1 Introduction

Ce rapport est le rendu du laboratoire n°3 du cours de PCO (Programmation Concurrente) de la HEIG-VD.

Il a pour but principal de résumer nos choix d'implémentation pour un programme de simulation de ventes entre des mines, grossistes et des usines. Les détails ci-dessous :

- Des mines qui produisent des matières premières et les revendent.
- Des grossistes qui achètent soit des matières premières au mines, soit des objets manufacturés aux usines.
- Des usines qui achètent des matières premières aux grossistes et qui produisent des objets.

2 Mines

2.1 Vente d'objets

Les fonctions appelées *buy(ItemType it, int qty)* sont similaires dans tous les autres entité (grossiste et usine) et ne seront plus discutées dans les autres classes. Elle fait globalement 3 actions :

- Vérification si l'objet existe dans nos stocks et si la quantité demandée d'objets est en stock.
- Ajout de l'argent versé dans les fonds de la mine (ou autre entité).
- Retrait du nombre d'objets demandés dans le stock.

Ces actions seront protégées de la concurrence par la suite étant donné l'accès aux variables globales.

3 Usines

3.1 Construction d'objet

Chaque usine peut construire un objet avec différents ingrédients récupérés des grossistes qui les ont eux-mêmes acheté aux mines. Elle fait 4 actions globales :

- Paiement du salaire du constructeur. Engendre la construction si les fonds sont suffisants.
- Retrait du nombre d'ingrédients à utiliser dans la réserve de l'usine.
- Incrément du nombre d'objet créé total de l'usine.
- Ajout du nouvel objet créé dans les stocks.

À nouveau, la concurrence sera gérée par la suite.

3.2 Commande des ressources

Une usine doit pouvoir commander les ressources qui lui sont nécessaires pour la construction de ses objets. Elle passe cette commande chez les grossistes. Concernant la fonction *Factory::order_ressources()*, elle fonctionne comme suit :

- Récupération des ressources à commander par ordre de disponibilité (à l'aide d'une nouvelle méthode *sort_ressources_by_stock()*).
- On vérifie si l'on possède assez d'argent pour l'achat et on demande au grossiste de nous vendre si possible les ressources dont on a besoin.
- Si oui, achète 1x l'ingrédient en déduisant le prix des fonds de l'usine et en rajoutant l'ingrédient acheté dans les stocks de l'usine.

4 Grossistes

4.1 Achat des ressources

Chaque grossiste peut acheter que ça soit aux mines ou aux usines des ressources. Ces achats sont choisis aléatoirement parmi la liste des entités disponibles chez le revendeur qui est lui aussi tiré aléatoirement.

Cette partie était déjà partiellement implémentée. Nous avons donc dû implémenter les actions suivantes :

- Vérifier si l'usine possède les fonds nécessaires et que le revendeur possède la quantité voulue.
- Si c'est le cas, on soustrait l'argent que doit le grossiste au vendeur et on ajoute la quantité qui lui est due.

5 Concurrence

5.1 Accès aux ressources partagées

Par ressources partagées, on parle principalement de l'argent et des stocks qu'ont les instances de chaque entité. Ces ressources sont critiques au bon fonctionnement du programme. Si la concurrence est mal gérée, alors ces ressources seront incorrectes en fin de programme et les calculs des fonds finaux seront faux.

Pour résoudre ce problème, nous avons donc décidé d'implémenter un mutex par instance. Celui-ci sera chargé de se verrouiller lorsqu'une écriture ou lecture sera faite sur une des ressources critiques citées précédemment. L'enjeu était de réussir à bloquer le nombre minimum d'instructions pour s'assurer de garder une fluidité dans le programme.

5.2 Terminaison du programme

Pour terminer le programme, l'utilisateur doit appuyer sur la touche «Enter». À la suite de cela, on effectue une requête d'arrêt sur chaque thread (*thread->requestStop()*).

Dans les fonctions d'exécution *run()* de chaque instance, on vérifie en permanence si la demande d'arrêt a été effectuée sur les threads (*PcoThread::thisThread()->stopRequested()*).

Si c'est le cas, alors les instances arrêtent de réaliser leurs actions et on entame la procédure de calcul des revenus et des dépenses pour avoir l'affichage final avec les fonds attendus et les fonds réellement obtenus.

Cette dernière étape peut prendre quelques instants car si un thread est en train de créer un objet, cela peut du temps avant qu'il ne teste si le thread doit être arrêté.

6 Tests effectués

Nous n'avons pas utilisé de libraire de tests unitaires tel que GoogleTest. Cependant, nous avons essayé de faire varier le nombre d'instances de chaque type de revendeur, ainsi que les fonds dans le fichier *utils.h*.

Le but de chaque test était de vérifier que les fonds obtenus étaient similaires aux fonds attendus par le programme qui nous était fourni. Pour ce faire, on exécutait le programme pendant plusieurs minutes (1 à 2 minutes) et on observait les résultats obtenus afin que ceux-ci soient conformes.