PROJET JAVA : LE JEU DE LA BALLE AUX PRISONNIERS

Par L'huissier Evan & Michallon Rémi

L’huissier Evan
Michallon Rémi

Table des matières

L'huissier Evan
Michallon Rémi

Présentation globale

Dans le cadre d'un projet de groupe, il s'agit de créer un jeu de balle aux prisonniers en Java.
Le jeu s'initialise avec 3 joueurs de chaque côté dont un joueur et deux bots par coté.
Le but est d'éliminer l'équipe adverse en touchant les ennemis avec la balle.
Nous avons choisi de réaliser ce projet sur Intellij, qui est plus accessibles et qui a une plus jolie interface, d'après nous.
Nous avons organiser notre code avec 3 designs patterns : MVC, Singleton et Factory

Expliquer choix d'architecture (Patterns) + diagramme de classe

Le Pattern MVC

Explications

Le Pattern MVC signifie **Model View Controller**.
L'organisation du code se fait en trois couches structurées :

- **View** = ce qui est perceptible par la vue (pour un client par exemple).
  Le code lié à l'interface y est contenu, on peut le modifier sans risque de casser du code métier ou de la structure de données.
- **Controller** = interprète une action reçue par View. Son objectif est de modifier les données du modèle.
- **Model** = son but est de créer des données, de lire des données, de les mettre à jour, de supprimer des données (CRUD).
  Il contient les définitions des structures de données, leurs fonctions.

Dans notre cas, l'utilisation du modèle MVC est obligatoire dans le cadre de notre projet scolaire, cependant nous reconnaissons son efficacité en termes d'organisation et sa facilité de mise en place.

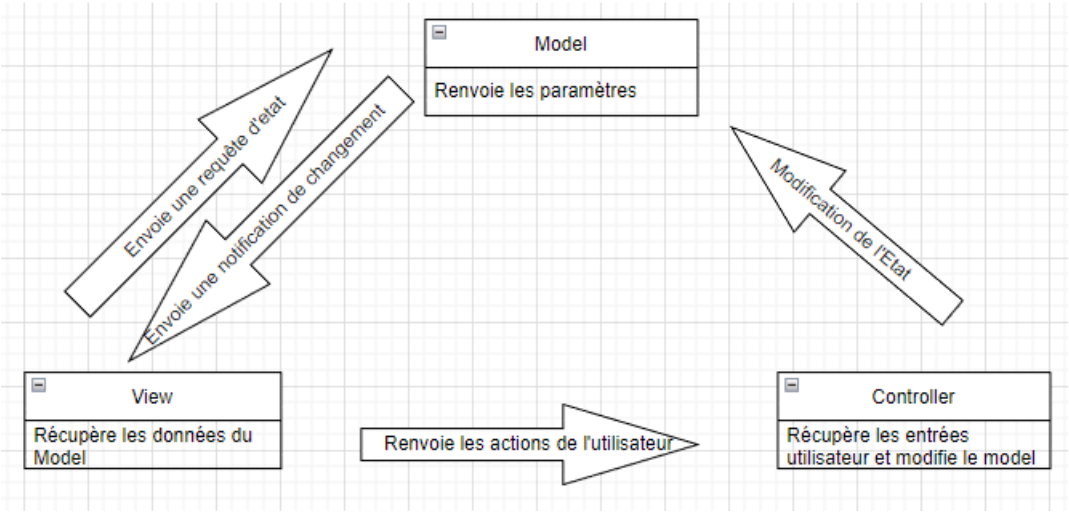
Notre classe **View** contient la sous-classe App qui est la classe principale du projet, elle renvoie aux éléments visuels du jeu, comme l'apparition du terrain en s'appuyant sur javafx.

Notre classe **Controller** regroupe la configuration des touches de déplacement et de tir des joueurs, les éléments qui influencent le gameplay.

Notre classe **Model** contient les sous classes qui vont préciser les éléments de notre classe **Controller**, en communiquant les axes de tir, les angles de déplacements, les vitesses de déplacements, les sprites des joueurs, il précise également les éléments de la classe **View** notamment en précisant la forme du terrain, ses couleurs, sa taille…

L’huissier Evan
Michallon Rémi

Le Diagramme de classe du modèle MVC



Le Pattern Façade

Explications

Il s’agit d’un pattern de conception structurel qui procure une interface offrant un accès simplifié à une librairie, un framework ou à n’importe quel ensemble complexe de classes.

C’est une classe qui procure une interface simple vers un sous-système complexe de parties mobiles.

Nous avons utilisé ce design Pattern lors de l’implémentation de notre interface, cette méthode nous semblait rapide à mettre en œuvre et était efficace c’est les raisons qui nous ont poussées a utiliser ce design Pattern.

Le Pattern Singleton

Explications

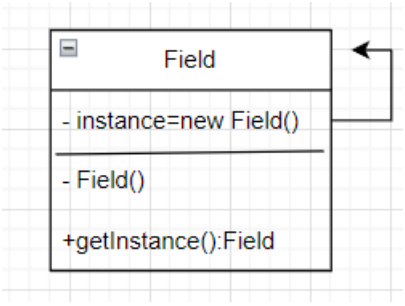
Il s’agit d’un pattern qui induit le fait qu’une classe n’existe qu’une fois et est accessible depuis tout autre classe statiquement.
Elle se renvoie à elle-même.

Dans notre cas, la classe Field déclare la méthode statique getInstance qui retourne la même instance de sa propre classe.

Nous avons choisi ce pattern car il permet de renvoyer l’objet sans devoir le passer en paramètres, il nous permet aussi d’appeler n’importe ou dans le code le Field ce qui est un gain de temps de plus ce modèle protège son instance des modifications.

L’huissier Evan
Michallon Rémi

Le Diagramme de classe du modèle Singleton



Conclusion

Cette partie de notre rapport consiste à faire le point sur ce que nous avons réussi à faire ainsi que les différents problèmes que nous avons rencontrés.

Nous ne sommes pas arrivés à implémenter l’intégralité de nos objectifs de départ

Nous sommes arrivés à configurer entièrement : les déplacements, la balle, les collisions …

Nous avons rencontré un problème lors de la mise en place des IAs. Une erreur de dépassement de la taille de la pile lors de l’appel de Field.getInstance() nous empêchait de récupérer les joueurs les plus proches et de déplacer les IAs. Malgré le fait que la fonction n’est pas appelée dans une boucle infinie (nous n’avons pas vu d’appel de fonctions excessifs). Nous ne savons pas expliquer ce problème et aucune solution sur internet n’a pu le régler.

Nous voulions insérer un design pattern Facade mais nous n’avions pas correctement compris son fonctionnement.
Nous avions compris qu’il fallait mettre en place une interface permettant de définir les fonctions communes aux classes l’implémentant. Nous avons obtenu une grosse duplication de code, qui nous a fait nous rendre compte de notre erreur.
Nous n’avons malheureusement pas eu le temps d’y remédier.