Donkey Kong, à la recherche de la banane perdue

Projet Java 3

HAJJIOUI Khalid

Professeur de l'A.A. : RIGGIO Jonathan

Table des matières

1. Introduction

Dans le cadre du projet de Java 3, il m'a été demandé de réaliser un jeu ayant comme genre : Tir/Adresse et comme thème : Nature.

C'est pour cela que j'ai réalisé Donkey Kong : à la recherche de la banane perdue, un jeu ou l'utilisateur incarne un gorille qui récupère des bananes et évite des bombes tombant du ciel.

Ce rapport sera divisé en plusieurs parties :

- Rappel des contraintes
- Présentation des fonctionnalités offertes par le jeu
- Gestion de la persistance des données
- Analyse
    - Modélisation UML des cas d'utilisations
    - Modélisation UML du diagramme de classe
- Application des Design Pattern
- Conclusion

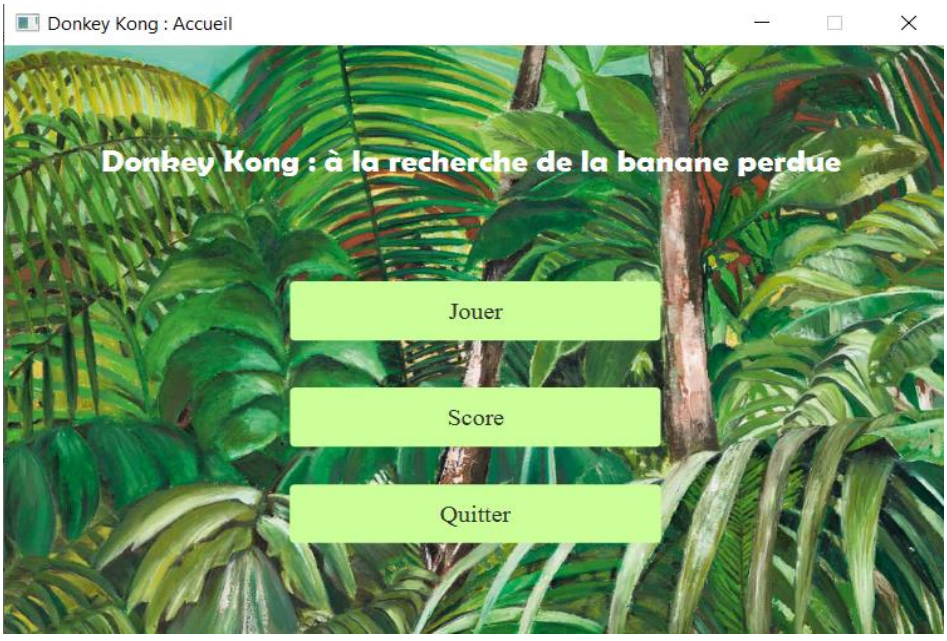
2. Rappel des contraintes

Il nous a été demandé de réaliser un jeu, avec la bibliothèque JavaFX ainsi que de :

- Créer un fichier .sh qui lancera notre projet dans un environnement Linux
- Implémenter une classe Position (templates)

3. Fonctionnalités

3.1. Avant partie



Voici la fenêtre de départ, l'utilisateur pourra voir les scores, quitter le jeu ou commencer à jouer.

En cliquant sur « Jouer », il sera amené vers la fenêtre de réglages de paramètres.

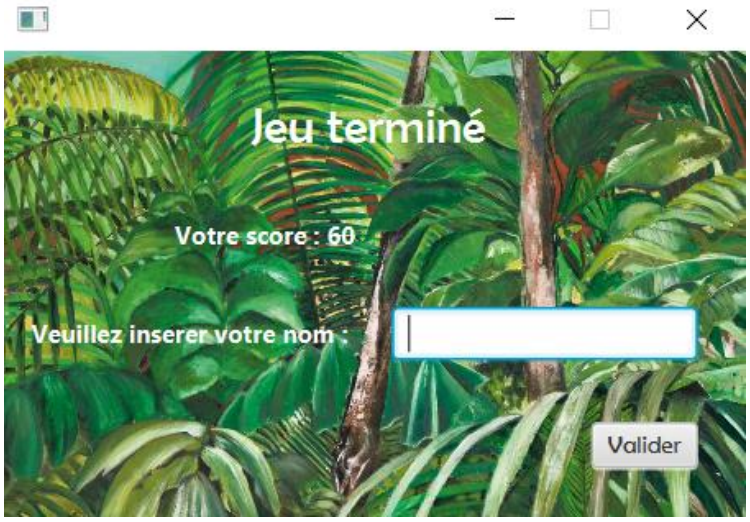
3.2. Réglage des paramètres

Le joueur pourra choisir s'il veut activer ou non le volume, ainsi que choisir la difficulté de base du jeu. Il ne pourra cocher qu'un RadioButton par paramètre (volume, difficultés). Dans le cas contraire, un message avertira l'utilisateur de ne pas cocher plusieurs RadioButton à la fois.

3.3. Interface du jeu

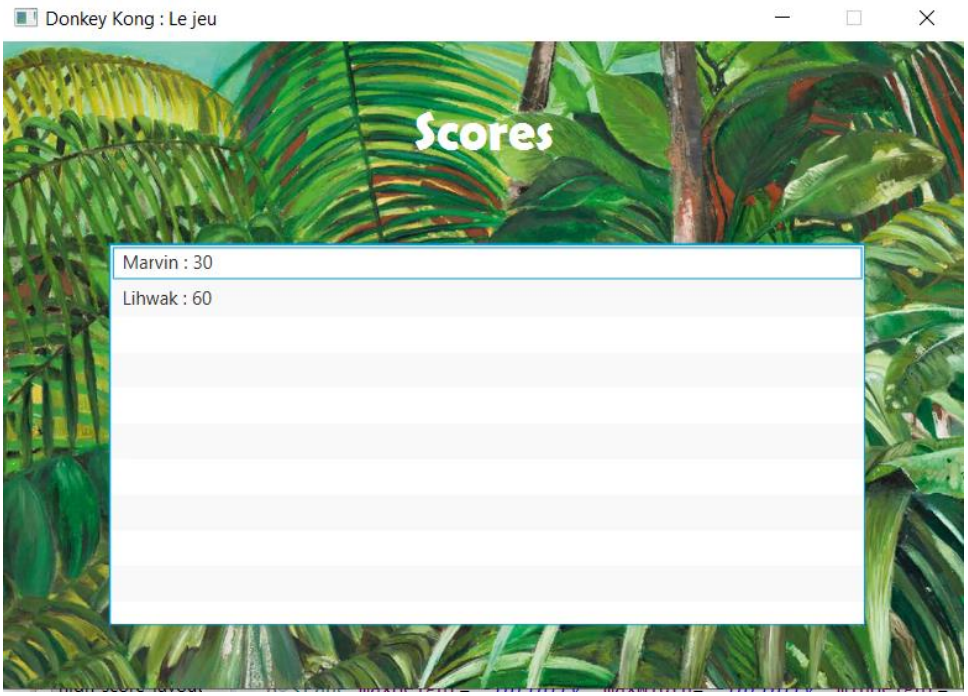
Le joueur devra se déplacer afin de récupérer des bananes et éviter des bombes tombant du ciel. Si le joueur ramasse une banane, il aura 15 points. Cependant, s'il ramasse une bombe, il perdra une de ses 5 vies.

3.4. Fin de partie



Une fois que le joueur a perdu (plus de vie), une fenêtre apparaitra où il pourra inscrire son nom afin de sauvegarder son score, qui sera affiché dans la fenêtre Score.

3.5. Score



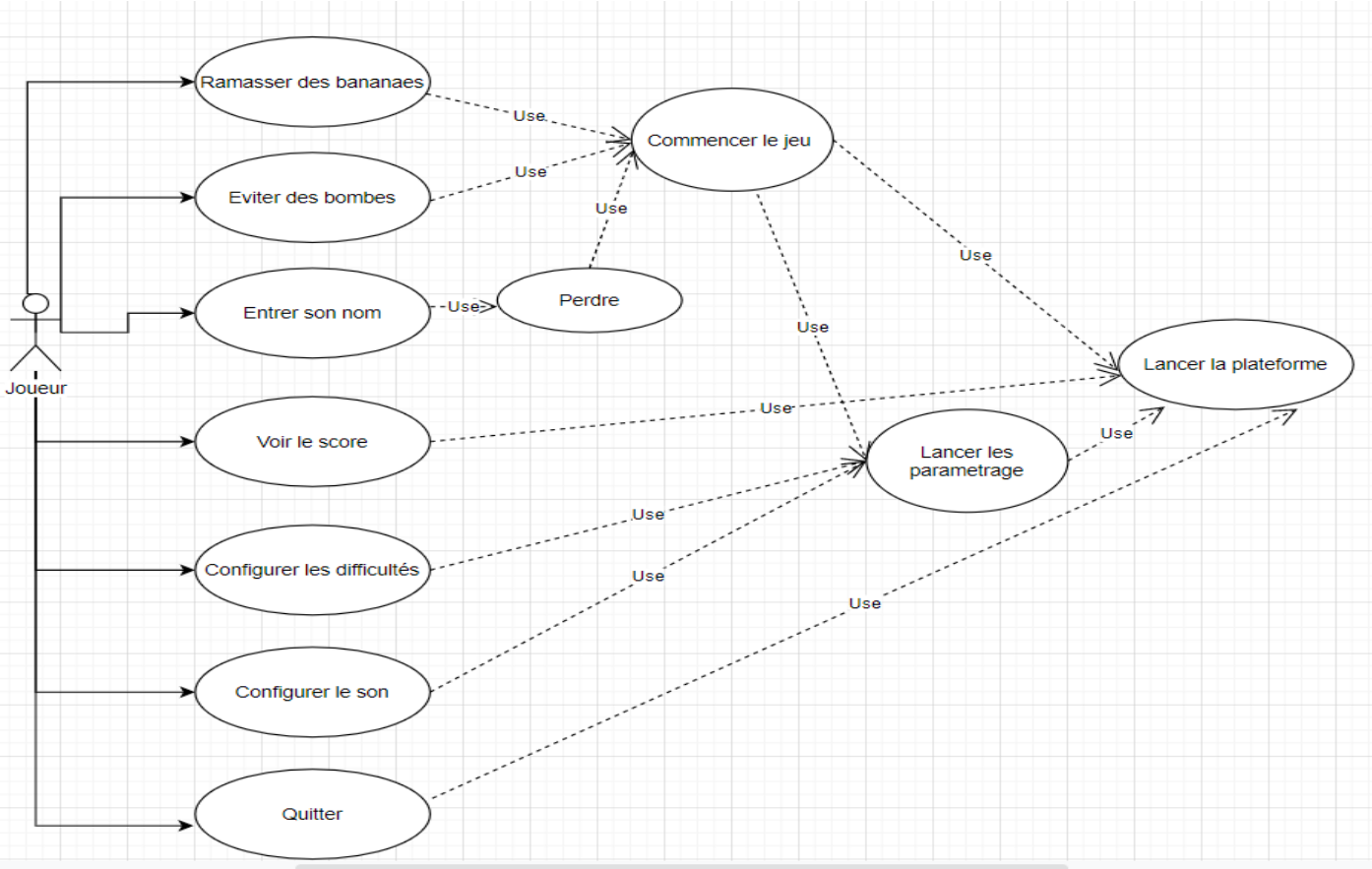
Cette fenêtre affichera le score des différents joueurs.

4. Analyse

Dans cette partie, nous allons voir la modélisation UML des diagrammes :

- Cas d'utilisation
- Diagramme de classe

4.1. Cas d'utilisations



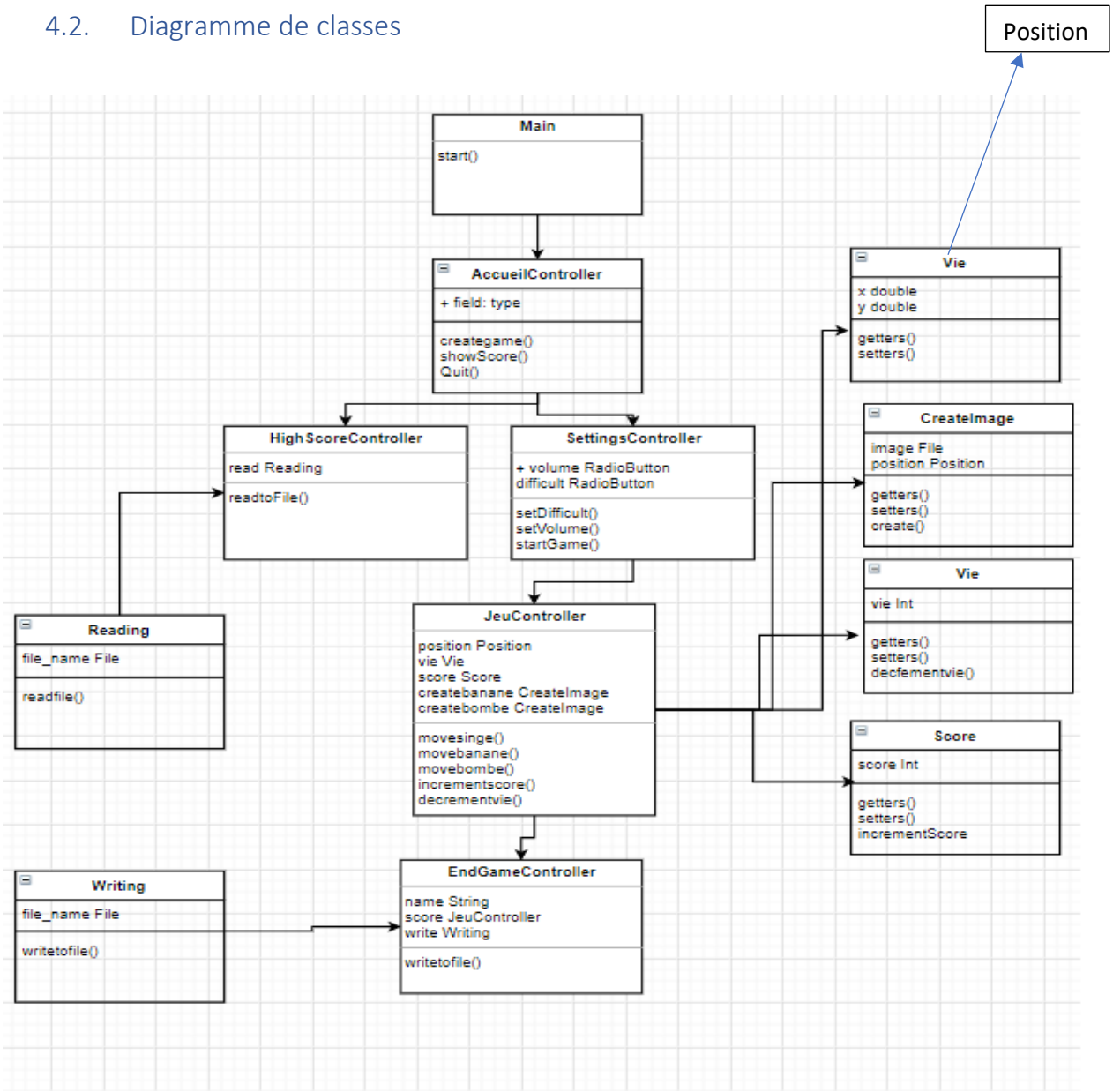
Use = « include »

Le joueur pourra ramasser des bananes, éviter des bombes, entrer son nom (s'il a perdu) seulement s'il a commencé le jeu, qui est possible que si l'on a lancé le paramétrage du jeu.

Le joueur peut également configurer la difficulté ainsi que le son s'il a lancé le paramétrage.

Pour finir, lancer le paramétrage, voir les scores ou quitter le jeu sont seulement possible une fois que la plateforme a été lancé.

4.2. Diagramme de classes



Classes modèles : Reading, Writing, CreateImage, Vie, Score, Position

La classe Main lancera AccueilController (qui est la fenêtre d'accueil). Ce contrôleur, quant à lui, pourra lancer le HighScoreController (fenêtre des scores) ou SettingsController (fenêtre de paramétrage).

SettingsController lancera JeuController (fenêtre d'interface du jeu) une fois que l'utilisateur aura paramétré le son et la difficulté.

JeuController communique avec les classes :

- CreateImage (classe se chargeant de créer des images)
  - Afin de créer des images de bananes et de bombes
- Vie (classe se chargeant de la vie)
  - Afin de mettre une vie de base et de la décrémenter si collision avec une bombe
- Score (classe se chargeant des points)
  - Afin de mettre une valeur d'incrémentation de base et d'incrémenter les points si collision avec banane
- Position
  - Positionner le singe, les bananes et bombes (les faire tomber)

Une fois que l'utilisateur n'a plus de vie, JeuControlleur lancera EndGameController (fenêtre de fin de jeu) afin que l'utilisateur puisse inscrire son nom pour que ses données soient inscrites dans un fichier .txt par le biais de la classe Writing (classe se chargeant d'inscrire des données dans un fichier).

Les données du fichier .txt peuvent être vues dans HighScoreController. En effet, ce contrôleur fera appel à la classe Reading qui lira le fichier en question.

5. Applications de Design Pattern

J'ai utilisé, dans ce projet, le design pattern : MVC

5.1. MVC

Le MVC, ou Modèle Vue Controller est un design pattern qui vise à découper le code en 3 parties :

- Modèle : les sources de données. Les classes modèles contiennent les données qui seront utilisés par le contrôleur (ex : dans un magasin en ligne, les données sont : titre de l'article, prix. On devra alors prévoir des classes modèles pour ces données).
- Controller : C'est le cœur de l'application, il gèrera la logique du code. Il sert aussi d'intermédiaire entre le modèle et la vue, car il traitera les données du modèle et il les affichera dans la vue. (Ex : récupérer le prix de la classe modèle, lui assigner une valeur de 50 euro, et l'affichera dans la vue)
- Vue : Les vues contiennent toute la partie graphique (les boutons, labels, images). Elles ne contiennent aucune logique, car toutes les données que l'on voit viennent des classes modèles et ont été traités par le contrôleur.

J'ai voulu utiliser ce design pattern car j'avais déjà élaboré une logique de développement reposant sur ce design. En effet, il est implémenté dans la plupart des interfaces graphiques (jeu, application,). De plus, l'utiliser (dans les GUI) permet une meilleure optimisation du code (lier les modèles, vues et les contrôleurs est très dangereux car si l'on fait une erreur dans le code, on devra tout vérifier).

Je l'ai utilisé de cette manière :

- Modèle : les sources de données sont les images, la vie, le score, la position ainsi que la manipulation de fichier. Quelques méthodes respectives accompagnent ces classes (ex : pour vie, on a DecrementVie(), pour score on a IncrementScore())
- Vues : Elles sont présentes dans le fichier (vues) du projet sous formes de fichiers FXML. Chaque vues est liés à un contrôleur (ex : accueil_layout.fxml est liés à AccueilController.java)
- Controller : Chaque contrôleur possède sa vue. En effet, lorsqu'il y a une manipulation à faire (évènement, modification d'un élément de la vue), le contrôleur le modifie sur base de la source de données des classes modèles (ex : récupérer la vie, la décrémenter et afficher le résultat dans la vue.

6. Persistance des données

Dans ce projet, j'ai utilisé un fichier texte géré par les classes modèles : « Reading » et « Writing ». Les points de l'utilisateur, une fois le jeu terminé, seront sauvegardés dans ce fichier.

7. Conclusion

Travailler de manière autonome sur ce projet m'a permis d'en apprendre d'avantages sur la POO, les interfaces graphiques ainsi que les design pattern. Et j'en suis très fier ! Toutes les fonctionnalités demandées sont présentes, malgré qu'une, que j'ai voulu implémenter (Menu pause) manque à l'appel.