Systèmes mobiles
Laboratoire n°4 : Environnement II
Capteurs et *Bluetooth Low Energy*

02.12.2019

Ce laboratoire est la seconde partie du travail concernant l'utilisation de données environnementales, celui-ci est consacré aux capteurs disponibles sur les smartphones (accéléromètre et magnétomètre principalement) ainsi qu'à la communication *Bluetooth Low Energy*.

1. Capteurs

Nous avons vu pendant le cours que les appareils mobiles disposent d'un certain nombre de capteurs. Ceux-ci permettent d'obtenir diverses informations sur l'environnement physique entourant le smartphone, on pensera, par exemple à l'accélération, au champ magnétique, à la pression atmosphérique ou encore à la température ambiante.

1.1 Manipulation

L'objectif de cette manipulation et de créer une boussole en 3D permettant d'afficher la direction du nord magnétique tout en restant à l'horizontale (voir Fig. 1). Nous nous baserons pour cela les capteurs d'accélération (`TYPE_ACCELEROMETER`) et magnétique (`TYPE_MAGNETIC_FIELD`) et nous utiliserons ensuite les données obtenues pour générer la matrice de rotation à l'aide de la méthode helper suivante :

```
SensorManager.getRotationMatrix(float[] R, null, float[] gravity, float[] geomagnetic)
```
[1]

Pour vous aider à réaliser ceci nous vous fournissons un template d'application dans lequel la partie de visualisation en 3D est fournie, nous utilisons pour réaliser cela la librairie graphique *OpenGL* intégrée au SDK *Android*[2], vous n'avez pas besoin de comprendre le code de cette partie mais n'hésitez pas à le consulter et à nous poser des questions en cas d'intérêt. Vous utiliserez uniquement la méthode `float[] swapRotMatrix(float[] rotMatrix)` mise à disposition par la classe *OpenGLRenderer* fournie dans le projet template. Une instance de cette classe `opglr` existe dans l'activité `CompassActivity`. Cette méthode permet de passer au moteur de rendu 3D la matrice de rotation obtenue lors de cette manipulation afin que la flèche 3D soit dessinée à l'écran dans la bonne direction. Pour des raisons de performances (dans les applications 3D nous travaillons en général à une fréquence de 60 images par seconde, ce qui laisse environs 16 ms pour préparer la prochaine image) cette méthode va retourner l'ancienne matrice de rotation, l'idée est de travailler avec uniquement 2 tableaux que l'on échange continuellement sans avoir à recréer un nouveau tableau (qui devra finalement être « garbage collecté ») plusieurs dizaines de fois par secondes.

[1] https://developer.android.com/reference/android/hardware/SensorManager.html#getRotationMatrix(float[],%20float[],%20float[],%20float[])

[2] https://developer.android.com/guide/topics/graphics/opengl.html

Figure 1 - Capture d'écran de la boussole 3D

1.2 Question

Une fois la manipulation effectuée, vous constaterez que les animations de la flèche ne sont pas fluides, il va y avoir un tremblement plus ou moins important même si le téléphone ne bouge pas. Veuillez expliquer quelle est la cause la plus probable de ce tremblement et donner une manière (sans forcément l'implémenter) d'y remédier.

2. Communication *Bluetooth Low Energy*

Nous allons voir dans cette partie la communication *Bluetooth Low Energy* entre un smartphone et un périphérique externe. Pour cela nous vous mettons à disposition un écran *Espruino Pixl.js*[3] (Fig. 2) simulant un périphérique et mettant à disposition deux services *Bluetooth Low Energy*.

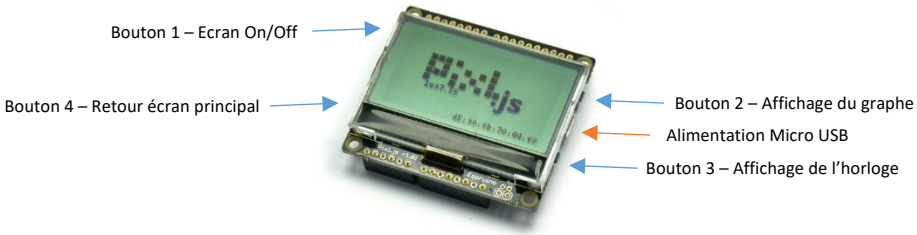

Figure 2 - Un écran Espruino Pixl.js – Configuration des 4 boutons

Les services configurés sur ce dispositif pour cette manipulation sont les suivants :

- Current Time Service **0x1805**[4]
  Ce service suit les spécifications officielles[5] du consortium *Bluetooth SIG*. Celui-ci permet de lire et de régler l'heure du périphérique. Les écrans Pixl.js n'étant pas alimentés en permanence, leur horloge interne se réinitialise lorsque que l'alimentation est coupée, ce service permet de remettre l'horloge à l'heure courante.

[3] Documentation *Espruino Pixl.js* : https://www.espruino.com/Pixl.js
[4] 0x1085 -> 0000**1805**-0000-1000-8000-00805f9b34fb – Conversion des UUID courts en UUID longs
[5] Spécification *Current Time Service* : https://www.bluetooth.com/wp-content/uploads/Sitecore-Media-Library/Gatt/Xml/Services/org.bluetooth.service.current_time.xml

Seule la caractéristique obligatoire est implémentée :

- 0x2A2B - Current Time
  Cette caractéristique permet de lire et d'écrire l'heure courante. Elle permet aussi de s'enregistrer au périphérique pour recevoir des notifications régulières contenant l'heure du périphérique. Le format de données échangé suit la spécification officielle[6], la documentation Bluetooth n'est pas toujours très simple à prendre en main, il faut suivre plusieurs liens pour obtenir toutes les informations nécessaires, pour ce laboratoire nous représentons ci-dessous le format des données à utiliser :

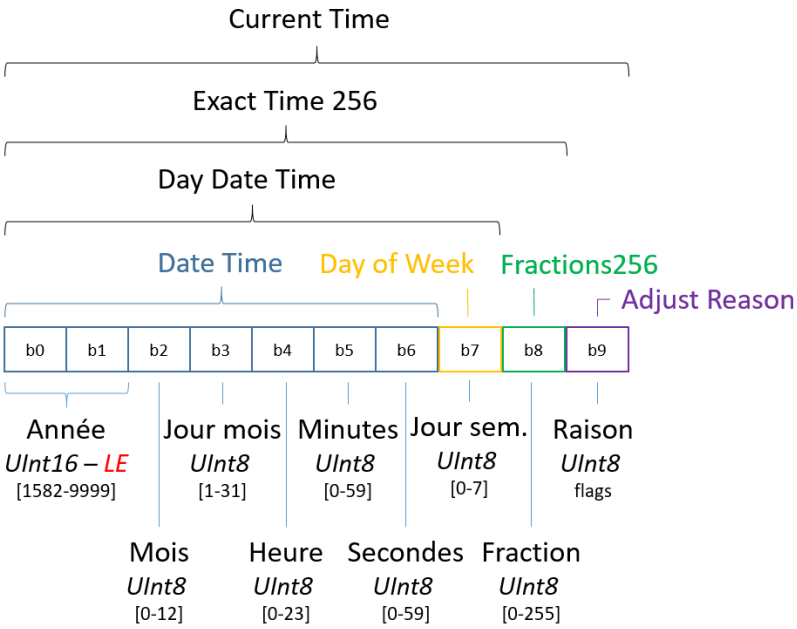

Figure 3 - Format d'échange de données "Current Time", les spécifications détaillées :
Current time : https://www.bluetooth.com/wp-content/uploads/Sitecore-Media-Library/Gatt/Xml/Characteristics/org.bluetooth.characteristic.current_time.xml
Exact Time 256 : https://www.bluetooth.com/wp-content/uploads/Sitecore-Media-Library/Gatt/Xml/Characteristics/org.bluetooth.characteristic.exact_time_256.xml
Day Date Time : https://www.bluetooth.com/wp-content/uploads/Sitecore-Media-Library/Gatt/Xml/Characteristics/org.bluetooth.characteristic.day_date_time.xml
Date Time : https://www.bluetooth.com/wp-content/uploads/Sitecore-Media-Library/Gatt/Xml/Characteristics/org.bluetooth.characteristic.date_time.xml
Day Of Week : https://www.bluetooth.com/wp-content/uploads/Sitecore-Media-Library/Gatt/Xml/Characteristics/org.bluetooth.characteristic.day_of_week.xml

- Service Custom SYM 3c0a**1000**-281d-4b48-b2a7-f15579a1c38f
  Il s'agit d'un service custom mis en place spécialement pour ce laboratoire. Celui-ci met à disposition 3 caractéristiques différentes :
  - 3c0a**1001**-281d-4b48-b2a7-f15579a1c38f - int (Uint32LE)
    Cette caractéristique permet au smartphone d'envoyer un entier codé sur 4 bytes, au format *UInt32LE*. Le périphérique stocke les 10 derniers entiers reçus et les affiche sur un graphe à l'écran.
  - 3c0a**1002**-281d-4b48-b2a7-f15579a1c38f - Temperature (Uint16LE)
    Cette caractéristique permet au smartphone de lire la température du thermomètre interne du périphérique. La température est retournée sous la forme d'un nombre entier codé sur 2 bytes, au format *UInt16LE*. Il faudra diviser ce nombre par 10 pour

[6] Spécification format de données *Current Time* : https://www.bluetooth.com/wp-content/uploads/Sitecore-Media-Library/Gatt/Xml/Characteristics/org.bluetooth.characteristic.current_time.xml

obtenir des degrés *Celsius*. Le périphérique rafraichit cette mesure toutes les 30 secondes.

- 3c0a**1003**-281d-4b48-b2a7-f15579a1c38f – BTN (Boutons)
  Le smartphone peut s'inscrire (notification) à cette caractéristique. Celle-ci retourne un compteur codé sur un byte (UInt8) qui est mis à jour à chaque fois qu'un des quatre boutons de l'écran est appuyé, celle valeur est envoyée au smartphone s'il s'est inscrit.

2.1 Manipulation

Pour cette seconde partie aussi, une partie du code vous est fournie, en particulier la fonctionnalité de scan et de recherche de périphériques BLE à portées ainsi que la structure de base pour l'intégration de la librairie *Android-BLE-Library*[7] utilisée pour simplifier la communication asynchrone avec le périphérique BLE.

Suivant le cas d'utilisation souhaité, la durée de la connexion avec le périphérique BLE va être plus longue que la période d'utilisation de l'application, nous devons donc bien réfléchir à quel niveau la connexion va être gérée : au niveau de l'activité, d'un service ou autre ? Dans le cadre de ce laboratoire, nous avons fait le choix de garder la connexion ouverte tant que l'activité est visible, on souhaite toutefois que celle-ci puisse survivre à un changement de contexte (un changement d'orientation de l'écran par exemple). C'est pourquoi nous utilisons les nouveaux outils introduits dans *Android Jetpack*[8] : *ViewModel* et *LiveData*. Le premier permet d'associer une instance de classe survivant à la suppression et la récréation d'une activité, le second propose un mécanisme d'observation sur des variables.

Dans le template fourni, l'activité se connecte au *ViewModel*, s'il n'existe pas une nouvelle instance est créé sinon l'existante est réutilisée, ensuite l'activité observe la *LiveData isConnected* cette variable booléenne indique si le *BleManager* contenu dans la *BleOperationsViewModel* est connecté (ou pas) à un périphérique. L'activité va rafraichir sa vue lors d'un changement de cet état. Vous pouvez ajouter toutes les *LiveData* nécessaires pour permettre la mise à jour de la vue de l'activité lorsque qu'une donnée est reçue depuis le périphérique (la température, l'heure ou le nombre de clics). Vous devrez aussi ajouter les méthodes nécessaires permettant de transmettre les entrées utilisateurs effectuées sur l'activité jusqu'au *BleManager* (voir par exemple la méthode `readTemperature()`).

Votre première tâche sera d'ajouter un filtre sur le scanner BLE afin de n'afficher que les périphériques disposant du service (3c0a1000-281d-4b48-b2a7-f15579a1c38f - *Service Custom SYM*).

Votre travail sera ensuite de créer la vue « opération » sur l'activité, depuis celle-ci l'utilisateur devra pouvoir visualiser les données reçues (lecture, notification) depuis le périphérique et déclencher l'envoi de données ou d'évènements vers celui-ci (lecture, écriture de données). Il faut que toutes les fonctionnalités mises à disposition dans les services *Current Time Service* et *Service Custom SYM* soient utilisées, votre solution devra permettre de :

- Afficher la température du périphérique (lecture) ;
- Afficher le nombre de bouton cliqués (notification) ;
- Envoi d'un nombre entier au périphérique (écriture) ;
- Afficher l'heure du périphérique (notification) ;
- Mettre à jour l'heure sur le périphérique (écriture).

[7] Android-BLE-Library : https://github.com/NordicSemiconductor/Android-BLE-Library
[8] Android Jetpack : https://developer.android.com/jetpack

2.2 Questions

- La caractéristique permettant de lire la température retourne la valeur en degrés Celsius, multipliée par 10, sous la forme d'un entier non-signé de 16 bits. Quel est l'intérêt de procéder de la sorte ? Pourquoi ne pas échanger un nombre à virgule flottante de type *float* par exemple ?
- Le niveau de charge de la pile est à présent indiqué uniquement sur l'écran du périphérique, mais nous souhaiterions que celui-ci puisse informer le smartphone sur son niveau de charge restante. Veuillez spécifier la(les) caractéristique(s) qui composerai(en)t un tel service, mis à disposition par le périphérique et permettant de communiquer le niveau de batterie restant via *Bluetooth Low Energy*. Pour chaque caractéristique, vous indiquerez les opérations supportées (lecture, écriture, notification, indication, etc.) ainsi que les données échangées et leur format.

3. Durée

- 8 périodes
- A rendre le dimanche **12.01.2020** à **23h55** au plus tard.

4. Rendu/Evaluation

Pour rendre votre code, nous vous demandons de bien vouloir zipper votre projet *Android Studio*, vous veillerez à bien supprimer les dossiers build (à la racine et dans app/) pour limiter la taille du rendu. En plus, vous remettrez un document **pdf** comportant au minimum les réponses aux questions posées.

Merci de rendre votre travail sur *CyberLearn* dans un zip unique. N'oubliez pas d'indiquer vos noms dans le code, sur vos réponses et de commenter vos solutions.

Bonne chance !