Synthèse finale du projet de jeu 2048

Guillaume ALMYRE Gaëtan CHAMBRES
Allan MAHAZOASY Chrystelle PETUREAU

21/04/2015

Que fait notre projet/ documentation utilisateur :

Nous avons crée un programme en langage C qui permet de jouer au jeu "2048" avec une interface graphique. Il y a aussi 2 stratégies (une lente et une rapide)pour qu'une intelligence artificielle joue à notre place. Notre travail est constitué d'un dossier contenant les fichiers suivant (le temps entre parenthèses est le temps de travail estimé) :

Makefile (3 heures) : permet de compiler le jeu en utilisant la commande "make" dans le terminale. Le plus long dans ce fichier a été d'arriver à trouver la ligne pour créer la bibliothèque libgrid et de compiler avec.

gric.h : est l'interface du jeu, à laquelle nous avons absolument pas touché.

grid.c (15 heures) : constitue le code principal du programme.

test-grid.c (8 heures) : est le fichier où tous les tests ont été effectués.

jeu.c (6 heures) : est le fichier pour créer l'exécutable pour jouer.

module stratégie rapide (1 heure) : (explication plus bas)

module stratégie lente (? ?) : (explication plus bas)

Répartition du travail :

Notre groupe constitué de 3 personnes au départ le 27 janvier, a été agrandi à 4 personnes, le 10 mars, avec l'arrivée d'Allan.
Nous nous sommes répartis le travail en fonction de notre savoir-faire ainsi que nos avis. Gaëtan et Guillaume ont beaucoup travailler sur l'implémentation de grid.c et stratégie.c. Chrystelle sur test-grid.c ainsi que tous les comptes rendus. Grâce à ce découpage, il y a eu presque deux équipes au sein du projet. Ce qui nous a permis de travailler sur la lisibilité du code : si une équipe arrive à comprendre et utiliser le code sans y avoir participé, cela veut dire que le code est lisible et bien commenté.
Allan a essayé de prendre "prendre le train en marche avec beaucoup de bonne volontés. Nous l'avons aidé du mieux qu'on a pu mais il n'avait jamais codé en C, ne connaissait pas le jeu 2048 et était très peu présent en cours donc son travail n'a pas été significatif.

L'utilisation d'un framapad, initialisé avec le contenu du PDF "sujet" puis mis à jour à chaque modification, a été d'une grande aide. Il nous a permis au fur et à mesure de savoir sur quelle partie du code travaillé. La fonction "historique" du framapad ainsi que celle du servit github ont été d'une grande aide pour retrouver les étapes de notre travail et surtout le temps passé dessus.

Stratégie :

Le fichier strategy.h est la base de tout notre travail. Nous ne l'avons absolument pas modifié. On doit définir une structure pour chaque stratégie. Cette structure contient : un nom sous forme de char, une variable

qui mémorise les moments, un pointeur de fonction qui applique la stratégie et trouve la direction à effectuer et enfin un pointeur de fonction qui libère la mémoire (pour ne pas avoir de problème de mémoire pleine). L'utilisation de pointeurs de fonctions permet de garder le code identique, sans gros changement. Cela évite la création involontaire de bogues.
Donc, en résumer, pour chaque stratégie il y a en commun :

- Makefile
- grid.h
- grid.c
- strategy.h
- strategy.c

Seulement 2 fichiers sont différents suivant les stratégies : fast.h, fast.c ou efficient.h, efficient.c.
Les fichiers fast.h et efficient.h sont obligatoires car nous avons crée un fichier test-stat pour tester ces stratégies. Les appels de test-stat à fast.c ou effecient.c doivent ce faire par le biais d'un .h. On trouve dans ce .h la signature d'un constructeur qui initialise la structure de la stratégie concernée. Il y a aussi la signature de la fonction qui applique la stratégie et retourne une direction.
Les tests sont faits à partir d'un simple copier-coller de "jeu.c" avec l'ajout de la ligne :

```
strategy s=fastInit();
```

ou

```
strategy s=efficientInit();
```

Stratégie rapide :

Dans le fast.c, on trouve, comme prévu, un constructeur fastInit qui initialise la structure s ainsi qu'une fonction stratFast qui choisi une direction au hasard.
La création de cette fonction a été relativement facile (environ 1 heure), la compréhension du travail demandé et l'utilisation des pointeurs de fonctions étant beaucoup plus chrono-phage (environ 5 heures).

Stratégie lente :

Le principe de base est de chercher la direction qui minimise l'évaluation e de la grille g. La formule utilisée est :

$$e = 2 * L + 2 * US + 6 * M$$

AVEC :

$$L = \sum_{i=0}^{GRIDESIDE} . \sum_{f=0}^{GRIDESIDE} [|(t(i,j) - t(i-1,j))| + [t(i,j) - t(i,j-1)|]$$

$$US = card(t \in g/(t(i,j) > 0 eti, j \in [0, grideside])$$

$$M = min(sup(i), inf(i)) + min(sup(j), inf(j))$$

Bogues vaincus :

Bogues restants :

Tests effectués :

tests : Nous avons crée des test spécifiques pour chaque fonctions. Ils sont disponibles dans test-grid.c. Cependant nous allons les détailler ici aussi :

afficher, set-tile et get-tile : Premières vraies fonctions qui ont été crées, elles sont vitales pour tester les autres. Depuis le début du projet, elles ont servit un nombre incalculable de fois sans jamais révélées aucun problème.

new-grid : On l'a testé dans le ficher test principal. Elle fait partie aussi de la base des tests des autres fonctions et n'a révélé aucun problème à ce jour.

add-tile : Son test a consisté principalement à associer new-grid, afficher et add-tile dans une boucle for pour vérifier son aspect aléatoire.

copy-grid : Associée à afficher, elle nous semble tout à fait identique.

delete-grid : On fait un fichier test avec new-grid, add-tile, copy-grid puis delete-grid. Ce fichier a été lancer avec valgrind pour vérifier les éventuelles fuites de mémoire.

do-move : Elle a été testée dans une boucle for avec l'utilisation de la fonction random.

grid-score et can-move : Sont testées à chaque boucle.

play et game-over : Un fichier particulier a été créer pour tester les possibilités de jeux. On a utilisé la bibliothèque NCURSES pour cela.

Avis critiques sur les outils proposés :

Makefile/Cmake :

Toute l'année de L1 ainsi que le semestre dernier nous avons rêvé d'un logiciel qui crée les Makefile automatiquement, ces derniers étant vraiment compliqués à créer surtout dans la bonne utilisation des variables. Cependant à force de persévérance, on a réussi à créé un Makefile "type" que l'on utilise dans tous les projets. Après le test de Cmake effectué par la feuille de TP 2, Nous avons décidé de garder notre Makefile car il semble finalement plus simple à utiliser, ne demande pas d'avoir Cmake d'installer sur l'ordinateur et enfin il crée une archive beaucoup plus propre.

SVN/git :

Après les tests de SVN et GIT de la feuille de TP3, nous avons choisi de privilégier Git par rapport à Subversion car l'outil nous a semblé plus intuitif.

L'adresse du dépôt utilisé est : `https://github.com/projetL2/2048.git`
Les identifiant sont :

username : projetL2

password : IN4001ggc (pour Guillaume, Gaëtan, Chrystelle)

GDB/valgrind/tests automatiques et preuves :

GBD : Nous avons pas utilisé GDB car la techniques des "printf" a suffit à nous faire comprendre ou été les bogues et comment le résoudre.

valgrind : Nous avons utilisé valgrind pour vérifier les fuites de mémoires qui étaient nulles. Cependant, ce test a été effectué avant l'utilisation de la bibliothèque NCURSIVE. qui crée à elle seule beaucoup de fuite mémoire. Si on test avec la ligne de commande suivante aucune fuites apparaît :

```
valgrind --leak-check=full --num-callers=50 --suppressions
=./valgrind_lif7.supp --show-reachable=yes -v EXUCUTABLE
```

tests automatiques et preuves : Nous avons pas crée de tests automatiques par méconnaissances de ce genre de système au moment opportun. Mais avec le recul, et en voyant les tests dans les codes de nos collègues,

on pensent que cela peut être une très bonne choses pour vérifier la compatibilités avec un autre système d'exploitation par exemple.
Nous avons pas utilisé de tests automatiques comme avec "gcov". "Framac" n'a pas été utilisé non plus car le jeu 2048 n'est pas "sensible" et ne demande pas d'être prouvé mais cela semble être un outil très intéressant pour des programmes qui exigent d'être sans faille.

Commentaires :

Eclipse :

Emacs :

lateX :

Nous avons décidé de faire les synthèses intermédiaires en lateX pour s'y habituer avant le rapport final.

Évaluation de 3 autres groupes :

Difficultés rencontrés :

Conclusion :