

Rapport de laboratoire

Programmation Orientée Objet (POO)

Rayane ANNEN

Joris DÉCOPPET

Hugo DUCOMMUN

Alexis MARTINS

12 janvier 2023

Ce rapport présente le laboratoire 8 de POO (Programmation Orientée Objet) de la HEIG-VD qui concernait l'implémentation d'un jeu d'échecs. Principalement, seront présentés la modélisation UML, la conception en Java, nos hypothèses de travail ainsi que les tests effectués.

```

classDiagram
    class Chessboard {
        <<abstract>>
        +int boardSize()
        +int boardWidth()
        +int boardHeight()
        +int boardDepth()
        +int boardLength()
        +int boardArea()
        +int boardVolume()
        +int boardPerimeter()
        +int boardSurface()
        +int boardMass()
        +int boardWeight()
        +int boardDensity()
        +int boardTemperature()
        +int boardHumidity()
        +int boardPressure()
        +int boardAcceleration()
        +int boardVelocity()
        +int boardPosition()
        +int boardOrientation()
        +int boardRotation()
        +int boardTranslation()
        +int boardScaling()
        +int boardDistortion()
        +int boardDeformation()
        +int boardStrain()
        +int boardStress()
        +int boardTension()
        +int boardCompression()
        +int boardBending()
        +int boardTwisting()
        +int boardBuckling()
        +int boardVibration()
        +int boardResonance()
        +int boardDamping()
        +int boardFriction()
        +int boardViscosity()
        +int boardElasticity()
        +int boardPlasticity()
        +int boardCreep()
        +int boardFatigue()
        +int boardCorrosion()
        +int boardOxidation()
        +int boardReduction()
        +int boardPolymerization()
        +int boardCrosslinking()
        +int boardDegradation()
        +int boardBiodegradation()
        +int boardPhotodegradation()
        +int boardThermodegradation()
        +int boardRadiolysis()
        +int boardPyrolysis()
        +int boardCombustion()
        +int boardOxidation()
        +int boardReduction()
        +int boardPolymerization()
        +int boardCrosslinking()
        +int boardDegradation()
        +int boardBiodegradation()
        +int boardPhotodegradation()
        +int boardThermodegradation()
        +int boardRadiolysis()
        +int boardPyrolysis()
        +int boardCombustion()
    }
    class Piece {
        +int pieceType()
        +int pieceColor()
        +int pieceValue()
        +int pieceWeight()
        +int pieceDensity()
        +int pieceTemperature()
        +int pieceHumidity()
        +int piecePressure()
        +int pieceAcceleration()
        +int pieceVelocity()
        +int piecePosition()
        +int pieceOrientation()
        +int pieceRotation()
        +int pieceTranslation()
        +int pieceScaling()
        +int pieceDistortion()
        +int pieceDeformation()
        +int pieceStrain()
        +int pieceStress()
        +int pieceTension()
        +int pieceCompression()
        +int pieceBending()
        +int pieceTwisting()
        +int pieceBuckling()
        +int pieceVibration()
        +int pieceResonance()
        +int pieceDamping()
        +int pieceFriction()
        +int pieceViscosity()
        +int pieceElasticity()
        +int piecePlasticity()
        +int pieceCreep()
        +int pieceFatigue()
        +int pieceCorrosion()
        +int pieceOxidation()
        +int pieceReduction()
        +int piecePolymerization()
        +int pieceCrosslinking()
        +int pieceDegradation()
        +int pieceBiodegradation()
        +int piecePhotodegradation()
        +int pieceThermodegradation()
        +int pieceRadiolysis()
        +int piecePyrolysis()
        +int pieceCombustion()
    }
    class Knight {
        +int knightType()
        +int knightColor()
        +int knightValue()
        +int knightWeight()
        +int knightDensity()
        +int knightTemperature()
        +int knightHumidity()
        +int knightPressure()
        +int knightAcceleration()
        +int knightVelocity()
        +int knightPosition()
        +int knightOrientation()
        +int knightRotation()
        +int knightTranslation()
        +int knightScaling()
        +int knightDistortion()
        +int knightDeformation()
        +int knightStrain()
        +int knightStress()
        +int knightTension()
        +int knightCompression()
        +int knightBending()
        +int knightTwisting()
        +int knightBuckling()
        +int knightVibration()
        +int knightResonance()
        +int knightDamping()
        +int knightFriction()
        +int knightViscosity()
        +int knightElasticity()
        +int knightPlasticity()
        +int knightCreep()
        +int knightFatigue()
        +int knightCorrosion()
        +int knightOxidation()
        +int knightReduction()
        +int knightPolymerization()
        +int knightCrosslinking()
        +int knightDegradation()
        +int knightBiodegradation()
        +int knightPhotodegradation()
        +int knightThermodegradation()
        +int knightRadiolysis()
        +int knightPyrolysis()
        +int knightCombustion()
    }
    class Bishop {
        +int bishopType()
        +int bishopColor()
        +int bishopValue()
        +int bishopWeight()
        +int bishopDensity()
        +int bishopTemperature()
        +int bishopHumidity()
        +int bishopPressure()
        +int bishopAcceleration()
        +int bishopVelocity()
        +int bishopPosition()
        +int bishopOrientation()
        +int bishopRotation()
        +int bishopTranslation()
        +int bishopScaling()
        +int bishopDistortion()
        +int bishopDeformation()
        +int bishopStrain()
        +int bishopStress()
        +int bishopTension()
        +int bishopCompression()
        +int bishopBending()
        +int bishopTwisting()
        +int bishopBuckling()
        +int bishopVibration()
        +int bishopResonance()
        +int bishopDamping()
        +int bishopFriction()
        +int bishopViscosity()
        +int bishopElasticity()
        +int bishopPlasticity()
        +int bishopCreep()
        +int bishopFatigue()
        +int bishopCorrosion()
        +int bishopOxidation()
        +int bishopReduction()
        +int bishopPolymerization()
        +int bishopCrosslinking()
        +int bishopDegradation()
        +int bishopBiodegradation()
        +int bishopPhotodegradation()
        +int bishopThermodegradation()
        +int bishopRadiolysis()
        +int bishopPyrolysis()
        +int bishopCombustion()
    }
    class Queen {
        +int queenType()
        +int queenColor()
        +int queenValue()
        +int queenWeight()
        +int queenDensity()
        +int queenTemperature()
        +int queenHumidity()
        +int queenPressure()
        +int queenAcceleration()
        +int queenVelocity()
        +int queenPosition()
        +int queenOrientation()
        +int queenRotation()
        +int queenTranslation()
        +int queenScaling()
        +int queenDistortion()
        +int queenDeformation()
        +int queenStrain()
        +int queenStress()
        +int queenTension()
        +int queenCompression()
        +int queenBending()
        +int queenTwisting()
        +int queenBuckling()
        +int queenVibration()
        +int queenResonance()
        +int queenDamping()
        +int queenFriction()
        +int queenViscosity()
        +int queenElasticity()
        +int queenPlasticity()
        +int queenCreep()
        +int queenFatigue()
        +int queenCorrosion()
        +int queenOxidation()
        +int queenReduction()
        +int queenPolymerization()
        +int queenCrosslinking()
        +int queenDegradation()
        +int queenBiodegradation()
        +int queenPhotodegradation()
        +int queenThermodegradation()
        +int queenRadiolysis()
        +int queenPyrolysis()
        +int queenCombustion()
    }
    class King {
        +int kingType()
        +int kingColor()
        +int kingValue()
        +int kingWeight()
        +int kingDensity()
        +int kingTemperature()
        +int kingHumidity()
        +int kingPressure()
        +int kingAcceleration()
        +int kingVelocity()
        +int kingPosition()
        +int kingOrientation()
        +int kingRotation()
        +int kingTranslation()
        +int kingScaling()
        +int kingDistortion()
        +int kingDeformation()
        +int kingStrain()
        +int kingStress()
        +int kingTension()
        +int kingCompression()
        +int kingBending()
        +int kingTwisting()
        +int kingBuckling()
        +int kingVibration()
        +int kingResonance()
        +int kingDamping()
        +int kingFriction()
        +int kingViscosity()
        +int kingElasticity()
        +int kingPlasticity()
        +int kingCreep()
        +int kingFatigue()
        +int kingCorrosion()
        +int kingOxidation()
        +int kingReduction()
        +int kingPolymerization()
        +int kingCrosslinking()
        +int kingDegradation()
        +int kingBiodegradation()
        +int kingPhotodegradation()
        +int kingThermodegradation()
        +int kingRadiolysis()
        +int kingPyrolysis()
        +int kingCombustion()
    }
    class Rook {
        +int rookType()
        +int rookColor()
        +int rookValue()
        +int rookWeight()
        +int rookDensity()
        +int rookTemperature()
        +int rookHumidity()
        +int rookPressure()
        +int rookAcceleration()
        +int rookVelocity()
        +int rookPosition()
        +int rookOrientation()
        +int rookRotation()
        +int rookTranslation()
        +int rookScaling()
        +int rookDistortion()
        +int rookDeformation()
        +int rookStrain()
        +int rookStress()
        +int rookTension()
        +int rookCompression()
        +int rookBending()
        +int rookTwisting()
        +int rookBuckling()
        +int rookVibration()
        +int rookResonance()
        +int rookDamping()
        +int rookFriction()
        +int rookViscosity()
        +int rookElasticity()
        +int rookPlasticity()
        +int rookCreep()
        +int rookFatigue()
        +int rookCorrosion()
        +int rookOxidation()
        +int rookReduction()
        +int rookPolymerization()
        +int rookCrosslinking()
        +int rookDegradation()
        +int rookBiodegradation()
        +int rookPhotodegradation()
        +int rookThermodegradation()
        +int rookRadiolysis()
        +int rookPyrolysis()
        +int rookCombustion()
    }
    class Pawn {
        +int pawnType()
        +int pawnColor()
        +int pawnValue()
        +int pawnWeight()
        +int pawnDensity()
        +int pawnTemperature()
        +int pawnHumidity()
        +int pawnPressure()
        +int pawnAcceleration()
        +int pawnVelocity()
        +int pawnPosition()
        +int pawnOrientation()
        +int pawnRotation()
        +int pawnTranslation()
        +int pawnScaling()
        +int pawnDistortion()
        +int pawnDeformation()
        +int pawnStrain()
        +int pawnStress()
        +int pawnTension()
        +int pawnCompression()
        +int pawnBending()
        +int pawnTwisting()
        +int pawnBuckling()
        +int pawnVibration()
        +int pawnResonance()
        +int pawnDamping()
        +int pawnFriction()
        +int pawnViscosity()
        +int pawnElasticity()
        +int pawnPlasticity()
        +int pawnCreep()
        +int pawnFatigue()
        +int pawnCorrosion()
        +int pawnOxidation()
        +int pawnReduction()
        +int pawnPolymerization()
        +int pawnCrosslinking()
        +int pawnDegradation()
        +int pawnBiodegradation()
        +int pawnPhotodegradation()
        +int pawnThermodegradation()
        +int pawnRadiolysis()
        +int pawnPyrolysis()
        +int pawnCombustion()
    }
    class Knight <<abstract>>
    class Bishop <<abstract>>
    class Queen <<abstract>>
    class King <<abstract>>
    class Rook <<abstract>>
    class Pawn <<abstract>>
    Chessboard <|-- Knight
    Chessboard <|-- Bishop
    Chessboard <|-- Queen
    Chessboard <|-- King
    Chessboard <|-- Rook
    Chessboard <|-- Pawn
    Knight <|-- KnightConcrete
    Bishop <|-- BishopConcrete
    Queen <|-- QueenConcrete
    King <|-- KingConcrete
    Rook <|-- RookConcrete
    Pawn <|-- PawnConcrete
    KnightConcrete <|-- KnightConcreteSub
    BishopConcrete <|-- BishopConcreteSub
    QueenConcrete <|-- QueenConcreteSub
    KingConcrete <|-- KingConcreteSub
    RookConcrete <|-- RookConcreteSub
    PawnConcrete <|-- PawnConcreteSub
    KnightConcreteSub <|-- KnightConcreteSubSub
    BishopConcreteSub <|-- BishopConcreteSubSub
    QueenConcreteSub <|-- QueenConcreteSubSub
    KingConcreteSub <|-- KingConcreteSubSub
    RookConcreteSub <|-- RookConcreteSubSub
    PawnConcreteSub <|-- PawnConcreteSubSub
    KnightConcreteSubSub <|-- KnightConcreteSubSubSub
    BishopConcreteSubSub <|-- BishopConcreteSubSubSub
    QueenConcreteSubSub <|-- QueenConcreteSubSubSub
    KingConcreteSubSub <|-- KingConcreteSubSubSub
    RookConcreteSubSub <|-- RookConcreteSubSubSub
    PawnConcreteSubSub <|-- PawnConcreteSubSubSub
    KnightConcreteSubSubSub <|-- KnightConcreteSubSubSubSub
    BishopConcreteSubSubSub <|-- BishopConcreteSubSubSubSub
    QueenConcreteSubSubSub <|-- QueenConcreteSubSubSubSub
    KingConcreteSubSubSub <|-- KingConcreteSubSubSubSub
    RookConcreteSubSubSub <|-- RookConcreteSubSubSubSub
    PawnConcreteSubSubSub <|-- PawnConcreteSubSubSubSub
    KnightConcreteSubSubSubSub <|-- KnightConcreteSubSubSubSubSub
    BishopConcreteSubSubSubSub <|-- BishopConcreteSubSubSubSubSub
    QueenConcreteSubSubSubSub <|-- QueenConcreteSubSubSubSubSub
    KingConcreteSubSubSubSub <|-- KingConcreteSubSubSubSubSub
    RookConcreteSubSubSubSub <|-- RookConcreteSubSubSubSubSub
    PawnConcreteSubSubSubSub <|-- PawnConcreteSubSubSubSubSub
    KnightConcreteSubSubSubSubSub <|-- KnightConcreteSubSubSubSubSubSub
    BishopConcreteSubSubSubSubSub <|-- BishopConcreteSubSubSubSubSubSub
    QueenConcreteSubSubSubSubSub <|-- QueenConcreteSubSubSubSubSubSub
    KingConcreteSubSubSubSubSub <|-- KingConcreteSubSubSubSubSubSub
    RookConcreteSubSubSubSubSub <|-- RookConcreteSubSubSubSubSubSub
    PawnConcreteSubSubSubSubSub <|-- PawnConcreteSubSubSubSubSubSub
    KnightConcreteSubSubSubSubSubSub <|-- KnightConcreteSubSubSubSubSubSubSub
    BishopConcreteSubSubSubSubSubSub <|-- BishopConcreteSubSubSubSubSubSubSub
    QueenConcreteSubSubSubSubSubSub <|-- QueenConcreteSubSubSubSubSubSubSub
    KingConcreteSubSubSubSubSubSub <|-- KingConcreteSubSubSubSubSubSubSub
    RookConcreteSubSubSubSubSubSub <|-- RookConcreteSubSubSubSubSubSubSub
    PawnConcreteSubSubSubSubSubSub <|-- PawnConcreteSubSubSubSubSub
```

Programmation orientée objet

3 Conception

Cette partie présente comment nous avons conçu notre programme et son but est de décrire le rôle et la responsabilité de chacune de nos classes et comment elles nous ont permis de créer le programme.

3.1 Pièces

Les pièces sont représentées par la classe abstraite `Piece` dans laquelle on retrouve en particulier le type, la couleur, sa position sur le plateau et la liste des mouvements légaux qu'elle peut effectuer.

Chaque pièce est spécialisée selon son type (`King`, `Queen`, `Rook`, `Pawn`, etc.). Dans la construction de ces pièces, on initialise la liste des mouvements qu'elle peut effectuer.

3.2 Mouvements des pièces

Les mouvements sont représentés par une classe `Move` dans lequel on stocke la pièce qui réalise le mouvement et si elle peut mettre en échec. Nous avons une méthode permettant de définir si le mouvement peut être effectué.

On spécialise ensuite les mouvements selon les règles du jeu qu'on veut implémenter : en passant, grand et petit roque, mouvement linéaire, etc.

Pour les mouvements, nous avons utilisé de manière importante le mécanisme de liaison dynamique. En effet, chaque mouvement implémente la méthode `canMove`, cette méthode est toujours appelée dans la classe la plus spécialisée et à chaque fois si le mouvement est possible selon la règle implémentée, on appelle le `canMove` de la classe parente, ainsi avec ce mécanisme et l'héritage, on peut composer différentes règles de déplacement des pièces.

3.3 Plateau du jeu

Le plateau du jeu représente l'état des pièces sur le jeu, on y référence les deux rois pour une rapidité d'accès pour les vérifications d'échec. En plus des rois, on garde une liste des pions, nous permettant de gérer plus simplement la promotion de pion.

On trouve aussi des informations concernant les potentielles pièces impliquées dans le dernier mouvement effectué (par exemple pour un roque ou une prise en passant), si un mouvement spécial est effectué et finalement des méthodes utilitaires nous permettant de modifier l'état du jeu (`put` et `remove` entre autre).

3.4 Contrôleur

La classe contrôleur va de pair avec celle du plateau de jeu. Le plateau fait la représentation et le contrôleur la logique.

Elle permet de vérifier l'état du jeu en permanence, en s'assurant que les mouvements effectués par les joueurs respectent les règles établies. Elle gère également toute la logique du jeu, comme la détection d'échec, mat et pat. En cas de mouvement illégal détecté, le contrôleur peut annuler le coup et informer les joueurs de l'erreur.

De plus, il peut également gérer les situations spéciales telles que les roques, les prises en passant et les promotions de pions. En somme, la classe contrôleur assure un fonctionnement fluide et juste du jeu en vérifiant constamment l'état du jeu et en traitant toute la logique de celui-ci.

4 Hypothèses de travail

4.1 Égalité par insuffisance de matériel

L'égalité par insuffisance de matériel est une règle qui varie selon les fédérations et compétitions. Nous nous sommes donc basés sur sa version la plus simple et la plus répandue. Dans cette version, l'égalité est déclarée dans le cas où les deux joueurs n'ont plus qu'un roi et/ou un fou et/ou un cavalier sur le plateau de jeu. Il ne doit rester qu'au maximum un exemplaire de chacune de ces pièces par couleur sur le plateau.

Nous avons utilisé Wikipédia qui répertorie toutes les règles du jeu pour les cas d'égalité et d'échec.

5 Tests

Les tests réalisés sont des tests fonctionnels et non unitaires. Chaque test est valide pour les deux joueurs, il n'y a pas de distinction. Nous les avons regroupés par catégories pour faciliter la lecture.

Les pièces qui n'ont pas de mouvements ou de fonctionnalités particulières ont eu uniquement des tests de déplacements (listés dans pièce). Le reste des pièces ont des tests spécifiques à leur fonctionnalité dans leur catégorie respective.

5.1 Pièces

Cette partie a été testée pour chaque pièce indépendamment.

Tests effectués	Résultats obtenus
Peut réaliser ses mouvements légaux	✓
Ne peut pas réaliser des mouvements illégaux	✓

5.1.1 Roi

Tests effectués	Résultats obtenus
Petit roque (les cases entre la tour et le roi sont libres)	✓
Grand roque (les cases entre la tour et le roi sont libres)	✓
Roque impossible en cas d'échec	✓
Roque impossible si une case sur le chemin du roi est attaquée	✓
Roque impossible si le roi ou la tour a déjà bougé	✓
Roque possible si la tour ou la case adjacente (grand roque) est attaquée	✓

5.1.2 Tour

Tests effectués	Résultats obtenus
Petit roque (réagit bien au petit roque du roi)	✓
Grand roque (réagit bien au grand roque du roi)	✓

5.1.3 Dame

À part ses mouvements légaux, la dame n'a pas de tests à part entière.

5.1.4 Fou

À part ses mouvements légaux, le fou n'a pas de tests à part entière.

5.1.5 Pion

Tests effectués	Résultats obtenus
Peut avancer sur une case libre devant lui mais ne peut pas manger	✓
Peut avancer de 2 au premier tour si les cases sont vides	✓
Peut prendre un pion en passant	✓
Ne peut pas prendre un pion en passant si ce dernier n'a pas bougé de 2 cases au tour précédent	✓
Peut être promu en Reine, Tour, Fou ou Cavalier en atteignant la dernière ligne	✓

5.1.6 Cavalier

Tests effectués	Résultats obtenus
Peut passer par dessus des pièces	✓

5.2 Jeu

Tests effectués	Résultats obtenus
Peut créer une nouvelle partie	✓
Début de partie pour les blancs	✓
Chaque joueur joue tour par tour en alternance	✓
Mise en échec détectée	✓
Peut bouger une pièce en échec seulement pour enlever l'échec	✓
Peut manger la pièce qui met en échec pour enlever l'échec	✓
Ne peut pas enlever une pièce pour rendre son roi vulnérable	✓
Début de partie pour les blancs	✓
L'égalité par PAT fonctionne (stalemate)	✓
L'égalité par insuffisance de matériels fonctionne	✓
La promotion peut échec et mat ou PAT	✓
Le roque peut échec et mat ou PAT	✓