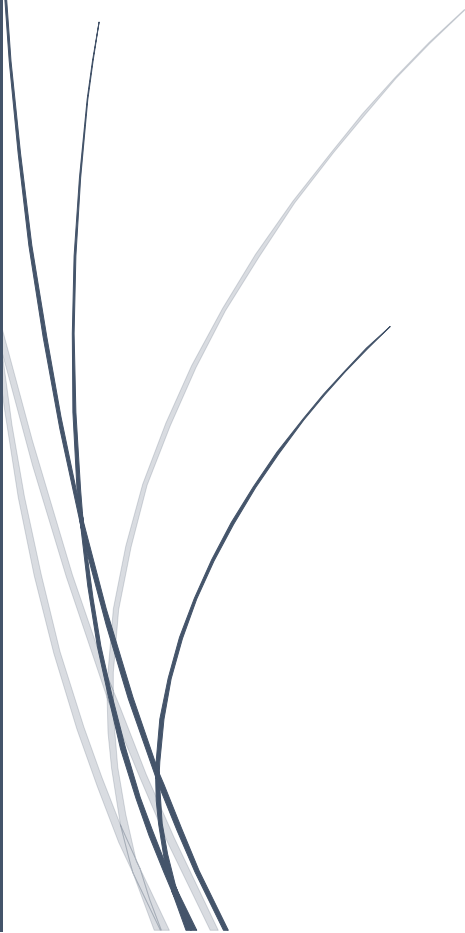


A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

13/01/2022

# Rapport Laboratoire 8 : Echecs

Several thin, curved lines in dark blue and light grey originate from the bottom left corner and sweep upwards and to the right.

Valentin Kaelin et Jonathan Friedli

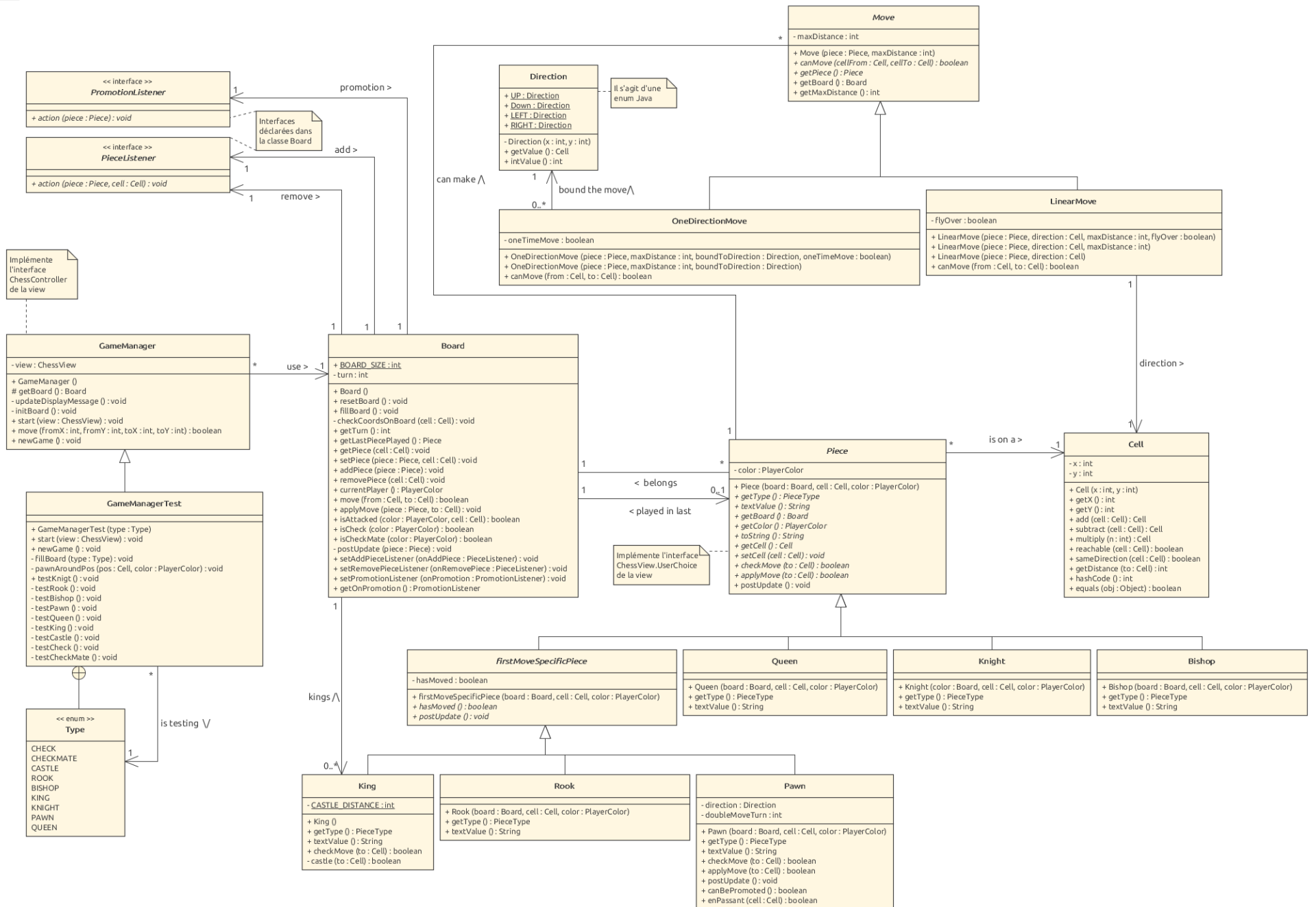
## Table des matières

Introduction .....	2
Diagramme des classes .....	3
Choix de modélisation et d'implémentation .....	4
GameManager.....	4
Board.....	4
Système d'événements .....	4
Stockage des pièces.....	4
Liste de rois .....	4
Compter les tours .....	5
Echec et mat.....	5
Cell .....	5
Moves.....	5
Piece.....	5
Interface UserChoice .....	5
postUpdate() .....	5
Déplacements spéciaux .....	5
FirstSpecificMove .....	6
Pawn.....	6
Tests effectués .....	6
Tests généraux .....	6
Tests de la Reine.....	7
Tests du Roi .....	7
Tests de la Tour .....	7
Tests du Fou .....	7
Tests du Cavalier .....	8
Tests du Pion .....	8

## Introduction

Dans le cadre de ce laboratoire n°8, nous avons dû implémenter un programme permettant de jouer aux échecs. Deux interfaces nous étaient fournies (une interface graphique ainsi qu'un mode console) et notre but a été de réaliser toute la partie logique du jeu. Pour finir, l'implémentations bonus de l'échec et mat a été réalisée contrairement à celle du pat.

Chess



## Choix de modélisation et d'implémentation

### GameManager

Nous avons séparé la gestion de la communication avec la vue du véritable plateau pour plusieurs raisons. Tout d'abord, cela nous a permis de bien séparer les différents contextes. En effet, la classe *GameManager* implémentant l'interface *ChessController* est la seule à interagir avec la vue et les autres classes communiquent avec elle via un système d'événements sur lequel nous reviendrons en détails plus tard. De plus, cela a été plus cohérent afin de transmettre aux différentes pièces le *Board* sur lequel elles se trouvent au lieu de leur passer une classe contenant plus de fonctionnalités et ayant moins de rapport avec elles.

La seule vraie part de logique dans la classe *GameManager* est la vérification qu'un roi est en échec. Il aurait également été possible de vérifier à chaque fin de tour directement dans la classe *Board* et mettre un accesseur à disposition mais notre choix plus simple nous a semblé raisonnable.

Lorsque l'utilisateur appuie sur le bouton « New Game », nous remettons à zéro l'état du plateau afin de ne pas avoir à recréer une nouvelle instance de la classe *Board*.

### Board

#### Système d'événements

La classe *Board* émet des événements qu'écoute la classe *GameManager* dans plusieurs cas :

- Une pièce a été supprimée d'une case.
- Une pièce a été ajoutée sur une case.
- La promotion d'un pion est demandée.

Cela nous permet d'avoir une interface graphique à jour sans devoir manuellement ajouter/supprimer les pièces de la vue à chaque modification du plateau virtuel.

Les interfaces des Listeners ont été définies comme internes à la classe *Board* car il est peu judicieux de les ajouter chacune dans un fichier séparé en perdant le contexte qu'elles sont liées à la classe *Board*.

#### Stockage des pièces

Les pièces sont stockés dans un simple tableau à deux dimensions afin d'y accéder en  $O(1)$  lors des recherches. Il aurait également été possible de faire une *HashMap* mais cela nous a semblé compliquer inutilement l'implémentation étant donné que les états des pièces ne sont pas forcément fixes.

Le tableau contenant les pièces est de taille fixe et cette taille est stockée dans une constante. Nous avons décidé de ne pas passer en paramètre du constructeur les dimensions souhaitées du tableau car le jeu d'échecs perd tout sans sens avec un plateau d'une autre taille. De plus, les différentes vues ne permettent pas non plus cette extension.

#### Liste de rois

En plus du tableau à deux dimensions, une *ArrayList* contenant les deux rois a été créée afin de faciliter la vérification de la mise en échec. En effet, cela nous évite de devoir parcourir le double tableau à la recherche du roi qui nous intéresse. Cette liste est mise à jour à chaque mouvement d'une pièce de type roi car autrement cela posait problème pour notre classe de tests. En effet, nous n'avons pas accès la liste des rois et nous ne pouvons donc pas les ajouter à cette dernière lors de la disposition initiale. Créer une méthode *fillBoard()* supplémentaire prenant un tableau de pièces pourrait être une

solution mais cela posait un nouveau problème avec notre méthode utilitaire permettant d'ajouter des pions autour d'une position.

### Compter les tours

Nous avons un compteur s'incrémentant après chaque tour afin de récupérer le joueur actuel ainsi que pour faciliter la gestion d'un des cas de nulle pour une future implémentation (50 coups sans avance de pion ni de prise). Ce compteur est également utile lors de l'implémentation de la prise en passant.

### Echec et mat

Nous avons implémenté l'échec et mat en bouclant sur toutes les cases. Si une case de l'équipe du roi en échec peut y aller, cela signifie que ce n'est pas un échec et mat car le mouvement annule donc la position d'échec du roi allié. La seule subtilité à prendre en compte a été d'annuler le mouvement si celui-ci a bien pu être appliqué afin de ne pas jouer à la place du joueur

### Cell

Une classe représentant une case du plateau a été réalisée afin d'offrir plus de possibilités. Il aurait été possible de stocker directement les deux coordonnées x et y dans la pièce mais cela nous aurait embêtés dans les actions suivantes :

- Réaliser des opérations mathématiques entre deux cases, afin de par exemple itérer sur toutes les cases d'un déplacement.
- Comparer facilement 2 positions (*.equals()*).
- Vérifier si une case est accessible depuis une autre.
- Calculer la distance entre deux cases.

### Moves

De nombreux constructeurs ont été mis à disposition afin d'éviter à l'utilisateur de la classe de devoir entrer des valeurs par défaut qui n'ont pas énormément de sens.

La classe *LinearMove* nous a permis de gérer à la fois les mouvements horizontaux et verticaux, les diagonales et les déplacements du cavalier grâce à l'utilisation des diverses méthodes de la classe *Cell*.

### Piece

#### Interface UserChoice

Les pièces implémentent l'interface *UserChoice* de la vue afin de faciliter la gestion du choix de la pièce lors de la promotion. Le point négatif de cette approche est qu'on lie la vue à l'implémentation logique alors qu'on avait par avant bien réussi à faire la distinction des deux. Il aurait sûrement été possible de créer des types d'abstraction au-dessus des pièces lors de la promotion mais cela nous a encore une fois semblé se compliquer beaucoup la vie.

#### postUpdate()

La méthode n'est pas abstraite car elle n'est pas redéfinie dans toutes les pièces. Il est donc plus facile de laisser simplement son corps vide. Elle gère les différents aspects à potentiellement appliquer à une pièce une fois celle-ci jouée.

### Déplacements spéciaux

Les déplacements spéciaux tels que les roques ou la prise en passant sont directement implémentés dans la pièce spécifique. Il aurait été possible de créer une abstraction en ajoutant des classes à part comme pour les autres mouvements mais comme ceux-ci sont utilisés dans une seule et unique classe, nous sommes allés droit au but.

### FirstSpecificMove

Une deuxième couche de classe abstraite a été implémentée afin de pouvoir gérer les 3 types de pièces pour lesquelles le premier déplacement nous est important (roi, pion, tour). Cela nous a permis d'éviter d'avoir un attribut boolean en plus dans les 7 classes ou avoir à répéter cet attribut dans les 3 classes citées ci-dessus.

### Pawn

C'est le pion lui-même qui vérifie s'il peut être promu dans la méthode *postUpdate()* afin de ne pas avoir à déléguer cette tâche. Par conséquent, un accesseur au listener de la classe *Board* a dû être implémenté.

## Tests effectués

Nous allons lister par la suite les différents tests que nous avons réalisés, en plus d'avoir joué quelques parties complètes. Il est possible d'avoir des créer des situations facilitant le test des différents fonctionnements en exécutant la classe *MainTest*. Dans cette classe nous faisons appel à une classe *GameManagerTest* qui prend en paramètre la situation initiale que nous souhaitons tester.

### Tests généraux

Description du test	Résultat attendu et observé
Appuyer sur le bouton « New Game » de l'interface lance bien une nouvelle partie en effaçant convenablement la précédente.	OK
Au début de la partie, c'est au joueur blanc de jouer.	OK
Les joueurs alternent. Un message indiquant quel joueur doit jouer est visible dans l'interface.	OK
Au début de la partie, les pièces sont disposées correctement afin de pouvoir commencer une partie d'échecs.	OK
Si une pièce met le roi adverse en échec, un message s'affiche.	OK
Si le joueur est en échec et qu'il ne peut plus bouger, un message indiquant qu'il est en échec et mat et que la partie est terminée s'affiche.	OK
Les pièces peuvent prendre des pièces adverses dans la limite de leurs mouvements (les mouvements de chaque pièce sont détaillés plus loin dans les tests).	OK
Le petit et le grand roque sont fonctionnels. On ne peut pas roque, si le roi a déjà bougé, s'il est mis en échec si la tour avec laquelle on veut roque a déjà bougé ou si une des cases sur laquelle passe le roi est en échec.	OK
Les pièces ne peuvent pas aller sur une case occupée par une pièce de la même couleur.	OK
Une pièce ne peut pas réaliser un mouvement si celui-ci met son roi en échec.	OK

## Tests de la Reine

Description du test	Résultat attendu et observé
La reine peut faire des mouvements sur les côtés (haut, bas, gauche, droite) ainsi qu'en diagonale. Le mouvement doit être rectiligne.	OK
Les mouvements de la reine n'ont pas de limite de longueur (elle peut traverser l'échiquier si aucune autre pièce ne la bloque).	OK
La reine ne peut pas sauter par-dessus une autre pièce.	OK

## Tests du Roi

Description du test	Résultat attendu et observé
Le roi ne peut faire des mouvements que d'une case de long.	OK
Le roi peut bouger sur toutes les cases autour de lui (pour autant qu'aucune pièce ne le bloque et qu'il ne soit pas sur le bord de l'échiquier).	OK
Le roi ne peut pas aller sur case qui est attaquée par une pièce adverse.	OK

## Tests de la Tour

Description du test	Résultat attendu et observé
La tour peut faire des mouvements sur les côtés mais ne peut pas faire de mouvement en diagonal.	OK
Les mouvements de la tour n'ont pas de limite de longueur.	OK
La tour ne peut pas sauter par-dessus une autre pièce.	OK

## Tests du Fou

Description du test	Résultat attendu et observé
Le fou ne peut faire des mouvements uniquement en diagonal et pas sur les côtés.	OK
Les mouvements du fou n'ont pas de limite de longueur.	OK

Le fou ne peut pas sauter par-dessus une autre pièce.	OK
---	----

### Tests du Cavalier

Description du test	Résultat attendu et observé
Le cavalier ne peut faire que des mouvements en « L ». Le mouvement en « L » combine un déplacement de deux cases dans une direction suivi d'un déplacement à une case perpendiculairement.	OK
Le cavalier ne peut pas faire de mouvement du type 2 cases vers la droite puis encore une case vers la gauche ou vers droite. Il est obligé de faire le premier mouvement dans une direction (haut-bas ou gauche-droite) et le deuxième dans l'autre sens.	OK
Le cavalier peut sauter par-dessus une autre pièce.	OK
Le mouvement en L ne compte que comme un seul mouvement.	OK

### Tests du Pion

Description du test	Résultat attendu et observé
Les pions ne peuvent bouger que dans une seule direction (dans notre case : les blancs vers le haut et les noirs vers le bas).	OK
Le pion ne peut faire des mouvements que d'une case de long.	OK
Exception : Si le pion n'a pas encore bougé, il peut faire un mouvement de deux cases devant lui (il ne peut pas sauter par-dessus une autre pièce).	OK
Le pion ne peut prendre de pièce uniquement si la pièce adverse est en diagonal de ce dernier, bien entendu uniquement dans la direction initiale du pion.	OK
Exception : La prise dite « En passant » est implémentée. Ce coup s'effectue en prenant un pion ayant avancé de deux cases au tour précédent comme s'il n'avait avancé que d'une case.	OK
Si un pion atteint l'autre bout de l'échiquier, il peut faire une promotion. Les quatre choix de promotion sont : En Reine, en Tour, en Fou, en Cavalier.	OK
La promotion est possible peu importe le mouvement réalisé (ex : prise en diagonale ou mouvement classique en avant).	OK