Laboratoire 4

Gestion de ressources partagées

Source : [Top 10 des trains panoramiques en Suisse | Trainline]

Introduction au problème

Nous sommes en train d'essayer de représenter et de gérer un circuit de rail avec des locomotives le parcourant. Leurs trajets se recouvreront en partie sur une section de leur trajectoire. De la sorte, il faut gérer le fait qu'une seule locomotive peut s'y engager puis, dans un second temps, que seulement la locomotive la plus prioritaire en attente peut y accéder.

Choix d'implémentation

Nous avons fait beaucoup de choix que je vais documenter ici. Pour le reste, c'est dans le code.

Pour pouvoir suivre le trajet d'une locomotive et définir les points de contact qui nous intéressent, nous avons décidé de stocker une liste des points de contact qui définissent le trajet. Pour que notre programme fonctionne, il est critique que cela soit la liste entière des contacts par lesquels la locomotive va passer dans ses tours.

Nous avons choisi de stocker dans le comportement de la locomotive l'entrée et la sortie de la zone partagée. Cependant, étant donné qu'on doit pouvoir gérer plusieurs trajectoires différentes avec une même zone partagée, nous avons choisi de faire en sorte que l'entrée et la sorte de la zone partagée soient et reste les mêmes, peu importe le sens dans lequel on va, ou la locomotive qu'on regarde.

De la sorte, on a besoin de gérer les différentes manières de rédiger notre section partagée. Nous avons identifié 4 grands cas :

Si on définit un exemple de la section critique avec des lettres : E->F->G->H

On définirait donc l'entrée à E et la sortie à H, et ce pour tous nos trains.

On peut définir 4 trains avec des trajets différentes qui traversent la même section partagée :

Train 1: A->B->C->D->**E->F->F->H** (Ici on l'écrit de gauche à droite, et il est en un seul bloc)

Train 2: K->J->I->**H->G->F->E** (Ici on l'écrit de droite à gauche, et il est en un seul bloc)

Train 3: **G->H**->V->W->X->Y->**E->F** (Ici on l'écrit de gauche à droite, mais il est en deux parties)

Train 4: **F->E**->P->O->N->M->**H->G** (Ici on l'écrit de droite à gauche, mais il est en deux parties)

Dans beaucoup de cas, on devra traiter ces cas séparément, c'est pour ça qu'on les aura testés séparément aussi (voir prochaine section). Dans le code, on doit aussi souvent gérer selon si la locomotive est en train de se déplacer de gauche à droite ou dans droite à gauche parmi notre liste de contacts (donc dans notre trajectoire, telle que définie dans le constructeur initial).

Nous avons aussi fait en sorte d'attendre le passage par l'entrée et la sortie dans le run des locomotives, malgré que ça ne soit pas nécessaire étant donné les points de contact défini avec le buffer, pour pouvoir afficher des messages et logger notre avancée.

On mémorise aussi les indexes d'entrée et de sortie de la zone partagée, car nous en avons besoin pour le calcule du buffer, et qu'on voudrait ne pas avoir à le calculer à chaque fois.

Nous avons défini des zones buffer autour de notre section partagée. Ces zones servent à définir les points de contact pour les requêtes, les obtentions d'accès, et la libération des accès dans notre code. Cependant, nous nous sommes rendus compte que nous pouvions avoir beaucoup de problèmes si la station ou la position initiale de la locomotive était dans cette zone de buffer. Nous pourrions avoir des interblocages, ou un manque d'exclusions mutuel, ce genre de choses.

De la sorte, nous avons choisi d'interdire le fait de poser la locomotive ou la station dans ces zones de buffer. Nous avons conscience qu'il y a des moyens de gérer ces cas, mais nous trouvions déjà notre code assez complexe.

Par exemple nous pourrions détecter si notre locomotive commence dans la zone de buffer, auquel cas on pourrait lui accorder le sémaphore, mais nous devrions alors nous assurer qu'une et une seule locomotive y était, rajoutant encore beaucoup de logique à notre code, et beaucoup de mots de têtes pour tout le monde.

Je noterai ici qu'étant donné qu'il n'y a pas de fin prévue à la simulation, nous arrêtons le programme via un contrôle-C ou via la fermeture de la fenêtre. Selon ma compréhension de la donnée, vous ne vous attendez pas à une gestion plus maîtrisée de la fermeture du programme, vu que vous nous dites de tourner à l'infini. De la sorte, nous ne nous sommes pas soucié de ceci.

Pour la liste des priorités pour l'exercice 2, nous avons choisi d'utiliser un deque et de gérer la recherche avec des stable_sorts. Cela nous permet de gérer la priorité en supprimant le premier élément à chaque fois (le plus prioritaire du coup vu qu'on a sort avant) sans avoir les problèmes de performance d'un vector. On a étudié la possibilité de le faire avec un vector mais en sortant de sorte à avoir l'élément le plus prioritaire à la fin, mais on rencontre des problèmes avec le fait de rester stables, dans ce cas-là.

Notre fonction run fonctionne en fonction de si on va vers la station ou si on va vers la section partagée (ces cas devrait complétement s'exclure mutuellement étant donné qu'on interdit la station dans le buffer). De là on effectue notre logique en boucle, en échangeant à chaque fois ce vers quoi on se dirige (cela veut dire que notre programme n'est prévu que pour avoir une seule station, et une seule section critique).

On ne gère pas le cas où il n'y a pas de section critique, ni s'il n'y a pas de station.

Concernant les fichiers qu'on n'était pas supposés changer. Nous avons relevé des incohérences que nous avons dû traiter, et de la sorte, nous avons choisi de modifier certaines fonctions de sorte à rendre cela cohérent avec ce qui nous était demandé, voir même cohérent avec ce qui était écrit directement au-dessus des déclarations des fonctions.

Concernant les aiguillages, nous avons identifié ceux qui étaient nécessaires pour chaque train, et avons initialisé ceux-ci comme il était nécessaire pour les trains dans le cppmain.cpp. Nous avons aussi stockés ceux qui étaient communs afin de pouvoir faire la modification à chaque fois.

Nous avons simplement stockés ceux-ci dans des paires de int, l'un représentant le numéro de l'aiguillage, l'autre l'orientation.

Nous n'avons pas pris en compte l'inertie, qui est une option sélectionnable dans l'interface graphique. Étant donné que la donnée n'en parle pas, nous avons choisi de définir qu'on créait notre simulation sans l'inertie. Chaque locomotive accélère et décélère en une intervalle de 0 secondes.

Tests effectués

Pour ce laboratoire, nous avons défini des tests dans cppmain.cpp de prog1 et prog2 pour pouvoir tester nos 4 types d'initialisation particulières (écrit en avant/arrière, coupé/en un seul morceau). Nous nous attendons à ce que, peu importe les tests parmi les 4, nous arrivions au même résultat étant donné que les autres variables restent les mêmes (on teste que peu importe la manière dont on nous fournit les données de la trajectoire, on obtient bien le même résultat).

Avec la dernière version, nous n'avons pas observé d'anomalie à ce niveau.

De là, nous avons effectué des variations de position de départ, de vitesse de départ, de position de la station (gare), de priorité dans le cas de la partie 2 (nous avons manuellement choisi la priorité afin de s'assurer que cela fonctionnait bien) (changé dans le fichier de locomotivebehavior.cpp, où on a aussi permis d'afficher l'état initial pour plus de visibilité).

Nous avons vérifié que les locomotives fonctionnaient si elles n'entraient pas par le même côté dans la section partagée, par le même côté dans un sens, dans l'autre, etc.

Normalement, nous nous attendons à ce que notre programme fonctionne aussi avec plus de deux locomotives. Cependant, nous n'avons pas pu trop tester cela étant donné la structure fournie pour les rails. Mais c'est prévu pour fonctionner à plus grande échelle. Mais à tester sur d'autres circuits.

Nous avons aussi tenter de voir ce qui était faisable avec des vitesses négatives. Cependant, le système n'a pas du tout l'air prévu pour, étant donné les animations qui se passent, donc nous avons abandonné l'idée et assumé une vitesse positive.

Merci de votre lecture !