1) Préface

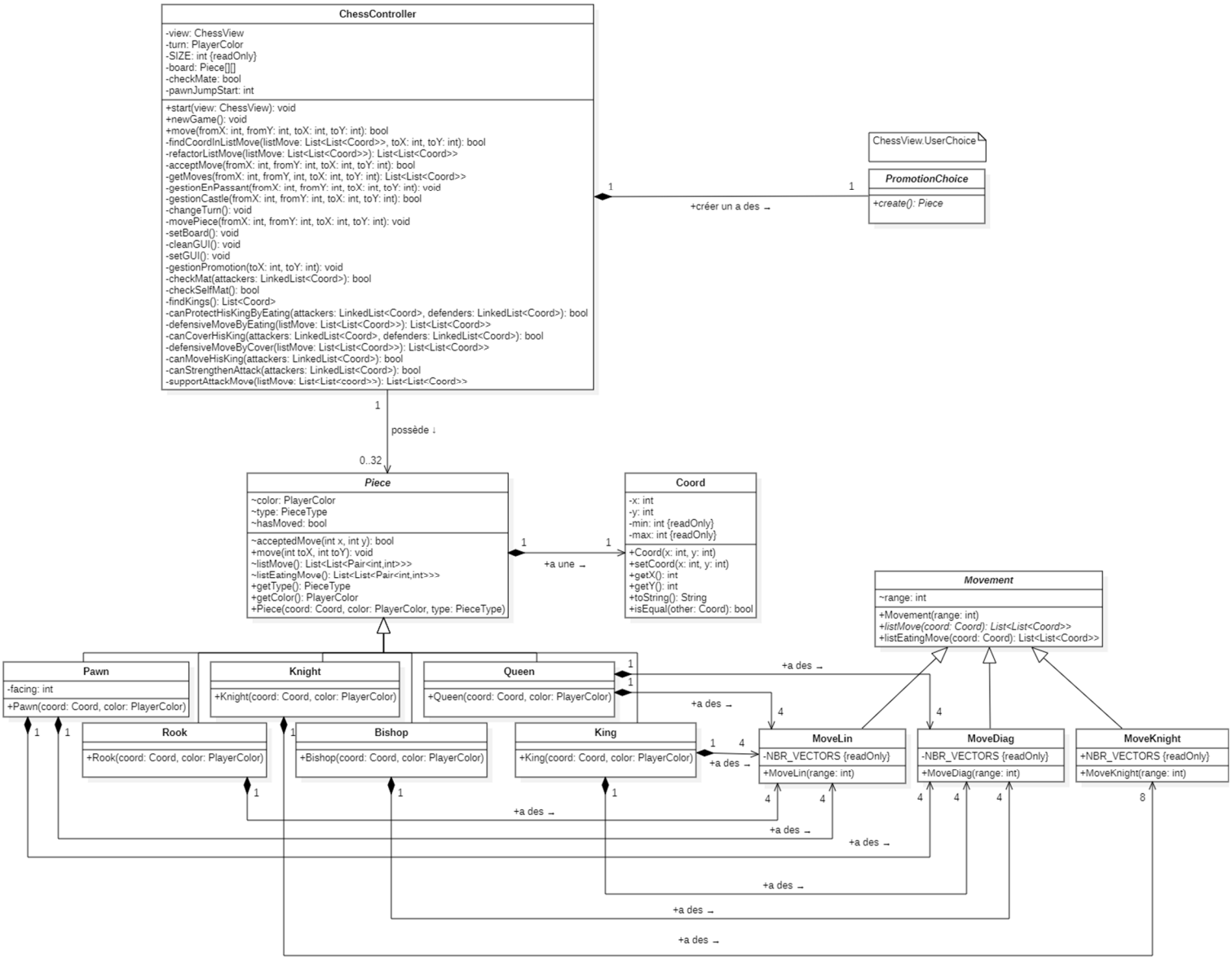
Dans le cadre de ce laboratoire numéro 8, nous avons dû implémenter un programme permettant de jouer aux échecs.

Toutefois, il nous a été demandé d'encoder uniquement le mouvement des pièces, car le reste nous a été fourni. En effet, deux interfaces nous ont été mises à disposition.

Ces dernières s'occupent de générer une interface graphique permettant de visualiser un échiquier ainsi que le matériel nécessaire au bon déroulement d'une partie d'échecs.

Toutes les fonctionnalités d'un jeu d'échecs traditionnelles doivent être fonctionnelles. Néanmoins, l'implémentation de la détection de l'échec et mat et des matchs nuls par pat ou par impossibilité de mater sont des points bonus.

2) UML


ChessController
-view: ChessView
-turn: PlayerColor
-SIZE: int {readOnly}
-board: Piece[][]
-checkMate: bool
-pawnJumpStart: int
+start(view: ChessView): void
+newGame(): void
+move(fromX: int, fromY: int, toX: int, toY: int): bool
-findCoordInListMove(listMove: List<List<Coord>>, toX: int, toY: int): bool
-refactorListMove(listMove: List<List<Coord>>): List<List<Coord>>
-acceptMove(fromX: int, fromY: int, toX: int, toY: int): bool
-getMoves(fromX: int, fromY, int, toX: int, toY: int): List<List<Coord>>
-gestionEnPassant(fromX: int, fromY: int, toX: int, toY: int): void
-gestionCastle(fromX: int, fromY: int, toX: int, toY: int): bool
-changeTurn(): void
-movePiece(fromX: int, fromY: int, toX: int, toY: int): void
-setBoard(): void
-cleanGUI(): void
-setGUI(): void
-gestionPromotion(toX: int, toY: int): void
-checkMat(attackers: LinkedList<Coord>): bool
-checkSelfMat(): bool
-findKings(): List<Coord>
-canProtectHisKingByEating(attackers: LinkedList<Coord>, defenders: LinkedList<Coord>): bool
-defensiveMoveByEating(listMove: List<List<Coord>>): List<List<Coord>>
-canCoverHisKing(attackers: LinkedList<Coord>, defenders: LinkedList<Coord>): bool
-defensiveMoveByCover(listMove: List<List<Coord>>): List<List<Coord>>
-canMoveHisKing(attackers: LinkedList<Coord>): bool
-canStrengthenAttack(attackers: LinkedList<Coord>): bool
-supportAttackMove(listMove: List<List<coord>>): List<List<Coord>>
ChessView.UserChoice
PromotionChoice
+create(): Piece
+créer un a des
possède
Piece
~color: PlayerColor
~type: PieceType
~hasMoved: bool
~acceptedMove(int x, int y): bool
+move(int toX, int toY): void
~listMove(): List<List<Pair<int,int>>>
~listEatingMove(): List<List<Pair<int,int>>>
+getType(): PieceType
+getColor(): PlayerColor
+Piece(coord: Coord, color: PlayerColor, type: PieceType)
+a une
Coord
-x: int
-y: int
-min: int {readOnly}
-max: int {readOnly}
+Coord(x: int, y: int)
+setCoord(x: int, y: int)
+getX(): int
+getY(): int
+toString(): String
+isEqual(other: Coord): bool
Movement
~range: int
+Movement(range: int)
+listMove(coord: Coord): List<List<Coord>>
+listEatingMove(coord: Coord): List<List<Coord>>
Pawn
-facing: int
+Pawn(coord: Coord, color: PlayerColor)
Knight
+Knight(coord: Coord, color: PlayerColor)
Queen
+Queen(coord: Coord, color: PlayerColor)
Rook
+Rook(coord: Coord, color: PlayerColor)
Bishop
+Bishop(coord: Coord, color: PlayerColor)
King
+King(coord: Coord, color: PlayerColor)
MoveLin
-NBR_VECTORS {readOnly}
+MoveLin(range: int)
MoveDiag
-NBR_VECTORS {readOnly}
+MoveDiag(range: int)
MoveKnight
+NBR_VECTORS {readOnly}
+MoveKnight(range: int)
+a des

2) Description des classes

2.1) ChessController

Cette classe implémente l'interface ChessController. De plus, cette classe permet de gérer toute l'intelligence nécessaire pour un jeu d'échecs. En effet, cette classe contient les méthodes permettant de gérer la prise en passant, le mat ou encore l'échec et mat par exemple.

2.2) Piece

Il s'agit d'une classe abstraite permettant de recenser toutes les choses communes des différents types de pièces.

2.3) Pawn, Knight, Queen, Rook, Bishop, King

Ces classes concrètes héritent de la classe abstraite Piece. Elle représente le matériel.

2.4) Movement

Il s'agit d'une classe abstraite permettant de stocker les parties communes des différents mouvements concrets que nous avions implémentés.

2.5) MoveLin, MoveDiag, MoveKnight

Ces classes permettent de générer des mouvements linéaires, diagonaux ou des mouvements propres pour le cavalier. Un mouvement correspond à plusieurs vecteurs de points. Par exemple, la classe MoveDiag permet de générer 4 vecteurs diagonales à partir d'un point.

2.6) Coord

Nous avions créé cette classe afin de stocker deux entiers afin de modéliser une coordonnée (x ; y). Cela a été fait, car nous n'avions pas réussi à utiliser une Paire. En outre, nous avions créé notre paire d'entiers.

2.7) Promotion Choice

Il s’agit d’une classe abstraite permettant de créer une pièce lors ce qu’un pion doit être promu.

3) Choix de conception

3.1) ChessController

Nous avons décidé d'implémenter un tableau de pièce à deux dimensions permettant de matérialiser l'échiquier, car nous pensions qu'il n'est pas nécessaire de faire une classe échiquier, car la gestion de l'affichage est déjà gérer grâce aux interfaces mit à disposition pour ce laboratoire. De plus, grâce au tableau nous pouvons accéder à n'importe quelle pièce en O(1).

3.2) Piece

Normalement, la pièce ne devrait bénéficier d'aucune intelligence particulière. Toutefois, nous avions décidé de mettre les coordonnées dans la pièce afin de simplifier grandement certains aspects du code. De plus, cela nous évite de devoir stocker des positions dans des listes de pièces. En outre, nous avions priorisé l'espace mémoire à la performance.

Cependant, après une nième réflexion / discussion nous pensions que ce choix n'est pas le plus optimal, car si nous souhaitions créer un ordinateur contre lequel jouer, cela augmenterait le temps de calcul de ce dernier (selon le nombre de coups à l'avance que l'on calcul, le temps peut drastiquement augmenter).

3.3) Movement

Nous avions décidé de faire ça comme ça afin d'éviter de la redondance dans le code des pièces concrètes. Lorsque nous avons encodé les déplacements possibles de la reine, nous avons constaté que ces mouvements correspondent à ceux du fou et à ceux de la tour. Nous avions donc décidé de factoriser le code de cette manière.

3.4) Check & Checkmat

Afin de réaliser la détection des échecs et mate, nous avions décidé de jouer le coup puis de contrôler s'il y a un échec ou pas (tout en contrôlant que le mouvement est légal). Dans le cas où le coup est légal, le coup est joué (la pièce est déplacée). En contrepartie si un joueur se met en échec

tout seul, le coup est joué puis contrôlé et enfin le coup est annulé. Nous avions décidé de faire au plus simple pour cette partie.

4) Tests effectués

Ci-dessous, voici la liste des différents tests que nous avons réalisés durant nos parties :

Description du test	Résultat observé
Le bouton « New Game » de l'interface graphique permet bien de lancer une nouvelle partie en effaçant la précédente.	OK
Au début d'une partie, le trait est au blanc.	OK
Le trait s'alterne. Un message indique quel joueur doit jouer est visible dans l'interface.	OK
Au commencement d'une partie, le matériel est réparti correctement.	OK
Si un joueur met le roi adverse en échec, un message s'affiche.	OK
Si un joueur met le roi adverse en échec et mat, un message s'affiche et il n'est plus possible de jouer sans recommencer une nouvelle partie.	OK
Le petit et le grand roque sont fonctionnels. Le roque ne fonctionne pas si, le roi ou la tour à déjà bougé, si le roi est mis en échec, si une des cases sur laquelle passe le roi est contrôlée par l'adversaire.	OK
Les pièces ne peuvent pas aller sur une case occupée par une pièce de la même couleur.	OK
Une pièce ne peut pas réaliser un mouvement si celui.ci met son propre roi en échec.	OK

Tableau 1 Tests généraux

Description du test	Résultat observé
La reine peut faire des mouvements linéaires et diagonaux rectilignes.	OK
Les mouvements de la reine n'ont pas de limites. Elle peut parcourir tout l'échiquier s'il n'y a pas de pièce qui la bloque.	OK
La reine ne peut pas sauter par-dessus une autre pièce.	OK

Tableau 2 Tests de la reine

Description du test	Résultat observé
Le roi peut se déplacer partout autour de lui avec une distance de 1 (pour autant que personne ne le bloque).	OK
Le roi ne peut pas aller sur une case qui est contrôlée par une pièce adverse.	OK

Tableau 3 Tests du roi

Description du test	Résultat observé
La tour peut faire des mouvements linéaires rectilignes.	OK
Les mouvements de la tour n'ont pas de limites. Elle peut parcourir tout l'échiquier s'il n'y a pas de pièce qui la bloque.	OK
La tour ne peut pas sauter par-dessus une autre pièce.	OK

Tableau 4 Tests de la tour

Description du test	Résultat observé
Le fou peut faire des mouvements diagonaux rectilignes.	OK
Les mouvements du fou n'ont pas de limites. Il peut parcourir tout l'échiquier s'il n'y a pas de pièce qui le bloque.	OK
Le fou ne peut pas sauter par-dessus une autre pièce.	OK

Tableau 5 Tests du fou

Description du test	Résultat observé
Le cavalier ne peut faire que des mouvements en « L ». Le mouvement en « L » combine un déplacement de deux cases dans une direction suivie d'un déplacement d'une case de façon perpendiculaire.	OK
Le cavalier peut sauter par-dessus une autre pièce.	OK

Tableau 6 Tests du cavalier

Description du test	Résultat observé
Les pions ne peuvent bouger que dans une seule direction (les blancs vers le haut et les noirs vers le bas).	OK
Si les pions n'ont pas bougé, ils peuvent se déplacer d'une ou deux cases. Dès lors qu'ils ont bougé, ils ne peuvent se déplacer que d'une case.	OK
Un pion ne peut manger uniquement si une pièce adverse se situe en diagonale de ce dernier (dans la bonne direction).	OK
La prise en passant est implémentée et fonctionnelle.	OK
La promotion d'un pion s'effectue lorsque ce dernier a atteint l'autre bout de l'échiquier. Les quatre choix de la promotion traditionnelle sont présents (Tour, Fou, Cavalier et Reine.).	OK

Tableau 7 Tests du pion

5) Exemple d'échec et mat

5.1) Mat du berger

Le fou de case blanc et la dame ont attaqué le pion noir F7. Lorsque le pion F7 est mangé dans cette position avec les noirs, le roi noir est en échec et mat.

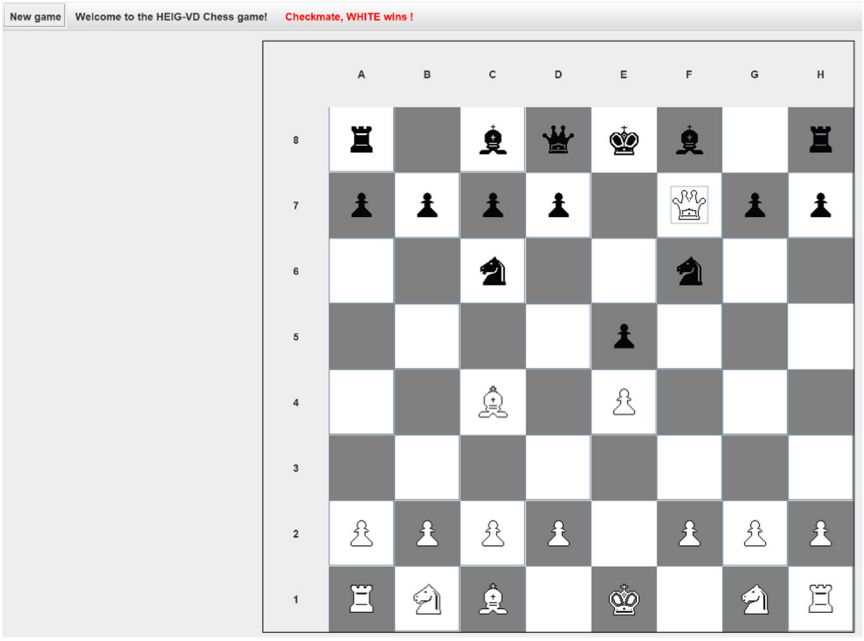


Figure 1 Mate du Berger

5.2) Mat à l'aide d'une batterie dame-tour

Le roi blanc est en échec et mat, car il ne peut pas manger la dame noire, car elle est protégée par la tour noire.

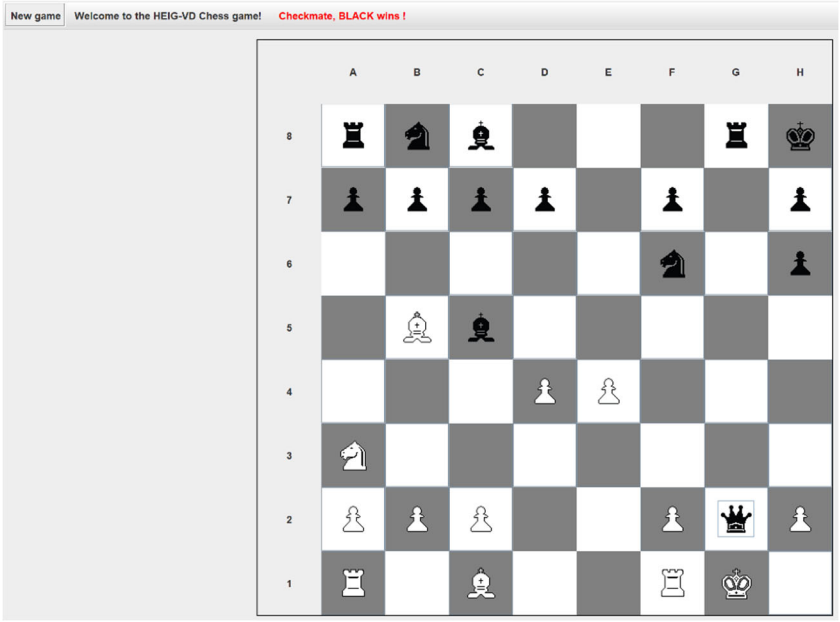


Figure 2 Batterie dame-tour qui met échec et mat au roi blanc

5.3) Mat d'Anastasie

Le roi noir est en échec, car, car il est maté par la tour blanche. Elle contrôle les cases H7 et H8, tandis que le cavalier contrôle la case G8.

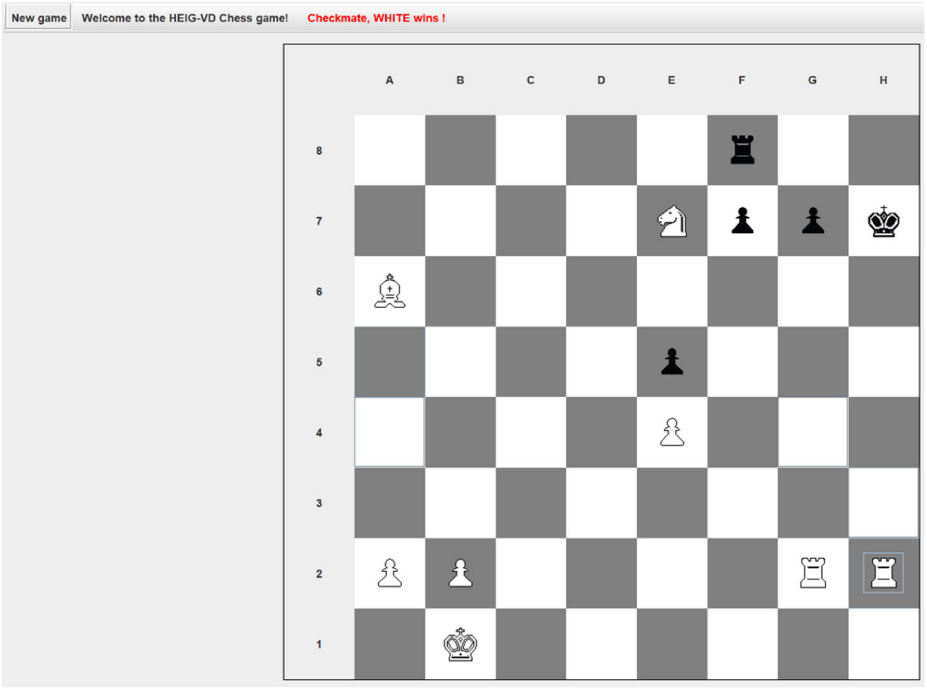


Figure 3 Mate d'Anastasie