Rapport Projet C - Semestre 5

DUCOS Joris, DOULIERY Baudouin
Bordeaux INP ENSEIRB

13 Décembre 2019

Atomic Teddy Investors

Contents

1 Introduction

Ce rapport a pour objectif de présenter et d'expliquer le travail réalisé sur le premier projet de programmation, en première année à l'ENSEIRB-MATMECA. Le projet porte sur l'implémentation d'un jeu, Atomic Teddy Investors, en langage C. On réalise dans un premier temps une version de base, puis un ensemble d'achievement.

Le but principal du projet est d'implémenter le jeu, en langage C. Le problème principal est donc de définir un ordre de jeu puis de faire réaliser à chaque joueur un nombre de transaction. On peut ainsi se demander comment les différents élements du jeu vont être modélisés ? Comment la boucle de jeu va fonctionner ? Comment les règles du jeu vont être respectées ? Tout ces problèmes vont être divisés en sous-problèmes, afin de faciliter le travail.

2 Résumé

Le projet porte sur le jeu Atomic Teddy Investors.

Extrait de *Atomic Teddy Investors :*
"La Sicile a été finalement envahie, et les ours ont vaincu l'armée du Grand-Duc. Se mêlant aux humains, ils se mêlent aux divers aspects de la société, reprenant à leur compte les divers métiers propres à la race humaine. L'un des aspects qui les fascine est celui de l'économie de marché. Sur les places de marché de l'île, ils se mettent à jouer les magnats, les barons, les grands seigneurs, afin d'arrondir leur pécule personnel ventripotent. Dans leurs costumes impeccables, ils prospectent les villes à la recherche d'opportunités d'achat et de revente, dans un but finalement assez simple : du miel, rien que du miel. Leur appât du gain est certainement plus gourmand que cupide, et on admettra que leur compréhension de la mécanique économique reste un peu hasardeuse, ce qui donne l'objet de ce sujet, qui décrit ces ensembles d'échanges à la manière d'un jeu."

Ce rapport va reprendre la méthode, et les outils qui ont été utilisés au cours de ce projet. Il explique alors la démarche employée, et les choix qui ont été fait.

3 Organisation

3.1 Répartition des tâches

Pour traiter au mieux le projet, il a été divisé en plusieurs sous-problèmes. Chaque membre travaille ainsi sur des sous-problèmes différents, afin d'avancer le plus vite possible. La répartition des tâches c'est faite de manière assez intuitive, sans trop de problème pour organiser le travail au sein le binôme. Afin de pouvoir travailler en simultané, et pour s'échanger des fichiers facilement, un système de dépôt a été mis en place.

3.2 Outils utilisés

3.2.1 Dépôt (forge)

Le partage des fichiers c'est fait en utilisant le sytème de dêpot Thor, mis en place par nos professeurs encadrants. L'utilisation de git a permis d'envoyer les fichiers sur le dêpot Thor, et ainsi d'échanger les informations entre coéquipiers. Le dêpot permet aussi de faire passer des test à nos fichiers, et ainsi de verifier que tout fonctionne correctement.

3.2.2 LaTeX

Ce rapport est rédigé en LaTeX. Les fichiers relatifs à son écriture sont présents sur le dépôt.

3.2.3 Makefile

Dans le but de faciliter et de controler l'étape de compilation, nous utilisons un ficher Makefile, qui suit le principe de fonctionnement ci-dessous :

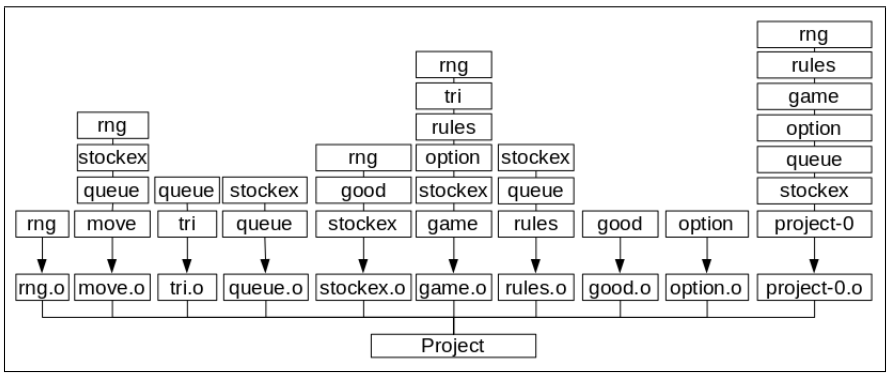

Figure 1: Makefile et Dépendance

4 Mise en place de l'algorithme

4.1 Principe de la boucle de jeu

À partir des consignes, et des explications de l'énoncé, nous avons pu mettre en place un algorithme pour la boucle de jeu. Le principe est de faire tourner la partie durant un temps prédéfini, tout en assurant le respect des règles. Au lancement de l'executable, il est possible de choisir certaines conditions, comme le nombre de joueurs, ou bien le nombre de tours.

Pendant un tour de la partie, la première étape est de selectionner le joueur prioritaire, c'est à dire le joueur avec le temps de jeu le plus faible. Donc une file de priorité doit être mise en place. Ensuite il faut le faire choisir une transaction parmis celles proposées par la place d'échange (au début du projet, il n'y avait qu'une seule place d'échange) de manière aléatoire, puis ensuite il va réaliser cette transaction un nombre de fois aléatoire entre **0 et 10**. Une fonction qui permet d'avoir un nombre aléatoire entre ces bornes est alors implémentée. Une fois la transaction réalisée, le joueur est sorti de la file, et replacé à la fin. Ce joueur ne jouera pas au tour suivant.

Au cours du projet, plusieurs places d'échanges ont été implantées, ce qui apporte en plus la notion de décision pour les teddies. Elles sont accessible ou non en fonction de la transaction choisie. Les joueurs doivent visiter toutes les places d'échanges, donc la code doit tenir compte de celles visitées avant.

À chaque tour, les joueurs doivent respecter les règles, sinon ils sont exclus. Autrement dit, si le teddy "choisit" un nombre de transactions supèrieur à ce que permet son portefeuille, la fonction *kick or keep* se charge de l'écarter du jeu. Après le dernier tour du jeu, les résultats des joueurs restants sont récupérés, les joueurs sont classés, et le classement est affiché.

4.2 Création des structures et variables globales

4.2.1 Les constantes

- Le premier type de constantes mis en place sont les constantes qui vont être utilisées au cours du jeu, comme par exemple les differentes ressources, qui sont définies dans le fichier **good.h**, ou bien comme les différentes places d'échanges, définies dans **stockex.h**.
- Les autres constantes définies sont les constantes relatives aux règles et limites du jeu, comme le nombre maximal de tours, de joueurs, ou bien de ressources. Ces constantes pourront être entrées comme argument lors de l'appel de l'executable, et ainsi les parties seront personnalisables.

4.2.2 Les structures

- **Wallet**

  La structure "Wallet" est la structure qui correspond au portefeuille d'un joueur. Elle est composée d'un tableau d'entiers, qui correspondent aux quantités de chaque ressources que le joueur possède.

```
1 // A wallet containing different amounts of
      goods
2 struct wallet
3 {
4   unsigned int data[MAX_GOOD];
5 };
```

- **Stockex**

  La structure "Stockex" est la structure qui correspond à une place d'échange. Elle est composée d'une chaine de caractères, qui donne le nom de la place d'échange, d'un entier indiquant le nombre de transactions réalisables dans cette place d' échange, et enfin d'un tableau de transactions. Elle est construite de la même manière que la structure Wallet.

- **Transac**

  La structure "Transac" est la structure qui correspond à une transaction. Elle est composée de deux portefeuilles, qui correspondent respectivement aux ressources vendues par la place d'échange, et à celles achetées par cette même place d'échange. Au cours du projet, l'identifiant du stockex auquel appartient la transaction est rajout à la structure, ainsi que l'identifiant du stockex auquel elle mène. Elle est construite de la même manière que la structure Wallet.

- **Queue**

  La structure "Queue" est la structure qui correspond à la file de joueurs. Elle est composée d'une liste de pointeurs vers des joueurs, ainsi que du nombre de joueurs présents dans la file. Elle est construite de la même manière que la structure Wallet.

- **Teddy**

  La structure "Teddy" est la structure qui correspond à un joueur. Elle est composée d'un portefeuille, d'un numéro d'identifiant, d'un temps de jeu, d'un nombre de ressource équivalent en "Honey". Durant le développement de l'achievement 1, nous avons ajouté le nombre de places d'échanges visitées, la liste avec ces places d'échanges, et enfin la localisation actuelle du teddy. Elle est construite de la même manière que la structure Wallet.

4.3 Implémentation d'une file de priorité

Après la création des structures et des variables globales, il faut mettre en place la file de priorité, dans les fichiers **queue.[ch]**. Les teddies intéragissent avec les places d'échanges en suivant un ordre de priorité. La règle dit que le teddy prioritaire est celui avec le temps de jeu le plus faible.
Donc la structure correspondante à un teddy comprend un portefeuille, un numéro d'identification, et un temps de jeu.

La solution choisit pour modéliser la file est une liste de pointeur vers des teddies, ainsi qu'un entier indiquant le nombre de teddies dans la file. Les fonctions codées permettent de créer une file, ou bien d'intéragir avec la file, en faisant entrer un teddy, ou en faisant sortir le teddy prioritaire, et ainsi pouvoir le faire jouer. C'est aussi dans ce fichier qu'est définie le nombre maximal de joueurs.

Les premiers problèmes rencontrés étaient liés à l'initialisation des teddies. Finalement, le code initialise les teddies un par un, et crée un tableau de pointeur vers ces derniers. Ensuite, l'autre problème concerne la file et le classement des teddies dedans. La solution adoptée est de ne sélectionner que le teddy prioritaire, et de ne pas appliquer de classement sur le reste de la file. Le défaut de cette méthode est que la recherche du teddy prioritaire doit se faire à chaque tour.

4.4 Implémentation des fichiers

- **good.[ch] :** Les premiers fichiers mis en place sont les fichiers good.[ch]. Les fichiers good vont définir les differentes ressources que peut acheter ou vendre un teddy, mais aussi la structure du portefeuille, le "wallet". À chaque ressource est associée une valeur. C'est dans ce fichier qu'est définie la valeur maximale de ressources differentes. Aucun problème n'a été rencontré pour coder good.[ch], mais il se peut que plus tard dans le projet il y ait des conflits, à cause de certaines modifications de types.

- **stockex.[ch] :** Les fichiers suivants ont été les fichiers **stockex.[ch]**, qui implémentent la notion de place d'échange, appellées ici "stockex".
  Une place d'échange va être caratérisée par une structure, contenant un nom,un nombre de transactions, et les transactions. Chaque transaction est définie par une portefeuille entrant, et un protefeuille sortant, aussi modélisée par une structure. C'est dans ce fichier qu'est définie la valeur maximale de transactions differentes par place d'échange.
  Lors de la mise en place des places d'échanges, les problèmes rencontrés sont principalement dus à l'écriture des pointeurs, mais ils ont vite été surmontés.

- **queue.[ch] :** La file de priorité est gérée dans les fichiers **queue.[ch]**.
  C'est à ce moment que la structure de la file est créée. Un ensemble de fonctions sont définies, afin de faire fonctionner la file, donc il y a une fonction qui définit le joueur prioritaire, une autre qui le selectionne, et le fait sortir de la file. Ces fichiers sont liés avec **stockex.[ch]**, car ils utilisent les caratéristiques des teddies pour faire évoluer la file.

- **game.[ch] :** Ces fichiers vont rassembler l'ensemble des différents codes, pour pouvoir mettre en place les focntions qui vont faire fonctionner le jeu, comme par exemple la fonction *play*. Cette fonction est la plus importante de l'executable.

- **rules.[ch] :** Le jeu est régi par certaines règles, qui vont être définies dans les fichiers **rules.[ch]**.
  Les fonctions vérifient que les règles sont respectées. Une autre fonction va indiquer si le teddy doit être exclue ou non.
  Ces fonctions sont utilisées dans la boucle de jeu finale.

- **move.[ch] :** L'achievement 1 a offert aux teddies la possibilité de changer de stockex. Afin de pouvoir réaliser ce mouvement, des fonctions ont été écrites dans **move.[ch]**.
  Elles prennent en compte la position actuelle du teddy, et les postions auxquelles il peut se rendre, en fonction des transactions à sa disposition.

- **rng.[ch] :** Ces fichiers implémentent simplement la fonction de génération de nombre aléatoire, qui sera utilisée dans le projet.

4.5 Affichage des résultats

Une fois la partie terminée, c'est à dire une fois que chaque teddy a un temps de jeu égal au temps de jeu maximal défini en début de partie, il faut afficher les résultats. Ainsi, la fonction *display results* se charge d'afficher le classement. Pour chaque teddy restant, elle calcule l'équivalent-Honey de son **wallet**, puis elle effectue un **tri par insertion** sur la file de priorité composée des teddies.

Nous avons décidé de choisir ce tri car, même s'il est d'une complexité asymptotique quadratique, il est en moyenne beaucoup plus rapide que le quicksort ou que tri fusion pour des jeux de données de petites tailles. Or, notre projet s'effectue dans un contexte de 20 teddies maximum.
C'est donc pour cette raison que le tri que nous avons choisi d'implémenter est le tri par insertion.

5 Tests réalisés

Afin de vérifier le code, un fichier de test est mis en place. Les tests réalisés permettent de mettre en avant tout les résultats faux, après l'execution de certaines fonctions. Ces tests n'ont été réalisé que sur certaine fonctions, mais auraient pu être effectués sur des fonctions différentes. Ces tests ne servent que à aider pour le code, mais ils ne remplacent pas les tests imposés par la forge de l'école.

6 Conclusion

Ce projet a permis à chaque membres du groupe d'améliorer ses compétences en langage C, et en LaTeX. La division du projet en sous-problèmes nous a appris comment relier différents fichiers, gérer les erreurs, et ainsi pouvoir construire un fichier executable. De plus, nous avons beaucoup appris sur le travail de groupe, et comment bien gérer les partages de fichiers. Cependant, le projet n'est pas encore fini, et il reste des achievements à traiter. Nous avons fini l'achievement 1, qui consistait à implanter dans le jeu plusieurs places d'échanges, et à faire intervenir le choix des teddies. L'achievement 2 leur implémentera la notion de stratégie. Ainsi, nous avons réussi à faire fonctionner le jeu, même si il y a encore beaucoup de points d'améliorations, et d'évolutions possibles sur le projet.

Sources images :

- *Reprsentation enfantine classique du "nounours"* par Frédéric Bellaiche.
- Logo Bordeaux INP ENSEIRB