PROJET BPO2
ÉCHECS

25 MAI

AKHAYAR_Mahir_104
HADDIOUI_Sélim_104
LUNION_Samuel_104

Université de Paris

Table des matières

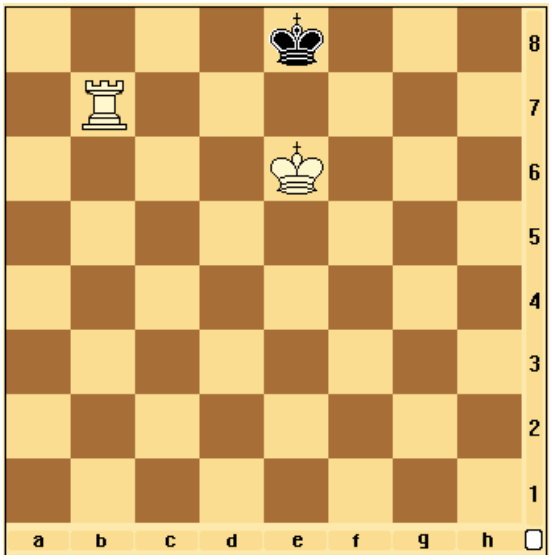
Notre objectif durant ce projet était de concevoir un programme respectant au mieux les notions étudiés en TD et TP. Nous avons réalisé toutes les consignes et chacune des fonctionnalités du programme marche correctement.

I. Présentation de l'application

a. Fonctionnement

Notre application a été conçu dans un but de divertissement. Celle-ci permet à des individus de jouer une finale d'échecs avec comme pièces utilisables le Roi et la Tour.

Le programme démarre avec le choix du type de partie à savoir Joueur contre Joueur, IA contre Joueur ou IA contre IA. Ensuite, une phrase indiquant comment abandonner ou proposer match nul s'affiche. Puis le jeu débute dans la même situation que celle illustrés ci-dessous durant le tour du joueur blanc. Les joueurs blancs sont représentés par une majuscule, les joueurs noirs par une minuscule. Avant chaque tour un damier s'affiche ainsi qu'un message indiquant quel joueur doit jouer si celui est Humain.



L'IA cherche un nombre définit de coup possible puis, à partir d'un nombre aléatoire choisis le X^e coup qu'il a trouvé.

Il est possible de déclarer forfait en saisissant STOP durant son tour mais aussi de proposer un match nul en saisissant NUL, dans le dernier cas, si le joueur en face est humain il devra accepter en écrivant à son tour lui aussi NUL. Si la proposition est refusée le jeu reprend au tour du joueur ayant fait la proposition.

Pour déplacer une pièce il faut indiquer les coordonnées sous le format suivant b7b8, ce coup aura pour conséquence de déplacer la tour blanche d'une case vers le haut et de mettre le roi noir en échec et mat.

Pour finir la partie différents scénario sont possible comme : le matériel insuffisant (roi contre roi), l'échec et mat, le pat, l'abandon ou la proposition de match nul.

b. Diagramme d'architecture

(voir fichier png fournit en annexe).

II. Listing test unitaire.

a) CoordTest.java

```
package test.coord;

import coord.Coord;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;

class CoordTest {
private final static int TAILLE = 8;
    @Test
    void coordCorrecte() {
        Coord test = new Coord(8,9);
        assertFalse(Coord.coordCorrecte(test, TAILLE));
        test = new Coord(-4,-7);
        assertFalse(Coord.coordCorrecte(test, TAILLE));
        test = new Coord(7,4);
        assertTrue(Coord.coordCorrecte(test, TAILLE));
    }
}
```

b) EchiquierTest.java

```
package test.echiquier;

import coord.Coord;
import echiquier.Echiquier;
import equipe.Team;
import org.junit.jupiter.api.Test;
import piece.Roi;
import piece.Tour;

import static org.junit.jupiter.api.Assertions.*;

class EchiquierTest {
@Test
    void Echiquier() {
        Echiquier plateau = new Echiquier();
        Roi test = new Roi(Team.blanche);
        Coord coordroi = new Coord(0,0);
        Coord coordtour = new Coord(1,1);
        plateau.setPiece(coordroi,  test);
        assertEquals(plateau.getCoord(test).getColonne(), coordroi.getColonne());
        assertEquals(plateau.getCoord(test).getRangee(), coordroi.getRangee());
        assertEquals(plateau.getPiece(coordroi), test);
        Tour adv = new Tour(Team.noir);
        plateau.setPiece(coordtour,adv);
        assertTrue(plateau.eatPiece(coordtour, coordroi, test));
        plateau.movePiece(coordroi, coordtour, test);
        assertFalse(plateau.eatPiece(coordroi, coordtour, adv));
        plateau.movePiece(coordroi, coordtour, test);
        assertEquals(plateau.getPiece(coordroi), test);
        assertFalse(plateau.DeplacementTest(new Coord(7,7), test));
        assertTrue(plateau.DeplacementTest(new Coord(0,1), test));
        assertTrue(plateau.Deplacement(new Coord(7, 7), test));
    }
@Test
void testAffichage() {
    Echiquier test = new Echiquier();
    String resAttendu =
            "    a   b   c   d   e   f   g   h\n" +
            "  --- --- --- --- --- --- --- ---\n" +
            "8 |   |   |   |   |   |   |   | 8\n" +
            "  --- --- --- --- --- --- --- ---\n" +
            "7 |   |   |   |   |   |   |   | 7\n" +
            "  --- --- --- --- --- --- --- ---\n" +
            "6 |   |   |   |   |   |   |   | 6\n" +
            "  --- --- --- --- --- --- --- ---\n" +
            "5 |   |   |   |   |   |   |   | 5\n" +
            "  --- --- --- --- --- --- --- ---\n" +
            "4 |   |   |   |   |   |   |   | 4\n" +
            "  --- --- --- --- --- --- --- ---\n" +
            "3 |   |   |   |   |   |   |   | 3\n" +
            "  --- --- --- --- --- --- --- ---\n" +
            "2 |   |   |   |   |   |   |   | 2\n" +
            "  --- --- --- --- --- --- --- ---\n" +
            "1 |   |   |   |   |   |   |   | 1\n" +
            "  --- --- --- --- --- --- --- ---\n" +
            "    a   b   c   d   e   f   g   h";
    System.out.println(resAttendu);
    assertFalse(test.toString().equals(resAttendu));
}
}
```

c) HumainTest.java

```java
package test.joueur;

import coord.Coord;
import echiquier.Echiquier;
import equipe.Team;
import joueur.Humain;
import org.junit.jupiter.api.Test;
import piece.FabriquePiece;
import piece.PiecesEnum;

import static org.junit.jupiter.api.Assertions.*;

class HumainTest {
    @Test
    void test() {
        Humain blanc = new Humain(Team.blanche);
        Humain noir = new Humain(Team.noir);
        Echiquier plateau = new Echiquier();
        plateau.setPiece(new Coord(0,0), FabriquePiece.fab(PiecesEnum.Roi, Team.blanche));
        plateau.setPiece(new Coord(7,7), FabriquePiece.fab(PiecesEnum.Roi, Team.noir));
        assertFalse(blanc.coupLegal(plateau, noir, new Coord(7,7), new Coord(7,6)));
        assertTrue(noir.coupLegal(plateau, noir, new Coord(7,7), new Coord(7,6)));
        assertTrue(blanc.coupLegal(plateau, noir, new Coord(0,0), new Coord(0,1)));
        assertFalse(noir.coupLegal(plateau, noir, new Coord(0,0), new Coord(0,1)));
    }
}
```

d) IATest.java

```java
package test.joueur;

import coord.Coord;
import echiquier.Echiquier;
import equipe.Team;
import joueur.IA;
import org.junit.jupiter.api.Test;
import piece.FabriquePiece;
import piece.PiecesEnum;

import static org.junit.jupiter.api.Assertions.*;

class IATest {

    @Test
    void test1() {
        IA test = new IA(Team.blanche);
        IA adv = new IA(Team.noir);
        Echiquier plateau = new Echiquier();
        assertTrue(test.coupLegal(plateau, adv, new Coord(0, 0), new Coord(17,98)));
    }
```

```
@Test
    void getCoup() {
        IA test = new IA(Team.blanche);
        IA notest = new IA(Team.noir);
        Echiquier plateau = new Echiquier();
        plateau.setPiece(new Coord(0,0), FabriquePiece.fab(PiecesEnum.Roi, Team.blanche));
        plateau.setPiece(new Coord(7,0), FabriquePiece.fab(PiecesEnum.Roi, Team.noir));
        plateau.setPiece(new Coord(0,0), FabriquePiece.fab(PiecesEnum.Tour, Team.blanche));
        assertTrue(test.jouer(plateau, notest));
        assertTrue(test.jouer(plateau, notest));
        assertTrue(test.jouer(plateau, notest));
        assertTrue(test.jouer(plateau, notest));
    }

    @Test
    void wantMatchNul() {
        IA test = new IA(Team.blanche);
        IA adv = new IA(Team.noir);
        assertTrue(test.wantMatchNul(adv));
    }
}
```

e) FabriquePieceTest.java

```
package test.piece;

import echiquier.IPiece;
import equipe.Team;
import org.junit.jupiter.api.Test;
import piece.FabriquePiece;
import piece.PiecesEnum;

import static org.junit.jupiter.api.Assertions.assertTrue;

class FabriquePieceTest {

    @Test
    void fab() {
        IPiece test = FabriquePiece.fab(PiecesEnum.Vide, Team.blanche);
        assertTrue(test.getType().equals(PiecesEnum.Vide));
        assertTrue(test.getTeam().equals(Team.autre));;
        test = FabriquePiece.fab(PiecesEnum.Vide, Team.noir);
        assertTrue(test.getType().equals(PiecesEnum.Vide));
        assertTrue(test.getTeam().equals(Team.autre));
        test = FabriquePiece.fab(PiecesEnum.Roi, Team.blanche);
        assertTrue(test.getType().equals(PiecesEnum.Roi));
        assertTrue(test.getTeam().isWhite());
        test = FabriquePiece.fab(PiecesEnum.Tour, Team.blanche);
        assertTrue(test.getType().equals(PiecesEnum.Tour));
        assertTrue(test.getTeam().isWhite());
        test = FabriquePiece.fab(PiecesEnum.Roi, Team.noir);
        assertTrue(test.getType().equals(PiecesEnum.Roi));
        assertTrue(test.getTeam().isBlack());
        test = FabriquePiece.fab(PiecesEnum.Tour, Team.noir);
        assertTrue(test.getType().equals(PiecesEnum.Tour));
        assertTrue(test.getTeam().isBlack());

    }
}
```

f) PieceVideTest.java

```
package test.piece;

import coord.Coord;
import echiquier.Echiquier;
import echiquier.IPiece;
import org.junit.jupiter.api.Test;
import piece.PieceVide;
import piece.PiecesEnum;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;

class PieceVideTest {
    private final static String NOM = " ";

    @Test
    void testToString() {
        PieceVide pV = new PieceVide();
         assertEquals(pV.toString(), NOM);
    }

    @Test
    void getType() {
        PieceVide pV = new PieceVide();
        assertEquals(pV.getType(), PiecesEnum.Vide );
    }

    @Test
    void deplacementValide_plageDeplacementInvalide_obstacle() {
        Echiquier plateau = new Echiquier();
        IPiece pV = plateau.getPiece(new Coord(3,3));
        PieceVide pv = (PieceVide) pV;
        assertFalse(pv.deplacementValide(plateau, new Coord(4,8)));
    }

}
```

g) RoiTest.java

```
package test.piece;

import coord.Coord;
import echiquier.Echiquier;
import equipe.Team;
import org.junit.jupiter.api.Test;
import piece.PiecesEnum;
import piece.Roi;
import piece.Tour;

import static org.junit.jupiter.api.Assertions.*;

class RoiTest {
    private final static String NOM = "R";

    @Test
    void testToString() {
        Roi B = new Roi(Team.blanche);
        Roi N = new Roi(Team.noir);
        assertEquals(B.toString(),NOM);
        assertEquals(N.toString(),NOM.toLowerCase());
    }

    @Test
    void obstacle() {
        Roi Roi = new Roi(Team.noir);
        Echiquier plateau = new Echiquier();
        assertFalse(Roi.obstacle(plateau, new Coord(2,2)));
    }

    @Test
    void getType() {
        Roi roi = new Roi(Team.noir);
        assertEquals(roi.getType(), PiecesEnum.Roi);
    }
    @Test
    void deplacementValide_plageDeplacementInvalide_obstacle() {
        Echiquier plateau = new Echiquier();
        Roi roi = new Roi(Team.blanche);
        Tour advt2 = new Tour(Team.noir);
        Tour t3 = new Tour(Team.blanche);
        Tour t4 = new Tour(Team.blanche);
        Tour advt5 = new Tour(Team.noir);
        Tour t6 = new Tour(Team.blanche);
        plateau.setPiece(new Coord(3,3),roi);
        plateau.setPiece(new Coord(3,2), t3); // gauche
        plateau.setPiece(new Coord(3,4), advt2); // haut
        plateau.setPiece(new Coord(4,3), t4); // bas
        plateau.setPiece(new Coord(5,1), advt5); // haut
        plateau.setPiece(new Coord(1,5), t6); // bas
//      case vide autorisée
        assertTrue(roi.deplacementValide(plateau, new Coord(2,2)));
        assertTrue(roi.deplacementValide(plateau, new Coord(2,4)));
        assertTrue(roi.deplacementValide(plateau, new Coord(4,4)));
        assertTrue(roi.deplacementValide(plateau, new Coord(4,2)));
```

```
//        case vide hors champ
        assertFalse(roi.deplacementValide(plateau, new Coord(1,1)));
        assertFalse(roi.deplacementValide(plateau, new Coord(5,5)));
//      mange une piece
        assertTrue(roi.deplacementValide(plateau, new Coord(3,4))); // mange adv
        assertFalse(roi.deplacementValide(plateau, new Coord(4,3))); // mange ami
        assertFalse(roi.deplacementValide(plateau, new Coord(3,2))); // mange ami
        assertFalse(roi.deplacementValide(plateau, new Coord(5,5))); // tour adv hors champ
        assertFalse(roi.deplacementValide(plateau, new Coord(1,5))); // tour ami hors champ
    }
}
```

h) TourTest.java

```
package test.piece;

import coord.Coord;
import echiquier.Echiquier;
import equipe.Team;
import org.junit.jupiter.api.Test;
import piece.PiecesEnum;
import piece.Tour;

import java.util.Locale;

import static org.junit.jupiter.api.Assertions.*;

class TourTest {
    private final static String NOM = "T";

    @Test
    void testToString() {
        Tour blanche = new Tour(Team.blanche);
        assertEquals(blanche.toString(), NOM);
        Tour noire = new Tour(Team.noir);
        assertEquals(noire.toString(), NOM.toLowerCase(Locale.ROOT));
    }

    @Test
    void getType() {
        Tour tour = new Tour(Team.blanche);
        assertEquals(tour.getType(), PiecesEnum.Tour);
    }

    @Test
    void deplacementValide_plageDeplacementInvalide_obstacle() {
        Echiquier plateau = new Echiquier();
        Tour tour = new Tour(Team.blanche);
        Tour t1 = new Tour(Team.noir);
        Tour t2 = new Tour(Team.noir);
        Tour t3 = new Tour(Team.noir);
        Tour t4 = new Tour(Team.noir);
        plateau.setPiece(new Coord(3,3), tour);
        plateau.setPiece(new Coord(3,1), t3); // gauche
        plateau.setPiece(new Coord(3,6), t1); // droite
        plateau.setPiece(new Coord(6,3), t2); // haut
        plateau.setPiece(new Coord(1,3), t4); // bas
```

```
//      Droite
        assertTrue(tour.deplacementValide(plateau, new Coord(3,5)));
        assertFalse(tour.deplacementValide(plateau, new Coord(3,7))); // obstacle
//      Gauche
        assertTrue(tour.deplacementValide(plateau, new Coord(3,2)));
        assertFalse(tour.deplacementValide(plateau, new Coord(3,0))); // obstacle
//      Bas
        assertTrue(tour.deplacementValide(plateau, new Coord(2,3)));
        assertFalse(tour.deplacementValide(plateau, new Coord(0,3))); // obstacle
//      Haut
        assertTrue(tour.deplacementValide(plateau, new Coord(5,3)));
        assertFalse(tour.deplacementValide(plateau, new Coord(7,3))); // obstacle
//      Haut droite
        assertFalse(tour.deplacementValide(plateau, new Coord(0,7)));
//      Haut gauche
        assertFalse(tour.deplacementValide(plateau, new Coord(0,0)));
//      Bas droite
        assertFalse(tour.deplacementValide(plateau, new Coord(7,7)));
//      Bas gauche
        assertFalse(tour.deplacementValide(plateau, new Coord(7,0)));
//      Mange adv
        assertTrue(tour.deplacementValide(plateau, new Coord(3,1)));
    }
}
```

III. Code source

Deplacement.java

```
package coord;

public class Coord {
    private int numR, numC;
    public Coord(int rangee, int colonne) {
        this.numR = rangee;
        this.numC = colonne;
    }
    /**
     * @return la rangee
     */
    public int getRangee() {
        return numR;
    }
    /**
     * @return la colonne
     */
    public int getColonne() {
        return numC;
    }
    /**
     * @param c coordonnee a tester
     * @param taille de l'echiquier
     * @return vrai si les coordonnee sont situes dans l'echiquier
     */
    public static boolean coordCorrecte(Coord c, int taille) {
        return 0 <= c.getColonne() && c.getColonne() < taille &&
                0 <= c.getRangee() && c.getRangee() < taille;
    }
}
```

Coord.java

```
package coord;

public class Coord {
    private int numR, numC;
    public Coord(int rangee, int colonne) {
        this.numR = rangee;
        this.numC = colonne;
    }
    /**
     * @return la rangee
     */
    public int getRangee() {
        return numR;
    }
    /**
     * @return la colonne
     */
    public int getColonne() {
        return numC;
    }
    /**
     * @param c coordonnee a tester
     * @param taille de l'echiquier
     * @return vrai si les coordonnee sont situes dans l'echiquier
     */
    public static boolean coordCorrecte(Coord c, int taille) {
        return 0 <= c.getColonne() && c.getColonne() < taille &&
                0 <= c.getRangee() && c.getRangee() < taille;
    }
}
```

Echiquier.java

```
package echiquier;

import coord.Coord;
import equipe.Team;
import piece.FabriquePiece;
import piece.PiecesEnum;

import static org.junit.jupiter.api.Assertions.assertTrue;

public class Echiquier {
    private IPiece[][] echiquier = new IPiece[taille][taille];
    private final static int taille = 8;

    public Echiquier() {
        for (int row = 0; row < taille; row++) {
            for (int column = 0; column < taille; column++) {
                  echiquier[row][column] = FabriquePiece.fab(PiecesEnum.Vide,
Team.autre);
            }
        }
    }

    /**
     * initialise la position de depart des pieces dans l'echiquier
     */
    public void init(){
        this.setPiece(new Coord(0,4), FabriquePiece.fab(PiecesEnum.Roi,
Team.noir));
        this.setPiece(new Coord(1,1), FabriquePiece.fab(PiecesEnum.Tour,
Team.blanche));
        this.setPiece(new Coord(2,4), FabriquePiece.fab(PiecesEnum.Roi,
Team.blanche));
    }

    /**
     * @param destination de la piece
     * @param toPut piece a deplacer
     * Si les coordonnees sont correctes, place la piece a l'emplacement indique
     */
    public void setPiece(Coord destination, IPiece toPut) {
        assert(Coord.coordCorrecte(destination, this.getTaille()));
        this.echiquier[destination.getRangee()][destination.getColonne()] = toPut;
// on remplace l'ancienne piece dans l'echiquier par la nouvelle
    }

    /**
     * @return taille de l'echiquier
     * obtient la taille de l'echiquier
     */
    public int getTaille() {
        return taille;
    }

    /**
     * @param p piece a trouver
     * @return les coordonnes de la piece dans l'echiquier
     * cherche une piece dans l'echiquier et renvoi ses coordonnees
     */
    public Coord getCoord(IPiece p) {
        for (int row = 0; row < taille; row++) {
```

```
            for (int column = 0; column < taille; column++) {
                if (echiquier[row][column].equals(p))
                    return new Coord(row, column);
            }
        }
        return new Coord(taille, taille);
    }

    /**
     * @param c coordonnee de la piece
     * @return la piece
     * cherche aux coordonnees dans l'echiquier une piece et la retourne
     */
    public IPiece getPiece(Coord c) {
        assertTrue(Coord.coordCorrecte(c, this.getTaille()));
        return this.echiquier[c.getRangee()][c.getColonne()];
    }

    /**
     * @param destination coord de la piece a manger
     * @param depart coord de la piece qui mange
     * @param toMove piece qui mange
     * @return vrai si l'on mange une piece autre que le roi sinon faux
     * creer une piece vide a l'emplacement de la piece qui mange puis remplace la
piece qui se fait manger par celle qui mange
     */
    public boolean eatPiece(Coord destination, Coord depart, IPiece toMove){
            if(!getPiece(destination).getType().equals(PiecesEnum.Roi)) {
                setPiece(depart, FabriquePiece.fab(PiecesEnum.Vide, Team.autre));//
setPiece(destination, Piece a mettre);
                setPiece(destination, toMove);
                return true;
            }
            return false;
    }

    /**
     * @param destination coord vers ou la piece va etre deplacer
     * @param depart coord de depart de la piece
     * @param toMove piece a deplacer
     * deplace une piece vers une case vide, permet de ne pas recreer de piece vide
a chaque mouvement
     */
    public void movePiece(Coord destination, Coord depart, IPiece toMove){ // pas
besoin d'instancier une nouvelle piece vide a chaque mouvement
        setPiece(depart, this.getPiece(destination));
        setPiece(destination, toMove);
    }

    /**
     * @param destination coord d'arrive a tester
     * @param toMove piece a deplacer
     * @return vrai si le coup est legal, faux si il ne l'est pas
     * verifie que l'on ne deplace pas une piece vide et que les deplacements sont
legaux
     */
    public boolean DeplacementTest(Coord destination, IPiece toMove){
        Coord depart = getCoord(toMove);
        if(!toMove.getType().equals(PiecesEnum.Vide) &&
toMove.deplacementValide(this, destination) )
            return Coord.coordCorrecte(depart, this.getTaille());
        return false;
    }
```

```
    /**
     * @param destination coord d'arrive a tester
     * @param toMove piece a deplacer
     * @return vrai si le deplacement est legal
     * effectue le deplacement a la seul condition que la piece ne tente pas de
manger un roi
     */
    public boolean Deplacement(Coord destination, IPiece toMove){
        Coord depart = getCoord(toMove);
        if(getPiece(destination).getType().equals(PiecesEnum.Vide)){
            movePiece(destination, depart, toMove);
        }
        else {
            return eatPiece(destination, depart, toMove);
        }
        return true;
    }

    public String toString(){
        int cptRow = 8;
        char col = 'a';
        StringBuilder column = new StringBuilder();
        StringBuilder dash = new StringBuilder();
        StringBuilder res = new StringBuilder();
        column.append(" ");
        dash.append("  ");
        for(int i = 0; i < taille; i++){
            column.append("   ").append(col);
            dash.append(" ---");
            col++;
        }
        column.append(System.lineSeparator());
        dash.append(System.lineSeparator());
        res.append(column).append(dash);
        for(IPiece[] row : echiquier ) {
            res.append(cptRow).append(" | ");
            for (IPiece box : row) {
                res.append(box.toString()).append(" | ");
            }
            res.append(cptRow).append(System.lineSeparator()).append(dash);
            cptRow--;
        }
        res.append(column);
        return res.toString();
    }
}
```

IPiece.java

```
package echiquier;

import coord.Coord;
import equipe.Team;
import piece.PiecesEnum;

public interface IPiece {

    /**
     * @return l'equipe de la piece
     */
    Team getTeam();

    /**
     * @param e l'echiquier sur lequel la piece se trouve
     * @param c les coordonnees ou la piece veut aller
     * @return vrai si le coup est legal sinon faux
     * Permet de savoir les deplacements autorisee d'une piece
     */
    boolean deplacementValide(Echiquier e, Coord c);

    /**
     * @param plateau l'echiquier sur lequel la piece se trouve
     * @param adversaire adversaire de la piece
     * @return vrai si une piece adverse peut manger la piece allie sinon faux
     * permet de savoir si le roi est en echec et peut servir a creer un IA
intelligent deplaçant les pieces importantes menacees
     */
    boolean estAttaque(Echiquier plateau, Team adversaire); // offrir cette methode
a toutes les classes permettrait a l'avenir de mettre en surbrillance les pions en
danger dans un mode de jeu facile

    /**
     * @return renvoi le type de la piece
     */
    PiecesEnum getType();

}
```

Team.java

```
package equipe;

public enum Team {
    blanche, noir, autre;

    /**
     * @return vrai si l'equipe est blanche sinon faux
     */
    public boolean isWhite(){ return this == blanche; }

    /**
     * @return vrai si l'equipe est noire sinon faux
     */
    public boolean isBlack(){ return this == noir; }
}
```

FabriqueIJoueur.java

```
package joueur;

import jeu.IFabriqueIJoueur;
import jeu.IJoueur;
import equipe.Team;

public class FabriqueIJoueur implements IFabriqueIJoueur {
    /**
     * @param choix  des joueurs a initialiser
     * @param joueur compteur du joueur inialise ( xieme joueur )
     * @return
     * initialise le joueur
     */
    public IJoueur initJoueur(int choix, int joueur) {
        switch (choix) {
            case 1:
                return (joueur == 0) ? new Humain(Team.blanche) : new
Humain(Team.noir);
            case 2:
                return (joueur == 0) ? new Humain(Team.blanche) : new
IA(Team.noir);
            case 3:
                return (joueur == 0) ? new IA(Team.blanche) : new IA(Team.noir);
        }
        return null;
    }

}
```

Humain.java

```
package joueur;

import jeu.IJoueur;
import coord.Coord;
import echiquier.Echiquier;
import equipe.Team;

import java.util.Scanner;

public class Humain extends Joueur{
    private final static String NUL = "NUL", ABANDON = "STOP";
    private final static char debutAlphabet = 'a';

    public Humain(Team team){
       super(team);
    }

    /**
     * @param plateau Echiquier concerne par le coup a jouer
     * @param adv     adversaire du joueur concerne
     * @return la chaine de caractere contenant les coords de depart de la piece et
d'arrive
     */
    @Override
    public String getCoup(Echiquier plateau, IJoueur adv) {
        System.out.println("Tour joueur : " + this);
        String coup = getScanner();
        StringBuilder res = new StringBuilder();
        if(coup.equalsIgnoreCase(ABANDON)) {
            this.setPerdant();
            return null;
        }
        if(coup.equalsIgnoreCase(NUL)) {
            if(adv.wantMatchNul(this))
                return null;
        }
        if(verifSyntaxe(coup)) {
            res.append(getStringCoord(coup.substring(char1,char2), plateau));
            res.append(getStringCoord(coup.substring(char2,tailleSaisie),
plateau));
            return res.toString();
        }
        return null;
    }

    /**
     * @param plateau Echiquier concerne par le test
     * @param adv     adversaire du joueur
     * @param depart  coord de depart de la piece
     * @param arrive  coord d'arrive de la pice
     * @return true si le coup n'est pas l'egal / false si le coup est legal
     */
    @Override
    public boolean coupLegal(Echiquier plateau, IJoueur adv, Coord depart, Coord
arrive){
        if(!plateau.getPiece(depart).getTeam().equals(this.getTeam()))
            return false;
        if(!plateau.DeplacementTest(arrive, plateau.getPiece(depart)))
            return false;
        return coupSimuleValide(plateau, depart, arrive, adv);
```

```
    }

    /**
     * @param adv adversaire du joueur
     * @return vrai si le joueur adverse accepte le match nul sinon faux
     */
    @Override
    public boolean wantMatchNul(IJoueur adv) {
        System.out.println("Joueur " + this + " propose de finir sur un match
nul.");
        System.out.println("Pour accepter saisissez NUL, pour refuser saisissez un
mot different");
        System.out.println("Tour joueur : " + adv);
        Scanner decision = new Scanner(System.in);
        String res = decision.next();
        if(res.equals(NUL)){
            this.setPat();
        }
        return res.equals(NUL);
    }

    /**
     * @param coupJoue
     * @return false si la chaine de charactere ne respecte pas la syntaxe
predefinie / true si la chaine de charactere respecte la syntaxe
     */
    private boolean verifSyntaxe(String coupJoue) {
        if(coupJoue.length() != tailleSaisie)
            return false;
        if(!Character.isAlphabetic(coupJoue.charAt(char1)) ||
!Character.isAlphabetic(coupJoue.charAt(char2)))
            return false;
        return Character.isDigit(coupJoue.charAt(num1)) &&
Character.isDigit(coupJoue.charAt(num2));
    }

    /**
     * @return la chaine de caractere saisie
     */
    private static String getScanner() {
        Scanner saisie = new Scanner(System.in);
        return saisie.next();
    }

    /**
     * @param coupDepart coord a convertir
     * @param plateau sur lequel le joueur joue
     * @return converti la saisie alphabetique en saisie numerique dans une chaine
     */
    private String getStringCoord(String coupDepart, Echiquier plateau) {
        int colonne = coupDepart.charAt(char1) - debutAlphabet;
        int rangee = plateau.getTaille() -
Integer.parseInt(Character.toString(coupDepart.charAt(num1))) ;
        return colonne + Integer.toString(rangee);
    }
}
```

IA.java

```
package joueur;

import jeu.IJoueur;
import coord.Coord;
import echiquier.Echiquier;
import echiquier.IPiece;
import equipe.Team;

import java.util.ArrayList;
import java.util.Random;

public class IA extends Joueur{
    private final static int NB_COUP_CALCULE = 10; // changer cette valeur pour
étendre le jeu ou le réduire
    public IA(Team team){
        super(team);
    }

    /**
     * @param plateau Echiquier concerne par le test
     * @param adv     adversaire du joueur
     * @param depart  coord de depart de la piece
     * @param arrive  coord d'arrive de la pice
     * @return true constamment car l'IA calcul uniquement ses coups parmis les
coups legaux
     */
    @Override
    public boolean coupLegal(Echiquier plateau, IJoueur adv, Coord depart, Coord
arrive) {
        return true;
    }

    /**
     * @param plateau Echiquier concerne par le coup a jouer
     * @param adv     adversaire du joueur concerne
     * @return une chaine de caractere contenant les coups jouables
     * Fais le tour de l'echiquier et cherche NB_COUP_CALCULE nombre de deplacement
possible pour le joueur
     * les stocks et en choisis aléatoirement un puis le renvoi
     */
    @Override
    public String getCoup(Echiquier plateau, IJoueur adv) {
        IPiece test;
        IPiece roi = getKing(plateau);
        ArrayList<Coord> depart= new ArrayList<>();
        ArrayList<Coord> arrivee= new ArrayList<>();
        Random rand = new Random();
        int indexCoord;
        for(int rangee = 0; rangee < plateau.getTaille(); rangee++)
            for(int colonne = 0; colonne < plateau.getTaille(); colonne++) {
                if(depart.size() == NB_COUP_CALCULE)
                    break;
                test = plateau.getPiece(new Coord(rangee, colonne));
                if(plateau.DeplacementTest(new Coord(rangee, colonne), roi))
                    if(coupSimuleValide(plateau, plateau.getCoord(roi), new
Coord(rangee, colonne), adv)){

                        depart.add(new Coord(plateau.getCoord(roi).getRangee(),
plateau.getCoord(roi).getColonne()));
                        arrivee.add(new Coord(rangee, colonne));
```

```
                    }
                if(test.getTeam().equals(this.getTeam())){
                    if(isKing(plateau, rangee, colonne))
                        continue;
                    for(int rawEnd = 0; rawEnd < plateau.getTaille(); rawEnd++)
                        for (int colEnd = 0; colEnd < plateau.getTaille();
colEnd++)
                            if(plateau.DeplacementTest(new Coord(rawEnd, colEnd),
test))
                                if(coupSimuleValide(plateau, new Coord(rangee,
colonne), new Coord(rawEnd, colEnd), adv)) {
                                    depart.add(new Coord(rangee, colonne));
                                    arrivee.add(new Coord(rawEnd, colEnd));
                                }
                }
            }
        indexCoord = rand.nextInt(arrivee.size());
        StringBuilder resultat = new
StringBuilder().append(depart.get(indexCoord).getColonne()).append(depart.get(index
Coord).getRangee());
        return
resultat.append(arrivee.get(indexCoord).getColonne()).append(arrivee.get(indexCoord
).getRangee()).toString();
    }

    /**
     * @param adv adversaire du joueur
     * @return toujours vrai car un IA acceptera toujours une demande de NUL
     */
    @Override
    public boolean wantMatchNul(IJoueur adv) {
        return true;
    }
}
```

Joueur.java

```
package joueur;

import jeu.IJoueur;
import coord.Coord;
import echiquier.Echiquier;
import echiquier.IPiece;
import equipe.Team;
import piece.FabriquePiece;
import piece.PiecesEnum;

public abstract class Joueur implements IJoueur {
    private final Team equipe;
    protected final static int char1 = 0, num1 = 1, char2 = 2, num2 = 3,
tailleSaisie = 4;
    private boolean perdant = false;
    private boolean nul = false;
    private boolean isTurn = false;


    public Joueur(Team team){
        equipe = team;
    }

    /**
     * @param plateau Echiquier concerne par le coup a jouer
     * @param adv adversaire du joueur concerne
     * @return les coordonnes du coup sous format de chaine de caracteres
     * Obtient une chaine de caractères des coordonnes du coup joué avec un
synthaxe correct
     */
    public abstract String getCoup(Echiquier plateau, IJoueur adv);

    /**
     * @param plateau Echiquier concerne par le test
     * @param adv adversaire du joueur
     * @param depart coord de depart de la piece
     * @param arrive coord d'arrive de la pice
     * @return vrai si le coup est legal sinon faux
     */
    public abstract boolean coupLegal(Echiquier plateau, IJoueur adv, Coord depart,
Coord arrive);

    /**
     * @param plateau Echiquier sur lequel on jouer
     * @param adv adversaire du joueur
     * @return vrai si le joueur a pu faire un coup, sinon faux
     * permet au joueur de jouer un coup si les conditions sont réunis et accepte
son coup si il est legal
     */
    @Override
    public boolean jouer(Echiquier plateau, IJoueur adv) {
        if (isMat(plateau, adv))
            return false;
        if(isNul())
            return false;
        String coupJoue = getCoup(plateau, adv);
        if(coupJoue == null) {
            System.out.println("1coup illegal, veuillez recommencer");
            return false;
        }
```

```
        Coord coordDepart = getCoord(coupJoue.substring(char1,char2));
        Coord coordArrive = getCoord(coupJoue.substring(char2,tailleSaisie));
        if(!coupLegal(plateau, adv, coordDepart, coordArrive)) {
            System.out.println("2coup illegal, veuillez recommencer");
            return false;
        }
        if(!plateau.Deplacement(coordArrive, plateau.getPiece(coordDepart))){
            System.out.println("3coup illegal, veuillez recommencer");
            return false;
        }
        return !materielInsuffisant(plateau);
    }

    /**
     * @param plateau dans lequel on cherche le materiel
     * @return vrai si le materiel est insuffisant pour finir sur autre chose qu'un
match nul, sinon faux
     */
    private boolean materielInsuffisant(Echiquier plateau) {
        int nbPieceBlanche = 0;
        int nbPieceNoir = 0;
        for(int rangee = 0; rangee < plateau.getTaille(); rangee++)
            for(int colonne = 0; colonne < plateau.getTaille(); colonne++) {
                if(plateau.getPiece(new Coord(rangee,
colonne)).getTeam().isBlack())
                    nbPieceNoir++;
                if(plateau.getPiece(new Coord(rangee,
colonne)).getTeam().isWhite())
                    nbPieceBlanche++;
            }
        if(nbPieceBlanche == 1 && nbPieceNoir == 1) {
            System.out.println(plateau);
            this.nul = true;
        }
        return nul;

    }

    /**
     * @return vrai si le joueur a perdu sinon faux
     */
    @Override
    public boolean aPerdu() {
        return perdant;
    }

    /**
     * @return vrai s'il y a match nul, sinon faux
     */
    @Override
    public boolean isNul() {
        return nul;
    }

    /**
     * @return vrai si c'est le tour du joueur sinon faux
     */
    @Override
    public boolean isTurn() {
        return isTurn;
    }

    /**
```

```
     * @param tour affecte l'attribut tour du joueur
     */
    @Override
    public void setTurn(boolean tour) {
        isTurn = tour;
    }

    /**
     * @return l'equipe du joueur
     */
    @Override
    public Team getTeam() {
        return equipe;
    }

    @Override
    public String toString(){
        String res;
        if(getTeam().isBlack())
            res = "NOIR";
        else
            res = "BLANC";
        return res;
    }

    /**
     * le joueur est pat
     */
    protected void setPat(){
        this.nul = true;
    }

    /**
     * definit le joueur comme perdant
     */
    protected void setPerdant(){
        this.perdant = true;
    }

    /**
     * @param plateau Echiquier sur lequel on test
     * @param coordDepart coord de depart de la piece
     * @param coordArrive coord d'arrivee de la piece
     * @param adv adv du joueur
     * @return vrai si le coup est legal faux sinon
     * Simule un coup pour voir si le roi est en echec après celui-ci puis renvoi
vrai si oui, sinon non
     * Remet les pieces dans l'etat d'avant appel de la methode
     */
    protected boolean coupSimuleValide(Echiquier plateau, Coord coordDepart, Coord
coordArrive, IJoueur adv){
        boolean res;
        IPiece temp = plateau.getPiece(coordArrive);
        plateau.setPiece(coordArrive, plateau.getPiece(coordDepart));
        plateau.setPiece(coordDepart, FabriquePiece.fab(PiecesEnum.Vide,
Team.autre));
        res = this.isEchec(plateau, adv);
        plateau.setPiece(coordDepart, plateau.getPiece(coordArrive)); // on remet
la piece deplace a sa place
        plateau.setPiece(coordArrive, temp); // on remet aussi la piece mange
        return !res;
    }
```

```
    /**
     * @param plateau Echiquier sur lequel on cherche
     * @param rangee de la piece a verifier
     * @param colonne de la piece a verifier
     * @return vrai si la piece est un roi, sinon faux
     */
    protected boolean isKing(Echiquier plateau, int rangee, int colonne) {
        if(plateau.getPiece(new Coord(rangee,
colonne)).getType().equals(PiecesEnum.Roi))
            return plateau.getPiece(new Coord(rangee,
colonne)).getTeam().equals(this.getTeam());
        return false;
    }

    /**
     * @param plateau Echiquier pris en compte
     * @return le roi du joueur
     * cherche le roi dans le plateau
     */
    protected IPiece getKing(Echiquier plateau) {
        for(int rangee = 0; rangee < plateau.getTaille(); rangee++)
            for(int colonne = 0; colonne < plateau.getTaille(); colonne++)
                if (isKing(plateau, rangee, colonne))
                    return plateau.getPiece(new Coord(rangee, colonne));
        return null;
    }

    /**
     * @param plateau Echiquier pris en compte
     * @param adv adversaire du joueur
     * @return vrai si le joueur est mat sinon faux
     */
    private boolean isMat(Echiquier plateau, IJoueur adv) {
        IPiece roi = getKing(plateau), test;
        for(int rangee = 0; rangee < plateau.getTaille(); rangee++)
            for(int colonne = 0; colonne < plateau.getTaille(); colonne++){
                test = plateau.getPiece(new Coord(rangee, colonne));
                if (roiPeutSeLibererSeul(plateau, adv, roi, rangee, colonne))
                    return false;
                if(test.getTeam().equals(this.getTeam()))
                    if (canSaveKing(plateau, adv, test, rangee, colonne))
                        return false;
            }
        try {
            if (getKing(plateau).estAttaque(plateau, adv.getTeam())) {
                perdant = true;
                return true;
            }
            nul = true;
        } catch(NullPointerException e){
            perdant = true;
        }
        return false;
    }

    /**
     * @param plateau Echiquier pris en compte
     * @param adv adversaire du joueur
     * @param test piece sur laquel la simulation est faite
     * @param rangee de la piece tester
     * @param colonne de la piece tester
     * @return vrai si une piece peut s'interposer ou manger la piece adversaire
mettant en echec le roi
```

```
     */
    private boolean canSaveKing(Echiquier plateau, IJoueur adv, IPiece test, int
rangee, int colonne) {
        if(isKing(plateau, rangee, colonne)) // les deplacements du roi ont deja
etait tester
            return false;
        for(int rawEnd = 0; rawEnd < plateau.getTaille(); rawEnd++)
            for (int colEnd = 0; colEnd < plateau.getTaille(); colEnd++)
                if(plateau.DeplacementTest(new Coord(rawEnd, colEnd), test))
                    if (coupSimuleValide(plateau, new Coord(rangee, colonne), new
Coord(rawEnd, colEnd), adv))
                        return true;
        return false;
    }

    /**
     * @param plateau Echiquier pris en compte
     * @param adv adversaire du joueur
     * @param roi du joueur
     * @param rangee de test
     * @param colonne de test
     * @return vrai si le roi peut se liberer seul sinon non
     */
    private boolean roiPeutSeLibererSeul(Echiquier plateau, IJoueur adv, IPiece
roi, int rangee, int colonne) {
        if(plateau.DeplacementTest(new Coord(rangee, colonne), roi))
            return coupSimuleValide(plateau, plateau.getCoord(roi), new
Coord(rangee, colonne), adv);
        return false;
    }

    /**
     * @param plateau Echiquier sur lequel la recherche est faite
     * @param adv du joueur
     * @return vrai si le joueur est en echec, faux s'il ne l'est pas
     */
    private boolean isEchec(Echiquier plateau, IJoueur adv) {
        try {
            return this.getKing(plateau).estAttaque(plateau, adv.getTeam());
        } catch(NullPointerException e){
            System.out.println("joueur " + this +  " n'a pas de roi, veillez à bien
initialiser les pieces de la partie");
            perdant = true;
        }
        return false;
    }

    /**
     * @param coupDepart chaine de caractere a changer en coordonnees
     * @return les coordonnees converti
     * Converti une chaine de caractere avec un format correcte en coordonnes
     */
    private Coord getCoord(String coupDepart) {
        int colonne =
Integer.parseInt(Character.toString(coupDepart.charAt(char1)));
        int rangee = Integer.parseInt(Character.toString(coupDepart.charAt(num1)))
;
        return new Coord(rangee, colonne);
    }

}
```

FabriquePiece.java

```
package piece;

import echiquier.IPiece;
import equipe.Team;

public class FabriquePiece {
    /**
     * @param piece type de la piece voulut
     * @param equipe de la piece voulut
     * @return la piece avec le type correspondant et l'equipe donnee en parametre
     */
    public static IPiece fab(PiecesEnum piece, Team equipe){
        if(piece.equals(PiecesEnum.Vide))
            return vide();
        if(piece.equals(PiecesEnum.Roi))
            return king(equipe);
        if(piece.equals(PiecesEnum.Tour))
            return tour(equipe);
        return vide();
    }

    /**
     * @return une piece vide
     */
    private static IPiece vide() {
        return new PieceVide();
    }

    /**
     * @param team
     * @return un Roi avec l'equipe donnee en parametre
     */
    private static IPiece king(Team team) {
        return new Roi(team) ;
    }

    /**
     * @param team
     * @returnun Tour avec l'equipe donnee en parametre
     */
    private static IPiece tour(Team team) {
        return new Tour(team);
    }
}
```

Piece.java

```
package piece;

import coord.Coord;
import coord.Deplacement;
import echiquier.Echiquier;
import echiquier.IPiece;
import equipe.Team;

public abstract class Piece implements IPiece{
    private final Team team;
    /**
     * @param crew equipe de la piece
     */
    public Piece(Team crew) {
        team = crew;
    }
    /**
     * @return format d'affichage de la piece
     */
    public abstract String toString();
    /**
     * @param d Vecteur de deplacement
     * @return vrai si la plage de déplacement est invalide, faux si la plage de
déplacement est valide
     * indique la plage de deplacement invalide de la piece
     */
    public abstract boolean plageDeplacementInvalide(Deplacement d);
    /**
     * @param plateau Echiquier a prendre en compte
     * @param destination Coordonnee d'arrivee de la piece a deplacer
     * @return vrai si un obstacle empeche le deplacement
     */
    public abstract boolean obstacle(Echiquier plateau, Coord destination);
    /**
     * @param plateau Echiquier a prendre en compte
     * @param destination Destination de la piece a deplacer
     * @return vrai si les coordonnees correspondent a une zone d'attaque possible
pour la piece
     */
    public abstract boolean zoneAttaqueValide(Echiquier plateau, Coord
destination);
    /**
     * @return l'equipe de la piece
     */
    public Team getTeam(){
        return this.team;
    }
    /**
     * @param plateau Echiquier a prendre en compte
     * @param c Coordonnee d'arrivee
     * @return vrai si le deplacement respecte toutes les conditions a l'exception
de la mise en echec du roi
     */
    @Override
    public boolean deplacementValide(Echiquier plateau, Coord c) {
        Deplacement d = new Deplacement(plateau.getCoord(this), c);
        if(!Coord.coordCorrecte(c, plateau.getTaille()))
            return false;
        if(this.getTeam().equals((plateau.getPiece(c).getTeam())))
            return false;
```

```
        if(!this.zoneAttaqueValide(plateau, c))
            return false;
        if(this.plageDeplacementInvalide(d))
            return false;
        if(obstacle(plateau,c))
            return false;
        return plateau.getCoord(this) != c;
    }
    /**
     * @param plateau Echiquier pris en compte
     * @param adversaire Equipe des pieces menaçantes pour la piece
     * @return vrai si la piece est attaque par une piece ennemi, faux si elle ne
l'est pas
     */
    @Override
    public boolean estAttaque(Echiquier plateau, Team adversaire) {
        IPiece test;
        Coord coordTest = plateau.getCoord(this);
        for(int colonne = 0; colonne < plateau.getTaille(); colonne++)
            for (int rangee = 0; rangee < plateau.getTaille(); rangee++) {
                if (colonne == coordTest.getColonne() && rangee ==
coordTest.getRangee())
                    continue;
                test = plateau.getPiece(new Coord(rangee, colonne));
                if (test.getTeam().equals(adversaire))
                    if (test.deplacementValide(plateau, coordTest))
                        return true;
            }
        return false;
    }
}
```

Tour.java

```
package piece;

import coord.Coord;
import coord.Deplacement;
import echiquier.Echiquier;
import equipe.Team;

import java.util.Locale;

public class Tour extends Piece {
    private final static String tour = "T";

    public Tour(Team crew) {
        super(crew);
    }

    @Override
    public String toString() {
        if(this.getTeam().isWhite())
            return tour;
        return tour.toLowerCase(Locale.ROOT);
    }

    /**
     * @param d Vecteur de deplacement
     * @return vrai si le deplacement de la tour est valide sinon faux
     * ou en horizontale.
     * indique la plage de deplacement invalide de la piece
     */
```

```
    @Override
    public boolean plageDeplacementInvalide(Deplacement d) {
        return d.getMoveColonne() != 0 && d.getMoveRangee() != 0; //
    }

    /**
     * @param plateau Echiquier a prendre en compte
     * @param finish coordonne d'arrive
     * @return true, si il y a un obstacle sinon false
     */
    @Override
    public boolean obstacle(Echiquier plateau, Coord finish) {
        Coord start = plateau.getCoord(this);       //obtention des coordonnées de
la piece a deplacer
        if(start.getRangee() == finish.getRangee()) {
            return rechercheObstacle(plateau, start.getColonne(),
finish.getColonne(), start.getRangee(), true); // vrai = deplacement en colonne
        }
        return rechercheObstacle(plateau, start.getRangee(), finish.getRangee(),
start.getColonne(), false); // faux = deplacement en rangee
    }

    /**
     * @param plateau     Echiquier a prendre en compte
     * @param destination Destination de la piece a deplacer
     * @return constamment true car le deplacement de la tour concorde avec sa zone
d'attaque
     */
    @Override
    public boolean zoneAttaqueValide(Echiquier plateau, Coord destination) {
        return true;
    }

    /**
     * @return le type de la piece à savoir TOUR
     */
    @Override
    public PiecesEnum getType() {
        return PiecesEnum.Tour;
    }

    /**
     * @param plateau Echiquier sur lequel la recherche est faite
     * @param depart Coord de départ de la piece
     * @param arrive Coord d'arrivee de la piece
     * @param c3 coord fixe
     * @param dim vrai si le deplacement est en colonne, faux si non
     * @return vrai si un obstacle est sur la route de la piece sinon faux
     */
    private boolean rechercheObstacle(Echiquier plateau, int depart, int arrive,
int c3, boolean dim) {
        if(depart + 1 == arrive || depart -1 == arrive)    // si le deplacement ne
se fait que d'une case
            return false;           // renvoyer qu'il n'y a pas d'obstacle entre le
depart et l'arrivee
        int start, finish;
        if(depart > arrive) {   // permet de partir du plus petit indice +1 et
d'arrive au plus grand
            start = arrive + 1; // partir de la destination
            finish = depart;    // arrive a la piece
        }
        else {
            start = depart + 1; // partir de la piece a deplacer
```

```
            finish = arrive;    // arrive a la piece de destination
        }
        if(dim){    // si le deplacement est en colonne
            for(int i = start; i < finish; i++){    // pour chacune des colonnes
dans la rangee c3
                if(!plateau.getPiece(new Coord(c3,
i)).getType().equals(PiecesEnum.Vide)) //verifier si la case est vide
                    return true; // renvoi qu'un obstale est trouve
            }
        }
        else {  // si le deplacement est en rangee
            for(int i = start; i < finish; i++){ // pour chacune des rangee dans la
colonne c3
                if(!plateau.getPiece(new Coord(i,
c3)).getType().equals(PiecesEnum.Vide)) //verifier si la case est vide
                    return true; // renvoi qu'un obstale est trouve
            }
        }
        return false; // renvoi qu'aucun obstale est trouve
    }
}
```

Roi.java

```
package piece;

import coord.Coord;
import coord.Deplacement;
import echiquier.Echiquier;
import equipe.Team;

import java.util.Locale;

public class Roi extends Piece {
    private final static String roi = "R";
    public Roi(Team team){
        super(team);
    }

    @Override
    public String toString() {
        if(this.getTeam().isWhite())
            return roi;
        else
            return roi.toLowerCase(Locale.ROOT);
    }

    /**
     * @param d Vecteur de deplacement
     * @return true si  le roi n'est pas dans sa plage de deplacement / false si le
roi est dans sa plage de deplacement
     * indique la plage de deplacement invalide de la piece
     */
    @Override
    public boolean plageDeplacementInvalide(Deplacement d) {
        return Math.abs(Math.abs(d.getMoveRangee()) - Math.abs(d.getMoveColonne()))
< -1 ||
                Math.abs(Math.abs(d.getMoveRangee()) -
Math.abs(d.getMoveColonne())) > 1 ||
                Math.abs((Math.abs(d.getMoveRangee()) *
Math.abs(d.getMoveColonne()))) > 1;
    }

    /**
     * @param plateau     Echiquier a prendre en compte
     * @param destination Coordonnee d'arrivee de la piece a deplacer
     * @return false car le roi n'a jamais d'obstacle, il se deplace toujours d'une
case
     */
    @Override
    public boolean obstacle(Echiquier plateau, Coord destination) {
        return false;
    }

    /**
     * @param plateau     Echiquier a prendre en compte
     * @param destination Destination de la piece a deplacer
     * @return true car le roi peut toujours attaquer un ennemie qui est zone de
deplacement
     * **/
    @Override
    public boolean zoneAttaqueValide(Echiquier plateau, Coord destination) {
        return true;
    }
```

```
    /**
     * @return PiecesEnum.Roi car la piece est un Roi
     */
    @Override
    public PiecesEnum getType() {
        return PiecesEnum.Roi;
    }

}
```

PieceVide.java

```
package piece;

import coord.Coord;
import coord.Deplacement;
import echiquier.Echiquier;
import equipe.Team;

public class PieceVide extends Piece {
    private final static String none = " ";

    public PieceVide(){
        super(Team.autre);
    }

    @Override
    public String toString() {
        return none;
    }

    /**
     * @param d Vecteur de deplacement
     * @return false car la piece vide n'a pas de plage de depalcement.
     * indique la plage de deplacement invalide de la piece
     */
    @Override
    public boolean plageDeplacementInvalide(Deplacement d) {
        return false;
    }

    /**
     * @param plateau     Echiquier a prendre en compte
     * @param destination Coordonnee d'arrivee de la piece a deplacer
     * @return true car partout où il va, ce sera bloqué.
     */
    @Override
    public boolean obstacle(Echiquier plateau, Coord destination) {
        return true;
    }

    /**
     * @param plateau     Echiquier a prendre en compte
     * @param destination Destination de la piece a deplacer
     * @return false car la piece vide n'attaque jamais.
     */
    @Override
    public boolean zoneAttaqueValide(Echiquier plateau, Coord destination) {
        return false;
    }

    /**
```

```
     * @return PiecesEnume.Vide car la piece est une Piece vide
     */
    @Override
    public PiecesEnum getType() {
        return PiecesEnum.Vide;
    }

}
```

PiecesEnum.java

```
package piece;

public enum PiecesEnum {
    Roi, Tour, Vide
}
```

IJoueur.java

```
package jeu;

import echiquier.Echiquier;
import equipe.Team;

public interface IJoueur {
    /**
     * @param plateau Echiquier sur lequel on jouer
     * @param adv adversaire du joueur
     * @return vrai si le coup jouer est legal
     * Propose au IJoueur de jouer un tour
     */
    boolean jouer(Echiquier plateau, IJoueur adv);

    /**
     * @return l'attribut perdant du joueur
     * Verifie si l'attribut du joueur a etait mis a jour et que la partie finie
sur un echec et mat
     */
    boolean aPerdu();

    /**
     * @return l'attribut nul du joueur
     * Verifie si l'attribut nul du jouer a etait mis a jouer et que la partie
finie sur un echec et mat
     */
    boolean isNul();

    /**
     * @param adv adversaire du joueur
     * @return vrai si l'adversaire accepte le match nul
     * propose un match nul a l'adversaire
     */
    boolean wantMatchNul(IJoueur adv);

    /**
     * @return vrai si c'est le tour du joueur sinon faux
     */
    boolean isTurn();

    /**
     * @param tour affecte vrai a l'attribut isTurn du joueur si c'est son tour
sinon non
```

```
     */
    void setTurn(boolean tour);

    /**
     * @return l'equipe du joueur
     */
    Team getTeam();

    String toString();
}
```

IFabriqueIJoueur

```
package jeu;

public interface IFabriqueIJoueur {
    /**
     * @param choix des joueurs a initialiser
     * @param joueur compteur du joueur inialise ( xieme joueur )
     * @return le IJoueur souhaite
     */
    IJoueur initJoueur(int choix, int joueur);
}
```

Application.java

```
package jeu;

import echiquier.Echiquier;
import joueur.FabriqueIJoueur;

import java.util.Scanner;


public class Application {
    private final static int etatParti = 3;
    private static final int JOUEUR_BLANC = 0, JOUEUR_NOIR = 1;

    public static void main(String[] args) {
        IJoueur blanc, noir;
        Echiquier echiquier = new Echiquier();
        echiquier.init();
        msgMenu();
        int choixJoueur = gameSetting();
        IFabriqueIJoueur fab = new FabriqueIJoueur();
        blanc = fab.initJoueur(choixJoueur, JOUEUR_BLANC);
        noir = fab.initJoueur(choixJoueur, JOUEUR_NOIR);
        assert(noir != null && blanc != null);

        msgPreambule();
        IJoueur Actif = blanc;
        blanc.setTurn(true);
        if(etatParti(blanc,noir) != 0){
            System.out.println(echiquier);
        }
        while(etatParti(blanc,noir) == 0) {
            if(etatParti(blanc,noir) == 0)
                System.out.println(echiquier);
            while (!Actif.jouer(echiquier, getInactif(blanc, noir)))
                if (etatParti(blanc, noir) != 0)
                    break;
            Actif = swap(blanc, noir);
```

```
        }

        msgFin(etatParti(blanc,noir), blanc, noir);
    }

    /**
     * @return le choix de l'utilisateur
     * propose une saisie a l'utilisateurs lui permettant de choisir entre 3
valeurs a savoir 1, 2 et 3 pour determiner le type de joueur de la partie
     */
    private static int gameSetting() {
        Scanner menu = new Scanner(System.in);
        int choixJoueur;
        while (true) {
            String choixMenu = menu.next();
            if(!Character.isDigit(choixMenu.charAt(0))) {
                System.out.println("Veuillez saisir une valeur numérique");
                continue;
            }
            choixJoueur = Integer.parseInt(choixMenu);
            switch (choixJoueur) {
                case 1:
                case 2:
                case 3:
                    return choixJoueur;
                default:
                    System.out.println("Veuillez saisir une valeur entre 1 et 3");
            }
        }
    }

    /**
     * @param blanc joueur de l'equipe blanche
     * @param noir joueur de l'equipe noire
     * @return retourne le IJoueur qui ne joue pas
     * obtient le joueur qui n'est pas en train de jouer
     */
    private static IJoueur getInactif(IJoueur blanc, IJoueur noir) {
        return (!blanc.isTurn()) ? blanc : noir;
    }

    /**
     * @param blanc joueur de l'equipe blanche
     * @param noir joueur de l'equipe noire
     * @return retourne le IJoueur qui joue le prochain tour
     * inverse le joueur actif avec le joueur inactif et renvoi le nouveau joueur
actif
     */
    private static IJoueur swap(IJoueur blanc, IJoueur noir) {
        blanc.setTurn(!blanc.isTurn());
        noir.setTurn(!noir.isTurn());
        return (blanc.isTurn()) ? blanc : noir;
    }

    /**
     * @param blanc joueur de l'equipe blanche
     * @param noir joueur de l'equipe noire
     * @return 3 si le blanc a perdu, 2 si le noir a perdu, 1 match nul et 0 si la
partie est en cours
     * Permet de connaître l'avancement de la partie
     */
    private static int etatParti(IJoueur blanc, IJoueur noir) {
        int fin = etatParti;
```

```
        if(blanc.aPerdu())
            return fin;
        fin--;
        if(noir.aPerdu())
            return fin;
        fin--;
        if(blanc.isNul() || noir.isNul())
            return fin;
        fin--;
        return fin;
    }

    /**
     * Affiche un menu au joueur permettant de choisir le type de joueur pour la
partie
     */
    private static void msgMenu(){
        System.out.println("Choisissez le nombre associe à votre partie : ");
        System.out.println("1 = Joueur contre Joueur");
        System.out.println("2 = Joueur contre IA");
        System.out.println("3 = IA contre IA");
    }

    /**
     * Indique au joueur la méthode pour abandonner et la méthode pour demander
match nul
     */
    private static void msgPreambule(){
        System.out.println("Pour proposer a votre adversaire un match nul saisissez
: NUL");
        System.out.println("Pour abandonner saisissez STOP");
    }

    /**
     * @param situation entier renvoyant la raison de fin de parti ( voir ligne 92
)
     * @param blanc joueur de l'equipe blanche
     * @param noir joueur de l'equipe noire
     * Affiche le message de fin avec la raison de la fin de partie
     */
    private static void msgFin(int situation, IJoueur blanc, IJoueur noir) {
        switch(situation){
            case 1:
                System.out.println("Match nul");
                break;
            case 2:
                System.out.println("BRAVO JOUEUR " + blanc.toString() + " VOUS AVEZ
VAINCU JOUEUR " + noir.toString());
                break;
            case 3:
                System.out.println("BRAVO JOUEUR " + noir.toString() + " VOUS AVEZ
VAINCU JOUEUR " + blanc.toString());
        }
    }
}
```

IV. Explication des tâches à accomplir

Pour ajouter des pièces il y a deux options :

Option 1 : Créer une nouvelle classe qui implémentera l'interface IPiece et définir les méthodes pour obtenir l'équipe de la pièce, pour savoir si la pièce est attaquée, pour obtenir son type, son toString et enfin pour savoir si son déplacement est valide.

Certaines méthodes peuvent ne pas être utile à toutes les classes l'implémentant (une pièce qui ne se déplacerait pas, qui ne serait jamais attaqué) il suffirait de renvoyer faux constamment ou renvoyer vrai constamment en fonction de la situation. La méthode la plus importante ici à redéfinir est deplacementValide, c'est elle qui indique les règles de déplacement pour la pièce et qui permettra de changer les règles pour les futurs classes.

Option 2 : Créer une nouvelle classe et la faire hériter de la classe abstraite Piece. Pour implanter les autre pièces du jeu d'échec cette classe est idéale, les méthodes à définir sont celles du toString pour l'affichage des pièces, la méthode plageDeplacementInvalide, qui indique les déplacements interdits en prenant uniquement en compte l'échiquier (ne pas y introduire les contraintes tels que l'interdiction de déplacement sur un allié, l'interdiction de manger un roi, l'interdiction de sortir du terrain).

Pour illustrer cette méthode interdit simplement à la tour de se déplacer autrement qu'horizontalement ou verticalement mais autorise tout déplacement respectant ces conditions. Une autre méthode déjà définit permet de gérer les autres contraintes, implémenter une nouvelle pièce ne serait donc pas compliquer. La méthode plageDeplacementInvalide renvoie faux si le déplacement n'est pas autorisé.

La seconde méthode à définir est obstacle, elle renvoi vrai si une quelconque pièce empêche la pièce à déplacer d'aller à la case voulut. Une pièce qui n'attaque pas aux mêmes emplacements que ses déplacements devront définir zoneAttaqueValide

Une fois cela fait le jeu fonctionnera parfaitement avec les nouvelles pièces.

Pour implémenter un nouvel IA, il faudrait simplement créer une nouvelle classe qui soit implémenterait l'interface IJoueur et définirait la méthode test pour qu'elle renvoi vrai et la méthode getCoup qui contiendrait l'algorithme choisissant le coup pertinent à faire. La 2nd méthode serait de créer une nouvelle classe IA qui hériterait de la classe IA actuelle et la faire redéfinir la méthode getCoup.

Une fois cela fait le jeu fonctionnera parfaitement avec les nouvelles pièces.

V. Bilan du projet

Notre trinôme a travaillé pendant près de 3 semaines sur ce projet, et nous pensons que, dans l'ensemble, le projet est une expérience très enrichissante. Il est vrai que nous avons dû faire face à quelques problèmes notamment nous avons eu du mal à amorcer. En effet, il y a eu une légère difficulté au niveau de la gestion des packages et des classes. Cependant grâce au TP et TD nous avons mieux saisit l'ordonnancement des interfaces, des classes abstraites, des héritages et les concepts de

polymorphisme et de délégation etc.... En effet, dans notre groupe, le travail collaboratif était très présent. Nous avons facilement pu trouver les moyens de s'appeler et planifier des réunions afin d'écrire le code ensemble. Ainsi, nous avons pu éviter des problèmes très importants et avoir de nombreux points de vue pour apporter un projet très réfléchis.

La notion de sous classe et héritage était au début incompris, en effet la difficulté était de rendre abstrait nos objets, pour que le code soit flexible et qu'il s'adapte aux maximums d'ajouts à l'avenir. Néanmoins grâce aux indications donné par Mr Poitrenaud, nous avons pu rendre notre code flexible et performant.

- ✓ Joueur vs Joueur, Joueur vs IA, IA vs IA
- ✓ Plateau flexible
- ✓ Faciliter à rajouter une nouvelle pièce dans le code
- ✓ Les coups illégaux sont détectés, signaler et rejeter
- ✓ Possibilité de faire un échec et mat
- ✓ Mise en place des règles Mat et Pat
- ✓ Ajout facile d'IA avec un algorithme plus performant
- ✓ Jeu d'échec évolutif

Ayant déjà fait des projets, nous avons appris de nos erreurs et nous sommes plutôt satisfait du travail que nous vous rendons.
A ce jour nous pensons que nous avons donné notre maximum pour ce projet et que nous n'avons pas le moyen de le rendre meilleurs pour l'instant.