M1103 : Projet de Programmation

DUCK HUNT

NÉDÉLEC Guillaume

--

Viola Gaëtan

Groupe B

Année
2015-2016

Enseignant Responsable : Nicholas Journet

Sommaire

I. Introduction

1.1. Contexte

Ce projet s'inscrit dans le cadre des projets du modules M1102 et M1103 en tant que second travail.
Ce travail est pris en compte dans l'Unité d'Enseignement des Bases de l'Informatique du Semestre 1.
Ce projet en binôme a débuté début novembre (vers le 6/11/15) et doit être achevé pour le 17 décembre.

1.2. Objectifs

L'objectif était de réaliser correctement le jeu avec toutes les animations ainsi que toutes les fonctionnalités. Si tout cela était parfaitement réalisé, nous voulions apporter des améliorations du jeu, des nouvelles fonctionnalités et un nouveau mode de jeu.

1.3. Outils utilisés

Pour ce projet nous avons essentiellement utilisé le langage de programmation C++ ainsi que la bibliothèque SDL. Nous avons développer ce projet à l'aide de l'IDE QTCreator.

II. Implémentation

2.1. Architecture générale

Nous avons choisi de ne mettre quasiment rien mettre dans notre fichier main.cpp. Nous avons pour cela effectue un découpage important. Nous avons x fichiers ".cc" chacun associés à un fichier ".h" qui contient les entêtes des fonctions associées et pour certains, la déclaration de structures ou de constantes. *(fichier rouge correspondant au mode secondaire)*

main.cpp : Contient seulement un appel des fonctions contenant le menu et les différents modes du jeu.

boucle.cpp : Contient nos 4 boucles d'affichage. Il s'agit de menu, du mode normal, du mode de jeu secondaire, et du menu des meilleurs score.

canard.cpp : Contient toutes les fonctions relatives aux canards ainsi que leur structures (leurs affichage, leurs déplacements, leurs initialisation)

chicken.cpp : Contient toutes les fonctions relatives aux poulets ainsi que leur structures (leurs affichage, leurs déplacements, leurs initialisation)

chien.cpp : Contient toutes les fonctions relatives aux animations du chien et à leur positionnement dans la fenêtre (quand il marche et saute dans les herbes au début de chaque niveau, quand il rigole ou quand il ramasse un ou plusieurs canards après chaque phase de jeu).

evenement.cpp : Contient toutes les fonctions d'évènements pour chaque mode du jeu. (touches au clavier, clics de souris, ect…)

gestionFichier.cpp : Contient toutes les fonctions relatives aux lectures et à la modification de fichiers texte (essentiellement pour les meilleurs scores)

gestionScore.cpp : Contient toutes les fonctions permettant le calcul et l'affichage des scores (affichage du nombre de cartouches, du score, du tableau de canards central, des points pour le mode secondaire, ect…)

image.cpp : Contient toutes les fonctions relatives aux chargements des images, leur affichages et également l'affichage des fontes.

2.2 Structure de données

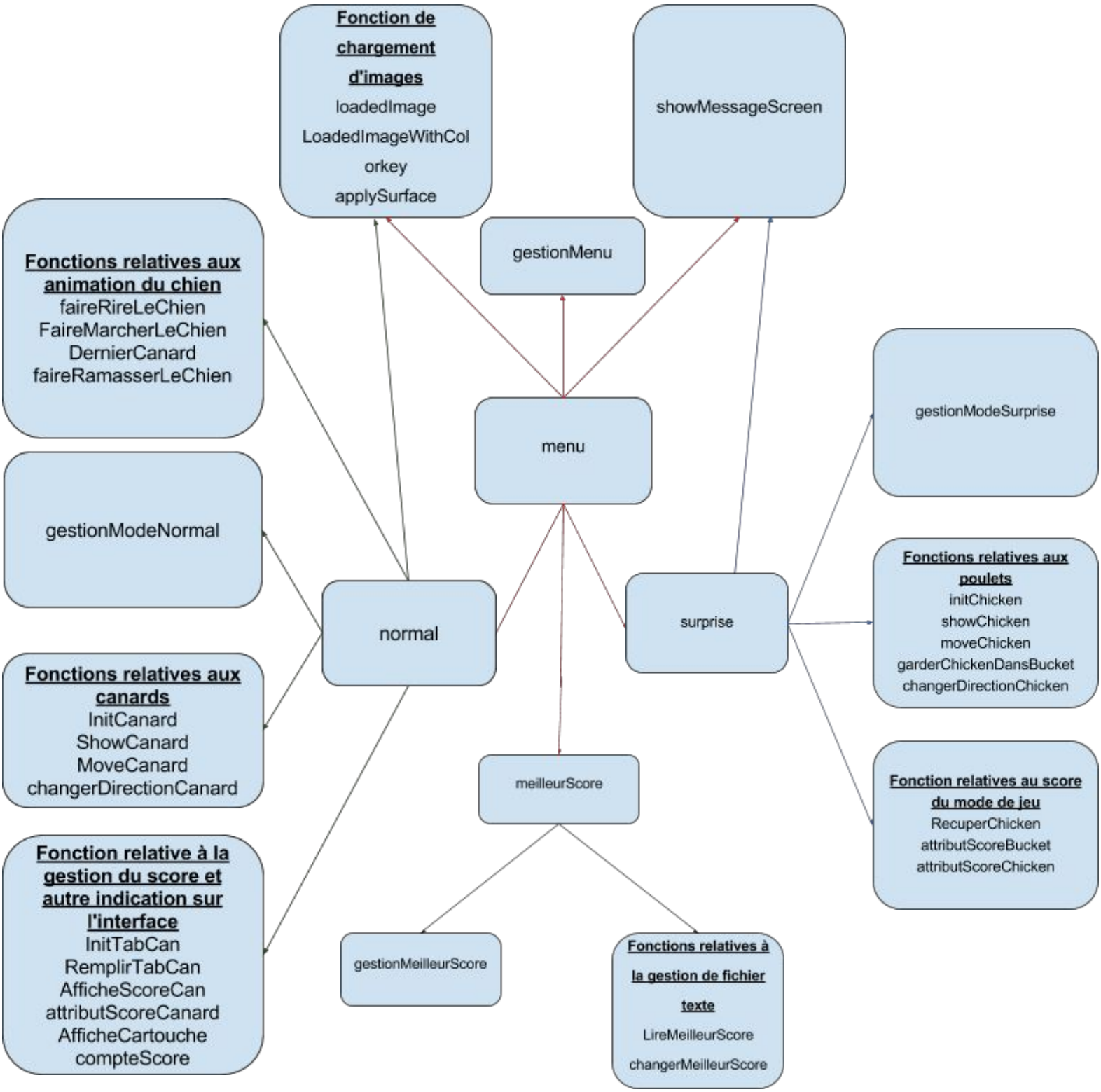
Nous avons utilisé 2 structures différentes dans notre projet :

- Pour les canards (mode classique)
- Pour les poulets (mode secondaire)

```
struct canard {
    int posX; -----> La position du canard sur l'axe des abscisses
    int posY; -----> La position du canard sur l'axe des ordonnées
    int w; -----> La largeur du canard
    int h; -----> La hauteur du canard
    int couleur; -----> La couleur du canard (0: noir, 1: rouge, 2: bleu)
    float vitesse_x; -----> La vitesse du canard sur l'axe des abscisses
    float vitesse_y; -----> La vitesse du canard sur l'axe des ordonnées
    int mort; -----> Sert de compteur jusqu'à que le canard soit touché
    bool tuer; -----> Indique si le canard est toujours en vie ou non
    bool sorti; -----> Indique si le canard n'est plus visible à l'écran (quand toute
                       les cartouches sont épuisées ou passage derrière le décor)
    bool scorer; -----> Permet  au score d'augmenter une seule fois pour chaque
                        canard
};

struct chicken {
    int x;  -----> La position du poulet sur l'axe des abscisses
    int y;  -----> La position du poulet sur l'axe des ordonnées
    int w; -----> La largeur du poulet
    int h; -----> La hauteur du poulet
    float vitesse_x; -----> La vitesse du canard sur l'axe des abscisses
    float vitesse_y; -----> La vitesse du canard sur l'axe des ordonnées
    int mort;  -----> Sert de compteur jusqu'à que le poulet soit touché
    bool tuer; -----> Indique si le poulet est toujours en vie ou non
    bool sorti; -----> Indique si le poulet n'est plus visible à l'écran (quand il est
                       passé derrière le décor)
    bool entreDansLeBucket; -----> Permet d'augmenter le score quand le bucket
                                   récupère une cuisse de poulet
    bool dansLeBucket; -----> permet de conserver l'affichage de la cuisse dans le
                              bucket même s'il bouge
    bool scorer; -----> Permet au score d'augmenter une seule fois quand on touche
                        le poulet
};
```

2.3 Organigramme d'appel des fonctions



III. Aspects Algorithmiques

3.1. Algorithme 1 : La fonction remplir tabCan

C’est un fonction qui va nous servir pour remplir un tableau qui lui même nous servira pour l’affichage des icones de canards. La premiere dimension du tableau a une taille de 5, elle correspond au nombre de round par niveau. La seconde dimension du tableau a une taille de 2 qui correspond au nombre de canard par round. Toutes les cases de se tableau sont initialisée à 0.
On passe en parametre le tableau, les booléens “tuer” de chaque canard, le round actuel, le nombre de tir.
On a donc le round, on lui enlève 1 pour l’utilisé pour la première dimension du tableau (tabCan[round][ ]).
Ensuite on test si les canards sont tués avec les booléens passé en parametre. Si c’est le cas la case correspondante au canard prends la valeur 1.
Par exemple, si le canard 1 est tué est tué au 3ème round on fait tabCan[round][1]=1;(round sera égale à 2).
A la fin du round, quand tir égale 0 on test si les canards ne sont pas tués. Si c’est le cas la case correspondantes du tableau prends la valeur -1;
Si on reprend l’exemple au dessus mais cette fois si on a raté le canard on fera tabCan[round][1]=-1;
A la suite de cette fonction il ne restera plus qu’à lire le tableau avec une autre fonction qui affichera le bon icone en fonction de la case (blanc pour 0, vert pour 1, rouge pour -1).

3.2. Algorithme 2 : La fonction moveCanard

Cette fonction gère tous les déplacements de canards ainsi que les changements de direction quand le canard rencontre un obstacles.
Le mouvement horizontal et vertical est initialisé par la fonction initCanard.
La fonction moveCanard multiplie le mouvement horizontal par -1 lorsque l'on touche un coté de la fenêtre, et multiplie le mouvement vertical par -1 quand le canard touche le haut de la fenêtre ou une limite définie proche des herbes.

De plus elle attribue au canard un mouvement vertical seulement lorsqu'il est touché jusqu'à qu'il sorte du champ de vision lorsqu'il meurt.
Elle permet aussi un arret du canard une fois qu'il est hors de vue du joueurs et aussi au moment ou il est touché, pour affiché une image pendant quelques secondes.

IV. Difficultés rencontrées

Nous avons rencontrés des difficultés sur plusieurs points :

- Gestion des fichiers de meilleurs scores et leurs affichages dans la fenêtre
- Dans le mode secondaire, gérer l'affichage avec les cuisses en fonctions du mouvements du bucket.
- Gestion de 2 évènements simultanés perturbant l'expérience de jeu.
- Installer la SDL à notre domicile (au début)
- Gérer les répertoire d'exécution du projet (au début)

V. Conclusion

Le projet s'est très bien déroulé. Nous avons tous les deux pris plaisir à réaliser ce projet et il y a eu une très bonne collaboration au sein du binôme qui a permis un travail efficace et beaucoup d'entraide entre nous. Le résultat du projet est ce que l'on espérait même si on aimerait encore lui apporter des améliorations.
Le travail a donc été réalisé dans les délais sans problème majeur.