RAPPORT PROJET 2048

Emery, Gouraud, Kirov, Pouteau

Mercredi 22 Avril 2015

Table des matières

1 Introduction

Dans le cadre de l'UE Environnement de Développement et Projet de Programmation 1, nous avons été amené à créer un jeu 2048.
Ce jeu consiste à obtenir une tuile de valeur 2048 en fusionnant des tuiles de même valeur de manière à en obtenir une deux fois plus grande. Le jeu commence avec deux tuiles de valeur aléatoire (2 ou 4). Il se joue sur une grille de 4x4, avec quatre déplacements possibles (Haut, Bas, Droite, Gauche). A chaque déplacement une tuile de valeur 2 ou 4 apparait aléatoirement sur la grille.
Ce jeu est une application mobile sortie en mars 2014, programmé par Gabriele Circulli, un web-designer italien. Il s'est inspiré du jeu 1024 dont le principe est identique sauf le but qui est d'atteindre pour le coup 1024. Ce jeu 1024 est lui-même basé sur Threes un jeu du même style.
Lien pour tester les différents jeux :

- lien 2048 : `http://gabrielecirulli.github.io/2048/`
- lien 1024 : `http://1024game.org`
- lien Threes : `http://threesjs.com`

Le projet a été séparé en deux parties différentes.

- Réalisation de l'implémentation du 2048 et création d'une interface graphique sur le terminal.
- Réalisation d'une interface graphique indépendante et conception de deux IA capables de résoudre le 2048.

2 Architecture du Projet

On a choisi une architecture classique pour notre projet.

- Un dossier include/ pour toutes les interfaces
- Un dossier src/ contenant
  - Un dossier grid/ contenant les implémentations du jeu général
  - Un dossier ncurses/ contenant l'implémentation de l'interface du terminal
  - Un dossier sdl/ contenant l'implémentation de l'interface graphique et les différentes images nécessaires.
  - Un dossier strategy/ contenant nos deux stratégies fast/low et différentes implémentations pour observer le déroulement des Intelligences Artificielles.
  - Un dossier test-fonction/ contenant l'implémentation des tests

cf : Annexe 1 - Architecture Projet

3 Implémentation du 2048

3.1 Fonction de Base

Implémentation des fonctions décrites dans l'interface fournis *grid.h*.

```
/* Créer une nouvelle grille */
grid new_grid ();

/* Supprime la grille passée en paramètre*/
void delete_grid (grid g);

/* Copie une grille src dans une grille dst */
void copy_grid (grid src, grid dst);

/* Renvoie le score actuel du jeu */
unsigned long int grid_score (grid g);

/* Renvoie la valeur d'une tuile */
tile get_tile (grid g, int x, int y);

/* Met la valeur t dans une tuile  */
void set_tile (grid g, int x, int y, tile t);

/* Verifie que la direction 'd' est valide */
bool can_move (grid g, dir d);

/* Renvoie game_over si il n'y a plus de déplacement possible */
bool game_over (grid g);

/* Effectue un mouvement selon la direction 'd' */
void do_move (grid g, dir d);

/* Ajoute une tuile aléatoirement de valeur 2 ou 4 sur la grille. */
void add_tile (grid g);

/* Effectue le mouvement et ajoute une tuile */
void play (grid g, dir d);
```

Ces fonctions permettent le bon fonctionnement du jeu. Nous avons essayé de simplifier toutes ces fonctions de bases. Pour cela, et afin d'éviter des duplications de code notamment dans `can_move` et `do_move`, nous avons fait appel à des fonctions static.

3.2 Fonction static

Déclaration des fonctions static de grid.c :

```
/* Permet d'incrementer deux variables ayant
  des incrementations differentes. */

static void
incrementation(int *i1, int *i2,
               int incrementationI1,
               int incrementationI2)

/*
 * Déplace l'ensemble de la grille dans la direction voulue
 * Les directions prérequis:
 *    UP    -> (grid, 0, 0, 0, 1)
 *    DOWN  -> (grid, 0, GRID_SIDE - 1, 0, -1)
 *    LEFT  -> (grid, 0, 0, 1, 0)
 *    RIGHT -> (grid, GRID_SIDE - 1, 0, -1, 0)
 */
static void move(grid g, int i, int j, int a, int b);

/*
 * Teste sur l'ensemble de la grille si le deplacement
 * dans la direction voulue est faisable
 * (en fonction des paramètres)
 * Les directions prérequis :
 *    UP    -> (grid, 0, 0, 0, 1)
 *    DOWN  -> (grid, 0, GRID_SIDE - 1, 0, -1)
 *    LEFT  -> (grid, 0, 0, 1, 0)
 *    RIGHT -> (grid, GRID_SIDE - 1, 0, -1, 0)
 */
static bool possible(grid g, int i, int j, int a, int b);

/* Fusionne deux cases et met la deuxième case à 0 */
static void fusion(grid g, int i, int j, int a, int b);

/* Ajoute add_score au score de la grid */
static void set_grid_score(grid g, unsigned long int add_score);
```

Description de l'utilisation des fonctions static :

- **move** : Elle sert à simplifier la fonction `do_move`, car elle effectue tous les déplacements en fonction des paramètres fournis. Par conséquent, `do_move` est reduite à appeler simplement `move` avec les bons paramètre.
- **fusion** : Elle est utilisée pour faire les fusions dans la fonction `move`.
- **incrementation** : Elle permet d'éviter la répétition de code due à l'incrémentation des variables dans `move` et `possible`.
- **possible** : Elle permet de simplifier la fonction `can_move`, car lorsqu'on lui passe certains paramètres, elle vérifie la possibilité de faire un mouvement. Ce qui réduit la fonction `can_move` à appeler la fonction `possible` avec les bons paramètres.

3.3 Réalisation de test

Les tests sont déclenchés par la commande **make check**.

Nous avons utilisé gcov, ce qui nous a permis de savoir que notre programme de test vérifie 94.96% de notre fichier grid.c. Les 5% manquant sont dus au fait que l'on ne teste pas la fonction `play` car elle utilise simplement deux autres fonctions testées précédemment.

```
File 'grid.c'
Ligne exécutées: 94.96% de 139
Creating 'grid.c.gcov'
```

Nous avons testé indépendamment chaque déplacement, pour savoir précisément quel déplacement pourrait provoquer une erreur.

Les tests ont été réalisé de manière à être extensible c'est-à-dire que quelque soit la valeur de `GRID_SIDE` les tests permettent de tester les différentes fonctions.

cf : Annexe 2 - Execution des tests

4 Interface Graphique

4.1 Interface Graphique sur terminal

Nous avons choisi d'utiliser la bibliothèque Ncurses pour pouvoir générer notre grille dans le terminal.
Le jeu se joue avec les flèches directionnelles. Nous avons intégré des couleurs afin d'égayer l'affichage de notre jeu.
Nous proposons également des options supplémentaires :

— q : permet de quitter le jeu
— r : permet de recommencer le jeu

De plus, si vous perdez, on vous offre la possibilité de soit rejouer, soit quitter le jeu.
Nous avons également intégré, par esprit de challenge, un Highscore.
L'affichage graphique s'adapte à la taille de la grille. Mais ne modifie pas la taille du terminal d'où il est lancé.

4.2 Interface Graphique SDL

A partir de la bibliothèque SDL et de ses bibliothèques tierces (`SDL_image`, `SDL_ttf` et `SDL_getenv`), nous avons créé l'interface graphique du 2048 sur une fenêtre indépendante. Cette fenêtre s'ouvre centrée à l'écran, et s'adapte à la taille de la grille (`GRID_SIDE`) définit dans le grid.h. Le texte apparait parfaitement centré et est inhérent à `GRID_SIDE`. La police du texte est aussi en adéquation avec la taille de la grille.
L'interface graphique propose aussi de recommencer la partie (touche "Entrée"), mais aussi d'abandonner (touche "Echap" ou la croix en haut à droite) et ferme le programme.
Lorsque la partie se termine (c'est-à-dire lorsqu'aucun déplacement n'est possible), le joueur (s'il a obtenu un nouvel Highscore) a la possibilité de mémoriser son pseudo. S'il n'entre pas son pseudo (par exemple : en relançant la partie ou en quittant le programme), le pseudo qui sera mis par défaut sera "Anyone".
Le code est commenté de façon à être compréhensible et le nom des variables est en anglais.
Il n'y a aucune fuite de mémoire, mis à part celle de base liée à la SDL et référencée.
Nous avons intégré des options supplémentaires. Le jeu se lance sur un menu qui permet de choisir le dégradé des tuiles du jeu. De plus, nous pouvons égalemment changer la couleur grâce aux touches "F1" "F2" et "F3" durant la partie. Cela n'est pas clairement précisé, car il s'agit en faite d'un Easter Egg.
Nous avons pour finir, intégré une petite animation dans le menu du jeu. Pour illustrer le tout, voici un screenshot du menu :
Annexe 3 - Image menu jeu sdl

5 Intelligence Artificielle

Nous avons implémenté deux Intelligences Artificielles, une qui doit réalisé le meilleur score en moins de 10 secondes et une autre en moins de 2 minutes.

Pour observer nos Intelligence Artificielle à l'oeuvre nous avons conçu deux programmes de visualisation. La première permettant de répeter n fois l'Intelligence Artificielle et une autre permettant de l'observer grâce à un affichage Ncurses, un mouvement est fait à chaque fois que l'on presse la touche "UP".

Les résultats des différentes Intelligence Artificielle sur 100 lancés sont disponibles en annexe 3.
cf : Annexe 4 - Intelligence Artificielle

Le principe de notre fonction d'exploration :
Elle crée un tableau de 4 cases permettant de stocker les scores issus des 4 directions. Si la profondeur est de 0 alors elle renvoie la note de la grille en cours. Tout d'abord, la fonction effectue un mouvement dans une direction à la profondeur n, puis elle génère l'apparition de la tuile 2 sur une case vide de la grille et la fonction se rappelle avec une profondeur de *n-1*. Elle fait de même avec l'apparition d'un 4 sur la même case vide. Elle effectue cette série d'opérations sur toutes les cases vides de la grille courante. Une fois qu'elle a parcouru toutes les cases vides, elle calcule la moyenne des notes obtenues. Elle la stock alors dans la case du tableau correspondant à la direction effectuée. Elle choisie ensuite la meilleure note parmi les 4 directions et la renvoie. En parallèle, elle modifie une variable de direction passée par référence. Le mouvement à effectuer est alors cette variable directionnelle.

5.1 Intelligence Artificielle Rapide

Dans l'Intelligence Artificielle Rapide, nous avons décidé de faire une exploration de 2 et d'étendre à 3 lorsque qu'il ne reste que 0 ou 1 case vide dans la grille.

5.2 Intelligence Artificielle Lente

Dans l'Intelligence Artificielle Lente, nous avons décidé de faire une exploration de profondeur :

- 2 par défaut
- 3 pour lorsque qu'il reste moins de 6 cases vides
- 4 pour lorsque qu'il reste moins de 3 cases vides
- 5 pour lorsque qu'il reste moins de 2 cases vides

5.3 Fonction de notation

Nous avons une fonction de note générale qui utilise une autre fonction.

Fonction `grid_note` :

```
static long int grid_note(grid g){
  long int cpt = 0;
  int cpt_case_empty = 0;
  int cpt_sign_change = 0;
  int max = 0;
  // Parcours de la grille
  for(int y = 0; y < GRID_SIDE; y++){
    for(int x = 0; x < GRID_SIDE; x++){
      cpt += get_tile(g, x, y);
      // Determination de la valeur maximum
      if(get_tile(g, x, y) > max)
                max = get_tile(g, x, y);
      // Determination du nombre de case vide
      if(get_tile(g, x, y) == 0)
                cpt_case_empty++;
      // Monotonicite de la grille
      cpt_sign_change += monotonicity(g, x, y);
    }
    // Bonus coin superieur gauche
    if(get_tile(g, 0, 0) == max)
      cpt += BONUS_CORNER * max;
    // Autres bonus
    cpt += BONUS_HIGH_MAXIMUM * pow(2, max);
    cpt += BONUS_SCORE * grid_score(g);
    cpt += BONUS_CASE_EMPTY * cpt_case_empty;
    // Malus si pas monotone
    cpt -= MALUS_SIGN_CHANGE * cpt_sign_change;

    return cpt;
  }
```

Notre fonction de notation `grid_note` parcours une fois la grille et durant le parcours elle :

— détermine la somme des tuiles de la grille
— Récupère la valeur maximale de la grille
— Compte le nombre de cases vides
— Evalue la monotonicité de la grille grace à la fonction `monotonicity` ci-dessous

Ensuite la fonction attribue des *bonus* et des *malus* en fonction de la disposition de la grille.

Bonus :

— Si le coin en haut à gauche possède la tuile maximale
— Selon le nombre de cases vides (plus il y en a, mieux c'est)
— Selon le score
— Selon la tuile maximale

Malus :

— Selon sa monotonicité

Fonction Monotonicité :

Nous avons besoin d'une fonction qui note la monotonicité de la grille c'est-à-dire nous vérifions que les grilles sont ordonnées de manière à ce que les meilleures soit en haut des colonnes et à gauche des lignes. Elle compte le nombre de cases mal ordonnées.

```
static long int monotonicity(grid g, int x, int y){
  int cpt_sign_change = 0;
  if(x < GRID_SIDE - 1){
    if(get_tile(g, x, y) < get_tile(g, x + 1, y))
      cpt_sign_change += get_tile(g, x + 1, y) - get_tile(g, x, y);
    if(get_tile(g, y, x) < get_tile(g, y, x + 1))
      cpt_sign_change += get_tile(g, y, x + 1) - get_tile(g, y, x);
  }
  return cpt_sign_change;
}
```

	$+\leftarrow$			
$+\uparrow$	64	32	16	8
	32	16	8	4
	16	8	4	2
	8	4	2	0

6 Oragnisation, Problème rencontré et critique

6.1 Répartition du travail

Nous avons tous les quatre travaillé sur l'implémentation de grid.c Nous nous sommes tous entraidés lors de la réalisation de ce projet (comme pour le rapport, la correction finale des fichiers, etc.) Ensuite nous nous sommes répartis le travail comme suit :

Ysabelle Emery :
- Participation à l'implémentation de la Ncurses
- Réalisation des premiers tests (avant la relecture par les autres groupes)
- Correction des erreurs et relecture des fichiers
- Création de toutes les images nécessaires à l'interface graphique (tiles, boutons, sprite de l'animation, etc.)
- Modification de tous les tests afin de les rendre extensible à `GRID_SIDE` (après la relecture)

Jimmy Gouraud :
- Réalisation de l'implémentation de l'interface graphique SDL
- Modification de la SDL afin de la rendre extensible à `GRID_SIDE` (après la relecture)
- Création des commentaires et correction orthographique du projet
- Relecture, mise en page du page et conversion des variables en anglais
- Intégration d'un menu et d'une animation pour l'interface graphique

Yordan Kirov :
- Participation à la création des tests
- Création des commentaires
- Conception de différentes fonctions de notation de grille pour l'Intelligence Artificielle
- Pondération des différentes fonctions de notation de grille
- Travail sur le LaTeX

Sébastien Pouteau :
- Création de l'interface Ncurses et modification afin de la rendre extensible (après la relecture)
- Factorisation de la fonction `do_move` et `can_move` (qui sont les fonctions finales utilisées)
- Conception de tous les Makefiles, de l'arborescence du projet, ainsi que de la création du dépot
- Réalisation des premiers tests (avant la relecture)
- Création de l'étude de profondeur des fonctions de l'Intelligence Artificielle
- Normalisation des fichiers de stratégie

6.2 Problèmes rencontrés

Liste des problèmes rencontrés :
— Synchronisation entre l'équipe
— Utilisation des outils au début (comme svn, gdb, Makefile ..)
— Problème de compréhension du contrat de certaine fonction
— Génération des librairies dynamiques

6.3 Critique des outils

— GDB : Difficulté de prise en main, mais permet un débuggage efficace
— SVN :
  — Problème lors des commit/up sur nos machines
  — Problème de conflit entre les versions
  — Permet une bonne gestion d'un projet
  — Permet d'avoir accés au ancienne version
— Doxygène : Normalisation de tous les contrats des fonctions assez fastidieux, mais produit une documentation complète.
— LaTeX :
  — Difficulté de prise en main, d'utilisation et de compréhension
  — Beau rendu final obtenu
  — Simplification de l'organisation du texte

7 Annexe

7.1 Annexe 1 - Architecture Projet

L'arbre du Projet :

```
Projet-Jeu
     |
     + - README.txt
     + - Makefile
     + - include
     |    |
     |    + - grid.h
     |    + - strategy.h
     |    + - gridSDL.h
     |
     + - src
          |
          + - grid
          |     |
          |     + - Makefile
          |     + - grid.c
          |
          + - test-fonction
          |     |
          |     + - Makefile
          |     + - fonction-test.c
          |     + - main-fonction-test.c
          |
          + - ncurses
          |     |
          |     + - Makefile
          |     + - main-ncurses.c
          |     + - highscore_ncurses.txt
          |
          + - sdl
          |     |
          |     + - Makefile
          |     + - gridSDL.c
          |     + - main-sdl.c
          |     + - arial.ttf
          |     + - leadcoat.ttf
          |     + - animation_penguin
          |     |     + ...
          |     |
          |     + - menu_button
          |     |     + ...
          |     |
          |     + - tiles
          |           + ...
          |
          + - startegy
                |
                + - Makefile
                + - A2_emery_gouraud_kirov_pouteau_fast.c
                + - A2_emery_gouraud_kirov_pouteau_low.c
                + - main-graphique-fast.c
                + - main-repetition-fast.c
                + - main-graphique-low.c
                + - main-repetition-low.c
```

7.2 Annexe 2 - Execution des tests

Résultat de la commande ***make check***.

```
valgrind ./test-fonction
==10886== Memcheck, a memory error detector
==10886== Copyright (C) 2002-2013, and GNU GPL'd,
==10886== by Julian Seward et al.
==10886== Using Valgrind-3.10.0 and LibVEX; rerun with -h for
==10886== copyright info
==10886== Command: ./test-fonction
==10886==
[TEST 1]  - new_grid ..................................... [ SUCCED ]
[TEST 2]  - get_tile ..................................... [ SUCCED ]
[TEST 3]  - set_tile ..................................... [ SUCCED ]
[TEST 4]  - get_score .................................... [ SUCCED ]
[TEST 5]  - copy_grid .................................... [ SUCCED ]
[TEST 6]  - game_over .................................... [ SUCCED ]
[TEST 7]  - do_move_UP ................................... [ SUCCED ]
[TEST 8]  - do_move_DOWN ................................. [ SUCCED ]
[TEST 9]  - do_move_LEFT ................................. [ SUCCED ]
[TEST 10] - do_move_RIGHT ................................ [ SUCCED ]
[TEST 11] - do_move_ALL .................................. [ SUCCED ]
[TEST 12] - test_can_move ................................ [ SUCCED ]
[TEST 13] - test_add_tile ................................ [ SUCCED ]
==10886==
==10886== HEAP SUMMARY:
==10886==     in use at exit: 0 bytes in 0 blocks
==10886==   total heap usage: 18 allocs, 18 frees,
==10886==   336 bytes allocated
==10886==
==10886== All heap blocks were freed -- no leaks are possible
==10886==
==10886== For counts of detected and suppressed errors,
==10886== rerun with: -v
==10886== ERROR SUMMARY: 0 errors from 0 contexts
==10886== (suppressed: 0 from 0)
```

7.3 Annexe 3 - Image menu jeu sdl

Voici un screen shot du menu de l'interface Graphique SDL.

FIGURE 1 – Screenshot du menu sdl - jeu 2048

7.4 Annexe 4 - Intelligence Artificielle

Résultat de l'exécution de l'Intelligence Artificielle Rapide :

```
Sur 1000 lancés :

Nombre de fois 16 : 0
Nombre de fois 32 : 0
Nombre de fois 64 : 0
Nombre de fois 128 : 0
Nombre de fois 256 : 0
Nombre de fois 512 : 9
Nombre de fois 1024 : 81
Nombre de fois 2048 : 585
Nombre de fois 4096 : 322
Nombre de fois 8192 : 3
```

Résultat de l'exécution de l'Intelligence Artificielle Lente :

```
Sur 100 lancés :

Nombre de fois 16 : 0
Nombre de fois 32 : 0
Nombre de fois 64 : 0
Nombre de fois 128 : 0
Nombre de fois 256 : 0
Nombre de fois 512 : 0
Nombre de fois 1024 : 5
Nombre de fois 2048 : 43
Nombre de fois 4096 : 51
Nombre de fois 8192 : 1
```