



PROGRAMMATION AVANCÉE EN C

---

## Approximation

Avec les méthodes des droites de regression ainsi que les ajustements

---



Clément PAYARD  
Mathieu LAURENÇOT

*Encadrant* : M. SUZANNE ÉLODIE

# Table des matières

<b>1</b>	<b>Problèmes potentiels identifiés au départ</b>	<b>2</b>
1.1	Gestion des fusions . . . . .	2
1.2	Différentes taille . . . . .	2
1.3	Gestion des sauvegardes . . . . .	2
<b>2</b>	<b>Présentation du programme console</b>	<b>2</b>
2.1	Introduction à la structure Terrain/ter . . . . .	2
2.2	Présentation des différentes fonctions avec leurs signatures . . . . .	3
2.3	<b>TODO</b> Explication du fonctionnement du jeu de base . . . . .	4
2.4	Présentation des nouvelles fonctionnalités . . . . .	4
<b>3</b>	<b>TODO La partie SDL</b>	<b>5</b>
3.1	Début . . . . .	5
3.2	Affichage et fonctionnalité . . . . .	5
3.3	Fin de SDL . . . . .	6
<b>4</b>	<b>Conclusion</b>	<b>6</b>

# 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 devons 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 sera 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

### 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. 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 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.

### 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.

### 5. 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;
    }
}
```

Nous testons donc tous les coup grâce à cette fonction à chaque fois après le coup du joueur. Si le joueur peut faire un déplacement, alors *CanDep* retournera 1, ce qui mettra la variable *GameOver* à 0, le joueur n'a pas perdu. Dans le cas contraire, *GameOver* restera à 1 et le joueur aura donc perdu.

### 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 la partie dans un fichier de sauvegarde, en prenant paramètres nécessitant une sauvegarde pour recharger la partie. Plus de détail dans la suite de ce document (ici)

## 2.3 TODO Explication du fonctionnement du jeu de base

Les fonctions étant définies 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'arrivée 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 3 cas majeurs :
  - Lancement d'un jeu
  - Voir un replay
  - Quitter le jeu
3. Subséquemment, le cas 3 nous permet de gérer le déplacement du joueur ainsi que l'affichage du jeu
4. Ensuite, le quatrième cas permet de gérer lorsque que le joueur a fini la partie. Affiche si il a gagné ou perdu (score  $\leq$  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. Enfin, le cas -1, qui 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 : Une lettre Un chiffre Un autre chiffre Une lettre Un chiffre Un autre chiffre Une lettre Un chiffre Un autre chiffre Une lettre Un chiffre Un autre chiffre [...]

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

Puis, les autres sont simplement la partie enregistrée. 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 chiffre 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.

#### (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 cette partie.

#### (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 `SAVE_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. 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 nombres. 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 sommes donc en présence du coup suivant.

2. Possibilité de mapper les touches de déplacement

3. Fichier paramètre

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

`usleep((int)(1000000*time));` // on effectue le déplacement déjà effectué par le joueur CaseMouve(&T, depNum);  
stocker le nombre de case vide, et

un fichier modifiable Param<sub>Name</sub>

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)

4. Replays

### 3 TODO La partie 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 variables permettant de dessiner le graphique
- etc.

#### 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'effacer 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. 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 fermer 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.