Projet balle aux prisonniers

Introduction

Avant de présenter notre projet de balle au prisonnier, nous souhaitons préciser quelques petits éléments de contexte. Tout d'abord, nous n'avons pas beaucoup d'expérience avec la librairie JavaFX et l'outil Maven. En effet, nous avions toutes les deux déjà développé en Java, mais nous n'avions utilisé que le mode console ou d'autres librairies. Nous avons donc rencontré de nombreuses difficultés dû à notre manque d'expérience avec ces outils en plus des autres difficultés de développement et de refactorisation du code. De plus, nous avons décidé de planifier nos différentes tâches au fil de la réalisation et des différents TD. Vous pourrez donc retrouver dans ce document l'organisation qui a été mise en place, les éléments présents et absents de notre jeu ainsi que sa structure finale.

Structure du code

1) Présentation du jeu

a. Organisation de la réalisation

Comme dit précédemment, nous souhaitions organiser notre création de projet afin de réaliser un maximum d'éléments du jeu et par la suite utiliser correctement le barème donné lors du TD3.

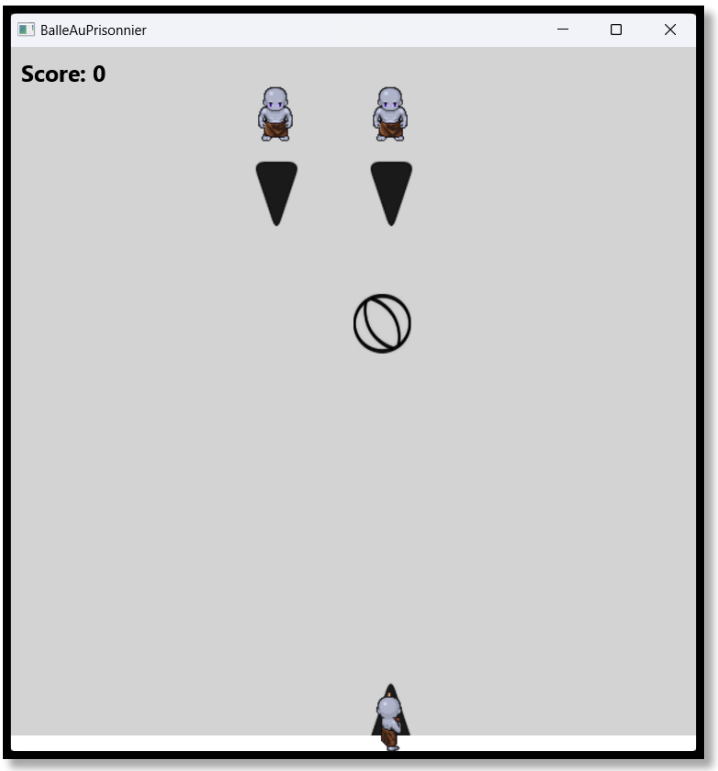
Nous savions que nous allions manquer de temps au vu de la quantité d'éléments à réaliser. Nous avons donc commencé par mettre en place les premiers éléments demandés au début du TD1, pour ensuite créer un dépôt Git et réaliser une refactorisation du code afin d'utiliser la structure MVC qui est obligatoire dans le cadre de ce projet.

Par la suite, nous avions décidé de réaliser les tâches suivantes en prenant en compte le barème de notation, ce que nous connaissions de la librairie ainsi que la complexité de chacune des tâches afin de faire le jeu le plus complet possible.

Voici donc la répartition finale des grandes parties du projet :

- MVC : Manon + aide Lina
- Tir : Lina
- Collisions : Manon
- Interface : Lina
- Rapport : Manon

b. Ce qui a été implémenté



Contenu du jeu :

- Une équipe d'IA (2 joueurs) VS une équipe d'humain (1 joueur)
- Un déplacement horizontal uniquement :
- Différents contrôles pour chaque joueurs (les IA sont contrôlables) :
    - Joueur 1 : flèches gauche – droite – haut – bas | tir : « Entrée » | prendre balle : « P »
    - IA 1 : Z – Q – S – D | tir : « ESPACE » | prendre balle : « P »
    - IA 2 : O – K – L – M | tir : « SHIFT » | prendre balle : « P »
- Un système de mort si les joueurs se font toucher
- Une balle :
    - Qui rebondit sur les murs
    - Que les joueurs peuvent attraper / prendre au sol
    - Que les joueurs peuvent garder en main en se déplaçant
    - Que les joueurs peuvent lancer sur l'équipe adverse ou passer à leurs coéquipiers
- Une interface qui affiche :
    - Le score actuel du joueur humain
    - Un écran de Game Over si ce dernier meurt
    - Un écran de victoire si le joueur touche tous les ennemis

Au niveau du code :

- Un dossier Git disponible sur Github
- Un outil de build : Maven
- 13 classes au total
- Une structure en MVC (presque complète)
- Utilisation d'un singleton pour la balle

c. Ce qui n'a pas été implémenté

Comme dit précédemment, nous n'avons malheureusement pas eu la possibilité d'implémenter certaines fonctionnalités :

- IA des ennemis :

Lorsque nous avions terminé la mise en place des tirs et des collisions, nous n'avions plus beaucoup de temps pour réaliser les 2 différents types d'IA ainsi que l'interface du jeu et le rapport. Nous nous sommes donc questionnées sur la complexité de mise en place de ces différents éléments et nous en avons conclu qu'il était plus pertinent de créer une interface et un rapport complet et soigné plutôt que de mettre en place les IA. En effet, nous aurions souhaité mettre en place le pattern design « Strategy » qui nous aurait permis de définir plusieurs classes avec des algorithmes différents pour créer le comportement des IA et de les appeler au bon moment : comportement de déplacements, comportement de tir, comportement de récupération de la balle au sol et en l'air. Tout cela nous aurait demandé trop de temps, surtout en comptant la session débogage dû aux bugs qui sont survenus après chaque gros ajouts ou changements.

- Déplacement des joueurs dans chaque moitié de la fenêtre et pas que sur l'axe x.
- Mise en place d'un panneau présentant les différents contrôles du jeu.
- Mise en place d'une option de pause.
- Autres ajouts d'amélioration.

2) Refactorisation du code : MVC

Avant de nous lancer dans l'ajout d'extensions, nous avons d'abord souhaité restructurer le code afin de pouvoir implémenter correctement les fonctionnalités suivantes. Il est en effet plus facile de restructurer un projet peu avancé et avec peu de fonctionnalités qu'un projet plus conséquent.

- *Game :* création des instances des joueurs et des ennemis et création des sprites
- *App :* met en route le jeu en appelant les différentes classes
- *Package Vue :*
    - Field : génère le terrain (taille et couleur)
    - GameVue : repère les inputs des joueurs et appel la bonne méthode du bon controller. Possède aussi la méthode Touched() qui gère les collisions balle/joueurs
    - PlayerVue : gère le sprite du joueur : appel animation et gère sa flèche
    - Sprite : gère les animations de tous les sprites du jeu
- *Package Model :*
    - Player : classe mère qui gère les joueurs de tous types : déplacements, mort, rotation de la flèche + actions de prendre la balle
    - Human : hérite de Player et possède un constructeur spécifique au joueur humain
    - IA : hérite de Player et possède un constructeur spécifique aux IA
- *Package Controller :*
    - Evènement *:* gère la boucle de jeu en parcourant tous les joueurs

Petite remarque : Nous aurions aimé changer certains éléments de notre MVC, comme mettre le contenu d'Évènement dans Game par exemple, afin de le rendre plus complet (voir parties rouges sur diagramme). Cependant nous avions eu de grosses difficultés à mettre en place la structure MVC. Nous

avons donc, à la suite de vos conseils, décidé de ne pas compléter notre MCV déjà bien avancé afin de nous concentrer sur les autres éléments de notre jeu. Nous aurions aussi aimé mieux organiser notre structure globale.

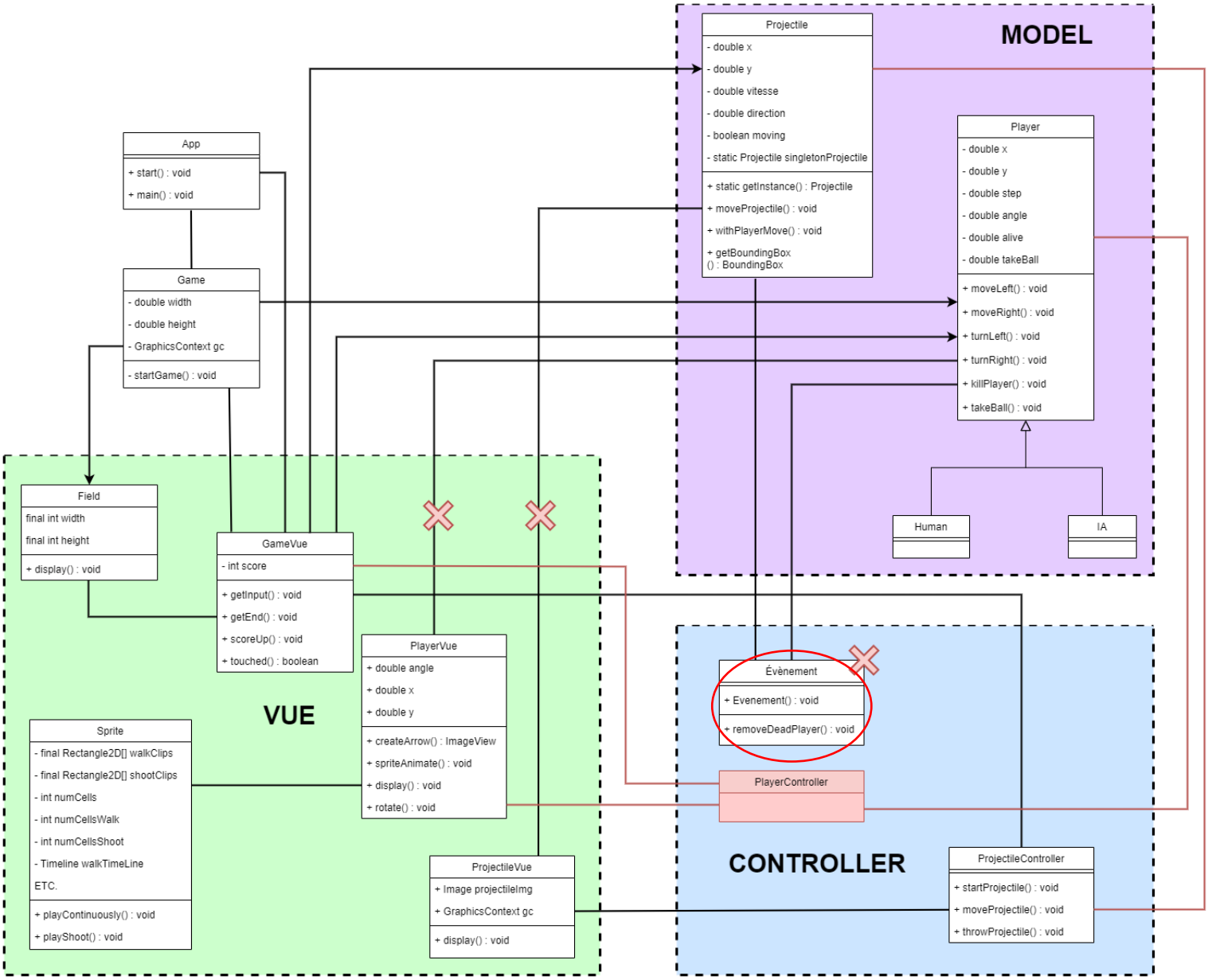


Figure 1 : Diagramme de classes - MVC

3) Gestion balle : tir et collisions

a. Tir

- Création de classes :
  - Projectile (Model) :
    - Création de Projectile en Singleton
    - getInstance() : permet de créer et de récupérer l'unique instance de la balle
    - moveProjectile() : créer le mouvement de la balle

- ProjectileController :
  - startProjectile() : créer une vue pour le projectile et l'affiche lorsque le joueur appuie pour la première fois sur « ENTRÉE »
  - moveProjectile() : vérifie que le projectile possède une vue et appelle la méthode moveProjectile() du model
  - throwProjectile() : lance la balle (setMoving(true)) et donne un angle (setDirection())
- ProjectileVue :
  - Constructeur de la vue du projectile
  - display() : affiche la balle en lui associant une image

- Dans Évènement : Dans la boucle de jeu, on récupère l'instance de la balle
- Dans GameVue :
  - Pour chaque type de joueur et chaque input possible parmi la liste : on appelle la méthode shoot() dans Player, on lance la balle et on change setTakeBall() à false.
  - Vérification de la position de la balle pour qu'elle s'arrête au niveau de la ligne des deux équipes et au niveau des limites de la fenêtre.
- Dans Game : instanciation de Projectile et de ProjectileVue, avec une vitesse de 1.5 et une position relative à celle du joueur humain (c'est lui qui commence le jeu donc qui lance la balle pour la première fois).

Petit résumé en diagramme :

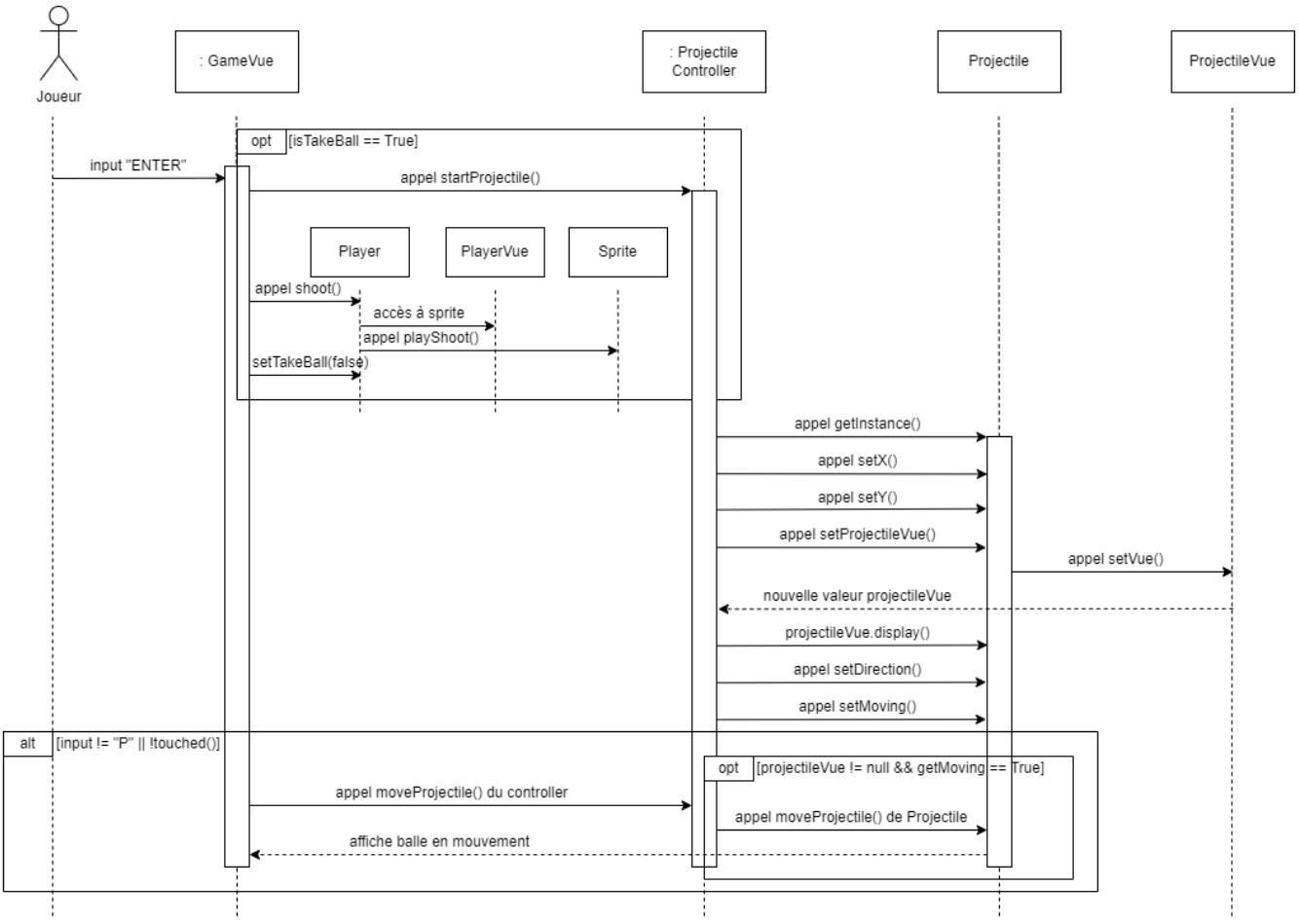


Figure 2 : Diagramme de séquence - Lancement de la balle par le joueur humain

Pourquoi l'utilisation d'un singleton pour le projectile ? Cette structure nous permet d'avoir une seule et unique balle sur le terrain. Projectile étant un singleton, le constructeur est privé et ne prend aucun argument. On initialise les variables de position et la vue avec des setters.

b. Collisions

- Dans Player :
  - killPlayer() : change la variable alive de true à false : le joueur meurt
  - takeBall() : change la variable TakeBall à true : le joueur possède la balle
- Dans Projectile :
  - withPlayerMove() : le joueur qui a attrapé la balle peut se déplacer avec
  - getBoundingBox() : créer une boite de collision autour de la balle
- Dans GameVue :
  - Si le joueur attrape la balle lorsqu'elle entre en collision avec lui, la balle s'affiche devant le joueur. Ce dernier possède la balle
  - Création de la méthode touched() qui permet de récupérer les boites de collision de la balle et du sprite des joueurs. La fonction retourne true ou false en fonction de si les boites de collision sont entrées en contact
- Dans Évènement :
  - Dans la boucle de jeu, on vérifie lorsque la balle est en mouvement, si le joueur a récupéré la balle en appelant le getter isTakeBall(). Selon s'il l'a récupéré il sera soit tué, soit il pourra relancer la balle
  - Vérification de la possession de la balle par le joueur pour pouvoir la lancer

4) Interface

Ajout d'un élément Text dans la classe App que l'on passe en argument à GameVue où l'on déterminera son contenu en fonction du score. Celui-ci est mis à jour dans Évènements lorsqu'un joueur est touché, en faisant appel à la fonction scoreUp() de GameVue. Toujours dans GameVue, la fonction getEnd() est appelée et permet d'afficher un message lorsque la partie est terminée : « Game Over ! » si le joueur est mort ou « You Win ! » si le joueur a touché tous les ennemis.

Conclusion

Malgré les difficultés rencontrées durant ce projet, nous avons réussi à créer un jeu fonctionnel, et nous avons atteints les objectifs que nous nous étions fixés en fonction du temps qui nous était imparti ainsi que de nos connaissances en Java et JavaFX. Le manque de temps nous a poussé à appliquer des principes de gestion de projet tels que ceux vu en cours, en priorisant certaines tâches et en faisant des compromis de manière à garder une balance équilibrée entre le temps passé sur une tâche et l'importance de celle-ci dans le barème.

Au-delà de l'aspect technique, la gestion de projet passe aussi par l'aspect social, et bien que n'ayant jamais travaillé ensemble avant ce projet, nous n'avons pas eu de problèmes pour nous organiser et nous avons su garder un équilibre dans l'attribution des tâches. Le projet s'est donc plutôt bien passé et a été une expérience enrichissante pour nous deux.