

PROJET JAVA : LE JEU DE LA BALLE AUX PRISONNIERS

Par L'huissier Evan & Michallon Rémi



Table des matières

Présentation globale.....	2
Expliquer choix d'architecture (Patterns) + diagramme de classe.....	2
Le Pattern MVC	2
Explications.....	2
Le Diagramme de classe du modèle MVC.....	3
Le Pattern	3
Le Pattern	4

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 côté.

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 accessible et qui a une plus jolie interface, d'après nous.

Nous avons organisé 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 contrôler les données qu'il reçoit.
- **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

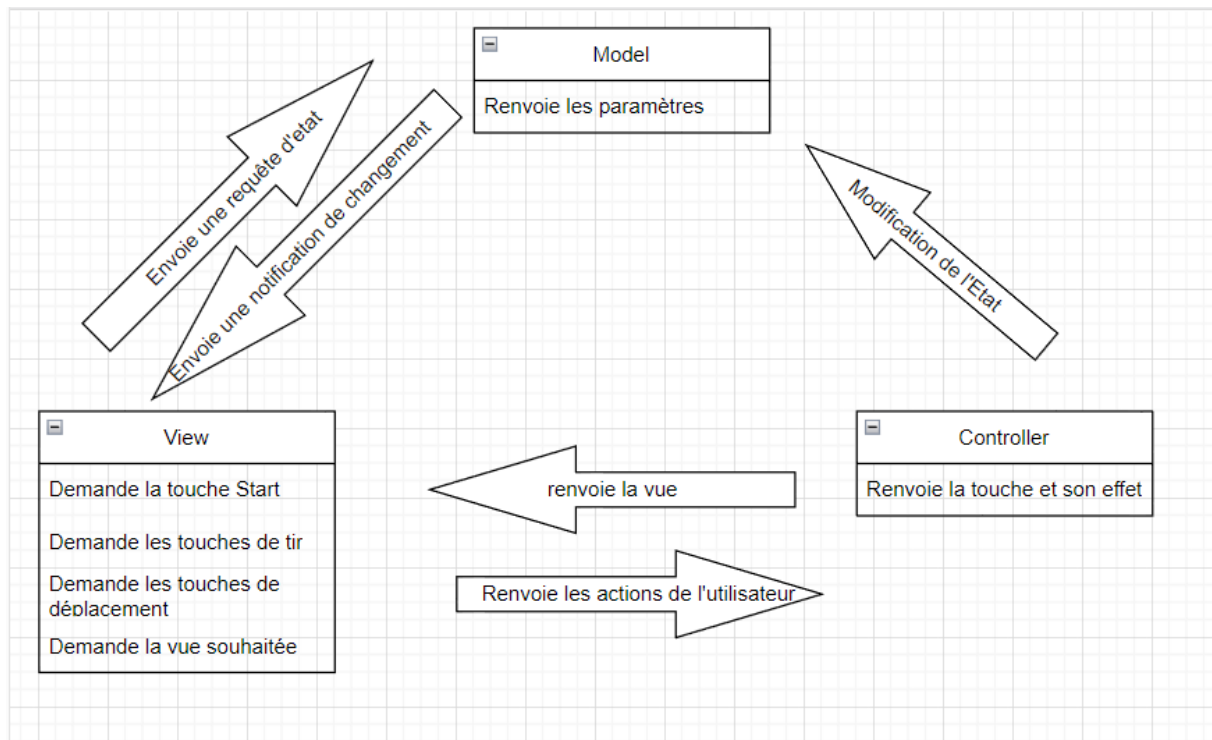
Dans notre cas, l'utilisation du modèle MVC est obligatoire dans le cadre de notre projet scolaire, cependant nous reconnaissons son efficacité en termes d'organisation et sa facilité de mise en place.

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...

Le Diagramme de classe du modèle MVC



Le Pattern Factory

Explications

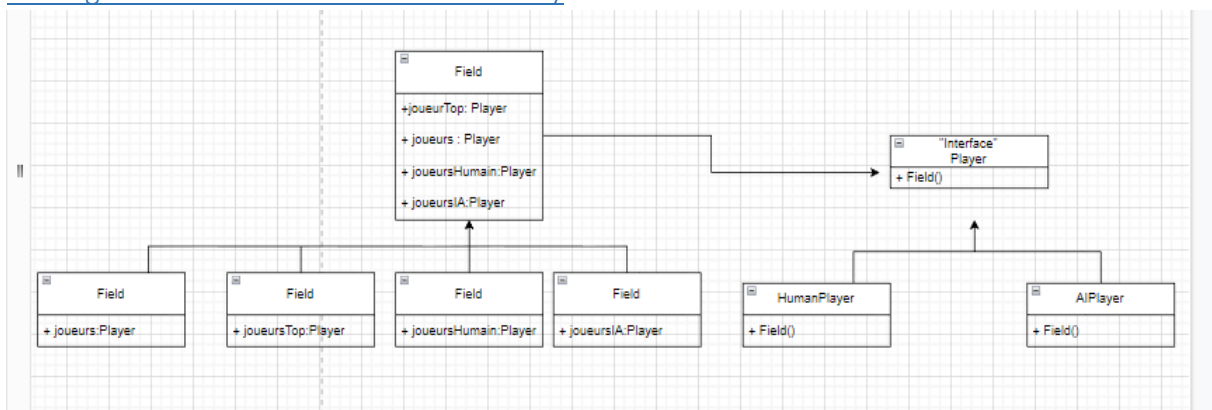
Le pattern Factory a pour but d'instancier une classe qui crée des objets.

Nous avons choisi d'ajouter ce pattern car il permet de gérer facilement plusieurs méthodes de construction.

Nous avons notamment utilisé ce pattern lors de la conception de notre classe joueur, afin de différencier nos différents types de joueurs et facilement attribuer leurs propres usages et modes de fonctionnement.

Cette méthode nous a permis d'éviter de multiplier les classes héritages et donc de simplifier la lecture du code.

Le Diagramme de classe du modèle Factory



Le Pattern Singleton

Explications

Il s'agit d'un pattern qui induit le fait qu'une classe n'existe qu'une fois et est ouverte pour toute l'instance.

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 le constructeur sans retourner un nouvel objet, il nous permet aussi d'appeler n'importe où dans le code le Field ce qui est un gain de temps de plus ce modèle protège son instance des modifications.

Le Diagramme de classe du modèle Singleton

