ULB FACULTÉ DES SCIENCES

Info-f202 : Languages de Programmation

Rapport Frogger

Alessandro Dorigo

Gabriel Goldsztajn

Année académique 2023-2024

Table des matières

1 Introduction

1.1 But

Il nous avait été demandé pour ce projet de recoder en c++, avec l'aide de la librairie graphique FLTK, une version simplifiée du célèbre jeu des années 80 "Frogger". Pour ce faire il nous a été demandé d'implémenter 4 fonctionnalités de base et ensuite une dizaine de fonctionnalités bonus, pour améliorer notre note.
Nous avons donc implémenté dans notre projet les 4 fonctionnalités de base et 9 fonctionnalités additionelles. C'est à dire :

1. Une grenouille Capable de se déplacer dans 4 directions.
2. 13 rangées de déplacements. Dont la première qui est la rangée de départ, la 13e qui est la rangée d'arrivée et les autres rangée représentant la route ou des vehicules circulent.
3. L'affichage d'un message d'echec en cas de colision avec un vehicule
4. L'affichage d'un message de victoire en cas d'arrivée sur la dernière rangée.

Nous avons ensuite réalisé les tâches additionnelles suivantes :

1. Rangée d'eau.
2. Nenuphars d'arrivée
3. 3 vies pour la grenouille.
4. Des tortues plongeantes.
5. Directions de la grenouille.
6. Un score généré au fur et à mesure que le joueur avance dans une partie.
7. Un meilleure score est enregistré et affiché.
8. Un écran d'accueil.
9. Plusieurs niveaux ainsi qu'un sélécteur de niveaux

2 Classes

2.1 Structure globale du projet

Pour ce projet, nous avons décidé de suivre les recommandations données dans le sujet du projet, et avons donc utilisé le modèle de conception MVC.

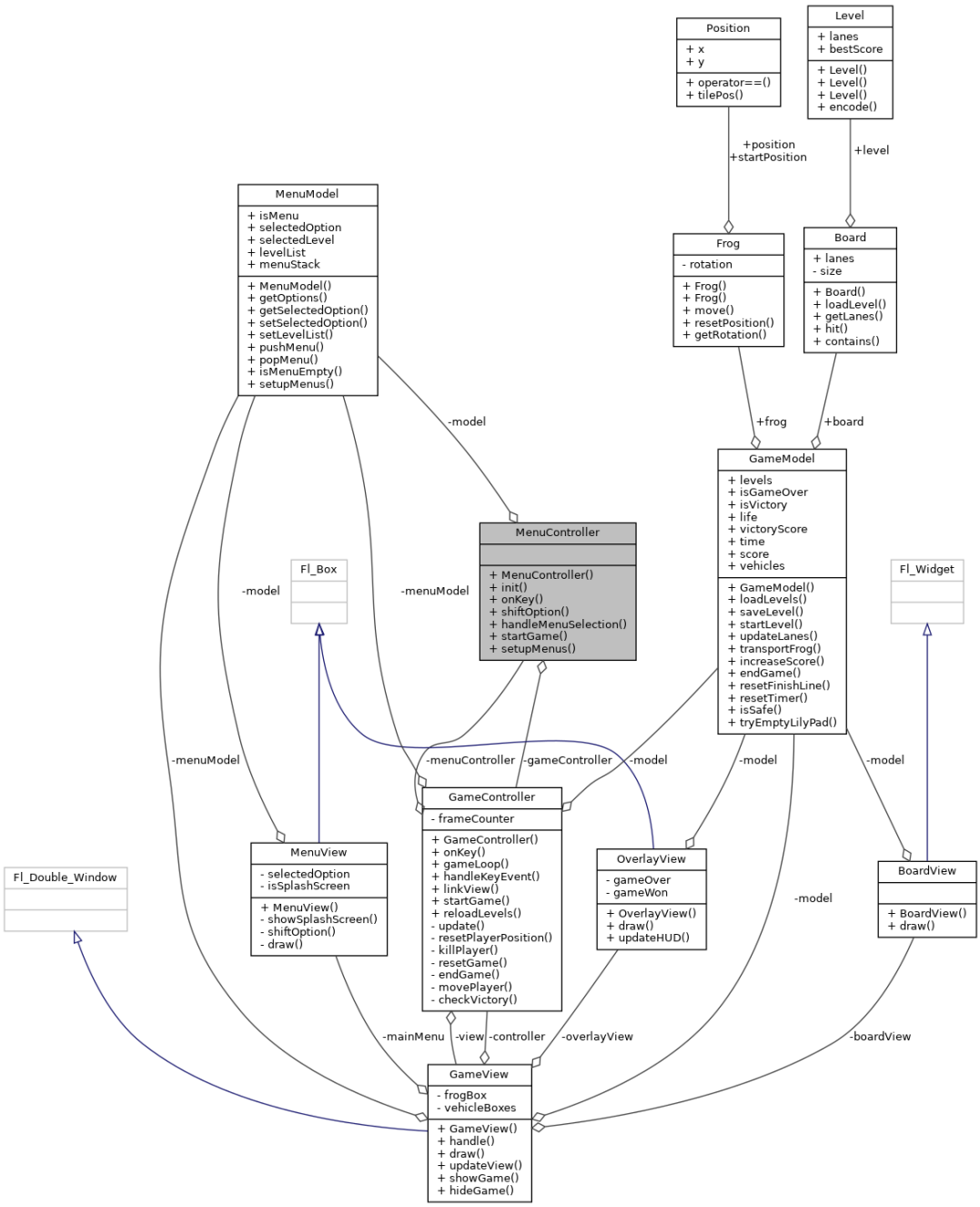

Figure 2.1 : Diagramme de classe complet

Notre projet contiens donc deux ensembles de trois classes, servant de modèle, vue et controller pour notre jeu :

1. GameController, GameView et GameModel, pour le jeu lui meme.
2. ainsi que MenuController, MenuView et MenuModel pour le menu principal et de selection de niveaux.

2.2 Interfaces

2.2.1 MenuController

La classe MenuController s'occupe de récupérer les input utilisateurs, pour permettre la navigation de l'utilisateur dans l'ecran de menu, avant ou apres une partie.

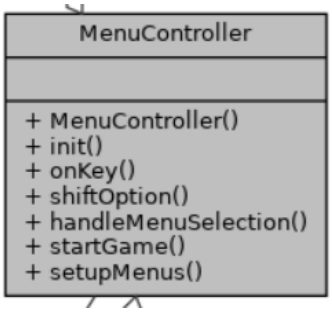

Figure 2.2 : Interface de la classe MenuController

2.2.2 MenuModel

La classe MenuModel stocke les variables permettant de choisir un niveau ou de lancer une partie. Elle est accedée et modifiée par `MenuController`

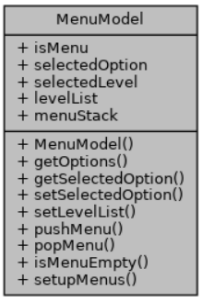

Figure 2.3 : Interface de la classe MenuModel

2.2.3 MenuView

`MenuView` est la classe s'occupant du bon affichage du menu du jeu, elle utilise les variables stockées dans `MenuModel` pour son bon fonctionnement.

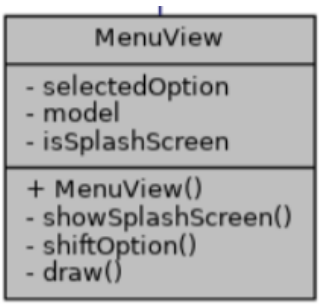

Figure 2.4 : Interface de la classe MenuView

2.2.4 GameController

La classe GameController s'occupe de récupérer les input utilisateurs, et de modifier les attributs de la classe gameModel en fonction de ceux-ci.

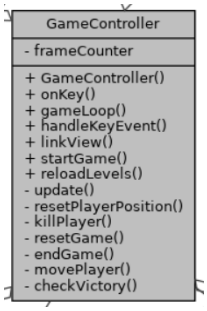

Figure 2.5 : Interface de la classe GameController

2.2.5 GameModel

GameModel est la classe agissant comme modèle stockant toutes les variables necessaires au bon dfe-roulement d'une partie.

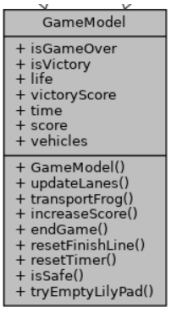

Figure 2.6 : Interface de la classe GameModel

2.2.6 GameView

GameView est la classe interagissant avec le modèle et s'occupant du bon affichage graphique de notre projet, avec l'aide des fonctions d'affichages définies dans le header DrawingUtils.hpp

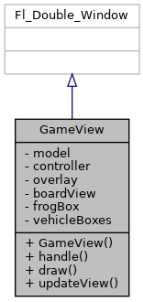

Figure 2.7 : Interface de la classe GameView

2.2.7 BoardView

BoardView s'occupe de l'affichage du tableau de jeu, ainsi que de l'ecran de fin.

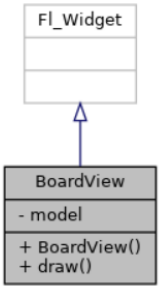

Figure 2.8 : Interface de la classe BoardView

2.2.8 OverlayView

OverlayView est la classe s'occupant de l'affichage des points de vie, ainsi que du score du joueur dans le jeu.

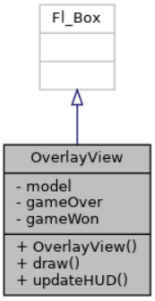

Figure 2.9 : Interface de la classe GameController

2.2.9 Frog

La classe Frog est une classe contenant la logique de deplacement des grenouilles controllées par le joueur, ainsi que l'orientation de celles-ci.

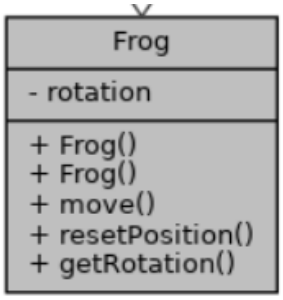

Figure 2.10 : Interface de la classe Frog

2.2.10 Board

La classe Board défini la composition de notre plateau de jeu, on y defini la taille de celui-ci, et la variable de classe `lanes` est un vecteur composé de `size` objets `Lane`.

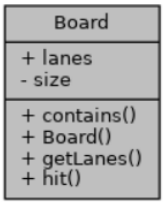

Figure 2.11 : Interface de la classe Board

2.2.11 Lane & SpawnPattern

`Lane` constitue une rangée de notre `Board`, est composée d'un vecteur de `length` objets `Tile` ainsi que d'un vecteur de `vehicles`.
La classe s'occupe, à chaque tick de jeu, de controler si un vehicule arrive à une éxtremité du board, auquel cas le vehicule est éliminé. De plus, chaque `Lane` est composée d'un objet `SpawnPattern`, qui determine le rythme auquel les vehicules seront générés.

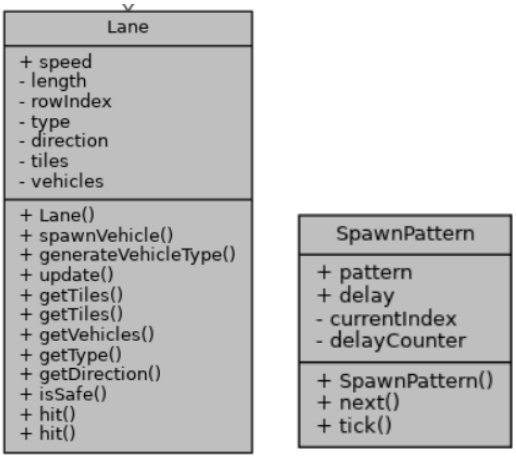

Figure 2.12 : Interfaces des classes Lane et SpawnPattern

2.2.12 Tile

Une `Tile` représente une case de notre `Board`, celle-ci est soit une case normale, soit un nenuphar vide, soit un nenuphar contenant déja une grenouille.

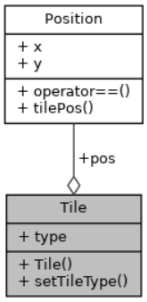

Figure 2.13 : Interface de la classe Tile

2.2.13 Vehicle VehicleConfig

La classe `Vehicle` est la classe déterminant la taille, position et deplacement d'un "vehicule", que celui-ci soit une voiture, un rondin ou une tortue.. La classe `VehicleConfig` quand à elle permet de determiner la taille et le type de vehicule a généré.

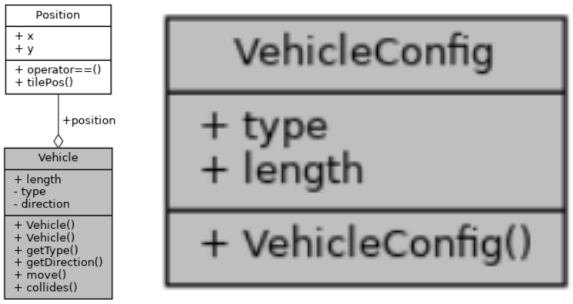

Figure 2.14 : Interface des classes Vehicle et VehicleConfig

2.2.14 Position & Coordinate

Les classes Position et Coordinate nous servent, respectivement, a indiquer la position dans BoardView, et les coordonnées absolues d'un pixel dans la fenetre de jeu. `Position` est utilisé pour les `Vehicule`, `Tile` et le `Frog` du joueur.

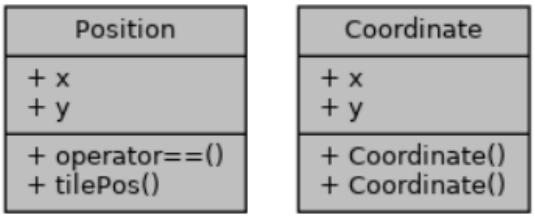

Figure 2.15 : Interfaces des classes Position et Coordinate

2.2.15 Level

`Level` est une classe stockant toutes les informations necessaires pour lancer une partie avec une certaine configuration.

Figure 2.16 : Interface de la classe Level

2.2.16 DisplaySettings

`DisplaySettings` est une classe statique contenant certaines valeurs valeures necessaires pour notre fenetre de jeu, ainsi que la taille de chaque rangée ainsi que les `Tile` a l'interieur de celles-ci.

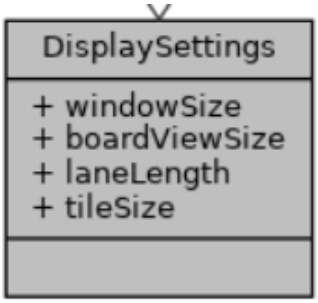

Figure 2.17 : Interface de la classe DisplaySettings

3 Logique du jeu

3.1 Logique d'une partie

Avant l'affichage de la fenetre de notre programme, nous initialisons les modeles du menu ainsi que du game. Nous initialisons alors le controller du jeu, avec en argument les modeles du menu et du game.
Ici le jeu attends alors l'input de l'utilisateur pour la selection de niveau, qui ont été chargés en memoire lors de la construction du `MenuModel`. Une fois le niveau selectionné, le contrôle de l'affichage est repassé au `GameController`, qui va modifier les parametres du game model avec les parametres du niveaux. Ici, si le joueur ne fait rien pendant une dizaine de secondes, les voitures, rondins de bois ou tortues vont circuler sur le plateau de jeu, en suivant le pattern de generation defini dans le fichier de creation de niveau.
Lorsque le joueur fera un deplacement, le `GameController` va contrôler si le coup est possible, et si c'est le cas va effectuer le deplacement et modifier les valeurs dans `GameModel`, qui va a terme se voir grace au `GameView`.