

Rapport de projet 2048

Daubasse Biffert Gallardo

22 avril 2015

Table des matières

1	Présentation du jeu 2048	2
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	3
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	4
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	5
3.2	Fonction perso	5
4	Stratégie pour automatiser la résolution du jeu	6
4.1	Stratégie rapide (Moins de dix secondes de temps de calcul) . . .	6
4.2	Stratégie lente (Moins de deux minutes de temps de calcul) . . .	6
5	Affichage	8
5.1	Ncurses	8
5.2	SDL	8

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 deux 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 quatres 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 dimensions qui servira de grille, nous avons choisi 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 à 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 structure de grille et parcourons le tableau deux dimensions et nous libérons la mémoire allouée 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 grilles, `src` la source et `dst` la destination, nous parcourons ensuite le tableau deux dimensions afin de copier chaque tuile 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 à faux qui sera ensuite modifiée, au fur et à mesure du parcours de notre tableau, si, en prenant une direction donnée, deux tuiles consécutives sont égales mais différentes de zéro ou si la tuile vérifiée est égale à 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 tuile, le bord de la grille ou encore un zéro.

On renvoie ensuite notre variable booléenne.

2.6.2 Mouvement

Pour le mouvement nous avons utilisé trois fonctions qui nous sont propres : `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 paramètres de position donnés, c'est à dire le paramètre `x` donné en argument, et sans prendre en compte les zéros ensuite pour repasser d'une grille à un tableau on retransforme donc notre ligne 1D en grille en remplaçant 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 fonction `invert_array`, afin de traiter toutes 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 case consécutive et identique du tableau les fusionne ajoutant leur somme à score qui est ensuite renvoyé à la fin de la fonction.

En utilisant ces fonctions 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 au hasard parmi ces cases celle que l'on placera à deux (ou quatre suivant le `rand()`). Nous choisissons de faire un `rand` entre 0 et 1000 car en effet le `rand()` rend des valeurs plus stables 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 crée une fonction `result` qui affiche, avec un code couleur, si le test est passer 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 etre 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 même puis que les scores sont les même. 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 vide.

On vérifie ensuite que la première, celle avec la case dans le coin ne peut bougée que vers les directions opposées au coin quand à la deuxième, celle avec la case entourée de cases vides, doit pouvoir bougée 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 bougée partout sauf vers le haut et la gauche après avoir bougé vers la gauche alors le test est passer avec succès.

Enfin pour le `add_tile` on fait autant de `add_tile` que de case puis on vérifie que toute les cases sont occupées.

3.2 Fonction perso

Afin d’éviter tout bug innatendu nous avons décider de tester nos fonctions qui se trouvent dans `grid_utilities`.

On remplit donc un tableau de zéro à `GRID_SIZE` (qui est la taille de toute nos grilles) et on l’inverse à l’aide de notre fonction, on vérifie alors que le tableau est bien inversé.

Pour tester la fonction `grid_to_array` on remplit trois cases d’une grille de façon à avoir les même valeur que le tableau précédent dans les cases du haut, on applique notre fonction a la grille et on vérifie que nos tableaux sont identique.

Pour le calcul de la nouvelle grille après mouvement on prend notre tableau de grille et l’on vérifie que ses valeurs sont bien les bonnes, c’està dire qu’elles doivent être rangée dans l’orde décroissant.

Pour finir la fonction `array_to_grid` on prend notre grille que l’on avait au préalable transformé en tableau que l’on retransforme en grille et on vérifie ensuite que la valeur des tuiles du haut corresponde au tableau qui nous sert de référence dans les autres fonctions. Si ce sont les même la fonction ne contient pas de bug.

4 Stratégie pour automatiser la résolution du jeu

4.1 Stratégie rapide (Moins de dix secondes de temps de calcul)

Au premier abord la stratégie qui semble la plus simple est celle dite du coin qui consiste à choisir une direction, si le mouvement est impossible changer de direction puis retourner à la première, si les deux premières sont bloquées passer à une troisième et de même pour la quatrième direction.

Cette solution est très simple, ne demande aucun calcul et demande vraiment peu de temps à coder évidemment elle est aussi très peu fiable.

Nous avons donc opté pour une stratégie qui commence comme celle du coin mais qui quand le choix de la troisième direction arrive choisit à l'aide de la fonction score la direction à prendre. La fonction score attribut à une grille un score rapidement calculé puisqu'il suffit d'enchaîner huit parcours de grille.

En effet à chaque parcours on pondère les valeurs de la grille ce qui nous aide à remplir un tableau dont on prendra l'indice de la valeur maximale.

4.2 Stratégie lente (Moins de deux minutes de temps de calcul)

Pour la stratégie longue nous avons opté pour une fonction récursive.

Nous avons à la base pensé à implémenter une structure d'arbre mais cela était trop fastidieux et au final n'avait pas beaucoup moins de calcul à réaliser comparé à notre fonction récursive.

Nous avons donc eu l'idée de calculer chaque sous grille, c'est à dire les différentes grilles possibles en fonction des coups et des apparitions aléatoires des tuiles, de notre grille courante afin de choisir, parmi les quatre directions, la direction qui serait susceptible de donner la meilleure grille.

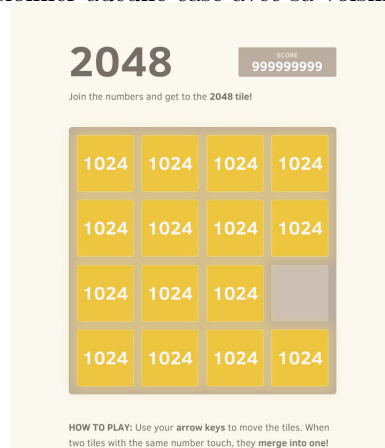
Pour choisir la meilleure grille nous attribuons encore une fois un score à ces grilles avec ces objectifs :

- Créer une grille progressive : plus précisément, il faut faire en sorte que la valeur des cases augmente ou descende quelle que soit la direction. Ainsi, 2 - 4 - 8 - 16 est acceptable, tout comme 32 - 8 - 4 - 2. Mais 2 - 8 - 2 - 16 ne l'est pas (à cause de la 3ème case).

8	32	64	512
4	8	16	256
2	4	8	32
		4	8

Chaque mouvement doit donc être choisi en fonction du résultat qu'il va produire : la grille sera-t-elle plus progressive après le mouvement qu'avant ? Si oui, on s'oriente sur un bon score. Si non, le score sera mauvais. L'idée ici est qu'il faut placer les valeurs les plus hautes sur les extérieurs pour éviter qu'une valeur faible se retrouve coincée entre deux valeurs hautes et ne puisse être fusionnée.

- Gérer la régularité des cellules : pour pouvoir fusionner, les cases doivent comporter des valeurs identiques. Ainsi une suite 2 - 16 - 64 - 256 respecte la première règle (progressivité) mais aboutira à un échec car on ne pourra fusionner aucune case avec sa voisine.



Il faut donc respecter un second critère : la régularité. Et faire en sorte de ne pas avoir de rupture dans les séries. Entre créer une série 2 - 4 - 8 - 16 et créer une série 16 - 64 - 256 - 1024 on préférera donc la première.

- Maximiser le nombre de cases libres sur le plateau : Si toutes les cases sont remplies, le jeu se termine sur une défaite, il faut donc faire en sorte d'avoir le plus de cases libres et privilégier les mouvements qui libèrent des cases.

Entre deux mouvements, l'un libérant une case et l'autre n'en libérant pas, on donnera un meilleur score au premier.

5 Affichage

5.1 Ncurses

5.2 SDL