2020-2021

Université de Paris

PROJET BPO 2

Tracy HONG

Nahean BADAR

Athana KUMARAKULASINGAM

Jeu Echecs

—

Rapport de Projet

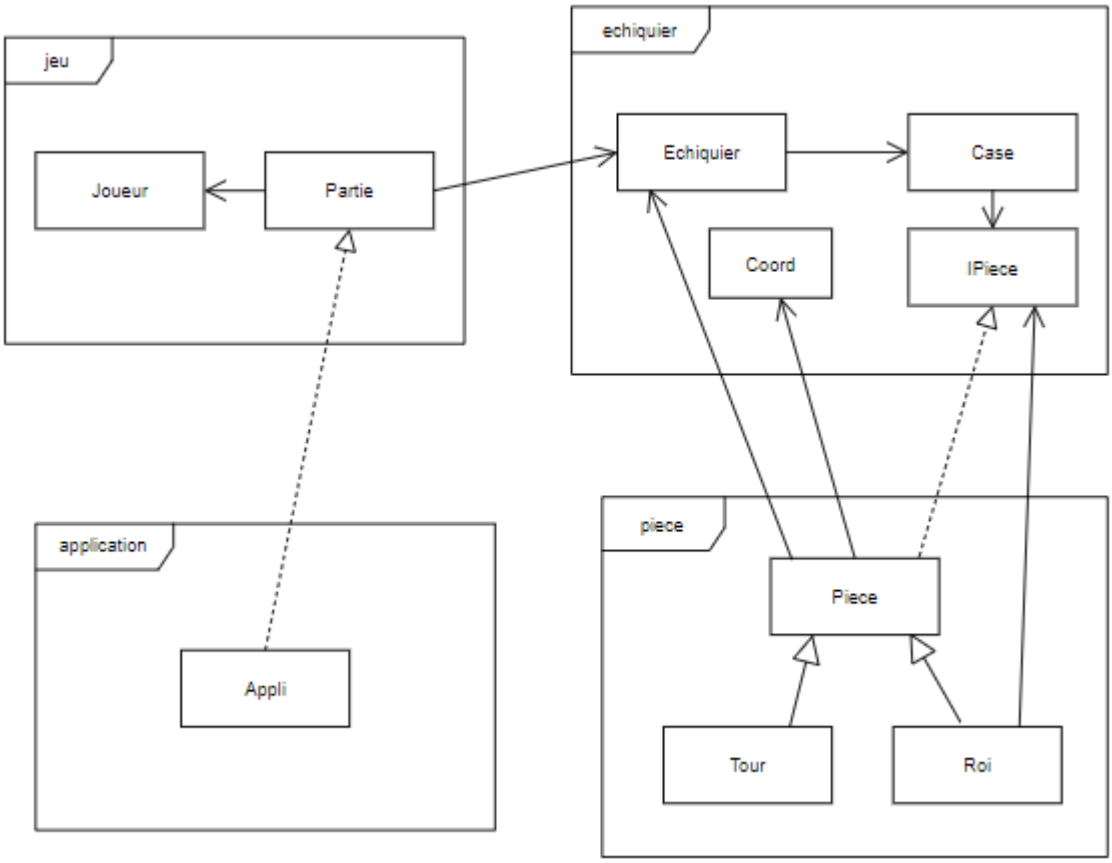
Université de Paris

Table des matières

Introduction :

Ce projet consiste à développer un jeu d'échecs à partir d'une certaine position de départ. Deux joueurs s'affrontent et pour cela il y a plusieurs combinaisons possibles : 2 joueurs humains, 1 joueur humain face à un algorithme ou encore 2 algorithmes. Dans notre cas, nos deux joueurs sont des joueurs humains. Nous n'avons pas pu faire les autres combinaisons possibles. En ce qui concerne les pièces dans l'échiquier, nous avons le roi et la tour. Le joueur blanc, qui commence la partie, possède un roi qui est représentée par un "R" majuscule et une tour représentée par un "T" majuscule. Le joueur adverse est le joueur noir qui possède un roi représenter en "r" minuscule. Par manque de temps nous n'avons pas pris en compte les autres pièces tels que le pion, la dame ou le cavalier. Cependant, nous avons bien respecté l'architecture du jeu afin de faciliter, dans le futur, l'ajout de nouveaux types de pièces. Pour chaque coup, il faut d'abord préciser les coordonnées de la case de départ (donc là où se trouve le joueur courant) puis celle de la case d'arrivée (donc là où on souhaite déplacer la pièce). Si le saisie n'est pas respecté dans cet ordre, le coup est rejeté et signalé. Durant la partie, pour chaque joueur et ses pièces, il y a des règles à respecter. Par exemple, le joueur peut déplacer sa tour en horizontale ou en verticale d'autant de case qu'il le souhaite. Cependant il ne peut déplacer le roi que d'une case mais dans n'importe quel direction (vers le haut, vers le bas, sur la gauche, sur la droite ou en diagonale). Notre jeu se termine lorsqu'il y a échec et mat, donc le roi ne peut plus se déplacer ou encore, la partie est PAT où le roi n'est pas attaqué mais il ne peut plus jouer sans se mettre en échec. Dans ces cas, la partie se termine et le programme affiche un message avec le joueur gagnant. On peut bien évidemment abandonner la partie si on le souhaite. Il suffit d'écrire "stop". Si on veut déclarer une proposition de partie nulle, il suffit d'écrire "null" dans la saisie.

Diagramme de classe (d'architecture) :



Tests unitaires :

Nous n'avons pas fait plus de tests. A chaque fois qu'un test ne fonctionnait pas, nous nous sommes concentrés dessus pour qu'il fonctionne et pour qu'on trouve le bug.

Nous avons fait beaucoup de tests intermédiaires, c'est-à-dire les méthodes qui étaient private pour qu'on puisse bien trouver la source de notre bug. Etant donné que notre code est découpé en plusieurs petites méthodes, il nous fallait faire ceci pour simplifier notre compréhension.

Voir code en Annexe.

On a donc fait 3 classes de tests au total : dans un premier temps tous les tests pour la saisie, puis les tests pour tout ce qui est déplacement et enfin une classe de test pour la partie du jeu.

Test de la saisie : tous les tests passent

- Saisie
- Vérification saisie
- Abandon
- Proposition de nul

Test des déplacements : tous les tests passent

- Déplacer
- Manger
- Coup
- Toutes les vérifications intermédiaires

Tests de partie, simulation de fin de partie : tous les tests passent

- Echec et mat
- Pat
- Fin de partie

Bilan du projet :

Pendant l'élaboration de ce projet nous avons rencontré pas mal de difficulté. Nous avons tout d'abord, commencé à faire un diagramme de classe pour avoir une vision plus claire sur la structure de notre jeu, qu'on a souvent modifié au fur et à mesure que le projet avançait. Ensuite, nous avons également revu les règles de jeu d'échec afin de bien avoir en tête fonctionnement des déplacements de chaque pièce.

Nous avons d'abord commencé à coder l'affichage du plateau. Pour cela nous avions utilisé le StringBuilder pour optimiser l'affichage. Notre première difficulté était de faire un tableau de case pour les pièces et non juste un tableau car chaque case n'était pas forcément une pièce. Pour savoir comment tout cela fonctionner, on a fait quelque recherche internet et regarder une vidéo YouTube. Puis, un autre problème était de faire en sorte que les pièces restent uniquement dans l'échiquier. En effet, à la saisie, il ne fallait pas que les pièces soient placées ou déplacées à l'extérieure de l'échiquier.

Ensuite, arrivant à la partie code de saisie, le problème était de savoir comment la décrypter et la décomposer. En effet pour chaque coup, il fallait d'abord indiquer la case où la pièce actuelle se trouve, en précisant la ligne puis la colonne. Puis, de la même manière, indiquer la case où on souhaite déplacer la pièce. Pour cela, nous avons décidé de décomposer, dans un premier temps, la saisie en un tableau de 4 char et ensuite convertir en int et faire la même en coordonnées car la saisie comporte des chiffres et des lettres. Donc il nous a semblé nécessaire de découper la méthode en plusieurs « morceaux » pour convertir à chaque fois. Pour la classe Coord, nous avions repris le code qu'on a vu en TD.

Lorsque nous avons commencé à coder les pièces, il nous était demandé de les faire en IPiece donc avec une interface. Cependant, au départ du projet nous n'avions pas compris le fonctionnement des interfaces mais après quelques cours en TD et des recherches sur internet nous avons résolu ce problème.

Une autre difficulté a été de savoir dans quel package, il fallait placer notre interface IPiece. En effet, on ne savait pas s'il fallait laisser l'interface avec les classes qui l'utilisent ou non. Cependant, en écoutant les conseils de notre professeur, nous avons compris qu'il fallait éloigner au maximum l'interface des classes qui l'utilisent. C'est pour cela que nous avons décidé de placer l'interface dans le package « échiquier ».

Ainsi, pour mieux structurer notre code et les simplifier on a découpé les méthodes en plusieurs petites méthodes tout en évitant des redondances. En effet, ça nous a aider de les appeler plus facilement pour les vérifications sans qu'il y ait d'ambiguïté.

Ce qui pourrait être améliorer dans notre code est l'implémentation des autres pièces d'échec et aussi faire une IA contre IA ou humain. On n'a pas pu les faire par manque de temps et de connaissance.

Pour finir, durant ce projet, nous avons réussi à mettre en pratique plusieurs notions et concepts vu en cours. Nous avons rencontré quelques difficultés, cependant grâce à un travail d'équipe réguliers ainsi que de longues discussions, portant sur ces difficultés, nous avons réussis à les surmonter. Ce projet a été très instructif et enrichissant. Enfin, la découverte de ce jeu, ainsi que sa programmation ont été une bonne expérience.

Annexe :

Code Java :

- Package application, Classe Appli : Main

```
package application;
import echiquier.Coord;
import jeu.Partie;

public class Appli {
	/**
	 * Menu du jeu
	 */
	public static void menu() {
		StringBuilder sb=new StringBuilder();
		sb.append("   ------------------------------------------------------------
---------------------------\n");
		sb.append("   | Dans cette version, le jeu simule la finale d'une partie
d'échec.                    |\n");
		sb.append("   | Le joueur blanc possède une tour et un roi et le joueur
noir possède seulement le roi.|\n");
		sb.append("   | Le joueur blanc commance !
|\n");
		//expliquer la saisie
		//que en minuscule
		sb.append("   | Pour saisir un coup, écrivez la case de départ de la pièce
puis la case d'arrivée     |\n");
		sb.append("   | comme ceci : b1b2 (en minuscule)
|\n");
		//expliquer l'abandon, la demande de nul
		sb.append("   | Pour abandonner, écrivez \"stop\"
|\n");
		sb.append("   | Pour la proposition de nul, écrivez \"null\"
|\n");
		sb.append("   ------------------------------------------------------------
---------------------------\n");
		System.out.println(sb.toString());
	}
	public static void main(String[] args) {
		int cptT=1;
		menu();
		Partie partie=new Partie();
		do {
			System.out.println("_____________\nRound "+cptT+"\n");
			cptT++;
			partie.afficheEchiquier();
			System.out.println("Joueur " + partie.getJcourantCouleur()+ " :");
			Coord[] coord=partie.saisie();
			partie.recupSaisie(coord[0]).coup(partie.getJcourant(),
coord,partie.getEchiquier() );
			partie.inverserJoueur();
		}
		while(!partie.finPartie());
	}
```

- Package echiquier, Classe Coord

```java
package echiquier;

public class Coord {
        /**
     * Coordonnee sur abscisse
     */
    public int x;

    /**
     * Coordonnee sur ordonnee
     */
    public int y;

    /**
     * Constructeur pour x et y
     * @param x valeur en abscisse
     * @param y caleur en ordonnee
     */
    public Coord(int x, int y){
        this.x = x;
        this.y = y;
    }

    /**
     * Constructeur : initialise x et y en 0
     */
    public Coord(){
        this(0,0);
    }

    /**
     * Constructeur : initialise x et y sur la meme valeur
     * @param xy valeur pour x et pour y
     */
    public Coord(int xy){
        this(xy, xy);
    }
    public Coord(Coord coord) {
      this.x=coord.getX();
      this.y=coord.getY();
    }
    public int getX() {
      return x;
    }
    public int getY() {
      return y;
    }
    /**
     * Setter pour x et y
     * @param x nouvelle valeur de x
     * @param y nouvelle valeur de y
     */
```

```
    public void set(int x, int y){
        this.x = x;
        this.y = y;
    }

      @Override
      public String toString() {
            return "Coord [x=" + x + ", y=" + y + "]";
      }

}
```

- **Package echiquier, Classe Case**

```
package echiquier;

public class Case {
      private IPiece contenu;

      /**
     * Défini une case qui contient une pièce
     * @param contenu
     */
      public Case(IPiece contenu){
            this.contenu = contenu;
      }

      /**
     * Défini une case vide
     */
      public Case(){
        this.contenu = null;
      }


    public Case(Case c){
        if (c.contenu == null){
            this.contenu = null;
        }
        else{
            this.contenu = c.contenu;
        }
    }

    /**
     * Retourne la pièce
     * @return
     */
      public IPiece getPiece(){
            return contenu;
      }
```

```
    /**
     * Affecte une pièce
     * @param p piece
     */
      public void setPiece(IPiece p){
            contenu = p;
      }

    /**
     * Indique si la case est vide
     * @return
     */
    public boolean estVide(){
        return (contenu == null);
    }

    /**
     * Vide la case
     */
    public void vider(){
        contenu = null;
    }

    @Override
    public String toString(){
        if (contenu != null)
            return contenu.toString();

        else return "   ";
    }

}
```

- Package echiquier, Classe Echiquier

```
package echiquier;
import java.util.ArrayList;
import jeu.Joueur;
import piece.*;

public class Echiquier {
      public static final int TAILLE_ECHIQUIER = 8;
      //attributs
      private static int nbLig=TAILLE_ECHIQUIER;
      private static int nbCol=TAILLE_ECHIQUIER;
      private Case [][] grille;

      // Constructeur plateau
```

```
	public Echiquier() {
		grille = new Case[nbLig][nbCol];
		for(int x=0; x<nbLig; x++) {
			for(int y=0; y<nbCol; y++) {
				grille[x][y] = new Case();
			}
		}
		miseEnPlace();
	}

	// COPIER ECHIQUIER A PARTIR D'UNE AUTRE
	public Echiquier(Echiquier e) {
		ArrayList<IPiece> pBlanc=e.getPiecesJoueur("blanc");
		ArrayList<IPiece> pNoir=e.getPiecesJoueur("noir");

		grille = new Case[nbLig][nbCol];
		for(int x=0; x<nbLig; x++) {
			for(int y=0; y<nbCol; y++) {
				grille[x][y] = new Case();
			}
		}

		for(IPiece p:pBlanc) {
			Coord c=new Coord(p.getCoord());
			if(p.getFamille()=="roi")
				placer(c.x,c.y,new Roi(c.x,c.y,'R',"blanc",this));
			if(p.getFamille()=="tour")
				placer(c.x,c.y,new Tour(c.x,c.y,'T',"blanc",this));

		}
		for(IPiece p:pNoir) {
			Coord c=new Coord(p.getCoord());
			if(p.getFamille()=="roi")
				placer(c.x,c.y,new Roi(c.x,c.y,'r',"noir",this));
			if(p.getFamille()=="tour")
				placer(c.x,c.y,new Tour(c.x,c.y,'t',"noir",this));
		}
	}
	/**
	 * place les pieces sur l'echiquier
	 */
	private void miseEnPlace() {
		placer(0,4,new Roi(0,4,'r',"noir",this));
		placer(2,4,new Roi(2,4,'R',"blanc",this));
		placer(1,1,new Tour(1,1,'T',"blanc",this));
	}
	/**
	 * affiche l'echiquier
	 */
	public void afficher() {
		StringBuilder echiquier=new StringBuilder();
		echiquier.append("    a   b   c   d   e   f   g   h \n");
		for(int ligne=0, num=8; ligne<nbLig;ligne++,num--) {
```

```
                echiquier.append("   --- --- --- --- --- --- --- ---\n");
                echiquier.append(num+" ");
                for(int col=0; col<nbCol; col++) {
                      echiquier.append("|"+grille[ligne][col]);
                }
                echiquier.append("| "+num);
                echiquier.append("\n");


          }
          echiquier.append("   --- --- --- --- --- --- --- ---\n");
          echiquier.append("    a   b   c   d   e   f   g   h \n");
          System.out.println(echiquier.toString());
    }
    /**
     * place les pieces sur l'echiquier
     * @param lig
     * @param col
     * @param piece
     */
    public void placer (int lig, int col, IPiece piece) {

      //On test si on se trouve bien sur la grille
      if(lig<0 || col<0 || lig > nbLig || col> nbCol ) {

          System.out.println("Erreur!");
          return;
      }

      if(grille[lig][col].estVide()) {
          grille[lig][col].setPiece(piece);
      }

      else {
          System.out.println("Erreur, cette zone n'est pas vide !");
      }
    }
  //-----------------------------------------------
    //getters et setters
    public IPiece getPiece(Coord coord) {
          return grille[coord.getX()][coord.getY()].getPiece();
    }
    public boolean caseVide(Coord coord) {
          return grille[coord.getX()][coord.getY()].estVide();
    }
    public void setPiece(Coord coord, IPiece p) {
          grille[coord.getX()][coord.getY()].setPiece(p);
    }
    public void viderCase(Coord coord) {
          grille[coord.getX()][coord.getY()].vider();
    }
    public ArrayList<IPiece> getPiecesJoueur(String couleur){
    ArrayList<IPiece> p = new ArrayList<IPiece>();
```

```
        for(int i=0; i<grille.length;i++){
                for(int j=0; j<grille.length;j++){
                        if(!this.grille[i][j].estVide()){

        if(this.grille[i][j].getPiece().getCouleur().equals(couleur))
                                        p.add(this.grille[i][j].getPiece());
                        }
                }
        }
        return p;
    }
        //-------------------------------------------------
        /**
         * Cherche le roi du joueur dans l'echiquier
         * @param Jcourant
         * @return la coord du roi
         */
        public Coord rechercheRoi(Joueur Jcourant) {
                ArrayList<IPiece>pieces=getPiecesJoueur(Jcourant.getCouleur());
                for(IPiece piece:pieces) {
                        if(piece.getFamille().equals("roi"))
                                return piece.getCoord();
                }
                return null;
        }
        /**
         * verifie si la coord est dans l'echiquier
         * @param coord
         * @return true ou false
         */
        public boolean dansEchiquier(Coord coord) {

        if(coord.getX()<0||coord.getX()>=nbLig||coord.getY()<0||coord.getY()>=nbCo
l)
                        return false;
                else return true;
        }
}
```

- Package echiquier, Interface IPiece

```
package echiquier;

import java.util.ArrayList;

import jeu.Joueur;

public interface IPiece {

	boolean coupPossible(Coord coord);
	boolean verifCheminLibre(Coord coord);
	boolean verifCaseLibre(Coord coord);
	boolean verifMouv(Joueur Jcourant,Coord coord);
	boolean verifManger(Joueur Jcourant, Coord coord);
	boolean verifCaseOccupeAdverse(Joueur Jcourant, Coord coord);

	void manger(Echiquier echiquier,Coord[] coord);
	void deplacer(Echiquier echiquier,Coord[] coord);
	void coup(Joueur Jcourant,Coord[] coord,Echiquier echiquier);
	ArrayList<Coord> casesVersRoi(Coord coordRoi);
	ArrayList<Coord> casesPossible();
	boolean verifCoup(Joueur Jcourant,Joueur Jadverse, Coord[] coord,Echiquier
echiquier);
	String getCouleur();
	Coord getCoord();
	String getFamille();
	boolean estEchec(Joueur Jcourant,Joueur Jadverse);
	boolean estEchecEtMat(Joueur Jcourant,Joueur Jadverse);
}
```

- Package jeu, Classe Joueur

```
package jeu;

public class Joueur {
	private String couleur;
	public Joueur(String couleur) {
		this.couleur=couleur;
	}
	public String getCouleur() {
		return this.couleur;
	}
}
```

- Package jeu, Classe Partie

```
package jeu;

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

import echiquier.Coord;
import echiquier.Echiquier;
import echiquier.IPiece;


public class Partie {
	private Echiquier echiquier;
	private Joueur jBlanc;
	private Joueur jNoir;

	private Joueur Jcourant;
	private Joueur Jadverse;
	/**
	 * Constructeur partie
	 */
	public Partie() {
		echiquier=new Echiquier();
		jBlanc=new Joueur("blanc");
		jNoir=new Joueur("noir");
		Jcourant=jBlanc;
		Jadverse=jNoir;

	}
	/**
	 * Affiche l'echiquier
	 */
	public void afficheEchiquier() {
		echiquier.afficher();
	}
	//-----------------------------------------------------------------
	//Tous les getters
    public Echiquier getEchiquier() {
       return echiquier;
    }
    public Joueur getJcourant() {
       return Jcourant;
    }
    public Joueur getJadverse() {
       return Jadverse;
    }
    public String getJcourantCouleur() {
       return this.Jcourant.getCouleur();
    }
    public String getJadverseCouleur() {
```

```
        return this.Jadverse.getCouleur();
    }
    //-------------------------------------------------------------
    //POUR SAISIE
    /**
     * Permet de convertir la saisie en int
     * @param saisieChar tableau de char
     * @return tableau de int
     */
    public int[] charToInt (char[] saisieChar) {

          String lettres="abcdefgh";
          String chiffres="87654321";//a l'envers car l'echiquier a l'envers
          int[] tab=new int[4];

       for(int i=0;i<4;i++) {
             if (Character.isAlphabetic(saisieChar[i]))
                   tab[i]=lettres.indexOf(saisieChar[i])+1;//y
             else if(Character.isDigit(saisieChar[i]))
                   tab[i]=chiffres.indexOf(saisieChar[i])+1;
                   //tab[i]=Character.getNumericValue(saisieChar[i]);//x
             //[lettre,chiffre,lettre,chiffre]
             //[y,x,y,x] (a l'envers)
       }

          return tab;
    }
    //convertir en coord
    /**
     * Permet de convertir la saisie en Coord
     * @param saisieInt : tableau de int
     * @return tableau de Coord
     */
    public Coord[] convertirEnCoord(int[] saisieInt) {
       Coord[] coord=new Coord[2]; //2 pour les 2 coord de la saisie

       //-1 pour bien correspondre aux indices du tableau a 2 dimension de
l'echiquier
       coord[0]=new Coord(saisieInt[1]-1,saisieInt[0]-1);
       coord[1]=new Coord(saisieInt[3]-1,saisieInt[2]-1);
       return coord;
    }
    //-------------------------------------------------------------------
    // LA SAISIE
    /**
    * Demande à saisir les coups
    * @return tableau de char de la saisie
    */
    private char[] saisir() {
       @SuppressWarnings("resource")
             Scanner sc = new Scanner(System.in);
       String s=sc.next();
             char[] saisie=new char[4];
```

```
            for(int i=0; i<4;i++) {
                  saisie[i]= s.charAt(i);
            }
            return saisie;
      }
      /**
       * Effectue la saisie
       * @return tableau de Coord
       */
      public Coord[] saisie() {
        char[] saisie=saisir();
        if(abandon(saisie)) {
              System.out.println("le gagnant est "+Jadverse.getCouleur()+" !");
              System.exit(0);}
        if(nul(saisie)) {
              System.out.println("MATCH NUL !");
              System.exit(0);}
        int[] saisieInt=charToInt(saisie);
        Coord[] coord=convertirEnCoord(saisieInt);
        while(!verifSaisie(saisie,coord)) {
              //signaler l'erreur
              System.out.println("Mauvaise saisie !");
              saisie=saisir();
              saisieInt=charToInt(saisie);
              coord=convertirEnCoord(saisieInt);
        }
        return coord;
      }

      //verifSaisie
      /**
       * Verifie la saisie
       * @param saisie : tableau de char de la saisie
       * @param coord : tableau de Coord de la saisie
       * @return true si la saisie est correcte, sinon false
       */
      public boolean verifSaisie(char[] saisie, Coord[]coord) {
        if(bonFormat(saisie)) {
              if(verifPieceJouee(Jcourant,coord[0]))

        if(recupSaisie(coord[0]).verifCoup(Jcourant,Jadverse,coord,this.echiquier)
)
                        return true;
        }
        return false;
      }
      //----------------------------------------
      //POUR VERIF SAISIE
      /**
       * Verifie si la saisie a le bon format
       * @param saisie : tableau de char de la saisie
       * @return true si c'est le bon format sinon false
       */
```

```
    public boolean bonFormat(char[] saisie) {

      if(Character.isAlphabetic(saisie[0])&&Character.isAlphabetic(saisie[2])
                          &&
Character.isDigit(saisie[1])&&Character.isDigit(saisie[3])) {

      if(Character.isLowerCase(saisie[0])&&Character.isLowerCase(saisie[2])) {
                          if(dansEchiquier(saisie))
                                return true;
                    }
              }
      return false;
    }
    /**
     * Verifie si la saisie est bien dans l'echiquier
     * @param saisie tableau de char
     * @return true ou false
     */
    private boolean dansEchiquier(char[] saisie) {


      if(saisie[0]>='a'&&saisie[0]<='h'&&saisie[1]>'0'&&saisie[1]<='8'/*Appli.TA
ILLE_ECHIQUIER*/

      &&saisie[2]>='a'&&saisie[2]<='h'&saisie[3]>'0'&&saisie[3]<='8'/*Appli.TAIL
LE_ECHIQUIER*/)
              return true;
      return false;
    }
    /**
     * Verifie si la piece qu'on veut jouer existe bien à la coord donnée
     * @param Jcourant
     * @param coord
     * @return true ou false
     */
    public boolean verifPieceJouee(Joueur Jcourant, Coord coord) {

      if((!echiquier.caseVide(coord))&&recupSaisie(coord).getCouleur().equals(Jc
ourant.getCouleur()))
              return true;
      else return false;
    }
    /**
     * identifier la piece à jouer
     * @param coord
     * @return la piece
     */
    public IPiece recupSaisie(Coord coord) {
      return echiquier.getPiece(coord);
    }

    //-----------------------------------------
    /**
```

```
     * Inverse à chaque tour les joueurs
     */
    public void inverserJoueur() {
      if (Jcourant==jBlanc) {
            Jcourant=jNoir;
            Jadverse=jBlanc;
      }
      else {
            Jcourant=jBlanc;
            Jadverse=jNoir;
      }
    }

    /**
     * Verifie si le jeu est pat c'est à dire :
     *    le roi n'est pas en echec
     *    il ne peut pas se déplacer sans se mettre en echec
     *    ses pieces ne peuvent pas bouger sans mettre le roi en danger
     * @return true ou false
     */
    public boolean estPat() {
      //roi pas en echec

      if(!this.echiquier.getPiece(this.echiquier.rechercheRoi(Jcourant)).estEche
c(Jcourant,Jadverse)) {
                  //roi ne peut pas se deplacer
                  ArrayList<Coord>
casesPossibleRoi=this.echiquier.getPiece(this.echiquier.rechercheRoi(Jcourant)).
casesPossible();
                  Coord[] coordCoup=new Coord[2];
                  coordCoup[0]=this.echiquier.rechercheRoi(Jcourant);
                  for(Coord coord:casesPossibleRoi) {
                        coordCoup[1]=coord;

      if(this.echiquier.getPiece(this.echiquier.rechercheRoi(Jcourant)).verifCou
p(Jcourant,Jadverse, coordCoup,this.echiquier))
                              return false;
                  }
                  //le joueur ne peut pas deplacer ses pieces
                  ArrayList<IPiece>
pieces=this.echiquier.getPiecesJoueur(getJcourantCouleur());
                  //enlever le roi

      pieces.remove(this.echiquier.getPiece(this.echiquier.rechercheRoi(Jcourant
)));
                  Coord[] coord=new Coord[2];
                  for(IPiece piece:pieces) {
                        ArrayList<Coord> casesPossible=piece.casesPossible();
                        coord[0]=new Coord(piece.getCoord());
                        for(Coord c:casesPossible) {
                              coord[1]=new Coord(c);
                              if(piece.verifCoup(Jcourant, Jadverse,coord,
this.echiquier))
```

```
                                        return false;
                                }
                        }
                        return true;
        }
                else return false;

        }
    /**
     * Permet d'abandonner la partie si le joueur saisit "stop"
     * @param saisie
     * @return true ou false
     */
    public boolean abandon(char[]saisie) {
        char[] stop= {'s','t','o','p'};
        return Arrays.equals(stop,saisie);
    }
    /**
     * Permet de declarer la proposition de partie nulle si le joueur saisit
"null"
     * @param saisie
     * @return true ou false
     */
    public boolean nul(char[]saisie) {
        char[] nul= {'n','u','l','l'};
        return Arrays.equals(nul,saisie);
    }
    /**
     * Verifie si la partie est fini : par echec et mat ou pat
     * @return true ou false
     */
    public boolean finPartie() {
        //echec et mat

        if(this.echiquier.getPiece(this.echiquier.rechercheRoi(Jcourant)).estEchec
EtMat(Jcourant,Jadverse)) {
                System.out.println("Le gagnant est "+Jadverse.getCouleur()+" !
Félicitation");
                        return true;}
        //pat
                else if(estPat()) {
                System.out.println("PAT ! Partie Nulle.");
                return true;}

        return false;
    }
}
```

- Package jeu, Classe Piece

```
package piece;

import echiquier.Coord;
import echiquier.Echiquier;
import echiquier.IPiece;
import jeu.Joueur;

public abstract class Piece implements IPiece {
	private char nom;
	private String couleur;
	private Coord coord;
	private Echiquier echiquier;
	private String famille;

	/**
	 * CONSTRUCTEUR DE PIECE
	 * @param x
	 * @param y
	 * @param nom
	 * @param couleur
	 * @param famille
	 * @param echiquier
	 */
	public Piece(int x, int y, char nom, String couleur,String
famille,Echiquier echiquier){
	this.coord=new Coord(x,y);
	this.nom = nom;
	this.couleur=couleur;
	this.famille=famille;
	this.echiquier=echiquier;
    }
	//-------------------------
	//getters
	public int getX() {
		return this.coord.getX();
	}
	public int getY() {
		return this.coord.getY();
	}
	public Coord getCoord() {
		return this.coord;
	}
	public char getNom() {
		return this.nom;
	}
	public String getCouleur() {
		return this.couleur;
	}
	public String getFamille() {
		return this.famille;
```

```
	}
	public Echiquier getEchiquier() {
		return echiquier;
	}
	//-------------------------
/**
 * Setter pour x et y a la fois
 * pour changer les coordonnes de la piece
 * @param coord
 */
public void setXY(Coord coord){
	this.coord=coord;
}
/**
 * Verifie si la case d'arrivée est vide
 * @return true ou false
 */
public boolean verifCaseLibre(Coord coord) {
	if(this.echiquier.caseVide(coord))
		return true;
	return false;
}
/**
 * Verifie si le mouvement est possible
 * @param Jcourant : joueur courant
 * @param coord : coord case arrivee
 * @return true ou false
 */
@Override
public boolean verifMouv(Joueur Jcourant,Coord coord) { //regle+cheminLibre
	if(verifCaseLibre(coord)) {
		if(verifCheminLibre(coord)) {
			if(coupPossible(coord))
				return true;
		}
	}
	return false;
}
/**
 * Verifie si la case d'arrivée est occupée par une piece adverse
 * @param Jcourant : joueur courant
 * @param coord : coord case arrivee
 * @return true ou false
 */
@Override
public boolean verifCaseOccupeAdverse(Joueur Jcourant, Coord coord) {
	if(Jcourant.getCouleur()=="blanc")
		if(!this.echiquier.caseVide(coord))

	if(this.echiquier.getPiece(coord).getCouleur().equals("noir"))
			return true;
	if(Jcourant.getCouleur()=="noir")
		if(!this.echiquier.caseVide(coord))
```

```
      if(this.echiquier.getPiece(coord).getCouleur().equals("blanc"))
                        return true;
      return false;
    }

    /**
     * Verifie si la piece peut manger la piece adverse
     * @param Jcourant : joueur courant
     * @param coord : coord case arrivee
     * @return true ou false
      */
      @Override
    public boolean verifManger(Joueur Jcourant, Coord coord) {//regle +
caseOccupAdverse
      if(verifCaseOccupeAdverse(Jcourant, coord)) {
            if(verifCheminLibre(coord)) {
                  if(coupPossible(coord))
                        return true;
            }
      }
      return false;
    }
    /**
     * Verifie si le coup est possible
     * @param Jcourant : joueur courant
     * @param Jadverse : joueur adverse
     * @param coord : tabelau de coord de la saisie
     * @param echiquier : echiquier
     * @return true ou false

     */
    @Override
    public boolean verifCoup(Joueur Jcourant,Joueur Jadverse, Coord[]
coord,Echiquier echiquier) {//pour la saisie?
      if(verifMouv(Jcourant,coord[1])||verifManger(Jcourant, coord[1])) {
            //simuler le coup
            //verifier apres si le roi n'est pas en echec
            return verifPasEchec(Jcourant,Jadverse,coord,echiquier);
      }
      return false;
    }
    /**
     * Verifie si le roi n'est pas en echec apres le coup
     * @param Jcourant
     * @param Jadverse
     * @param coord
     * @param echiquier
     * @return true ou false
     */
    private boolean verifPasEchec(Joueur Jcourant,Joueur Jadverse,Coord[]
coord,Echiquier echiquier) {
      Echiquier simulation=new Echiquier(echiquier);
```

```
      simulation.getPiece(simulation.rechercheRoi(Jcourant)).coup(Jcourant,coord
,simulation);

      if(simulation.getPiece(simulation.rechercheRoi(Jcourant)).estEchec(Jcouran
t,Jadverse))   //a verifier
            return false;
      else return true;
    }

    /**
     * Deplace la piece : change les coordonnees de la piece et place dans
l'echiquier
     * @param coord : coordonnees de la case d'arrivée
     */
    @Override
    public void deplacer(Echiquier echiquier,Coord[] coord) {
      setXY(coord[1]);
      echiquier.setPiece(coord[1], this);
      echiquier.viderCase(coord[0]);
    }
    /**
     * Mange la piece adverse : vide la case et deplace la piece
     * @param echiquier
     * @param coord : tableau de coord de la saisie
     */
    @Override
      public void manger(Echiquier echiquier,Coord[] coord) {
      //enregistrer la piece adverse dans le tab prise
      echiquier.viderCase(coord[1]);
      deplacer(echiquier,coord);
      echiquier.viderCase(coord[0]);

    }
    /**
     * Effectue le coup d'apres la saisie
     * @param Jcourant
     * @param coord
     * @param echiquier
     */
    @Override
      public void coup(Joueur Jcourant,Coord[] coord,Echiquier echiquier) {
      if(verifMouv(Jcourant,coord[1])) {
            deplacer(echiquier,coord);
      }
      else if(verifManger(Jcourant, coord[1])) {
            manger(echiquier,coord);
      }
    }
    /**
     * @return le nom de la piece
     */
    public String toString() {
```

```
        return " " + this.nom + " ";
    }
}
```

- Package piece, Classe Roi

```
package piece;


import java.util.ArrayList;

import echiquier.Coord;
import echiquier.Echiquier;
import echiquier.IPiece;
import jeu.Joueur;

public class Roi extends Piece{
	private IPiece pieceMenacante;
	//CONSTRUCTEUR DU ROI
	public Roi(int x, int y, char nom,String couleur,Echiquier echiquier){
		super(x,y,nom,couleur,"roi",echiquier);
	}
	/**
	 * Verifie si le roi est en echec
	 * @param Jcourant
	 * @param Jadverse
	 */
	public boolean estEchec(Joueur Jcourant,Joueur Jadverse) {

	ArrayList<IPiece> piecesAdverse = new ArrayList<>();	//a optimiser si
on finit le proj !! car il loue de la memoire a chaque fois
	//recupere toutes les pieces adverses
	if(Jcourant.getCouleur().equals("blanc")) {
		piecesAdverse=this.getEchiquier().getPiecesJoueur("noir");
	}
	else
		piecesAdverse=this.getEchiquier().getPiecesJoueur("blanc");

	for(IPiece piece : piecesAdverse){
		if(piece.coupPossible(this.getCoord())) {
			if(piece.getFamille()=="roi") {
				if(piece.verifManger(Jadverse, this.getCoord())){
//Jcourant
					pieceMenacante=piece;
				return true;
				}
			}
			else if(piece.verifCheminLibre(this.getCoord())) {
				pieceMenacante=piece; //recuperer la piece menacante
				return true;
```

```
                        }
                  }
            }
                  return false;
            }
            /**
             * Verifie si le roi est echec et mat
             * @param Jcourant
             * @param Jadverse
             * @return true ou false
             *
            */
            public boolean estEchecEtMat(Joueur Jcourant,Joueur Jadverse) {
                  //assert(estEchec)
                  if(this.estEchec(Jcourant,Jadverse)) {
                        //roi ne peut pas se deplacer
                        ArrayList<Coord> casesPossible=this.casesPossible();
                        Coord[] tabcoord=new Coord[2];
                        tabcoord[0]=this.getCoord();
                        for(Coord coord:casesPossible) {
                              tabcoord[1]=coord;

            if(this.verifCoup(Jcourant,Jadverse,tabcoord,this.getEchiquier()))
                                    return false;
                        }
                        //les autres pieces ne peuvent pas sauver le roi
                        //recuperer les autres pieces du joueur
                        ArrayList<IPiece>
pieces=this.getEchiquier().getPiecesJoueur(this.getCouleur());
                        //if(pieces.contains(this))
                        //enlever le roi de la liste
                        pieces.remove(this);

                        //tester pour chaque piece si elle peut manger la piece
menacante ou se placer entre la piece menacante et le roi
                        for(IPiece piece:pieces) {
                              if(piece.verifManger(Jcourant,
pieceMenacante.getCoord())) {
                                    return false;
                              }
                              else {
                                    ArrayList<Coord>
cases=pieceMenacante.casesVersRoi(this.getCoord());
                                    for(Coord c:cases){
                                          if(piece.verifMouv(Jcourant, c));
                                                return false;
                                    }
                              }


                        }
                        return true;
                  }
```

```
            else return false;
      }
      /**
       * Recupere les cases de deplacement possible du roi
       * @return coord dans arrayList
       */
      public ArrayList<Coord> casesPossible(){
            //recuperer les cases ou le roi peut se deplacer
            ArrayList<Coord> cases=new ArrayList<Coord>();
            Coord coord=new Coord();
             for(int y=this.getY()-1; y<=this.getY()+1; y++) {
            for(int x=this.getX()-1; x<=this.getX()+1; x++) {
               coord.set(x, y);
                if(this.getEchiquier().dansEchiquier(coord)&&(x!=this.getX() ||
y!=this.getY()))
                     cases.add(new Coord(x,y));
            }
             }
            return cases;
      }
      @Override
      /**
       * Verifie si le coup est possible selon les règles de la piece
       * ici verifie si la piece se deplace une case autour
       * @param xy coordonnées de la case d'arrivée
       * @return true si le coup est possible sinon false
       */
      public boolean coupPossible(Coord coord) {
            //deplacement horizontal
      if(coord.getX()==this.getX()+1 && coord.getY()==this.getY() ||
coord.getX()==this.getX()-1 && coord.getY()==this.getY()){
            return true;
      //deplacement vertical
      }else if(coord.getX()==this.getX() && coord.getY()==this.getY()+1 ||
coord.getX()==this.getX() && coord.getY()==this.getY()-1){
            return true;
      //deplacement dans les 4 diagonales
      }else if(coord.getX()==this.getX()+1 && coord.getY()==this.getY()+1 ||
coord.getX()==this.getX()+1 && coord.getY()==this.getY()-1 ||
coord.getX()==this.getX()-1 && coord.getY()==this.getY()-1 ||
coord.getX()==this.getX()-1 && coord.getY()==this.getY()+1){
            return true;
      }
            return false;
      }

      /**
       * verifie si le chemin est libre jusqua la case d'arrivée sans compter
celle la
       * @param coord
       * @return true ou false
       */
      @Override
```

```
	public boolean verifCheminLibre(Coord coord) {
		if(verifCaseLibre(coord))
			return true;
		return false;
	}
	/**
	 * verifie si le roi peut manger une piece
	 * @param Jcourant
	 * @param coord
	 * @return true ou false
	 */
	@Override
	 public boolean verifManger(Joueur Jcourant, Coord coord) {//regle +
caseOccupAdverse
		if(verifCaseOccupeAdverse(Jcourant, coord)) {
		  if(coupPossible(coord))
			return true;
		}
		return false;
	 }

	@Override
	public ArrayList<Coord> casesVersRoi(Coord coordRoi) {
		// TODO Auto-generated method stub
		return null;
	}


}
```

- Package piece, Classe Tour

```
package piece;

import java.util.ArrayList;

import echiquier.Coord;
import echiquier.Echiquier;
import jeu.Joueur;

public class Tour extends Piece{
	/**
	 * CONSTRUCTEUR
	 * @param x
	 * @param y
	 * @param couleur
	 */
	public Tour(int x, int y, char nom,String couleur,Echiquier echiquier){
		super(x,y,nom,couleur,"tour",echiquier);
	}
```

```
@Override
/**
 * Verifie si le coup est possible selon les règles de la piece
 * ici verifie si la piece se deplace verticalement ou horizontalement
 * @param xy coordonnées de la case d'arrivée
 * @return true si le coup est possible sinon false
 */
public boolean coupPossible(Coord coord) {
	for(int i = 0;i <=Echiquier.TAILLE_ECHIQUIER;i++){
		//deplacement vertical (sur la meme colonne)
	if(i==coord.getX() && coord.getY()==this.getY()){
		return true;
	}
	//deplacement horizontal (sur la meme ligne)
	if(i==coord.getY() && coord.getX()==this.getX()){
		return true;
	}
}
	return false;
}
/**
 * Verifie si le chemin est libre de la tour jusqu'a la case d'arrivée
 * @param coord : case d'arrivee
 * @return true ou false
 */
@Override
public boolean verifCheminLibre(Coord coord) {
	Coord coordCase=new Coord();
    if(this.getX() != coord.getX() && this.getY() ==coord.getY()){
      if(this.getX() > coord.getX()){ // vers haut
		for(int i=this.getX()-1;i>coord.getX();i--){
			coordCase.set(i, coord.getY());
			if(!this.getEchiquier().caseVide(coordCase)){
				return false;
			}
		}
		return true;
	}
	if(this.getX() < coord.getX()){ // vers bas
		for(int i=this.getX()+1;i<coord.getX();i++){
			coordCase.set(i, coord.getY());
			if(!this.getEchiquier().caseVide(coordCase)){
				return false;
			}
		}
		return true;
	}
    }
    if(this.getY() != coord.getY() && this.getX() == coord.getX()){
      if(this.getY() > coord.getY()){ // vers gauche
		for(int i=this.getY()-1;i>coord.getY();i--){
			coordCase.set(coord.getX(),i);
```

```
                        if(!this.getEchiquier().caseVide(coordCase)){
                              return false;
                        }
                  }
                  return true;
            }
            if(this.getY() < coord.getY()){ // vers droite
                  for(int i=this.getY()+1;i<coord.getY();i++){
                        coordCase.set(coord.getX(),i);
                        if(!this.getEchiquier().caseVide(coordCase)){
                              return false;
                        }
                  }
            return true;
            }
          }
          return false;
      }
      /**
       * Recupere les cases de deplacement possible
       * @return les coord dans arrayList
       */
      @Override
      public ArrayList<Coord> casesPossible() {
            ArrayList<Coord> cases=new ArrayList<Coord>();
            Coord coord=new Coord();
            // vertical haut
            for (int x = this.getX() - 1; x >= 0; x--) {
                  coord.set(x, this.getY());
                  if (this.getEchiquier().dansEchiquier(coord) && (x !=
this.getX()) && this.getEchiquier().caseVide(coord))
                        cases.add(new Coord(x, this.getY()));
                  else
                        break;
            }
            //vertical bas
            for(int x=this.getX()+1; x<Echiquier.TAILLE_ECHIQUIER; x++) {
                coord.set(x, this.getY());

if(this.getEchiquier().dansEchiquier(coord)&&(x!=this.getX())&&this.getEchiquier
().caseVide(coord))
                      cases.add(new Coord(x,this.getY()));
                else break;
            }
            //horizontal gauche
            for(int y=this.getY()-1; y>=0; y--) {
                coord.set(this.getX(), y);

if(this.getEchiquier().dansEchiquier(coord)&&(y!=this.getY())&&this.getEchiquier
().caseVide(coord))
                  cases.add(new Coord(this.getX(),y));
                  else break;
            }
```

```
            //horizontal droite
            for (int y = this.getY() + 1; y < Echiquier.TAILLE_ECHIQUIER; y++)
{
                  coord.set(this.getX(), y);
                  if (this.getEchiquier().dansEchiquier(coord) && (y !=
this.getY()) && this.getEchiquier().caseVide(coord))
                        cases.add(new Coord(this.getX(), y));
                  else
                        break;
            }
            return cases;
      }

      //si la piece menace le roi
      /**
       * Recupere les cases jusqu'au roi
       * @param coordRoi
       * @return coord dans arrayList
       */
      @Override
      public ArrayList<Coord> casesVersRoi(Coord coordRoi){ //a revoir
      ArrayList<Coord> cases=new ArrayList<Coord>();
            //si le roi est en haut
            if(coordRoi.getX()<this.getX()&&coordRoi.getY()==this.getY()) {
                  for(int i=this.getX()-1;i>coordRoi.getX();i--) {

                        cases.add(new Coord(i,this.getY()));
                  }
            }
            //si le roi est en bas
            if(coordRoi.getX()>this.getX()&&coordRoi.getY()==this.getY()) {
                  for(int i=this.getX()+1;i<coordRoi.getX();i++) {
                        cases.add(new Coord(i,this.getY()));
                  }
            }
            //si le roi est a droite
            if(coordRoi.getX()==this.getX()&&coordRoi.getY()>this.getY()) {
                  for(int i=this.getY()+1;i<coordRoi.getY();i++) {

                        cases.add(new Coord(this.getX(),i));
                  }
            }
            //si le roi est a gauche
            if(coordRoi.getX()==this.getX()&&coordRoi.getY()<this.getY()) {
                  for(int i=this.getY()-1;i>coordRoi.getY();i--) {

                        cases.add(new Coord(this.getX(),i));
                  }
            }

      return cases;
    }
```

```
	@Override
	public boolean estEchecEtMat(Joueur Jcourant,Joueur Jadverse) {
		// TODO Auto-generated method stub
		return false;
	}

	@Override
	public boolean estEchec(Joueur Jcourant,Joueur Jadverse) {
		// TODO Auto-generated method stub
		return false;
	}

}
```

- Package test, Junit test case TestSaisie

```
package tests;

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

import org.junit.jupiter.api.Test;

import echiquier.Coord;
import jeu.Partie;

class TestSaisie {
	public static final int TAILLE_ECHIQUIER = 8;
	@Test
	void testSaisie() {
		Partie partie=new Partie();
		char[] saisie=new char[] {'a','1','a','2'};
	  int[] saisieInt=partie.charToInt(saisie);
	  int[] testSaisieInt=new int[] {1,8,1,7};
	  assertEquals(saisieInt[0],testSaisieInt[0]);
	  assertEquals(saisieInt[1],testSaisieInt[1]);
	  assertEquals(saisieInt[2],testSaisieInt[2]);
	  assertEquals(saisieInt[3],testSaisieInt[3]);

	  Coord[] saisieCoord=partie.convertirEnCoord(saisieInt);
	  Coord[] testSaisieCoord= new Coord[2];
	  testSaisieCoord[0]=new Coord(7,0);
	  testSaisieCoord[1]=new Coord(6,0);
	  assertEquals(saisieCoord[0].getX(),testSaisieCoord[0].getX());
	  assertEquals(saisieCoord[0].getY(),testSaisieCoord[0].getY());
	  assertEquals(saisieCoord[1].getX(),testSaisieCoord[1].getX());
	  assertEquals(saisieCoord[1].getY(),testSaisieCoord[1].getY());
	}

	@Test
	void testBonFormat() {
```

```
            Partie partie=new Partie();
            char[] saisie=new char[] {'a','1','a','2'};
            assertTrue(partie.bonFormat(saisie));
            char[] saisie1=new char[] {'1','a','a','2'};
            assertFalse(partie.bonFormat(saisie1));
            char[] saisie2=new char[] {'r','6','a','2'};
            assertFalse(partie.bonFormat(saisie2));
      }

      void testVerifPieceJouee(char c1, char c2, char c3, char c4,char c11, char
c22, char c33, char c44) {
            Partie partie=new Partie();

            char[] saisie=new char[] {c1,c2,c3,c4};
        int[] saisieInt=partie.charToInt(saisie);
        Coord[] saisieCoord=partie.convertirEnCoord(saisieInt);
        assertTrue(partie.verifPieceJouee(partie.getJcourant(),
saisieCoord[0]));

        char[] saisie1=new char[] {c11,c22,c33,c44};
        int[] saisieInt1=partie.charToInt(saisie1);
        Coord[] saisieCoord1=partie.convertirEnCoord(saisieInt1);
        assertFalse(partie.verifPieceJouee(partie.getJcourant(),
saisieCoord1[0]));
      }
      @Test
      void testRecupSaisie() {
            Partie partie=new Partie();
            char[] saisie=new char[] {'a','1','a','2'};
        int[] saisieInt=partie.charToInt(saisie);
        Coord[] saisieCoord=partie.convertirEnCoord(saisieInt);

assertEquals(partie.recupSaisie(saisieCoord[0]),partie.getEchiquier().getPiece(s
aisieCoord[0]));
      }
      @Test
      void testInverserJoueur() {
            Partie partie=new Partie();
            assertTrue(partie.getJcourantCouleur()=="blanc");
            partie.inverserJoueur();
            assertFalse(partie.getJcourantCouleur()=="blanc");
            assertTrue(partie.getJcourantCouleur()=="noir");
            assertTrue(partie.getJadverseCouleur()=="blanc");

            //testVerifPieceJouee
            char[] saisie=new char[] {'e','8','e','7'};
        int[] saisieInt=partie.charToInt(saisie);
        Coord[] saisieCoord=partie.convertirEnCoord(saisieInt);
        assertTrue(partie.verifPieceJouee(partie.getJcourant(),
saisieCoord[0]));

        partie.inverserJoueur();
        testVerifPieceJouee('b','7','b','6','b','1','b','2');
```

```
	}
	@Test
	void testVerifSaisie() {
		Partie partie=new Partie();
		partie.afficheEchiquier();
		char[] saisie=new char[] {'b','7','b','8'};
	  int[] saisieInt=partie.charToInt(saisie);
	  Coord[] saisieCoord=partie.convertirEnCoord(saisieInt);
	  assertTrue(partie.verifSaisie(saisie,saisieCoord)); // A RETESTER APRES
PB SIMULATION
	}

}
```

- Package test, Junit test case TestDeplacement

```
package tests;

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

import org.junit.jupiter.api.Test;

import echiquier.Coord;
import echiquier.IPiece;
import jeu.Partie;
import piece.Tour;

class TestDeplacement {
	public static final int TAILLE_ECHIQUIER = 8;
	@Test
	void testVerifCaseLibre() {
		Partie partie=new Partie();

		//partie.afficheEchiquier();
		char[] saisie=new char[] {'b','7','b','8'};
	  int[] saisieInt=partie.charToInt(saisie);
	  Coord[] saisieCoord=partie.convertirEnCoord(saisieInt);

assertTrue(partie.recupSaisie(saisieCoord[0]).verifCaseLibre(saisieCoord[1]));
	  partie.getEchiquier().placer(0,1, new
Tour(0,1,'t',"noir",partie.getEchiquier()));
	  //partie.afficheEchiquier();

assertFalse(partie.recupSaisie(saisieCoord[0]).verifCaseLibre(saisieCoord[1]));
	}
	@Test
	void testVerifCheminLibre() {
		Partie partie=new Partie();

		//pour la tour blanche
```

```
            char[] saisie=new char[] {'b','7','b','8'};
        int[] saisieInt=partie.charToInt(saisie);
        Coord[] saisieCoord=partie.convertirEnCoord(saisieInt);

assertTrue(partie.recupSaisie(saisieCoord[0]).verifCheminLibre(saisieCoord[1]));

        Coord[] coord=new Coord[2];
        coord[0]=new Coord(1,1);
        coord[1]=new Coord(2,1);
        partie.getEchiquier().getPiece(coord[0]).deplacer(partie.getEchiquier(),
coord);
        char[] saisie1=new char[] {'b','6','f','6'};
        int[] saisieInt1=partie.charToInt(saisie1);
        Coord[] saisieCoord1=partie.convertirEnCoord(saisieInt1);

assertFalse(partie.recupSaisie(saisieCoord1[0]).verifCheminLibre(saisieCoord1[1]
));
        //pour le roi blanc
      char[] saisie2=new char[] {'e','6','d','6'};
        int[] saisieInt2=partie.charToInt(saisie2);
        Coord[] saisieCoord2=partie.convertirEnCoord(saisieInt2);

assertTrue(partie.recupSaisie(saisieCoord2[0]).verifCheminLibre(saisieCoord2[1])
);
        partie.getEchiquier().placer(2,3, new
Tour(2,3,'t',"noir",partie.getEchiquier()));
        //partie.afficheEchiquier();


assertFalse(partie.recupSaisie(saisieCoord2[0]).verifCheminLibre(saisieCoord2[1]
));
      }
      @Test
      void testCoupPossible() {
            Partie partie=new Partie();
            //TOUR
            char[] saisieTour=new char[] {'b','7','b','1'};
        int[] saisieIntTour=partie.charToInt(saisieTour);
        Coord[] saisieCoordTour=partie.convertirEnCoord(saisieIntTour);

assertTrue(partie.recupSaisie(saisieCoordTour[0]).coupPossible(saisieCoordTour[1
]));
        char[] saisieTour1=new char[] {'b','7','a','8'};
        int[] saisieIntTour1=partie.charToInt(saisieTour1);
        Coord[] saisieCoordTour1=partie.convertirEnCoord(saisieIntTour1);

assertFalse(partie.recupSaisie(saisieCoordTour1[0]).coupPossible(saisieCoordTour
1[1]));
        char[] saisieTour2=new char[] {'b','7','a','3'};
        int[] saisieIntTour2=partie.charToInt(saisieTour2);
        Coord[] saisieCoordTour2=partie.convertirEnCoord(saisieIntTour2);
```

```
assertFalse(partie.recupSaisie(saisieCoordTour2[0]).coupPossible(saisieCoordTour
2[1]));

        //ROI
        char[] saisieRoi=new char[] {'e','6','d','7'};
        int[] saisieIntRoi=partie.charToInt(saisieRoi);
        Coord[] sasisieCoordRoi=partie.convertirEnCoord(saisieIntRoi);

assertTrue(partie.recupSaisie(sasisieCoordRoi[0]).coupPossible(sasisieCoordRoi[1
]));
        char[] saisieRoi1=new char[] {'e','6','e','3'};
        int[] saisieIntRoi1=partie.charToInt(saisieRoi1);
        Coord[] saisieCoordRoi1=partie.convertirEnCoord(saisieIntRoi1);

assertFalse(partie.recupSaisie(saisieCoordRoi1[0]).coupPossible(saisieCoordRoi1[
1]));
        char[] saisieRoi2=new char[] {'e','6','c','5'};
        int[] saisieIntRoi2=partie.charToInt(saisieRoi2);
        Coord[] saisieCoordRoi2=partie.convertirEnCoord(saisieIntRoi2);

assertFalse(partie.recupSaisie(saisieCoordRoi2[0]).coupPossible(saisieCoordRoi2[
1]));
      }
      @Test
      void testVerifMouv() {
            Partie partie=new Partie();
            //partie.afficheEchiquier();
            char[] saisieTour=new char[] {'b','7','b','2'};
        int[] saisieIntTour=partie.charToInt(saisieTour);
        Coord[] saisieCoordTour=partie.convertirEnCoord(saisieIntTour);

assertTrue(partie.recupSaisie(saisieCoordTour[0]).verifMouv(partie.getJcourant()
,saisieCoordTour[1]));
        Coord[] coord=new Coord[2];
        coord[0]=new Coord(1,1);
        coord[1]=new Coord(2,1);
        partie.getEchiquier().getPiece(coord[0]).deplacer(partie.getEchiquier(),
coord);
        char[] saisieTour1=new char[] {'b','6','g','6'};
        int[] saisieIntTour1=partie.charToInt(saisieTour1);
        Coord[] saisieCoordTour1=partie.convertirEnCoord(saisieIntTour1);

assertFalse(partie.recupSaisie(saisieCoordTour1[0]).verifMouv(partie.getJcourant
(),saisieCoordTour1[1]));
      }
      @Test
      void testVerifCaseOccupAdverse() {
            Partie partie=new Partie();
            partie.getEchiquier().placer(0,1, new
Tour(0,1,'t',"noir",partie.getEchiquier()));
            //partie.afficheEchiquier();
            char[] saisieTour=new char[] {'b','7','b','8'};
```

```
        int[] saisieIntTour=partie.charToInt(saisieTour);
        Coord[] saisieCoordTour=partie.convertirEnCoord(saisieIntTour);

assertTrue(partie.recupSaisie(saisieCoordTour[0]).verifCaseOccupeAdverse(partie.
getJcourant(), saisieCoordTour[1]));
       }
       @Test
       void testVerifManger() {
              Partie partie=new Partie();
              partie.getEchiquier().placer(0,1, new
Tour(0,1,'t',"noir",partie.getEchiquier()));
              //partie.afficheEchiquier();
              char[] saisieTour=new char[] {'b','7','b','8'};
        int[] saisieIntTour=partie.charToInt(saisieTour);
        Coord[] saisieCoordTour=partie.convertirEnCoord(saisieIntTour);

assertTrue(partie.recupSaisie(saisieCoordTour[0]).verifManger(partie.getJcourant
(),saisieCoordTour[1]));
       }




       @Test
       void testVerifCoup() {
              Partie partie=new Partie();
              partie.getEchiquier().placer(1,7, new
Tour(1,7,'t',"noir",partie.getEchiquier()));
              partie.afficheEchiquier();
              //CAS 1 : verifMouv
              char[] saisieTour=new char[] {'b','7','g','7'};
        int[] saisieIntTour=partie.charToInt(saisieTour);
        Coord[] saisieCoordTour=partie.convertirEnCoord(saisieIntTour);

assertTrue(partie.recupSaisie(saisieCoordTour[0]).verifCoup(partie.getJcourant()
,partie.getJadverse(),saisieCoordTour,partie.getEchiquier()));
         //PB COPIER ECHIQUIER
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        //CAS 2 : verifManger
        partie.afficheEchiquier();
              char[] saisieTour1=new char[] {'b','7','h','7'};
        int[] saisieIntTour1=partie.charToInt(saisieTour1);
        Coord[] saisieCoordTour1=partie.convertirEnCoord(saisieIntTour1);

assertTrue(partie.recupSaisie(saisieCoordTour1[0]).verifCoup(partie.getJcourant(
),partie.getJadverse(),saisieCoordTour1,partie.getEchiquier()));
       }
       @Test
       void testDeplacer() {
              Partie partie=new Partie();
```

```
            //partie.afficheEchiquier();
            char[] saisieTour=new char[] {'b','7','b','6'};
        int[] saisieIntTour=partie.charToInt(saisieTour);
        Coord[] saisieCoordTour=partie.convertirEnCoord(saisieIntTour);

partie.recupSaisie(saisieCoordTour[0]).deplacer(partie.getEchiquier(),saisieCoor
dTour);
            //partie.afficheEchiquier();

      assertEquals(partie.recupSaisie(saisieCoordTour[1]),partie.getEchiquier().
getPiece(saisieCoordTour[1]));
            assertTrue(partie.getEchiquier().caseVide(saisieCoordTour[0]));
      }
      @Test
      void testManger() {
            Partie partie=new Partie();
            IPiece t=new Tour(0,1,'t',"noir",partie.getEchiquier());
            partie.getEchiquier().placer(0,1,t );
            //partie.afficheEchiquier();
            char[] saisieTour=new char[] {'b','7','b','8'};
        int[] saisieIntTour=partie.charToInt(saisieTour);
        Coord[] saisieCoordTour=partie.convertirEnCoord(saisieIntTour);

partie.recupSaisie(saisieCoordTour[0]).manger(partie.getEchiquier(),saisieCoordT
our);
            //partie.afficheEchiquier();

      assertEquals(partie.recupSaisie(saisieCoordTour[1]),partie.getEchiquier().
getPiece(saisieCoordTour[1]));
            assertTrue(partie.getEchiquier().caseVide(saisieCoordTour[0]));
            assertNotEquals(partie.recupSaisie(saisieCoordTour[1]),t);
      }
      @Test
      void testCoup() {
            Partie partie=new Partie();
            //partie.afficheEchiquier();
            //DEPLACER
            char[] saisieTour=new char[] {'b','7','g','7'};
        int[] saisieIntTour=partie.charToInt(saisieTour);
        Coord[] saisieCoordTour=partie.convertirEnCoord(saisieIntTour);

partie.recupSaisie(saisieCoordTour[0]).coup(partie.getJcourant(),saisieCoordTour
,partie.getEchiquier());
            //partie.afficheEchiquier();

      assertEquals(partie.recupSaisie(saisieCoordTour[1]),partie.getEchiquier().
getPiece(saisieCoordTour[1]));
            assertTrue(partie.getEchiquier().caseVide(saisieCoordTour[0]));
            //MANGER
            IPiece t=new Tour(1,7,'t',"noir",partie.getEchiquier());
            partie.getEchiquier().placer(1,7,t );
            //partie.afficheEchiquier();
            char[] saisieTour1=new char[] {'g','7','h','7'};
```

```
        int[] saisieIntTour1=partie.charToInt(saisieTour1);
        Coord[] saisieCoordTour1=partie.convertirEnCoord(saisieIntTour1);

partie.recupSaisie(saisieCoordTour1[0]).coup(partie.getJcourant(),saisieCoordTou
r1,partie.getEchiquier());
            //partie.afficheEchiquier();

      assertEquals(partie.recupSaisie(saisieCoordTour[1]),partie.getEchiquier().
getPiece(saisieCoordTour[1]));
            assertTrue(partie.getEchiquier().caseVide(saisieCoordTour[0]));
            assertNotEquals(partie.recupSaisie(saisieCoordTour[1]),t);
      }

}
```

- Package test, Junit test case TestPartie

```
package tests;

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


import org.junit.jupiter.api.Test;

import echiquier.Coord;
import jeu.Partie;
import piece.*;

class TestPartie {

      @Test
      void testEchecEtMat() { // test sur le roi noir
            Partie partie=new Partie();
            partie.inverserJoueur();

      assertFalse(partie.getEchiquier().getPiece(partie.getEchiquier().recherche
Roi(partie.getJcourant())).estEchec(partie.getJcourant(),partie.getJadverse()));
            //partie.afficheEchiquier();
            partie.inverserJoueur();
            char[] saisieTour=new char[] {'b','7','b','8'};
        int[] saisieIntTour=partie.charToInt(saisieTour);
        Coord[] saisieCoordTour=partie.convertirEnCoord(saisieIntTour);

partie.recupSaisie(saisieCoordTour[0]).coup(partie.getJcourant(),saisieCoordTour
,partie.getEchiquier());
        partie.inverserJoueur();
        //partie.afficheEchiquier();

assertTrue(partie.getEchiquier().getPiece(partie.getEchiquier().rechercheRoi(par
```

```
tie.getJcourant())).estEchec(partie.getJcourant(),partie.getJadverse()));//NON
si verifCoup dans estEchec et OUI si coupPossible et verifChemin libre

assertTrue(partie.getEchiquier().getPiece(partie.getEchiquier().rechercheRoi(par
tie.getJcourant())).estEchecEtMat(partie.getJcourant(),partie.getJadverse()));
        partie.inverserJoueur();
        //partie.afficheEchiquier();
             char[] saisieRoi=new char[] {'e','6','e','5'};
        int[] saisieIntRoi=partie.charToInt(saisieRoi);
        Coord[] saisieCoordRoi=partie.convertirEnCoord(saisieIntRoi);

partie.recupSaisie(saisieCoordRoi[0]).coup(partie.getJcourant(),saisieCoordRoi,p
artie.getEchiquier());
        partie.inverserJoueur();
        //partie.afficheEchiquier();

assertFalse(partie.getEchiquier().getPiece(partie.getEchiquier().rechercheRoi(pa
rtie.getJcourant())).estEchecEtMat(partie.getJcourant(),partie.getJadverse()));
      }

      @Test
      void testPat() {
            Partie partie = new Partie();
            assertFalse(partie.estPat());
            //partie.afficheEchiquier();
            //enlever toutes les pieces pour les replacer

      partie.getEchiquier().viderCase(partie.getEchiquier().rechercheRoi(partie.
getJcourant()));

      partie.getEchiquier().viderCase(partie.getEchiquier().rechercheRoi(partie.
getJadverse()));
            Coord coordTour=new Coord(1,1);
            partie.getEchiquier().viderCase(coordTour);
            //partie.afficheEchiquier();
            partie.getEchiquier().placer(7, 7, new
Roi(7,7,'r',"noir",partie.getEchiquier()));
            partie.getEchiquier().placer(6,5, new
Roi(6,5,'R',"blanc",partie.getEchiquier()));
            partie.getEchiquier().placer(6, 6, new
Tour(6,6,'T',"blanc",partie.getEchiquier()));
            //partie.afficheEchiquier();
            partie.inverserJoueur();
            assertTrue(partie.estPat());
      }
      @Test
      void testFinPartie() {
            Partie partie=new Partie();
            assertFalse(partie.finPartie());
            //partie.afficheEchiquier();
            //ECHEC ET MAT pour le noir
            char[] saisieTour=new char[] {'b','7','b','8'};
        int[] saisieIntTour=partie.charToInt(saisieTour);
```

```
        Coord[] saisieCoordTour=partie.convertirEnCoord(saisieIntTour);

partie.recupSaisie(saisieCoordTour[0]).coup(partie.getJcourant(),saisieCoordTour
,partie.getEchiquier());
        partie.inverserJoueur();
        partie.afficheEchiquier();
        assertTrue(partie.finPartie());

        //PAT

partie.getEchiquier().viderCase(partie.getEchiquier().rechercheRoi(partie.getJco
urant()));

      partie.getEchiquier().viderCase(partie.getEchiquier().rechercheRoi(partie.
getJadverse()));
            Coord coordTour=new Coord(0,1);
            partie.getEchiquier().viderCase(coordTour);
            partie.afficheEchiquier();
            partie.getEchiquier().placer(7, 7, new
Roi(7,7,'r',"noir",partie.getEchiquier()));
            partie.getEchiquier().placer(6,5, new
Roi(6,5,'R',"blanc",partie.getEchiquier()));
            partie.getEchiquier().placer(6, 6, new
Tour(6,6,'T',"blanc",partie.getEchiquier()));
            partie.afficheEchiquier();
            assertTrue(partie.finPartie());
            partie.inverserJoueur();
            assertFalse(partie.finPartie());

      }

}
```