Albane Nicoullaud
Morgan Raveleau
Vincent Renault

Master 1 Informatique

- Projet -
Conception agile de projets informatiques et génie logiciel :

I. Présentation :

L'objectif de ce projet est de réaliser un jeu de balle aux prisonniers. Il nous est demandé de mettre en place quelques classes afin de nous permettre de mieux comprendre les grands principes de la programmation orientée objet.

Le jeu comprend trois joueurs par équipes, avec deux joueurs "ordinateurs" et un joueur "humain". Il nous a fallu s'occuper de différents paramètres tels que leur vitesse et les touches de clavier qui leur permettent de se déplacer et de tirer (chaque équipe a des touches différentes).

Pour la gestion des tirs nous avons ajouté un projectile qui ne se déplace que lorsque la touche "tir" de l'équipe est actionnée. Le but étant qu'il touche un joueur adverse afin que celui-ci soit éliminé.

II. Les différents patterns :

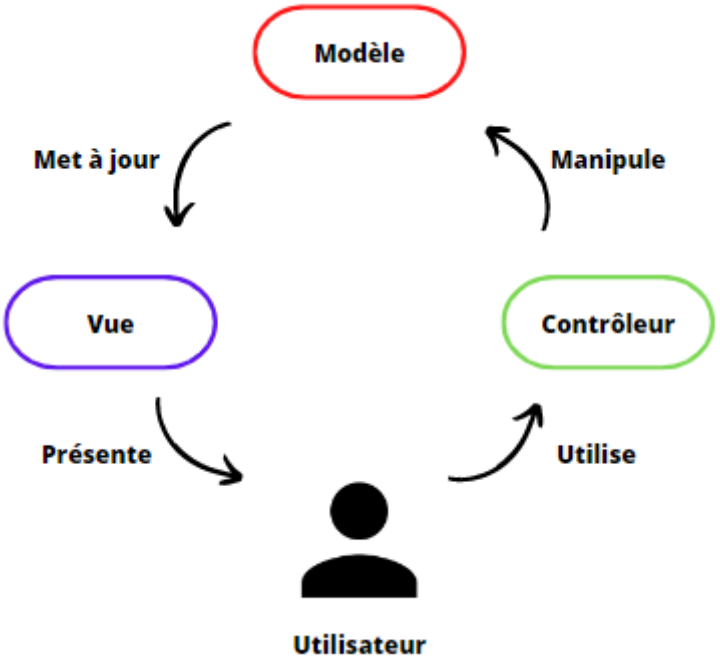
Afin de mener à bien notre projet nous avons la possibilité d'utiliser différents patrons de conception. Ces derniers permettent de résoudre un problème\ une tâche spécifique. Il en existe trois types différents : les patrons architecturaux, les patrons de création et les patrons de structure.quatre types différents: les patrons architecturaux, les patrons de création, les patrons de structure et les patrons de comportement. Nous expliquerons les types de patrons que nous allons utiliser ultérieurement.

Le premier design pattern qui nous intéresse est de type architectural, c'est-à-dire qu'il permet d'avoir une vue globale du système, c'est un pattern dit de "haut-niveau".

Modèle - Vue - Contrôleur (MVC):

Ce modèle de conception met en avant le fait qu'une application se compose d'un modèle de données, d'informations de présentation et d'information de contrôle. Ce modèle exige que chacun d'eux soit séparé en différents objets:

- **Modèle:** ne contient que les données d'application pures, il ne contient aucune logique décrivant comment présenter les données à l'utilisateur.
- **Vue:** présente les données du modèle à l'utilisateur. Elle permet d'accéder aux données du modèle, cependant La vue sait comment accéder aux données du modèle sans savoir ce qu'elles signifient ni comment l'utilisateur peut les manipuler.
- **Contrôleur:** fait le lien entre l'objet modèle et l'objet vue. Il permet d'appeler une méthode de Modèle pour l'afficher dans Vue.



Utilisation du MCV dans notre projet:

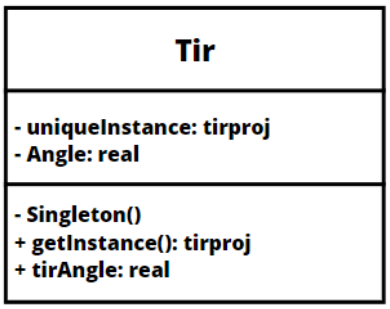
Dans notre projet le patron MCV se présenterai de la manière suivante:

- **Modèle**: Contient l'ensemble des règles qui initailisent les joueurs ainsi que la balle. On en trouve une partie dans les classes Field.java et Projectile.java.
- **Vue**: Contient l'ensemble des classes qui concerne la présentation, on en retrouve une partie dans la classe App.java.
- **Contrôleur**: Contient les paramètres de contrôle des personnages que l'on retrouve dans la classe Field.java.

Le patron de conception qui va suivre est de type création, c'est-à-dire qu'il est utilisé pour créer des objets.

Singleton:

Ce modèle permet de s'assurer que l'on a une instance unique pour une classe. Il fournit également un point d'accès unique et global pour les autres objets. Pour le faire fonctionner il faut que le contrôleur de la classe soit privé, c'est-à-dire que seul les méthodes de la classe peuvent y accéder. De plus, il faut que l'instance unique de la classe soit stockée dans une variable statique privée . Enfin, pour créer une méthode statique de la classe, il faut pour commencer créer l'instance dès le premier appel pour la retourner par la suite.



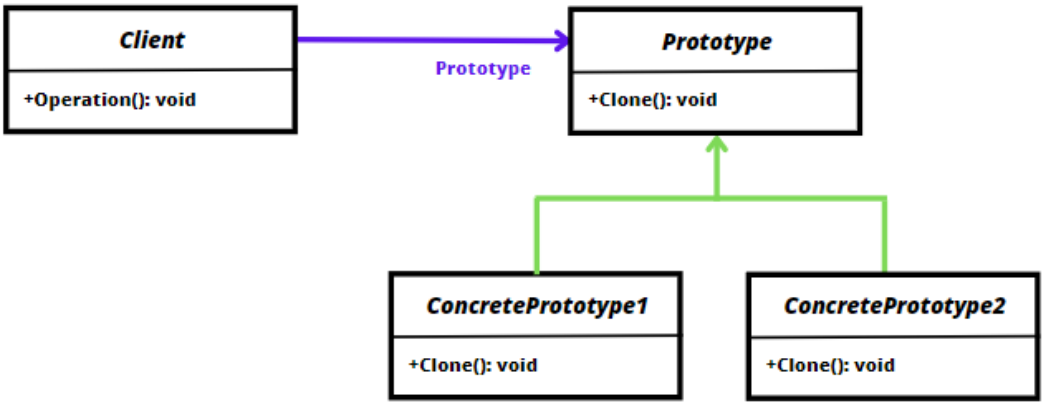
Utilisation de Singleton dans notre projet:

Nous pouvons utiliser le patron Singleton en créant une instance qui initialise l'ensemble des joueurs,tels que leurs graphiques et de la balle par exemple. Ce sont des éléments uniques qui ne seront pas modifiés par la suite.

Le dernier patron de conception est lui aussi un patron de création.

Prototype:

Ce design pattern permet de réutiliser un comportement sans recréer une instance et d'économiser des ressources. Pour le faire fonctionner, il faut recopier une instance existante avec la méthode clone(). Par la suite, il faut ajouter des comportements spécifiques, c'est ce qu'on appelle le "polymorphisme pas cher". C'est la méthode qui est la plus choisie en JavaScript, elle ne contient pas de classes.



Utilisation de Prototype dans notre projet:

Nous pouvons l'utiliser pour initialiser les paramètres de l'ia en s'appuyant des paramètres des joueurs humains. Ce qui nous permet de gagner du temps et d'avoir des joueurs machine et des joueurs humains avec des paramètres identiques.

III. Les choix d'architecture:

App.java:

La classe App est hérité de Application (classe à partir de laquelle les applications de JavaFX s'étendent), nous initialisons le nom de la fenêtre, le terrain de jeu que l'on ajoute à la racine de la scène et la scène à la fenêtre pour l'affichage.

Field.java:

La classe Fields hérite de la classe Canvas (est une image sur laquelle on peut dessiner). Dans cette classe, nous initialisons, les joueurs (humain et ia) et les couleurs qu'ils peuvent avoir, ainsi que le projectile. De plus, nous paramétrons la taille de la scène.
Une fonction Field est organisée afin de capturer les évènements de la souris et du clavier, elle initialise aussi les joueurs et leurs emplacements, de même pour le projectile. Par défaut, la scène est instanciée à 600 pixels de largeur et hauteur. Nous avons des fonctions publiques afin de récupérer un joueur, le projectile et les dimensions de la scène.

Player:

La classe Player est le constructeur d'un joueur, définit par sa position, son déplacement, sa visée, lorsqu'il porte la balle et l'angle de visée. Cette classe servira de base pour les joueurs humains et artificiels.

PlayerIA.java:

La classe PlayerIA hérite de la classe Player, elle définit la partie intelligence artificielle, elle dirige les joueurs non contrôlés par les utilisateurs. Dans le projet, nous avons réussi à les afficher en gérant leurs positions et images. Nous avons pris en compte leurs déplacements, rotations, shoots et animations. Cependant, nous n'arrivons pas à les faire jouer en autonomie complète.

Player.java:

La classe PlayerHumain hérite de la classe Player, qui définit la position du joueur par rapport à la scène, l'angle de tir qu'il a par défaut et son pas. De plus, nous instancions le joueur par le constructeur Player, composé de divers paramètres afin de le positionner, de choisir son équipe, l'asset que nous allons afficher et la vitesse de déplacement du joueur. Une fonction display permettra l'affichage du joueur ainsi que la position de son viseur. Diverses fonctions vont permettre le changement de direction du viseur, gérer les déplacements de celui-ci, le shoot avec le projectile, le déplacement en mode boost, l'animation de la balle et la création de celle-ci.

Projectile.java:

La classe Projectile définit la balle envoyée par les joueurs. Que ce soit par sa position, son angle ou sa vitesse de déplacement. Pour cela, nous disposons d'une asset ball.png, que nous allons paramétrer pour gérer sa rotation, son mouvement, ses collisions et le fait qu'un joueur soit touché ou non. Si la balle arrive derrière un joueur sans le toucher, elle sera donnée automatiquement à celui-ci afin qu'il puisse tirer.

Sprite.java:

La classe Sprite hérite de ImageView afin de redimensionner le format de l'image. Le constructeur Sprite va gérer les animations des images, que ce soit ceux de la balle ou des joueurs, lors d'un shoot, d'un mouvement, d'une collision ou de la mort d'un joueur.

IV. Conclusion:

La réalisation de ce projet a été pour nous très compliqué, n'ayant les connaissances suffisantes pour mener à bien ce projet nous avons fait de notre mieux pour améliorer le jeu du mieux que nous pouvions.
De plus, l'installation de JavaFX a été très compliquée, pour certains d'entre nous l'installer sur nos ordinateurs personnels n'a pas été possible.

Cependant, les différentes améliorations auxquelles nous avons procédé nous ont permis de nous donner une idée de ce que pouvait produire un code en java. Même si notre jeu n'est pas complet, nous avons acquis de nouvelles connaissances qui nous seront utiles.