Enzo Allemano

Yohann Paulus

Donini Pier

Decorvet Grégoire

07.01.2021

**Programmation Orientée Objet**

**Rapport Laboratoire 8 :**

**Echecs**

Contents

[Description des classes 3](#_Toc60868448)

[Package « mouvements » 3](#_Toc60868449)

[Package « pieces » 4](#_Toc60868450)

[« Case » et « Plateau » 4](#_Toc60868451)

[Implémentation 4](#_Toc60868452)

[Les mouvements simples 4](#_Toc60868453)

[Les mouvements complexes (Pions) 5](#_Toc60868454)

[Les mouvements complexes (cavalier) 5](#_Toc60868455)

[La promotion 5](#_Toc60868456)

[L’échec 5](#_Toc60868457)

[La prise en passant 6](#_Toc60868458)

[Le grand et le petit roque 6](#_Toc60868459)

[La fonction « move » du plateau 6](#_Toc60868460)

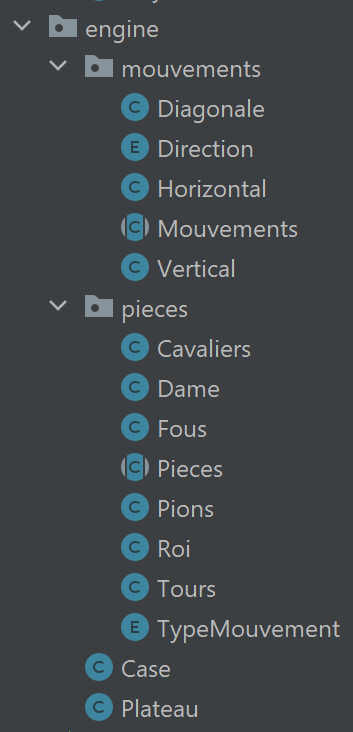
[Spécificités 6](#_Toc60868461)

[Remarque et conclusion 6](#_Toc60868462)

# Description des classes

L’arborescence des classes que nous avons implémentées se présente de la façon suivante. Comme demandé dans la donnée du laboratoire nous avons créé un package « engine » dans lequel se trouve toutes les classes nécessaires au contrôleur.

Le package « engine » contient la classe « Case » et « Plateau », il contient aussi deux autres packages, « mouvements » et « pieces », qui nous servent à mieux séparer les différentes classes du utilisées.



## Package « mouvements »

Ce package implémente tous les mouvements que peux faire une pièce (excepté cavalier) ainsi que les directions que peut prendre un mouvement.

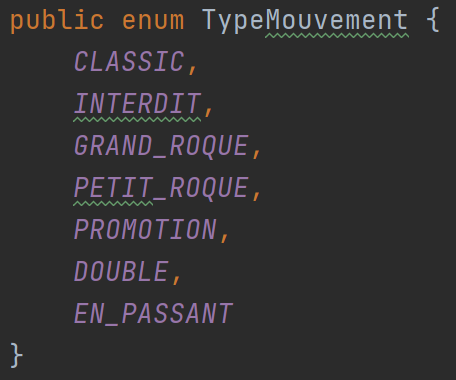
On déclare la classe abstraite « Mouvements » qui implémente la fonction trajectoireLibre et qui a pour attribut une direction de type Direction. Les directions sont implémentées dans l’enum Direction.

Ainsi on aura un mouvement différent pour chaque direction :

* Vertical GAUCHE
* Vertical DROITE
* Horizontale GAUCHE
* Horizontale DROITE
* Diagonale HAUTE\_GAUCHE
* Diagonale HAUTE\_DROITE
* Diagonale BAS\_GAUCHE
* Diagonale BAS\_DROITE

## Package « pieces »

Ce package implémente toutes les pièces que contient un jeu d’échecs ainsi que les types de mouvements qu’une pièce pourra faire durant une partie.



On déclare la classe abstraite « Pieces » qui implémente la fonction mouvementValide puis des seter et des geter pour ses attributs.

## « Case » et « Plateau »

Ces deux classes se trouvent dans le package « engine ».

La classe « case » possède un position x et y ainsi qu’un attribut pièce. Nous pouvons donc peupler l’échiquier de 64 cases. Cette classe possède notamment diverses fonctions pour récupérer et définir les informations sur ladite case.

La classe « plateau » nous permet d’initialiser et jouer une partie, mais pas de la terminée (fonction échec et mat pas implémentée). Elle contient notamment les fonctions pour *roquer*, pour *la prise en passant*, pour *la promotion*, pour la détection d’un échec et les attributs pour l’interface graphique.

# Implémentation

## Les mouvements simples

Chaque pièce possède une liste de mouvements. Par exemple, la tour possède les mouvements :

* horizontal GAUCHE
* horizontal DROITE
* vertical HAUT
* vertical BAS

On peut donc facilement encadrer les déplacements d’une pièce pour la restreindre aux déplacements autorisés.

Pour vérifier si son déplacement est valide on utilise principalement deux méthodes :

* mouvementValide : qui se trouve dans la classe abstraite « Pieces » et qui vérifie les mouvements valides en fonction des mouvements disponibles de la pièce.
* trajectoireLibre : qui se trouve dans la classe « Mouvements » et qui est surchargée dans chaque mouvements.

## Les mouvements complexes (Pions)

Le pion à des déplacement bien spéciaux. Premièrement, il peut se déplacer de 2 uniquement lors du premier tour. Nous avons donc défini la distance qu’il pouvait parcourir à chaque tour à 2. Ensuite, lors de son premier déplacement nous devons faire deux choses :

* Lui retirer son droit de déplacer de 2
* Se souvenir s’il a fait un coup double (déplacement de deux cases) ou pas.
  + Utile pour la prise en passant

Deuxièmement, le pion ne peut pas manger en utilisant les directions avec lesquelles il peut se déplacer. Nous lui avons donc attribuer en plus les déplacements diagonaux HAUTE\_GAUCHE et HAUTE\_DROITE et veillons à ce qu’ils ne les utilisent que pour manger.

Pour se déplacer tout droit, la fonction trajectoireLibre de « Vertical » vérifie que lorsqu’elle est appelée par un pion et que la case devant se pion est occupée par un adversaire la fonction retourne false (la trajectoire n’est pas libre).

## Les mouvements complexes (cavalier)

Vu que le cavalier peut « sauter » par-dessus d’autres pièces, ces déplacements sont facilités. Il n’est pas nécessaire de vérifier si le chemin qu’il emprunte est libre. La méthode mouvementValide est donc redéfinie pour les objets « cavaliers ». Le seul contrôle établis est pour s’assurer que le déplacement souhaité respecte la forme de L avec des proportions 2 pour 1.

## La promotion

La promotion d’un pion se déroule comme suit; la fonction mouvementValide est surchargée à l’intérieur de la classe « Pions » pour pouvoir tester si le mouvement que le pion courant s’apprête à faire est susceptible de provoquer une promotion. Si non, on retourne un mouvement de type classique ou interdit(selon le mouvement). Si oui, il s’agit bien d’une promotion, alors on retourne un mouvement de type promotion.

*Pour tester si on promotion aura lieu il suffit de regarder si le pion courant s’apprête à aller sur la première ou la dernière ligne de l’échiquier. Il n’y a pas besoin de faire de distinction entre le joueur noir et blanc car les pions ne peuvent reculer dans aucun cas.*

Si la fonction « move » du plateau détecte un mouvement de type promotion, alors on rentre dans la fonction « promouvoir » du plateau. Dans ce cas, on propose à l’utilisateur une dame, un cavalier, une tour ou un fou. Le pion est supprimé et remplacé par la sélection de l’utilisateur.

## L’échec

Afin de déterminer si notre roi est en danger, nous avons implémenter la fonction « echec ». Cette dernière parcoure tout l’échiquier et dès qu’elle trouve une pièce de notre adversaire, elle regarde si se déplacer sur la case actuelle de notre roi est un coup autorisé pour cette dernière. Si c’est le cas, le message s’affiche, nous sommes en échec. Les seuls coups autorisés sont alors des coups qui mettent notre roi à l’abri. Une fois notre roi protégé le message disparait et la partie continue normalement.

## La prise en passant

Pour rendre possible ce mouvement, nous avons dû stocker le type de mouvement du dernier coup qu’un pion ai joué et également le dernier tour ou il a été joué. De ce fait nous pouvons nous assurer qu’un pion effectuant cette prise ne puisse le faire uniquement au tour suivant le déplacement de deux cases du pion adverse. Pour le reste nous contrôlons juste que la case en diagonale soit vide et que le pion que l’on va manger se trouve belle est bien sur la même ligne que le pion qui s’apprête à manger.

## Le grand et le petit roque

Pour ce coup crucial lors d’une partie d’échec, nous avons adaptés la méthode mouvementValide pour les rois. Nous commençons par appeler la méthode super(), si cette dernière nous retourne un déplacement interdit, avant de retourner ce résultat, nous testons si l’utilisateur essais de roque. Lors d’un roque, nous déplaçons le rois case par case, cela nous permet de vérifier à chaque étape du processus si le roi se retrouverait en échec ce qui rendrait le roque illégal. Le reste des mécaniques est assez simple.

## La fonction « move » du plateau

Afin de mieux comprendre cette « grande » fonction nous avons mis en annexe un pseudo code de celle-ci.

Move.png contient le pseudo code et Plateau.png contient les variables pour le mise en place du pseudo code.

# Spécificités

La classe « Pieces » implémente « ChessView.UserChoice » pour nous permettre de donner le choix de sélectionner une pièce à l’utilisateur lors d’une promotion.



# Remarque et conclusion

L’ensemble de la réalisation de ce laboratoire c’est bien passé. La réalisation du rapport ainsi que la création de l’UML nous ont permis d’avoir une bonne vue d’ensemble sur notre projet.