MOSSER Hugo
RODRIGUES Samuel
MEUNIER William
PAESA Théo

Algorithmique et programmation procédurale

Projet de fin d'année
2048

14 juin 2020

École Internationale des Sciences du Traitement de l'Information

Table des matières

1 Introduction

Pour ce projet nous avons choisi le jeu de 2048.

Le 2048 est un jeu aux règles simples et qui est devenu très populaire, il y a quelques années. Notre but est donc de reproduire le fonctionnement de ce jeu par nos propres moyens. De plus nous avons aussi décidé de créer une interface graphique permettant de mieux appréhender le jeu.

2 Description du programme

2.1 Le jeu

Le jeu de 2048 est un jeu simple où il faut fusionner les cases d'un plateau carré de taille variable. Ces cases sont des puissances de 2 jusqu'à arriver au nombre 2048.

Pour fusionner les cases, il y a certaines contraintes :

- Les cases se déplacent toutes en même temps dans la même direction jusqu'à rencontrer une autre case ou le bord du plateau.
- Si une case rencontre une autre case de valeur identique sur son chemin , ces deux fusionnent
- Les cases ne peuvent être déplacées que dans quatre directions différentes. Haut, Bas, Gauche, Droite
- Le jeu s'arrête lorsque le plateau est plein et que la fusion n'est plus possible.
- Le joueur gagne lorsque l'une des cases atteint la valeur de 2048. On peut lui proposer de continuer la partie s'il veut atteindre des valeurs supérieures.

2.2 Sauvegarde de la partie

Nous avons décidé d'implémenter la fonction permettant de sauvegarder et de charger la partie. Pour cela nous devrons sauvegarder l'état du jeu dans un fichier et être capable de le lire et de recréer un plateau de la bonne taille à partir des données de ce fichier

2.3 Interface

2.3.1 Affichage

Dans un premier temps, à des fins de test nous créerons un affiche simple dans un terminal affichant juste le plateau. Par la suite une fois le jeu fonctionnel, nous pourrons créer un affichage dans une fenêtre grâce à la bibliothèque SDL.

Il nous faudra donc un moyen de lire le plateau et de l'afficher case par case de manière efficace et au bon endroit dans la fenêtre. De plus, nous comptons intégrer des menus permettant au lancement de choisir la taille du plateau ou bien lorsque l'on quitte le jeu de demander au joueur s'il veut sauvegarder la partie. Nous souhaiterions aussi afficher un score de la partie courante ainsi que le temps écoulé depuis le début.

2.3.2 Contrôle du jeu

Pour ce qui est de contrôler le jeu dans un premier temps, nous allons nous contenter d'entrer la direction de déplacement dans le terminal. Mais une fois que nous commencerons l'affichage avec SDL nous utiliserons les fonctionnalités fournies par la bibliothèque pour pouvoir contrôler le jeu grâce au clavier.

3 Répartition des taches

- L'affichage du plateau de jeu a été réalisé par tout les membres du groupe.
- Les déplacements ont été codé par Théo et Hugo.
- Samuel s'est occupé de la sauvegarde et du chargement des parties.
- Hugo et Théo ont fait la partie changement d'état (Menu) en cas de victoire/défaite ou de sauvegarde.
- Les conditions de fin de partie et de victoire ont été travaillé par William.
- La partie esthétique de l'affichage (couleurs et forme des cases) a été fait par Hugo et la partie pour géré les images a afficher par Samuel.
- L'affichage et le rendu a était fait en majorité par Hugo et le reste par Samuel.
- Les interactions souris et flèche clavier (evenementSDL) a été codé par Théo.
- William s'est occupé de l'initialisation du plateau et de l'affichage (Testé des erreurs d'initialisations) en général.

4 Implémentions et structure du programme

4.1 2048

La première chose à faire est d'implémenter le fonctionnement du jeu de 2048.

4.1.1 Le plateau de jeu

1re étape : la création du plateau de jeu qui contiendra les valeurs de chaque case. Nous pourrons ainsi stocker l'état du jeu à chaque instant et le modifier facilement ultérieurement. Nous avons décidé de construire le plateau grâce à un tableau dynamique de deux dimensions. Cela nous permettra de pouvoir se repérer facilement sur le jeu et de plus, pouvoir modifier la taille du plateau.

```
typedef struct
{
    /*! Tableau à deux dimensions d'entier
        non signé */
    uint **tab;
    /*! taille du plateau */
    int taille;
} plateau;
```

- **tab** : Un tableau a deux dimensions d'entier non signés. Les entiers non signés ont été choisi pour être sûr que la valeur de la case soit toujours positive.
- **taille** : Taille du plateau.

En début de partie et à chaque coup, il nous faut créer une case de valeur aléatoire qui sera soit 2 soit 4. Pour cela, il nous a fallu créer la fonction ajouteCase choisissant une case vide du plateau de manière aléatoire et à l'intérieur de celle-ci y mettre le nombre 2 ou 4.

Pour cela dans un premier temps on liste les cases vides du plateau dans un tableau ou on stock leurs coordonnées. Ensuite on choisie un indice aléatoire dans le tableau qui nous donnera les coordonnées de la case où placer la nouvelle case dans le plateau de jeu. Finalement on choisie aléatoirement entre 2 et 4 la valeur de la case à placer.

4.2 Déplacement

Pour la deuxième étape, il nous faut définir les déplacements du plateau (car ce ne sont pas les cases qui bougent mais bel et bien le plateau qui fera transiter les tuiles de score). Pour ce faire, il a fallu créer 4 fonctions pour chacune des 4 directions (respectivement appelées nord, sud, est et ouest) sur le même schéma. On vérifie d'abord si le plateau est vide puis on associe une variable à la coordonnée 'y' horizontal (pour le nord et le sud et 'x' pour l'est et l'ouest par exemple). Si la coordonnée associée est définie sur le plateau et que la parcelle est vide alors on définie la nouvelle position des anciennes cordonnées sur la nouvelle parcelle (soit un $+1$(x) pour aller d'une case à gauche ou -1(y) pour descendre d'une case en bas). De plus, dans 2048, quand on se déplace vers un coté, on ne se déplace pas "seulement" d'une case mais on fait basculer toutes les tuiles vers le coté souhaité. Ainsi, il nous a fallu créer une boucle qui puisse (sur un déplacement souhaité) faire déplacer toutes les cases vers l'angle souhaité d'une traite. Cela nous offre au final, la possibilité de faire déplacer toutes nos cases(tuiles) vers le bord désiré comme dans le vrai jeu.

```
 /* Exemple de déplacement pour la fonction Sud */
int y = int_y; //variable de position
/* tant que la position existe et qu'on ne sort pas du plateau on procède */
while ((y - 1 >= 0) && (jeu->plateau->tab[y - 1][int_x] == 0))
{
    /* On attribue une nouvelle position à la case voulant être déplacé */
    jeu->plateau->tab[y - 1][int_x] = jeu->plateau->tab[y][int_x];
    /* l'ancienne valeur de la case (déplacé) devient nulle ( elle est donc libre si rien
         ne colle l'ancienne case ) */
    jeu->plateau->tab[y][int_x] = 0;
    /* On descend d'une case */
    y--;
}
```

- **plateau** : Pointeur vers la structure contenant le tableau
- **x** : coordonnées horizontales.
- **y** : coordonnées verticales .

4.3 Conditions de victoires

Pour qu'une partie se finisse, il fallait que toutes les cases du tableau soient remplies et que plus aucun mouvement ne soit possible. Dans le cas contraire, s'il y a possibilité de déplacer les tuiles, la partie continue. L'objectif étant d'atteindre une tuile d'une valeur de 2048. Mais le joueur peut quitter le jeu quand il veut, ainsi les conditions de fin de partie sont une défaite (plus de mouvements possible) ou un arrêt du jeu par le joueur.

4.4 Affichage

Le fait d'utiliser SDL comme bibliothèque pour créer un affichage nous a poussé a repenser la manière dont l'état du jeu est stocké. En effet, il nous faut maintenant aussi se souvenir du menu dans lequel on se trouve (Accueil du jeu, Plateau de Jeu, Pause, Fin), Mais aussi nous voulons garder une trace du nombre de coups joué. On a donc crée la structure suivante pour ce souvenir de tout ça.

```
typedef struct
{
        /*! pointeur vers plateau  */
        plateau *plateau;
        /*! compteur du nombre de coups */
        uint nbCoups;
        /*! Etat (ou menu) dans lequel ce
            trouve le jeu */
        uint etatJeu;
} etatJeu;
```

- **plateau** : Pointeur vers la structure contenant le tableau
- **nbCoups** : Nombre d'action réaliser depuis le lancement de la partie.
- **etatJeu** : Nombre representant l'etat du jeu (le menu dans lequel il doit ce trouver)

4.4.1 Plateau

Maintenant que l'on dispose d'un moyen d'accéder à toutes les informations nécessaire à l'affichage du jeu nous pouvons afficher le plateau.

Nous avons commencé par charger un texture différente pour chaque nombre et en fonction du nombre contenu dans la case actuellement afficher on dessine la texture correspondante.

Pour afficher la texture au bon endroit, il nous suffit de connaître la taille de la fenêtre et de la découper en carré de taille égale ou chaque carré représentera une case du plateau. On a décidé de décaler le plateau sur la droite de la fenêtre afin de laisser de l'espace pour afficher le nombre de coups joué et quelques boutons pour pouvoir par exemple quitter le jeu ou le sauvegarder.

```
int int_y;
int int_x;
/* On calcule la taille que doit faire une case en fonction de la taille de la fenêtre et
     du plateau */
int tailleCasePx  = FENETRE_H / (jeu->plateau->taille + 2);
/* On calcule le decalage verticale du plateau pour pouvoir laisser de l'espace au
    information sur le jeu */
int decallageVert = tailleCasePx - (4 * (jeu->plateau->taille - 1));
/* Pour chaque case du plateau de jeu on dessine la case dans la fenêtre */
for (int_y = 0; int_y < jeu->plateau->taille; int_y++) {
        for (int_x = 0; int_x < jeu->plateau->taille; int_x++) {
            /* La fonction renderCase prend en parammetre le "renderer" ou dessiner la
                case, la position x et y de la case ainsi que la taille en x et y de la
                case */
                renderCase(renderer, int_x * (tailleCasePx + 4)) + decallageVert,(int_y *
                     (tailleCasePx + 4)) + decallageVert, tailleCasePx, tailleCasePx)
                }
        }
}
```

4.4.2 Gestion des évènements

Une fois l'affichage géré et le plateau créé, Il nous faut gérer les évènements. C'est-à-dire, les interactions entre la machine et l'utilisateur. Ces interactions peuvent être multiples comme par exemple lorsque l'utilisateur appuie sur un bouton. Tout d'abord, il nous fallait gérer les différents moments du jeu. Nous avons opté pour un "switch" qui est la fonction execBouton afin de les traiter. Par exemple, lorsque l'utilisateur clique sur le bouton "sauvegarde" il enregistre sa partie qui se met en pause. Il faut maintenant gérer les interactions de l'utilisateur sur le jeu. Nous avons également utilisé un "switch" qui en fonction de l'évènement SDL réalisé va appelé une fonction qui va gérer cette évènement. Ces évènements sont :

- quitter le jeu
- cliquer sur une touche (pour les déplacements des tuiles)
- cliquer avec la souris

La gestion de touche consiste a appelé les fonctions de déplacements (déjà codées et expliquées ci-dessus) en fonction de la touche cliquée dans un "switch". Cependant, la gestion de la souris nous a demandé plus de réflexion. En effet, lorsque un clique de souris est effectué il faut vérifier chaque bouton de la fenêtre pour savoir où le clique a été effectuer. Lorsque le bouton est trouvé, on éxecute la fonction associée à ce bouton.

```
uint     int_button;
    /* variable de type structure bouton qui contient les informations d'un bouton.*/
button *bouton;
/* On parcourt tous les boutons de la fenêtre */
for (int_button = 0; int_button < Jeu->boutons->nbButtons; int_button++) {
        bouton = Jeu->boutons->tabButton[int_button];
        /* On compare la position de la souris sur l'écran à celle du bouton et on
            recentre la zone de recherche sur le centre du bouton et non sur un des coins
            */
        if (event->x > bouton->x - (bouton->w / 2) && event->y > bouton->y - (bouton->h /
             2) &&
                event->x < bouton->x + (bouton->w / 2) && event->y < bouton->y + (bouton
                    ->h / 2)) {
                /* Si la souris ce trouve sur ce bouton on exécute l'action associée à ce
                     bouton*/
                execBouton(Jeu, int_button);
        }
}
```

4.4.3 Gestion des textures

Étant donné que nous avons décider que le joueur pourrais continué de jouer après avoir fait un 2048, nous nous sommes rendu compte qu'il n'était pas possible de créer une texture pour chaque puissance de 2 à l'avance nous avons donc décidé de créer les texture de manière dynamique en cours de partie.

A chaque fois, qu'une case avec un nombre qui n'est pas encore apparu est créé, le jeu crée un nouvelle texture pour ce nombre et la stock dans un tableau dynamique contenant toutes les textures des cases. Il suffit donc par la suite de trouver une relation entre le nombre contenu dans la case et l'index de la texture du tableau. Heureusement tous les nombres possible dans le 2048 sont des puissances de deux et apparaissent dans l'ordre croissant dans le jeu et donc seront génèrés dans l'ordre croissant dans le tableau. Ainsi pour accéder par exemple à la texture d'une case dont la valeur est 1024, il nous suffit d'appeler le tableau de la manière suivante :

```
tabTex[(int)(log2(1024)) - 1]
```

Pour ce qui est de la création de la texture cela nous a posé quelque problème car la taille du texte change constamment en fonction de la valeur de la case et nous utilisons des textures avec de la transparence. Dans un premier temps, il nous a fallu trouver un moyen de calculer la taille que doit avoir le texte à ajouter sur la texture pour qu'il ne dépasse pas de la case.
Pour cela dans un premier temps on créer la texture du texte ,ainsi on possède les dimensions de cette texture et on pourra par la suite la dimensionner correctement sans l'étirer. Par la suite, on crée la texture du fond de la case avec une couleur qui dépend de la valeur de la case. Une fois que nous avons nos deux textures nous n'avons plus qu'a calculer le ratio entre la taille de la case et la longueur du texte et l'appliquer aux dimensions du texte que l'on dessine finalement sur la texture finale de la case qui sera ensuite sauvegardée en mémoire pour être réutilisée plus tard.
Pour ce qui de la transparence au début, nous avons essayé d'utiliser les masques Alpha mais nous ne comprenions pas comment les utiliser. Heureusement, la SDL dispose d'une extension "SDL_image" qui permet de charger des images au format png qui supporte nativement la transparence.

4.4.4 Gestion des menus

La gestion des menus est la dernière chose sur laquelle nous avons travaillé sur le projet car elle n'était pas très importante et n'apportait pas grand chose à l'utilisation du programme. Cependant, cela nous a permis de pratiquer un peut plus la programmation et réfléchir à certains concepts auxquels, nous n'avions jamais vraiment réfléchit en particulier quand un affichage graphique est présent. En effet, pour pouvoir gérer plusieurs "menu" nous devions pouvoir échanger entre plusieurs états du jeu avec des comportements diffèrents. Il nous a donc fallu repenser certaines parties du programme pour pouvoir redéfinir certains aspects du jeu à n'importe quel moment de l'exécution pour cela nous avons crée une nouvelle structure qui est passé dans presque toutes les fonctions de la partie affichage et gestion des évènements contenant toutes les données sur l'état du jeu.

De plus pour avoir des menus avec lesquels on peut interagir il nous a fallu intégrer des boutons sur lesquels on peut cliquer. Pour cela on a crée une structure stockant toutes les informations nécessaires pour afficher un bouton et savoir ou il se trouve à l'écran en cas de clique souris. Ensuite, il nous suffit de créer une fonction permettant d'afficher un bouton à l'écran et un autre pour savoir si l'utilisateur a bien cliquer sur un bouton et exécuté les actions associées avec ce bouton.

```
typedef struct
{
        etatJeu *          etatJeu;
        SDL_Window *  window;
        SDL_Renderer *renderer;
        TTF_Font *         font;
        TextureBank * TextureBank;
        Buttons *          boutons;
} jeu;
```

- **etatJeu** : Pointeur vers la structure contenant l'état du 2048.
- **window** : Pointeur vers la structure de la fenêtre SDL courante
- **renderer** : Pointeur vers le l'écran SDL actuel.
- **font** : Pointeur la police utilisé lors de l'affichage d'un texte.
- **TextureBank** : Pointeur vers le tableau contenant toutes les texture du jeu.
- **boutons** : Pointeur vers le tableau de bouton actuellement affiché a l'écran.

5 Bilan personnel

5.1 Mosser Hugo

Pour ma part, j'ai trouvé que ce sujet de projet était intéressant car on retrouvait tout ce qu'on a appris depuis le début de l'année en matière de codage C mais aussi de nouvelles choses comme la SDL pour tout ce qui est graphisme. Cependant, même si ce projet semblait amusant, je trouve qu'il était tout de même très complexe et j'ai eu beaucoup de mal sur la réalisation de certaines parties où j'ai nécessité d'aides et de conseils. Toutefois, j'ai appris énormément de nouvelles choses dans ce projet comme de nouvelles techniques de codage ou l'utilisation de plate-forme d'édition de code en ligne ou de partage de code (Vscode et Git).

5.2 Rodrigues Samuel

Ce projet a été pour moi une expérience intéressante sur laquelle j'ai pu m'exercer et m'améliorer en programmation avec le langage C. J'ai ainsi découvert la bibliothèque SDL qui est très complète et qui, si bien utilisée, permet de réaliser des interfaces complètes et performantes. J'ai aussi pu partager mes connaissances avec les autres membres du groupe et leur apprendre à utiliser l'outil Git de manière succincte. En tant que projet de fin d'année, je trouve que ce dernier nous a permis de bien reprendre et réutiliser la plupart des outils qui nous ont était introduit cette année.

5.3 Meunier William

Comme chaque projet celui-là été très complet, coder le 2048 nécessitait d'avoir acquis tout se que nous avons appris cette année. Hormis le SDL ou nous nous sommes aidé de Open Classroom afin d'avoir un affichage plus esthétique que celui du terminale. Le fait de pouvoir faire le projet à 4 nous a permit de pouvoir le faire sans réel problème de temps grâce à VShare et Git.

5.4 Paesa Théo

Cette fin d'année se termine sur ce projet qui a été une piqûre de rappel sur les différentes notions vues en C tout au long de l'année. Mais, selon moi, la partie la plus intéressante de ce projet a été l'apprentissage de la bibliothèque SDL. En effet, jusque là, nos projets avaient une interface sur le terminal. Mais pour ce dernier, nous avons dû coder une interface graphique. L'évolution de notre fenêtre de jeu tout au long du projet a été fascinante et m'a donné de la motivation pour l'améliorer. Au final, nous savons maintenant faire une interface graphique, ce qui rend le codage bien plus ludique et nous a permis de maîtriser un nouvel outil qui peut nous permettre, à l'avenir, de coder des jeux en 2D.