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 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.

Choix de conception

Gestion des états des pièces

Les classes `SpecialFirstMovePiece` gèrent l'attribut `hasMoved` pour les pièces concernées par le roque ou les mouvements spéciaux. Cette approche permet de suivre l'état de chaque pièce et d'appliquer les règles du jeu de manière dynamique.

Utilisation des stratégies de mouvement

Les mouvements sont implémentés via des classes internes qui définissent des stratégies spécifiques pour chaque type de pièce. Cette approche permet une extension facile pour ajouter de nouveaux types de mouvements ou des règles spéciales.

Validation des chemins

Pour garantir que les pièces ne traversent pas d'autres pièces lorsqu'elles se déplacent, nous avons utilisé un validateur de chemin `PathValidator`. Chaque classe de pièce utilise une instance de ce validateur pour vérifier que le chemin emprunté est libre avant d'exécuter le mouvement.

Gestion des mouvements spéciaux

- **Roque** : La validation du roque vérifie que ni le roi ni la tour n'ont déjà bougé, que les cases intermédiaires sont libres et qu'aucune case ne se trouve sous attaque.

- **Prise en passant** : Cette fonctionnalité est implémentée en vérifiant que le dernier coup joué était un double pas d'un pion adverse. La prise est effectuée sur la case que le pion aurait occupé s'il avait avancé d'une seule case.
- **Promotion** : Lorsqu'un pion atteint la dernière rangée, le joueur est invité à choisir une pièce de promotion. Cette fonctionnalité est gérée via une interaction avec la vue.

Architecture MVC

L'application suit une architecture Modèle-Vue-Contrôleur (MVC). Le contrôleur gère la logique de jeu, le modèle représente les éléments du jeu (pièces, plateau) et la vue présente les données à l'utilisateur. Cette architecture permet une séparation claire des responsabilités et facilite la maintenance et les modifications futures.

Encapsulation des données

Chaque pièce encapsule ses propriétés (couleur, type, position) et gère ses propres règles de mouvement. Cela permet d'éviter les tests conditionnels complexes et garantit que chaque classe respecte le principe de responsabilité unique.

Gestion des exceptions

Nous avons implémenté une gestion robuste des erreurs en utilisant des exceptions pour traiter les mouvements invalides. Cela permet d'assurer que le programme reste stable même en cas d'entrées incorrectes de la part de l'utilisateur.

Testabilité

La séparation des stratégies de mouvement et des validations permet d'écrire des tests unitaires ciblés pour chaque composant, garantissant ainsi la fiabilité du code.

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.

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.