Prép'ISIMA

Programmation avancée en C

2048

Projet final en C

Clément Payard
Mathieu Laurençot

Encadrant : M. Vincent Limouzi

19/11/2021

Table des matières

1 Problèmes potentiels identifiés au départ

Ce jeu à en effet quelques spécificités :

1.1 Gestion des fusions

Tout d'abord, le jeu peut fusionner plusieurs cases en même temps, sur différentes lignes et colonnes. De plus, sur ces mêmes lignes et colonnes comme, par exemple, avec cet exemple :

2	2	4	4
4	8	2	2
0	0	0	0
0	0	0	0

Si le joueur fusionne à gauche, nous devrons fusionner les 2 ET les 4 présents sur la première ligne.

De plus, lorsque l'on doit fusionner plusieurs cases, nous devons mettre les résultats de la fusion "collés" dans la direction voulue. Nous devons donc gérer ce cas en plusieurs étapes.

1.2 Différentes taille

Nous souhaitons faire plusieurs tailles de 2048, nous voulons donc avoir la possibilité de changer la taille de la grille, en laissant le choix à l'utilisateur de choisir, pourquoi pas, une taille 100x100 ou une taille 2x5.

1.3 Gestion des sauvegardes

Nous voulons que notre joueur puisse faire une pause. Nous nous imposons donc de faire un système de sauvegarde rapide et efficace.

2 Présentation du programme console

Nous ne mettrons pas l'intégralité du code des différentes résolutions, mais il est consultable dans les fichiers joints.

2.1 Introduction à la structure Terrain/ter

Nous avons choisis d'utiliser une structure pour pouvoir manipuler notre grille aisément, cette structure s'appelant *Terrain*.

```
  typedef struct Terrain{
  int **tab;
  int max;
  int tailleX;
  int tailleY;
  int vide;
  int score;
}ter;
```

Elle est composé de divers éléments, mais seulement d'entiers :

1. Un tableau en 2 dimensions, qui serra la grille en elle même.
2. max, qui sera l'entier présent dans la grille maximal. Permet de gérer le cas de "victoire" si l'on fait 2048.
3. tailleX, qui sera le nombre de colonne
4. tailleY, qui sera le nombre de ligne

5. vide, qui sera le nombre de case vide du tableau (utile pour plus tard)
6. score, qui sera le score du Terrain en cours

Toutes ces éléments sont présents pour simplifier le nombre de paramètre pris par les divers fonctions suivantes. Dans la suite de ce document, et ainsi que dans le code, *Terrain* sera abrégé en *ter*

2.2 Présentation des différentes fonctions avec leurs signatures

1. InitVide

   ```
   ter InitVide(int n, int y);
   ```

   Cette fonction prend deux entiers et renvoie un ter de la taille de n et de y. Elle nous permet de créer notre grille proprement.

2. afficheTer

   ```
   void afficheTer(ter T);
   ```

   afficheTer nous permet, comme son nom l'indique, d'afficher le ter donné en paramètre.

3. SetRandomCase

   ```
   int SetRandomCase(ter *T, int n);
   ```

   SetRandomCase permet d'initialiser une case `vide` à une certaine valeur. Elle prend en paramètre un entier, qui détermine la valeur de la case (2 ou 4).
   Pour placer la case, nous prenons un nombre aléatoire dans le nombre de case vide(qu'on a décrit dans la structure *Terrain*). Puis, nous parcourons le tableau pour trouver l'emplacement de cette case. Petit particularité : le nombre de case x parcouru ne s'incrémente pas si la case est déjà rempli. C'est donc après x case vide que l'on initialise la case numéro x + **nombre de case déjà remplie** à une valeur 2 ou 4.

4. CaseMouve

   ```
   void CaseMouve(ter *T, int depNum);
   ```

   Cette fonction est sûrement une des plus importante. En effet, elle prend en paramètre un pointeur sur un terrain ainsi qu'un déplacement. Par manque de place sur ce rapport, nous ne pouvons la décrire entièrement. En revanche, le code de cette fonction est particulièrement bien détaillé et est disponible dans Fonction.c.
   Pour expliquer simplement le fonctionnement de cette fonction, nous utilisons une double boucle *for* qui permet de parcourir l'ensemble de la grille dans différentes direction en fonction de *depNum*, qui est la direction choisie. Dans le cas d'une fusion, on vient stocker la position de la case fusionnée afin de ne pas la refusionner (voir gestion des problèmes au début du document). \épend de la fonction CanDep pour vérifier si le joueur peut se déplacer ou à perdu.

5. CanDep

   ```
   int CanDep(ter T, int depNum);
   ```

   Également essentielle, cette fonction permet de savoir si un déplacement dans la direction choisi est possible sur le ter passé en paramètre.
   Nous avons géré le game over comme ceci (dans le main.c) :

   ```
   for(int i = 1;i < 5; i++){
     if(CanDep(plateau, i)){
       // si le joueur peu bouger on change cette variable
       GameOver = 0;
       break;
     }
    }
   ```

   Nous testons donc tous les coup grâce à cette fonction à chaque fois après le coup du joueur. Si le jeu détecte que le joueur pourra faire un déplacement, alors *CanDep* retournera 1, ce qui mettra la variable *GameOver* à 0, le joueur n'aura donc pas perdu. Dans le cas contraire, *GameOver* restera à 1 et le joueur aura donc perdu, car il ne pourra pas faire de mouvement au prochain coup.

6. LibereTer

   ```
   void LibereTer(ter *T);
   ```

   Comme son nom l'indique, cette fonction permet simplement de libérer un *ter* proprement.

7. ReadEnCoursSave

   ```
   ter ReadEnCoursSave(float time, char *name);
   ```

   Grâce à cette fonction permet, nous pouvons lire le fichier de sauvegarde. Plus de détail dans la suite du document (ici).

8. copiFile

   ```
   void copiFile(char *old, char *nouv);
   ```

   Cette fonction permet de prendre deux chaînes de caractères qui sont les noms de fichiers, et qui copie le contenu de *old* dans *nouv*, qui seront les noms des fichiers.

9. writesaveFile

   ```
   void writesaveFile(int hightscore, int endedGamesWin, int endedLoseGame, char mo
   ```

   Enfin, cette dernière fonction nous permet de sauvegarder les options / information du joueur dans un fichier que nous pouvons récupérer, en prenant en paramètres divers éléments. Plus de détail dans la suite de ce document (ici).

2.3 Explication du fonctionnement du jeu de base

Les fonctions étant définis dans *FonctionJeu.c*, le jeu en lui même est entièrement présent dans le *main.c*.

Notre programme est basé sur un switch, qui nous laisse 6 choix (dont 1 de fermeture) :

1. Le premier cas est lors de l'arrivé dans le jeu. Il renvoie systématiquement à :
2. Notre menu, qui est le cas numéro 2. C'est ici que le joueur décide si il veut lancer une nouvelle partie, voir un replay, etc. On peut distinguer 4 cas majeurs :
   — Lancement d'un jeu
   — Voir un replay
   — Modifier les options
   — Quitter le jeu
3. Subséquemment, le cas 3 nous permet de gérer les déplacement du joueur ainsi que l'affichage du jeu.
4. Ensuite, le quatrième cas permet de gérer lorsque que le joueur à fini la partie. Affiche si il a gagné ou perdu (score inférieur/supérieur à 2048), ainsi que son meilleur score.
5. Le cas 5 est le cas où l'initialisation du terrain s'est mal passé. On va alors au cas -1.
6. Le cas 6 est dédié à la modification des paramètres du jeu (tel que les touches par exemple).
7. le septième cas, nous affichons la fenêtre graphique.
8. Enfin, le cas -1 est réservé pour arrêter le jeu et le fermer proprement.

2.4 Présentation des nouvelles fonctionnalités

1. Gestion des sauvegardes

   (a) Présentation du fichier de sauvegarde

   Un fichier de sauvegarde est présenté de cette manière :

   Unelettre Unchiffre Unautrechiffre Unelettre Unchiffre Unautrechiffre Unelettre Unchiffre Unautrechiffre Unelettre Unchiffre Unautrechiffre [...]

   Un exemple concret :

   N 4 2
   n 2 3 n 2 4 h 2 3 b 2 2 g 2 4 g

   Tout d'abord, les trois premiers éléments :

   i. Sert à déterminer le "type" de la sauvegarde (une seule présente lors du rendu).

   ii. Les deux nombres suivant permettent d'avoir la taille du terrain.

   Puis, les autres sont simplement la partie enregistrer. Comment me direz vous ? Et bien, la lettre donne la direction (g gauche, d droite, etc, n rien (c'est le moment où l'on ajoute à une case vide le chiffe 2 ou 4)), le deuxième nombre donne la valeur obtenue lors de la génération aléatoire, et le dernier donne l'emplacement où ce nombre est apparu en suivant le principe décrit dans SetRandomCase.

(b) La sauvegarde en elle-même

La gestion des sauvegardes est particulière. En effet, lorsque le joueur joue un coup, nous enregistrons, selon la méthode décrite ici, les différents éléments nécessaires à la sauvegarde. C'est-à-dire qu'à chaque fois que le joueur effectue un coup, nous récupérons les éléments nécessaires pour pouvoir simuler sa partie pour plus tard.

(c) Récupération de la sauvegarde (avec ReadEnCoursSave)

La sauvegarde est récupérée en deux étapes :

i. Lecture du fichier

Pour lire le fichier de sauvegarde, nous avons besoin de la fonction ReadEnCoursSave. En effet, cette fonction permet de lire les informations présentes dans le fichier de sauvegarde donné

```
FILE *save = fopen(SAVE_NAME, "r");
```

où $\text{SAVE}_{\text{NAME}}$ est une macro qui est le nom du fichier de la sauvegarde (non modifiable par le joueur).

De plus, cette fonction permet de recréer exactement le même terrain de la sauvegarde avec, on l'avoue, un petit subterfuge :

ii. Replay automatique

Lors de l'appel de la fonction, nous créons d'abord le terrain avec le tableau de la bonne dimension. Nous lisons donc la première ligne du fichier des options. Puis, nous parcourons la deuxième lignes. Nous stockons dans une chaîne de caractère un seul et unique coup. Nous avons donc besoin de stocker une lettre et 2 nombre. Le fichier de sauvegarde ressemblant à ceci :

```
N unnombre unautre
n 2 31 n 2 4 h 2 3 b 2 22 g 2 4 g
```

nous devons compter le nombre d'espace, et non le nombre de caractère (ici nous avons 31 et 22 qui ne marcherait pas) pour séparer deux coups. Quand ce nombre d'espace atteint 3, nous somme donc en présence du coup suivant.

Enfin, pour jouer le replay, nous avons juste à "relancer" le jeu, mais au lieu de demander au joueur de jouer, nous mettons les différents coups récupérer précédemment pour faire le fameux replay. De plus, pour continuer une partie, nous ne faisons que faire un replay instantané de la partie précédemment chargée.

2. Fichier paramètre

Le fichier paramètre s'appelle Parametres, et posséde plusieurs éléments :

```
High score
Nombre de victoire
Nombre de défaite
Les nouvelles touches  x  4
Nombre de replay (pour le nom du fichier des replays au fur et à mesure)
La taille de l'écran graphique en X
La taille de l'écran graphique en Y
Fullscreen (0 = non et 1 = oui)
Volume du son lors des mouvements
Volume de la musique
Variable qui détermine la vitesse de jeu (TIME dans les paramètres SDL)
```

3. Possibilité de mapper les touches de déplacement

Comme on peut le voir dans la partie précédente, le joueur peut lier ses touches de déplacements avec d'autres touches du clavier.

3 La partie SDL

Le code qui nous permet de gérer la fenêtre SDL est disponible ici, et ne sera pas entièrement expliqué (limité par les 5 pages).

3.1 Début

Pour démarrer SDL, nous devons initialiser de nombreuses variables, comme par exemple :

— La variable *etapeactuelledujeu*, qui nous permet de fermer SDL si elle est égale à 0,
— *TailleX et TailleY*, qui vont nous permettre de gérer la taille de l'écran,
— des variables permettant de garder un nombre d'image par seconde (fps) constant et agréable
— des variables permettant de détecter où le curseur de la souris se trouve sur l'écran
— les varibles permettant de dessiner le graphique
— etc.

Tout ceci est stocké dans une structure de type *screen*.

3.2 Affichage et fonctionnalité

1. Affichage
   Pour effectuer l'affichage d'une fenêtre SDL, nous devons passer par une boucle *while*.
   Puis, nous distinguerons trois cas grâce à un *if* (et *else if*).
   (a) Dans le premier cas, SDL dessinera l'écran, s'il n'a pas été dessiné depuis un certain temps
   (b) Sinon, nous vérifierons également si les courbes sont en adéquation avec les polynômes. Si ce n'est pas le cas, nous entrons alors dans le *else if* qui va nous permettre d'écraser l'image précédente. Enfin,
   (c) si nous passons les deux conditions précédentes, nous devons **absolument** endormir le Central Processing Unit (CPU). Cela nous permet de ne pas utiliser tout le processeur de l'ordinateur.

   Puis, nous avons aussi un cas de débogage. En effet, si l'on n'est pas entré dans le while depuis une seconde ou plus, il peut y avoir un problème. On recommence alors une seconde "propre", en mettant certaines variables à 0.
2. Les différentes fonctionnalités présentes
   (a) Le bouton console
   Lorsque vous cliquez sur le bouton en bas à gauche, le jeu bascule avec l'interface console, ce qui permet au joueur de "choisir" s'il veut jouer avec SDL ou non.
   (b) F11
   La touche F11 nous permet de mettre en plein écran, et d'avoir ainsi un petit effet sympas !
   (c) Taille de la fenêtre
   La taille de la fenêtre est complètement géré par les variables *TailleX et TailleY* qui, lorsque l'utilisateur change la taille de la fenêtre, se mettent à jour en temps réel.
   (d) Animations des cases
   Pour faire cette animation, nous récupérons dans chaque cas de bloc le nombre de déplacement qu'il doit faire, puis, nous divisons ce déplacement par le temps choisi par l'utilisateur.
   (e) Animation menu
   Toutes les 0.15 seconde, on actualise toutes les images des boutons. Pour cela, il y a plusieurs cas :
      i. Si l'utilisateur à la souris sur le bouton, nous affichons les images correspondant à la descente de 3 blocs
      ii. Sinon, soit nous terminons le mouvement d'un 2 en cours, soit, si aucun déplacement n'est en cours, il y a une probabilité de 1/30 de faire commencer un mouvement de deux sur le côté.
   (f) Les barres dans les options
   Pour ceci, nous avons utilisé une petite astuce : la "barre" est en fait un bouton, qui, lorsque l'utilisateur clique dessus, enverra la position de la souris au programme. Suite à cela, la valeur de la variable correspondante est calculée avec un produit en croix. Enfin, nous modifions la position du rectangle qui joue le rôle de curseur sur l'écran.

3.3 Fin de SDL

La fonction *end-sdl* nous permet de fermet la fenêtre SDL proprement, ainsi que faire les opérations nécessaires pour vider la mémoire qui a besoin d'être libéré.

4 Conclusion

En guise de conclusion, nous vous souhaitons de bien vous amusez avec notre 2048 ◡̈.