```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : Main.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Déclaration de la classe : Main

Remarque(s) : Util pour démarrer l'interface graphique

Compilateur : jdk1.8.0_221

--------------------------- */
package engine;

import chess.ChessController;
import chess.ChessView;
import chess.views.console.ConsoleView;
import chess.views.gui.GUIView;

public class Main {
    public static void main(String ... args){
        ChessController chessboard = new Plateau();
        //ChessView view = new GUIView(chessboard);
        ChessView view = new ConsoleView(chessboard);
        chessboard.start(view);
    }
}

```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : Plateau.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Déclaration de la classe : Plateau

Remarque(s) : Permet d'assurer le déroulement d'une partie en fonction des règles implémentée.
              Hérite de ChessController

Compilateur : jdk1.8.0_221

--------------------------- */
package engine;

import chess.ChessController;
import chess.ChessView;
import chess.PieceType;
import chess.PlayerColor;
import engine.pieces.*;

import java.util.ArrayList;
import java.util.Arrays;

public class Plateau implements ChessController {

    private static final int DIMENSION = 8; //Défini la dimension standard d'un échiquier

    private int tour;           //Compte le nombre de tour durant une partie
    private Case[][] plateau;   //Echiquier de la partie courante
    private ChessView view;     //Necessaire à l'UI

    /**
     * Constructeur, crée le plateau de jeu
     */
    public Plateau() {
        plateau = new Case[DIMENSION][DIMENSION];
        for (int colonne = 0; colonne < DIMENSION; colonne++) {
            for (int ligne = 0; ligne < DIMENSION; ligne++) {
                plateau[colonne][ligne] = new Case(colonne, ligne);
            }
        }
    }

    @Override
    public void start(ChessView view) {
        this.view = view;
        view.startView();
    }

    @Override
    public boolean move(int fromX, int fromY, int toX, int toY) {
        view.displayMessage("");

        Case caseFrom = plateau[fromX][fromY];
        Case caseTo = plateau[toX][toY];

        if (caseFrom.estVide()) {
            return false;
        }

        // Piece deplacee
        Piece p = caseFrom.getPieceCourante();

        // Permet de savoir quelle couleur doit jouer
        if (tour % 2 == 1 && p.getColor() != PlayerColor.WHITE || tour % 2 == 0 && p.getColor()
        != PlayerColor.BLACK) {
            return false;
        }

        // Si la trajectoire est libre ou que la pièce est un cavalier
        // Test de mise en echec de son propre roi dans un plateau temporaire
        if (trajectoireLibre(caseFrom, caseTo, plateau) || p.getPieceType() ==
        PieceType.KNIGHT) {
            Case[][] plateauTestEchec = dupliquerPlateau();
            Case caseFromTestEchec = plateauTestEchec[fromX][fromY];
```

```
76                  Case caseToTestEchec = plateauTestEchec[toX][toY];
77                  Piece pTestEchec = caseFromTestEchec.getPieceCourante();
78
79                  MouvementType mouvementTypeTestEchec = pTestEchec.deplacer(caseFromTestEchec,
                    caseToTestEchec);
80
81                  // Test si la case parcouru par le roi pendant son roque le mettrait en echec
82                  if (mouvementTypeTestEchec == MouvementType.PETIT_ROQUE || mouvementTypeTestEchec
                    == MouvementType.GRAND_ROQUE) {
83                      if (echec(pTestEchec.getColor() == PlayerColor.BLACK ? PlayerColor.WHITE :
                        PlayerColor.BLACK, plateauTestEchec[mouvementTypeTestEchec ==
                        MouvementType.GRAND_ROQUE ? toX + 1 : toX - 1][toY], plateauTestEchec)) {
84                          view.displayMessage("Roque interdit : la case parcouru met le roi en échec");
85                          return false;
86                      }
87                  }
88
89                  // Test si le mouvement met le roi en echec
90                  if (echec(pTestEchec.getColor() == PlayerColor.BLACK ? PlayerColor.WHITE :
                    PlayerColor.BLACK, trouverRoi(p.getColor(), plateauTestEchec), plateauTestEchec)) {
91                      view.displayMessage("Interdit de mettre en echec son roi");
92                      return false;
93                  }
94              } else {
95                  return false;
96              }
97
98              MouvementType mouvementTypeActuel = p.deplacer(caseFrom, caseTo);
99
100             // Si le mouvement est non-valide
101             if (mouvementTypeActuel == MouvementType.NON_VALIDE) {
102                 return false;
103             }
104
105
106             // Prise en passant
107             if (mouvementTypeActuel == MouvementType.EN_PASSANT) {
108                 // Position du vrai pion par rapport à la case du pion fantôme
109                 // Est inversé si le joueur est noir
110                 int pionPosition = -1;
111                 if (p.getColor() == PlayerColor.BLACK)
112                     pionPosition = 1;
113
114                 // Suppresion du vrai pion
115                 plateau[toX][toY + pionPosition].supprimerPiece();
116                 view.removePiece(toX, toY + pionPosition);
117
118                 view.displayMessage("Prise en passant");
119             }
120
121             // Si le mouvement est une promotion on promeut
122             if (mouvementTypeActuel == MouvementType.PROMOTION) {
123                 promouvoir(p.getColor(), caseTo);
124
125                 view.displayMessage("Promotion");
126             }
127
128             // Si le mouvement est une tentative de petit roque
129             if (mouvementTypeActuel == MouvementType.PETIT_ROQUE) {
130                 if (roque(1, toX, toY))
131                     view.displayMessage("Petit roque");
132
133
134             }
135
136             // Si le mouvement est une tentative de grand roque
137             if (mouvementTypeActuel == MouvementType.GRAND_ROQUE) {
138                 if (plateau[toX - 1][toY].estVide()) {
139                     if (roque(-2, toX, toY))
140                         view.displayMessage("Grand roque");
141                 } else {
142                     view.displayMessage("Roque interdit : le chemin entre le roi et la tour n'est
                        pas libre");
143                     return false;
144                 }
145
146             }
```

```java

        // Vider le plateau de tous les pions fantomes
        for (int x = 0; x < DIMENSION; ++x) {
            for (int y = 0; y < DIMENSION; ++y) {
                plateau[x][y].supprimerPionFantome();
            }
        }

        // Si le mouvement est un premier mouvement d'un pion de 2 en avant
        if (mouvementTypeActuel == MouvementType.DOUBLE) {
            // Placer le pion fantome a la case entre la source et la destination
            plateau[(toX + fromX) / 2][(toY + fromY) / 2].placerPionFantome((Pion) p);

            view.displayMessage("Premier déplacement d'un pion (avancée de 2)");
        }


        if (echec(p.getColor(), trouverRoi(p.getColor() == PlayerColor.BLACK ?
        PlayerColor.WHITE : PlayerColor.BLACK, plateau), plateau)) {
            view.displayMessage("Echec");
        }

        view.removePiece(fromX, fromY);

        // Dans le cas d'une promotion on n'aimerait pas remettre un pion sur la dernière case
        // On a donc pas besoin de d'appeler view.putPiece puisqu'on s'en charge déjà dans la
        fonction promouvoir
        if (mouvementTypeActuel != MouvementType.PROMOTION) {
            view.putPiece(p.getPieceType(), p.getColor(), toX, toY);
        }

        tour++;

        return true;
    }

    /**
     * Permet de faire une copie de l'échiquier actuel
     * @return une copie de l'échiquier actuel
     */
    private Case[][] dupliquerPlateau() {
        Case[][] plateauDuplique = new Case[DIMENSION][DIMENSION];
        for (int x = 0; x < DIMENSION; x++) {
            for (int y = 0; y < DIMENSION; y++) {
                plateauDuplique[x][y] = new Case(plateau[x][y]);
            }
        }
        return plateauDuplique;
    }

    /**
     * Determine si une case est en échec
     *
     * @param color   Couleur de l'équipe attaquante
     * @param c       Case cible
     * @param plateau Plateau
     * @return
     */
    public boolean echec(PlayerColor color, Case c, Case[][] plateau) {
        for (int x = 0; x < DIMENSION; ++x) {
            for (int y = 0; y < DIMENSION; ++y) {
                Piece piece = plateau[x][y].getPieceCourante();
                if (piece != null) {
                    if (piece.getColor() == color) {

                        if (piece.mouvementPossible(plateau[x][y], c) !=
                        MouvementType.NON_VALIDE) {
                            if (trajectoireLibre(plateau[x][y], c, plateau) ||
                            piece.getPieceType() == PieceType.KNIGHT) {
                                return true;
                            }
                        }
                    }
                }
            }
        }
```

```
220             return false;
221         }
222
223         /**
224          * Permet de déplacer la tour lors du petit et du grand roque, après avoir vérifier si le
             roque est permis
225          *
226          * @param roque vaut 1 si c'est un petit roque, et -1 si c'est un grand roque
227          * @param x     x de destination du roi
228          * @param y     y de destination du roi
229          */
230         public boolean roque(int roque, int x, int y) {
231             if (plateau[x + roque][y].getPieceCourante().getPieceType() == PieceType.ROOK) {
232                 Tour tourRoque = (Tour) plateau[x + roque][y].getPieceCourante();
233                 if (tourRoque.isPremierDeplacement()) {
234                     view.removePiece(x + roque, y);
235                     int tourPosition = x - (roque / Math.abs(roque));
236                     view.putPiece(tourRoque.getPieceType(), tourRoque.getColor(), tourPosition, y);
237                     plateau[x + roque][y].supprimerPiece();
238                     plateau[tourPosition][y].placerPiece(tourRoque);
239
240                     return true;
241                 }
242             }
243             return false;
244         }
245
246         /**
247          * Permet de définir si une trajectoir est libre ou non sur un échiquier, en fonction d'une
             case source et d'une case destination
248          * @param src       case source
249          * @param dest      case destination
250          * @param plateau   échiquier en question
251          * @return true si libre, false sinon
252          */
253         public boolean trajectoireLibre(Case src, Case dest, Case[][] plateau) {
254             int deltaX = Math.abs(src.getX() - dest.getX());
255             int deltaY = Math.abs(src.getY() - dest.getY());
256
257             int dirX = 0;
258             int dirY = 0;
259
260             if (deltaX != 0)
261                 dirX = (src.getX() > dest.getX()) ? -1 : 1;
262             if (deltaY != 0)
263                 dirY = (src.getY() > dest.getY()) ? -1 : 1;
264
265             int x = src.getX() + dirX;
266             int y = src.getY() + dirY;
267             for (int i = 1; i < Math.max(deltaX, deltaY); i++) {
268
269                 if (plateau[x][y].aUnePiece())
270                     return false;
271                 x += dirX;
272                 y += dirY;
273             }
274             return true;
275         }
276
277         /**
278          * Permet de promouvoir un pion en une autre pièce choisi par l'utilisateur
279          * @param color Couleur du joueur qui joue son tour
280          * @param dest  Case de destination du pion promu
281          */
282         private void promouvoir(PlayerColor color, Case dest) {
283             // Les nouvelles pieces possibles
284             Piece dame = new Dame(color);
285             Piece cavalier = new Cavalier(color);
286             Piece tour = new Tour(color);
287             Piece fou = new Fou(color);
288
289             Piece pieceSelectionee = null;
290
291             while (pieceSelectionee == null) {
292                 pieceSelectionee = view.askUser("Promotion", "Choisir une pièce pour la promotion",
                     dame, cavalier, tour, fou);
293             }
```

```
294
295             int x = dest.getX(), y = dest.getY();
296
297             // On effectue la promotion du pion
298             plateau[x][y].supprimerPiece();
299             plateau[x][y].placerPiece(pieceSelectionee);
300             view.removePiece(x, y);
301             view.putPiece(pieceSelectionee.getPieceType(), pieceSelectionee.getColor(), x, y);
302         }
303
304         @Override
305         public void newGame() {
306             tour = 1;
307
308             // Vider le plateau de tous les pions fantomes
309             for (int x = 0; x < DIMENSION; ++x) {
310                 for (int y = 0; y < DIMENSION; ++y) {
311                     plateau[x][y].supprimerPionFantome();
312                 }
313             }
314
315             // On vide l'echiquier
316             for (int x = 0; x < DIMENSION; ++x) {
317                 for (int y = 0; y < DIMENSION; ++y) {
318                     Case caseCourrante = plateau[x][y];
319                     if (!caseCourrante.estVide()) {
320                         plateau[x][y].supprimerPiece();
321                     }
322                 }
323             }
324
325             ArrayList<Piece> whitePieces = new ArrayList<>(
326                     Arrays.asList(
327                             new Tour(PlayerColor.WHITE),
328                             new Cavalier(PlayerColor.WHITE),
329                             new Fou(PlayerColor.WHITE),
330                             new Dame(PlayerColor.WHITE),
331                             new Roi(PlayerColor.WHITE),
332                             new Fou(PlayerColor.WHITE),
333                             new Cavalier(PlayerColor.WHITE),
334                             new Tour(PlayerColor.WHITE)
335                     )
336             );
337
338             ArrayList<Piece> blackPieces = new ArrayList<>(
339                     Arrays.asList(
340                             new Tour(PlayerColor.BLACK),
341                             new Cavalier(PlayerColor.BLACK),
342                             new Fou(PlayerColor.BLACK),
343                             new Dame(PlayerColor.BLACK),
344                             new Roi(PlayerColor.BLACK),
345                             new Fou(PlayerColor.BLACK),
346                             new Cavalier(PlayerColor.BLACK),
347                             new Tour(PlayerColor.BLACK)
348                     )
349             );
350
351             for (int col = 0; col < DIMENSION; ++col) {
352                 Piece pionBlanc = new Pion(PlayerColor.WHITE);
353                 Piece pionNoir = new Pion(PlayerColor.BLACK);
354                 Piece pieceBlanche = whitePieces.get(col);
355                 Piece pieceNoire = blackPieces.get(col);
356
357                 plateau[col][1].placerPiece(pionBlanc);
358                 plateau[col][6].placerPiece(pionNoir);
359                 plateau[col][0].placerPiece(pieceBlanche);
360                 plateau[col][7].placerPiece(pieceNoire);
361
362                 view.putPiece(pieceBlanche.getPieceType(), pieceBlanche.getColor(), col, 0);
363                 view.putPiece(pieceNoire.getPieceType(), pieceNoire.getColor(), col, 7);
364                 view.putPiece(pionBlanc.getPieceType(), pionBlanc.getColor(), col, 1);
365                 view.putPiece(pionNoir.getPieceType(), pionNoir.getColor(), col, 6);
366             }
367         }
368
369         /**
370          * Permet de trouver le roi d'une couleur donnée sur échiquier donné
```

```
371         * @param color     Couleur du roi à trouver
372         * @param plateau   Echiquier sur lequel le roi doit être trouvé
373         * @return la case où se trouve le roi de la couleur voulue
374         */
375        private Case trouverRoi(PlayerColor color, Case[][] plateau) {
376            for (int x = 0; x < DIMENSION; ++x) {
377                for (int y = 0; y < DIMENSION; ++y) {
378                    Piece p = plateau[x][y].getPieceCourante();
379                    if (p == null)
380                        continue;
381                    if (p.getPieceType() == PieceType.KING && p.getColor() == color) {
382                        return plateau[x][y];
383                    }
384                }
385            }
386            return null;
387        }
388    }
389
```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : Case.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Déclaration de la classe : Case

Remarque(s) : Naviguabilité /!\ -> Une case connaît la pièce qui se trouve dessus, pas l'inverse

Compilateur : jdk1.8.0_221

--------------------------- */
package engine;

import engine.pieces.Piece;
import engine.pieces.Pion;
import sun.reflect.generics.reflectiveObjects.NotImplementedException;

public class Case {
    private final int x;    //Position de la case sur l'axe X
    private final int y;    //Position de la case sur l'axe Y

    private Piece pieceCourante;    //Pièce se trouvant sur la case
    private Pion pionFantome;       //Pion Fantôme pour la prise en passant

    Case(int x, int y){
        this.x = x;
        this.y = y;
    }

    Case(Case c){
        this.x = c.x;
        this.y = c.y;
        if(c.pieceCourante != null)
            this.pieceCourante = c.pieceCourante.clone();
        if(c.pionFantome != null)
            this.pionFantome = (Pion) c.pionFantome.clone();
    }

    /**
     * Vérifie s'il y a une pièce ou un pion fantôme sur la case
     * @return false si pas vide, true sinon
     */
    public boolean estVide(){
        return (pieceCourante == null && pionFantome == null);
    }

    /**
     * Vérifie s'il y a une pièce sur la case
     * @return false si pas vide, true sinon
     */
    public boolean aUnePiece(){
        return pieceCourante != null;
    }

    /**
     * Vérifie s'il y a un pion fantôme sur la case
     * @return false si pas vide, true sinon
     */
    public boolean aUnPionFantome(){
        return pionFantome != null;
    }

    /**
     * Getter de la pièce courante
     * @return la pièce courante
     */
    public Piece getPieceCourante() {return pieceCourante;}

    /**
     * Setter de la pièce courante
     * @param piece Pièce à mettre sur la case
     */
    public void placerPiece(Piece piece){
        pieceCourante = piece;
    }
```

```java

    /**
     * Setter du pion fantôme
     * @param pion Pion fantôme à mettre sur la case
     */
    public void placerPionFantome(Pion pion){
        pionFantome = pion;
    }

    /**
     * Supprimer la pièce courante
     * @return la pièce qui vient d'être supprimée
     */
    public Piece supprimerPiece(){
        Piece p = pieceCourante;
        pieceCourante = null;
        return p;
    }

    /**
     * Supprime le pion fantôme courant
     * @return le pion fantôme qui vient d'être supprimé
     */
    public Pion supprimerPionFantome(){
        Pion p = pionFantome;
        pionFantome = null;
        return p;
    }

    /**
     * Getter de la valeur de l'axe X de la case
     * @return position X
     */
    public int getX() {
        return x;
    }

    /**
     * Getter de la valeur de l'axe Y de la case
     * @return position Y
     */
    public int getY() {
        return y;
    }
}

```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : DirectionType.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Déclaration de l'enum : DirectionType

Remarque(s) : -

Compilateur : jdk1.8.0_221

--------------------------- */
package engine.pieces;

public enum DirectionType {
    DIAGONALE,
    DROIT,
    TOUS,
}

```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : MouvementType.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Déclaration de l'enum : MouvementType

Remarque(s) : -

Compilateur : jdk1.8.0_221

--------------------------- */
package engine.pieces;

public enum MouvementType {
    CLASSIQUE,
    NON_VALIDE,
    GRAND_ROQUE,
    PETIT_ROQUE,
    PROMOTION,
    DOUBLE,
    EN_PASSANT
}

```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : Piece.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Déclaration de la classe abstraite : Piece

Remarque(s) : Toutes les pièces utilisées dans le jeu hérite de cette classe

Compilateur : jdk1.8.0_221

--------------------------- */
package engine.pieces;

import chess.ChessView;
import chess.PieceType;
import chess.PlayerColor;
import engine.Case;

public abstract class Piece implements ChessView.UserChoice{
    PieceType pieceType;    //Type de la pièce
    PlayerColor color;      //Couleur de la pièce

    public Piece(PlayerColor color) {
        this.color = color;
    }

    /**
     * Determine si un mouvement est possible depuis une case source vers une case destination
     * @param src Case source
     * @param dest Case destination
     * @return un MouvementType en fonction du mouvement
     */
    public MouvementType mouvementPossible(Case src, Case dest){

        //Permet de ne pas pouvoir manger ces propres pièces
        if(dest.getPieceCourante() != null && dest.getPieceCourante().getColor() == this.color){
            return MouvementType.NON_VALIDE;
        }
        return MouvementType.CLASSIQUE;
    }

    /**
     * Permet de déplacer une pièce depuis une case source vers une case destination si le
     mouvement est valide
     * @param src Case source
     * @param dest Case destination
     * @return un MouvementType en fonction du mouvement
     */
    public MouvementType deplacer(Case src, Case dest){
        MouvementType mouvementType = mouvementPossible(src, dest);
        if(mouvementType == MouvementType.NON_VALIDE)
            return MouvementType.NON_VALIDE;

        src.supprimerPiece();
        dest.placerPiece(this);
        return mouvementType;
    }

    /**
     * Getter du type de la pièce
     * @return le type de la pièce
     */
    public PieceType getPieceType(){return pieceType;}

    /**
     * Getter de la couleur de la pièce
     * @return la couleur de la pièce
     */
    public PlayerColor getColor(){return  color;}

    public abstract Piece clone();

    @Override
    public String textValue() {
        return toString();
```

```
77          }
78
79          public abstract String toString();
80      }
81
```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : PieceDeplacementStandard.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Déclaration de la classe abstraite : PieceDeplacementStandard

Remarque(s) : Toutes les pièces utilisées dans le jeu hérite de cette classe, sauf le cavalier

Compilateur : jdk1.8.0_221

--------------------------- */
package engine.pieces;

import chess.PlayerColor;
import engine.Case;

public abstract class PieceDeplacementStandard extends Piece{
    int distanceDeplacementMax;     // Définis la distance maxmium qu'une pièce peux parcourir
    DirectionType directionType;    // Définis le type de direction que peux utiliser une pièce

    public PieceDeplacementStandard(PlayerColor color) {
        super(color);
    }

    @Override
    public MouvementType mouvementPossible(Case src, Case dest) {

        if (super.mouvementPossible(src, dest) == MouvementType.NON_VALIDE){
            return MouvementType.NON_VALIDE;
        }

        int deltaX = Math.abs(src.getX() - dest.getX());
        int deltaY = Math.abs(src.getY() - dest.getY());

        if((directionType == DirectionType.DROIT || directionType == DirectionType.TOUS) &&
        (src.getX() != dest.getX() ^ src.getY() != dest.getY())){
            if(Math.max(deltaX, deltaY) <= Math.abs(distanceDeplacementMax))
                return MouvementType.CLASSIQUE;
        }

        if((directionType == DirectionType.DIAGONALE || directionType == DirectionType.TOUS) &&
        deltaX == deltaY){
            if(deltaX <= distanceDeplacementMax)
                return MouvementType.CLASSIQUE;
        }

        return MouvementType.NON_VALIDE;
    }
}

```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : PiecePremierDeplacement.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Déclaration de la classe abstraite : PiecePremierDeplacement

Remarque(s) : Toutes les pièces qui nécessites de savoir si elles ont déjà été déplacées héritent de cette classe

Compilateur : jdk1.8.0_221

--------------------------- */
package engine.pieces;

import chess.PlayerColor;
import engine.Case;

public abstract class PiecePremierDeplacement extends PieceDeplacementStandard{
    boolean premierDeplacement = true;  // Permet de déterminer si une pièce a déjà effectué son premier mouvement

    public PiecePremierDeplacement(PlayerColor color) {
        super(color);
    }

    @Override
    public MouvementType deplacer(Case src, Case dest) {
        MouvementType mouvementType = super.deplacer(src, dest);
        if(mouvementType == MouvementType.NON_VALIDE)
            return MouvementType.NON_VALIDE;
        premierDeplacement = false;
        return mouvementType;
    }

    /**
     * Getter de la varible premierDeplacement
     * @return la valeur de premierDeplacement
     */
    public boolean isPremierDeplacement() {
        return premierDeplacement;
    }
}

```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : Cavalier.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Implémentation de la pièce : cavalier

Remarque(s) : -

Compilateur : jdk1.8.0_221

--------------------------- */
package engine.pieces;

import chess.PieceType;
import chess.PlayerColor;
import engine.Case;

public class Cavalier extends Piece{
    public Cavalier(PlayerColor color) {
        super(color);
        pieceType= PieceType.KNIGHT;
    }

    @Override
    public MouvementType mouvementPossible(Case src, Case dest) {
        if (super.mouvementPossible(src,dest) == MouvementType.NON_VALIDE){
            return MouvementType.NON_VALIDE;
        }

        int deltaX = Math.abs(src.getX() - dest.getX());
        int deltaY = Math.abs(src.getY() - dest.getY());

        if((deltaX == 1 && deltaY == 2) || (deltaX == 2 && deltaY == 1)){

            return MouvementType.CLASSIQUE;
        }
        return MouvementType.NON_VALIDE;
    }

    @Override
    public Piece clone() {
        return new Cavalier(this.color);
    }

    @Override
    public String toString() {
        return "Cavalier";
    }
}

```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : Dame.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Implémentation de la pièce : cavalier

Remarque(s) : -

Compilateur : jdk1.8.0_221

--------------------------- */
package engine.pieces;

import chess.PieceType;
import chess.PlayerColor;
import engine.Case;

public class Dame extends PieceDeplacementStandard{
    public Dame(PlayerColor color) {
        super(color);
        pieceType = PieceType.QUEEN;
        distanceDeplacementMax = 10;
        directionType = DirectionType.TOUS;
    }

    @Override
    public MouvementType mouvementPossible(Case src, Case dest) {
        if (super.mouvementPossible(src,dest) == MouvementType.NON_VALIDE){
            return MouvementType.NON_VALIDE;
        }

        return super.mouvementPossible(src, dest);
    }

    @Override
    public Piece clone() {
        return new Dame(this.color);
    }

    @Override
    public String toString() {
        return "Dame";
    }
}

```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : Fou.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Implémentation de la pièce : fou

Remarque(s) : -

Compilateur : jdk1.8.0_221

--------------------------- */
package engine.pieces;

import chess.PieceType;
import chess.PlayerColor;
import engine.Case;

public class Fou extends PieceDeplacementStandard{
    public Fou(PlayerColor color) {
        super(color);
        pieceType = PieceType.BISHOP;
        distanceDeplacementMax = 10;
        directionType = DirectionType.DIAGONALE;
    }

    @Override
    public MouvementType mouvementPossible(Case src, Case dest) {
        if (super.mouvementPossible(src,dest) == MouvementType.NON_VALIDE){
            return MouvementType.NON_VALIDE;
        }

        return super.mouvementPossible(src, dest);
    }


    @Override
    public Piece clone() {
        return new Fou(this.color);
    }

    @Override
    public String toString() {
        return "Fou";
    }
}

```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : Pion.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Implémentation de la pièce : pion

Remarque(s) : Fonction mouvementPossible spécifique

Compilateur : jdk1.8.0_221

--------------------------- */
package engine.pieces;

import chess.PieceType;
import chess.PlayerColor;
import engine.Case;

public class Pion extends PiecePremierDeplacement{
    public Pion(PlayerColor color) {
        super(color);
        pieceType = PieceType.PAWN;
        distanceDeplacementMax = 2;
        directionType = DirectionType.TOUS;

        if(color == PlayerColor.BLACK)
            distanceDeplacementMax *= (-1);
    }

    @Override
    public MouvementType mouvementPossible(Case src, Case dest) {

        if (super.mouvementPossible(src,dest) == MouvementType.NON_VALIDE){
            return MouvementType.NON_VALIDE;
        }

        if(!dest.estVide() && dest.getX() == src.getX())
            return MouvementType.NON_VALIDE;

        //On promeut un pion s'il se trouve sur la première ou dernière ligne de l'échiquier
        if ((dest.getY() == 0 || dest.getY() == 7) && (dest.getY() == src.getY() +
        distanceDeplacementMax / 2)){ // 0 et 7 représentent la première et derrière ligne de
        l'échiquier
            return MouvementType.PROMOTION;
        }

        // Deplacement standard de 1 en avant
        if(src.getX() == dest.getX() && src.getY() + distanceDeplacementMax / 2 == dest.getY()){
            return MouvementType.CLASSIQUE;
        }

        // Premier déplacement de 2 en avant
        if(src.getX() == dest.getX() && this.premierDeplacement && src.getY() +
        distanceDeplacementMax == dest.getY()){
            return MouvementType.DOUBLE;
        }

        // Deplacement en diagonale en avant de 1, pour une prise
        if((src.getX() == dest.getX()+1 || src.getX() == dest.getX()-1) && src.getY() +
        distanceDeplacementMax / 2 == dest.getY() && dest.aUnePiece()){
            return MouvementType.CLASSIQUE;
        }

        // Deplacement en diagonale en avant de 1, pour une prise
        if((src.getX() == dest.getX()+1 || src.getX() == dest.getX()-1) && src.getY() +
        distanceDeplacementMax / 2 == dest.getY() && dest.aUnPionFantome()){
            return MouvementType.EN_PASSANT;
        }

        return MouvementType.NON_VALIDE;
    }

    @Override
    public Piece clone() {
        Pion p = new Pion(this.color);
        p.premierDeplacement = this.premierDeplacement;
```

```
73              return p;
74          }
75
76          @Override
77          public String toString() {
78              return "Pion";
79          }
80      }
81
```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : Roi.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Implémentation de la pièce : roi

Remarque(s) : -

Compilateur : jdk1.8.0_221

--------------------------- */
package engine.pieces;

import chess.PieceType;
import chess.PlayerColor;
import engine.Case;

public class Roi extends PiecePremierDeplacement {
    public Roi(PlayerColor color) {
        super(color);
        pieceType = PieceType.KING;
        directionType = DirectionType.TOUS;
        distanceDeplacementMax = 2;
    }

    @Override
    public MouvementType mouvementPossible(Case src, Case dest) {
        MouvementType mouvementType = super.mouvementPossible(src, dest);

        if (mouvementType == MouvementType.NON_VALIDE) {
            return MouvementType.NON_VALIDE;
        }

        int deltaX = Math.abs(src.getX() - dest.getX());
        int deltaY = Math.abs(src.getY() - dest.getY());

        if (mouvementType == MouvementType.CLASSIQUE && deltaX <=1 && deltaY <= 1) {
            premierDeplacement = false;
            return MouvementType.CLASSIQUE;
        }

        if (premierDeplacement) {
            if (color == PlayerColor.WHITE) {
                if (dest.getX() == 6 && dest.getY() == 0) {
                    return MouvementType.PETIT_ROQUE;
                }
                if (dest.getX() == 2 && dest.getY() == 0) {
                    return MouvementType.GRAND_ROQUE;
                }
            }
            if (color == PlayerColor.BLACK) {
                if (dest.getX() == 6 && dest.getY() == 7) {
                    return MouvementType.PETIT_ROQUE;
                }
                if (dest.getX() == 2 && dest.getY() == 7) {
                    return MouvementType.GRAND_ROQUE;
                }
            }
        }
        return MouvementType.NON_VALIDE;
    }

    @Override
    public Piece clone() {
        Roi p = new Roi(this.color);
        p.premierDeplacement = this.premierDeplacement;
        return p;
    }
    @Override
    public String toString() {
        return "Roi";
    }
}

```

```java
/* ---------------------------
Laboratoire : Labo 08: Jeu d'échecs
Fichier : Tour.java
Auteur(s) : Romano Malo et Enzo Allemano
Date : 12.01.2022

But : Implémentation de la pièce : tour

Remarque(s) : -

Compilateur : jdk1.8.0_221

--------------------------- */
package engine.pieces;

import chess.PieceType;
import chess.PlayerColor;
import engine.Case;

public class Tour extends PiecePremierDeplacement {
    public Tour(PlayerColor color) {
        super(color);
        pieceType = PieceType.ROOK;
        directionType = DirectionType.DROIT;
        distanceDeplacementMax = 10;
    }

    @Override
    public MouvementType mouvementPossible(Case src, Case dest) {
        if (super.mouvementPossible(src,dest) == MouvementType.NON_VALIDE){
            return MouvementType.NON_VALIDE;
        }
        return super.mouvementPossible(src, dest);
    }

    @Override
    public Piece clone() {
        Tour p = new Tour(this.color);
        p.premierDeplacement = this.premierDeplacement;
        return p;
    }

    @Override
    public String toString() {
        return "Tour";
    }
}

```

```java
package chess.views.console;

import chess.ChessController;
import chess.PieceType;
import chess.PlayerColor;
import chess.assets.ConsoleAssets;
import chess.views.BaseView;
import chess.views.DrawableResource;

import java.util.Scanner;
import java.util.regex.Pattern;

public class ConsoleView extends BaseView<String> {

  private static class StringResource implements DrawableResource<String> {
    private String value;

    private StringResource(String value, PlayerColor color) {
      this.value = (color == PlayerColor.BLACK ? "\u001B[31m" : "") + value + "\u001B[30m";
    }

    @Override
    public String getResource() {
      return value;
    }
  };

  public static DrawableResource<String> createResource(String value, PlayerColor color) {
    return new StringResource(value, color);
  }

  private final static Scanner scanner = new Scanner(System.in);
  private final static Pattern movementPattern = Pattern.compile("[a-h][1-8][a-h][1-8]|exit");
  private final static String EMPTY_CELL = " ";
  private final static String UNKNOWN_CELL = "?";

  private String checkMessage = "";
  private String[][] buffer;

  public ConsoleView(ChessController controller) {
    super(controller);
    ConsoleAssets.loadAssets(this);
    initialize();
    clearView();
  }

  @Override
  public void startView() {
    System.out.println("Chess game...");
    controller.newGame();
    while (true) { // TODO: ajouter un exit (comportement repris de la V1)
      printBoard();
      System.out.println("Type \"exit\" to exit the game");
      askMovement();
    }
  }

  @Override
  public void removePiece(int x, int y) {
    buffer[x][y] = EMPTY_CELL;
  }

  @Override
  public void putPiece(PieceType type, PlayerColor color, int x, int y) {
    buffer[x][y] = loadResourceFor(type, color, UNKNOWN_CELL);
  }

  @Override
  public void displayMessage(String msg) {
    System.out.println(msg);
  }

  @Override
  public <T extends UserChoice> T askUser(String title, String question, T... possibilities) {
    T result = possibilities.length > 0 ? possibilities[0] : null;
    if (possibilities.length > 1) {
      int i = 0;
```

```
78          for (T choice : possibilities) {
79            System.out.println(i + ". " + choice.textValue());
80          }
81
82          int userChoice;
83          do {
84            userChoice = -1;
85            System.out.println("Enter the desired number > ");
86
87            try {
88              userChoice = Integer.parseInt(scanner.nextLine());
89              if (userChoice > 0 && userChoice < possibilities.length)
90                result = possibilities[userChoice];
91              else
92                userChoice = -1;
93            }
94            catch (NumberFormatException e) { // nothing
95            }
96
97            if (userChoice < 0)
98              System.out.println("Error. Choose a value between 0 and " + (possibilities.length -
                1));
99
100         }
101         while (userChoice < 0);
102       }
103       return result;
104     }
105
106     private void initialize() {
107       buffer = new String[8][8];
108     }
109
110     private void clearView() {
111       for (int i = 0; i < buffer.length; ++i) {
112         for (int j = 0; j < buffer[i].length; ++j) {
113           removePiece(i, j);
114         }
115       }
116     }
117
118
119
120     private void printBoard() {
121       for (int y = 7; y >= 0; --y) {
122         System.out.print(y + 1 + " |");
123         for (int x = 0; x < 8; ++x) {
124           System.out.print(buffer[x][y]);
125           System.out.print(" ");
126         }
127         System.out.print("\n");
128       }
129       System.out.println("--------------------");
130       System.out.println("   A B C D E F G H ");
131     }
132
133     private static int charCoordinateToIndex(char c) {
134       assert (c >= 'a' && c < 'i');
135       return c - 'a';
136     }
137
138     private static String askPattern(Pattern pattern, String text) {
139       String in = null;
140       while (in == null) {
141         System.out.println(text);
142         in = scanner.findInLine(pattern);
143         scanner.nextLine();//clean buffer
144       }
145       return in;
146
147     }
148
149     private static int intCoordinateToIndex(char c) {
150       assert (c >= '1' && c <= '9');
151       return c - '1';
152     }
153
```

```
154     private void askMovement() {
155       boolean ok = false;
156       while (!ok) {
157         String in = askPattern(movementPattern, "Next move?");
158         System.out.println(in);
159         if(in.equals("exit")){
160           System.out.println("Program ended");
161           System.exit(0);
162         }
163         ok = controller.move(charCoordinateToIndex(in.charAt(0)),
            intCoordinateToIndex(in.charAt(1)),
164             charCoordinateToIndex(in.charAt(2)), intCoordinateToIndex(in.charAt(3)));
165
166         if (!ok) {
167           System.out.println("Invalid move");
168           printBoard();
169         }
170       }
171     }
172
173   }
174
```