

---

# **RAPPORT DE PROJET**

## **BPO2 ECHECS**

---

**Binôme:**

CHEN Lorie

*Groupe 105*

LIM Réléna

*Groupe 102*

A rendre le lundi 24 mai 2021

## Table des matières

<b>Table des matières</b>	<b>2</b>
Présentation du projet	3
Diagramme d'architecture	4
Tests Unitaires	5
AppliTest.java	5
CoordonnéeTest.java	6
EchiquierTest.java	7
JoueurTest.java	10
PartieTest.java	12
PièceTest.java	17
CavalierTest.java	18
FouTest.java	20
PionTest.java	21
ReineTest.java	23
RoiTest.java	25
TourTest.java	28
Tâches à accomplir	29
Bilan du projet	30
Code Source	31
Appli.java	31
Coordonnée.java	34
Echiquier.java	36
Joueur.java	41
Partie.java	43
Pièce.java	50
Cavalier.java	54
Fou.java	58
Pion.java	62
Reine.java	66
Roi.java	71
Tour.java	75

## Présentation du projet

- Le but de ce projet est de programmer le jeu des échecs en java et de réaliser le diagramme d'architecture correspondant. L'objectif est d'utiliser le polymorphisme et l'héritage.

Le minimum demandé pour ce projet était de coder le déplacement des deux tours et des deux rois, ainsi que de coder les règles du pat, du mat et de l'abandon. Ces règles là ainsi que le déplacement des pièces marchent.

Pour coder le roi et la tour nous avons codé une super classe Pièce afin que Tour et Roi héritent des méthodes de Pièce, qu'on a spécialisé par la suite. Cela nous a facilité l'ajout des autres pièces (Reine, Fou, Cavalier et Pion) puisqu'il suffisait de faire la même chose en spécialisant les méthodes pour chaque pièce. Les déplacements des autres pièces marchent également.

Pour le pion, nous avons fait en sorte qu'il puisse se déplacer de 2 cases en avant au premier coup. Et pour le roi, nous avons vérifié s'il est en échec et s'il peut être protégé.

Nous avons codé d'autres règles : la règle des cinquante coups et la nulle si matériel insuffisant. Nous n'avons pas codé le roque, la prise au passant, ni la promotion du pion. Dans notre programme, lorsqu'un pion arrive au bout, il est bloqué. Nous avons codé des IA mais qui ne sont pas intelligentes : elles jouent des coups aléatoires parmi la liste des coups possibles de chaque pièce.

Ceci nous a permis de choisir entre 3 modes de jeu : Humain VS Humain, Humain VS IA ou IA VS IA. Il y a peu de chance d'avoir un échec et mat si deux IA s'affrontent alors la règle des cinquante coups finira par s'appliquer quand il n'y aura plus beaucoup de pièces.



## Tests Unitaires

Nous avons fait des tests unitaires pour chaque méthode de chaque classe (sauf pour les getters et setters). Tous les tests se sont exécutés avec succès. (Les méthodes rendues publiques ont été remises privées après les avoir testées.)

## AppliTest.java

```
package test;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.Test;

import appli.Appli;

public class AppliTest {

    @Test
    public void testEstValide() {
        assertTrue(Appli.estValide(""));
        assertTrue(Appli.estValide("b4g6"));
        assertTrue(Appli.estValide("H4D5"));
        assertTrue(Appli.estValide("  h4    d5  "));

        assertFalse(Appli.estValide("r4t5"));
        assertFalse(Appli.estValide("d8g9"));
        assertFalse(Appli.estValide("r4"));
        assertFalse(Appli.estValide("4f5d"));
        assertFalse(Appli.estValide("3456"));
        assertFalse(Appli.estValide("erhv"));
        assertFalse(Appli.estValide("h4h5f4"));
    }
}
```









```

@Test
public void testToString() {
    Echiquier e = new Echiquier();
    String attendu = "      a  b  c  d  e  f  g  h  \n"
        + "      --- --- --- --- --- --- --- --- \n"
        + "8 | t | c | f | d | r | f | c | t | 8\n"
        + "      --- --- --- --- --- --- --- --- \n"
        + "7 | p | p | p | p | p | p | p | p | 7\n"
        + "      --- --- --- --- --- --- --- --- \n"
        + "6 |  |  |  |  |  |  |  |  | 6\n"
        + "      --- --- --- --- --- --- --- --- \n"
        + "5 |  |  |  |  |  |  |  |  | 5\n"
        + "      --- --- --- --- --- --- --- --- \n"
        + "4 |  |  |  |  |  |  |  |  | 4\n"
        + "      --- --- --- --- --- --- --- --- \n"
        + "3 |  |  |  |  |  |  |  |  | 3\n"
        + "      --- --- --- --- --- --- --- --- \n"
        + "2 | P | P | P | P | P | P | P | P | 2\n"
        + "      --- --- --- --- --- --- --- --- \n"
        + "1 | T | C | F | D | R | F | C | T | 1\n"
        + "      --- --- --- --- --- --- --- --- \n"
        + "      a  b  c  d  e  f  g  h  \n";
    assertEquals(e.toString(), attendu);
}
}

```

## JoueurTest.java

```
package test.echecTest;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;

import org.junit.Test;

import echec.Joueur;
import echec.Partie;

public class JoueurTest {

    @Test
    public void testJouerIA() {
        Partie p = new Partie("IA", "IA");
        Joueur j = new Joueur("IA", "BLANC");
        String echiquier = "      a   b   c   d   e   f   g   h       \n"
            + "      +---+---+---+---+---+---+---+---+ \n"
            + "8 | t | c | f | d | r | f | c | t | 8\n"
            + "      +---+---+---+---+---+---+---+---+ \n"
            + "7 | p | p | p | p | p | p | p | p | 7\n"
            + "      +---+---+---+---+---+---+---+---+ \n"
            + "6 |   |   |   |   |   |   |   |   | 6\n"
            + "      +---+---+---+---+---+---+---+---+ \n"
            + "5 |   |   |   |   |   |   |   |   | 5\n"
            + "      +---+---+---+---+---+---+---+---+ \n"
            + "4 |   |   |   |   |   |   |   |   | 4\n"
            + "      +---+---+---+---+---+---+---+---+ \n"
            + "3 |   |   |   |   |   |   |   |   | 3\n"
            + "      +---+---+---+---+---+---+---+---+ \n"
            + "2 | P | P | P | P | P | P | P | P | 2\n"
            + "      +---+---+---+---+---+---+---+---+ \n"
            + "1 | T | C | F | D | R | F | C | T | 1\n"
            + "      +---+---+---+---+---+---+---+---+ \n"
            + "      a   b   c   d   e   f   g   h       \n";
        assertEquals(echiquier, p.getEchiquier().toString());

        j.jouerIA(p);
        assertNotEquals(echiquier, p.getEchiquier().toString());
    }
}
```



## PartieTest.java

```

package test.echecTest;

import echec.Partie;
import echec.pieces.Cavalier;
import echec.pieces.Fou;
import echec.pieces.Roi;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertTrue;

import org.junit.Test;

public class PartieTest {

    @Test
    public void testJouer() {
        Partie p = new Partie("Humain", "IA");
        String echiquier = "      a  b  c  d  e  f  g  h      \n"
            + "      --- --- --- --- --- --- --- --- \n"
            + "8 | t | c | f | d | r | f | c | t | 8\n"
            + "      --- --- --- --- --- --- --- --- \n"
            + "7 | p | p | p | p | p | p | p | p | 7\n"
            + "      --- --- --- --- --- --- --- --- \n"
            + "6 |   |   |   |   |   |   |   |   | 6\n"
            + "      --- --- --- --- --- --- --- --- \n"
            + "5 |   |   |   |   |   |   |   |   | 5\n"
            + "      --- --- --- --- --- --- --- --- \n"
            + "4 |   |   |   | P |   |   |   |   | 4\n"
            + "      --- --- --- --- --- --- --- --- \n"
            + "3 |   |   |   |   |   |   |   |   | 3\n"
            + "      --- --- --- --- --- --- --- --- \n"
            + "2 | P | P | P |   | P | P | P | P | 2\n"
            + "      --- --- --- --- --- --- --- --- \n"
            + "1 | T | C | F | D | R | F | C | T | 1\n"
            + "      --- --- --- --- --- --- --- --- \n"
            + "      a  b  c  d  e  f  g  h      \n";
        p.jouer("d2d4");
        assertEquals(p.getEchiquier().toString(), echiquier);
        p.setTourDeBlanc(false);
        p.jouer(null);
        assertNotEquals(p.getEchiquier().toString(), echiquier);
    }
}

```



```

@Test
public void testMatérielInsuffisant() {
    Partie p = new Partie("Humain", "Humain");
    assertFalse(p.matérielInsuffisant());
    for(int i = 0; i < 8; ++i) {
        for(int j = 0; j < 8; ++j) {
            p.getEchiquier().setPièce(i, j, null);
        }
    }
    p.getEchiquier().setPièce(0, 4, new Roi("NOIR", 0, 4));
    p.getEchiquier().setPièce(7, 4, new Roi("BLANC", 7, 4));
    p.getEchiquier().setPièce(7, 3, new Cavalier("BLANC", 7,
3));

    assertTrue(p.matérielInsuffisant());

    p.getEchiquier().setPièce(7, 3, new Fou("BLANC", 7, 3));
    assertTrue(p.matérielInsuffisant());

    p.getEchiquier().setPièce(7, 3, null);
    assertTrue(p.matérielInsuffisant());

    p.getEchiquier().setPièce(5, 3, new Fou("NOIR", 5, 3));
    assertTrue(p.matérielInsuffisant());

    p.getEchiquier().setPièce(5, 3, null);
    assertTrue(p.matérielInsuffisant());
}

@Test
public void testEchecEtMat() {
    Partie p = new Partie("Humain", "Humain");
    p.jouer("e7e5");
    p.jouer("d1e5");
    assertFalse(p.échecEtMat(p.getEchiquier().getRoiNoir()));

    p.jouer("e8e7");
    assertTrue(p.échecEtMat(p.getEchiquier().getRoiNoir()));

    p.jouer("e7e6");
    assertFalse(p.échecEtMat(p.getEchiquier().getRoiNoir()));

    p.jouer("h1h5");
    assertTrue(p.échecEtMat(p.getEchiquier().getRoiNoir()));
}

@Test
public void abandon() {
    Partie p = new Partie("Humain", "Humain");
    assertTrue(p.abandon(""));
    assertFalse(p.abandon("test"));
}

```



```

        // matériel insuffisant
        for(int i = 0; i < 8; ++i) {
            for(int j = 0; j < 8; ++j) {
                p.getEchiquier().setPièce(i, j, null);
            }
        }
        p.getEchiquier().setPièce(0, 4, new Roi("NOIR", 0, 4));
        p.getEchiquier().setPièce(7, 4, new Roi("BLANC", 7, 4));
        p.getEchiquier().setPièce(7, 3, new Cavalier("BLANC", 7,
3));

        assertTrue(p.matérielInsuffisant());

        attendu = "partie nulle, matériel insuffisant";
        assertEquals(p.toStringFin("test"), attendu);

        // echet et mat blanc
        p = new Partie("Humain", "Humain");
        p.jouer("e7e5");
        p.jouer("d1e5");
        p.jouer("e8e7");
        assertTrue(p.échecEtMat(p.getEchiquier().getRoiNoir()));

        attendu = "les Blancs ont gagné par Echec Et Mat";
        assertEquals(p.toStringFin("test"), attendu);

        // echec et mat noir
        p = new Partie("Humain", "Humain");
        p.jouer("e2e4");
        p.jouer("d8e4");
        p.jouer("e1e2");
        assertTrue(p.échecEtMat(p.getEchiquier().getRoiBlanc()));

        attendu = "les Noirs ont gagné par Echec Et Mat";
        assertEquals(p.toStringFin("test"), attendu);
    }
}

```



PièceTest.java

---

```
package test.piecesTest;

import static org.junit.Assert.assertTrue;

import org.junit.Test;

import echec.Coordonnée;
import echec.Echiquier;
import echec.pieces.Pièce;

public class PièceTest {

    @Test
    public void testDéplacer() {
        Echiquier e = new Echiquier();
        Pièce p = new Pièce("BLANC", 4, 4);
        e.setPièce(4, 4, p);
        Coordonnée départ = new Coordonnée(4, 4);
        Coordonnée arrivée = new Coordonnée(0, 0);

        p.déplacer(e, départ, arrivée);
        assertTrue(e.getPièce(4, 4) == null);
        assertTrue(e.getPièce(0, 0) == p);
    }
}
```





## FouTest.java

```
package test.piecesTest;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.Test;

import echec.Echiquier;
import echec.pieces.Fou;

public class FouTest {
    @Test
    public void testEstPossible() {
        Echiquier e = new Echiquier();
        Fou f = new Fou("BLANC", 3, 4);

        assertTrue(f.estPossible(e, 2, 3));
        assertFalse(f.estPossible(e, 6, 7));
        assertTrue(f.estPossible(e, 1, 2));
        assertFalse(f.estPossible(e, 4, 2));
    }

    @Test
    public void testCoupsPossibles() {
        Echiquier e = new Echiquier();
        Fou f = new Fou("BLANC", 3, 4);

        String attendu = "[[2, 3], [1, 2], [2, 5], [1, 6], "
            + "[4, 5], [5, 6], [4, 3], [5, 2]]";
        assertEquals(f.coupsPossibles(e).toString(), attendu);
    }

    @Test
    public void testGetSymbole() {
        Fou f = new Fou("BLANC", 0, 0);
        assertEquals(f.getSymbole(), 'F');

        f = new Fou("NOIR", 0, 0);
        assertEquals(f.getSymbole(), 'f');
    }
}
```

## PionTest.java

```
package test.piecesTest;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.Test;

import echec.Echiquier;
import echec.pieces.Pion;

public class PionTest {
    @Test
    public void testEstPossible() {
        Echiquier e = new Echiquier();
        Pion p = new Pion("BLANC", 4, 4);

        assertTrue(p.estPossible(e, 2, 4));
        assertTrue(p.estPossible(e, 3, 4));
        assertFalse(p.estPossible(e, 3, 5));
        assertFalse(p.estPossible(e, 5, 4));

        p.setPremierCoup(false);
        assertFalse(p.estPossible(e, 2, 4));

        p = new Pion("BLANC", 2, 4);
        assertFalse(p.estPossible(e, 1, 4));
    }

    @Test
    public void testCoupsPossibles() {
        Echiquier e = new Echiquier();
        Pion p = new Pion("BLANC", 4, 4);

        String attendu = "[[3, 4], [2, 4]]";
        assertEquals(p.coupsPossibles(e).toString(), attendu);

        p.setPremierCoup(false);
        attendu = "[[3, 4]]";
        assertEquals(p.coupsPossibles(e).toString(), attendu);

        p = new Pion("BLANC", 2, 4);
        attendu = "[[1, 5], [1, 3]]";
        assertEquals(p.coupsPossibles(e).toString(), attendu);
    }
}
```









## RoiTest.java

```
package test.piecesTest;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.Test;

import echec.Coordonnée;
import echec.Echiquier;
import echec.pieces.Roi;

public class RoiTest {
    @Test
    public void testEstPossible() {
        Echiquier e = new Echiquier();
        Roi r = new Roi("BLANC", 4, 4);

        assertTrue(r.estPossible(e, 3, 3));
        assertTrue(r.estPossible(e, 3, 4));
        assertTrue(r.estPossible(e, 3, 5));
        assertTrue(r.estPossible(e, 4, 3));
        assertTrue(r.estPossible(e, 4, 5));
        assertTrue(r.estPossible(e, 5, 3));
        assertTrue(r.estPossible(e, 5, 4));
        assertTrue(r.estPossible(e, 5, 4));

        assertFalse(r.estPossible(e, 5, 6));
        assertFalse(r.estPossible(e, 6, 5));
        assertFalse(r.estPossible(e, 2, 2));
    }

    @Test
    public void testCoupsPossibles() {
        Echiquier e = new Echiquier();
        Roi r = new Roi("BLANC", 4, 4);
        e.setPièce(4, 4, r);

        String attendu = "[[3, 3], [3, 4], [3, 5], [4, 3], "
            + "[4, 5], [5, 3], [5, 4], [5, 5]]";
        assertEquals(r.coupsPossibles(e).toString(), attendu);
    }
}
```



```
        départ = new Coordonnée(0, 3);
        arrivée = new Coordonnée(5, 4);
        e.déplacer(départ, arrivée);
        assertTrue(e.getRoiNoir().peutEtreProtégé(e));

        départ = new Coordonnée(5, 4);
        arrivée = new Coordonnée(5, 7);
        e.déplacer(départ, arrivée);
        assertTrue(e.getRoiNoir().peutEtreProtégé(e));
    }
}
```



## **Tâches à accomplir**

Pour intégrer un IA intelligent, il faut modifier les méthodes jouerIA dans la classe Joueur.  
Pour la promotion du pion il faut créer une nouvelle méthode dans la classe Pion et il faudra vérifier si un Pion arrive à la fin du plateau dans la méthode déplacer de la classe Echiquier.

Pour le roque, il faudra ajouter des attributs à la classe Roi et Tour pour vérifier si ces pièces n'avaient pas été déplacées auparavant. Il faudra ensuite ajouter une méthode dans la classe Echiquier pour effectuer ce déplacement.

## Bilan du projet

Nous avons réussi à coder ce qui était demandé et à ajouter les autres types de pièce et des règles supplémentaires (matériel insuffisant et les cinquante coups). Nous avons réussi à faire en sorte que tout marche et à utiliser l'héritage.

En revanche, il aurait fallu améliorer la structure du programme. Notre programme présente trop de dépendances : l'échiquier comporte deux attributs Roi pour pouvoir vérifier s'il est en échec ou en échec et mat sur le plateau. Pour l'échec et mat il faut pouvoir vérifier les coups possibles de chaque pièce et vérifier si le roi sera en échec au prochain coup. Pour ce faire, on a déplacé le roi en essayant chacun de ses coups possibles pour vérifier s'il sera en échec, et on a annulé ce dernier coup pour le remettre à sa place. S'il est en échec au prochain tour et qu'il ne peut être protégé, alors il y a échec et mat.

On a besoin de connaître de quel roi il s'agit sur l'échiquier alors on a dû faire cette dépendance. Vérifier l'échec et mat et le faire fonctionner était une des difficultés du projet car cela a aussi impacté la structure du projet, qui n'est pas assez stable.

Une autre des difficultés pour l'une de nous deux était de bien comprendre les règles d'échec, de pat et de mat, n'ayant pas beaucoup joué aux échecs. Comprendre la logique et retranscrire la vérification des règles en code était donc plus complexe et confus.

Pour améliorer la structure nous avons pensé à créer une interface IPièce en rendant la classe Pièce abstraite pour réduire les dépendances avec l'échiquier. Les méthodes `estPossible(Echiquier e, int x, int y)`, `coupsPossibles(Echiquier e)` et `getSymbole()` de Pièce n'ont pas de code dans la classe Pièce et donc peuvent être rendues abstraites. On les spécialise ensuite ces méthodes dans les autres classes comme on l'a fait.

Mais le problème se pose ensuite pour les deux rois, nous avons deux attributs Roi dans la classe Echiquier afin de vérifier l'échec et le mat. Nous n'avons pas réussi à enlever cette dépendance. L'idéal aurait été de faire une classe de fabrique de Pièces mais dans le temps qu'il nous restait, nous n'avons pas pu mettre en place cette solution.

Nous avons aussi essayé et réfléchi à comment faire une IA plus intelligente mais nous n'avons pas réussi.

Vérifier le déplacement des pièces était sans doute le moins compliqué, une fois que nous l'avons fait pour deux types de pièces. Mais une partie d'échecs ne ressemble jamais à une autre alors vérifier tous les cas d'échec, de pat ou d'échec et mat est un peu plus complexe. Il faut vraiment comprendre la logique derrière et bien organiser la structure du programme.

## Code Source

## Appli.java

```
package appli;

import java.util.Scanner;
import echec.Partie;

public class Appli {
    public static void main(String[] args) throws InterruptedException
    {
        Partie p = null;
        Scanner sc = new Scanner(System.in);
        String s = "tmp";

        System.out.println(toStringMenu());
        while(!(s.equals("1") || s.equals("2") || s.equals("3"))){
            System.out.print("> ");
            s = sc.nextLine();
        }

        if(s.equals("1")) {
            p = new Partie("Humain", "Humain");
            while (!p.fin()) {
                System.out.println(p.getEchiquier());
                System.out.print("> ");
                s = sc.nextLine();

                while (!p.abandon(s) && !(estValide(s)
                    && p.estPossible(s) &&
!p.donneEchec(s))) {
                    System.out.print("#> ");
                    s = sc.nextLine();
                }
                if(p.abandon(s))
                    break;
                p.jouer(s);
            }
        }

        if(s.equals("2")) {
            p = new Partie("Humain", "IA");
            while (!p.fin()) {
```

```

        System.out.println(p.getEchiquier());
        System.out.print("> ");
        s = sc.nextLine();

        while (!p.abandon(s) && !(estValide(s)
            && p.estPossible(s) &&
!p.donneEchec(s))) {

            System.out.print("#> ");
            s = sc.nextLine();
        }
        if(p.abandon(s))
            break;
        p.jouer(s);
        System.out.println(p.getEchiquier());
        if(!p.fin()) {
            p.jouer(s);
            Thread.sleep(1000);
        }
    }
}

if(s.equals("3")) {
    p = new Partie("IA", "IA");
    while (!p.fin()) {
        System.out.println(p.getEchiquier());
        p.jouer(s);

        Thread.sleep(1000);
    }
}

System.out.println(p.getEchiquier());
System.out.print(p.toStringFin(s));
}

/**
 * Affiche le menu pour choisir le mode de jeu
 * @return s le menu
 */
public static String toStringMenu() {
    String s = "Saisissez un chiffre selon le mode désiré : \n";
    s += "1. Humain VS Humain\n";
    s += "2. Humain VS IA\n";
    s += "3. IA VS IA\n";
    return s;
}

/**

```













```

    /**
     * Retourne la dernière capture
     * @return priseDernierCoup
     */
    public Pièce getPriseDernierCoup() {
        return priseDernierCoup;
    }

    /**
     * Met à jour la dernière capture
     * @param p la nouvelle capture
     */
    public void setPriseDernierCoup(Pièce p) {
        this.priseDernierCoup = p;
    }

    /**
     * Déplace la pièce d'une position de départ à la position
     d'arrivée
     * @param départ
     * @param arrivée
     */
    public void déplacer(Coordonnée départ, Coordonnée arrivée) {
        int x = départ.getLigne(), ligne = arrivée.getLigne();
        int y = départ.getColonne(), colonne = arrivée.getColonne();

        priseDernierCoup = null;
        if(getPièce(ligne, colonne) != null)
            priseDernierCoup = this.echiquier[ligne][colonne];

        this.echiquier[x][y].déplacer(this, départ, arrivée);
    }

    /**
     * Annule le dernier coup joué
     * @param départ les coordonnées de départ du dernier coup
     * @param arrivée les coordonnées d'arrivée du dernier coup
     */
    public void annulerDernierCoup(Coordonnée départ, Coordonnée
    arrivée) {
        int x = arrivée.getLigne();
        int y = arrivée.getColonne();

        this.echiquier[x][y].déplacer(this, arrivée, départ);
        if(priseDernierCoup != null)
            setPièce(arrivée.getLigne(), arrivée.getColonne(),
            priseDernierCoup);
        priseDernierCoup = null;
    }

```

```

/**
 * Retourne toutes les pièces blanches du plateau
 * @return p la liste des pièces blanches
 */
public ArrayList<Pièce> getPiècesBlanches() {
    ArrayList<Pièce> p = new ArrayList<>();
    for(int i = 0; i < MAX; ++i) {
        for(int j = 0; j < MAX; ++j) {
            if(getPièce(i, j) != null
                && getPièce(i, j).getCouleur() == "BLANC")
                p.add(getPièce(i, j));
        }
    }
    return p;
}

/**
 * Retourne toutes les pièces noires du plateau
 * @return p la liste des pièces noires
 */
public ArrayList<Pièce> getPiècesNoires() {
    ArrayList<Pièce> p = new ArrayList<>();
    for(int i = 0; i < MAX; ++i) {
        for(int j = 0; j < MAX; ++j) {
            if(getPièce(i, j) != null
                && getPièce(i, j).getCouleur() == "NOIR")
                p.add(getPièce(i, j));
        }
    }
    return p;
}

/**
 * Affiche le plateau et les pièces sous forme de grille
 * @return s la grille
 */
public String toString() {
    String lettres = ("    a    b    c    d    e    f    g    h    \n");
    String trait = "    --- --- --- --- --- --- --- --- \n";
    String s = lettres + trait;
    // ligne
    for (int i = MIN; i <= MAX; ++i) {
        s += MAX - i + 1 + " | ";
        // colonne
        for (int j = MIN; j <= MAX; ++j) {
            if(this.echiquier[i-1][j-1] != null)
                s += this.echiquier[i-1][j-1].getSymbole();
            else
                s += " ";
            s += " | ";
        }
    }
}

```

```
        s += (MAX-i+1) + "\n";
        s += trait;
    }
    s += lettres;
    return s;
}

}
```



## Joueur.java

```
package echec;

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

import echec.pieces.Pièce;

public class Joueur {
    private String type;
    private String couleur;

    /**
     * Constructeur d'un Joueur
     * @param type
     * @param couleur
     */
    public Joueur(String type, String couleur) {
        this.type = type;
        this.couleur = couleur;
    }

    /**
     * Retourne le type de la pièce
     * @return type
     */
    public String getType() {
        return type;
    }

    /**
     * Fait jouer l'IA
     * @param p la partie actuelle
     */
    public void jouerIA(Partie p) {
        String s = getCoupIA(p);
        while(p.donneEchec(s)) {
            s = getCoupIA(p);
        }
        Coordonnée départ = Coordonnée.stringToInt(s)[0];
        Coordonnée arrivée = Coordonnée.stringToInt(s)[1];
        p.getEchiquier().déplacer(départ, arrivée);
    }
}
```



## Partie.java

```
package echec;

import java.util.ArrayList;

import echec.pieces.Pièce;
import echec.pieces.Roi;

public class Partie {
    private Joueur blanc;
    private Joueur noir;
    private Echiquier echiquier;
    private boolean tourDeBlanc;
    private int nbCoupsSansPrises;

    /**
     * Constructeur d'une Partie
     * @param typeBlanc Humain ou IA pour le joueur blanc
     * @param typeNoir Humain ou IA pour le joueur noir
     */
    public Partie(String typeBlanc, String typeNoir) {
        if(typeBlanc.equals("Humain"))
            blanc = new Joueur("Humain", "BLANC");
        else
            blanc = new Joueur("IA", "BLANC");

        if(typeNoir.equals("Humain"))
            noir = new Joueur("Humain", "NOIR");
        else
            noir = new Joueur("IA", "NOIR");

        this.echiquier = new Echiquier();
        this.tourDeBlanc = true;
        this.nbCoupsSansPrises = 0;
    }

    /**
     * Vérifie si c'est au tour du joueur blanc
     * @return tourDeBlanc
     */
    public boolean isTourDeBlanc() {
        return tourDeBlanc;
    }
}
```

```

/**
 * Passe d'un tour à l'autre
 * @param b le booléen pour changer le tour
 */
public void setTourDeBlanc(boolean b) {
    this.tourDeBlanc = b;
}

/**
 * Retourne l'échiquier
 * @return échiquier
 */
public Echiquier getEchiquier() {
    return echiquier;
}

/**
 * Le joueur saisit son coup et déplace la pièce
 * @param s le coup saisi
 */
public void jouer(String s) {
    if(tourDeBlanc) {
        if(blanc.getType() == "Humain") {
            Coordonnée départ =
Coordonnée.stringToInt(s)[0];
            Coordonnée arrivée =
Coordonnée.stringToInt(s)[1];
            echiquier.déplacer(départ, arrivée);
        }
        else
            blanc.jouerIA(this);
    }
    else {
        if(noir.getType() == "Humain") {
            Coordonnée départ =
Coordonnée.stringToInt(s)[0];
            Coordonnée arrivée =
Coordonnée.stringToInt(s)[1];
            echiquier.déplacer(départ, arrivée);
        }
        else
            noir.jouerIA(this);
    }
    if(echiquier.getPriseDernierCoup() == null)
        nbCoupsSansPrises += 1;
    else
        nbCoupsSansPrises = 0;
    if(isTourDeBlanc())
        tourDeBlanc = false;
    else
        tourDeBlanc = true;
}

```

```

    }

    /**
     * Vérifie si le coup saisi est possible
     * @param s le coup saisi
     * @return true si le coup saisi est possible
     */
    public boolean estPossible(String s) {
        Coordonnée départ = Coordonnée.stringToInt(s)[0];
        Coordonnée arrivée = Coordonnée.stringToInt(s)[1];
        Pièce p = echiquier.getPièce(départ.getLigne(),
départ.getColonne());
        if(p == null)
            return false;
        if(!p.estPossible(echiquier, arrivée.getLigne(),
arrivée.getColonne())
            || (p.getCouleur().equals("NOIR") && tourDeBlanc)
            || (p.getCouleur().equals("BLANC") && !tourDeBlanc))
            return false;
        return true;
    }

    /**
     * Vérifie si le coup joué met en échec le roi du joueur actif
     * @param s le coup saisi
     * @return b true si le coup met en échec le roi
     */
    public boolean donneEchec(String s) {
        Coordonnée départ = Coordonnée.stringToInt(s)[0];
        Coordonnée arrivée = Coordonnée.stringToInt(s)[1];
        boolean b = false;

        if(echiquier.getPièce(arrivée.getLigne(), arrivée.getColonne())
!= null)

echiquier.setPriseDonneEchec(echiquier.getPièce(arrivée.getLigne(),
arrivée.getColonne()));
        echiquier.déplacer(départ, arrivée);

        if(tourDeBlanc
            && echiquier.getRoiBlanc().échec(echiquier))
            b = true;

        if(!tourDeBlanc
            && echiquier.getRoiNoir().échec(echiquier))
            b = true;

        echiquier.annulerDernierCoup(départ, arrivée);
        if(echiquier.getPriseDonneEchec() != null)
            echiquier.setPièce(arrivée.getLigne(), arrivée.getColonne(),
echiquier.getPriseDonneEchec());
    }

```

```

        echiquier.setPriseDonneEchec(null);
        return b;
    }

    /**
     * Vérifie si le joueur n'a plus aucun coup à jouer
     * @return true s'il n'a plus aucun coup possible
     */
    private boolean pat() {
        for(int i = 0; i < Echiquier.MAX; ++i) {
            for(int j = 0; j < Echiquier.MAX; ++j) {
                Pièce p = echiquier.getPièce(i, j);
                if(p != null) {
                    if(tourDeBlanc) {
                        if(p.getCouleur() == "BLANC"
                            &&
                            !p.coupsPossibles(echiquier).isEmpty())
                            return false;
                    }
                    else {
                        if(p.getCouleur() == "NOIR"
                            &&
                            !p.coupsPossibles(echiquier).isEmpty())
                            return false;
                    }
                }
            }
        }
        return true;
    }

    /**
     * Vérifie s'il y a eu 50 coups sans prise ou non
     * @return true s'il y a eu 50 coups sans prise
     */
    private boolean cinquanteCoups() {
        return (nbCoupsSansPrises == 50);
    }

    /**
     * Vérifie s'il y a insuffisance de matériel ou non
     * @return true s'il y a insuffisance de matériel
     */
    private boolean matérielInsuffisant() {
        ArrayList<Pièce> blancs = echiquier.getPiècesBlanches();
        ArrayList<Pièce> noirs = echiquier.getPiècesNoires();

        if(blancs.size() == 1 && noirs.size() == 1)
            return true;

        if(blancs.size() == 1 && noirs.size() == 2)

```



```

/**
 * Vérifie si la partie est finie ou non
 * @return true si elle est finie
 */
public boolean fin() {
    if(this.pat() || cinquanteCoups() || matérielInsuffisant()
        || (échecEtMat(echiquier.getRoiBlanc()) &&
tourDeBlanc)
        || (échecEtMat(echiquier.getRoiNoir()) &&
!tourDeBlanc))
        return true;
    return false;
}

/**
 * Vérifie si un joueur abandonne la partie
 * @param s le coups saisi
 * @return true si le coup saisie est vide
 */
public boolean abandon(String s) {
    return (s.length() == 0);
}

/**
 * Affiche le résultat de fin de partie selon la situation
 * @param s le dernier coup saisi
 * @return fin le résultat
 */
public String toStringFin(String s) {
    String fin = "";

    if(s.length() == 0) {
        fin += "partie finie, abandon des ";
        if(tourDeBlanc)
            fin += "Blancs";
        else
            fin += "Noirs";
    }

    else if(cinquanteCoups())
        fin += "partie nulle, règle des Cinquantes Coups";

    else if(matérielInsuffisant())
        fin += "partie nulle, matériel insuffisant";

    else if(pat())
        fin += "partie nulle, PAT";

    else if(échecEtMat(echiquier.getRoiBlanc()))
        fin += "les Noirs ont gagné par Echec Et Mat";
}

```



```
        else if (échecEtMat (echiquier.getRoiNoir()))  
            fin += "les Blancs ont gagné par Echec Et Mat";  
  
        return fin;  
    }  
}
```

Pièce.java

---

```
package echec.pieces;

import java.util.ArrayList;

import echec.Coordonnée;
import echec.Echiquier;

public class Pièce {
    protected String couleur;
    protected String type;
    protected int ligne;
    protected int colonne;

    /**
     * Constructeur d'une Pièce
     * @param couleur
     * @param ligne
     * @param colonne
     */
    public Pièce(String couleur, int ligne, int colonne) {
        this.couleur = couleur;
        setCoordonnée(ligne, colonne);
    }

    /**
     * Retourne tous les coups possibles d'une pièce sur l'échiquier
     * @param e l'échiquier actuel
     * @return null
     */
    public ArrayList<Coordonnée> coupsPossibles(Echiquier e){
        return null;
    }

    /**
     * Retourne la couleur d'une pièce
     * @return couleur
     */
    public String getCouleur() {
        return couleur;
    }
}
```

```

/**
 * Retourne la ligne d'une pièce
 * @return ligne
 */
public int getLigne() {
    return ligne;
}

/**
 * Retourne la colonne d'une pièce
 * @return colonne
 */
public int getColonne() {
    return colonne;
}

/**
 * Retourne le type de la pièce
 * @return type
 */
public String getType() {
    return type;
}

/**
 * Modifie les coordonnées ligne et colonne
 * @param ligne
 * @param colonne
 */
public void setCoordonnée(int ligne, int colonne) {
    this.ligne = ligne;
    this.colonne = colonne;
}

/**
 * Déplace la pièce aux coordonnées d'arrivée
 * @param e l'échiquier actuel
 * @param départ
 * @param arrivée
 */
public void déplacer(Echiquier e, Coordonnée départ, Coordonnée
arrivée) {
    int x = départ.getLigne(), ligne = arrivée.getLigne();
    int y = départ.getColonne(), colonne = arrivée.getColonne();

    if(this.type == "PION") {
        Pion p = (Pion) this;
        p.setPremierCoup(false);
    }
}

```





Cavalier.java

---

```
package echec.pieces;

import java.util.ArrayList;

import echec.Coordonn  e;
import echec.Echiquier;

public class Cavalier extends Pi  ce {

    /**
     * Constructeur d'un Cavalier
     * @param couleur
     * @param ligne
     * @param colonne
     */
    public Cavalier(String couleur, int ligne, int colonne) {
        super(couleur, ligne, colonne);
        this.type = "CAVALIER";
    }

    /**
     * V  rifie qu'un coup est possible pour le cavalier
     * @param e l'  chiquier actuel
     * @param x ligne
     * @param y colonne
     * @return true si le coup est possible
     */
    @Override
    public boolean estPossible(Echiquier e, int x, int y) {
        if(e.getPi  ce(x, y) != null
            && e.getPi  ce(x, y).getCouleur() == this.getCouleur())
            return false;

        // 2 bas + 1 gauche
        if(ligne + 2 == x && colonne - 1 == y)
            return true;

        // 2 bas + 1 droite
        if(ligne + 2 == x && colonne + 1 == y)
            return true;

        // 2 haut + 1 droite
        if(ligne - 2 == x && colonne + 1 == y)
            return true;
```

```

        // 2 haut + 1 gauche
        if(ligne - 2 == x && colonne - 1 == y)
            return true;

        // 2 droite + 1 bas
        if(ligne + 1 == x && colonne + 2 == y)
            return true;

        // 2 droite + 1 haut
        if(ligne - 1 == x && colonne + 2 == y)
            return true;

        // 2 gauche + 1 bas
        if(ligne + 1 == x && colonne - 2 == y)
            return true;

        // 2 gauche + 1 haut
        if(ligne - 1 == x && colonne - 2 == y)
            return true;

        return false;
    }

    /**
     * Retourne tous les coups possibles d'un cavalier sur
     l'échiquier
     * @param e l'échiquier actuel
     * @return coups la liste des coups possibles
     */
    @Override
    public ArrayList<Coordonnée> coupsPossibles(Echiquier e){
        ArrayList<Coordonnée> coups = new ArrayList<Coordonnée>();

        // 2 bas + 1 gauche
        if(ligne + 2 < Echiquier.MAX && colonne - 1 >= Echiquier.MIN - 1
            && (e.getPièce(ligne + 2, colonne - 1) == null
            || (e.getPièce(ligne + 2, colonne - 1) != null
            && e.getPièce(ligne + 2, colonne - 1).getCouleur() !=
this.getCouleur()))))
            coups.add(new Coordonnée(ligne + 2, colonne - 1));

        // 2 bas + 1 droite
        if(ligne + 2 < Echiquier.MAX && colonne + 1 < Echiquier.MAX
            && (e.getPièce(ligne + 2, colonne + 1) == null
            || (e.getPièce(ligne + 2, colonne + 1) != null
            && e.getPièce(ligne + 2, colonne + 1).getCouleur() !=
this.getCouleur()))))
            coups.add(new Coordonnée(ligne + 2, colonne + 1));
    }

```

```

        // 2 haut + 1 droite
        if(ligne - 2 >= Echiquier.MIN - 1 && colonne + 1 < Echiquier.MAX
            && (e.getPi  ce(ligne - 2, colonne + 1) == null
            || (e.getPi  ce(ligne - 2, colonne + 1) != null
            && e.getPi  ce(ligne - 2, colonne + 1).getCouleur() !=
this.getCouleur()))))
            coups.add(new Coordonn  e(ligne - 2, colonne + 1));

        // 2 haut + 1 gauche
        if(ligne - 2 >= Echiquier.MIN - 1 && colonne - 1 >= Echiquier.MIN
- 1
            && (e.getPi  ce(ligne - 2, colonne - 1) == null
            || (e.getPi  ce(ligne - 2, colonne - 1) != null
            && e.getPi  ce(ligne - 2, colonne - 1).getCouleur() !=
this.getCouleur()))))
            coups.add(new Coordonn  e(ligne - 2, colonne - 1));

        // 2 droite + 1 bas
        if(ligne + 1 < Echiquier.MAX && colonne + 2 < Echiquier.MAX
            && (e.getPi  ce(ligne + 1, colonne + 2) == null
            || (e.getPi  ce(ligne + 1, colonne + 2) != null
            && e.getPi  ce(ligne + 1, colonne + 2).getCouleur() !=
this.getCouleur()))))
            coups.add(new Coordonn  e(ligne + 1, colonne + 2));

        // 2 droite + 1 haut
        if(ligne - 1 >= Echiquier.MIN - 1 && colonne + 2 < Echiquier.MAX
            && (e.getPi  ce(ligne - 1, colonne + 2) == null
            || (e.getPi  ce(ligne - 1, colonne + 2) != null
            && e.getPi  ce(ligne - 1, colonne + 2).getCouleur() !=
this.getCouleur()))))
            coups.add(new Coordonn  e(ligne - 1, colonne + 2));

        // 2 gauche + 1 bas
        if(ligne + 1 < Echiquier.MAX && colonne - 2 >= Echiquier.MIN - 1
            && (e.getPi  ce(ligne + 1, colonne - 2) == null
            || (e.getPi  ce(ligne + 1, colonne - 2) != null
            && e.getPi  ce(ligne + 1, colonne - 2).getCouleur() !=
this.getCouleur()))))
            coups.add(new Coordonn  e(ligne + 1, colonne - 2));

        // 2 gauche + 1 haut
        if(ligne - 1 >= Echiquier.MIN - 1 && colonne - 2 >= Echiquier.MIN
- 1
            && (e.getPi  ce(ligne - 1, colonne - 2) == null
            || (e.getPi  ce(ligne - 1, colonne - 2) != null
            && e.getPi  ce(ligne - 1, colonne - 2).getCouleur() !=
this.getCouleur()))))
            coups.add(new Coordonn  e(ligne - 1, colonne - 2));

```



```
        return coups;
    }

    /**
     * Retourne le caractère représentant la pièce
     * @return le caractère
     */
    @Override
    public char getSymbole() {
        return (couleur.equals("BLANC") ? 'C':'c');
    }
}
```







```
        return true;
    }

    /**
     * Retourne le caractère représentant la pièce
     * @return le caractère
     */
    @Override
    public char getSymbole() {
        return (couleur.equals("BLANC") ? 'F':'f');
    }
}
```

## Pion.java

```
package echec.pieces;

import java.util.ArrayList;

import echec.Coordonnée;
import echec.Echiquier;

public class Pion extends Pièce {
    private boolean premierCoup;

    /**
     * Constructeur d'un Pion
     * @param couleur
     * @param ligne
     * @param colonne
     */
    public Pion(String couleur, int ligne, int colonne){
        super(couleur, ligne, colonne);
        this.premierCoup = true;
        this.type = "PION";
    }

    /**
     * Vérifie qu'un coup est possible pour le pion
     * @param e l'échiquier actuel
     * @param x ligne
     * @param y colonne
     * @return true si le coup est possible
     */
    @Override
    public boolean estPossible(Echiquier e, int x, int y) {
        // pion blanc
        if(this.couleur == "BLANC") {
            // devant
            if(ligne - 1 == x && colonne == y
                && e.getPièce(x, colonne) == null)
                return true;

            // devant droite
            if(ligne - 1 == x && colonne + 1 == y
                && e.getPièce(x, y) != null)
                if(e.getPièce(x, y).getCouleur() != this.couleur)
                    return true;
        }
    }
}
```



```

/**
 * Retourne tous les coups possibles d'un pion sur l'échiquier
 * @param e l'échiquier actuel
 * @return coups la liste des coups possibles
 */
@Override
public ArrayList<Coordonnée> coupsPossibles(Echiquier e) {
    ArrayList<Coordonnée> coups = new ArrayList<Coordonnée>();

    // pion blanc
    if(this.couleur == "BLANC") {
        // devant
        if(ligne - 1 >= Echiquier.MIN - 1
            && e.getPièce(ligne - 1, colonne) == null)
            coups.add(new Coordonnée(ligne - 1, colonne));

        // devant droite
        if(ligne - 1 >= Echiquier.MIN - 1 && colonne + 1 <
Echiquier.MAX
            && e.getPièce(ligne - 1, colonne + 1) != null)
            if(e.getPièce(ligne - 1, colonne + 1).getCouleur() !=
this.couleur)
                coups.add(new Coordonnée(ligne - 1, colonne +
1));

        // devant gauche
        if(ligne - 1 >= Echiquier.MIN - 1 && colonne - 1 >=
Echiquier.MIN - 1
            && e.getPièce(ligne - 1, colonne - 1) != null)
            if(e.getPièce(ligne - 1, colonne - 1).getCouleur() !=
this.couleur)
                coups.add(new Coordonnée(ligne - 1, colonne -
1));

        // 2 cases en avant
        if(ligne - 2 >= Echiquier.MIN - 1
            && premierCoup
            && e.getPièce(ligne - 2, colonne) == null
            && e.getPièce(ligne - 1, colonne) == null)
            coups.add(new Coordonnée(ligne - 2, colonne));
    }
    // pion noir
    else {
        // devant
        if(ligne + 1 < Echiquier.MAX
            && e.getPièce(ligne + 1, colonne) == null)
            coups.add(new Coordonnée(ligne + 1, colonne));
    }
}

```





## Reine.java

```
package echec.pieces;

import java.util.ArrayList;

import echec.Coordonnée;
import echec.Echiquier;

public class Reine extends Pièce {

    /**
     * Constructeur d'une Reine
     * @param couleur
     * @param ligne
     * @param colonne
     */
    public Reine(String couleur, int ligne, int colonne) {
        super(couleur, ligne, colonne);
        this.type = "REINE";
    }

    /**
     * Retourne tous les coups possibles d'une reine sur l'échiquier
     * @param e l'échiquier actuel
     * @return coups la liste des coups possibles
     */
    @Override
    public ArrayList<Coordonnée> coupsPossibles(Echiquier e) {
        ArrayList<Coordonnée> coups = new ArrayList<Coordonnée>();

        // haut
        for (int i = ligne + 1; i < Echiquier.MAX; ++i) {
            if(e.getPièce(i, colonne) != null
                && this.getCouleur() == e.getPièce(i,
colonne).getCouleur())
                break;
            coups.add(new Coordonnée(i, colonne));
            if(e.getPièce(i, colonne) != null)
                break;
        }
    }
}
```



```

        // nord-est
        for (int i = ligne - 1, j = colonne + 1;
            i >= Echiquier.MIN - 1 && j < Echiquier.MAX; --i, ++j)
        {
            if(e.getPi  ce(i, j) != null
                && this.couleur == e.getPi  ce(i,
j).getCouleur())
                break;
            coups.add(new Coordonn  e(i, j));
            if(e.getPi  ce(i, j) != null)
                break;
        }

        // sud-est
        for (int i = ligne + 1, j = colonne + 1;
            i < Echiquier.MAX && j < Echiquier.MAX; ++i, ++j) {
            if(e.getPi  ce(i, j) != null
                && this.couleur == e.getPi  ce(i,
j).getCouleur())
                break;
            coups.add(new Coordonn  e(i, j));
            if(e.getPi  ce(i, j) != null)
                break;
        }

        // sud-ouest
        for (int i = ligne + 1, j = colonne - 1;
            i < Echiquier.MAX && j >= Echiquier.MIN - 1; ++i, --j)
        {
            if(e.getPi  ce(i, j) != null
                && this.couleur == e.getPi  ce(i,
j).getCouleur())
                break;
            coups.add(new Coordonn  e(i, j));
            if(e.getPi  ce(i, j) != null)
                break;
        }

        return coups;
    }

```

```

/**
 * Vérifie qu'un coup est possible pour la reine
 * @param e l'échiquier actuel
 * @param x ligne
 * @param y colonne
 * @return true si le coup est possible
 */
@Override
public boolean estPossible(Echiquier e, int x, int y) {
    if(e.getPièce(x, y) != null
        && e.getPièce(x, y).getCouleur() == this.getCouleur())
        return false;

    if(!((colonne == y && ligne != x) || (colonne != y && ligne == x)
        || (Math.abs(ligne - x) == Math.abs(colonne - y))))
        return false;

    if(colonne == y) {
        // bas
        if(ligne < x) {
            for (int i = ligne + 1; i < x; ++i) {
                if(e.getPièce(i, colonne) != null)
                    return false;
            }
        }

        // haut
        if(ligne > x) {
            for (int i = ligne - 1; i > x; --i) {
                if(e.getPièce(i, colonne) != null)
                    return false;
            }
        }
    }

    if(ligne == x) {
        // droite
        if(colonne < y) {
            for (int i = colonne + 1; i < y; ++i) {
                if(e.getPièce(ligne, i) != null)
                    return false;
            }
        }

        //gauche
        if(colonne > y) {
            for (int i = colonne - 1; i > y; --i) {
                if(e.getPièce(ligne, i) != null)
                    return false;
            }
        }
    }
}

```

```

    }

    // nord-ouest
    if(ligne > x && colonne > y) {
        for (int i = ligne - 1, j = colonne - 1; i > x && j > y;
--i, --j) {
            if(e.getPi  ce(i, j) != null)
                return false;
        }
    }

    // nord-est
    if(ligne > x && colonne < y) {
        for (int i = ligne - 1, j = colonne + 1; i > x && j < y;
--i, ++j) {
            if(e.getPi  ce(i, j) != null)
                return false;
        }
    }

    // sud-est
    if(ligne < x && colonne < y) {
        for (int i = ligne + 1, j = colonne + 1; i < x && j < y;
++i, ++j) {
            if(e.getPi  ce(i, j) != null)
                return false;
        }
    }

    // sud-ouest
    if(ligne < x && colonne > y) {
        for (int i = ligne + 1, j = colonne - 1; i < x && j > y;
++i, --j) {
            if(e.getPi  ce(i, j) != null)
                return false;
        }
    }

    return true;
}

/**
 * Retourne le caract  re repr  sentant la pi  ce
 * @return le caract  re
 */
@Override
public char getSymbole() {
    return (couleur.equals("BLANC") ? 'D':'d');
}
}

```

## Roi.java

```
package echec.pieces;

import java.util.ArrayList;

import echec.Coordonnée;
import echec.Echiquier;

public class Roi extends Pièce {

    private static Pièce priseVérif = null;
    private static Pièce priseCoupPossible = null;

    /**
     * Constructeur d'un Roi
     * @param couleur
     * @param ligne
     * @param colonne
     */
    public Roi(String couleur, int ligne, int colonne) {
        super(couleur, ligne, colonne);
        this.type = "ROI";
    }

    /**
     * Vérifie qu'un coup est possible pour le roi
     * @param e l'échiquier actuel
     * @param x ligne
     * @param y colonne
     * @return true si le coup est possible
     */
    @Override
    public boolean estPossible(Echiquier e, int x, int y) {
        if(e.getPièce(x, y) != null
            && e.getPièce(x, y).getCouleur() == this.getCouleur())
            return false;

        // vertical ou horizontal
        if((ligne == (x + 1) && colonne == y)
            || (ligne == (x - 1) && colonne == y)
            || (ligne == x && colonne == (y + 1))
            || (ligne == x && colonne == (y - 1))) {
            return true;
        }
    }
}
```

```

        // diagonale
        if((ligne == (x + 1) && colonne == (y + 1))
            || (ligne == (x + 1) && colonne == (y - 1))
            || (ligne == (x - 1) && colonne == (y + 1))
            || (ligne == (x - 1) && colonne == (y - 1))) {
            return true;
        }

        return false;
    }

    /**
     * Retourne le caractère représentant la pièce
     * @return le caractère
     */
    @Override
    public char getSymbole() {
        return (couleur.equals("BLANC") ? 'R':'r');
    }

    /**
     * Retourne tous les coups possibles d'un roi sur l'échiquier
     * @param e l'échiquier en cours
     * @return coups
     */
    @Override
    public ArrayList<Coordonnée> coupsPossibles(Echiquier e) {
        ArrayList<Coordonnée> coups = new ArrayList<Coordonnée>();

        for (int i = ligne - 1; i <= ligne + 1 && i < Echiquier.MAX;
++i) {
            if(i < 0)
                ++i;
            for (int j = colonne - 1; j <= colonne + 1 && j <
Echiquier.MAX; ++j) {
                if(j < 0)
                    ++j;

                if(estPossible(e, i, j)) {
                    Coordonnée départ = new Coordonnée(ligne, colonne);
                    Coordonnée arrivée = new Coordonnée(i, j);
                    if(e.getPriseDernierCoup() != null)
                        priseCoupPossible = e.getPriseDernierCoup();
                    e.déplacer(départ, arrivée);
                    if(!échec(e))
                        coups.add(new Coordonnée(i, j));
                    e.annulerDernierCoup(départ, arrivée);
                    e.setPriseDernierCoup(priseCoupPossible);
                    priseCoupPossible = null;
                }
            }
        }
    }

```



```

    }
    return coups;
}

/**
 * Vérifie si le roi est en échec ou non (la pièce adverse peut se
 * déplacer à sa position)
 * @param e l'échiquier en cours
 * @return false si le roi n'est pas en échec
 */
public boolean échec(Echiquier e) {
    for (int i = 0; i < Echiquier.MAX; ++i) {
        for (int j = 0; j < Echiquier.MAX; ++j) {
            Pièce p = e.getPièce(i, j);
            if (p != null && !(ligne == i && colonne == j))
                if (p.couleur != this.couleur && p.estPossible(e,
ligne, colonne)) {
                    return true;
                }
        }
    }
    return false;
}

/**
 * Vérifie si le roi peut être protégé
 * @param e l'échiquier actuel
 * @return false s'il ne peut pas être protégé
 */
public boolean peutEtreProtégé(Echiquier e) {
    ArrayList<Pièce> pièces = new ArrayList<Pièce>();

    for (int i = 0; i < Echiquier.MAX; ++i) {
        for (int j = 0; j < Echiquier.MAX; ++j) {
            Pièce p = e.getPièce(i, j);

            if (!(ligne == i && colonne == j) && p != null)
                if (p.getCouleur() == this.getCouleur())
                    pièces.add(p);
        }
    }

    for (Pièce p : pièces) {
        ArrayList<Coordonnée> aTester = p.coupsPossibles(e);

        for (Coordonnée coord : aTester) {
            Coordonnée départ = new Coordonnée(p.getLigne(),
p.getColonne());
            Coordonnée arrivée = new Coordonnée(coord.getLigne(),
coord.getColonne());

```

```

        if (e.getPièce(arrivée.getLigne(), arrivée.getColonne()) !=
null)
            priseVérif = e.getPièce(arrivée.getLigne(),
arrivée.getColonne());
            e.déplacer(départ, arrivée);

            if(!this.échec(e)) {
                e.annulerDernierCoup(départ, arrivée);
                e.setPièce(arrivée.getLigne(), arrivée.getColonne(),
priseVérif);
                    return true;
            }

            e.annulerDernierCoup(départ, arrivée);
            e.setPièce(arrivée.getLigne(), arrivée.getColonne(),
priseVérif);
            priseVérif = null;
        }
        return false;
    }
}

```

## Tour.java

```
package echec.pieces;

import java.util.ArrayList;

import echec.Coordonnée;
import echec.Echiquier;

public class Tour extends Pièce {

    /**
     * Constructeur d'une Tour
     * @param couleur
     * @param ligne
     * @param colonne
     */
    public Tour(String couleur, int ligne, int colonne) {
        super(couleur, ligne, colonne);
        this.type = "TOUR";
    }

    /**
     * Vérifie qu'un coup est possible pour la tour
     * @param e l'échiquier actuel
     * @param x ligne
     * @param y colonne
     * @return true si le coup est possible
     */
    @Override
    public ArrayList<Coordonnée> coupsPossibles(Echiquier e) {
        ArrayList<Coordonnée> coups = new ArrayList<Coordonnée>();

        // bas
        for (int i = ligne + 1; i < Echiquier.MAX; ++i) {
            if(e.getPièce(i, colonne) != null
                && this.getCouleur() == e.getPièce(i,
colonne).getCouleur())
                break;
            coups.add(new Coordonnée(i, colonne));
            if(e.getPièce(i, colonne) != null)
                break;
        }
    }
}
```



```

        if(colonne == y) {
            // bas
            if(ligne < x) {
                for (int i = ligne + 1; i < x; ++i) {
                    if(e.getPièce(i, colonne) != null)
                        return false;
                }
            }

            // haut
            if(ligne > x) {
                for (int i = ligne - 1; i > x; --i) {
                    if(e.getPièce(i, colonne) != null)
                        return false;
                }
            }
        }

        if(ligne == x) {
            // droite
            if(colonne < y) {
                for (int i = colonne + 1; i < y; ++i) {
                    if(e.getPièce(ligne, i) != null)
                        return false;
                }
            }

            //gauche
            if(colonne > y) {
                for (int i = colonne - 1; i > y; --i) {
                    if(e.getPièce(ligne, i) != null)
                        return false;
                }
            }
        }

        return true;
    }

    /**
     * Retourne le caractère représentant la pièce
     * @return le caractère
     */
    @Override
    public char getSymbole() {
        return (couleur.equals("BLANC") ? 'T':'t');
    }
}

```