Université Lumiére Lyon 2
Conception agile de projets

RAPPORT DU TP/PROJET BALLE AU PRISONNIER

GRIMAULT NATHAN
MAURIN CELIA

INTRODUCTION

Dans le cadre de notre UE Conception agile de projets informatiques et génie logiciel, nous avions à réaliser en binôme un projet sur le thème de la **balle au prisonnier** afin d'aquérir les spécificités de la gestion de projet agile et leurs différences par rapport à une gestion de projet plus classique.

Ce rapport comprend dans un premier temps, une **présentation** globale du projet, puis dans un second temps la motivation **des choix d'architecture**(patterns) et leurs explications. Pour explicités ces choix nous nous appuierons sur un **diagramme UML** qui sera lui aussi détaillé en quelques lignes.

BALLE AU PRISONNIER

FONCTIONALITÉS

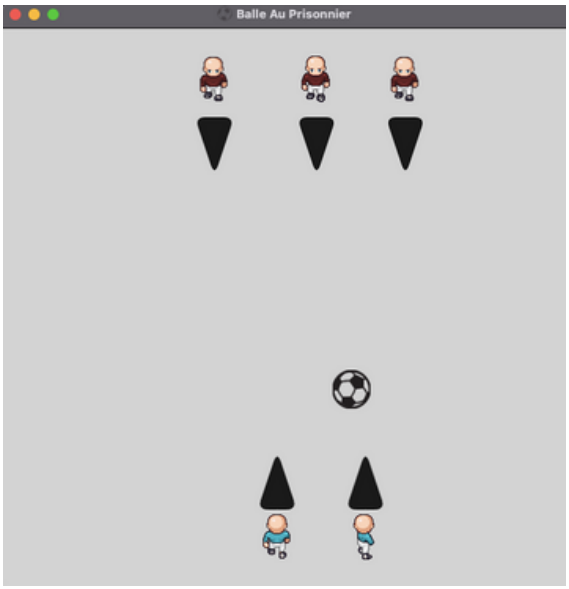
Voici les fonctionalités que nous avons implémentés dans la balle au prisonnier de départ.

MENU - INTERFACE

Nous avons ajouté au jeu une **interface** avant de jouer avec plusieurs fonctionnalités.
Nous avons ajouter les contrôles pour chaque camp, ainsi qu'un bouton START pour commencer à jouer.

Une fois le Bouton START enclenché, une partie commence sur la fenêtre courante.

JEU - EXTENSIONS



Pour la partie Jeu, nous sommes parvenus à implémenter plusieurs fonctionnalités :

- Les **collisions** ont étés implémentés entre les joueurs et la balle ainsi que la balle et les murs. Les joueurs touchés par la balle disparaissent du terrain

1. Nous avons mis par défault le déplacement des Bots, ils se dirigent vers la balle et dirigent leurs tirs vers les adversaires.

- Nous avons crées une classe abstraite **Strategy** afin que les bots aient plusieurs stratégies

Strategy1: Les Bots se déplacent aléatoirement avec une vitesse aléatoire.

- Nous avons ajoutés un **menu de fin** une fois qu'une des 2 équipes à gagner la partie. Le menu propose alors de recommencer une nouvelle partie avec le bouton **RESTART**.

- On peut aussi arrêter la partie et fermer la fenêtre avec le bouton **QUITTER**.

CHOIX D'ARCHITECTURE
LES DESIGN PATTERNS EXPLICATIONS

Pour ce projet, nous avions la taches d'ajouter 3 designs patterns avec au minimum l'architecture **MVC**. Pour les 2 autres, nous avons d'ajouter les patterns **Singleton** et **Strategy**.

MVC

L'implémentation du design patterns **Architecture MVC** était obligatoire pour notre projet, ainsi, très tôt nous avons modifié la structure du code de départ en 3 dossiers : **View, Controler, Model.** Ceux-ci regroupent les fichiers Java correspondant chacun à un rôle précis dans l'interface. À l'aide du Diagramme de Classe ci-dessous, on peut voir que cette Architecture a bien été respecté. **Model** contient les données manipulées par le programme. Il assure la gestion de ces données et garantit leur intégrité. La classe **View** fait l'interface avec l'utilisateur. **Controler** est chargé de la synchronisation du modèle et de la vue.

L'architecture **MVC** est principalement essentielle pour la programmation objet et pour des projets comme c'est le cas ici pour notre projet de balle au prisonnier.
L'architecture MVC est un patterns essentiel car il est **facile à maintenir**, avec sa séparation en fichiers de manière logique. Le développement peut se faire à plusieurs niveaux en parallèle. Cette architecture permet de **tester** le code de manière plus **simple et efficace,** aussi il permet de travailler plus efficacement notamment à **plusieurs**. Dans notre code, la classe **Field** est le Model, la classe **App** pour la Vue et **Controler** pour le Contrôler.

Singleton

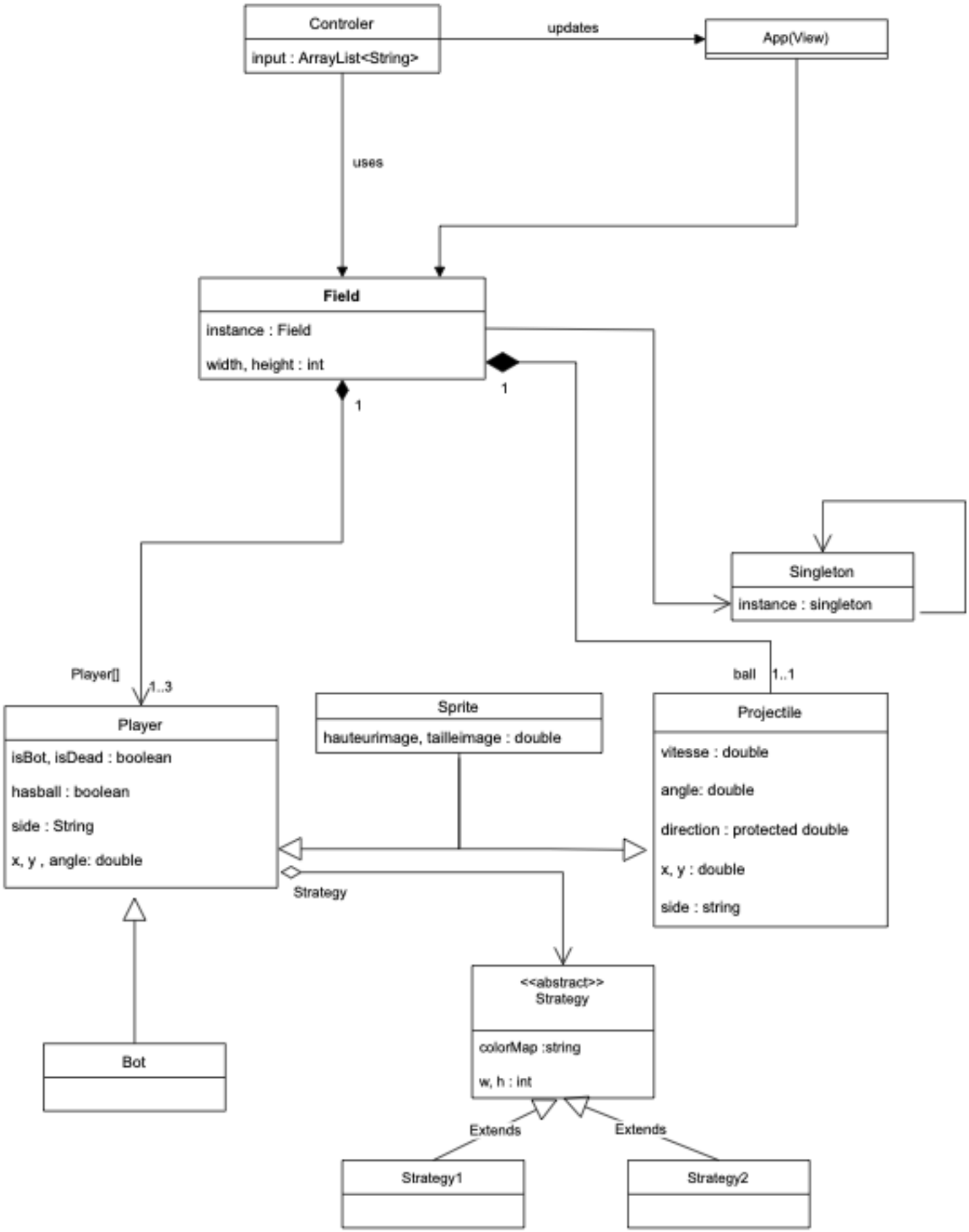
Comme deuxième patterns nous avons choisis d'implémenter le Singleton. Nous avons choisis ce deuxième pattern car il garantit qu'une **classe(Field)** ne reste réellement qu'avec cette seule instance singleton. Le singleton rend **Field** globalement accessible dans le logiciel.
Singleton permet de répondre à la problématique de n'avoir qu'une seule et même instance de la classe **Field**. L'implémentation d'une telle structure est très rapide et simple à mettre en place.

Strategy

Pour le troisième Design Patterns, le sujet du projet demandait **différentes statégies** pour les Bots du jeu. Ainsi, il nous a semblé logique d'implémenter le Patterns **Strategy**, celui-ci permet réaliser différentes opérations avec un seul et même objet.

Ici, il s'agit de la stratégie des Bots qui peut être différente, on a donc crée une **classe abstraite** qui permet de généraliser sur la stratégie des Bots qui est ensuite composé de 2 instance **Strategy1** et **Strategy2** qui sont encapsulés dans Strategy.

DIAGRAMME UML
DIAGRAMME DE CLASSE



Explications Diagramme UML

Le Diagramme de classe a été réalisé directement à la main à partir du code. On peut commencer par le **modèle MVC** qui est représenté avec Controler, Field et App qui ont été présentés précédement dans la partie MVC.
La classe **Field** représente le terrain qui représente le terrain avec sa taille et son unicité avec le lien du **Singleton.** Field possède deux tableaux de **joueurs**, un pour les joueurs du haut un autre pour ceux du bas, il peut y avoir de 1 à 3 joueurs pour chaque côté. Aussi, Field a une et une seule balle.

Ensuite, les classes **Projectile** et **Player** possèdent un **sprite** qui permet de definir les champs de déplacements pour les joueurs et la balle.
La classe **Bot** hérite de la classe **Player**, car il s'agit d'un joueur un peu particulier qui représente des actions aléatoires ou dirigés par le **Model.**
Comme expliqué précédemment, nous avons instancié une classe abstraite **Strategy** qui est relié avec **Player** car il représente la stratégie des joueurs qui sera employé dans la partie.

Conclusion

À travers ce projet, nous avons appris sur les notions de **conception**, **gestion** de projet et de temps.
Ce projet nous a permis de nous rendre compte que les systèmes informatique sont **complexes** et demandent des méthodes de conception pour être menés à bien.