en place, que faudrait-il faire pour pouvoir continuer le monitoring alors que l'activité n'est plus active ?

Le moment idéal pour activer le monitoring des balises et lors de l'entrée dans une zone d'intérêt, comme un magasin , un musée ou une zone contenant de beacons. Cela peut être détecter à l'aide de la gélocalisation ou de Geofencing. De la même manière, lorsque l'utiisateur sort de la zone d'intérêt, il faudrait éteindre le monitoring des balises, ou lorsqu'aucune balise n'a été détectée depuis un certain temps.

On va activer le monitoring des balises en arrière plan lorsque l'application entre en arrière plan et/ou est fermée et que le monitoring est toujours nécessaire. Le monitoring peut être arrêté à l'arrêt de l'application, à la demande de l'utilisateur, ou lorsque l'application revient en premier plan Pour continuer le monitoring lorsque l'activité n'est plus active, on pourrait utiliser les services foreground d'android.

On pourrait définir une classe service pour notre beacon pour faire le monitoring des données en arrière plan. Il faudrait définir la méthode `startForeground()` et définir une notification à afficher lorsque l'application run en arrière plan.

Dans le AndroidManifest, il faut demander les autorisations pour le service foreground et déclarer le service que l'on vient de créer.

> 1.1.3 On souhaite trier la liste des balises détectées de la plus proche à la plus éloignée, quelles sont les valeurs présentes dans les annonces reçues qui nous permettraient de le faire ? Comment sont-elles calculées et quelle est leur fiabilité ? Hint : N'hésitez pas à mettre en place un filtre pour limiter la détection uniquement aux iBeacons de votre groupe, le numéro mineur des balises est indiqué sur celles-ci.

En consultant la documentation officielle de la librairie concernant l'estimation des distance, nous constatons que l'API fournit une estimation directe de la distance en mètres, basée sur le RSSI. Le RSSI est un indicateur du signal Bluetooth reçu : plus il est élevé (moins négatif), plus la balise est proche. La valeur à utiliser pour le tri est donc la distance estimée.

D'après la documentation, la manière la plus précise de calculer la distance basée sur le RSSI est obtenue en faisant une régression de puissance en utilisant une table connue de de valeurs distance/ RSSI pour un certain appareil. La fomrule utilisée est: `d=A*(r/t)^B+C` où

- `d` est la distance estimée en mètres
- `r` est le RSSI mesuré
- `t` est la puissance du signal à 1 mètre (TxPower)
- `A`, `B` et `C` sont des constantes spécifiques à l'appareil

Au niveau de la fiabilité, il ne faut pas attendre une trop grande précision pour la distance estimée. En effet, à 1 mètre, la variation peut être de 0.5 à 2 mètres, et au-delà, elle devient encore plus grande (exemple : à 20 mètres, l'erreur peut aller de 10 à 40 mètres). Il y a plusieurs raisons à cela :

- Les réflexions et obstacles (murs, objets, personnes).

- Les variations du signal sur le temps. La distance est estimée sur les 20 dernières secondes donc si l'appareil bouge, il faut attendre 20 secondes d'immobilité pour que la nouvele distance se stabilise (le filtrage appliqué par la bibliothèque tente de réduire cela).
- Les différences matérielles entre modèles de smartphones.

La stratégie de filtrage est donc la suivante :

- Récupérer la distance estimée fournie par l'API (beacon.getDistance()).
- Trier la liste des balises par ordre croissant de distance.
- Si besoin: appliquer un filtre pour n'afficher que les iBeacons du groupe (via le numéro mineur des balises).

Dans notre labo, nous avons finalement décidé de ne pas garder uniquement les beacons de notre groupe, mais nous aurions pu le faire facilement en déclarant un liste de mineurs à garder dans `BeaconCache` et en filtrant la liste des beacons détectés dans la méthode `updateCache`.

```
companion object {
    val GROUP_MINORS = intArrayOf(31, 38, 94)
}

fun updateCache(newBeacons: List<PersistentBeacon>): List<PersistentBeacon> {
    ...
    for (beacon in newBeacons) {
        // Only keep beacons from our lab group
        if (beacon.major !in GROUP_MINORS) continue
        ...
}
```

2. Déterminer sa position

Choix d'implémentation

Afin de stocker les positions du smartphone par rapport aux différentes balises, nous avons choisi d'utiliser une Map, déclarée dans le ViewModel, dont la **clé** est le numéro **mineur** de la balise et la **valeur** est le **nom de la "pièce"** dans laquelle se trouve le smartphone. On aurait pu choisir l'id complet de la balise (uuid + major + mineur), afin de ne pas avoir de collisions mais, dans le cadre de ce labo, utiliser les mineurs est suffisant. Le **facteur déterminant** la balise plus proche est la **distance** estimée fournie par l'API (beacon.getDistance()).

Questions

> 2.1.1 Comment pouvons-nous déterminer notre position ? Est-ce uniquement basé sur notion de proximité étudiée dans la question 1.1.3, selon vous est-ce que d'autres paramètres peuvent être pertinents ?

Comme déjà dit dans la réponse à la question 1.1.3, les paramètres pertinents sont:

- `distance`: estime la distance entre le smartphone et le beacon en mètres, peu précis, basée sur la comparaison du RSSI reçu avec la puissance du signal à 1 mètre (txPower).

- `rssi`: représente la puissance du signal Bluetooth reçu, plus le RSSI est élevé, plus le beacon est proche. Cette valeur fluctue beaucoup en raison des interférences environnementales.
- `txPower`: indique la puissance du signal Bluetooth à 1 mètre de la balise, permet de calibrer le RSSI. C'est une valeur fixée lors de la fabrication de l'iBeacon et intégrée à chaque paquet Bluetooth transmis, elle ne change pas avec le temps.

Ces paramètres sont en lien avec la notion de proximité. Cependant, d'autres paramètres peuvent être utilisés de manière intelligente pour améliorer la précision du positionnement. Dans ce labo, tous les beacons ont le même major mais on peut penser à une utilisation du major qui nous permettrait d'obtenir de plus amples informations sur la position de la balise. Par exemple, si on utilise les beacons dans un bâtiment, on pourrait imaginer que le major correspond à un étage et le mineur à une pièce. Grâce à cela, on pourrait savoir dans quelle pièce et à quel étage on se trouve, ce qui permet d'affiner le positionnement.

> 2.1.2 Les iBeacons sont conçus pour permettre du positionnement en intérieur. D'après l'expérience que vous avez acquise sur cette technologie dans ce laboratoire, quels sont les cas d'utilisation pour lesquels les iBeacons sont pleinement adaptés (minimum deux) ? Est-ce que vous voyez des limitations qui rendraient difficile leur utilisation pour certaines applications ?

Cas d'utilisation adaptés:

1. **Guidage et navigation en intérieur** : Les iBeacons sont bien adaptés aux environnements où le GPS est inefficace, comme les centres commerciaux, les musées, ou les aéroports. Ils permettent aux utilisateurs de se repérer facilement grâce à une application qui détecte leur position approximative en fonction des balises les plus proches et leur propose un itinéraire.
2. **Proximité de points d'intérêt** :Les commerces et espaces publics peuvent utiliser les iBeacons pour envoyer des notifications personnalisées aux utilisateurs lorsqu'ils passent à proximité d'un point d'intérêt. Par exemple, un magasin peut proposer une réduction sur un produit situé dans un rayon proche du client.

Limitations:

1. **Précision limitée** : Les estimations de distance basées sur le RSSI (Received Signal Strength Indicator) sont sujettes à des variations dues aux obstacles, aux interférences et aux différences entre les modèles d'appareils. Cela limite leur précision pour un positionnement précis.
2. **Latence et temps de stabilisation** : La technologie applique un lissage des mesures sur 20 secondes, ce qui entraîne un retard dans la mise à jour de la position. Cela peut être problématique pour des applications nécessitant un suivi en temps réel.
3. **Dépendance au Bluetooth activé** : L'utilisateur doit activer le Bluetooth sur son appareil, ce qui peut limiter l'adoption de cette technologie pour certaines applications.