26/06/2022

MANUEL DEVELOPPEUR SNAKE 2022

Makan KONATÉ
ESIEE - PARIS

ESIEE PARIS

Table des matières

1. Introduction

Le jeu du snake est un jeu d'action sur ordinateur dont le but est de contrôler un serpent qui se déplace et collecte de la nourriture sur une carte. Le serpent doit manger tous les fruits sans se mordre la queue et sans entrer en collision dans un mur.
Vous devez vous déplacer à l'aide des touches directionnelles.

Ce programme comporte trois options possibles :

- Sans paramètre donné lors de l'exécution du jeu. Le jeu se lance avec la grille par défaut.
- Avec l'option « -h ». Le terminal affiche les choix possibles pour lancer le jeu.
- Avec l'option « -i src/grid.txt ». Le jeu se lance avec la grille contenue dans le fichier grid.txt

Ce projet a été réalisé avec la librairie MLV. Pour savoir comment installer la librairie MLV, veuillez suivre les instructions présentes dans le README.
Le programme ne pourra être exécutée sans la Librairie MLV.

2. L'archive et les fichiers

L'archive comporte un répertoire qui contient :

- Un fichier README, décrivant brièvement le contenu de l'archive et la procédure de compilation et lancement du programme
- Un fichier Makefile avec les cibles suivantes :
  - Snake qui permet d'obtenir un exécutable pour jouer au jeu
  - Clean qui permet d'effacer tous les fichiers de compilation temporaires (les .o) mais pas l'exécutable
  - Clear qui permet d'effacer tous les fichiers de compilation temporaires (les .o) et l'exécutable.
  - tgz qui génère l'archive au format tgz

L'archive comporte 3 sous répertoires :

- src/ contenant les sources et le fichiers levels
- bin/ qui est vide lorsqu'on décompresse l'archive, et contenant les .o et l'exécutable après compilation
- doc/ contenant deux documents **user** et **dev** au format pdf.

2.1 Fichiers grid.h et grid.c

Les fichiers grid.h et grid.c contiennent les fonctions et constantes permettant de représenter une grille de jeu. Ces fichiers contiennent aussi les fonctions permettant de réserver en mémoire l'espace pour le serpent et de l'afficher sur la grille de jeu.

Le fichier grid.h contient les prototypes des fonctions du fichier grid.c, une structure grid qui permet de réaliser la grille de jeu.

2.2 Fichiers snake

Les fichiers snake.c et snake.h contiennent les fonctions permettant d'initialiser le serpent et de calculer ces coordonnées.
Le fichier snake.h contient les prototypes du fichier snake.c ainsi que deux structures :

- Coord qui correspond aux coordonnées du serpent
- enum direction qui représente les 4 directions possibles

2.3 Fichiers game

Le fichier game.c contient une fonction main() qui sert à démarrer le programme.

Tous ces fichiers incluent les librairies nécessaires à la réalisation du programme.

3. Les fonctions

3.1 Grid.c

3.1.1 Fonction debug

```
void debug(Grid *grid)
```

La fonction debug prend en paramètre un pointeur de type Grid. Elle permet d'afficher la grille de jeu.

3.1.2 Fonction compute_size

```
int compute_size(int w, int h, Grid *grid)
```

Cette fonction prend en paramètre deux entiers w, h qui représentent respectivement la largeur et la hauteur d'une fenêtre graphique. Elle prend aussi en paramètre un pointeur de type Grid. La fonction calcule alors la plus grande valeur a possible pour représenter toute la grille avec des carrés de côté a de sorte qu'elle tienne entièrement dans la fenêtre.

3.1.3 Fonction draw_grid

```
void draw_grid(Grid *grid)
```

Cette fonction prend en paramètre la grille et l'affiche dans une fenêtre graphique. Les cases sont des carrés de côté calculé avec la fonction compute_size().
La taille de la fenêtre est récupéré à l'aide de la bibliothèque MLV.
Cette fonction affiche aussi les objets présents dans la grille (WALL, FRUIT, EMPTY et le SNAKE)
Chaque objet est représenté par des rectangles de couleur différents.
Ces objets ont été réalisé grâce à la bibliothèque MLV.

3.1.4 Fonction place_snake

```
void place_snake(Grid *grid, Snake *serpent)
```

Cette fonction prend en paramètre un pointeur de type Grid et un pointeur de type Snake.
Cette fonction a pour but de placer le serpent dans la grille. On parcourt la segments_list (liste chaînée) afin de connaître toutes les coordonnées du serpent. Et on affecte SNAKE à chaque coordonnée du serpent.

3.1.5 Fonction move_snake

```
enum Element move_snake(Grid *grid, Snake *serpent)
```

La fonction prend en paramètres un serpent et une grille.
Cette fonction :

- Efface la queue du serpent de la grille (met à EMPTY dans la grille la position stockée en fin du tableau représentant les positions du serpent)
- Appelle la fonction **crawl();** (voir 3.2.1).

Cette fonction dessine la nouvelle du serpent.

3.1.6 Fonction allocate_grid

```
Grid *allocate_grid(int n, int m)
```

Cette fonction prend en paramètre deux entiers.
La fonction allocate_grid alloue une variable de type **Grid** sur le tas et alloue son champs grid.
Cette fonction parcourt chaque ligne du tableau dynamique et pour chaque ligne de ce tableau, on alloue un tableau dynamique représentant le nombre de colonne pour chaque ligne.
Cette fonction retourne une variable « grid » de type **Grid.**
Pour résumer cette fonction créée un tableau à deux dimensions (grille du jeu).

3.1.7 Fonction free_grid

```
void free_grid(Grid *grid)
```

Cette fonction libère toute la mémoire occupée par la grille (le tableau à deux dimensions puis la variable de type Grid.

3.2 Snake.c

3.2.1 Fonction Crawl

```
void crawl(Snake * serpent, int nbc, int nbl)
```

Cette fonction prend en paramètre un pointeur de type **Snake** et deux entiers représentant le nombre de ligne et de colonne.
Cette fonction a pour but de déplacer le serpent selon les directions.
La fonction supprime la queue du serpent et on ajoute la tête du serpent aux coordonnées calculés.
Les coordonnées de la tête du serpent sont calculés en fonction de la direction choisie.

3.2.2 Fonction new_snake

```
Snake* new_snake()
```

Cette fonction ne prend pas de paramètre.
J'ai réalisé une allocation dynamique d'une variable de type **Snake**.
Cette fonction renvoie un nouveau serpent alloué dynamiquement dans lequel le champs size est à 0 et le champs segments_list est à NULL.

3.2.3 Fonction add_segment

```
void add_segment(Snake * serpent, int x, int y)
```

La fonction add_segment prend en paramètre un pointeur de type **Snake** et deux entiers x et y, elle créée une Position à partir de x et y, allouée dynamiquement et dont le champs next est à NULL.
Cette fonction ajoute cette Position en queue de la liste des segments.

3.2.4 Fonction free_snake

```
void free_snake(Snake* serpent)
```

Cette fonction prend en paramètre un pointeur de type **Snake**. Celle-ci libère toute la mémoire associée à un serpent. Elle libère aussi la liste chaînée.

3.3 Game.c

Le fichier game.c contient 3 variable prédéfinie :

- X et Y qui correspondent à la taille de l'écran de jeu
- DIFFICULTY qui correspond à la difficulté du jeu. Plus la valeur affectée à la difficulté est basse, plus le serpent sera rapide. Donc le jeu sera plus difficile.

3.3.1 Fonction nb_fruits

```
int nb_fruits(Grid *grid)
```

Cette fonction prend en paramètre un pointeur de type **Grid.** Elle parcourt la grille de jeu et à chaque fruit rencontré elle incrémente le nombre de fruit.
Cette fonction compte le nombre de fruit présent dans la grille de jeu.
Le nombre de fruit s'affiche dans le terminal une fois la partie terminée.

3.3.2 Fonction count_nb_nligne

```
int count_nb_ligne(FILE *f)
```

Cette fonction prend en paramètre un fichier. Elle compte le nombre de ligne qu'il y'a dans le fichier contenant la grille de jeu (dans mon cas **grid.txt**).

3.3.3 Fonction copy

```
void copy(char *source, char *grid)
```

Cette fonction copy la grille d'un fichier à un autre (ligne par ligne).

3.4 Main

```
int main(int argc, char *argv[])
```

Toutes les variables ont été initialisé en début de fonction pour respecter la norme ANSI.
La fonction main appelle les fonctions nécessaires pour la réalisation du jeu snake.
Pour afficher les aides d'exécution du jeu, j'ai utilisé la librairie « **getopt.h** ».

Pour voir où sont appelés les fonctions, veuillez vous diriger dans le fichier main. Des commentaires y sont ajoutés pour les appels de fonctions.

4. Rapport d'erreur, warnings

Ce programme ne comporte pas d'erreur. Seulement deux warnings et une note qui sont les suivants :

```
┌──(kali㉿kali)-[~/Documents/KONATE_makan_snake2022/snake2022]
└─$ make
gcc -o bin/game.o -c src/game.c -W -Wall -ansi -pedantic -g -lMLV
src/game.c: In function 'main':
src/game.c:123:11: warning: implicit declaration of function 'getline' [-Wimplicit-function-declaration]
  123 |     nbc = getline(&buf, &size_buf, stream);
      |           ^~~~~~~
src/game.c:107:9: warning: '__builtin_memcpy' writing 19 bytes into a region of size 1 overflows the destination [-Wstri
ngop-overflow=]
  107 |         strcpy(filename, "src/levels/default");
      |         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
src/game.c:76:10: note: destination object 'filename' of size 1
   76 |     char filename[] = "";
      |          ^~~~~~~~
gcc -o bin/grid.o -c src/grid.c -W -Wall -ansi -pedantic -g -lMLV
gcc -o bin/snake.o -c src/snake.c -W -Wall -ansi -pedantic -g -lMLV
gcc -o bin/snake bin/game.o bin/grid.o bin/snake.o -W -Wall -ansi -pedantic -g -lMLV
```

5.Compilation

Pour compiler le jeu et l'exécuter, veuillez vous référez au document user.pdf qui se trouve dans le sous répertoire doc.

6. Conclusion

Ce projet m'a permis de mettre en application toutes les notions en C que j'ai pu apprendre tels que les structures, les allocations dynamiques et les listes chaînées. J'ai notamment pu découvrir l'utilité du Makefile qui permet de pouvoir compiler tous les fichiers en une seule commande « **make** ».
Ce projet permet de faire un bon résumé sur les bases à avoir en langage C.

Vous avez ci-dessus une explication globale du programme. Des commentaires sont présents dans le programme pour vous aider. A vous de vous amuser avec ce programme et de le modifier ou bien de l'améliorer.

Bon courage 😉