Auteurs

Bastian Chollet, Kevin Ferati

Implémentation

MainActivity

- Dans `onCreate`, si les permissions sont accordées, l'application lance un scan en premier-plan :

```
 permissionsGranted.observe(this) {permGranted ->
            if (permGranted) {
                val manager = BeaconManager.getInstanceForApplication(this)
                manager.beaconParsers.add(BeaconParser().setBeaconLayout("m:2-
3=0215,i:4-19,i:20-21,i:22-23,p:24-24"))
                manager.addRangeNotifier(this)
                manager.startRangingBeacons(scanRegion)
                manager.foregroundScanPeriod = 1100
            }
        }
```

- Dans `onDestroy`, on arrête le scan :

```
BeaconManager.getInstanceForApplication(this).stopRangingBeacons(scanRegion)
```

- La région est la suivante :

```
Region("wildcardRegion", null, null, null)
```

- MainActivity doit implémenter l'interface `RangeNotifier` et la fonction `didRangeBeaconsInRegion(beacons: MutableCollection<Beacon>?, region: Region?)` de celle-ci :

```
override fun didRangeBeaconsInRegion(beacons: MutableCollection<Beacon>?, region:
Region?) {
        if (beacons != null) {

beaconsViewModel.setNearbyBeacons(beacons.map(PersistentBeacon.Companion::fromBeac
on))
        }
    }
```

Ceci permet de notifier la VM que des balises ont été trouvées dans une annonce.

BeaconsViewModel

- `_nearbyBeacons` n'est plus une liste mais une map pour gérer la persistence (c.f. question 1.1.1)
- Nous avons ajouté une fonction `setNearbyBeacons` qui s'occupe de gérer les balises existantes ou non :

```
fun setNearbyBeacons(newBeacons: Iterable<PersistentBeacon>) {
        val beacons = _nearbyBeacons.value!!
        newBeacons.forEach {beacon ->
            if (beacons.containsKey(beacon.minor)) {
                beacons[beacon.minor]!!.apply {
                    this.distance = beacon.distance
                    this.rssi = beacon.rssi
                    this.txPower = beacon.txPower
                    this.lastAppeared = Instant.now()
                }
            } else {
                beacons[beacon.minor] = beacon
            }
        }
        _nearbyBeacons.postValue(beacons.filterValues {

it.lastAppeared.plusSeconds(TIMEOUT_BEACONS_SECONDS).isAfter(Instant.now())
        }.toMutableMap())
    }
```

Questions

1.1.1

On remarque que, certaines fois, l'appareil ne reçoit pas les annonces. Pour pallier à ça et éviter que la liste ne se rafraichisse à chaque fois, on pourrait maintenir une liste persistente des balises qui ont été détectées. Ceci peut se faire simplement via une map dans la viewmodel. Il faut également penser à un système permettant de retirer les balises qui ne sont plus détectées après une certaine durée (par exemple, 30 secondes). Ceci peut être fait à l'aide d'un paramètre supplémentaire dans PersistentBeacon indiquant la date de dernière apparition et ensuite modifier BeaconsViewModel::nearbyBeacons pour ne retourner que les balises récentes.

1.1.2

On pourrait implémenter ceci sous la forme d'un "foreground service", liée à une notification qui indiquerait que le monitoring est en cours. Il faudra ensuite utiliser la fonction `manager.setBackgroundScanPeriod(x)` pour déterminer la fréquence des scans et `manager.enableForegroundServiceScanning(NotificationBuilder)` pour lancer le scan en arrière-plan.

A noter que cette fonctionnalité demanderait plus de privilèges: `ACCESS_BACKGROUND_LOCATION` pour Android 10+, notamment.

1.1.3

La librairie propose une estimation des distances en mètre à l'aide de la propriété `distance` de la classe `Beacon`. Selon la documentation, la précision est d'environ 0.5x à 2x la distance réelle de la balise : par exemple, 10 à 40 mètres si l'objet est à 20m environ. Cette propriété est, derrière, une implémentation d'une formule d'estimation de la distance en fonction de la puissance du signal (`rssi`) et des capacités du téléphone.

La librairie permet de lisser les erreurs et d'avoir une valeur plus précises en obtenant les moyennes d'un certain nombre de mesures.

2.1.1

On pourrait soit se baser sur la distance retournée par la balise ou alors selon la force du signal reçu (rssi) en partant du principe que plus le signal reçu est fort, plus on se trouve proche de la balise.

Nous pourrions également se munir de davantages de balises pour se localiser au moyen d'une trilatération. Depuis 2019 les BLE propose une fonctionnalité optionnelle permettant de mesurer l'angle de départ et d'arrivée des signaux, mais cette fonctionnalité n'est pas forcément présente sur tous les smartphones.

2.1.2

On peut utiliser ces balises dans un cadre similaire que du geofencing. On pourrait imaginer une routine déclenchée lorsqu'une application détecte ou perde les signaux reçu d'une balise précise (par exemple, allumer les lumière quand on arrive chez soi).

On peut également imaginer des transports en commun équipé d'une balise qui à l'ouverture d'une app donnerait l'horaire associé au véhicule dans lequel la balise se trouve. Ainsi, on aurait pas besoin de taper l'itinéraire pour connaître les horaires d'arrivée, mais juste de se tenir à proximité du véhicule ou dans celui-ci directement.

Toutefois, ces balises restent limitées notamment sur leur portée qui perd rapidement en précision. La distance retournée par les balises est très approximative, et est facilement perturbable par d'autres signaux ou pas des obstacles. Elle ne sont pas non plus très sécurisée, elles sont sujettes au spoofing et au piggybacking.