Le 9 décembre 2021

Génie Logiciel

Daniel MURRAY, Océane PERROUAULT, Joshua AUBRY, Jules ROCHE

L3 Informatique
Groupe 1A-1B
Année 2021-2022

Dans le cadre du projet de Méthode de Conception, il nous a été demandé de réaliser par groupe de 4, un jeu de stratégie respectant le modèle MVC à l'aide du langage de programmation JAVA. L'objectif principal, par le biais du jeu de stratégie au tour par tour imposé, était de mettre en pratique nos connaissances du semestre sur les différents patterns appris en cours.

Table des matières

1 Présentation et Organisation

Ce projet étant à être réalisé en groupe de 4 élèves, le notre est composé de Daniel MURRAY, Océane PERROUAULT, Joshua AUBRY et Jules ROCHE. Dans le cadre de ce projet, nous devions réaliser un jeu de stratégie au tour par tour.

1.1 Description du projet

Le but de notre projet est de réaliser une application de jeu de combat qui pourrait opposer 2 à n joueurs sur une grille de jeu à 2 dimensions, à taille paramétrable. Chaque joueur représente un combattant qui dispose initialement d'un certain nombre d'armes de différents types (dont chacunes possèdent un nombre limité de munitions) et d'un certain nombre d'énergie. Les règles du jeu sont nombreuses.

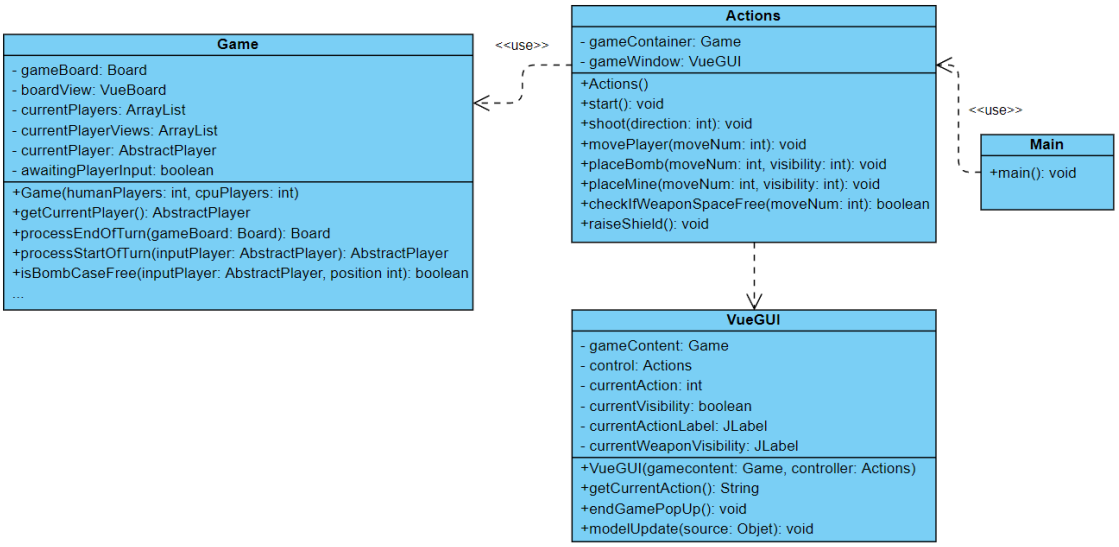
1.2 Présentation du jeu et de ses règles

Le jeu comporte de nombreuses règles.
Pour commencer, il y a 5 actions possibles : Se déplacer d'une case (4 directions possibles, haut, bas, gauche, droite) ; Déposer une arme (Mine ou Bombe) sur 8 cases voisines ; La possibilité d'effectuer un tir, horizontalement ou verticalement ; Déclencher un bouclier, qui permet de se protéger des tirs et des bombes lors du tour suivant ; Ou ne rien faire pour économiser son énergie.
Il y a donc plusieurs armes dans le jeu : Un tir, une bombe, et une mine. La mine explose lorsque qu'un combattant se place sur la case qu'elle occupe. La bombe, de son côté, explose au bout d'un temps t ou quand un joueur se place dessus. A la différence de la mine, la bombe impacte sur ses 8 cases voisines. Le tir, tant qu'à lui, impacte les combats sur la même ligne horizontale ou verticale.
Toute explosion ou tir diminue l'énergie d'un combattant.
Une des particularités des bombes et des mines, c'est qu'elles peuvent être visible pour tous les combattants, ou seulement visible pour le combattant qui l'a déposé.
Un déplacement ainsi que l'utilisation du bouclier diminuent également l'énergie.
Un joueur perd la partie lorsque l'énergie de son combattant est négatif ou nulle. Au contraire, un joueur gagne la partie lorsqu'il est le seul combattant encore en vie.

2 Elements techniques

2.1 Le modèle MVC

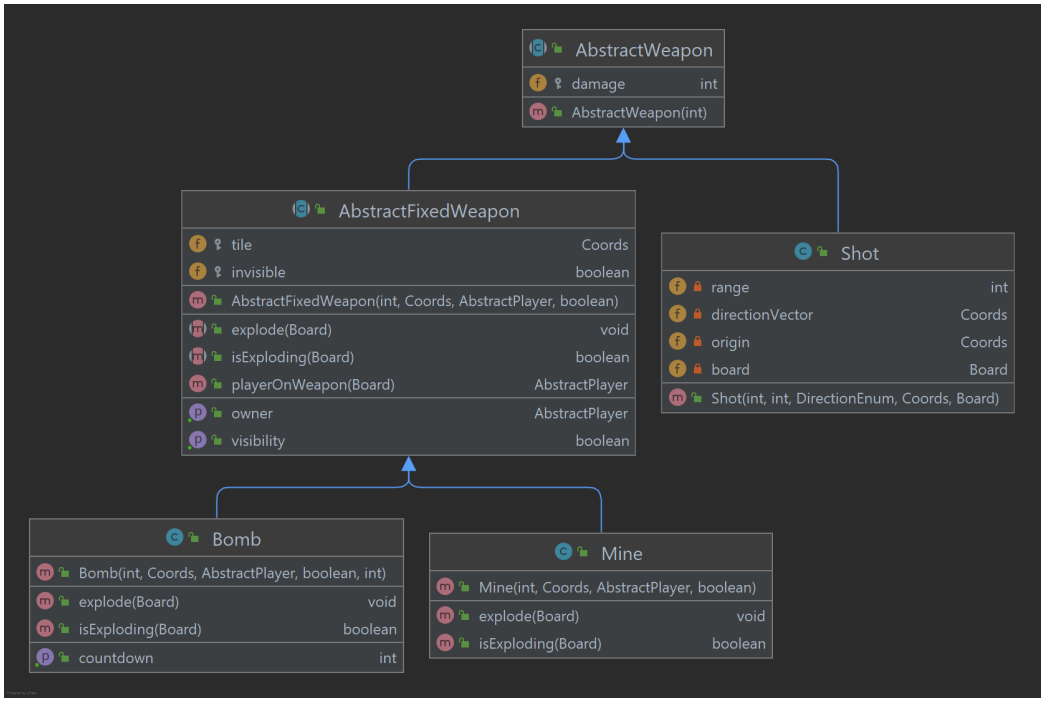
L'objectif et la notion technique la plus importante de ce projet était l'élaboration d'un modèle MVC organisé et fonctionnel. En effet, ce modèle est très adapté à la conception de jeu car il permet de séparer trois éléments majeurs d'un programme : Le stockage des données, l'affichage graphique des données et les actions à effectuer lorsque l'utilisateur interagit avec le programme. Cela permet, en plus d'avoir une arborescence claire, d'avoir un code facilement modulable, modifiable et compréhensible.



2.2 Objets

2.2.1 Les "armes"

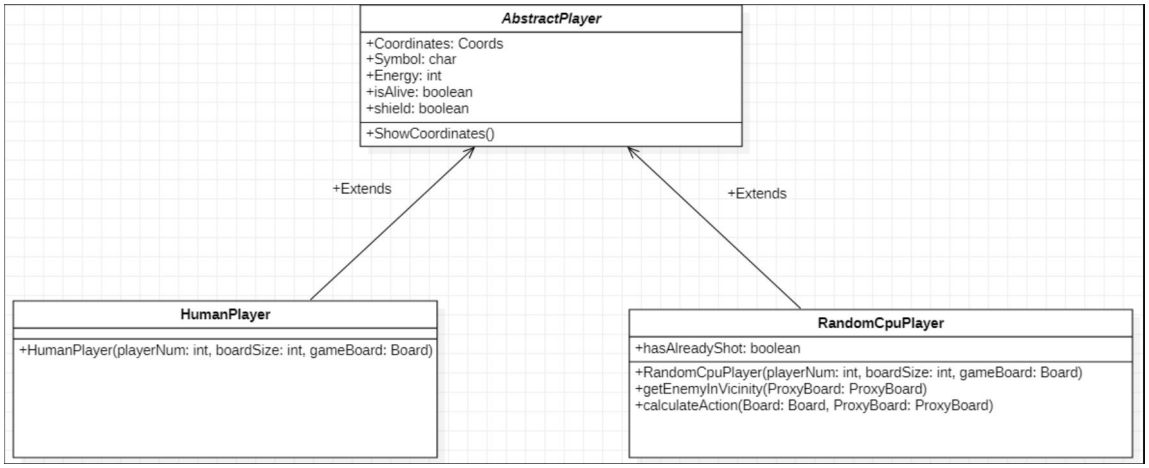
Diagramme de class du sous package **weapons** permettant la création des mines, bombes et définition des tirs :



Ce qui est important de voir dans le sous-package **Weapons**, c'est la séparation des armes *fixes*, et *mobiles*. C'est pour cela que nous avons créé la classe abstraite **AbstractFixedWeapon**, car *Bomb* et *Mine* partagent ces caractéristiques communes.

2.2.2 Les types de joueurs

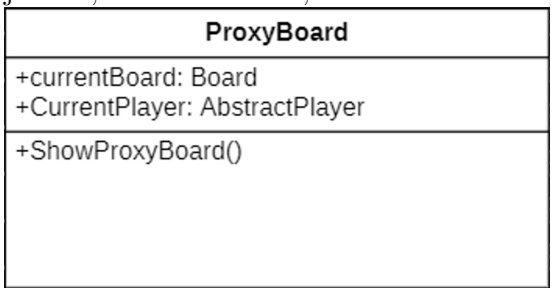
Pour permettre au jeu d'utiliser des joueurs humains, et des joueurs contrôlés par l'ordinateur, nous proposons une classe abstraite "AbstractPlayer", qui contient des variables et fonctions de base. Notre classe "HumanPlayer" est la classe utilisée pour construire des joueurs qui peuvent être contrôlés par des humains. Etant donné que toutes les actions ou coups seront effectués via l'interface graphique pour les joueurs humains, la classe HumanPlayer ne nécessite aucune fonction ou variable supplémentaire. Contrairement à la classe "RandomCpuPlayer", qui contrôle les joueurs ordinateurs, nécessitant plusieurs variables et fonctions supplémentaires pour pouvoir interpréter l'état actuel du plateau du jeu (getEnemyInVicinity recherche si un joueur adversaire se trouve sur la même ligne ou colonne que le joueur actuel), et décide ensuite de la prochaine action à effectuer. A la suite, ce n'est pas du "vrai" aléatoire, mais les calculs effectués permettent d'éviter les coups qui ne servent à rien (déposer une mine en dehors du plateau par exemple).


AbstractPlayer
+Coordinates: Coords
+Symbol: char
+Energy: int
+isAlive: boolean
+shield: boolean
+ShowCoordinates()
+Extends
+Extends
HumanPlayer
+HumanPlayer(playerNum: int, boardSize: int, gameBoard: Board)
RandomCpuPlayer
+hasAlreadyShot: boolean
+RandomCpuPlayer(playerNum: int, boardSize: int, gameBoard: Board)
+getEnemyInVicinity(ProxyBoard: ProxyBoard)
+calculateAction(Board: Board, ProxyBoard: ProxyBoard)

2.3 Pattern utilisé

2.3.1 Proxy

Parmis les patterns que nous avons utilisés, le pattern Proxy nous a permis de camoufler les actions qui n'appartiennent pas au joueur actuel, tel que les bombes, les mines ou encore par exemple l'activation du bouclier. Il fait une copie de la grille et vérifie à chaque "objet" si il appartient au joueur, si c'est le cas, on lui affiche "l'objet" sinon, on affiche un "vide".



2.4 Deux types d'affichage

2.4.1 Affichage dans la console

Voici ce que nous affiche le programme dans la console :

```
Global board
- - # - - - #
- - 0 - - - 0
- - - - - - 1 0
- - # - - # -
- - 2 - - - 0
- - - - - - - -
- - - - 0 - #
------
Here's what player 1 sees (before turn):
Player is currently at: 2 - 5
Current proxy player: 1
- - # - - - #
- - 0 - - - 0
- - - - - - 1 0
- - # - - # -
- - 2 - - - 0
- - - - - - - -
- - - - 0 - #
------
Human player!
Fin du tour
Human turn ended
Here's what player 1 sees (after turn):
Current proxy player: 1
- - # - - - #
- - 0 - - - 0
- - - - - - 1 0
- - # - - # -
- - 2 - - - 0
- - - - - - - -
- - - - 0 - #
------
Ending turn
```

Le joueur 1 est representé par un 1, tandis que le joueur 2 par un 2.
Les 0 sont des obstacles, tandis que les # sont des énergies.

2.4.2 Affichage via interface graphique : Java Swing

— Le bouton **Attendre** permet de passer son tour pour récuperer de l'énergie, changer d'action permet de changer entre effectuer l'action "tirer", "poser une bombe" et "poser une mine". La direction dans laquelle l'action sera effectuée est déterminée par les 8 boutons directionnels disposés sur la droite du JPanel en gris.
— Le bouton **Changer de visibilité** permet de changer la visibilité d'une action, "Seul au joueur" et "Visible par tous", cela concerne le fait de poser une mine.
— Le bouton **Bouclier** permet au joueur d'utiliser le bouclier.
— Pour finir à gauche du panel, nous avons 4 touches directionnelles qui permettent de déplacer le joueur.

3 Conclusion

Ce projet fut une bonne expérience pour nous tous. Nous avons pu mettre en oeuvre les connaissances acquises au courant de ce semestre (les différents patterns, mais également le modèle MVC).
Malgré cela, nous n'avons pas pu finaliser la partie visuelle graphique de notre projet, faute de temps. En effet, nous avons eu tous un peu de mal avec l'utilisation de JAVA Swing, en particulier pour afficher visuellement notre plateau de jeu (fonctionnel dans la console).