19/12/2019

Rapport projet Snake

Baptiste Asselin et Anatole Pain

Table des matières

Introduction

Le but du projet était de faire une version du célèbre jeux Snake. Pour pouvoir avancer plus efficacement, nous nous sommes dès le début répartit le travail comme suit : l'un sur la gestion du Snake en mémoire en utilisant une liste chainée pour une meilleur optimisation et l'autre sur tout l'aspect graphique : c'est-à-dire l'affichage du Snake, son mouvement, mais aussi le timer et la gestion des touches.

Fonctionnalités du programme

Notre Snake démarre directement après l'exécution du programme, (mais attention sur certain PC une attente de 2 ou 3 secondes peux survenir avant que le Snake démarre, ce beug qui survient selon les machines n'a pas réussi à être résolu).

Le joueur peut aussi quitter le jeu à tout moment en appuyant sur échappe, le mettre en pause en utilisant la barre d'espace. De plus la vitesse du serpent s'accélère en fonction du temps.

Structure des données :

Les données qui sont stockées en mémoire et qui représente le serpent sont les coordonnées de chaque élément du serpent. Celles-ci sont stockées sous la forme d'une pile qui fonctionne d'après un système d'ajout d'une nouvelle valeur au début de la pile et de la suppression d'une valeur à la fin de la pile. Ainsi chaque élément pointe celui qui le précède jusqu'à l'élément qui correspond à la tête du serpent et qui lui pointe sur l'adresse NULL ;

La pile est organisée de cette façon :

- Une variable désignant les coordonnées en abscisse de l'élément.
- Une variable désignant les coordonnées en ordonnées de l'élément.
- Un pointeur vers l'élément qui précède.

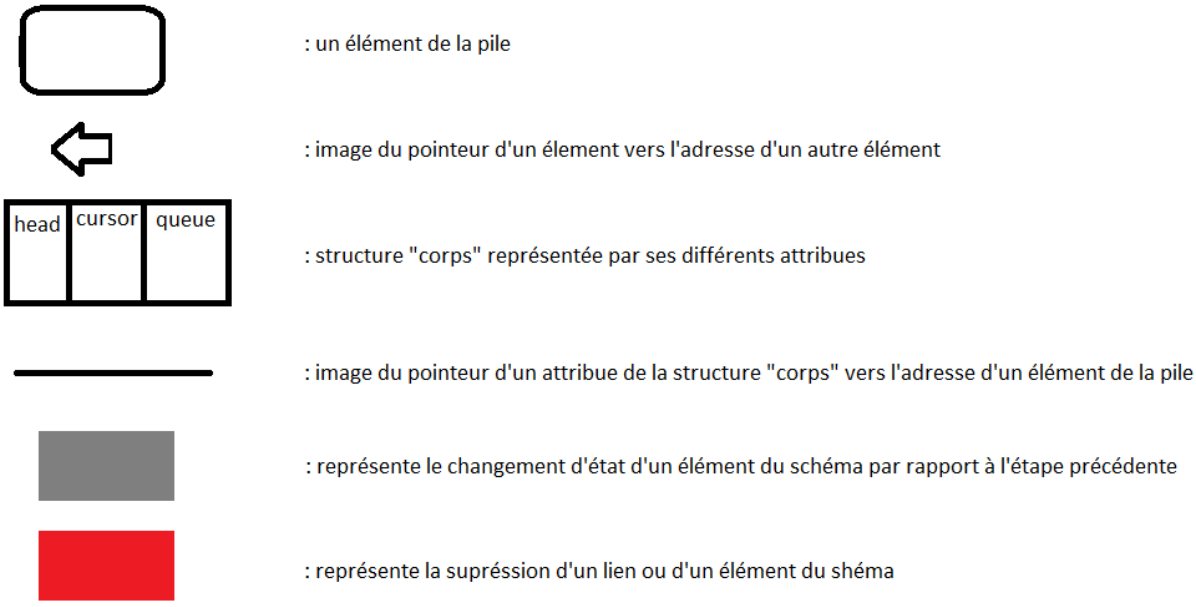
Toute la pile est gérée par une structure « corps » composé de trois pointeurs d'éléments, un pointeur pour la tête, un pointeur pour la queue et un autre pointeur qui sert essentiellement de mémoire tampon.

Les principales fonctions qui permettent de manipuler cette pile :

- enqueue() : qui permet d'ajouter un élément au début de la pile.
- dequeue() : qui permet d'enlever un élément à la fin de la pile.
- queueToHead() : qui permet de raccrocher qui ce situe l'élément à la fin de la pile et de l'ajouter au début de la pile.

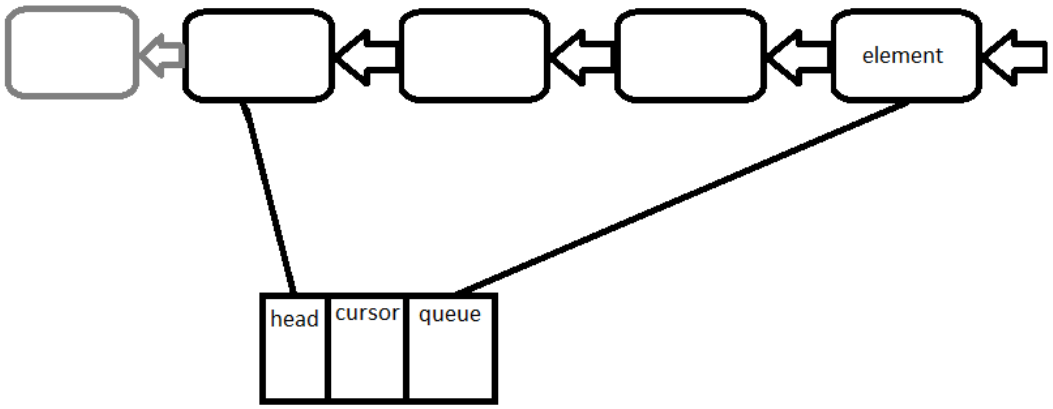
Sémantique de ces différentes fonctions :

Légende :



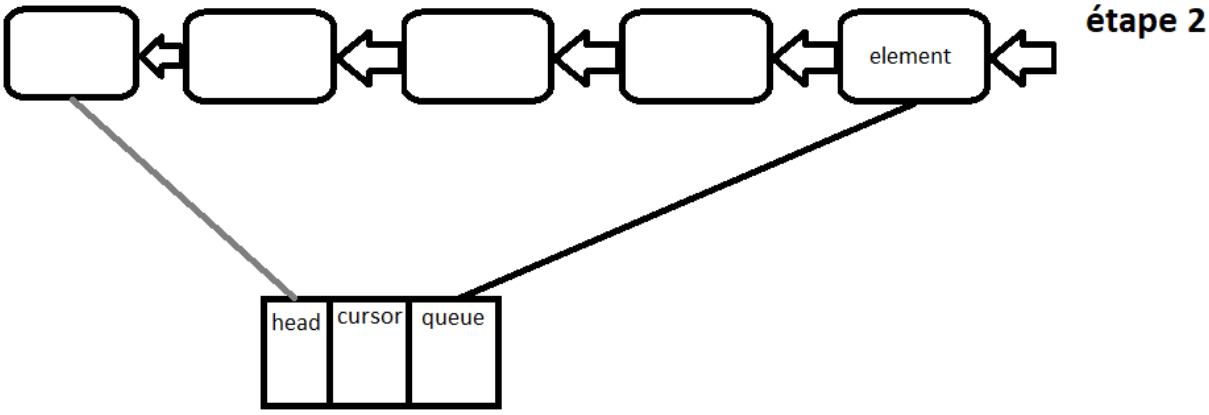
- enqueue() :

étape 1



On créé un nouvel élément qui est pointé par l'élément de tête actuel.

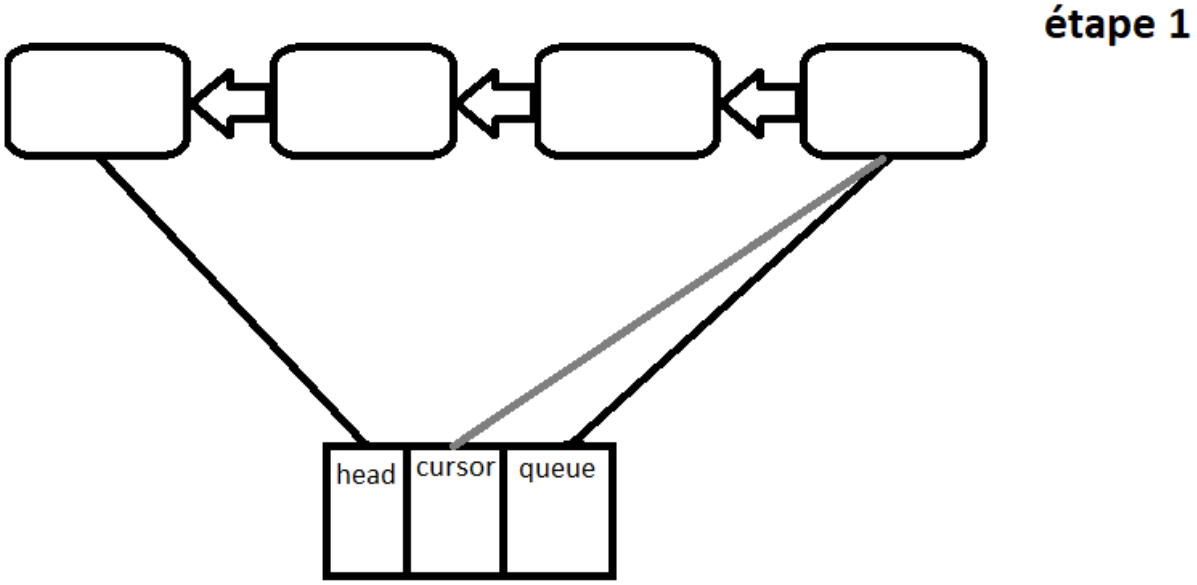
étape 2



On fait pointer le pointeur de la tête de la structure du corps sur la nouvelle tête.

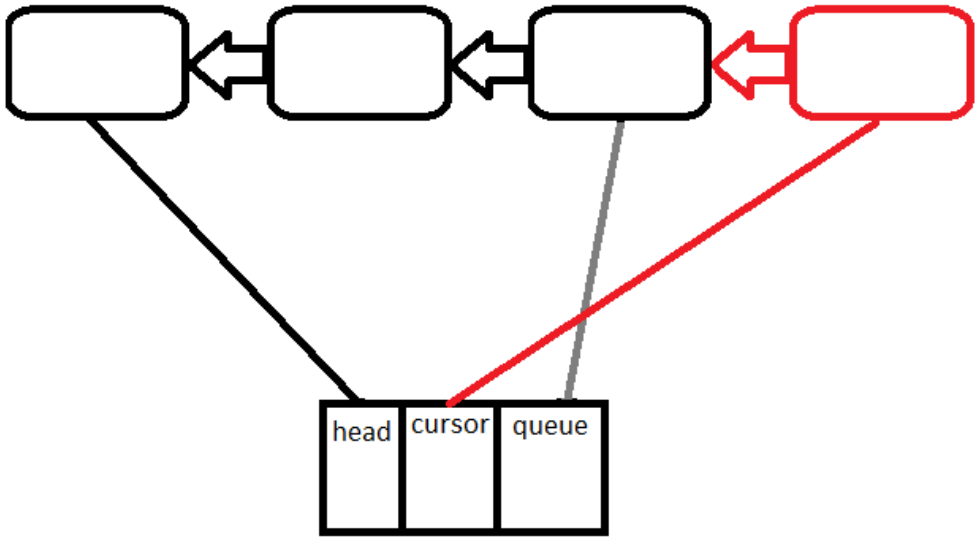
- dequeue() :

étape 1



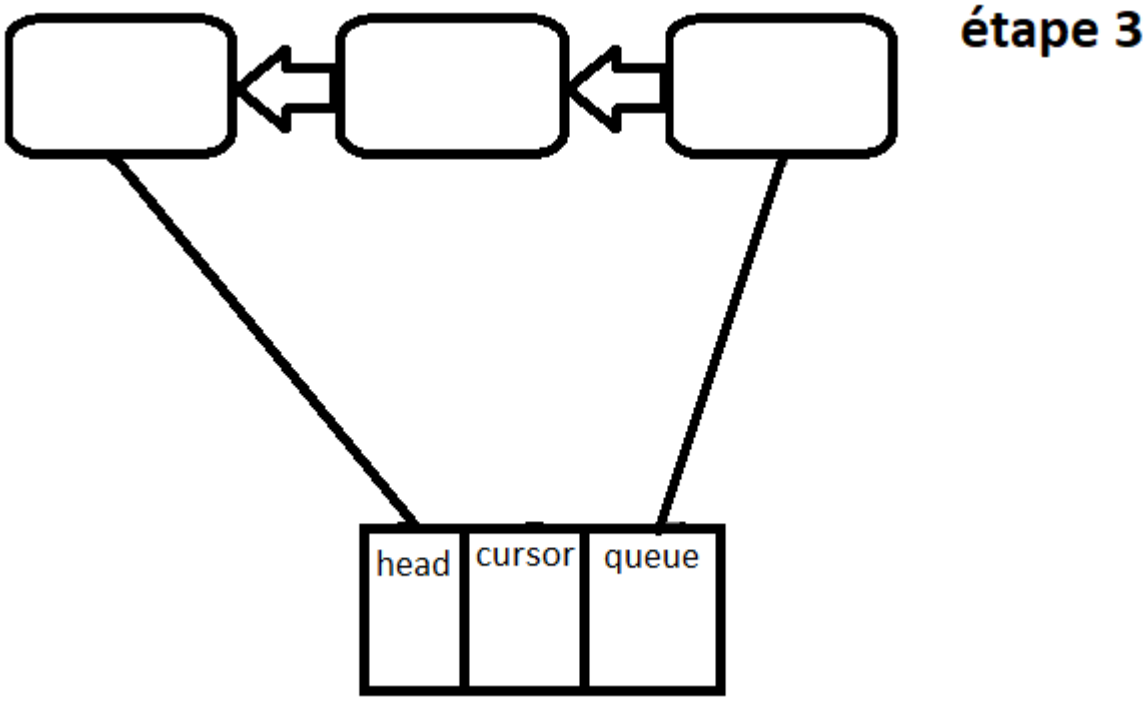
On fait pointer le curseur de la structure du corps sur la queue (le curseur prend la valeur du pointeur de queue)

étape 2



On fait pointer le pointeur de la queue de la structure du corps sur l'élément qui précède la queue et on libère l'élément qui est pointé par le curseur de la mémoire.

étape 3



On retrouve alors la pile et le corps dans leur état initial avec un élément en moins.

Autres fonctions relatives à la gestion de la pile :

Toutes les fonctions qui portent sur la gestion et l'accessibilité de la pile sont situés dans le fichier « snake.c » et leur sémantiques dénotationnels sont situés dans le fichier « snake.h »

C'est différentes fonctions sont :

- initSnakeBody()
- empty()
- firstEnqueue()
- initCursor()
- cursorNext()
- getHead()
- getCursor()
- setCoord()
- setHead()

Si vous souhaitez des informations concernant ces différentes fonctions je vous invite à lire le header et le code source cité plus haut.

Organisation des données relative à la grille de jeu :

La grille de jeu est matérialisée par une structure « grille » qui se compose d'un tableau à deux dimensions qui correspondent aux tailles demandées pour le terrain, soit 40 lignes pour 60 colonnes.

Chaque case du tableau prend une valeur précise défini par des constantes et qui correspondant aux éléments qui la compose.
par exemple :

- La valeur '1' défini par la constante 'GRASS' et qui désigne que la case est vide
- La valeur '2' défini par la constante 'SNAKE' et qui désigne une case avec un élément du serpent dessus.
- Etc…

Ce système permet de gérer les différentes collisions entre le serpent et les pommes ou les obstacles, les fonctions relatives à la gestion de la grille sont situées dans le fichier « grid.c » muni du header « grid.h ».

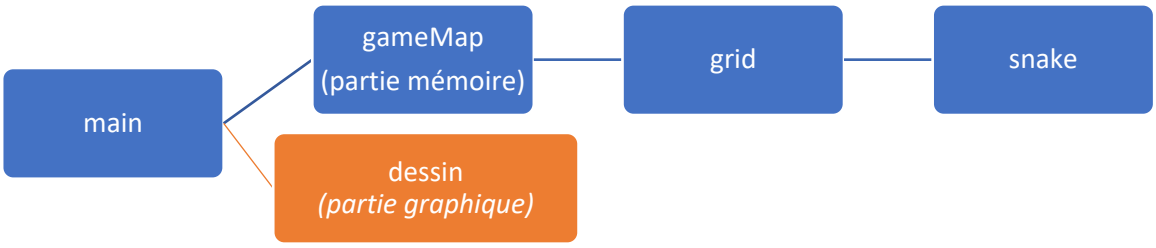
Ces quelques fonctions sont :

- initGrid()
- setValue()
- getValue()

la sémantique de ces fonctions se situe dans les fichiers (header et source).

Structure du programme :

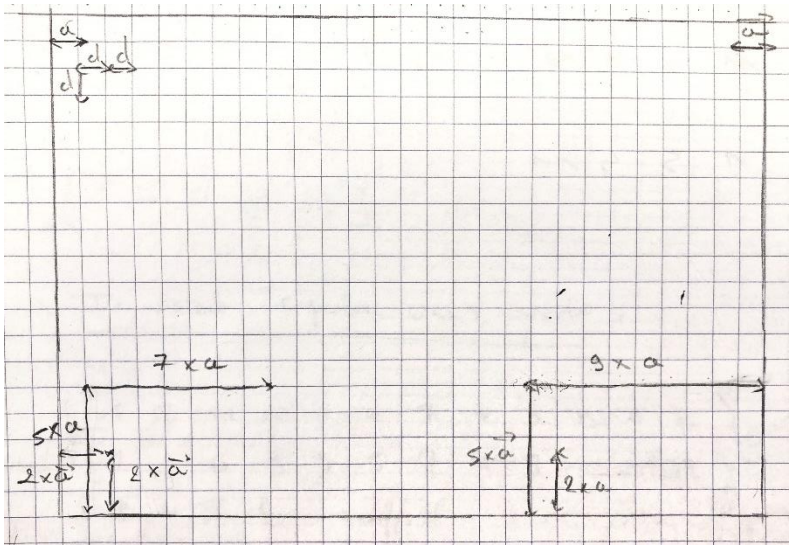
Le programme est composé de 5 fichiers sources qui sont organisé selon le diagramme suivant :



La partie mémoire du code est composée de trois fichiers organiser du plus proche au plus éloigné du système de stockage des données du programme :

- Le fichier « snake.c » contient les fonctions qui manipule directement la pile qui stocks les différentes positions des éléments du serpent.
- Le fichier « grid.c » contient les fonctions qui manipule directement la grille des collisions.
- Le fichier « gameMap.c » contient les fonctions qui manipule indirectement la grille et la pile par l'intermédiaire des fonctions de « grid.c » et « snake.c »

Le fichier « dessin.c » contient toutes les fonctions relatives à l'affichage des éléments du jeu comme le serpent, les pommes, le score, les cailloux et le timer. Le code a été organisé à l'aide de nombreuse structure créé dans le header et déclarer dans le main. Les fonctions utilisent ses adresses pour modifier les valeurs.



Dans la déclaration de segment l'idée était de faire un système de proportion qui varie en fonction de la largeur sélectionner (#define LARGEUR). Il fallait donc crée des segments : il y'a le « a » utile au proportion et le « b » qui correspond au côté des cases (Cf schéma). On peut donc facilement afficher le serpent en affichant chaque carré.

Après l'initialisation tous les structure au début, l'affichage de ce qui change se passe dans un boucle while() qui continue jusqu'à ce qu'on la quitte en utilisant **exitFunction()**.

Dans cette boucle pour faire avancer le serpant on utilise, comme pour afficher le temps, les microsecondes. Une boucle if() compare le temps actuelle et au bout de 66666 microsecondes le Snake se réactualise pour le faire bouger (environ 15 fois par secondes). Cette durée va être diminuer au cours du temps, se qui va avoir pour effet d'augmenter la vitesse du serpent.

Il faut aussi incrémenter ou décrémenter les coordonnés x ou y du Snake en fonction de la direction grâce à la fonction **sensDeDeplacement()** puis afficher un carré vert à la queue et un jaune a la tête.

De plus on doit vérifier en permanence les coordonnés de la nouvelle tête pour vérifier les collisions et en cas de pomme afficher deux carrés jaunes pour l'augmenter de 2 segments.

Pour mettre en pause le snake (fonction **pause()**) on bloque la boucle while() principale en faisant tourné une deuxième jusqu'à ce qu'on rappuie sur la barre d'espace.

Conclusion personnelle :

Baptiste Asselin,

Le développement de ce projet m'a permis de comprendre le fonctionnement assez avancé des différentes mécaniques du langage C, notamment avec les pointeurs qui mon permis de développer le système de stockage des données du serpent à travers une pile. De plus ce projet de groupe m'a également permis de m'initier au développement en groupe et à l'organisation que cela entraine (utilisation de git, mise en commun, répartition des tâches, etc…).

Anatole Pain,

Ce projet étant l'un de mes premier je pensais la tâche difficile mais j'ai été agréablement surpris de ma capacité à produire le jeu. Je pense avoir beaucoup progresser sur l'utilisation des adresses, des structures et des pointeurs. Cela m'a permis aussi a mieux organisé mon code.
Niveaux collaboration, l'organisation nécessaire, l'utilisation de git et de différente branche était très instructif.