

Laboratoire 8 - Chess

Cseres Leonard, Aladin Iseni

3 janvier 2025

Table des matières

1	Introduction	2
2	Conception et Architecture	2
2.1	Composants Clés	2
2.2	Diagramme UML	2
3	Caractéristiques Principales	4
3.1	Détection de Fin de Jeu	4
3.2	Règles Spéciales	4
4	Tests Effectués	4
5	Extensions	4
5.1	Génération des Mouvements	4
5.2	Gestion des États de Jeu	4
6	Conclusion	5
A	Annexes	6
A.1	Listing Java	6

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.



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

TODO

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.