Projet de Programmation - Rush Hour / Âne Rouge

Lisa Arneau - Lucas Ghibert - Clément Boutet

27 avril 2016

1 Introduction

Ce rapport présentera le travail effectué en cours d'EDD et PP durant le semestre de Printemps de l'année 2016 du groupe A2-C.
Il mettra en lumière les résultats auxquels nous sommes arrivés lors de la programmation de notre jeu, les problèmes rencontrés et les méthodes de résolutions mises en places pour s'en débarrasser.

2 Présentation du projet

Ce programme permet de jouer à deux jeux, Rush-Hour et l'Âne rouge.
Lorsque le programme est lancé, la possibilité vous est donnée de jouer à deux jeux différents :

— Rush Hour
— Âne Rouge

Ce sont deux jeux de type puzzle.
Une fois le jeu choisi, vous avez la possibilité de choisir la configuration de votre choix, dans une liste qui vous est proposée. Les niveaux sont chargés à partir de fichiers txt écrits dans une configuration particulière.

2.1 Objectifs atteints

Le jeu est fonctionnel. Aucun bug, ainsi qu'aucune fuite mémoire n'ont été détectés lors des nombreux crash-test que nous avons effectués sur le programme.
Le chargement des configurations de jeu se fait correctement, aucun problème de lecture détecté.
La fonction intersection fonctionne bien, les déplacements autorisés se font sans problème.
Il y a un affichage de messages d'erreur pour expliquer au joueur ce qu'il doit et ne doit pas faire.

Le make se passe correctement, les fichiers sont construits au bon endroit. Le jeu se lance sans problème.

2.2 Objectifs non-atteints

Le solveur est fonctionnel, mais sa vitesse optimale n'est pas atteinte.

3 Problèmes techniques et leurs résolutions

On ne va rien vous cacher : nous avons rencontré de nombreux problèmes lors de l'écriture de notre code. Ils allaient des plus simples (*"Eh, mais t'es bête, t'as oublié un #include ici !"*), aux plus complexes (*"Hm, non, c'est pas bon du tout là, je pense qu'on est bons pour recommencer toute cette portion de code... Regarde, là t'as une boucle infinie, ici t'as oublié un free, cette fonction ne marche pas du tout..."*).
Plusieurs nous ont particulièrement donné envie de lancer nos ordinateurs par la fenêtre.

Tout d'abord, la fonction permettant de lire quelque chose tapé dans le terminal de commande. Il nous a également fallu se rappeler le fonctionnement des fonctions gets, scanf, etc, etc, et finalement, nous avons décidé d'écrire notre propre fonction pour éviter les duplications de code à chaque fois qu'il fallait lire un chiffre dans la configurations. Cette fonction, nommé scan, utilise une chaîne de caractère buffer, créée avant toute utilisation de la fonction. Elle est passée en paramètre de scan. Ensuite, à l'intérieur de scan, on utilise la fonction fgets, pour stocker dans buffer les caractères tapés dans le terminal. Ensuite, rewind(stdin) réinitialise l'entrée standard pour réutiliser le buffer dans une autre lecture de caractères tapés.

Ensuite, les fuites mémoires. Beaucoup, beaucoup de fuites mémoires. Les fuites mémoires fûrent au final assez facile à résoudre, un petit coup de valgrind, une bonne relecture du code, et l'ajout de tous les free oubliés. Il fut cependant assez long de tout résoudre car la fatigue aidant parfois, on passait à côté de certaines erreurs...

La fonction intersect nous a également beaucoup posé problème. On cherchait l'optimisation du code avec un minimum de boucles for, et finalement nous avons décidé d'opter pour une solution qui marchait mais n'étant pas forcément la plus élégante (à savoir 4 boucles for imbriquées, plus un if). Mais bon, ça marche !

Au final, les problèmes rencontrés étaient essentiellement dûs à la recherche d'optimisation, des fautes d'inattention, des petits oublis provoquant des gros bugs... Et une bonne partie de ces problèmes ont été résolus grâce à une seconde relecture par une personne autre que celle qui avait écrit le code, afin d'être plus à même de détecter les erreurs.

(*"Tu te rends compte que tu as juste pas mis les configurations dans le bon dossier, et que c'est pour ça que t'arrives pas à les charger ?"*)

(*"Hm, t'as oublié de modifier tes fichiers quand t'as fait ton merge..."*)

(*"Peut-être que si tu pull la partie de code qu'il te manque ça marchera mieux, tu penses pas ... ?*)

4 Factorisation du code

Le code a été, à nos yeux, factorisé au maximum. Nous avons créé de nombreuses fonctions permettant de réduire la quantité de lignes dans nos fichiers sources.

Citons par exemple la fonction scan, qui nous évite d'avoir à nettoyer le buffer après chaque lecture de caractère. Ce nettoyage est compris dans la fonction et donc fait automatiquement.

Ou encore la fonction Choice_config, qui elle, permet de traiter l'ouverture du fichier, la création de différentes variables et l'initialisation du jeu en prenant en paramètre le numéro de la configuration choisie par le joueur. Ca nous évite d'avoir à ré-écrire ces vérifications et initialisations à chaque cas de choix dans le main.

Il y a également la fonction new_piece_rh que l'on a codée de manière à utiliser la fonction new_piece pour créer nos pièces, évitant ainsi de faire deux fonctions quasiment identiques.