

Rapport de projet 2048

Daubasse Biffert Gallardo

22 avril 2015

Table des matières

1	Présentation du jeu 2048	1
2	Implémentation de la grille	2
2.1	Fonctions simples	2
2.1.1	Valeur d'une tuile	2
2.1.2	Initialiser une tuile	2
2.1.3	Valeur du score	2
2.2	Structure	2
2.3	Instanciation d'une grille	2
2.4	Suppression d'une grille	3
2.5	Copie d'une grille	3
2.6	Mouvement de la grille	3
2.6.1	Possibilité de mouvement	3
2.6.2	Mouvement	3
2.7	Ajout d'une tuile	4
2.8	Jouer un coup	4
2.9	Fin du jeu	4
3	Test sur grid.c	4
3.1	Fonctions générales	4
3.2	Fonction perso	5
4	Stratégie pour automatiser la résolution du jeu	5

1 Présentation du jeu 2048

Le but du jeu est de fusionner des nombres ensemble (puissances de deux) afin d'atteindre le nombre ultime '2048' et gagner la partie. L'aire du jeu 2048 est une grille de quatre lignes par quatre colonnes avec donc 16 cellules carrées. Chaque cellule peut être vide ou contenir un nombre. Au début du jeu, il y a

deux carrés (également appelés « tuiles ») avec un chiffre '2' ou un '4' suivant votre chance à l'intérieur.

Lorsque vous parvenez à faire entrer en collision 2 briques avec le même numéro dedans, elles fusionnent en une seule nouvelle brique dont le numéro sera l'addition des deux nombres précédents : $2+2=4$, $4+4=8$, ... $1024+1024=2048$!

Pour déplacer les briques sur la grille, vous devez juste choisir une direction (haut, droite, bas ou gauche). Toutes les briques vont se déplacer dans la direction choisie, jusqu'à ce qu'elles fusionnent avec une brique de même valeur ou bien qu'elles soient bloquées par une brique avec un numéro différent. A chaque mouvement une tuile va aléatoirement apparaître dans une case vide, cette tuile a une chance sur dix d'être égale à quatre sinon elle sera égale à deux.

Ici on utilisera simplement les quatre flèches directionnelles du clavier pour déplacer les briques.

2 Implémentation de la grille

2.1 Fonctions simples

2.1.1 Valeur d'une tuile

On donne les coordonnées et la grille afin de directement retourner la valeur de la tuile situé dans cette grille et à ces coordonnées.

2.1.2 Initialiser une tuile

On donne les coordonnées et la grille afin de directement modifier la valeur de la tuile situé dans cette grille et à ces coordonnées.

2.1.3 Valeur du score

On donne la grille et la valeur du pointeur du score de la grille est retourner.

2.2 Structure

Nous avons choisis d'implémenter la grille par un pointeur de pointeur d'entier pour la grille et d'un entier représentant le score. Le pointeur nous sert à créer un tableau deux dimension qui servira de grille, nous avons choisis la solution du pointeur car elle permet de facilement étendre notre grille à une taille supérieure à quatre, le tableau deux dimensions quand à lui nous semblait la solution la plus naturelle pour créer une grille c'est pourquoi nous n'avons pas implémenter la grille avec un tableau une dimension.

2.3 Instanciation d'une grille

Pour l'instanciation, dans la fonction `new_grid`, d'une grille on alloue la mémoire pour une structure de grille puis on initialise le score à zéro et ensuite

on alloue le tableau deux dimensions en initialisant chacune des tuiles a zéro. On retourne ensuite la structure nouvellement créée.

2.4 Suppression d'une grille

Pour supprimer une instance de grille nous prenons en argument une stucture de grille et parcourons le tableau deux dimensions et nous libérons la mémoire allouer pour les pointeurs du pointeur tiles de notre structure puis on libère la mémoire prise par le pointeur tiles lui même et enfin on libère la mémoire de notre instance de grille.

2.5 Copie d'une grille

Pour copier une grille nous prenons en paramètre deux grille, src la source et dst la destination, nous parcourons ensuite le tableau deux dimension afin de copier chaque tuiles de src et de la placer dans la grille de dst une fois cela fait on copie le score de src et on le place dans le score de dst.

2.6 Mouvement de la grille

2.6.1 Possibilité de mouvement

Dans la fonction `can_move` nous prenons en argument une grille et une direction. L'idée ici est de se dire que si une tuile peut bouger alors l'ensemble de la grille le peut c'est pourquoi nous initialisons une variable booléenne a faux qui sera ensuite modifié, au fur et a mesure du parcours de notre tableau, si, en prenant une direction donnée, deux tuiles consécutives sont égales mais différentes de zéro ou si la tuiles vérifié est égale a zéro, c'est à dire que l'on a un trou entre notre tuile et ce qu'il y a après que ce soit une autre tuiles, le bord de la grille ou encore un zéro. On renvoie ensuite notre variable booléenne.

2.6.2 Mouvement

Pour le mouvement nous avons utilisée trois fonctions qui nous sont propre que sont `array_to_grid`, `grid_to_array` et `compute_array`.

- Passage d'un tableau à une grille et d'une grille à un tableau :

Ces fonctions servent à transformer une grille en une ligne (tableau 1D de taille `size`) selon les parametres de position donnés, c'est à dire le paramètre `x` donnés en argument, et sans prendre en compte les zeros ensuite pour repasser d'une grille a un tableau on retransforme donc notre ligne 1D en grille en combatnt les tuiles vides par des zéros. En fonction du sens dans lequel nous voulons faire notre mouvement nous inversons ou non notre tableau, à l'aide de la foncton `invert_array`, afin de traiter toute les directions de la même manière.

- Calcul du score :

La fonction `compute_array` prend un tableau 1D et une taille, initialise un score à zéro et ensuite parcourt ce tableau et pour chaque cases consécutives et

identique du tableau les fusionnent ajoutant leur somme à score qui est ensuite renvoyer à la fin de la fonction.

En utilisant ces fonction il est donc plus facile de faire le `do_move` puisque que nous transformons notre grille en tableau 1D puis calculons notre nouvelle grille et le score dans ce tableau pour enfin remettre ce tableau dans notre grille.

2.7 Ajout d'une tuile

Ici nous créons un tableau deux dimensions et copions les cases vides de notre grille prise en argument pour ensuite choisir hasard parmi ces cases celle que l'on placera à deux (ou quatre suivant le `rand()`). Nous choisisons de faire un `rand` entre 0 et 1000 car en effet le `rand()` rend des valeurs plus stable s'il est pris entre 0 et 1.

2.8 Jouer un coup

Jouer un coup revient à vérifier que l'on peut bouger dans la direction demandée par l'utilisateur et si c'est le cas faire le mouvement avec le `do_move` et ajouter une tuile.

2.9 Fin du jeu

Nous détectons la fin du jeu quand aucune des directions n'est jouable, c'est à dire quand on ne peut plus bouger de tuile.

3 Test sur `grid.c`

3.1 Fonctions générales

Dans `test_grid.c` nous avons essayer de vérifier le plus de code possible automatiquement. Pour nous aider nous avons créé une fonction `result` qui affiche, avec un code couleur, si le test est passé avec succès ou non. D'abord on teste que `new_grid` crée bien une grille vide, on vérifie donc que chaque tuile est nulle. Ensuite on teste que la `copy_grid` copie bien toute la grille c'est à dire le tableau deux dimension doit être le même et le score lui aussi doit être copié. On crée donc deux grilles et vérifions à l'aide de la fonction `equals` du fichier `grid_utilities` que chaque cases du tableau sont les mêmes puis que les scores sont les mêmes. Le test du score se fait juste par l'anticipation du score de la grille, on fixe donc deux cases et on vérifie que le score de la grille est bien le bon. Pour le test du `can_move` nous créons deux grilles chacune avec une case occupée, une dans un coin et l'autre entourée par des cases vides. On vérifie ensuite que la première, celle avec la case dans le coin ne peut bouger que vers les directions opposées au coin quand à la deuxième, celle avec la case entourée de cases vides, doit pouvoir bouger dans tout les sens. Avec la deuxième grille nous testons notre `do_move` en bougeant vers le haut puis vers la gauche et si

on peut bougé dans toute les directions sauf le haut après avoir bougé en haut puis que l'on peut boug   partout sauf vers le haut et la gauche apr  s avoir boug   vers la gauche alors le test est passer avec succ  s. Enfin pour le `add_tile` on fait autant de `add_tile` que de case puis on v  rifie que toute les cases sont occup  es.

3.2 Fonction perso

Afin d  viter tout bug innatendu nous avons d  cider de tester nos fonctions qui se trouvent dans `grid_utilities`. On remplit donc un tableau de z  ro    `GRID_SIZE` (qui est la taille de toute nos grilles) et on l'inverse    l'aide de notre fonction, on v  rifie alors que le tableau est bien invers  . Pour tester la fonction `grid_to_array` on remplit trois cases d'une grille de fa  on    avoir les m  me valeur que le tableau pr  c  dent dans les cases du haut, on applique notre fonction a la grille et on v  rifie que nos tableaux sont identique. Pour le calcul de la nouvelle grille apr  s mouvement on prend notre tableau de grille et l'on v  rifie que ses valeurs sont bien les bonnes, c'est   dire qu'elles doivent   tre rang  e dans l'orde d  croissant. Pour finir la fonction `array_to_grid` on prend notre grille que l'on avait au pr  alable transform   en tableau que l'on retransforme en grille et on v  rifie ensuite que la valeur des tuiles du haut corresponde au tableau qui nous sert de r  f  rence dans les autres fonctions. Si ce sont les m  me la fonction ne contient pas de bug.

4 Strat  gie pour automatiser la r  solution du jeu