HEIG-VD HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Yann Thoma
Florian Vaussard

Laboratoire de Programmation Concurrente

semestre automne 2023

Gestion de ressources

Temps à disposition : 6 périodes (travail débutant en semaine 6)

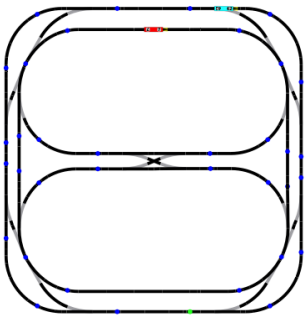
1 Objectifs pédagogiques

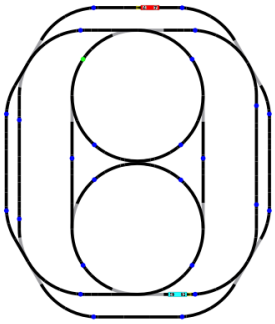
— Réaliser un programme concurrent comportant de la synchronisation et de la gestion de ressources avec des **sémaphores**.

2 Cahier des charges

Grâce au simulateur qui vous est fourni, implémentez un programme en C++ qui réalise la gestion et le contrôle de deux locomotives. Chaque locomotive est exécutée par un thread et part depuis un point prédéterminé. Vous êtes libres de choisir le tracé des locomotives, tant qu'elles suivent un parcours cyclique pour revenir à leur point de départ. De plus, les deux tracés devront comporter au moins un tronçon commun. Ce tronçon commun ne peut être emprunté que par une seule locomotive à la fois. Si une locomotive est déjà présente, la deuxième doit s'arrêter et attendre que le tronçon soit libre. Attention, une locomotive ne doit pas s'arrêter pour redémarrer immédiatement, afin d'éviter de donner des hauts le cœur aux voyageurs.

Vous pouvez choisir entre deux maquettes différentes (nommées A et B) au moyen de la fonction `selection_maquette()` dans le fichier `cppmain.c`. Choisissez une maquette et déterminez le parcours que chaque locomotive devra emprunter.





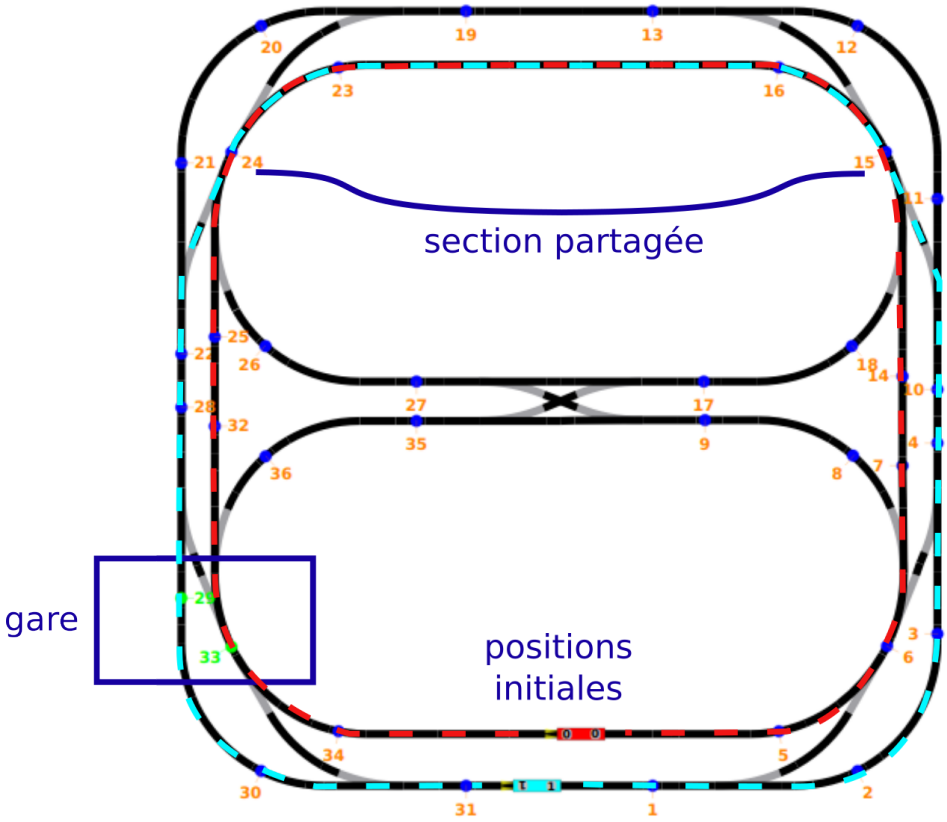
Les points de départ des locomotives devront être proches et sur des tronçons parallèles afin de simuler une gare. Les deux locomotives doivent assurer une correspondance et permettre aux voyageurs de changer de train. Ainsi, quand la première locomotive arrive à la gare, elle doit s'arrêter et attendre le deuxième train. Dès que celui-ci s'arrête en gare, les deux trains devront repartir après 5 secondes d'attente pour continuer leurs parcours.

Étant donné que les deux locomotives vont quitter la gare plus au moins au même moment, l'accès au tronçon commun est défini de la manière suivante : La deuxième locomotive arrivée en gare aura la priorité pour accéder au tronçon commun. Cela signifie que si l'autre locomotive arrive avant, elle devra céder sa place, et donc attendre que la prioritaire quitte le tronçon pour pouvoir s'y aventurer.

Ceci fonctionne bien si les locomotives sont déjà passées par la gare. Pour le départ, veillez donc à placer les locomotives un peu avant le point correspondant à l'arrivée en gare.

Les parcours se réalisent indéfiniment. Le seul moyen d'arrêter les locomotives est d'actionner un *arrêt d'urgence*. Aussitôt actionné, les deux locomotives devront immédiatement s'arrêter.

A titre d'exemple, voici un parcours possible sur le circuit A :



3 Interface

Le code qui vous est fourni est décomposé de manière à avoir une classe `Synchro` qui fournit la synchronisation. Celle-ci dérive de l'interface suivante :

```
class SynchroInterface
{
public:

// Called when a locomotive wants to access the shared section.
// Should stop the locomotive if required
virtual void access(Locomotive& loco) = 0;

// Called when a locomotive leaves the shared section
virtual void leave(Locomotive& loco) = 0;

// Called when the locomotive stops at the railway station.
// The first call shall block until the second call is done.
virtual void stopAtStation(Locomotive& loco) = 0;
};
```

Il est impératif de respecter cette interface, car les tests du code l'utiliseront tel quel.

4 Simulateur

Le code ainsi que la documentation du simulateur de maquettes sont fournis. Voici quelques indications indispensables sur le simulateur :

- Lors de votre première compilation, il faut aller dans le dossier de build et lancer la commande `make install` avant de pouvoir lancer l'application. Ceci peut aussi être fait depuis QtCreator (cf. documentation de QTrainSim).
- Il est possible de modifier certains paramètres dans le fichier `cppmain.cpp`, tels que :
  - La position et la vitesse initiales de chaque locomotive
  - La maquette utilisée (A ou B)
  - La position de départ des aiguillages
- Le contrôle du trajet emprunté par chaque locomotive s'effectue au moyen de points de contacts et d'aiguillages. On peut afficher ou masquer le numéro des points de contact et des aiguillages depuis le menu *View* de l'interface du simulateur.
- Il est possible de mettre en "pause" chaque locomotive depuis l'interface du simulateur, ce qui permet de tester certains scénarios plus facilement. De même, il est possible de changer un aiguillage en cliquant dessus.

5 Remarques

Quelques dernières remarques :

- Afin de faire rentrer ou sortir les locomotives du tronçon commun (section partagée) il vous faudra commander les aiguillages de manière appropriée.
- Ne pas modifier les fichiers `locomotive.h/cpp`, `launchable.h` et `synchrointerface.h`. Vous pouvez créer de nouveaux fichiers ou classes pour gérer vos parcours au besoin.
- L'arrêt d'urgence est une nécessité pour tout système critique où un arrêt peut être réalisé. Dans le présent laboratoire, une manière simple et triviale de réaliser cet arrêt consiste à faire un appel à la procédure `mettre_maquette_hors_service`. Ceci équivaut à couper toute l'alimentation du réseau ferroviaire. Dans le contexte de ce laboratoire, il est demandé d'arrêter les deux locomotives sans utiliser cette fonction.
- La description de l'implémentation, ses différentes étapes, la manière dont vous avez vérifié son fonctionnement et toute autre information pertinente doivent figurer dans un petit rapport rendu avec le code. Celui-ci se nommera `rapport.pdf` et se trouvera au même niveau que le script `pco_rendu.sh`, ce dernier servant à générer l'archive à rendre sur Cyberlearn.
- Inspirez-vous du barème de correction pour savoir là où il faut mettre votre effort.
- Vous pouvez travailler en équipes de deux personnes.

6 Barème de correction

Conception et conformité à l'énoncé	35%
Exécution et fonctionnement	15%
Codage	15%
Documentation (rapport) et en-têtes des fonctions	25%
Commentaires au niveau du code	10%