alexandre.lombard@utbm.fr

CONSIGNES

Les robots EV3 utilisés dans ce TP sont fournis déjà montés, il est déconseillé de les démonter ou de les modifier.

Votre code devra être clair, organisé et documenté. Vous respecterez les conventions de programmation de Java et commenterez chacune de vos fonctions en utilisant le standard Javadoc.

OBJECTIFS DU TP

L'objectif du TP est de mettre en œuvre la synchronisation d'horloge entre plusieurs robots capables de communiquer en réseau.

INSTALLATION DU PROJET DE DEPART

Dans un premier temps, récupérez les sources disponibles à l'adresse suivante via Git :

https://github.com/alexandrelombard/LejosMusic

Ce projet contient un ensemble de classes permettant à un robot de jouer une partition musicale (les ressources du projet contiennent aussi des exemples de partitions). Afin, de faciliter votre travail, des classes permettant la communication réseau en UDP sont également disponibles.

La classe **Launcher** contient la fonction **main()**. Celle-ci charge des partitions depuis les fichiers ressources et joue une partition prédéfinie en fonction de la touche enfoncée au démarrage du programme.

CONFIGURATION DE LA CONNEXION

Les robots EV3 fournis sont équipés d'une clé Wifi qui sera utilisée pour la communication réseau entre les robots. Ces derniers ne peuvent fonctionner en mode ad-hoc ou en mode point d'accès, vous devrez donc connecter vos deux robots à un même point d'accès Wifi (par exemple un téléphone Android ou un ordinateur portable) afin qu'ils puissent communiquer entre eux. Ce point d'accès devra utiliser un cryptage WPA-PSK.

SYNCHRONISATION

Chaque robot ne peut jouer qu'une partition simultanément (une partition correspondant à un seul instrument). Afin de jouer le morceau dans son intégralité, c'est-à-dire tous les instruments en même temps, il vous faudra utiliser plusieurs robots.

Les robots devront nécessairement être synchronisés, c'est-à-dire à la même position dans le morceau (si un robot est en retard par rapport à un autre, la musique ne sera pas correctement jouée).

Afin de savoir à quelle position un robot en est dans l'exécution de sa partition, vous pourrez utiliser la méthode **getTime()** de la classe **Track**. Afin de redéfinir la position du robot dans l'exécution de sa partition, vous pourrez utiliser la méthode **setTime()** de la classe **Track**.

Du fait de l'imprécision des robots, le temps mis pour jouer une note est variable. Afin d'éviter que les robots ne divergent dans l'exécution de la partition, la synchronisation devra être effectuée en continu.

SYNCHRONISATION CENTRALISEE

Dans un premier temps, vous implémenterez une politique de synchronisation centralisée. Un robot aura le rôle de coordinateur (ou chef d'orchestre), et imposera son temps aux autres robots.

Un robot « suiveur » n'ajustera son temps qu'à condition que la différence entre son temps propre et le temps du chef d'orchestre soit supérieur à un seuil prédéfini ΔT.

SYNCHRONISATION DECENTRALISEE

Dans un second temps, vous implémenterez une politique de synchronisation décentralisée. Chaque robot devra transmettre aux autres robots son temps, et devra ajuster son propre temps en utilisant la moyenne des temps reçu en provenance des autres robots pendant une période de temps T.

AJUSTEMENT DES PARAMETRES (BONUS)

Mesurer la dérive maximale entre deux robots. Utilisez la valeur obtenue pour définir théoriquement ΔT et T.