Laboratoire 8
Echecs

Part 1 : Schéma UML

Départements : TIC

Unité d'enseignement POO

Version : **1.0**

Groupe : **Groupe C**

Auteurs : **Anthony David**
Alexandre Ioro
Léo Zmoos

Professeur : **Marcel Graf**
Assistant : **René Rentsch**

Classe : **POO-C**
Salle de labo : **H03 / B23**

Date : **14 décembre 2022**

Table des matières

Table des matières *1*
1 Introduction *2*
2 Choix d'implémentation *2*
2.1 Pièce **2**
2.1.1 Roi, Tour et Pion (King, Rook and Pawn) 2
2.2 Board **3**
2.3 Controller **3**
2.4 Position **3**
2.5 Move **3**
2.6 ChessApp **3**
3 Schéma UML *4*
4 Conclusion *5*

1 Introduction

Le but de ce laboratoire est d'implémenter une jeu d'échec fonctionnel en Java.

La première étape de ce laboratoire est la création d'un diagramme de classe (schéma UML) sur lequel est basé le code de notre programme. Un rendu de ce schéma pour validation est demandé et c'est ce que contient ce présent rapport.

2 Choix d'implémentation

2.1 Pièce

Bon nombre d'éléments en commun entre toutes les pièces, on a décidé de faire une classe Piece dont hérite toutes les types de pièces de jeu.

La classe Piece est abstraite car elle devra obligatoirement être initialisée par une des types de pièce qui en hérite.

On donne la possibilité d'initialiser une pièce en lui indiquant sa position (second constructeur) afin de pouvoir faire des tests avec une situation de jeu spécifique.

Méthodes « KingBeCheck() » et « KingBeCheckmate() » qui nous permet de savoir si le roi est en échec et en échec-et-mat.

Méhtode « canMove() » qui est abstraite et redéfinie dans les pièces afin de savoir si en fonction de la position passée en paramètre, en déplaçant à cette dernière est autorisée pour la pièce.

« moveToSameColor() » permet de savoir si au mouvement on va aller sur une pièce à nous ou une pièce de l'adversaire.

2.1.1 Roi, Tour et Pion (King, Rook and Pawn)

En plus des différentes attributs hérités, on y ajoute un attribut pour savoir lors d'un mouvement de ce dernier est le premier afin de savoir si un roque est possible. Afin de factoriser, on fait hériter ces 3 pièces d'une classe « PieceExtension » qui est elle-même héritée de « Piece ».

De plus, il faut savoir, de combien de case le pion a avancé lors du dernier mouvement pour savoir si une prise en passant (takenInPassing()) est possible. Un attribut private dans la classe Pawn est donc ajouté.

2.2 Board

Représente l'échiquier de jeu.

2.3 Controller

Hérite de l'interface ChessControler fournit avec la consigne du laboratoire.
Il permet de gérer le déroulement du jeu d'échec.

2.4 Position

Représente la position sur le plateau d'une pièce.

2.5 Move

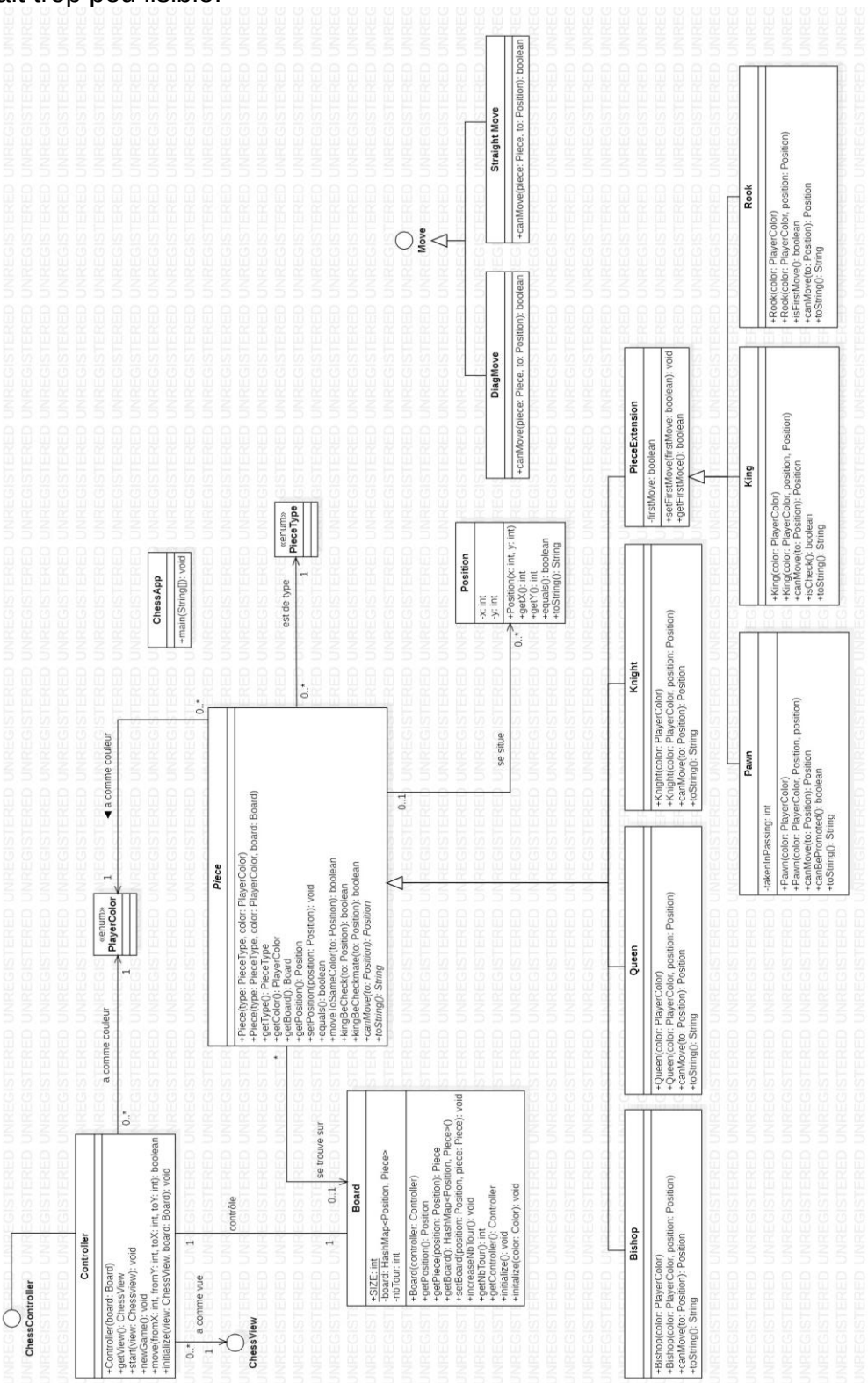
Permet de représenter directement les mouvements les plus présents dans un jeu d'échec qui sont les mouvements en diagonale et les mouvements en ligne droite.

2.6 ChessApp

Classe comprenant la fonction main() principale du programme.

3 Schéma UML

Note : On a été contraint de faire un export de l'image depuis StarUml car la capture d'écran était trop peu lisible.



4 Conclusion

Le schéma proposé ci-dessus représente une première idée de comment pourrais être représenté un jeu d'échec développé en programmation orienté objet. On peut quand même spécifier qu'aucun des membres du groupe ne connaissait les règles des échecs avant de commencer ce laboratoire et qu'il nous a été très difficile de s'en imprégner et de représenter cela en programmation en seulement quelques jour.

Il sera intéressant de voir à l'implémentation si le schéma imaginé ici correspond de manière optimale à un fonctionnement attendu.