Makkes Malik

UNIVERSITE DE CLERMONT AUVERGNE

LICENCE 2 INFORMATIQUE 2021-2022

Rapport du jeu 2048

Avant tout il faut savoir que nous avons créé un fichier qui s'appel README.md pour vous aider à compilez et à exécuter le jeu, lisez-le s'il vous plait il est très important.

Maintenant nous allons voir ce que nous avons dans le dossier du jeu, d'abord nous avons le document 2048.c qui contient les fonctions utilisées dans le main.c, ensuite nous avons un document cases.png qui représente la grille de notre jeu 2048, après nous avons un document CMakeLists.txt pour la compilation, par la suite nous avons notre main.c, plus bas nous avons le document Score.txt qui contient notre score de chaque partie et enfin nous avons le document Style.ttf qui correspond au type de police utilisé.

Ensuite nous allons voir ce que contient le document 2048.c et ce que chaque fonction fait, pour cela nous allons prendre des lignes de codes et les mettre ici et les expliquer :

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <SDL/SDL.h>
#include <SDL/SDL_ttf.h>
#include <SDL/SDL_image.h>
```

Voici les includes utilisés par notre jeu, nous aurons besoin des bibliothèques : stdlib.h, stdio.h, time.h pour pouvoir utiliser la fonction random pour nos cases de 2 et 4, math.h pour utiliser les puissances de 2 ainsi que les calculs mathématiques, SDL/SDL.h qui contient les fonctions pour le jeu, SDL/SDL_ttf.h pour le font c'est-à-dire le type de texte et enfin SDL/SDL_image.h pour pouvoir utiliser notre image de la grille.

```c
enum { HAUT, BAS, GAUCHE, DROITE };
```

On utilise enum pour énumérer les mouvements Haut, Bas, Gauche et Droite qui seront utilisés en tant que des entiers pour le sens dans la fonction Mouvement.

```c
int deplacerCases(int* premiereCase, int* secondeCase)
{
    if (*secondeCase == 0)
    {
        *secondeCase = *premiereCase;
        *premiereCase = 0;
        return 0;
    }
    else  if (*secondeCase == *premiereCase)
    {
        (*secondeCase)++;
        *premiereCase = 0;
        return *secondeCase;
    }
    else
        return -1;
}
```

Fonction qui sera utilisé plus tard pour les déplacements des cases, elle prend en paramètre deux pointeurs de type entiers, son objectif est de retourner 0 si la case a été déplacée, sinon la puissance de deux si elle a été fusionnée sinon elle retourne -1. On verra plus tard son utilité dans notre jeu.

```
void Mouvement(int* score, int cases[][4], int sens)
{
  int i = 0, j = 0, k = 0;
  int retourDeplacer = 0;
}
```

Fonction pour les mouvements des cases, elle prend en paramètre un pointeur de type entier, un tableau avec 4 colonnes et un entier sens.

```
switch (sens)
    {
    case HAUT:
        for (i = 0; i < 4; i++)
        {
            for (j = 1; j < 4; j++)
            {
                switch (j)
                {
                case 1:
                    retourDeplacer =
deplacerCases(&cases[i][j], &cases[i][j - 1]);
                    if (retourDeplacer > 0)
                        *score += pow(2,
retourDeplacer);
                    break;
                case 2:
                    retourDeplacer =
deplacerCases(&cases[i][j], &cases[i][j - 1]);
                    if (retourDeplacer == 0)
                        retourDeplacer =
deplacerCases(&cases[i][j - 1], &cases[i][j -
2]);
                    if (retourDeplacer > 0)
                        *score += pow(2,
retourDeplacer);
                    break;
                case 3:
                    retourDeplacer =
deplacerCases(&cases[i][j], &cases[i][j - 1]);
                    if (retourDeplacer > 0)
                        *score += pow(2,
retourDeplacer);
                    if (retourDeplacer == 0)
                    {
                        retourDeplacer =
deplacerCases(&cases[i][j - 1], &cases[i][j -
2]);
                        if (retourDeplacer == 0)
                            retourDeplacer =
deplacerCases(&cases[i][j - 2], &cases[i][j -
3]);
                        if (retourDeplacer > 0)
                            *score += pow(2,
retourDeplacer);
                    }
                    break;
                }
            }
        }
        break;
```

Par manque de place et surtout pour la lisibilité je vais seulement expliquer en détail un seul cas de la boucle, vu que les autres boucles ont le même fonctionnement. On utilisera un switch qui prend en paramètre le sens, ensuite on aura 4 cas, cad autant de types que de mouvements que l'utilisateur utilisera. Dans le 1er cas on va traiter le cas l'utilisateur clique sur la touche HAUT, on va parcourir toute la première ligne du tableau case et les 3 colonnes du tableau en commençant par la 2eme colonnes. Ensuite nous avons 3 cas, 1er cas le déplacement a causé soit un simple déplacement ou un fusionnage avec une case de la colonne 0 et donc on évalue la condition si retourDeplacer supérieur à 0, donc on donne à score sa propre valeur ainsi que la puissance de 2 par la valeur retourDeplacer. 2eme cas le déplacement a causé un fusionnage soit avec une case de la colonne 0 soit avec une case de la colonne 1 ou soit un simple déplacement et donc on fait la même chose que précédemment on évalue si retourDeplcer est égale à 0 ou supérieur à 0 et on applique à score le nouveau score si y a un changement ou rien. Enfin dans notre dernier cas le déplacement a soit cause un simple déplacement donc le score reste inchangé sinon on évalue 3 conditions cette fois ci, avec le fusionnage des colonnes 0 ou 1 ou 2 et encore une fois on finit toujours par mettre à jour le score. Le reste de la fonction fait pareil mais pour les autres mouvements, elle évalue chaque cas et met à jour le score.

```
int Chiffres(int nombre)
{
    int nbChiffres = 0;

    if (nombre == 0)
    {
        nbChiffres = 1;
    }
    else
    {
        while (nombre != 0)
        {
            nbChiffres += 1;
            nombre /= 10;
        }
    }
    return nbChiffres;
}
```

Fonction très simple qui renvoie le nombre de chiffres qui composent le nombre en paramètre en base 10. Pour cela on évalue si nombre est égale à 0 si oui on donne juste à nbChiffres la valeur 1, sinon elle est différente de 0 et donc on utilise une boucle while le nombre est différents de 0 on donne à nbChiffres sa propre valeur plus 1 à chaque parcours et on donne à nombre sa valeur divisé par 10. Bien évidemment on utilisera cette fonction plus tard.

```
int detecterCasesVides(int cases[][4], int casesVides[16 + 1])
{
    int i = 0, j = 0, nbCasesVides = 0;

    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < 4; j++)
        {
            if (cases[i][j] == 0)
            {
                casesVides[nbCasesVides] = i * 10 + j;

                nbCasesVides++;
            }
        }
    }
    casesVides[nbCasesVides] = 0;

    return nbCasesVides;
}
```

Fonction qui détecte combien de cases vides on a, d'où son nom, elle prend en paramètre un tableau d'entier à 2 dimensions ainsi qu'un tableau à 17 cases. Elle parcourt toutes les cases du tableau cases et si une case du tableau cases est égale à 0 alors casesVides prendra la valeur de i*10+j et on incrémente nbCasesVides de 1, ainsi de suite, et on finit par réinitialiser la dernière case du tableau casesVides à 0 et on retourne nbCasesVides.

Vous allez constater que le jeu se ferme dès qu'il y a plus de cases disponibles, et il va revenir vers la fenêtre de menu, c'est tout à fait normal, car on a voulu faire jouer la personne jusqu'à ce qu'elle décide d'arrêter ou jusqu'à qu'elle soit satisfaite de son score et veut qu'il soit enregistré dans le document score.txt, sinon elle peut enchainer les parties autant qu'elle veut. Pour les couleurs nous avons utilisez paint, pour mélanger les couleurs red, green, blue afin d'obtenir les exact mêmes couleurs que le vraie jeu 2048 pour nos 16 cases.

Nous allons continuer maintenant avec l'explication des autres fonctions du document 2048.c :

```
void placerNbAlea(int cases[][4], int casesVides[16])
{
    int nbCasesVides = 0, caseChoisie = 0, nombre = 0;
    srand(time(NULL));
    nombre = rand() % 30 + 1;
    if (nombre < 29)
        nombre = 1;
    else
        nombre = 2;
    nbCasesVides = detecterCasesVides(cases, casesVides);
    caseChoisie = casesVides[rand() % nbCasesVides];
    if (cases[caseChoisie / 10][caseChoisie -
(caseChoisie / 10) * 10] != 0)
    {
        exit(EXIT_FAILURE);
    }
    cases[caseChoisie / 10][caseChoisie - (caseChoisie /
10) * 10] = nombre;
}
```

Fonction assez explicite qui prend en paramètre un tableau à deux dimensions ainsi qu'un tableau casesVides, on générera avec cette fonction une case avec un nombre égale a 2 puissances 1 ou 2 puissances 2 et la case apparait dans une case sur la grille qui est vide aléatoirement. On crée aussi un cas d'arrêt s'il n'y a plus de case vide dans le jeu, la partie est terminé avec le exit, pour forcer l'arrêt du jeu.

```
int TheGame(SDL_Surface* ecran)
```

On a la fonction int TheGame qui prend en paramètre une surface graphique et qui retourne le score, nous verrons son contenu par la suite.

```
SDL_Surface* grille = NULL;
SDL_Rect positionGrille;
SDL_Event event;
TTF_Font* police = NULL;
SDL_Color couleur = { 0, 0, 0 };
Uint32 time;
grille = IMG_Load("cases.png");
positionGrille.x = 0;
positionGrille.y = 75;
```

Je vais expliquer quelques parties du code encore une fois par manque de place, SDL_Surface sert a créé une surface graphique avec SDL, SDL_Rect sert a créé une surface rectangulaire de l'écran, SDL_Event pour créer un évènement, TTF_Font pour la police, Uint32 pour initialiser le timer en tant que unsigned interger, IMG_Load, on importe l'image, positionGrille.x et positionGrille.y sert à placer la grille sur notre surface graphique.

```
nbChiffres = Chiffres(nb);
taille = 150 / nbChiffres * 0.95;
police = TTF_OpenFont("Style.ttf", taille);
sprintf(puissanceDeux, "%d", nb);
texte = TTF_RenderText_Blended(police, puissanceDeux,
couleur);
position.x = 150 / 2 - texte->w / 2;
position.y = 150 / 2 - texte->h / 2;
```

TTF_OpenFont sert a ouvrir notre fichier Style.ttf et on lui donne la taille qui est proportionnelle à la case et on utilisera texte->w/2 et texte->h/2 pour placer le texte au milieu de la case, w sert à récupérer la hauteur et w pour récupérer sa largeur.

```
SDL_WaitEvent(&event);
SDL_QUIT;
SDL_KEYDOWN;
SDLK_ESCAPE;
SDLK_UP;
SDLK_DOWN
SDLK_LEFT
SDLK_RIGHT
```

SDL_WaitEvent(&event) attend qu'un évènement soit enclenché, SDL_QUIT quitte SDL, SDL_KEYDOWN ou SDLK_ESCAPE ou SDLK_UP ou SDLK_DOWN ou SDLK_LEFT et SDLK_RIGHT correspondent tous a des évènements des touches du clavier, leur nom est assez explicite, car c'est juste SDL Key puis le nom de la touche.

```
SDL_FillRect(ecran, NULL,
SDL_MapRGB(ecran->format, 204, 192, 179));

time = SDL_GetTicks();

sprintf(chronometre, "Timer : %d sec", time /
1000);

sprintf(puissanceDeux, "Score : %d", score);
chrono = TTF_RenderText_Blended(police,
chronometre, couleur);

texte = TTF_RenderText_Blended(police,
puissanceDeux, couleur);

SDL_BlitSurface(chrono, NULL, ecran,
&chronoPosition);

SDL_BlitSurface(texte, NULL, ecran,
&positionTexte);

SDL_Flip(ecran);

SDL_FreeSurface(grille);
```

SDL_FillRect sert à remplir un SDL_Rect, il prend en paramètre notre fenêtre, NULL, et SDL_Map avec le mélange des couleurs red, green et blue. On donne à time SDL _GetTicks() ; qui correspond au temps depuis lequel le jeu fonctionne en millisecondes. Sprintf sert à stocker la conversion en chaine de caractères. TTF_RenderText_Blended sert à donner à chrono la chaine de caractère chronomètre avec la police et sa couleur dans notre cas. Par la suite on SDL_BlitSurface qui sert à coller une certaine surface crée sur notre surface principale de jeu, dans notre cas ça sera chrono et texte. SDL_Flip sert à mettre à jour la fenêtre avec les nouvelles modifications et enfin SDL_FreeSurface sert à vider la case mémoire de ce qu'elle prend en paramètre étant une surface.

Voilà, c'est quasiment toutes les fonctions que nous avons utilisées dans le document 2048.c, bien évidemment il est fort contestable que les parties expliquées ont été expliqués assez brièvement, car le document 2048.c contient aussi beaucoup de commentaire détaillant certaines lignes de codes assez complexes, en plus elles sont très explicites.

Passons maintenant au document main.c, je vais pas revenir par manque de place sur les fonctions vues précédemment, nous allons juste expliqués les nouvelles fonctions :

```
SDL_Init(SDL_INIT_VIDEO);
SDL_SetVideoMode(600, 675, 32, SDL_HWSURFACE);
SDL_WM_SetCaption("Malik et Anis 2048", NULL);

if (score > 2048)
    {
      printf("Bien Joué ton score finale
      est : %d", score);
    }
else
    {
      printf("Ton score finale est : %d",
      score);
    }
    FILE* fp = fopen("Score.txt", "a+");
    fprintf(fp, "Votre socre de la dernière
    partie est : %d\n", score);
    fclose(fp);
```

On utilise SDL_Init(SDL_INIT_VIDEO); pour initialiser la SDL, SDL_SetVideoMode(600, 675, 32, SDL_HWSURFACE); sert a initialiser l'affichage graphique, c'est-à-dire la fenêtre qui va avoir une taille ici de 600 pixels en largeur et 675 en longueurs avec 32 bits par pixel et on utilisera SDL_HardwareSurface. SDL_WM_SetCaption pour le titre de la fenêtre. Enfin lorsque la personne décide de fermer le jeu s'il est satisfait de son score, on lui affiche son score et on l'enregistre dans le document Score.txt.

Merci pour avoir tester notre jeu.