Rapport de laboratoire

**HEIG-VD**

TIC

Laboratoire POO

Labo 8

**Réalisé par :**

Ivan Vecerina

Thibault Seem

Alice Grunder

**A l'attention de :**

M. Donini

M. Decorvet

**Dates :**

Début du laboratoire : 21 octobre 2021

Fin du laboratoire : 4 novembre 2021

Table des matières

[1 Introduction 3](#_Toc93521308)

[2 Diagramme de classe 3](#_Toc93521309)

[3 Choix de modélisation 3](#_Toc93521310)

[4 Tests 4](#_Toc93521311)

[4.1 Opérations avec matrices à valeurs aléatoires (0x0) 4](#_Toc93521312)

[4.1.1 M1 et M2 sont vides 4](#_Toc93521313)

[4.1.2 M1 vide et M2 non-vide 4](#_Toc93521314)

[4.1.3 M1 non-vide et M2 vide 4](#_Toc93521315)

[4.2 Opérations excédant limites de int 5](#_Toc93521316)

[4.3 Operations avec matrices de bases différentes 5](#_Toc93521317)

[4.4 Valeurs de paramètres problématiques 6](#_Toc93521318)

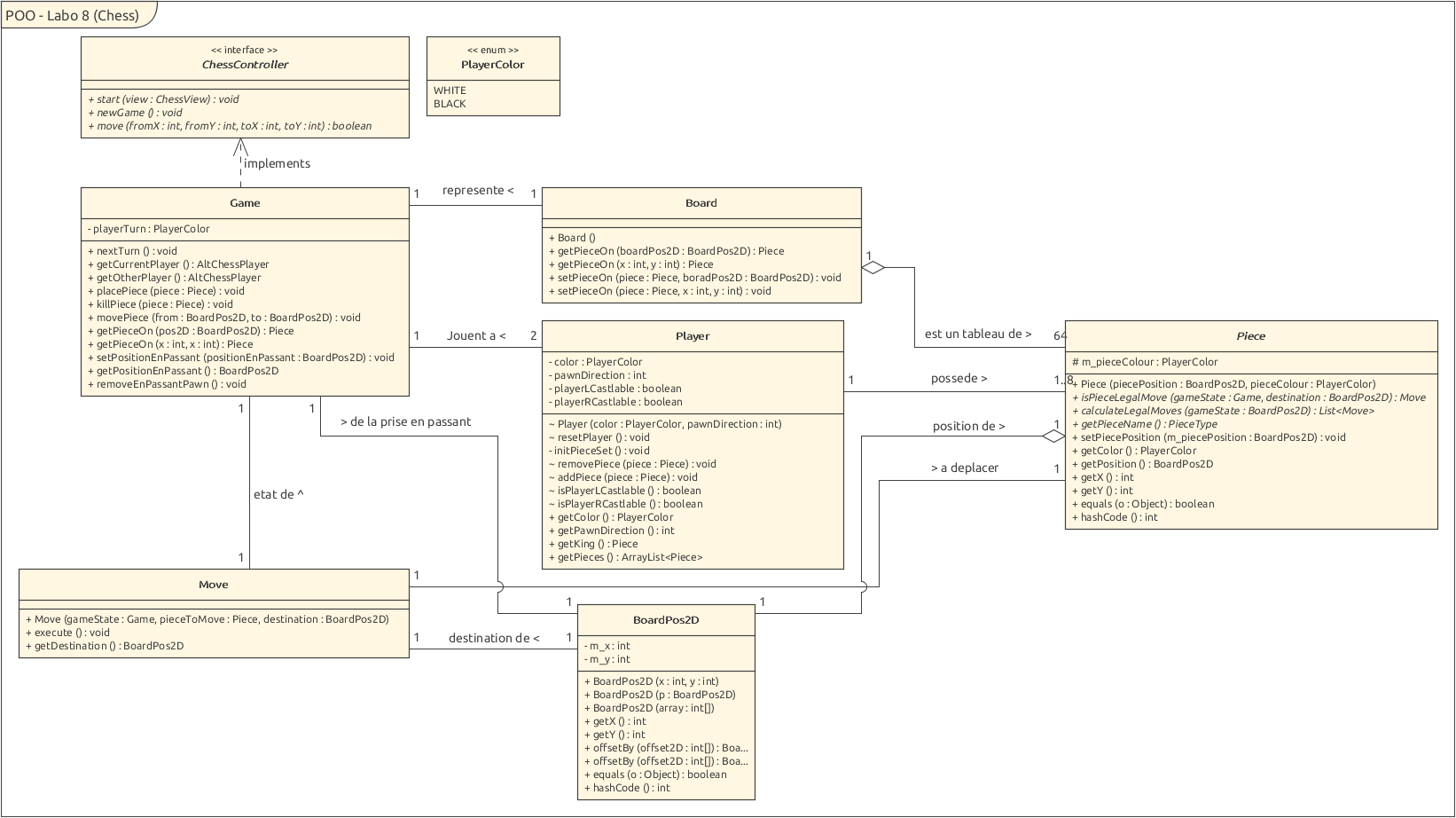
[4.5 Entrées constructeur par valeurs particulières 6](#_Toc93521319)

# Introduction

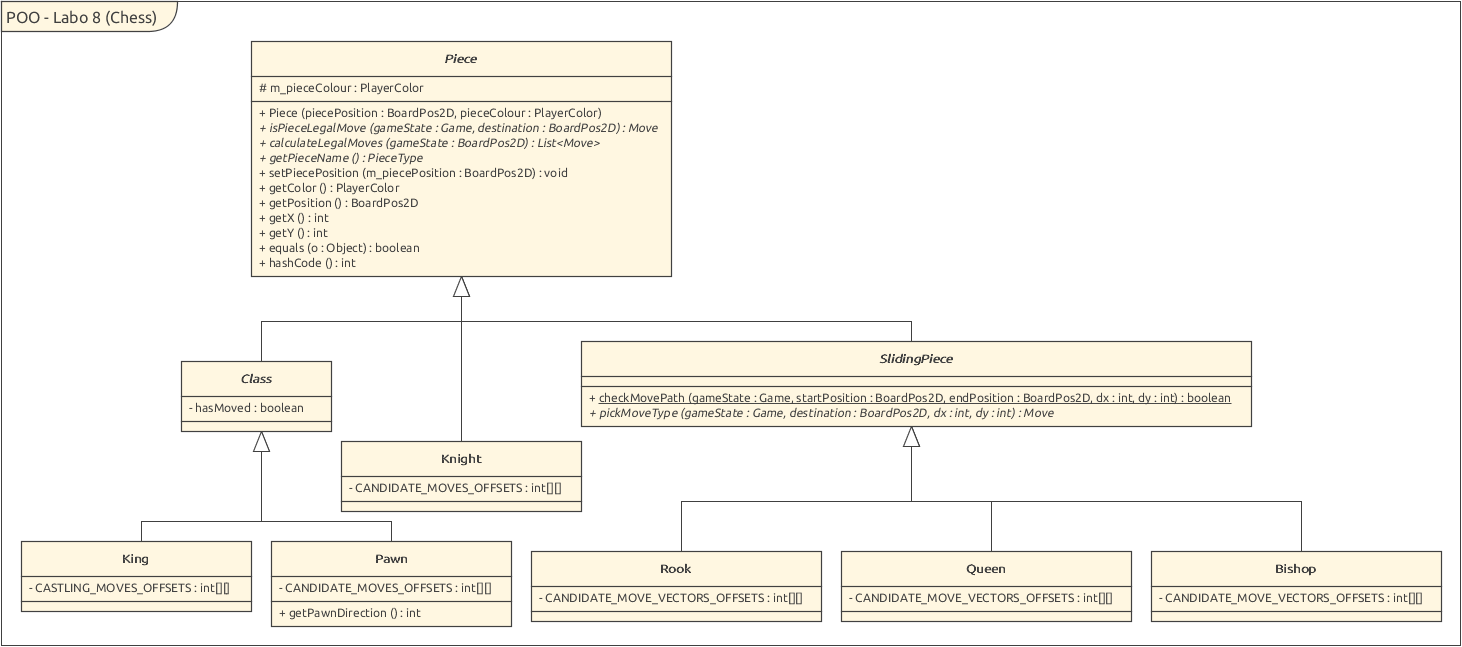
Dans ce laboratoire, nous devons développer la logique d'un jeu d'échec. L'interface nous est déjà fournie, elle consiste d'un échiquier et des ses pièces. Elle permet sélectionner deux cases avant d'envoyer cette information au contrôleur ; la première case choisie étant celle avec la pièce à déplacer et la seconde case la cible du déplacement. Nous devons implémenter les déplacement, la gestion des tours de joueur, la prise de pièces adverse, implémenter la prise en passant er le roque ainsi que la mise en échec.

# Diagramme de classe

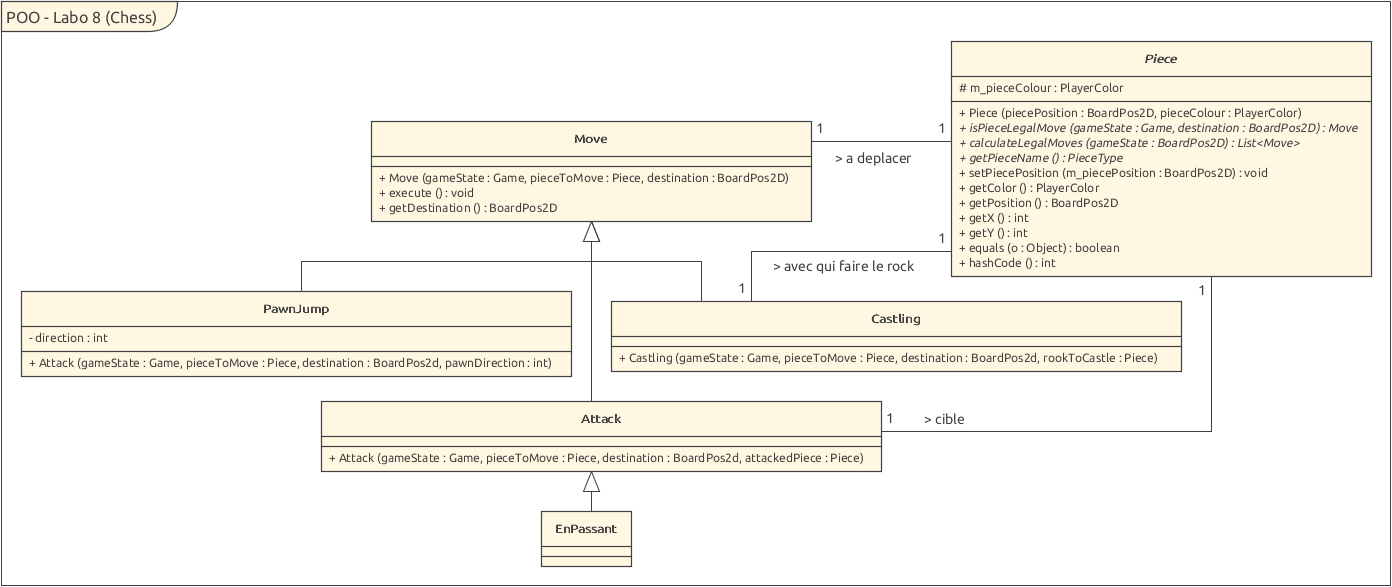
## Base



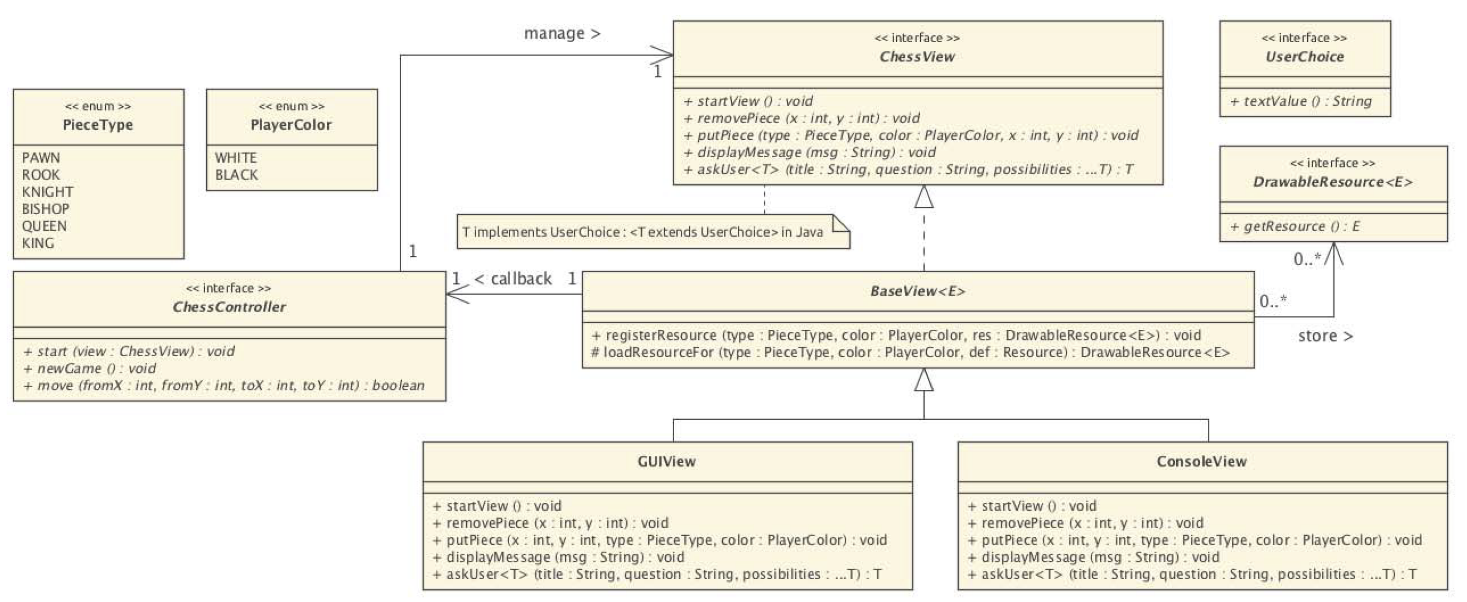
## Pieces



## Moves



## Diagramme de la donnée



# Choix de modélisation

## Description des classes

### Game

La classe *Game* implémente l'interface *ChessControler* fournie pour le laboratoire. Elle représente l'univers du jeu. C'est cette classe qui communique avec l'interface. Elle gère le démarrage du jeu, le déplacement effectif des pièces, permet la communication entre les différentes parties du jeu et qui indique si un mouvement est faisable (Mouvement sur soi-même, sur un allié, en dehors du terrain, le reste est géré par une méthode de *Piece* qui est appelée ensuite). Elle gère aussi la promotion de pion quand ils arrivent sur la ligne finale en face. Finalement c'est cette classe qui met en place la méthode qui vérifie si un roi est en échec.

### Board

Cette classe est simplement un tableau 8x8 de *Piece*. Elle met en place le plateau de jeu et permet d'interagir avec ses méthodes. C'est grâce à elle que l'ont peut trouver si une case à une pièces et quelle est cette pièce.

### Player

Cette classe permet de gérer les joueurs, elle liste les pièces de chaque joueur, indique si le grand ou petit roque sont faisables et permet d'initialiser le placement des pièces au début de partie.

### BoardPos2D

Cette classe permet d'obtenir une position sur l'échiquier. Toutes les pièces ont leur position enregistrée de cette manière. La classe met à dispositions des méthodes pour obtenir une position en offset d'une autre, si une position donnée se trouve dans les limites du plateau et overide les méthodes *equals* et *hashCode.*

### Piece

Cette classe abstraite est centrale au jeu, elle permet de représenter les pièces du jeu. Une pièce à une position et une couleur, elle a une méthode qui lui permette d'indiquer si un mouvement donné est légal (vérifie qu'il n'y ait pas de pièces sur le chemin si nécessaire et que c'est bien un mouvement que la pièce peut effectuer : par exemple le fou ne peut se déplacer qu'en diagonale), cette méthode renvoi le type de mouvement (*Move*) qui sera effectué : un simple déplacement, une prise ou d'autre encore.

#### MoveTrackedPiece

Cette classe abstraite permet de mettre en place les pièces dont il est important de savoir si elles ont déjà effectué : Le roi et les pions.

##### King

Cette pièce est le roi, il peut seulement se déplacer dans les 8 cases adjacentes ou effectuer un roque : s'il n'y a pas de pièces entre un roi et une des ses tours, que les deux n'ont jamais bougé, que le roi n'est pas en échec ou les deux cases en direction de la tour ne sont pas menacée par l'adversaire alors le roi se déplace de deux cases en direction de la tour et cette dernière se déplace de l'autre côté.

##### Pawn

Cette pièce est le pion, il ne peut qu'avancer d'une case, ne peut pas prendre en vertical mais seulement en diagonale d'une case en avant. Si c'est son premier mouvement il peut faire un "saut" et avancer de 2 cases (si elles sont toutes les 2 vides) plutôt. Si un tel mouvement est effectué un pion adverse peut effectuer ce qu'on appel une prise en passant : il prend comme si le pion précédant n'avait avancé que d'une case.

#### SlidingPiece

Cette classe abstraite permet de mettre en place les pièces qui se déplace et prendre sans limites de distance dans les directions "normale" (verticalement, horizontalement, ou en diagonales) et qui "glisse" : elles ne peuvent pas "sauter" des pièces dans leur déplacement. Cette classe met en place des méthodes qui permettent de vérifier qu'un mouvement demandé respecte bien cette règle de glissement et de donner le mouvement effectué.

##### Queen

Cette pièce est la dame, elle peut se déplacer dans toutes les directions : verticalement, horizontalement, ou en diagonales

##### Rook

Cette pièce est la tour, elle peut se déplacer verticalement et horizontalement.

##### Bishop

Cette pièce est le fou, elle peut se déplacer diagonalement.

#### Knight

Cette pièce est le cavalier, il est particulier car il ne peut se déplacer et prendre que en sautant d'une case à une autre en "L" deux cases dans une direction verticale ou horizontale et une case dans l'autre. Le cavalier n'est pas embêté par les pièces sur son chemin.

#### Move et ses filles

La classe *Move* (et ses filles : *Attack* et sa fille *EnPassant, Casteling,* et *PawnJump*) est la classe qui essaye de déplacer une pièce. Si cela devrait mettre le joueur actuel en échec ce mouvement est annulé. Les méthodes *doMove, tryMove, checkMove, et rollbackMove* font cela, dans *doMove* on essaye (*tryMove*) de se déplacer, puis on vérifie qu'on ne se met pas en échec (*checkMove*) et s'il faut on revient en arrière (*rollbackMove*)

# Tests