Ce document a pour but d'expliquer aux développeurs comment fonctionne le jeu Snake. Par la suite, les développeurs pourront reprendre le développement du jeu, corriger ces bugs et ajouter des fonctionnalités. Les fichiers et les fonctions seront expliqués en détails pour permettre aux développeurs de continuer le jeu sans avoir besoin de lire le code.

Sommaire :

I.Explication du projet

Ce projet a été réalisé dans le cadre académique au sein de l'établissement ESIEE Paris par une élève en 3ème année d'école d'ingénieur. Le but de ce projet est de créer un jeu Snake en langage de programmation C.

II.Liste des fichiers et répertoires

Le projet est composé de 8 fichiers contenant les codes sources. Ces fichiers sont :

- game.c
- getopt_long.c
- getopt_long.h
- grid.c
- grid.h
- snake.c
- snake.h
- makefile

Ces fichiers sont présents dans le répertoire "src". Ce répertoire est aussi composé d'un répertoire "bin" qui regroupe tous les fichiers ".o " créés au moment du make (compilation) du projet. L'exécutable "game" permet quant à lui de lancer le jeu en ligne de commande. La commande pour lancer le jeu avec le niveau 1 est :

./game -i level1.txt

Par ailleurs, le répertoire "src" contient aussi un autre répertoire appelé levels. Il est composé de seulement un fichier default qui permet au jeu de se lancer si le fichier mis en paramètre au moment du lancement n'existe ou ne fonctionne pas.

III. Description des fichiers et des fonctions

Les fonctions seront présentées et expliquées à partir du fichier game.c car c'est le fichier dont provient le « main » et la boucle principale du jeu.

game.c :

Le fichier source game.c contient le point de départ du jeu qui repose sur la librairie graphique MLV. De plus, il gère l'initialisation du jeu avec sa boucle principale et les actions attendues à la fin du lancement du jeu.

Au total, onze fonctions sont appelées dans ce code source depuis les headers grid.h, snake.h et getopt_long.h.

Les fonctions sont :

- print_help	- add_segment
- allocate_grid	- place_snake
- charge_grid_fichier	- move_snake
- compte_fruits	- draw_grid
- new_snake	- free_snake
- free_grid	

La fonction **print_help**, déclarée dans le header getopt_long.h, traite les arguments mis en ligne de commande au moment du lancement du jeu. Elle est déclarée dans le fichier getopt_long.h et est dans le fichier getopt_long.c utilisé pour la gestion des options de lignes de commande. Cette fonction prend en paramètre le nombre d'argument et un tableau de chaîne de caractères composait des arguments passés en ligne de commande. De plus, cette fonction gère les options comme :

- -i, –input
- -h, –help
- -o, –ouput
- -v, –verbose

Le code source du fichier getopt_long.c a été récupéré à partir du man getopt_long.

La fonction **allocate_grid,** déclarée dans le header grid.h, prend en paramètres le nombre de ligne et de colonne et permet l'initialisation de la grille de jeu et alloue dynamiquement de la mémoire à la structure "Grid" déclarée dans le fichier grid.h. Si au départ l'allocation de mémoire ne fonctionne pas, le message d'erreur "*Allocation mémoire impossible pour la structure de la grille*" s'affiche et la mémoire est libérée. Les dimensions de grilles qui sont les nombres de lignes et les nombres de colonnes choisies dans game.c sont stockées dans la structure "Grid".

Ensuite la fonction va allouer de la mémoire pour les lignes et les colonnes. Si l'une des deux allocations échoue, l'exécution du jeu s'arrête, la mémoire est libérée et un des messages suivants va s'afficher. "*Allocation mémoire impossible pour les lignes de la grille*" ou "*Allocation mémoire impossible pour la colonne de la grille*". Si toutes les allocations ont réussi alors la fonction va retourner un pointeur vers la structure "Grid". Cette fonction a donc créé une grille de jeu de la taille spécifiée à partir de game.c avec le nombre de ligne et le nombre de colonne. Par la suite des éléments sont placés sur la grille comme des fruits, des murs et le serpent.

La fonction **charge_grid_fichier,** déclarée dans le header grid.h, prend en paramètres un pointeur vers la structure "Grid" où la grille est chargée et une chaîne de caractères qui est le nom du fichier qui contient la grille. Elle permet donc de charger les données de la grille à partir du fichier mit en paramètres en ligne de commande. Elle initialise la structure "Grid" défini au départ avec les données du fichier.
Si l'ouverture du fichier est échec alors la fonction retourne qu'il est impossible d'ouvrir le fichier pour lire la grille. Sinon, la fonction fait appel aux fonctions **getline** pour lire la première ligne du fichier et compter le nombre de colonne et à la fonction **count_nb_lines** pour compter le nombre de lignes dans le fichier. Par la suite la première ligne qui est lue dans le fichier est recopié dans la grille.
Cette fonction a donc permis d'initialiser la grille de jeu à partir des données chargées du fichier précisé en paramètre lors de son lancement. De plus, si le chargement du fichier échoue alors la fonction chargera les données à partir du fichier default.

La fonction **compte_fruits,** déclarée dans le header grid.h, prend en paramètre un pointeur constant vers la structure "Grid" qui représente la grille de jeu. Elle permet de compter le nombre de fruit présent dans le fichier qui est mis en paramètre et parcours donc toutes les cellules de la grille et incrémente un compteur. Elle permet donc de savoir si le joueur a gagné sa partie car il mangé tous les fruits et le compteur est maintenant à zéro.

La fonction **new_snake**, déclaré dans le header snake.h, est déclarée dans le header snake.h. Cette fonction initialise un nouveau serpent pour le jeu. La mémoire de la structure "Snake" est allouée dynamiquement et renvoie ensuite un pointeur vers cette structure qui vient d'être créée.
Par ailleurs, si l'allocation pour la structure Snake échoue alors un message d'erreur s'affiche, *"Impossible d'allouer la mémoire pour le serpent"*.

La fonction **add_segment,** déclarée dans le header snake.h, prend en paramètres un pointeur vers la structure "Snake" pour laquelle on veut ajouter un segment, la position x et y du nouveau segment. Cette fonction a donc pour but d'ajouter un nouveau segment au niveau de la fin de la queue du serpent qui est au format d'une liste chaînée.
La mémoire pour le nouveau segment du serpent est allouée dynamiquement. Si l'allocation ne fonctionne pas alors le lancement du jeu est arrêté est un message d'erreur s'affiche qui est : *"Impossible d'allouer la mémoire pour le nouveau segment"*. Par ailleurs, les champs x et y du champ sont initialisés avec les autres coordonnées du serpent. La taille du serpent est augmentée à partir de cette fonction avec le paramètre size qui est incrémenté chaque fois que le serpent mange un fruit.

La fonction **place_snake**, déclarée dans le header snake.h, prend en paramètre un pointeur vers la structure “Grid” qui représente la grille du jeu et un autre pointeur vers la structure “Snake” qui représente le serpent. Cette fonction permet de placer le serpent sur la grille du jeu et parcours la liste chaînée du serpent pour mettre à jour sa position. Elle est aussi utilisée quand le serpent mange un fruit et qu’il faut ajouter un nouveau segment.

La fonction Element **move_snake,** déclarée dans le header grid.h, prend en paramètre un pointeur vers la structure “Grid” et un pointeur vers la structure “Snake”. Elle gère le déplacement du serpent sur la grille et calcul la nouvelle position de sa tête en fonction de sa direction. Elle permet aussi de vérifier les collisions avec le mur et modifie la longueur du serpent s’il mange un fruit. De plus, si le serpent ne mange pas de fruit alors sa queue est supprimé pour respecter sa taille.
Par ailleurs, si le serpent se prend un mur alors la partie est terminée est la fonction retourne le message suivant, *“Partie perdue”*. En revanche si le serpent a mangé tous les fruits sur la grille alors la fonction retourne, *“Bravo vous avez mangé tous les fruits de la grille”*, et la partie est aussi terminée.

La **fonction draw_grid**, déclarée dans le header grid.h, prend en paramètre un pointeur constant vers la structure “Grid”. Cette fonction permet de dessiner la grille sur la fenêtre graphique. La bibliothèque MLV permet de changer la couleur de chaque cellule avec la couleur attendue. Il y a donc 4 couleurs qui sont pour le mur, les fruits, le serpent et le vide.
De plus, elle permet de calculer la taille de chaque cellule en fonction de la largeur et de la hauteur de la fenêtre. Par défaut la fenêtre a pour couleur marron.

La fonction **free_snake**, déclarée dans le header snake.h, prend en paramètre la structure “Snake” et permet de libérer la mémoire qui a été allouer dynamiquement au serpent à la fin du jeu. Cette fonction traverse donc la liste chaînée du serpent puis libère sa mémoire.

La fonction **free_grid**, déclarée dans le header grid.h, prend en paramètre la structure “Grid” et libère la mémoire qui a été allouer dynamiquement à la grille du jeu. Chaque ligne de ligne et sa structure sont donc libérées. Cela permet donc de s’assurer qu’il n’y a pas de fuite de mémoire.

La fonction **compute_size**, déclarée dans le header grid.h, prend en paramètre la largeur et la hauteur de la fenêtre, le nombre de colonnes et de lignes dans la grille. Cette fonction permet de calculer la taille maximale d’un carré (cellule) qui donnera des cellules uniformes dans la fenêtre de jeu. Elle s’adapte à la grille et à la dimension de la fenêtre. Par conséquent, les éléments de la grille sont proportionnels à la taille de la fenêtre. De plus, cette fonction est utilisée par la fonction draw_grid pour déterminer la taille de ses carrés.

IV. Bugs

Chaque fois que je fais un make clean et que je compile mon code, j'ai une erreur qui s'affiche concernant le type de fichier FILE. J'ai cette erreur qui s'affiche depuis que j'ai changé mon makefile pour qu'il compile les fichiers .o dans le répertoire bin. Je dois maintenant ajouter stdio.h dans mon header grid.h.

Il arrive parfois que la partie se termine alors que le serpent ne s'est pas pris un mur mais a mangé un fruit. Pour régler ce problème je dois recompiler mon code et ensuite relancer le jeu.

V.Exercices et options non réalisés

Je n'ai pas réalisé les exercices et options présents dans le TP7 qui sont :

- utiliser le champs size du serpent lors de la capture des fruits et test de fin du jeu
- serpent grandit lorsqu'il mange un fruit
    - calcul de la position du nouvel élément
    - puis appel à *add_segment*
- ajouter plusieurs niveaux et passage au niveau suivant lorsque qu'on gagne
- ajouter la lecture de la position initiale du serpent directement depuis le fichier. s pour serpent et S pour tète. Il faudra vérifier la connexité du serpent pour valider le fichier. La direction sera la première case non-mur et non snake entre droite bas gauche et haut. Pour cette question il faudra ajouter les éléments dans l'ordre où on les lit, puis faire un tri. L'algo n'est pas simple dans le cas d'un serpent vertical avec la tête en haut ou en bas, ou même un serpent qui fait des virages...
- ajouter un item qui donne des "continus" et qui apparait a une case aléatoire différente d'un mur, d'un druit ou du serpent au bout d'un nombre de passage aléatoire dans la boucle de jeu.
- ajouter les options que vous voulez, remplacez les carrés de la grille par des sprites, ajoutez des sons...