POO - Laboratoire 08

Jeu d'échecs

Maxime Lestiboudois & Parisod Nathan

09.01.2025

Contents

Introduction

Ce laboratoire consiste à implémenter un jeu d'échecs fonctionnel avec une interface graphique (GUI) et un mode console. Les règles à respecter incluent les mouvements de toutes les pièces, le roque, la prise en passant, et la promotion des pions. L'objectif est de créer une application robuste tout en respectant les principes de la programmation orientée objet (POO).

Cahier des charges

Objectif du projet

Le but de ce projet est de développer un jeu d'échecs complet en suivant les règles officielles. L'application doit inclure une gestion complète des pièces et des mouvements, une interface utilisateur graphique ainsi qu'un mode console.

Spécifications fonctionnelles

- Prise en charge des mouvements des pièces : rois, dames, tours, fous, cavaliers et pions.
- Gestion du roque (petit et grand).
- Implémentation de la prise en passant.
- Gestion de la promotion des pions avec choix de la pièce de promotion.
- Si le roi est mis en échec, il est obligé de se "protéger".

Contraintes techniques

- Utilisation des classes et interfaces fournies dans le dossier de départ.
- Respect des principes de POO, en particulier l'encapsulation et la modularité.
- Implémentation des règles du jeu sans recours à des structures conditionnelles basées sur les types de pièces.

,

Modélisation des pièces

Chaque pièce du jeu d'échecs est modélisée par une classe spécifique qui hérite de la classe abstraite `Piece`. Voici les principales classes et leurs rôles :

- `King` : Représente un roi et gère les mouvements du roi, y compris le roque.
- `Rook` : Représente une tour et implémente le mouvement des tours et participe au roque.

- `Bishop` : Représente un fou et modélise le mouvement des fous.
- `Knight` : Représente un cavalier gère le mouvement en forme de L des cavaliers.
- `Pawn` : Représente un pion et gère les mouvements spéciaux des pions, y compris la prise en passant et la promotion.
- `Queen` : Représente une reine et modélise les mouvements de la dame.

Gestion des spécificités des mouvements

Des classes internes implémentent les différentes stratégies de mouvements propres à chaque pièce. La validation des mouvements utilise des instances de `PathValidator` pour vérifier que les chemins empruntés sont libres.

Gestion des mouvements spéciaux

- **Roque** : Le mouvement du roi de deux cases enclenche le roque. La validation du roque vérifie que ni le roi ni la tour n'ont déjà bougé et que les cases intermédiaires ne sont pas sous attaque.
- **Prise en passant** : Ce mouvement est validé en vérifiant que le dernier coup joué était un double pas d'un pion adverse.
- **Promotion** : Lorsqu'un pion atteint la dernière rangée, le joueur choisit une pièce pour le promouvoir.

Tests effectués

Des tests ont été réalisés pour vérifier la validité des mouvements et des règles spéciales. Voici quelques scénarios de test :

- Déplacements valides et invalides pour chaque type de pièce.
- Exécution du roque dans les conditions appropriées.
- Prise en passant uniquement lorsqu'elle est permise.
- Promotion de pion avec choix de la pièce.

Choix de conception

Gestion des état des pièces

La classe `SpecialFirstMovePiece` gère l'état `hasMoved` pour les pièces concernées par le roque ou les mouvements spéciaux.

Stratégies de mouvement

Les mouvements sont implémentés via des stratégies internes dans les classes des pièces, permettant une extension future facilitée.

Conclusion

Ce laboratoire a permis de mettre en pratique les principes de la programmation orientée objet tout en réalisant un projet concret et amusant. Le respect de l'architecture MVC et l'utilisation de classes abstraites et d'interfaces ont permis de structurer le code de manière claire et modulaire.