BAB 1 - Sciences Informatiques
Rapport du Projet d'informatique 2018 - 2019
Création d'un jeu de plateau Stratego[1]

Oussama Ouanane, Sofyane Timsal

2019
Vendredi 17 Mai

1. Stratego est un jeu de société de stratégie et de bluff, créé en 1947 (dérivé du jeu L'Attaque, breveté par Hermance Edan en 1909). `https://fr.wikipedia.org/wiki/Stratego`

Table des matières

Première partie

Introduction

Chapitre 1

Présentation du projet

1.1 Présentation du projet

Dans le cadre du cours de Projet d'informatique, il nous a été demandé de créer une réplique du jeu de plateau 'Stratego' afin de mettre en pratique les enseignements donnés en Programmation et Algorithmique I/II.

1.2 Rappel des règles du jeu

Le *Stratego* a l'allure d'un jeu complexe mais il n'en est rien. Il se joue à deux personnes et est plus communément ce qu'on appelle dans le jargon des jeux-vidéos, un *Turn-based game*. Chaque joueur possède 40 pions, les pions subissent une hiérarchie basée sur la puissance comme suit :

Puissance	Nom du pion
1	Drapeau
2	Espion
3	Éclaireur
4	Démineur
5	Sergent
6	Lieutenant
7	Capitaine
8	Commandant
9	Colonel
10	Général
11	Maréchal
12	Bombe

Cependant il existe deux exceptions :

Pion	Pion	Pion gagnant
Bombe	Démineur	Démineur
Maréchal	Espion	Espion

Chaque pion possède la capacité de se déplacer horizontalement et verticalement d'une case, cependant, l'éclaireur, peut se déplacer d'autant de cases qu'il veut tant qu'il ne chevauche aucun pion. Si un pion se trouve dans la case du déplacement, alors un combat surgit et le pion gagnant est calculé selon hiérarchie des pions qui se trouve en page 3.

1.3 Mini-guide utilisateur

La compilation et le lancement du jeu se font avec Apache Ant (`https://ant.apache.org/`), les commandes disponibles sont :

Nom de la commande	Fonction
clean	Permet de nettoyer le répertoire et supprimer les fichiers de compilation
build	Permet de compiler les fichiers Java (avec javac)
run	Permet d'exécuter le jeu [REQUIERT D'AVOIR 'build']
test	Permet de lancer les tests JUnit
jar	Permet de générer un exécutable JAR
javadoc	Permet de générer la Javadoc

Une fois arrivé au menu principal, il vous est possible de démarrer une partie contre l'une des deux AI disponibles ou de charger une sauvegarde déjà existante. La première phase consistera à placer les pions, vous pouvez modifier si vous avez fait une mauvaise manipulation, suffit de cliquer sur le pion sur la case et sur le pion voulu dans 'Placement de pions'.

Une fois le placement terminé, cliquer le pion que vous voulez déplacer fera apparaître les pions de l'intelligence artificielle et l'affichage du score.

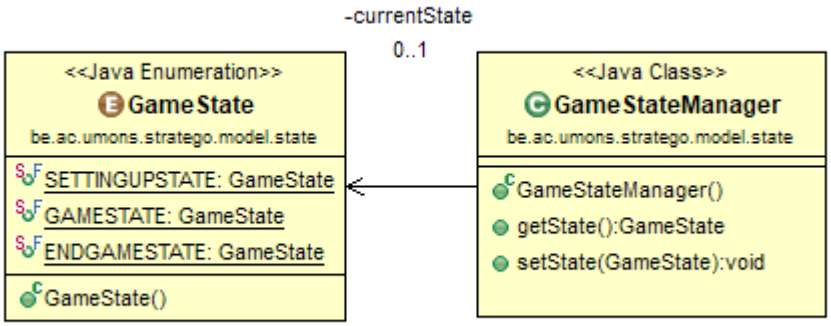
Chapitre 2

Processus de développement

2.1 Fonctionnement

La première étape a été d'écrire sur papier, tous les éléments qui surgissent lors d'une partie *Stratego*, on déduit assez facilement que le jeu se compose en trois étapes distinctes. On a d'abord le placement des pions, un état où, les joueurs sont libres de placer leurs pions comme bon leur semble. Ensuite, on a le déroulement du jeu, cette étape ne s'arrête pas tant qu'un des joueurs n'a gagné, on marque la fin d'un tour à chaque déplacement. Enfin, le jeu marque sa fin, cette étape peut-être interprété comme un lot de conditions qu'on vérifie à chaque tour.

Dans le but de créer ce mécanisme, une énumération et une classe ont été créées :



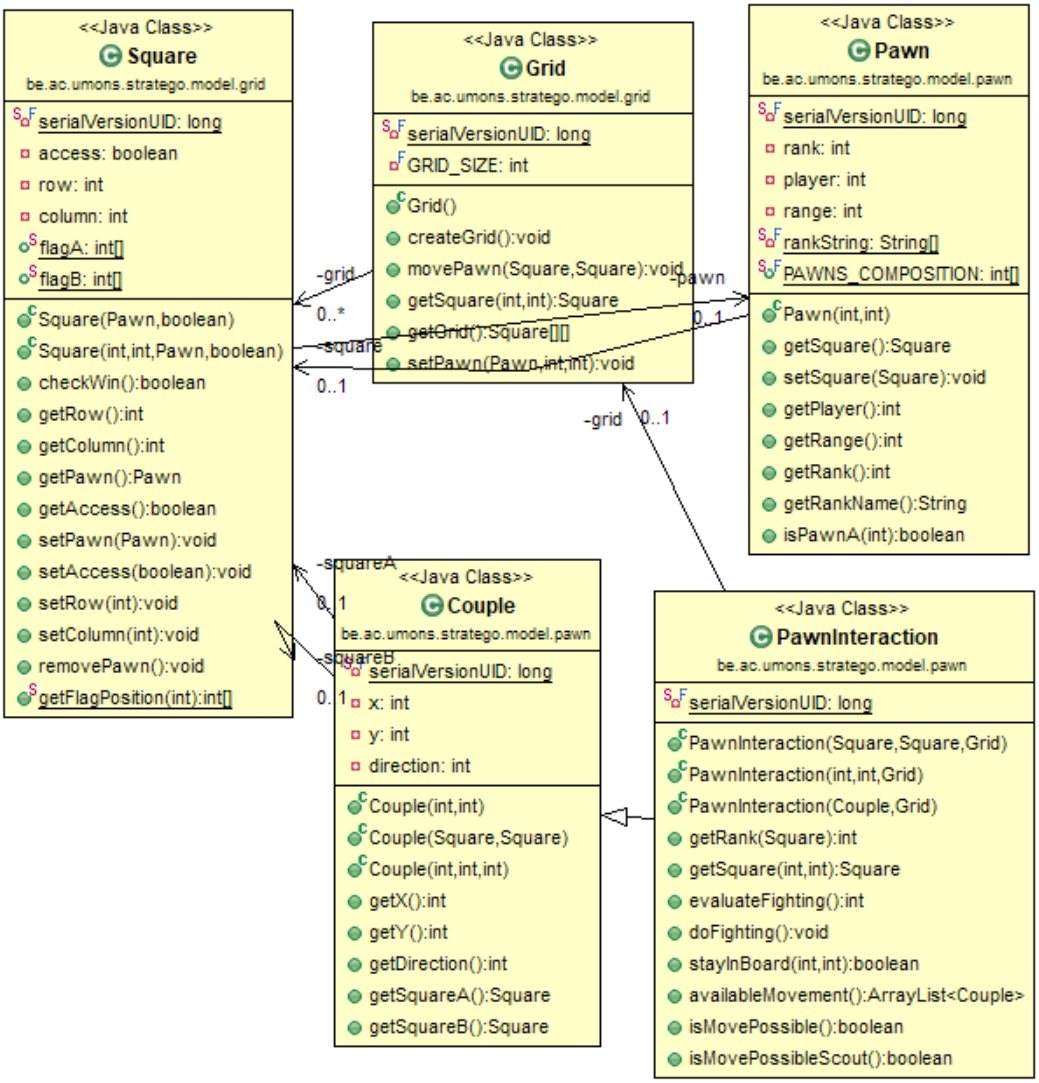
GameState sert à représenter les trois états :

- **SETTINGUPSTATE** : le placement des pions
- **GAMESTATE** : le déroulement du jeu
- **ENDGAMESTATE** la fin du jeu

Quant à la classe GameStateManager, elle sert à récupérer/mettre à jour l'état. Chaque état possède son affichage, par exemple, entre **SETTINGUPSTATE** et **GA-**

MESTATE, un score s'affiche et l'ancien panneau qui servait de conteneur pour les pions à placer est vidé. **ENDGAMESTATE** va lui, afficher une fenêtre pour mettre fin à la partie. Lire la classe *be.ac.umons.stratego.view.GameView* pour plus d'informations.

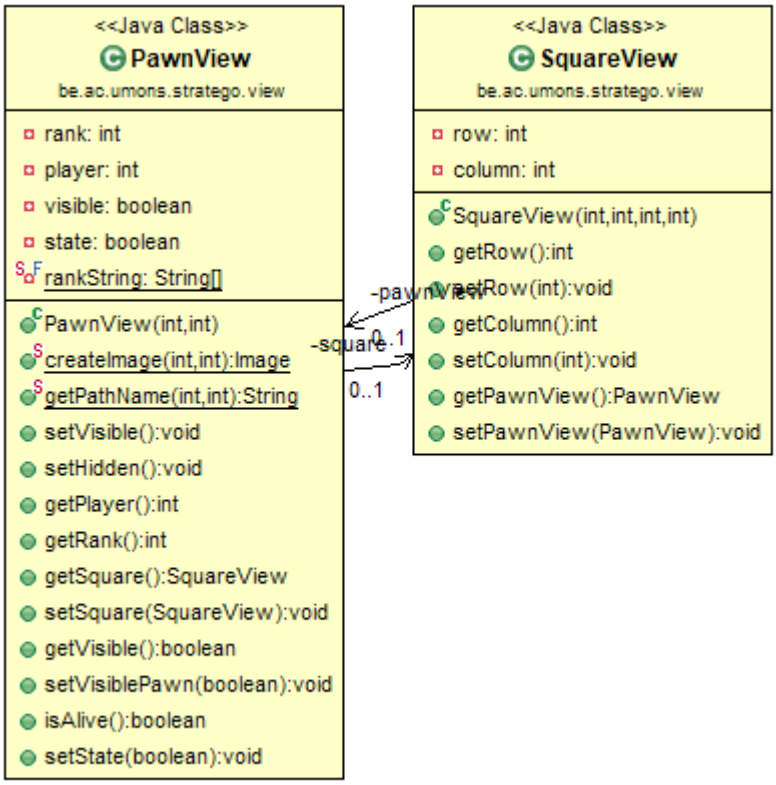
Ceci étant fixé, il fallait trouver un moyen de représenter chaque composante, les pions, le plateau,... Pour cela, il fallait d'abord réfléchir à une architecture, on a finalement décidé de partir sur l'architecture classique ***MVC*** *(model-view-controller)* - Model ayant servi pour la représentation logique des composantes :



— **Square** : permet de représenter une case du plateau (formellement)
— **Pawn** : permet de représenter un pion (formellement)
— **Grid** : permet de représenter le plateau (matrice de 100 *Square*)

— **PawnInteraction** : permet de représenter les actions d'un *Pawn*

Passons maintenant à View, le framework utilisé pour représenter le jeu graphiquement est JavaFX, Swing étant une vieille technologie et ne se situant pas dans l'ère de temps, notre choix s'est donc naturellement porté vers JavaFX (`https://openjfx.io/`). View a servi pour la représentation graphique des composantes :



— **PawnView** : permet de représenter un Pawn (graphiquement)
— **SquareView** : permet de représenter un Square (graphiquement)

2.2 Répartition des tâches

On ne savait pas comment répartir les tâches donc chaque classe du projet a été réalisé par nous deux, ce n'est certes pas la manière la plus efficace, mais au moins chacun de nous avait le mérite de bien comprendre le fonctionement de notre jeu. Oussama Ouanane avait un peu d'expérience avec le fonctionnement des interfaces graphiques car très similaires à ceux qu'il employait en C Sharp, ce qui a facilité la tâche. Nous avons utilisé Git pour gérer les mises à jours.

Deuxième partie

Conclusion

Chapitre 3

Points forts et faibles

3.1 Points forts

— L'ensemble des ressources graphiques ont été créé par nous-même.
— Génération de la javadoc avec ant
— *User-friendly*

3.2 Points faibles

— S'il y a besoin de rajouter un mode de jeu, ce sera compliqué car le code est très peu modulaire.
— Le jeu peut être lent suivant la configuration de l'hôte.
— Aucune fonctionnalité supplémentaire.

3.3 Problèmes connus

Lors du lancement de tests unitaires, il arrive parfois que certains tests plantent et retournent "crash error". Il arrive aussi extrêmement rarement qu'un pion disparaisse lors du placement des pions (ce n'est arrivé qu'une seule fois).

3.4 Apports positifs

La constante exposition à l'orienté-objet permet de très bien comprendre le fonctionnement du cours de Programmation et Algorithmie II. Il permet aussi de se rendre compte des difficultés de gestion de logiciels, nous avons dû faire face à énormément de problèmes alors que la taille du projet n'est pas très grande.

3.5 Apports négatifs

La développement du projet est plutôt chronophage, non pas à cause de lacunes en programmation Java mais à cause du temps consacré à la compréhension de JavaFX. Très peu de ressources en ligne qui nous ont fait perdre beaucoup trop de temps.

3.6 Remerciements

Merci à Rémy Decocq, élève en Master 1 en Sciences Informatique pour les précieux conseils.