IUT de Fontainebleau

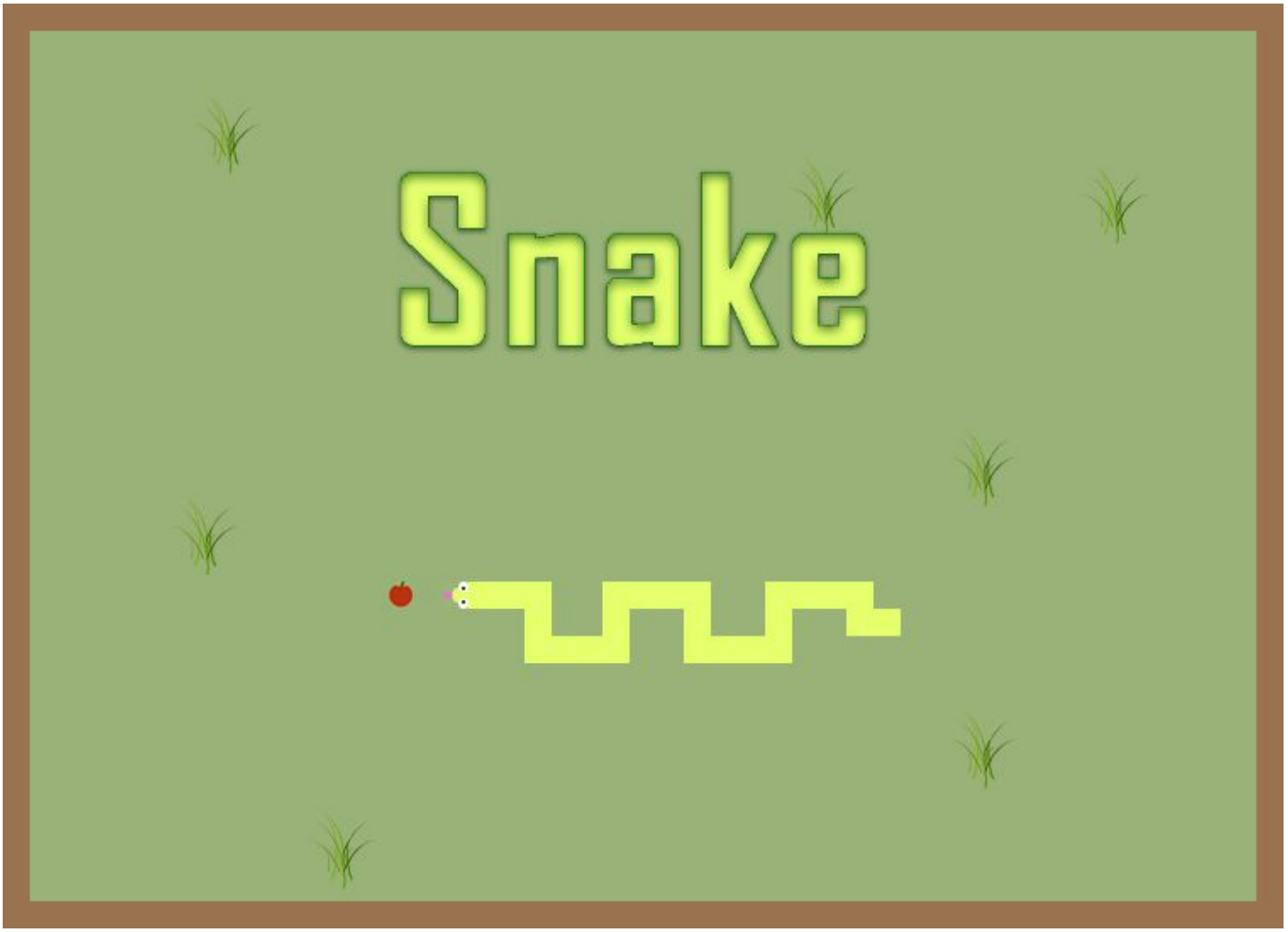
Département informatique

Année 2019-2020

Groupe 1

RAPPORT PROJET EN C:

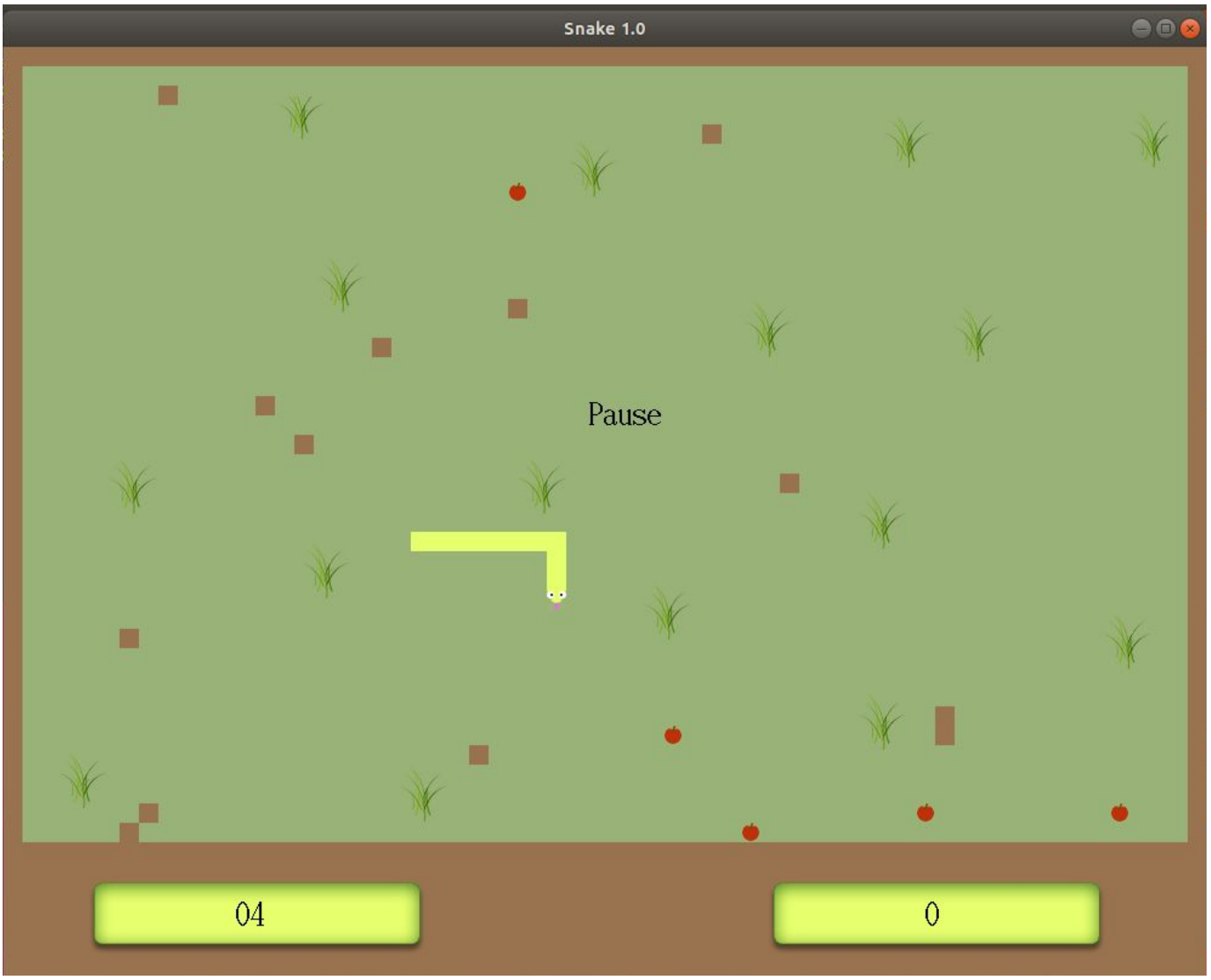
PROJET : SNAKE



Par : Bastien Barbieri
Marie Pellier

SOMMAIRE

INTRODUCTION

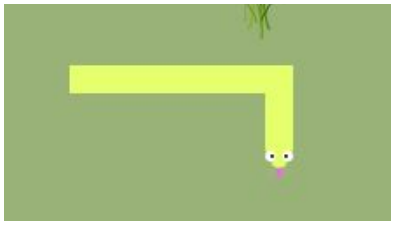


Pour ce projet nous devions réaliser un jeu en C , le jeu du Snake.
Un serpent qui bouge en continue et doit manger des pommes en évitant de se cogner contre des murs et sur lui même tout en sachant qu'à chaque fois qu'il mange une pomme il grandit. Les directions de celui-ci sont choisies par le joueur à l'aide des touches du clavier. Le jeu doit pouvoir se mettre en pause, compter le temps écoulé et le score.
Une ou plusieurs fonctionnalités devaient être ajoutées pour complexifier le jeu.

PRÉSENTATION

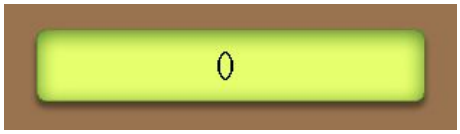
1) Fonctionnalités

Il nous été demandé d'avoir certaines fonctionnalités pour s'assurer du bon déroulement du jeu, les voici :



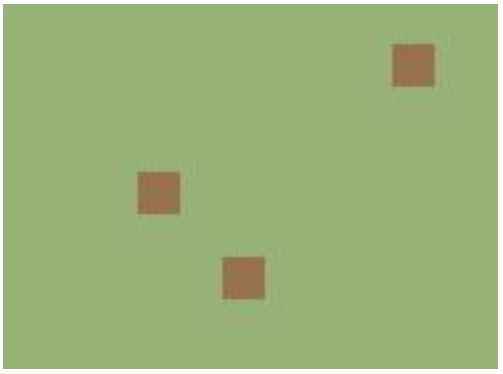
Le serpent : débute à 10 cases et s'agrandit de 2 à chaque pomme mangée, son déplacement est contenu. *Nous expliquerons son fonctionnement plus en détail plus tard.*

Les pommes : au nombre de 5, elles ont des positions générées aléatoirement (parmi 2400 cases disponibles).



Le score : s'incrémente de 5 en 5 à chaque pomme mangée.

Le chronomètre : affiche le nombre de secondes écoulées depuis le début du jeu.



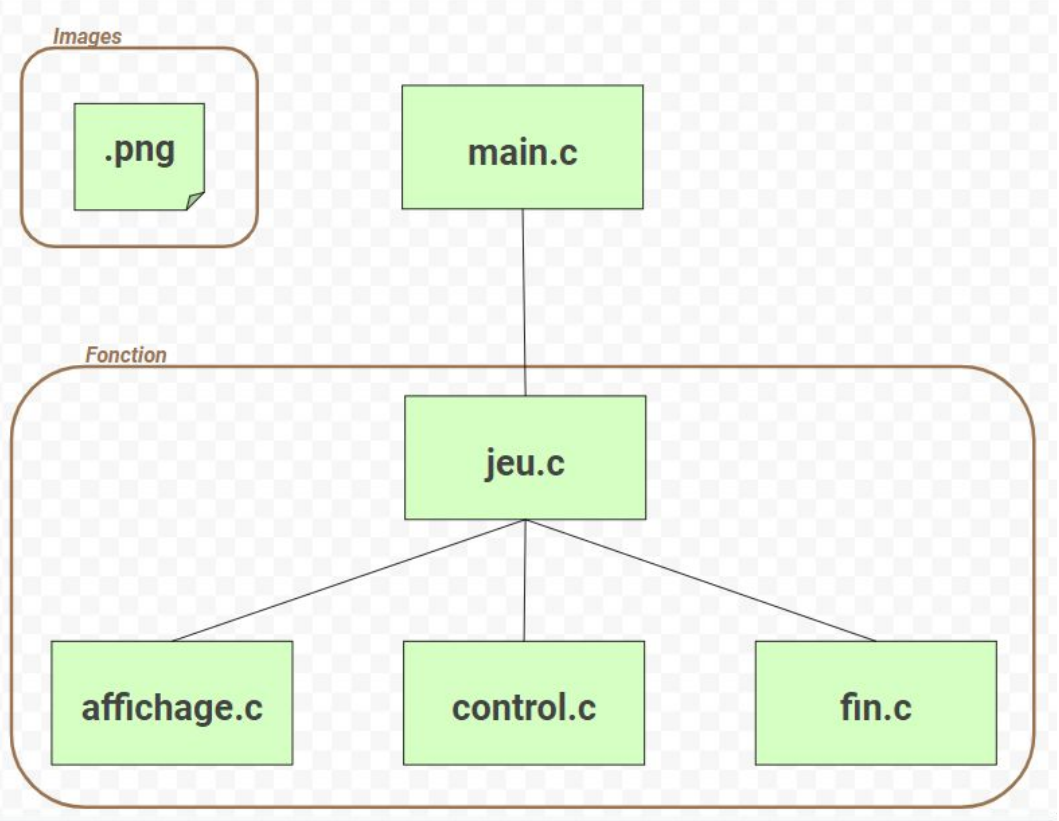
Les obstacles : à chaque partie, des obstacles (entre 14 et 6 inclus) sont générés à des positions aléatoires (non occupées par le serpent ou les pommes). A chaque rafraîchissement ils peuvent ou non se déplacer de manière aléatoire sur le terrain. Un obstacle peut se trouver juste devant la tête du serpent.

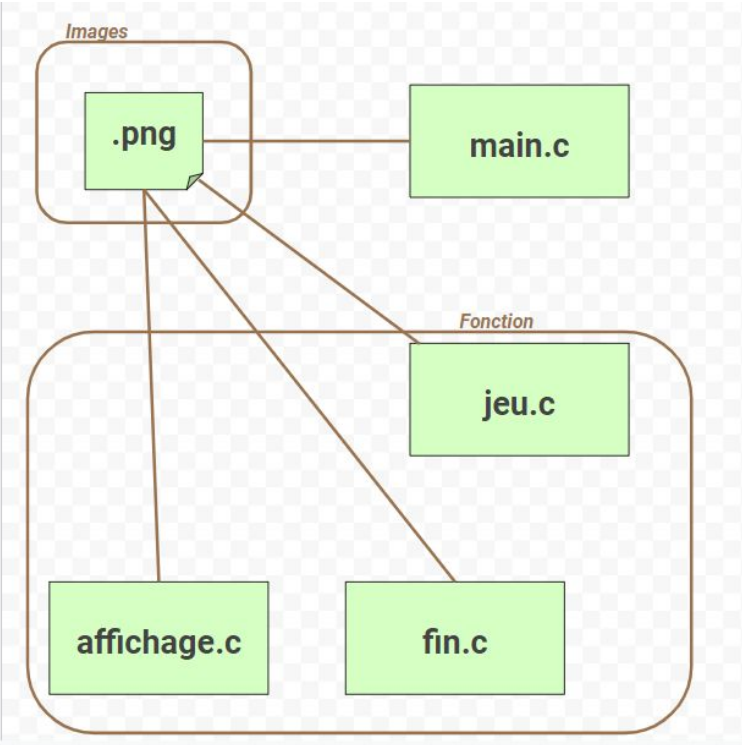
La pause : en appuyant sur espace, le déplacement du serpent et du chrono se stoppent. La pause peut être désactivée en appuyant à nouveau sur espace, mais aussi en appuyant sur une touche de direction.

L'écran de fin : le jeu n'ayant pas de véritable fin, seul un écran de "game over" s'affiche si le serpent se cogne. Cet écran affiche le score ainsi que le temps écoulé (en seconde).

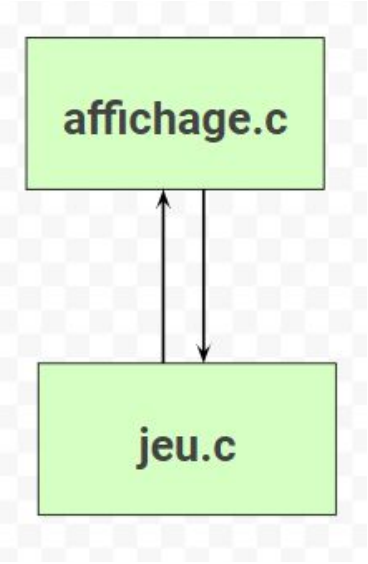
2) Structure

Notre programme est divisé en plusieurs dossiers qui contiennent chacun un type de fichier et donc un rôle spécifique. Il y a le dossier image qui contient les png qui seront affichés dans le jeu, le dossier fonctions où l'on retrouve toutes les fonctions nécessaires au déroulement du jeu.

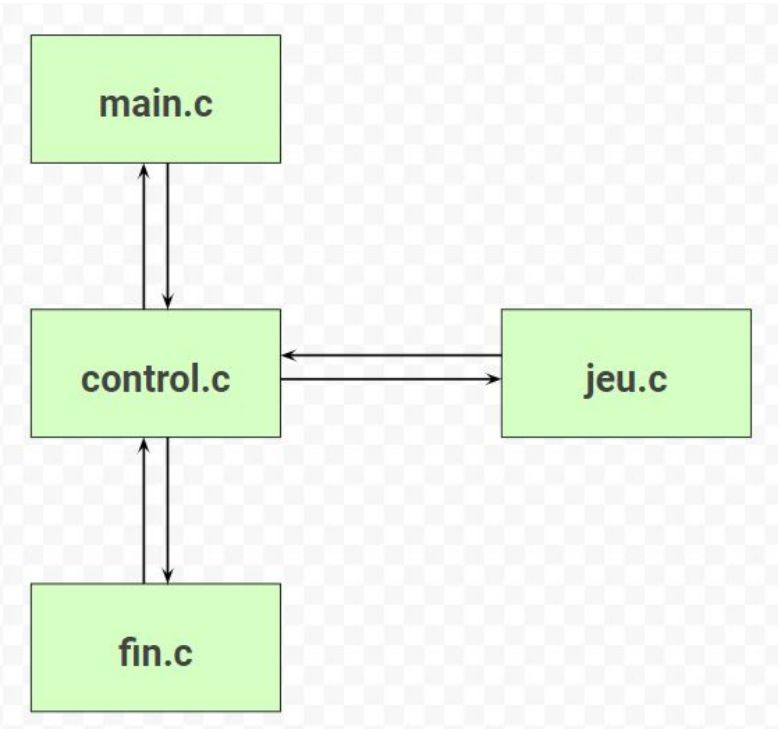




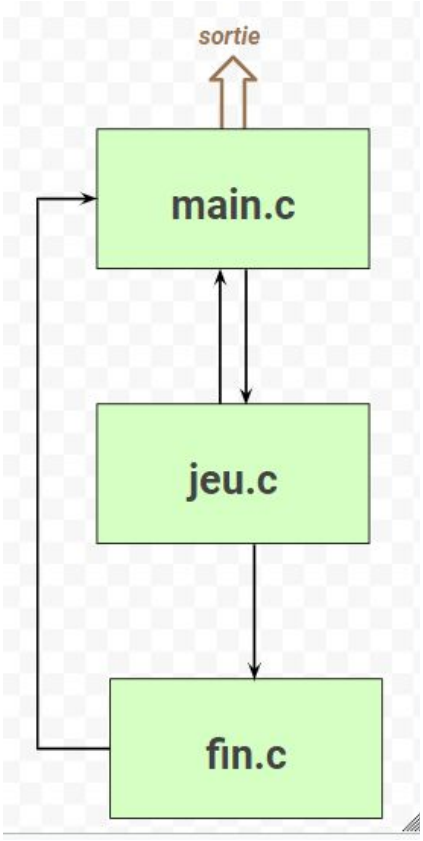
Les fichiers du dossier images sont sollicités lors du déroulement du jeu. En effet, en fonction du menu (dans main.c), de plateau de jeu (dans jeu.c) ou de l'écran game over (fin.c), on va charger une image de fond différente. Le fichier affichage.c contient la fonction *affiche_sprite_jeu()* qui va afficher les sprite un à un sur le plateau de jeu.



La fonction *jeu()* (contenue dans jeu.c) appelle à chaque itération de la boucle while, la fonction *affiche_sprite_jeu()*. En faisant cela, elle lui passe plusieurs variables en argument : les variables de chargement des sprites et le tableau général (le fonctionnement du tableau sera détaillé plus tard). Comme dit plus tôt, la *affiche_sprite_jeu()* va afficher les sprites qui auront été chargé et donné en argument. Elle va aussi afficher le fond du plateau du jeu… Et compter le nombre de pomme présentes sur le plateau ! En effet, à chaque foi qu'elle affichera une pomme, la variable *nombre_pomme_presentes* va s'incrémenter et sera renvoyée à la fonction *jeu()* afin d'ajuster le nombre de pomme.



La fonction ChoixTouche() (contenue dans control.c) renvoie un entier qui permet de savoir quelle touche a été appuyée. Ainsi, cette fonction est appelée à chaque étape du jeu : le menu, le plateau et la fin pour pouvoir quitter à tout moment. En argument, elle reçoit la valeur de la dernière touche utilisée (cela permet de savoir si le jeu est en pause et de retirer la pause si la touche précédente était déjà la pause). Ainsi que la valeur de la touche en attente (utilisée en continue tant qu'il n'y a pas eu de changement).



Afin de pouvoir quitter, le programme doit repasser à l'écran de départ car c'est là que la fenêtre s'ouvre/se ferme avec les fonctions de la bibliothèque graphique graph.h. Ainsi, à chaque fois que les fonctions jeu() et fin() reçoivent la valeur de la touche echap, qui est de 0, leur boucle while sera quitté une à une jusqu'à retourner à la première étape qui sera elle aussi quittée.

FONCTIONNEMENT

1) En général

Afin de faire fonctionner notre jeu, nous avons décidé de réaliser un tableau qui contient des indices. Chacun de ces indices correspond à un affichage spécifique (case libre, obstacle, serpent, pomme…). Comme vu précédemment, nous avons réalisé une fonction qui va parcourir le tableau à chaque itération de la boucle while et va rafraîchir l'affichage du plateau en fonction des changements d'indices.

Ainsi, un tableau de 62x42 va être initialisé, sa première ligne/colonne et sa dernière ligne/colonne seront remplies de 5 (=murs), laissant 60x40 cases libres pour le jeu en lui même :

	1	2	3	4	5	6
1	5	5	5	5	5	5
2	5	0	0	0	0	5
3	5	0	0	0	0	5
4	5	0	0	0	0	5
5	5	5	5	5	5	5

Ensuite, les indices 2 (=corps du serpent) et 3 (=tête du serpent) vont être placés à des places prédéfinies dans le tableau à partir de deux tableaux (un pour les x et l'autres pour les y afin de pouvoir déplacer le serpent correctement par la suite : *Nous verrons le déplacement du serpent plus en détail plus tard.)*

	1	2	3	4	5	6
1	5	5	5	5	5	5
2	5	0	0	0	0	5
3	5	0	2	3	0	5
4	5	0	0	0	0	5
5	5	5	5	5	5	5

Après, un rand sera effectué entre 14 et 6 afin de choisir le nombre d'obstacles présent sur le jeu. Deux autres rand seront fait pour leur positionnement parmi les cases libres. Si une case est déjà occupée, alors le rand se refait. Si la case est libre, alors on attribue 5(=mur) en valeur à la case :

	1	2	3	4	5	6
1	5	5	5	5	5	5
2	5	0	0	0	0	5
3	5	0	2	3	0	5
4	5	0	0	0	5	5
5	5	5	5	5	5	5

Enfin, pour le même principe que les mur, des indices 4 (=pommes) seront attribués de manière aléatoire parmi les cases disponibles. Une variable comptant le nombre de pomme présentes sera indentée. Elle sera réduite quand l'affichage ne comptera que 4 pommes à partir du moment où l'indice de la tête du serpent remplacera celui de la pomme. Un nouveau rand de pomme sera alors effectué.

On obtient ainsi un tableau global comme ceci (échelle réduite) :

	1	2	3	4	5	6
1	5	5	5	5	5	5
2	5	4	0	0	0	5
3	5	5	2	3	0	5
4	5	0	0	0	5	5
5	5	5	5	5	5	5

5=mur/obstacle, 4=pomme, 2=corps du serpent, 3=tête du serpent, 0=case libre

Des conditions seront vérifiées au cours du jeu et provoquent des événements spécifiques. Par exemple : si une case d'indice 5 (mur/obstacle) se fait remplacer par la case d'indice 3 (tête du serpent) alors le serpent meurt.

2) Le serpent

En ce qui concerne le serpent nous utilisons deux tableaux de 2400 cases chacun. Un pour stocker les abscisses et l'autre pour stocker les ordonnées des différentes cases du serpent. Le couple composé de la première case de chaque tableau correspond aux coordonnées de la tête du serpent. Dans notre programme on initialise les 10 premières valeurs de chacun de ces tableaux puis on s'en sert pour renseigner le tableau principal (de taille 62*42) sur les endroits où placer les indices du serpent.

C'est sur ce même principe que fonctionne le déplacement du serpent :

Au sein des deux tableaux du serpent à chaque tour de boucle de while, chacune des cases prends les coordonnées de la case précédente (chaque case va avancer et prendre la place de la case devant elle) et on remonte tout le tableau jusqu'à la tête. Cette dernière voit ses coordonnées évoluer en fonction de la direction du serpent.

Bouger à droite revient à ajouter 1 à l'abscisse de la tête du serpent
C'est à dire augmenter de 1 la valeur de la première case du tableau qui stocke toute les abscisses.
Idem pour les autres directions si ce n'est qu'il faut modifier dans le tableau des ordonnées pour aller en haut et en bas.
Le mouvement de la tête se propage ensuite aux autres cases à la boucle suivante.

Les deux tableaux communiquent au tableau à deux dimensions les nouvelles cases où se trouvent le serpent pour modifier l'affichage. Et on reproduit ce schéma à chaque tour de boucle de while.

CONCLUSIONS

Marie Pellier :

Le déroulement de la conception du projet fut assez complexe; Nous nous sommes réparti les tâches, mais mon binôme a eu du mal à être productif... La date butoire approchant, je devais donc avancer sans son aide. Mais il m'a quand même bien aidé sur la fin. Ce qui est dommage car avec une meilleure gestion et avec nos capacités on aurait pu optimiser et améliorer le programme. Je pense aussi que j'aurai du être plus attentive vis à vis de mon binôme afin de le motiver à communiquer davantage avec moi sur l'avancé de son travail.

Bastien Barbieri

J'étais en charge de la partie technique sur le projet et j'aurai du commencer par une structure moins ambitieuse et ensuite seulement aborder des structures plus complexe pour permettre à mon binôme d'avancer de son côté plus rapidement. Je suis tout de même content de la structure actuelle qui fonctionne malgré une perte de mémoire.