RAPPORT

MATTHIEU PETIT

SOMMAIRE

Introduction

I. Règles

Snake, est un jeu vidéo dans lequel le joueur contrôle une ligne qui forme le corps d'un serpent.

Ce corps est en mouvement continu et la partie s'arrête lorsque la tête du serpent entre en contact avec un des bords de l'écran ou son propre corps.

Le joueur peut diriger le serpent selon les directions suivantes : Haut, Bas, Gauche, Droite.

Des pastilles et des obstacles sont présents sur la grille de jeu, la tête du serpent ne doit pas entrer en contact avec l'un des obstacles.

Si une pastille est mangée, le corps du serpent s'allonge de deux cases.

Pour passer au prochain niveau, le serpent doit manger toutes les pastilles présentes.

Chaque niveau présente une pastille et un obstacle de plus que le niveau précédent, et la vitesse du serpent est augmentée.

II. Sujet

Il nous a été demandé pour ce projet de réaliser notre version du Snake.

Notre programme doit comporter un menu permettant de :

- Lancer une partie
- Quitter le jeu
- Changer la taille de la grille de jeu
- Changer la taille initiale du serpent
- Changer le nombre de pastilles au premier niveau

Par défaut :

- La taille de la grille jeu est de 60 lignes par 40 colonnes.
- La taille initiale du serpent est de 10 cases.
- Le premier niveau comporte 5 pastilles et 0 obstacle.

Fonctionnalités

I. Menu

Le jeu comporte un **Menu** permettant de :

- **Lancer** une partie
- Afficher les **Meilleurs scores**
- Ouvrir les **Paramètres**
- **Quitter** le jeu

II. Jouer

Lorsque le joueur lance une partie, il arrive sur la grille de jeu et le serpent est placé au milieu de l'écran.

Le **score**, le **niveau** en cours et le **temps écoulé** sont affichés en bas de l'écran.

Lorsque la partie se termine, l'écran **Game Over** apparaît.

Score *Niveau* *Temps*

1. Game Over

Pseudo *Meilleurs scores*

Cette fenêtre affiche le **score** obtenu ainsi que le **niveau** atteint.

Le joueur peut retourner au **menu** en appuyant sur **Echap**.

Le bouton **pseudo** *(voir suite)* permet au joueur de changer son nom.

Le bouton **meilleurs scores** renvoie directement vers la fenêtre du même nom.

III. Meilleurs scores

Reset *Pseudo*

Cette fenêtre affiche les meilleurs scores, atteints par les précédents joueurs.

Ces scores sont lus à partir du fichier *scores.*

Le bouton **reset** permet de supprimer ce fichier.

Le bouton **pseudo** permet au joueur de changer le nom qui s'affichera dans cette même fenêtre.

1. Pseudo

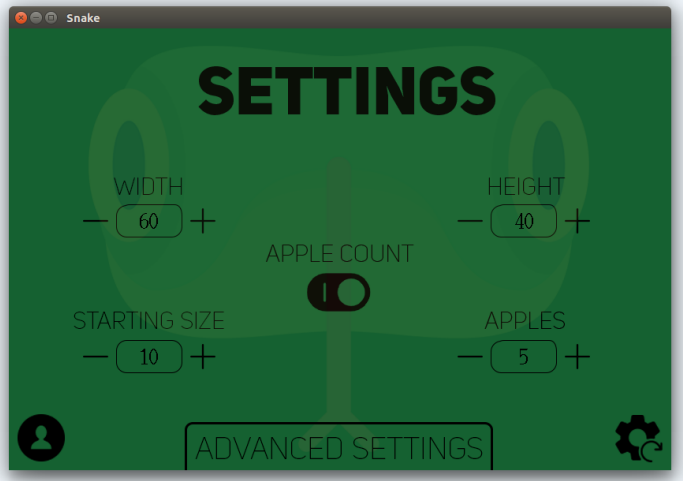
Annuler *Valider*

Le joueur a la possibilité de changer le nom qui l'identifie dans la fenêtre des **meilleurs scores**.

Le bouton **valider** et la touche **Entrée** permettent de valider le changement.

Le bouton **annuler** et la touche **Echap** permettent d'annuler tout changement.

IV. Paramètres



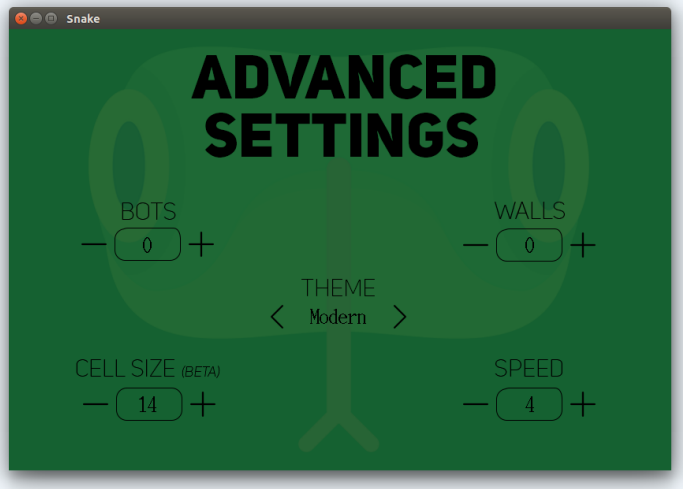
Pseudo *Reset*

Le joueur a la possibilité de modifier dans les paramètres respectivement, la **largeur**, la **hauteur**, la **taille initiale**, le **nombre de pastilles** présentes au niveau 1, et peut choisir d'afficher le **nombre de pastilles** qu'il a mangé.

Le bouton **pseudo** permet de changer le nom du joueur, et le bouton **advanced settings** permet d'accéder aux **paramètres avancés**.

Ces paramètres sont stockés dans le fichier *settings*. Le bouton **reset** permet d'effacer ce fichier et de mettre les paramètres par défaut.

1. Paramètres avancés



Cette fênetre permet de modifier respectivement, le **nombre de bots**, le nombre d'**obstacles** présents au niveau 1, la **taille d'une case** *(beta)*, la **vitesse** initiale du serpent, et le **thème** appliqué lors d'une partie.

Structure

I. Découpage

Par souci de cohérence et de lisibilité, l'entièreté du code est en anglais à l'exception des commentaires qui sont en français.

Ce projet est composé de 4 fichiers **sources** (.c) ainsi que leurs **headers** (.h) correspondants. Pour éviter les erreurs d'inclusion et par clarté, un fichier **header** supplémentaire contient l'intégralité des structures utilisées.

Globalement, chacun des fichiers sources correspond à une **fonctionnalité.**

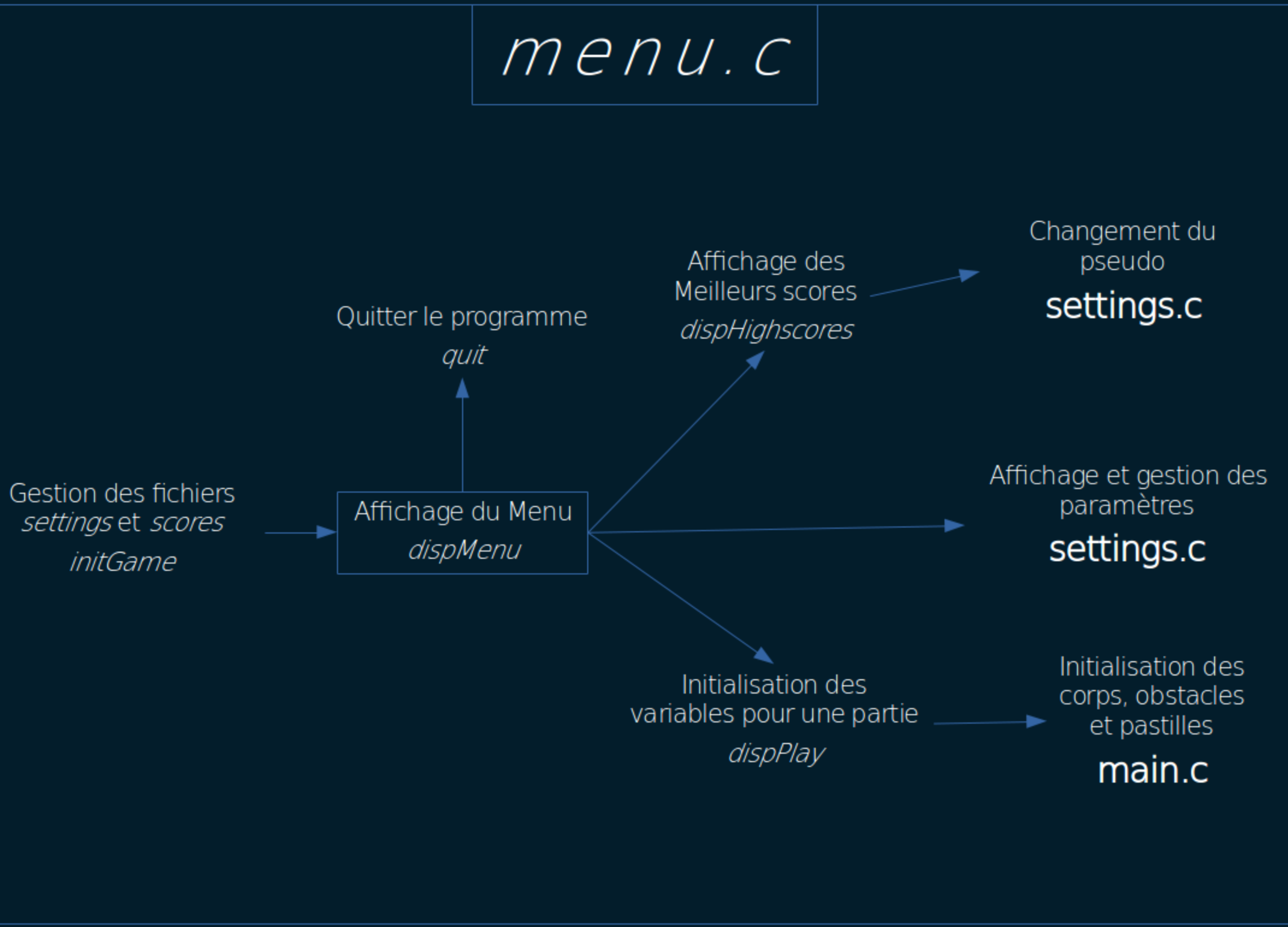
1. Menu

Le fichier **menu.c** contient toutes les fonctions nécessaires à l'écran du **Menu**.

Voici la liste des tâches effectuées par ces fonctions :

- Commencer l'éxecution et lire, ou créer le cas échéant les fichiers *scores* et *settings*
- Gérer l'affichage du **Menu**
- Initialiser tous les paramètres en cas de **lancement** d'une partie
- Afficher et gérer les **meilleurs scores** à l'aide du fichier *scores*
- **Quitter** totalement le programme

menu.c


Gestion des fichiers
settings et *scores*
initGame
Affichage du Menu
dispMenu
Quitter le programme
quit
Affichage des
Meilleurs scores
dispHighscores
Changement du
pseudo
settings.c
Affichage et gestion des
paramètres
settings.c
Initialisation des
variables pour une partie
dispPlay
Initialisation des
corps, obstacles
et pastilles
main.c

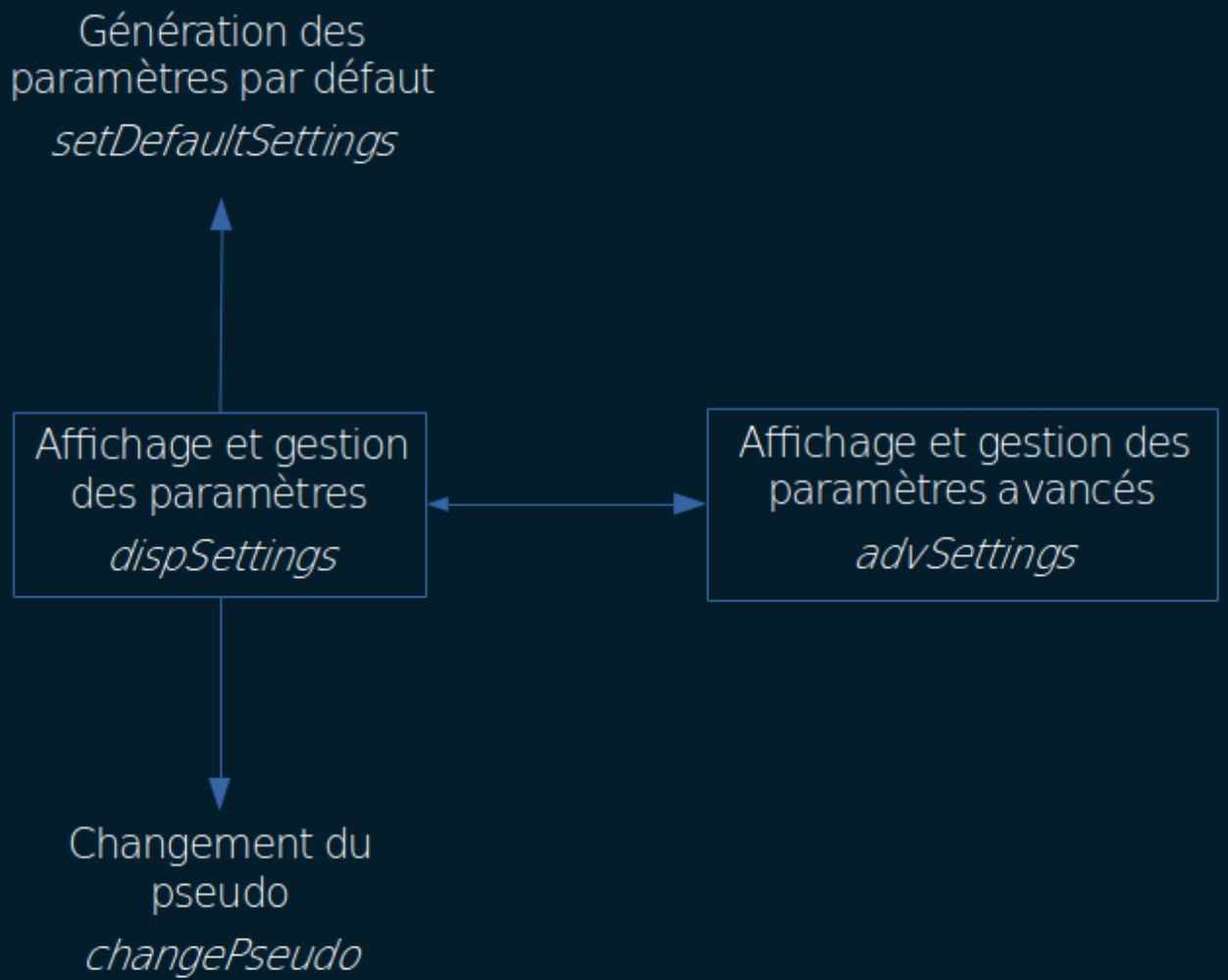
2. Settings

Le fichier settings.c contient toutes les fonctions nécessaires à la gestion des paramètres et des paramètres avancés.

Voici la liste des tâches effectuées par ces fonctions :

- Afficher les écrans des paramètres
- Effacer le fichier *settings* et générer les paramètres par défaut.
- Actualiser le fichier *settings* selon les paramètres modifiés par le joueur.
- Lire le fichier *settings* afin d'initialiser les paramètres sauvegardés
- Gérer le changement de pseudo
- Gérer les thèmes

settings.c

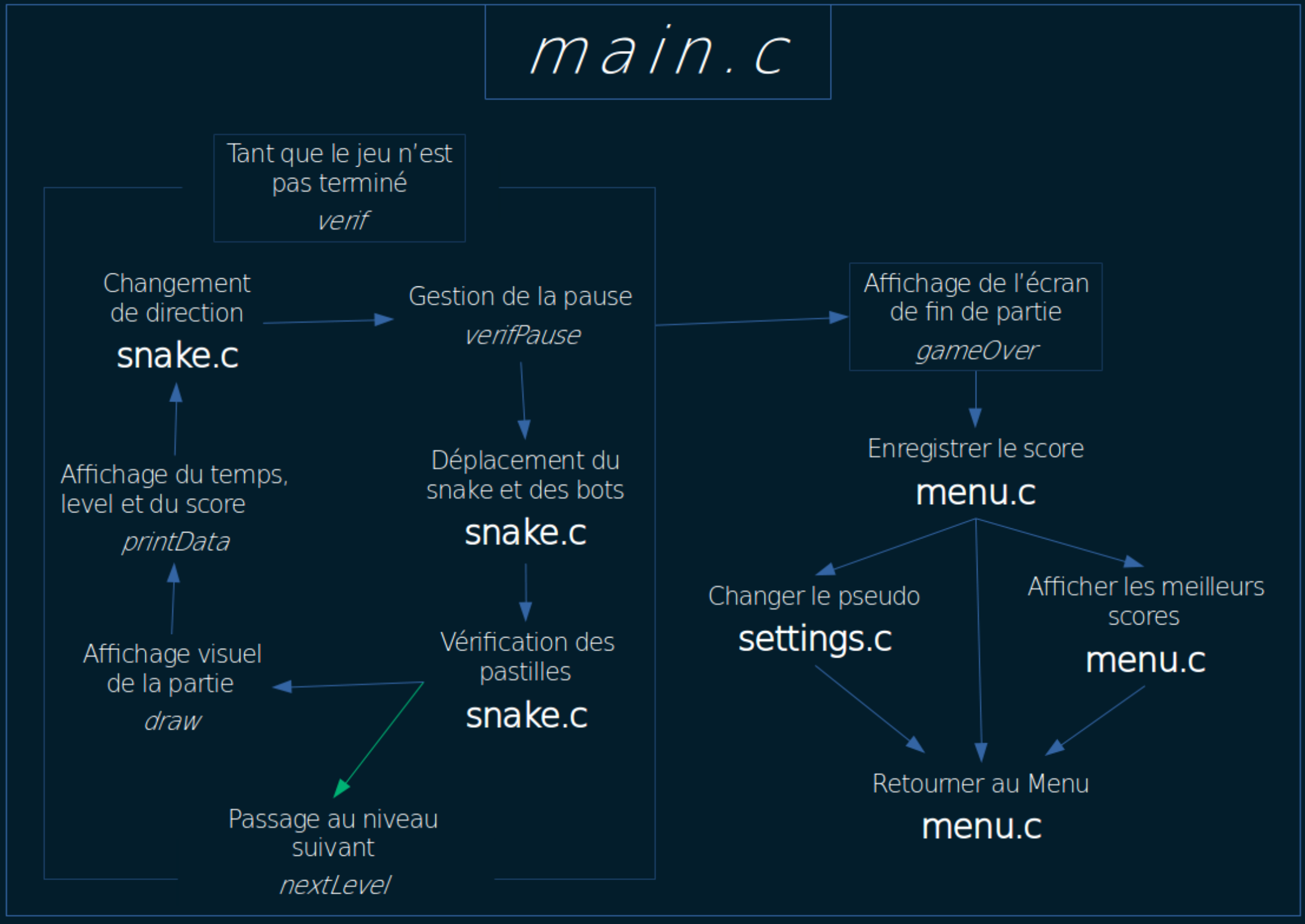

Génération des paramètres par défaut
setDefaultSettings
Affichage et gestion des paramètres
dispSettings
Affichage et gestion des paramètres avancés
advSettings
Changement du pseudo
changePseudo

3. Main

Le fichier main.c contient naturellement la boucle d'exécution principale.
Sont présentes dans ce fichier source les fonctions élémentaires telles que l'affichage de la partie et des données correspondantes, la gestion de la pause, le passage au niveau suivant et la fin de partie.

Ainsi, les fonctions suivantes sont présentes dans ce fichier :

- Le dessin de la grille de jeu
- L'affichage des données (score, niveau, temps, nombre de pommes *si activé*)
- La gestion du menu pause
- Le passage au niveau suivant
- Lécran de fin de jeu

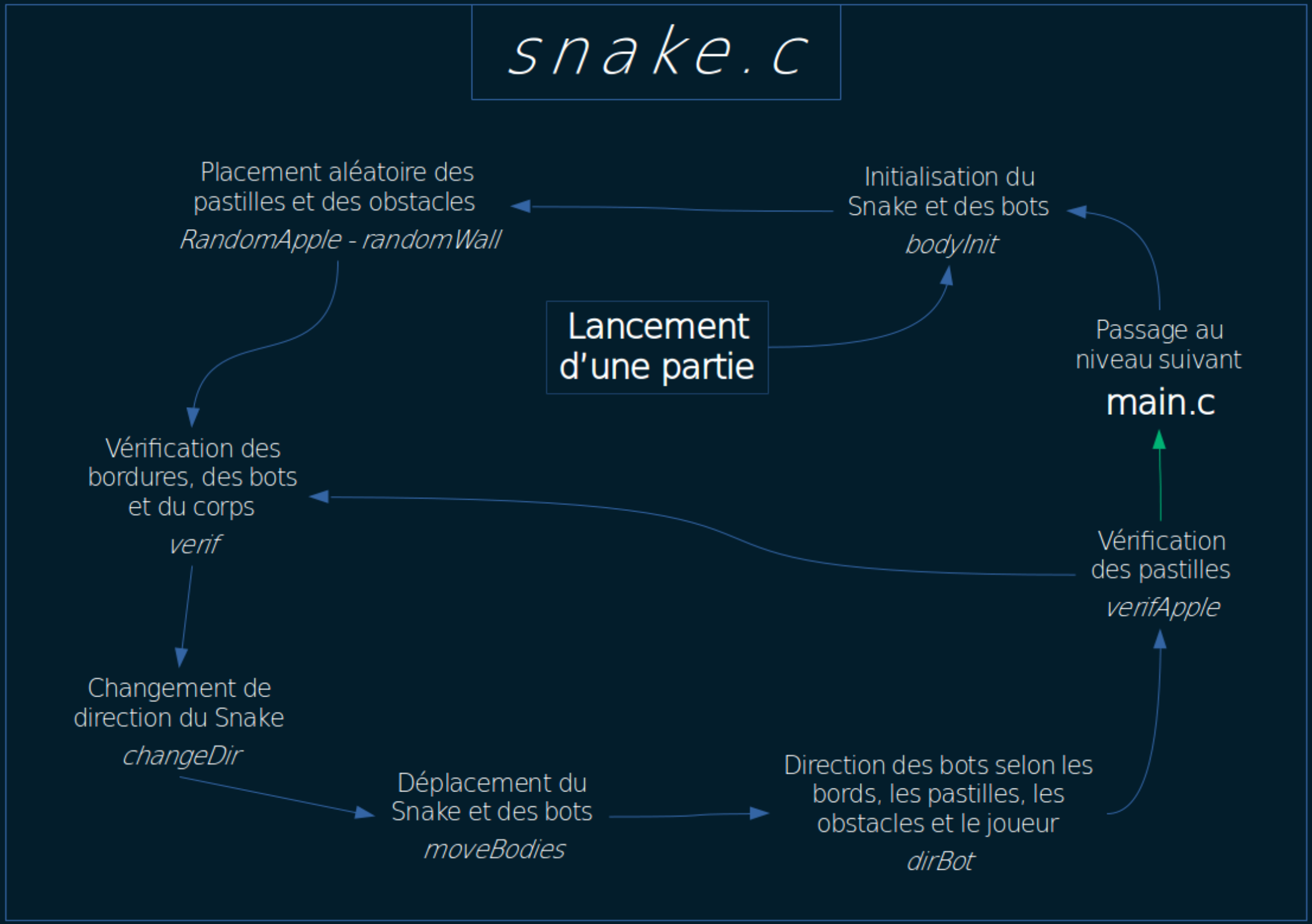

main.c
Tant que le jeu n'est pas terminé
verif
Changement de direction
snake.c
Gestion de la pause
verifPause
Affichage de l'écran de fin de partie
gameOver
Affichage du temps, level et du score
printData
Déplacement du snake et des bots
snake.c
Enregistrer le score
menu.c
Changer le pseudo
settings.c
Afficher les meilleurs scores
menu.c
Affichage visuel de la partie
draw
Vérification des pastilles
snake.c
Passage au niveau suivant
nextLevel
Retourner au Menu
menu.c

4. Snake

Le fichier snake.c est composé des fonctions qui gèrent l'initialisation et la gestion de différentes données telles que le snake, les bots, les obstacles et les pastilles.

Voici la liste des tâches effectuées par ces fonctions :

- Initialisation et déplacement du snake et des bots
- Déplacement des bots selon les bordures, la position des obstacles, du joueur et des pastilles.
- Vérification des bordures et des obstacles
- Génération aléatoire des obstacles et des pastilles


snake.c
Placement aléatoire des pastilles et des obstacles
RandomApple - randomWall
Initialisation du Snake et des bots
bodyInit
Lancement d'une partie
Passage au niveau suivant
main.c
Vérification des bordures, des bots et du corps
verif
Vérification des pastilles
verifApple
Changement de direction du Snake
changeDir
Déplacement du Snake et des bots
moveBodies
Direction des bots selon les bords, les pastilles, les obstacles et le joueur
dirBot

Données

I. Struct et tableaux

Les structures ont été utilisées autant que possible, afin de simplifier l'appel aux fonctions, améliorer la lisibilité et faciliter la gestion des données.

J'ai ainsi choisi pour ce projet de représenter le serpent sous forme d'une structure.

Cette structure est composée d'une vitesse, d'un nombre de segments, d'une direction et d'un tableau de segments, chaque case étant une structure composée d'une position x et une position y.

Ayant commencé ce projet avant le chapitre sur les listes chaînées, j'ai choisi d'utiliser les tableaux pour le stockage des données concernant le snake, les pastilles et les obstacles.

II. Modifications

Cette structure peut subir des modifications au cours de la partie.

En effet, lorsque le serpent mange une pastille, sa taille, et donc le nombre de segments se voit augmenter de 2. Ainsi, on alloue exactement la mémoire nécessaire à chaque étape du programme.

A chaque niveau, les bots (eux-aussi formés d'une structure similaire) obtiennent une case en plus. De plus, la vitesse de jeu est augmentée à chaque passage de niveau.

La direction est elle, définie par l'utilisateur au cours de la partie.

III. Représentation

Chaque segment est affiché individuellement et représente la taille d'une case -1 pixel en largeur et en hauteur, pour afficher une séparation entre les segments ; sachant qu'il y a un segment en tête qui comporte des yeux.

Un segment peut être présenté sous forme d'arrondi s'il est dans un tournant.

J'ai choisi de ne pas prendre d'images pour ces segments, afin d'avoir la liberté d'en modifier les couleurs avec les thèmes.

Conclusion

I. Apport personnel

Ce projet m'a permis d'apprendre beaucoup de choses par moi-même, et d'approfondir certains domaines intéressants.

J'ai par exemple appris à me servir de la lecture des fichiers en C pour pouvoir sauvegarder les paramètres, ainsi que les meilleurs scores.

J'ai approfondi ma connaissance de Photoshop(Windows) et de Photoflare(Ubuntu) pour réaliser les menus et arrières-plans.

J'ai aussi appris à mieux me servir de la suite LibreOffice.

GitHub a aussi été un outil très utile pour synchroniser et sauvegarder mon travail entre plusieurs appareils.

J'ai appris à réaliser des schémas pour clarifier ma pensée, avant de me mettre à coder sur machine.

Ayant commencé ce projet en avance, j'ai eu le temps de re-structurer plusieurs parties de mon code dans un objectif de clarté et de modularité, mais aussi d'ajouter quelques éléments qui n'étaient pas demandés.

De plus, j'ai essayé à chaque étape de simuler chaque action que le joueur peut avoir, afin d'éviter tout bug ou faille du programme. Ainsi, il ne devrait y avoir aucune fuite mémoire, ni faille de sécurité.

Finalement, j'ai su organiser mon code selon une norme que je me suis fixé et que j'ai essayé de respecter au fil du projet.

Pour finir, le sujet m'aura apporté une satisfaction importante ainsi qu'une envie de travailler et donner de mon temps pour un projet scolaire.

II. Ajouts

Voici la liste détaillée des éléments et fonctions que j'ai décidé d'ajouter.

- Un menu des paramètres avancés qui permettent de modifier de faire varier l'expérience de jeu (*voir Fonctionnalités*)

- Lecture et sauvegarde des paramètres et des meilleurs scores dans des fichiers

- Fenêtre de changement du pseudo

- Bots dirigés de façon semi-intelligente : si un obstacle, un mur ou un joueur se trouve à proximité, le bot va l'éviter. Au contraire, si une pastille se trouve à proximité, le bot va essayer de s'en rapprocher

- Ajouts de thèmes

- Ajout de modes de jeu masqués

III. Notes

Un thème peut être très facilement ajouté en suivant l'exemple donné dans le code.

→ Voir la fonction *choisirCouleur* dans le fichier settings.c

Voici la liste des modes de jeu :

`./snake zombie` `./snake minimoys` `./snake flash`

Il est aussi très simple d'ajouter un mode de jeu.

La variable 'opt' de la structure Game permet de stocker ces modes.

Il suffit de choisir un argument correspondant au mode de jeu choisi, puis, ajoutez votre mode jeu dans la fonction gameModes du fichier main.

→ Un exemple est aussi présent dans le code

Des #define DEV/DEBUG sont présents en haut des fichiers sources.

Activer ces define permet de faciliter grandement le développement et le debug.

IV. Crédits

Police utilisée pour le menu : Silombol

Police utilisée pour tous les chiffres et le reste du texte : 8Bit Wonder

Pommes (pastilles) : Dessin librement inspiré de Minecraft

Icônes : flaticon.com