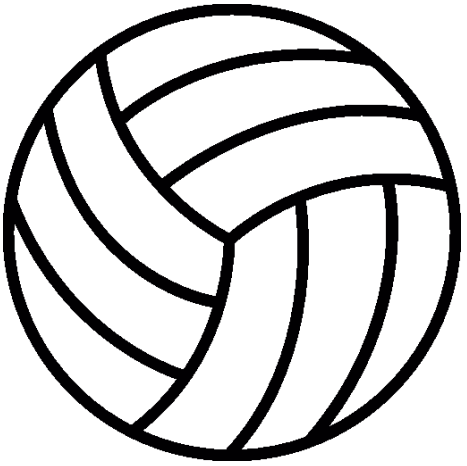


Jeu de Balle au Prisonnier

Application JAVA

Rapport de Projet

Université Lumière Lyon 2

Conception agile et génie logiciel, Automne 2022

KOWAL--PICCHI Tom | FAVRE Lou

I. Organisation

A. Présentation du projet

Nous avons dû, lors du cours de conception agile de projet informatique, implémenter une version de jeu du style balle au prisonnier fonctionnelle en JAVA à partir d'une base donnée. Lors du développement de l'application, nous avons dû respecter quelques contraintes données par l'enseignant ou par nous-même, tel que le MVC (modèle vue contrôleur). Ou l'utilisation de javaFX (un framework ainsi qu'une bibliothèque d'UI permettant de créer et de gérer des interfaces graphiques en JAVA) pour l'aspect graphique de notre jeu.

Pour travailler dans les meilleures conditions possibles, nous avons utilisé un outil de versionning, ***GITHUB***, ce qui nous a permis de travailler simultanément sur des parties différentes du projet et de mieux se répartir les tâches. Et comme fortement conseillé par notre enseignant, nous avons utilisé une méthode ***AGILE***, le pair programming (ou programmation en binôme).

Un code était fourni, cependant, il avait pour but d'uniquement créer les joueurs, les équipes, et les afficher. Il nous restait donc à implémenter de nombreuses fonctionnalités pour avoir un jeu fonctionnel.

B. Initialisation du projet

Avant de commencer à développer l'application, nous avons choisi de retranscrire sur un diagramme de classe le code fourni pour une meilleure compréhension et pour préparer un plan de développement. Grâce à cela, nous avons pu, une fois cette étape terminée, commencer à développer efficacement.

Nous avons choisi, dans un premier temps, de commencer par transformer la base que l'on avait, de sorte à ce qu'elle applique le modèle MVC, ce qui nous sera plus utile à l'avenir pour implémenter les diverses fonctionnalités. Pour finir par développer la logique du jeu, comme la balle, le système de tir et des collisions, etc...

II. Synthèse du développement

A. Principales fonctionnalités développées

Système d'équipe et de l'IA :

Les différents joueurs sont répartis dans deux équipes composées d'un joueur humain chacun, et de X joueurs IA. Ces derniers ont une IA très simple, ils vont se déplacer aléatoirement sur le terrain (sans chercher à esquiver ou à récupérer la balle). Et vont tirer a chaque fois devant eux, lorsqu'ils le peuvent (donc lorsqu'ils ont récupérés la balle)

Collisions, ramassage de balle:

Lors d'une partie, un joueur peut se faire toucher par la balle de deux manières différentes, quand la balle est en mouvement ou arrêtée. Si la balle en mouvement va percuter un joueur, ce dernier se retrouve exclu du terrain et n'apparaît plus sur l'écran (et par conséquent ne peut plus être contrôlé si c'était un joueur humain). Si un joueur touche la balle lorsqu'elle est à l'arrêt, il va récupérer la balle, et aura la possibilité de tirer (en choisissant l'angle si c'est un joueur humain, sinon l'ia va faire tirer la balle tout droit). Cependant, si un joueur tire la balle sur un mur, elle va s'arrêter et être bloquée, il faudra alors appuyer sur la touche "R" pour débloquer la balle (elle va être redistribuée aléatoirement à un joueur). Au début d'une partie, un joueur, désigné aléatoirement, possède la balle.

Menu et démarrage de la partie :

Nous avons décidé de mettre en place un menu, nous permettant au choix de lancer une partie, d'afficher de l'aide pour savoir comment jouer, et de pouvoir quitter l'application.

Menu de démarrage du jeu

Gestion de la victoire et des scores :

Nous avons implémenté un système de scoring, un pour chaque équipe, qui va s'incrémenter quand un joueur élimine un joueur adverse. Dans ce cas là, elle est considérée comme victorieuse, et va avoir ses points augmenter.

Gestion des collisions :

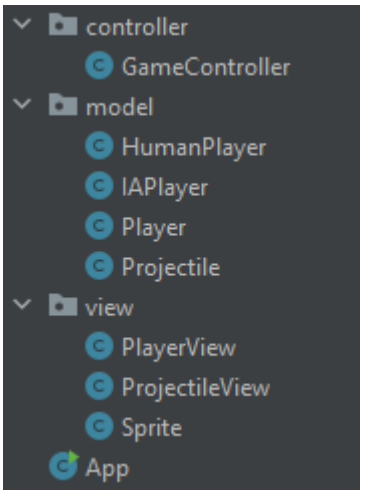
Nous avons déterminé que la balle pouvait entrer en collision avec deux types de surface, le bord du terrain, dans ce cas elle s'arrête directement. Ou un joueur, ici, c'est l'état de la balle qui va être important (la réaction avec le joueur va différer selon si la balle est en mouvement ou non).

B. Implémentation de designs patterns

Au cours du projet, nous avions comme consigne d'intégrer trois design patterns différents. Nous avons choisis d'implémenter le MVC, le singleton, et l'iterator.

Le Modèle Vue Contrôleur :

Dans le but d'implémenter ce pattern, nous avons choisis de séparer certaines classes dans différents packages, un pour chaque fonctionnalité du MVC. La classe App va lancer l'application et afficher les différentes options (Jouer, savoir comment jouer et quitter). Lorsque l'on va vouloir jouer, le controller gameController, va afficher les différents personnages, et va se charger de les animer. Lorsque le modèle va changer (dans notre cas la position ou l'état des personnages / de la balle), c'est gameController, qui va indiquer à la vue ces changements.



Singleton :

Nous sommes partis du postulat qu'il ne pouvait y avoir qu'une seule balle dans le jeu. Donc, nous avons décidé d'implémenter dans la classe

Projectile (représentant une balle), un singleton. C'est un pattern qui va restreindre l'instanciation d'une classe à un seul objet (c'est-à-dire qu'on ne va pouvoir créer qu' une seule balle, et que cette même balle va être rappelée au lieu d'être créée).

```
public static Projectile getProjectile(double speed , double direction, double x, double y, int sens) {
        if(projectile == null) {
            projectile = new Projectile(speed ,direction, x, y,sens);
        }
        else {
            projectile.speed = speed;
            projectile.direction = direction;
            projectile.x = x;
            projectile.y = y;
            projectile.sens = sens;
            projectile.ballMoving = false;
        }
        return projectile;
    }
```

Au lieu d'appeler le constructeur de la classe Projectile, nous appelons la méthode getProjectile, qui va appeler ce dernier uniquement si aucune instance de balle n'a été créé, sinon elle va renvoyer un projectile avec ses paramètres modifiés.

Iterator :

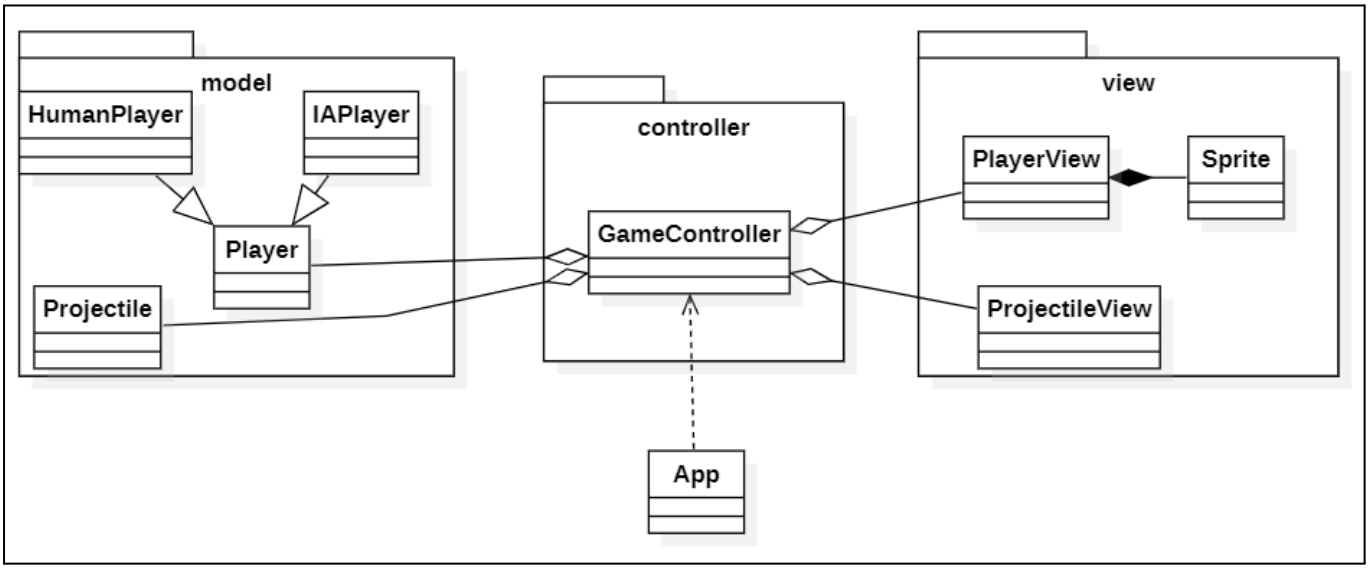
Nous utilisons indirectement le design pattern Iterator. Cependant, nous ne l'avons pas implémenté. Nous l'avons utilisé lorsque nous voulions savoir si un joueur d'une équipe précise avait été touché par la balle (méthode checkCollisionOfTeam de la classe gameController). Ce pattern a été utilisé car il permet de parcourir très simplement une collection d'objets, pour faire des opérations dessus (dans notre cas pour une liste de joueur cela convenait parfaitement).

C. Difficultées rencontrées

Les principales difficultés que l'on a rencontrées étaient liées aux contraintes posées, c'est-à-dire, le respect du MVC, et la compréhension de javaFX (qui est une bibliothèque que nous n'avions que très peu manipulée par le passé). Cependant, nous avons pu, au fur et à mesure du projet, surmonter ces contraintes.

III. Bilan

A. Diagramme de classe final



Ci-dessus nous pouvons voir notre diagramme de classe, avec les différents packages (qui nous ont aidé à la réalisation du MVC), et notre découpage de l'application en différentes classes. Pour des questions de lisibilité, nous avons fait le choix de ne pas représenter les méthodes et attributs des différentes classes.

B. Analyse, conclusion et perspective d’évolution

Pour conclure, ce projet nous aura permis de parfaire nos connaissances en JAVA, mais aussi plus globalement en programmation. En effet, nous avons pu apprendre et découvrir de nombreuses notions, comme les design patterns, mais aussi le pair-programming, et ce malgré les difficultés que cela a pu causer.

Cependant, le projet peut continuer de s'améliorer. Il nous manque notamment une méthode pour relancer une partie une fois qu’il ne reste plus qu’un joueur. Ou encore un système de rebond sur les murs latéraux.