Rapport de projet RushHour

Emilio Abril, Raphaël Anquetil, Benjamin Camblor, Brian Lebreton

28 avril 2016

Table des matières

1 Débuts

1.1 Organisation

Au début nous avons eu un peu de mal à nous organiser. En effet nous nous connaissions peu et nous avons eu un peu de mal à nous voir pour parler du projet. Une fois ceci accompli nous nous sommes répartis les tâches puis avons commencé à travailler. Nous avons d'abord commencé à travailler sur les machines du CREMI puis ensuite migrer sur nos machines personnelles en utilisant un IDE sous Windows. Ce qui a été une erreur car nous avons perdu du temps pour débugguer nos erreurs par la suite. Pour la gestion des versions nous avons utilisé Git.

Pour la partie compilation des programmes nous avons d'abord commencé par utiliser des Makefile puis avons vite migrer sur cmake qui était beaucoup plus simple à utiliser vu que le cahier des charges allait changer dans le futur.

Au niveau du débuggage, nous avons beaucoup travaillé avec gdb et valgrind tout au long du projet.

1.2 Mise en place de RushHour

La première partie du développement de ce projet à consister à implémenter les fichiers sources associés aux fichiers include que l'on nous avait fourni. Il s'agissait de définir les fonctions principales et essentielles du jeu à travers deux objets : une pièce et un jeu. Ces deux éléments constituaient les briques élémentaires qui composaient un jeu. Le but était d'offir une implémentation simple et modulable et de proposer des services de construction d'un jeu.
Cette partie demanda un temps considérable du fait que nous ne savions pas trop comment implémenter ces deux structures. Les fontions nous paraissaient un peu floues au départ mais nous avons au final réussi à obtenir un résultat fonctionnel et plus ou moins optimisé.
Le plus gros problème que nous avons eu lors du développement est survenu lorsque nous avons vérifié les fuites mémoires de notre programme avec valgrind. En effet nous avions une multitude de fuites car nous avions dès le début mal implémenté les constructeurs de 'piece' et 'game'. Cela nous a donc retardé dans l'avancement du projet et avons donc rendu un programme avec des fuites.

1.3 Affichage en mode console

La création de la grille du jeu était simple, par contre, comment afficher les numéros des pièces dans la grille a été autrement plus compliqué. Pour cela nous avons utilisé une matrice de la taille du jeu de RushHour dans laquelle on mettait toutes les cases à un nombre non utilisé dans le jeu : -1. Ensuite il a fallu grâce aux pièces déjà créées, placé le numéro de chaque pièce à leur place et suivant leur forme (verticale ou horizontale et petite ou grande).
C'est à ce moment que l'on commence à afficher le squelette de la grille et on parcourt la matrice comprenant les pièces et les cases vides contenant la valeur '-1'. Si le contenu de la matrice est égal à -1 on affiche rien, on obtient donc une case vide, sinon on affiche le numéro de la pièce.

Le seul problème rencontré ici à été de savoir comment parcourir la matrice pour afficher les pièces pour qu'elle soit dans le bon sens pour un repère x/y.

2 Modification du cahier des charges

2.1 Généricité du code

Le cahier des charges étant changé et devant pouvoir s'appliquer à un nouveau jeu 'Ane rouge', il a fallu modifier la structure et certaines fonctions de piece.c ainsi qu'en rajouter de nouvelles.
Par exemple, certaines fonctions propres à RushHour étaient identiques ou presque à celles devant être créées pour l'Ane rouge. Dans la précipitation certains paramètres n'ont pas été modifiés et ont laissé place à des erreurs lors du lancement de l'Ane rouge. En effet, la taille de la grille de jeu de RushHour est restée dans les fonctions de l'Ane rouge ce qui est gênant car les pièces pouvaient alors sortir de la grille du jeu de l'Ane rouge car l'affichage lui avait été modifié.

Il a donc fallu repasser sur ce que nous avions modifié afin de bien vérifier que tout était en ordre et générique aux 2 jeux.

3 Solveur

La réalisation du solveur a été un objectif que nous n'avons pas réussi à mener jusqu'au bout. Nous n'avons pas pu le rendre dans les temps, et bien que nous ayons continué à travailler dessus passé ce délai, l'exécution du programme engendre un plantage après l'exploration de plusieurs couches de l'arbre des issues au jeu d'origine. L'idée était claire, nous avons utilisé 3 structures : une structure FIFO, et deux structures de listes doublement chaînées, l'une contenant des game et l'autre contenant des listes DC de game. Nous avons donc implémenté ces structures ainsi que toutes les fonctions dont nous avions besoin pour les manipuler. Notre programme est sensé fonctionner (globalement) de la manière suivante :

```
initialiser jeu d'origine
enfiler jeu dans FIFO
ajouter jeu à liste: ListeDC_game
ajouter liste à chemins: ListeDC_list

tantque valeur(FIFO) n'est pas jeu gagnant
    current = defiler(FIFO)

    récupérer toutes les issues à current

    chercher la première liste de chemins
    tq son dernier game est égal à current

    ajouter à la fin de chemins (nb issues)-1
    copies de la liste précédente

    pour i de 0 à (nb issues)-1
        ajouter issues[i] en queue de
        la i-ème liste de chemins dont
        le dernier game est égal à current

        enfiler issues[i]

    fin pour
fin tantque
```

Nous avons implémenté toutes les fonctions permettant de faire fonctionner cet algorithme (récupération des issues dans un tableau de game à partir d'un game donné, contrôle des déplacements autorisés, etc). Une erreur empêche le fonctionnement du solveur, il semble que le plantage survient lorsque l'exploration de l'arbre aboutit sur une configuration gagnante. Malgré cette piste, nous n'avons pas réussi à résoudre le problème, ce dernier se révélant après que le programme ait prit une complexité importante. Nous avons donc décidé de ne pas rendre le solveur, plutôt que de rendre quelque chose qui ne fonctionne pas. Le code source est disponible dans différents fichiers (solv.c, solv.h, liste_dc.c, liste_dc.h, fifo.c et fifo.h). Nous avons également apporté quelques modifications au game.c pour résoudre des bugs lors de l'implémentation de certaines fonctions du solveur (exemple : la fonction piece* game_pieces_copy(cgame g); sans laquelle la fonction get_issues de solv.c ne fonctionnerait pas). En conclusion, nous n'avons pas atteint l'objectif de rendre un solveur qui marche, mais nous avons tout de même dépensé beaucoup de temps et d'énergie pour le coder et tenter de le rendre fonctionnel.

4 Partie graphique

4.1 Développement de l'interface graphique

1ere étape : création des composants de l'interface Avant tout, il s'agissait de créer les composants de notre interface. Tout d'abord, un fond sur lequel était superposé des cases blanches et grises qui étaient censées représenter le cadrillage de la grille de jeu. Ces composants sont communs aux deux jeux. Ensuite, il a fallu créer les pièces pour chaque jue : des voitures pour RushHour et de simples cases colorées pour l'AneRouge. Nous avons aussi intégré une image qui est affichée lorsque le jeu est terminé. Enfin, des flèches directionnelles ont été intégrées pour aider l'utilisateur lors du mouvement des pièces. Nous avons aussi ajoutés quelques polices éxotiques au projet.

2eme étape : Découpage fonctionnel de l'affichage

-afficher_grille : Cette fonction se charge d'afficher la grille de jeu principale, en fixant la taille de la fenetre. Cette fonction est appelée à chaque fois que l'on réaffiche l'intégralité du jeu.

-afficher_voitures/afficher_pieces : Cette fonction se charge d'afficher au bon endroit et dans le bon sens les voitures/pieces sur la grille de jeu. Cette fonction est appelée à chaque fois que l'on réaffiche l'intégralité du jeu.

-afficher_sortie : Cette fonction se charge d'afficher la sortie de jeu. Cette fonction est appelée à chaque fois que l'on réaffiche l'intégralité du jeu.

-affichage_graphique : Cette fonction appelle les fonctions suivantes afficger_grille, afficher_voitures/afficher_pieces, afficher_sortie. Elle permet en somme de rafraichir le contenu de la fenêtre après qu'un mouvement ait été effectué. Cette fonction est donc appellés à chaque fois qu'un mouvement est effectué.

-identifier_voitures/identifier_pieces : Cette fonction permet d'identifier quelle voiture/piece a été sélectionné au moyen d'un clic de la souris de la part de l'utilisateur.

-afficher_directions : Cette fonction permet d'afficher les directions disponibles lors du mouvement d'une piece.

-debut_jeu_graphique : Cette fonction se charge de demander à l'utilisateur d'effectuer un mouvement tant que le jeu n'est pas terminé et d'afficher l'intégralité du jeu.

4.2 Choix de la bibliothèque graphique

Au début de la phase de développement de l'interface graphique, deux bibliothèques nous ont été proposées : -SDL qui permet la gestion d'images, de fichiers sons et d'évènements liés à la souris et au clavier. -MLV qui est est une interface simplifiée de la bibliothèque SDL.
Bien que SDL soit plus flexible et permette plus d'options que MLV, nous avons privilégié la facilité d'accès pour perdre le moins de temps sur la phase d'apprentissage. Les interfaces des deux jeux nous étaient apparues assez simples et épurées et nous n'avions pas jugé nécessaire de se compliquer la tâche avec SDL alors que MLV nous aurait apporté un résultat très similaire, sinon le même. Une fois celle-ci choisie, l'utilisation de celle-ci fut très simple grâce à sa documentation enrichie. Des exemples d'utilisation des fontions essentielles étaient aussi disponibles ce qui nous a permi de rentré très vite dans le bain. Le plus difficile n'a donc pas été de se servir de la bibliothèque mais plutôt de réfléchir au meilleur moyen de s'en servir pour réaliser une interface graphique propre et légère

5 Critiques

Dans un premier temps, nous utilisons le langage C pour faire de la programmation objet. Il est vrai que c'est plus rapide car c'est un langage bas niveau mais cela nous aurait grandement facilité la tâche si nous avions pu utiliser un langage de haut niveau comme Java qui est fait pour ce genre de projet. Ensuite nous n'avions pas la liberté d'implémenter le jeu à notre manière, nous devions impérativement faire les méthodes de game.c et piece.c présentes dans les fichiers '.h' fournis.
Nous regrettons le manque de clarté sur la façon dont nous devions organiser le code(quelle fonction dans quel fichier) car à plusieurs reprises on nous a dit de mettre ces fonctions dans un certain fichier puis on nous dit le contraire.
Dans l'ensemble ce projet a été un petit entraînement de ce que sera le travail en groupe dans le monde du travail et nous a permis d'en avoir un aperçu grâce aux commanditaires(les professeurs), au cahier des charges modifié par le client au fil du projet et enfin la date limite de rendu.