

Rapport Final projet EDD

Pinero, Borde, Bonnet

19 avril 2015

Table des matières

Table des matières	1
1 Le projet	3
1.1 Présentation	3
1.2 Le sujet	3
2 Développement du 2048 et tests	4
2.1 Le developpement de grid.c	4
2.1.1 Implementation de « can_move »	5
2.1.2 Implementation de « do_move »	5
2.2 L’affichage de la grille dans la console	6
2.3 Les tests	7
3 Relectures critiques	8
3.1 Les relectures que nous avons réalisé	8
3.2 Les reslectures réalisé par les autres groupes	8
4 Les stratégies	10
5 Une vrai interface graphique ? !	11
6 En bref	12
Références	13

7 Annexes	14
Table des figures	14

1 Le projet

1.1 Présentation

Le projet se compose de plusieurs parties de développement sur un sujet commun, le jeu du 2048. Ce jeu consiste à additionner des tuiles de même puissance de 2, sur un plateau de 16 tuiles, jusqu'à ce que le plateau soit plein et qu'aucune addition ne soit possible. Le but du jeu est d'arriver à construire une brique de valeur la plus grande possible (2048, 4096, 8192, ...)

POURQUOI PAS AJOUTER UNE PHOTO EXEMPLE ?!

1.2 Le sujet

Dans le projet nous avons eu à réaliser dans un premier temps l'ensemble du moteur du jeu et quelques tests qui nous garantissent un maximum la véracité pendant le développement. Dans un deuxième temps, nous devons faire la « critique » du développement de la première partie faite par d'autres groupes basé sur une grille d'évaluation qui nous a été fournie. Ensuite nous avons réalisé un ensemble de stratégie de jeu qui sont capable de jouer de manière autonome une partie entière, jusqu'à la fin de la partie. Ces stratégies doivent à chaque tours calculer le meilleur prochain coups pour atteindre le plus grand score possible. La dernière partie du projet consiste à réaliser une véritable interface graphique pour jouer au jeu.

2 Développement du 2048 et tests

Pour le développement du 2048, le langage de programmation était imposé, nous devons utilisé le langage C. Chaque groupe ont dû fournir les sources permettant de générer une bibliothèque libgrid.a (implémentant les fonctions d'écrites dans grid.h en annexe 1) ainsi qu'un exécutable permettant de jouer au 2048 dans la console. En plus de ça, nous avons réalisé des tests pour vérifier toutes les fonctions implémentés.

Avant de commencer le travail de développement, nous avons utilisé un outil de gestion de versions décentralisé pour que chaque membre du groupe puisse avoir accès à chaque instant à la dernière version du projet. Pour ça nous avons choisit le logiciel GIT, et la plateforme GitHub. Les sources du projet ont été disponible tout au long du développement ici.

En ce qui concerne l'arborescence de notre projet, nous avons décidé d'avoir une organisation bien précise pour quelle puisse être le plus possible générique, c'est à dire quelle s'adapte aux évolutions du projet. Mais surtout pour quelle soit le plus clair possible.

```
EDD_projet/      # Racine du projet
  bin/           # Fichiers exécutables.
  build/         # Fichiers de compilation du projet
  include/       # Fichiers .h des librairies générés
  lib/           # Libraires
  src/           # Fichiers tous les fichiers sources
    2048/        # Fichiers sources de l'interface graphique
    grid/        # Fichiers du moteur de jeu
      tests/     # Fichiers qui testent le moteur de jeu
```

Tout au long du développement du projet, tous les membres du groupe se sont mis d'accord sur le fait d'utiliser l'outils CMake pour compiler le projet, générer les Makefiles qui servent eux-mêmes à générer les exécutables, les libraires et exécuter les tests.

2.1 Le developpement de grid.c

En ce qui concerne le développement de grid.c, qui est le moteur du jeu, c'été relativement simple. Les fonctions qui ont mérité un réflexion sérieuse

sont « do_move() » et « can_move() ». Pour ces fonctions nous devons limité au mieux la duplication de code et la complexité de l'algorithme. Chose que nous n'avions au final pas très bien réussi, mais nous veront dans la pratie 3.2.

2.1.1 Implementation de « can_move »

Pour cette fonction, nous avons choisi l'algorithme suivant :

```
Si la direction donnée est « UP » ou « DOWN » alors
  Pour chaque colonne faire :
    Pour chaque tuile de la colonne faire :
      Si la tuile précédente est égale à la courante
        retourner vrai

      Si la tuile courante et non vide
        et qu'il y a une tuile vide dans la colonne
          retourner vrai
Si la direction donnée est « RIGHT » ou « LEFT » alors
  Pour chaque ligne faire
    Pour chaque tuile de la ligne faire
      Si la tuile précédente est égale à la courante
        retourner vrai

      Si la tuile courante et non vide alors
        et qu'il y a une tuile vide dans la ligne
          retourner vrai
retourner faux
```

Cette fonction est utilisée à plusieurs reprises dans d'autres fonction comme par exemple « game_over » ou « do_move ». d'où l'importance qu'elle n'ai pas une trop grande compèxité de calcul.

2.1.2 Implementation de « do_move »

Pour cette fonction, nous avons choisi l'algorithme suivant :

```

Si la direction donnée ne peut pas être jouer
    Ne rien faire

Si la direction donnée est « UP » ou « DOWN » alors
    Pour chaque colonne faire :
        Pour chaque tuile de la colonne faire :
            Si la tuile précédente est égale à la courante
                On fusionne les deux tuilles
                On modifie le score de la partie

            Si la tuile courante est vide alors
                On change sa place avec la première tuile non vide

Si la direction donnée est « RIGHT » ou « LEFT » alors
    Pour chaque ligne faire
        Pour chaque tuile de la ligne faire :
            Si la tuile précédente est égale à la courante
                On fusionne les deux tuilles
                On modifie le score de la partie

            Si la tuile courante est vide alors
                On change sa place avec la première tuile non vide

```

2.2 L’affichage de la grille dans la console

Pour l’affichage de la grille dans la console, nous avons eu plusieurs version et design. La toute première version était une simple grille dans la console où était affiché le score et les valeurs des tuilles. Dans cette version nous devons jouer avec les lettres Z, Q, S et D du clavier car dans le langage C, la fonction « scanf » de la librairie « stdio » qui permet de lire l’entrée standard, ne prend pas en compte les flèches du clavier. En annexe vous trouverez une capture d’écran de cette version.

Dans une deuxième version, nous avons décidé de mettre des couleurs dans les tuilles en fonction de leurs valeurs. (à voir en annexe 2).

Enfin pour répondre correctement au sujet, nous avons utilisé la librairie

« NCurses ». L'utilisation de cette librairie nous a permis de se libérer de deux de nos difficultés. La première était celle que nous devions jouer avec des lettres et non les flèches du clavier, la seconde était que nous n'avions d'affichage dynamique de la grille à chaque coup. Pour être plus clair, lorsqu'on joue un coup avec les deux premières versions de l'affichage, la grille du coup précédent était toujours présente dans la console. Désormais, la console n'affiche plus qu'une grille et elle met à jour après chaque coup.

2.3 Les tests

Les Tests sont écrits pour confronter une réalisation à sa spécification. Le test définit un critère d'arrêt (état ou sorties à l'issue de l'exécution) et permet de statuer sur le succès ou sur l'échec d'une vérification. Grâce à la spécification, on est en mesure de faire correspondre un état d'entrée donné à un résultat ou à une sortie. Le test permet de vérifier que la relation d'entrée / sortie donnée par la spécification est bel et bien réalisée. [Wik15]

Sur ce principe nous avons réalisé un test par fonction décrite dans le fichier « grid.h » et un test sur la réalisation de plusieurs parties qui s'exécutent avec une stratégie basique qui consiste à jouer à gauche si on le peut sinon à droite, sinon en haut enfin en bas. Ainsi lorsqu'on fait des modifications de l'une de ces fonctions, nous nous assurons que cela n'a en rien changé à la véracité des résultats. L'exécution des tests s'effectue avec la commande « make test ». L'utilisation de la technologie « CMake » nous permet d'avoir un affichage complet à la fin des tests sur leurs résultats pour savoir s'ils sont passés avec succès ou non (exemple en annexe 3).

3 Relectures critiques

À l'issue de la première phase de travail, chaque groupe ont dû relire le code produit par trois autres groupes. Pour chaque groupe à relire, on a rempli un formulaire d'évaluation. Ce formulaire d'évaluation nous a été fourni par notre chargé de TD.

Cette feuille d'évaluation était composée de 13 questions orientées sur trois axes. Le point de vue utilisateur, qui pousse à vérifier principalement la propreté de l'archive, que la compilation ne génère pas d'erreur et qu'il y a un fichier « README » pour expliquer l'utilisation du programme si ce n'est pas évident. Dans un deuxième temps, d'un point de vu fonctionnel, nous devons vérifier si le programme ne présente pas de bogue ou de fuite mémoire. Enfin, d'un point de vue programmation, nous devons porter notre attention sur les patrons de conception (ou *design patterns* en anglais).

3.1 Les relectures que nous avons réalisé

Nous avons eu à faire la relecture des groupes A, D et G. Comme nous sommes trois, nous nous sommes divisé le travail et avons évalué un groupe chacun. Nous nous sommes ensuite concerté pour harmoniser nos commentaires. Cette relecture fût très intéressante car elle nous a permis de mettre au premier plan les points importants sur lesquels il faut insister lors du développement d'un projet. En plus de cela, comme chaque groupe à eu le choix sur la façon d'implémenter son programme, nous avons vu les différentes possibilités d'implémentation parfois meilleure et plus efficace ou pas. Le but n'était pas d'être trop sévère ou au contraire laxiste sur nos commentaires car ceux que nous avons donné sur chaque projet n'ont pas été noté.

3.2 Les relectures réalisé par les autres groupes

En ce qui concerne les commentaires qui ont été faits par les autres groupes sur notre projet, nous les avons trouvé justifié. Memê si, il faut l'avouer, recevoir des « critiques » sur le travail que nous avons réalisé pendant près de trois ou quatre semaines fait un peu grincer des dents. Nous avons pris ces critiques en compte et en avons corrigé un maximum avant le rendu final.

Nous avons principalement reçu des commentaires négatifs sur ma duplication de code dans l'implémentation du moteur de jeu. Certaines fonctions

dans utilisé dans les fonctions « `can_move` » et « `do_move` » se ressemble fortement. Un autre point à été remarqué, le nomage des variables et les commentaires dans le code source étaient certaines fois en français et d'autre en anglais alors que les patrons de conceptions nous imposé de les mettre qu'en anglais.

4 Les stratégies

5 Une vrai interface graphique ? !

6 En bref

Références

[Wik15] Wikipédia. Test unitaire. http://fr.wikipedia.org/wiki/Test_unitaire, 2015. [Online ; accessed 19-April-2015].

7 Annexes

Table des figures

1	Debut du fichier grid.h	15
2	Deuxième version d’affichage de la grille	16
3	Résultat exécution des tests	16

FIGURE 1 – Debut du fichier grid.h

```

1  #ifndef _GRILLE_H_
2  #define _GRILLE_H_
3  /**
4   * \file grid.h
5   * \brief Contains structures and functions needed to play 2048 game.
6   **/
7
8  #include <stdbool.h>
9
10 /** Grid dimension */
11 #define GRID_SIDE 4
12
13 /**
14  * \brief Contain game's status: tiles and current score.
15  *
16  *
17  *      X
18  *      0 1 ... GRID_SIDE-1
19  *      +---+---+ ... +---+
20  *      | | | ... + | 0
21  *      +---+---+ ... +---+
22  *      | | | ... | | 1
23  *      +---+---+ ... +---+ Y
24  *      ...
25  *      +---+---+ ... +---+
26  *      | | | ... | | GRID_SIDE-1
27  *      +---+---+ ... +---+
28  */
29
30 typedef struct grid_s* grid;
31
32 /**
33  * \brief Log_2-encoded tile : 0 is empty, i is 2**i.
34  */
35 typedef unsigned int tile;
36
37 /**
38  * \brief List of accepted movement in the game
39  */

```

FIGURE 2 – Deuxième version d’affichage de la grille

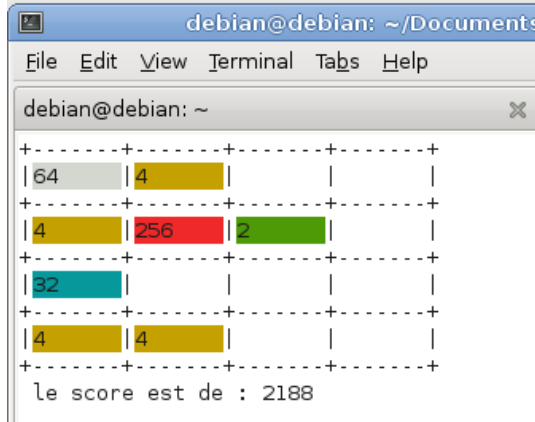


FIGURE 3 – Résultat exécution des tests

```

debian@debian:~/Documents/Cours/EDD/EDD_project/build$ make test
Running tests...
Test project /home/debian/Documents/Cours/EDD/EDD_project/build
  Start 1: test_new_grid
1/7 Test #1: test_new_grid ..... Passed    0.01 sec
  Start 2: test_homogeneite_add_tile
2/7 Test #2: test_homogeneite_add_tile ..... Passed    0.02 sec
  Start 3: test_copy_grid
3/7 Test #3: test_copy_grid ..... Passed    0.01 sec
  Start 4: test_can_move
4/7 Test #4: test_can_move ..... Passed    0.02 sec
  Start 5: test_game_over
5/7 Test #5: test_game_over ..... Passed    0.00 sec
  Start 6: test_do_move
5/7 Test #6: test_do_move ..... Passed    0.01 sec
  Start 7: test_basic_strategy
7/7 Test #7: test_basic_strategy ..... Passed    1.00 sec

100% tests passed, 0 tests failed out of 7

Total Test time (real) =  1.12 sec
debian@debian:~/Documents/Cours/EDD/EDD_project/build$ █
  
```