Prép'ISIMA

Programmation avancée en C

Approximation

Avec les méthodes des droites de regression ainsi que les ajustements

Clément Payard
Mathieu Laurençot

Encadrant : M. Suzanne ÉLODIE

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 deux 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.

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 notre Terrain pour pouvoir manipuler notre grille aisément.

```
  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
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.

2.2 Présentation des différentes fonctions avec leurs signatures

2.2.1 InitVide

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

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

2.2.2 afficheTer

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

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

2.2.3 SetRandomCase

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

SetRandomCase permet d'initialiser une case `vide` à une certain valeur. Elle prend en paramètre un entier, qui détermine l'emplacement de la case. Petit particularité : le nombre de case *n* ne s'incrémente pas si la case est déjà rempli. C'est donc après *n* case vide que l'on initialise la case numéro *n* + **nombre de case déjà remplie** à une valeur 2 ou 4.

2.2.4 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. Dépend de la fonction CanDep pour vérifier si le joueur peut se déplacer ou à perdu.

2.2.5 TODO CanDep

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

Également essentielle, cette fonction permet de savoir si le

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;
      }
    }
```

2.2.6 LibereTer

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

2.2.7 ReadEnCoursSave

2.2.8 copiFile

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

Cette fonction permet de prendre deux chaînes de caractères qui sont les noms de fichiers, et qui copi *old* dans *nouv*.

2.2.9 writesaveFile

2.3 Explication du fonctionnement du jeu de base

2.4 Présentation des nouvelles fonctionnalités

2.4.1 Possibilité de mapper les touches de déplacement

2.4.2 Fichier paramètre

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

usleep((int)(1000000*time)); // on effectue le deplacement deja effectue par le joueur CaseMouve(&T, depNum);

stocker le nombre de case vide, et

un fichier moddifiable $\text{Param}_{\text{Name}}$

High score Nb de victoire Nb de défaite les nouvelles touches*4 nb de replay (pour le nom du fichier des replays au fur et à mesure)

2.4.3 Gestion des sauvegardes

1. Présentation du fichier de sauvegarde
   Un fichier de sauvegarde est présenter 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 :
   (a) Sert à déterminer le "type" de la sauvegarde.
   (b) Les deux nombres suivant permette 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, 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 suivant.
2. 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 cette partie.
3. Récupération de la sauvegarde (avec ReadEnCoursSave)
   La sauvegarde est récupérée en deux étapes :
   (a) 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 un exactement le même terrain de la sauvegarde avec, on l'avoue, un petit subterfuge :
   (b) 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. Puis, nous parcourons la deuxième. 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 coup. Quand ce nombre d'espace atteint 3, nous somme donc en présence du coup suivant.

2.4.4 Replays

3 SDL

Le code qui nous permet de gérer la fenêtre SDL est disponible ici.

3.1 Début

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

— La variable *Stape*, qui nous permet de fermer SDL si elle est égale à 0,
— *size*, qui va 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.

3.2 Affichage et fonctionnalité

3.2.1 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*).

1. Dans le premier cas, SDL dessinera l'écran, s'il n'a pas été dessiné depuis un certain temps
2. 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,
3. 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.

3.2.2 Fonctionnalités :

Divers fonctionnalités sont présentes : Vous pouvez afficher la fenêtre grâce à la touche "g". Vous pouvez désormais voir la liste des points, les courbes représentant les différentes méthodes d'approximation, ainsi que la liste de points à droite.

De plus, si jamais vous voulez rajouter des points à la liste, cette fonctionnalité est disponible grâce au bouton gauche de la souris. Le bouton droit aura pour effet de supprimer le point sélectionné.

Le curseur aura alors une position (x et y) qui sera automatiquement ajouté dans la liste des points. Les courbes ainsi que les polynômes vont s'adapter automatiquement !

D'autre part, vous pouvez zoomer et dézoomer grâce à la molette de la souris.

Enfin, vous pouvez activer ou désactiver les différentes courbes des fonctions en appuyant sur leur nom.

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

Maintenant, on vous souhaite bonne chance.