KAYA RHABBY Pascia Hershe
DIALLO Abdoul Aziz

14 mai 2023

Développement d’un jeu d’échec en Java

I. Introduction

A. Présentation du jeu d'échecs

Le jeu d'échecs est l'un des jeux de société les plus populaires dans le monde entier. Il se joue entre deux joueurs, chacun contrôlant une armée de 16 pièces sur un plateau de 64 cases. Le but du jeu est de mettre en échec le roi de l'adversaire, ce qui signifie que le roi est menacé d'être capturé. Si le roi est capturé, la partie est terminée.

B. Objectif du rapport

L'objectif de ce rapport est de présenter notre approche pour réaliser un jeu d'échecs en Java, en décrivant les choix de conception et d'implémentation que nous avons faits, ainsi que les fonctionnalités et l'interface utilisateur de notre jeu.

C. Présentation de l'approche

Notre approche consiste à utiliser une architecture orientée objet pour modéliser le jeu d'échecs en Java. Nous avons conçu des classes pour représenter les différentes pièces et le plateau, ainsi que pour gérer les interactions entre les joueurs. Nous avons également conçu une interface utilisateur simple et intuitive pour permettre aux utilisateurs de jouer facilement.

II. Conception

A. Choix de l'architecture

Nous avons choisi une architecture orientée objet pour notre jeu d'échecs, car cela nous permet de modéliser le jeu de manière simple et efficace. Nous avons créé des classes pour représenter les différentes pièces et le plateau, et avons utilisé l'héritage pour gérer les différentes règles de déplacement et d'interaction entre les pièces. Nous avons également utilisé la composition afin de gérer plus facilement l’utilisation de la grille de jeu ainsi que l'affichage de cette dernière au niveau de l’interface graphique afin de faciliter les interactions hommes-machines .

B. Modélisation des données

Nous avons modélisé les données nécessaires pour le jeu d'échecs en Java, telles que les pièces,le plateau et les règles de fin de jeu. Nous avons utilisé des classes pour représenter chacune de ces données, en définissant les attributs et les méthodes nécessaires pour les manipuler.

C. Conception des classes

Nous avons conçu des classes pour représenter chaque pièce dans le jeu d'échecs, telles que le roi, la reine, le fou, le cavalier, la tour et le pion. Les différentes classes des pièces sont en anglais telles que King pour le roi , Queen pour la reine , Bishop pour le fou, Rook pour la tour , Knight pour le cavalier et enfin Pawn pour le pion. Nous avons également conçu des classes pour représenter le plateau de jeu. Nous avons ainsi la classe ChessBoard qui représente le plateau du jeu , la classe ChessBoardView qui contrôle la partie interface graphique du jeu et pour finir les classes GameRules et Game qui respectivement contrôlent la fin de la partie et le lancement du jeu .

D. Diagrammes de classes

Nous avons utilisé des diagrammes de classes pour visualiser la structure de notre code. Les diagrammes de classes nous ont permis de voir comment les différentes classes sont interconnectées, et comment les méthodes et les attributs sont partagés entre les classes (voir dans la section annexe).

III. Implémentation

A. Environnement de développement

Nous avons utilisé l'environnement de développement IntelliJ IDEA pour développer notre jeu d'échecs en Java.

B. Choix des bibliothèques

Nous avons utilisé la bibliothèque Swing de Java pour développer l'interface utilisateur.

C. Développement des classes

Nous avons développé les différentes classes pour notre jeu d'échecs en utilisant l'architecture orientée objet que nous avons choisie. Nous avons commencé par créer des classes pour représenter les pièces du jeu, en utilisant l'héritage pour gérer les différents déplacements et interactions entre les pièces.

Ensuite, nous avons développé une classe pour représenter le plateau de jeu, que nous avons appelé ChessBoard, qui est une grille de pièces de 8x8 cases.

Enfin, nous avons développé une classe permettant de contrôler les règles de fin de jeu, GameRules. Cette classe ne possède que des méthodes statiques utilisables directement en passant par la classe elle même et permet de tester échecs et mat et est pat .

D. Développement de l'interface utilisateur

Nous avons développé une interface utilisateur simple et intuitive pour notre jeu d'échecs. Nous avons utilisé la bibliothèque Swing de Java pour créer une fenêtre de jeu.

Nous avons également ajouté des fonctionnalités telles que la mise en évidence des cases disponibles pour les déplacements des pièces ainsi que la possibilité de choisir une promotion de pion.

IV. Fonctionnalités

A. Déplacement des pièces

Notre jeu d'échecs permet aux joueurs de déplacer les pièces en utilisant les règles standard du jeu d'échecs. Les joueurs peuvent sélectionner une pièce en cliquant dessus, et les cases disponibles pour le déplacement sont mises en évidence. Les joueurs peuvent alors cliquer sur la case de destination pour déplacer la pièce.

B. Promotion de pion

Si un pion atteint la dernière rangée du plateau de jeu, il peut être promu en une autre pièce. Notre jeu d'échecs permet aux joueurs de choisir la pièce de promotion en cliquant sur la pièce de son choix en laquelle il veut que le promotion s’effectue.

C. Roque

Dans notre jeu d'échecs , nous avons implémenté la fonctionnalité de roque, qui est une manœuvre spéciale permettant au roi et à l'une des tours de se déplacer simultanément. Nous avons implémenté les règles standards du roque, à savoir :

- Le roi ne doit pas être en échec avant le roque.
- Le roi ne doit pas traverser une case contrôlée par une pièce adverse lors du roque.
- Le roi ne doit pas se déplacer sur une case qui est attaquée par une pièce adverse lors du roque.
- La tour impliquée dans le roque ne doit pas avoir été déplacée précédemment.
- Il ne doit y avoir aucune pièce entre le roi et la tour impliquée dans le roque.

Lorsqu'un joueur souhaite effectuer un roque, il doit sélectionner le roi et cliquer sur la case de destination appropriée.

Nous avons également ajouté une fonctionnalité visuelle pour le roque, en faisant déplacer simultanément le roi et la tour impliquée vers leurs nouvelles positions. Cette animation aide à clarifier le déplacement pour les joueurs et ajoute une touche d'interactivité à l'expérience de jeu.

D. En passant

Dans notre jeu d'échecs , nous avons également implémenté la règle de l'en passant, qui est une règle spéciale qui permet à un pion de prendre un pion adverse qui vient d'avancer de deux cases à partir de sa position initiale.

Lorsqu'un pion avance de deux cases à partir de sa position initiale, un joueur adverse peut effectuer une capture en passant en déplaçant un de ses propres pions qui se trouve sur une colonne adjacente à la case où le pion a atterri. Le pion adverse est alors capturé et retiré du plateau.

Nous avons implémenté cette règle en ajoutant une vérification spéciale lorsqu'un pion avance deux cases à partir de sa position initiale, un boolean hasJustMoveDouble intégré dans la classe pion est mise à true . Si un pion adverse se trouve sur une colonne adjacente à la case où le pion est arrivé, le joueur peut effectuer une capture en passant en sélectionnant le pion approprié et en cliquant sur la case de destination. Si le joueur adverse ne joue pas le coup en passant , ce dernier n’est plus valide au prochain coups qu’il va jouer, car la variable boolean repasse à false .

Nous avons également ajouté une fonctionnalité visuelle pour l'en passant, en faisant disparaître le pion capturé de manière animée pour renforcer l'immersion des joueurs dans le jeu.

V. Variant (Echec capablanca)

A. Présentation

Les échecs Capablanca sont une variante du jeu d'échecs, se jouant sur un échiquier 10×8. Le jeu est nommé d'après son inventeur, l'ancien champion du monde d'échecs, José Raúl Capablanca (1888-1942). En plus des pièces standard, chaque joueur dispose de deux pièces féeriques, à savoir :

- une Impératrice (ou Chancelier), qui combine les mouvements du cavalier et de la tour;
- une Princesse (ou Centaure), qui combine les mouvements du cavalier et du fou.

Ces nouvelles pièces complexifient un peu le jeu. Ce faisant dans le souci de respecter la contrainte d’implanter au minimum quatres(4) pièces féeriques , ce dernier se voit donc légèrement modifié . En effet, dans cette version variant d'échecs capablanca, deux pièces

nouvelles peuvent être introduites dans le jeu lors de la promotion du pion . Un pion peut être promu en reine, fou,cavalier, tour, impératrice,princesse, noctambule et enfin sauterelle. Également , vu la quantité des pièces de haut rang qui peuvent se trouver simultanément sur le plateau de jeu , des coups spéciaux comme le roque et le en passant ne sont donc pas possible.

VII. Répartition du travail entre étudiant

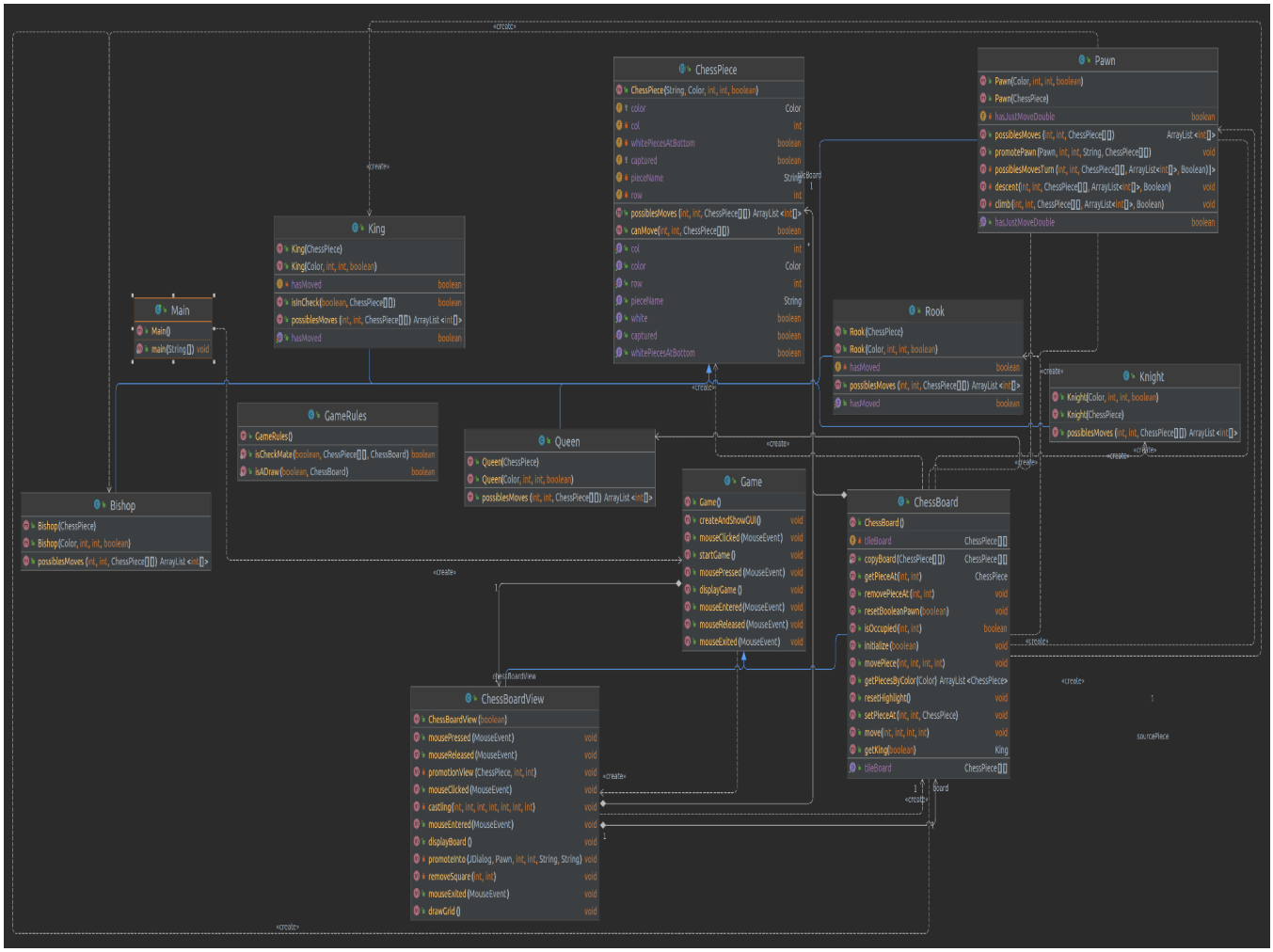
Durant tout le déroulement du projet, chaque classe , chaque méthode implémentée et chaque attribut ajouté ou supprimé était sous l'approbation des deux parties. Toute action devrait être justifiée par les deux parties. Ce faisant , le travail n'a pas été partagé de façon à ce que chacun puisse faire une partie spécifique , au contraire l'implémentation a été faite à deux à tout moment. Cependant , chacun pouvait avancer de son côté comme il le pouvait et donc avancer autant qu'il veut à condition que tout ce qui a été implémenté ait été expliqué à l'autre partie , dans le but d'améliorer le code si besoin il y a.

VIII. Conclusion

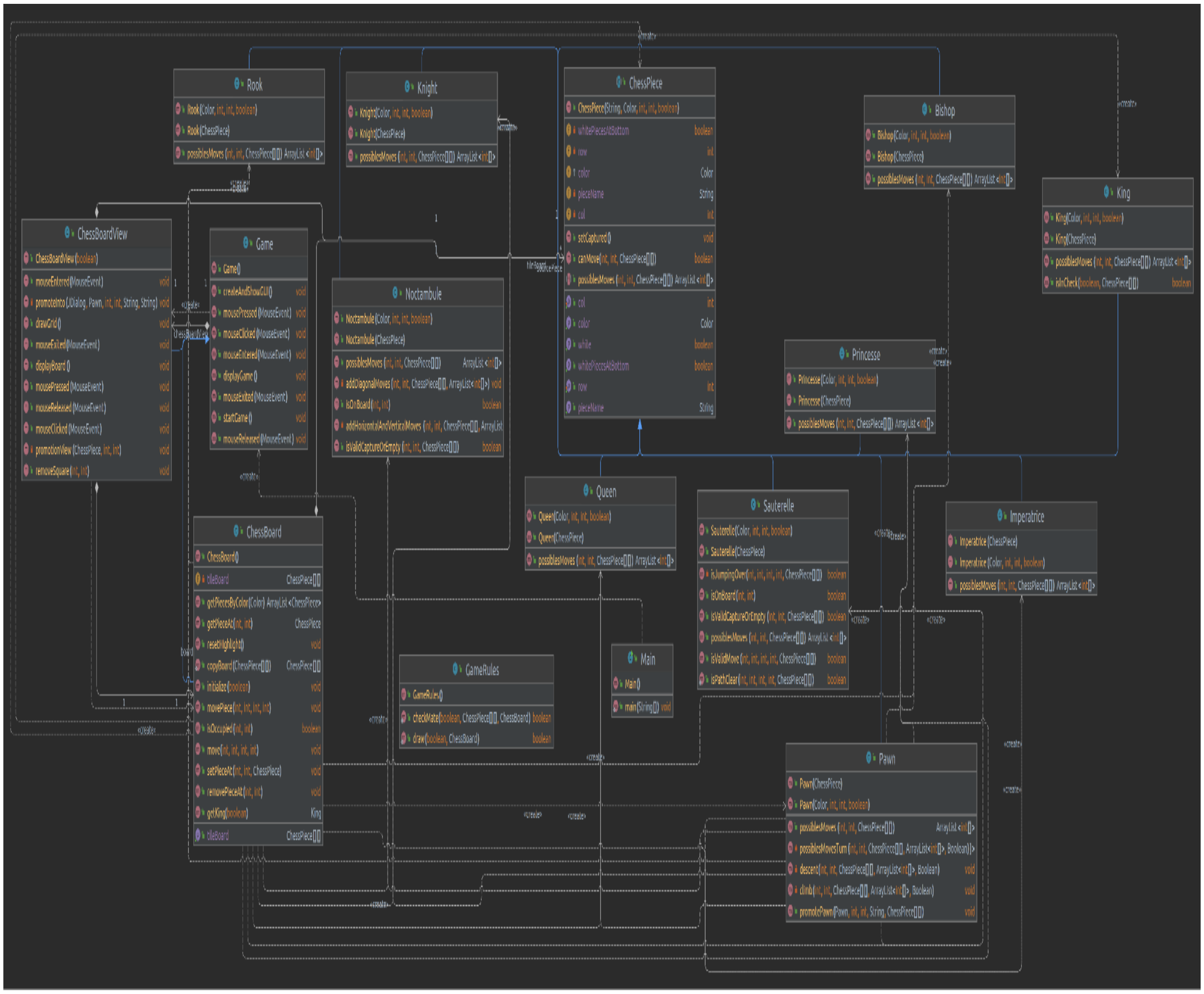
Dans ce rapport, nous avons présenté notre approche pour réaliser un jeu d'échecs en Java en utilisant une architecture orientée objet. Nous avons expliqué les choix de conception et d'implémentation que nous avons fait, ainsi que les fonctionnalités et l'interface utilisateur de notre jeu.

ANNEXE :

- Diagramme de classe classic jeu d'échec



- Diagramme de classe variant jeu d'échec


Rook
Rook(Color, int, int, boolean)
Rook(ChessPiece)
possiblesMoves (int, int, ChessPiece[][]) ArrayList <int[]>
Knight
Knight(Color, int, int, boolean)
Knight(ChessPiece)
possiblesMoves (int, int, ChessPiece[][]) ArrayList <int[]>
ChessPiece
ChessPiece(String, Color, int, int, boolean)
whitePiecesAtBottom boolean
row int
color Color
pieceName String
col int
setCaptured() void
canMove(int, int, ChessPiece[][]) boolean
possiblesMoves (int, int, ChessPiece[][]) ArrayList <int[]>
col int
color Color
white boolean
whitePiecesAtBottom boolean
row int
pieceName String
Bishop
Bishop(Color, int, int, boolean)
Bishop(ChessPiece)
possiblesMoves (int, int, ChessPiece[][]) ArrayList <int[]>
King
King(Color, int, int, boolean)
King(ChessPiece)
possiblesMoves (int, int, ChessPiece[][]) ArrayList <int[]>
isInCheck(boolean, ChessPiece[][]) boolean
ChessBoardView
ChessBoardView(boolean)
mouseEntered(MouseEvent) void
promoteInto (JDialog, Pawn, int, int, String, String) void
drawGrid() void
mouseExited(MouseEvent) void
displayBoard () void
mousePressed (MouseEvent) void
mouseReleased (MouseEvent) void
mouseClicked (MouseEvent) void
promotionView (ChessPiece, int, int) void
removeSquare(int, int) void
Game
Game()
createAndShowGUI() void
mousePressed (MouseEvent) void
mouseClicked (MouseEvent) void
mouseEntered(MouseEvent) void
displayGame () void
mouseExited (MouseEvent) void
startGame () void
mouseReleased (MouseEvent) void
Noctambule
Noctambule(Color, int, int, boolean)
Noctambule(ChessPiece)
possiblesMoves (int, int, ChessPiece[][]) ArrayList <int[]>
addDiagonalMoves (int, int, ChessPiece[][], ArrayList<int[]>) void
isOnBoard(int, int) boolean
addHorizontalAndVerticalMoves (int, int, ChessPiece[][], ArrayList
isValidCaptureOrEmpty (int, int, ChessPiece[][]) boolean
Princesse
Princesse(Color, int, int, boolean)
Princesse(ChessPiece)
possiblesMoves (int, int, ChessPiece[][]) ArrayList <int[]>
Queen
Queen(Color, int, int, boolean)
Queen(ChessPiece)
possiblesMoves (int, int, ChessPiece[][]) ArrayList <int[]>
Sauterelle
Sauterelle(Color, int, int, boolean)
Sauterelle(ChessPiece)
isJumpingOver(int, int, int, int, ChessPiece[][]) boolean
isOnBoard(int, int) boolean
isValidCaptureOrEmpty (int, int, ChessPiece[][]) boolean
possiblesMoves (int, int, ChessPiece[][]) ArrayList <int[]>
isValidMove (int, int, int, int, ChessPiece[][]) boolean
isPathClear(int, int, int, int, ChessPiece[][]) boolean
Imperatrice
Imperatrice (ChessPiece)
Imperatrice (Color, int, int, boolean)
possiblesMoves (int, int, ChessPiece[][]) ArrayList <int[]>
ChessBoard
ChessBoard()
tileBoard ChessPiece[][]
getPiecesByColor(Color) ArrayList <ChessPiece>
getPieceAt(int, int) ChessPiece
resetHighlight() void
copyBoard(ChessPiece[][]) ChessPiece[][]
initialize(boolean) void
movePiece(int, int, int, int) void
isOccupied(int, int) boolean
move(int, int, int, int) void
setPieceAt(int, int, ChessPiece) void
removePieceAt(int, int) void
getKing(boolean) King
tileBoard ChessPiece[][]
GameRules
GameRules()
checkMate(boolean, ChessPiece[][], ChessBoard) boolean
draw(boolean, ChessBoard) boolean
Main
Main()
main(String[]) void
Pawn
Pawn(ChessPiece)
Pawn(Color, int, int, boolean)
possiblesMoves (int, int, ChessPiece[][]) ArrayList <int[]>
possiblesMovesTurn (int, int, ChessPiece[][], ArrayList<int[]>, Boolean)[]>
descent(int, int, ChessPiece[][], ArrayList<int[]>, Boolean) void
climb(int, int, ChessPiece[][], ArrayList<int[]>, Boolean) void
promotePawn(Pawn, int, int, String, ChessPiece[][]) void