Laboratoire 8 - Chess

Cseres Leonard, Aladin Iseni

5 janvier 2025

Table des matières

1 Introduction

L'objectif de ce laboratoire est de développer un jeu d'échecs fonctionnel respectant les règles de base. Le projet inclut les fonctionnalités suivantes: déplacements des pièces, coups spéciaux (roque, prise en passant, promotion des pions) et gestion des états de jeu (par exemple, échec). Les objectifs bonus consistent à implémenter la détection de l'échec et mat ainsi que du pat.

Pour simplifier le développement, les éléments suivants nous ont été fournis:

- **Enums:** `PieceType` pour les types de pièces et `PlayerColor` pour les couleurs des joueurs.
- **Interfaces:** `ChessController` et `ChessView` pour la gestion du jeu et de l'interface utilisateur.
- **Vues préconstruites:** Une vue graphique (`GUIView`) et une vue en mode texte (`ConsoleView`).

L'implémentation se concentre sur un nouveau package `engine` qui encapsule la logique du jeu tout en exploitant les interfaces fournies pour l'interaction.

---

2 Conception et Architecture

Notre approche respecte les principes de conception orientée objet, en garantissant l'encapsulation, la réutilisabilité et la modularité. Le package `engine` contient les classes et la logique pour la gestion du jeu, le suivi de l'état de l'échiquier et la génération des mouvements.

2.1 Composants Clés

- **`ChessEngine`:** Gère le déroulement du jeu et communique avec la vue.
- **`ChessBoard`:** Représente l'échiquier, suit les pièces et valide les états du jeu.
- **`ChessBoardView`:** Interface de lecture (view) de l'échiquier, qui ne permet pas de le modifier.
- **`ChessPiece`:** Classe abstraite définissant le comportement commun à toutes les pièces, étendue par des sous-classes spécifiques (par exemple, `Pawn`, `Rook`, etc.).
- **`MoveGenerator`:** Classe abstraite responsable de la génération des mouvements possibles pour les pièces.

2.2 Diagramme UML

Le diagramme UML fournit une vue d'ensemble de la structure et des relations du système. Les élements grisés représentent le code que nous avons utilisé et non pas implémenté.

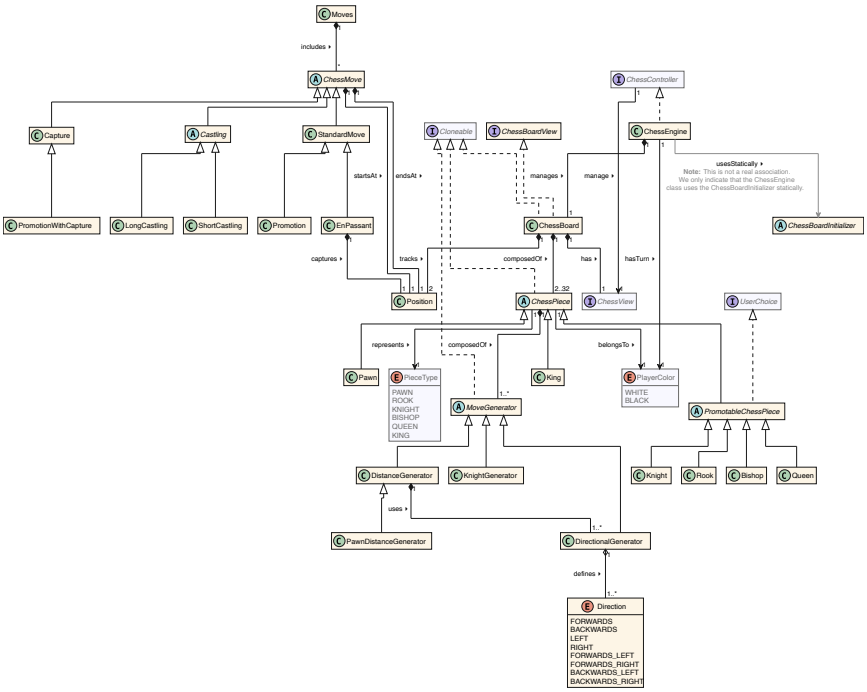


Figure 1: Schéma UML (Vue simplifiée)

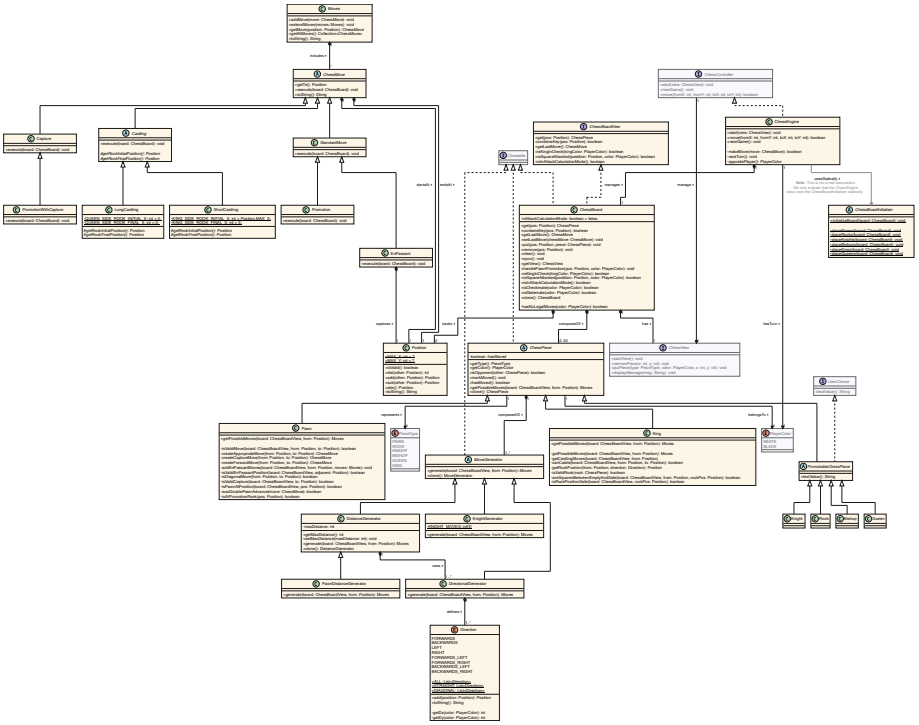


Figure 2: Schéma UML (Vue détaillée)

3 Caractéristiques Principales

3.1 Détection de Fin de Jeu

Le système vérifie:

- **Échec et mat:** Lorsque le roi est en échec et qu'aucun mouvement légal n'est possible.
- **Pat:** Lorsque aucun mouvement légal n'est possible, mais que le roi n'est pas en échec.

3.2 Règles Spéciales

- **Roque:** Vérifie que le roi et la tour concernés n'ont pas bougé, que le chemin est libre et que les cases traversées ne sont pas attaquées.
- **Prise en passant:** Implémente la capture d'un pion adjacent qui a avancé de deux cases à son premier mouvement.
- **Promotion de pions:** Demande au joueur de choisir un type de promotion (tour, cavalier, fou ou dame).

---

4 Tests Effectués

Tests effectuées	Résultat
Mettre le roi blanc en échec où le seule mouvement possible est l'attaque de la pièce blanche par une pièce noire	V
En Passant est uniquement pratiquable lorsque le pion adverse avance de deux cases	V
En passant est praticable uniquement au tour suivant et pas 2 tours après	V
Le roque est uniquement praticable si le roi et la tour en question n'ont pas bougé	V
Le roque est pratiquable uniquement si les cases sur lesquelles passe le roi ne sont pas attaquées	V
Les pions peuvent avancer de deux cases uniquement lors de leur premier déplacement	V
Chaque pièce avance dans la bonne direction	V
Uniquement les chevaux peuvent sauter des pièces	V
Les pièces ne peuvent pas découvrir un échec	V
Le roi ne peut pas se mettre en échec	V
Lorsque le roi est en échec, uniquement les mouvements de défenses sont pratiquables	V
Une pièce ne peut que capturer les pièces d'une autre couleur	V
Un pion peut être promu en reine, fou, chevalier ou tour	V
Un message Check s'affiche lorsque le roi est en échec et Checkmate lorsque quelqu'un a gagné	X

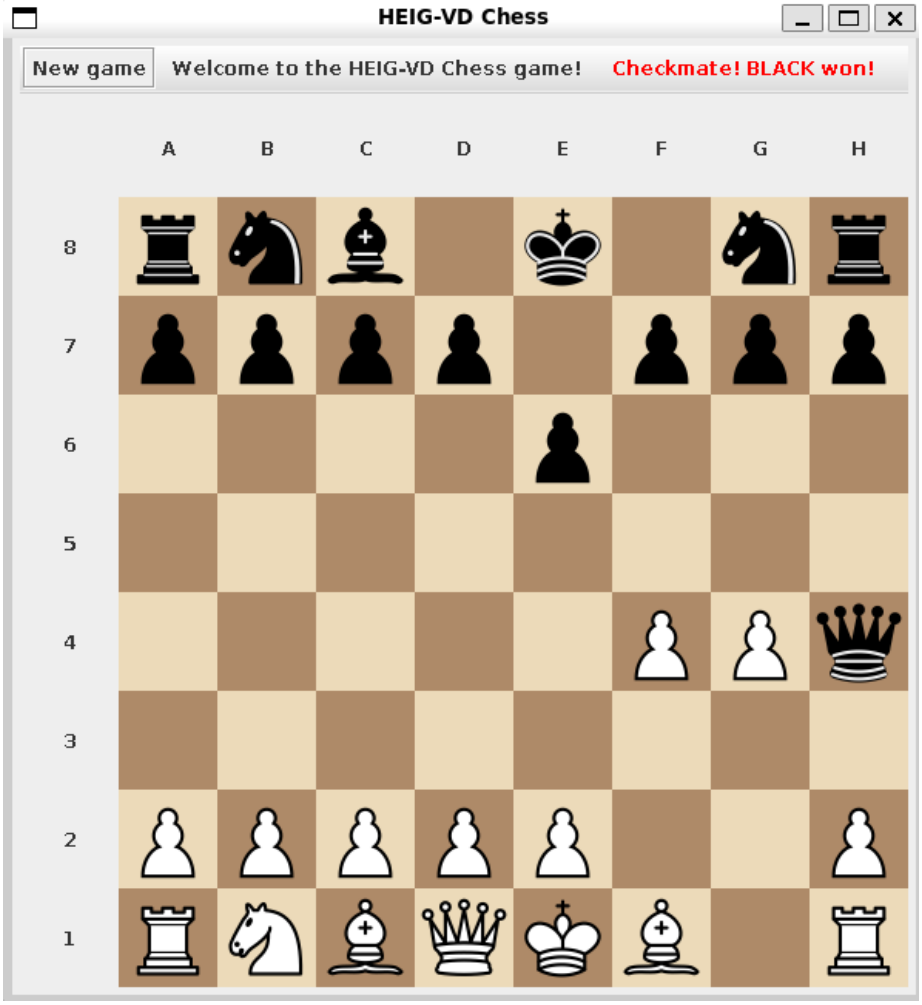
4.1 Défense par l'attaque

Les images suivantes montrent que le joueur blanc est obligé d'attaque le fou en H4 avec le cavalier en F3 afin de défendre son roi.


HEIG-VD Chess
New game
Welcome to the HEIG-VD Chess game!
A
B
C
D
E
F
G
H
8
7
6
5
4
3
2
1

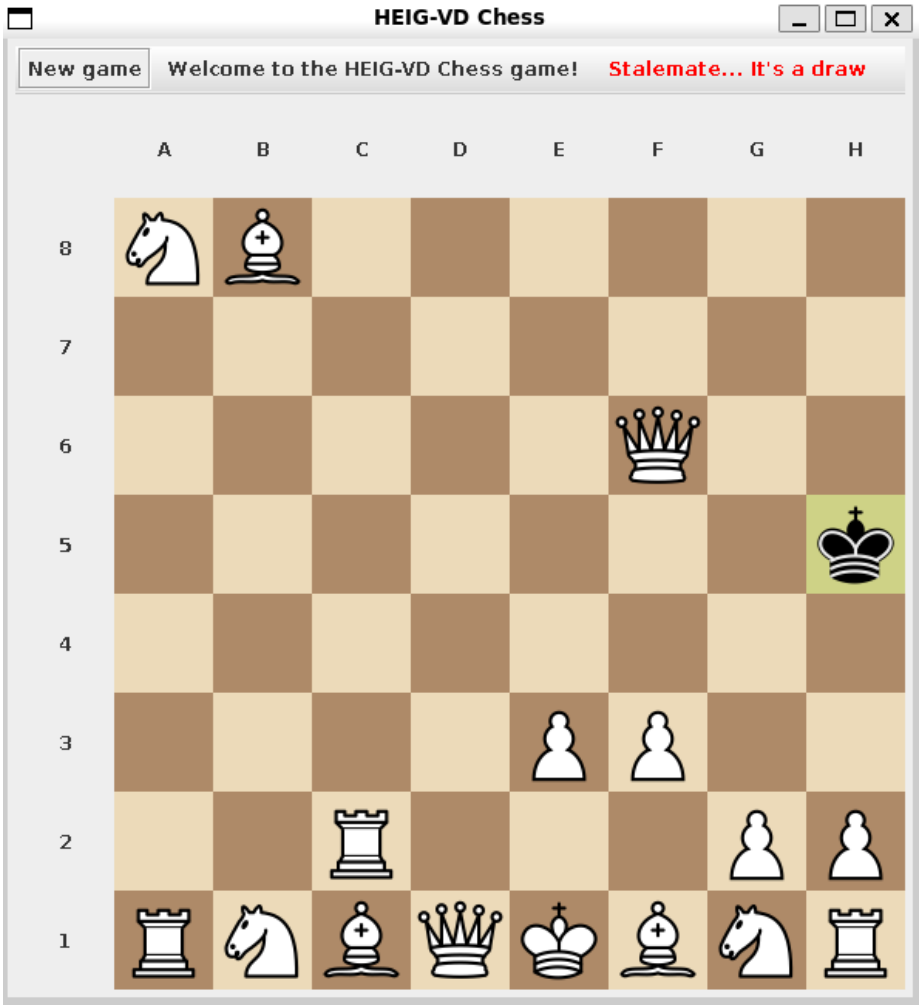
4.2 Checkmate

Cette image montre que notre jeu est capable de détecter un échec et mat.



4.3 Stalemate

Cette image montre que notre jeu est capable de détecter un pat.



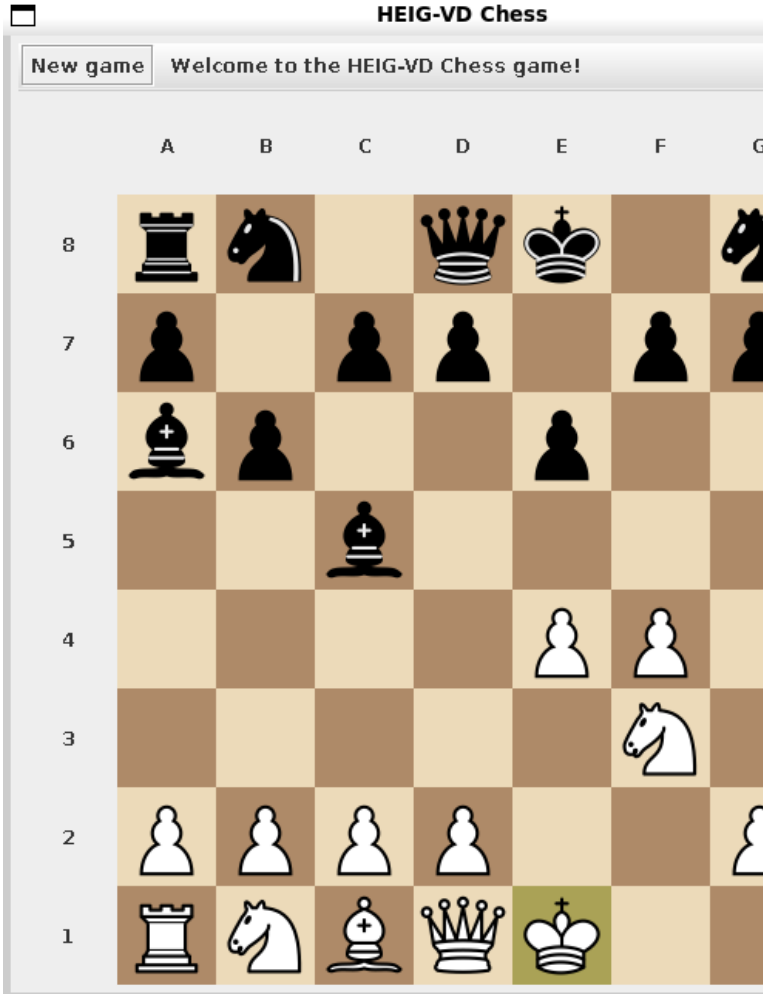
4.4 En Passant

Sur les deux images ci-dessous, nous pouvons observer que notre jeu propose l'attaque En Passant et permet de l'exécuter.


HEIG-VD Chess
New game
Welcome to the HEIG-VD Chess game!
A
B
C
D
E
F
G
H
8
7
6
5
4
3
2
1

4.5 Castling

Les trois images suivantes montrent qu'il n'est pas possible d'effectuer un castling si les cases du passage du roi sont attaquées.




HEIG-VD Chess
New game
Welcome to the HEIG-VD Chess game!
A
B
C
D
E
F
G
H
8
7
6
5
4
3
2
1

4.6 Promotion

Ci-dessous, nous observons qu'il est possible de promouvoir un pion en reine, tour, fou ou cavalier à l'aide d'une fenêtre de sélection.


HEIG-VD Chess
New game
Welcome to the HEIG-VD Chess game!
A
B
C
D
E
F
G
H
8
7
6
5
4
3
2
1

5 Extensions

L'implémentation étend les fonctionnalités au-delà des exigences de base:

- **Logique Réutilisable:** La génération des mouvements est abstraite dans des classes réutilisables, simplifiant les extensions et les futures modifications.
- **Gestion des États de Jeu:** La détection de l'échec et mat et du pat améliore l'expérience utilisateur et respecte les règles réelles des échecs.

5.1 Génération des Mouvements

La hiérarchie `MoveGenerator` encapsule la logique de génération des mouvements :

- **`DirectionalGenerator`:** Pour les mouvements linéaires (par exemple, tour, fou).
- **`KnightGenerator`:** Pour les mouvements en L propres aux cavaliers.
- **`DistanceGenerator`:** Gère les mouvements avec des portées variables, comme les pions.

5.2 Gestion des États de Jeu

TODO

6 Conclusion

Ce projet a renforcé les principes de programmation orientée objet tout en abordant des règles et interactions complexes. Les défis ont inclus:

- Garantir l'encapsulation tout en gérant les comportements variés des pièces.
- Traiter les cas limites dans les coups spéciaux et les conditions de fin de jeu.

Améliorations futures possibles:

- Ajouter une IA pour un mode solo.
- Proposer des suggestions de mouvements ou mettre en évidence les mouvements valides pour améliorer l'expérience utilisateur.

A Annexes

A.1 Listing Java

c.f. page suivante.