# *Rendu du 14 juin 2019 :*

- Création de la classe *RWFile* permettant de lire et d'écrire des fichiers. Elle possède les méthodes writeFile() et readFile() (Ces classes sont statiques).
- Implémentation de la méthode jeu() de la classe *Humain*
- Implémentation de la méthode nextCoup() de la classe *Pion*
- Changement de la méthode toString(), suppression de drawCase(), changement du damier en tableau de booléens et ajout de la classe getDamier() dans la classe *Plateau*. Ce changement pour le damier rendra plus facile la vérification des coups.
- Ajout de l'attribut humain (Boolean) et de la méthode isHumain() dans la classe *Joueur*. Cette méthode permet de savoir si le joueur sélectionné est un humain ou non. Egalement, implémentation de la méthode deplacerPion().
- Ajout d'un attribut référencent les barrières déjà placées et d'une méthode addBarriere() dans la classe *Partie*. Cette méthode permet d'ajouter une barrière.
- Implémentation de la méthode setCoordonnee() et ajout d'un attribut référencent la plateau de jeu dans la classe Barriere.
- Modification du build.xml pour que la compilation prenne également en compte le package utilities.
- Changement des conditions de taille pour les coordonnées et suppression des setters dans la classe *Coordonnee*.

**Classes modifiées :**

*Classe Barriere*

```
private Plateau plateau;
// ---- //

public Barriere(String couleur, Plateau plateau) {
      try {
        if (couleur == null) {
          throw new Exception("Barriere constructeur - La couleur de la barrière doit
exister.");
        }
        else if (plateau == null) {
          throw new Exception("Barriere constructeur - Le plateau doit exister.");
        }
        else {
          this.COULEUR = couleur;
          this.coordonnee = null;
          this.plateau = plateau;
        }
      }
      catch(Exception e) {
        System.err.println(e.getMessage());
      }
```

```
    }
// --- //

public void setCoordonnee(Coordonnee coordonnee) {
      try {
        if (coordonnee == null) {
          throw new Exception("Barriere setCoordonnee() - Les coordonnees a changer doivent
exister.");
        }
        else {
          int x3,y3;
          boolean[][] damier = this.plateau.getDamier();
          int x1 = coordonnee.getX1();
          int x2 = coordonnee.getX2();
          int y1 = coordonnee.getY1();
          int y2 = coordonnee.getY2();

          if (coordonnee.getX1() < coordonnee.getX2()) {
            x3 = coordonnee.getX2() - 1;
          }
          else {
            x3 = coordonnee.getX2() + 1;
          }

          if (coordonnee.getY1() < coordonnee.getY2()) {
            y3 = coordonnee.getY2() - 1;
          }
          else {
            y3 = coordonnee.getY2() + 1;
          }


          if ((damier[x1][y1]) && (damier[x2][y2]) && (damier[x3][y3])) {
            this.coordonnee = coordonnee;
            ArrayList<int[]> aChanger = new ArrayList<int[]>();
            int[] tab1 = {x1,y1};
            int[] tab2 = {x2,y2};
            int[] tab3 = {x3,y3};
            aChanger.add(tab1);
            aChanger.add(tab2);
            aChanger.add(tab3);
            this.plateau.setDisponibilite(aChanger);
          }
        }
      }
      catch(Exception e) {
        System.err.println();
      }
    }
```

*Classe Coordonnee*

```
  public Coordonnee(int x1, int y1, int x2, int y2) {
```

```
      try {
        if ((x1 < 0) || (x1 > 17) || (x2 < -1) || (x2 > 17) || (y1 < 0) || (y1 > 17) || (y2
< -1) || (y2 > 17)) {
          throw new Exception("Coordonnee constructeur - Les coordonées doivent comprises
entre 0 (ou -1 s'il s'agit d'un pion) et 17.");
        }
        else {
          this.x1 = x1;
          this.y1 = y1;
          this.x2 = x2;
          this.y2 = y2;
        }
      }
      catch(Exception e) {
        System.err.println(e.getMessage());
      }
    }
```

*Classe Humain*

```
public void jeu() {
      try {
        int[][] deplacementsPossibles = this.pion.getDeplacementPossibles();
        String nPosition = this.scanner.nextLine();
        boolean ok = false;
        String[] letters = {"A","B","C","D","E","F","G","H","I"};
        while (!nPosition.trim().equalsIgnoreCase("pass") && !ok) {
          if (nPosition.trim().equalsIgnoreCase("help")) {

          }
          else if (nPosition.split(" ")[0].trim().equalsIgnoreCase("move")) {
            for (int[] deplacement : this.pion.getDeplacementPossibles()) {
              if (deplacement[0] == Integer.parseInt(nPosition.split(" ")[1].split(",")
[0].split("(")[1].trim())) {
                if (deplacement[1] == Array.asList(letters).indexOf(nPosition.split(" ")
[1].split(",")[1].split(")")[0].trim())) {
                  deplacerPion(new Coordonnee(Integer.parseInt(nPosition.split(" ")
[1].split(",")[0].split("(")[1].trim()),Array.asList(letters).indexOf(nPosition.split(" ")
[1].split(",")[1].split(")")[0].trim()),-1,-1));
                }
              }
            }
          }
          else {
            nPosition = this.scanner.nextLine();
          }
        }
      }
      catch (NumberFormatException e) {
        System.err.println("Erreur dans le format des coordonnees, tapez 'help' pour plus
d'informations");
      }
```

```
    }
```

*Classe Joueur*

```
protected boolean humain;

// --- //

public void deplacerPion(Coordonnee coordonnee, boolean[][] damier) {
      try {
        if (damier == null || coordonnee == null) {
          throw new Exception ("Erreur deplacerPion(), parametre null");
        }
        else {
          damier[this.pion.getCoordonnee().getX1()][this.pion.getCoordonnee().getY1()] =
true;
          damier[coordonnee.getX1()][coordonnee.getY1()] = false;
          this.pion.setCoordonnee(coordonnee);
        }
      }
      catch (Exception e) {
        System.err.println(e.getMessage());
      }

    }

// --- //

public boolean isHumain() {
      return this.humain;
    }
```

*Classe Partie*

```
private ArrayList<Barriere> barrieres;

// --- //

public void addBarriere (Barriere barriere) {
        try {
          if (barriere == null) {
            throw new Exception ("Erreur addBarriere(), parametre null");
          }
          else {
            this.barrieres.add(barriere);
          }
        }
        catch (Exception e) {
          System.err.println(e.getMessage());
        }
      }
```

*Classe Pion*

```
public void nextCoup(boolean[][] damier) {
      int x = this.coordonnee.getX1();
      int y = this.coordonnee.getY1();
      ArrayList<int[]> temp = new ArrayList<int[]>();
      int[] deplacement;
      for (int i = -2 ; i <= 2 ; i++) {
        for (int j = -2 ; j <= 2 ; j++) {
          if (i % 2 == 0 && j % 2 == 0) {
            if (damier[x+i][y+j] == true && damier[x+((int)i/2)][y+((int)j/2)] == true) {
              deplacement = new int[2];
              deplacement[0] = x+i;
              deplacement[1] = y+j;
              temp.add(deplacement);
            }
          }
        }
      }
      this.deplacementPossibles = new int[temp.size()][temp.size()];
      for (int i = 0 ; i < temp.size() ; i++) {
        this.deplacementPossibles[i] = temp.get(i);
      }
    }
```

*Classe Plateau*

```
private boolean[][] DAMIER;

// --- //

  public Plateau(int taille) {
      try {
        if (taille < 2) {
          throw new Exception("Erreur Plateau(), taille trop petite");
        }
        else {
          this.TAILLE = taille;
          this.DAMIER = new boolean[taille][taille];
          for (int i = 0 ; i < this.TAILLE ; i++) {
            for (int j = 0 ; j < this.TAILLE ; j++) {
              this.DAMIER[i][j] = true;
            }
          }
        }
      }
      catch (Exception e) {
        System.err.println(e.getMessage());
      }

    }
```

```
// --- //

public String toString(ArrayList<Pion> listePion , ArrayList<Barriere> listeBarriere) {
      String ret = "";
      try  {
        ret += "\n\n\n";
        ret += "\t\t   1    2    3    4    5    6    7    8    9 \n";
        String[] letters = {"A","B","C","D","E","F","G","H","I"};
        String icon =  "";
        for (int i = 0 ; i < this.TAILLE ; i++) {
          ret += "\n\t\t    _    _    _    _    _    _    _    _    _  \n";
          ret += "\t\t" + letters[i] + "  ";
          for (int j = 0 ; j < this.TAILLE ; j++) {
            for (Pion p : listePion) {
              if (p.getCoordonnee().getX1() == i && p.getCoordonnee().getY1() == j) {
                if (p.getCouleur().length() == 1) {
                  icon = p.getCouleur();
                }
              }
            }
            ret += "|"+icon+"|  ";
          }
          ret += "\n\t\t    -    -    -    -    -    -    -    -    -  \n";
          icon = "";
        }
      }
      catch (NullPointerException e) {
        ret = "";
        System.err.println("Erreur, liste de pions null");
      }
      finally {
        return ret;
      }
    }

// --- //

public void setDisponibilite (ArrayList<int[]> listCases) {
        for (int[] co : listCases) {
          this.DAMIER[co[0]][co[1]] = !this.DAMIER[co[0]][co[1]];
        }
      }
```

*Classe RWFile*

```
package utilities;
import java.util.ArrayList;
import java.io.*;
import quoridor.*;

public class RWFile {

  /**
```

```
    * Lit et retourne toutes les informations contenues dans le fichier donné
    * @param fileName le nom du fichier que nous voulons lire
    * @return toutes les informations contenues dans le fichier sous la forme d'un tableau
de String.
    */
  public static ArrayList<String> readFile(String fileName) {
    ArrayList<String> liste = new ArrayList<String>();
    try {
      DataInputStream dataIn = new DataInputStream(new FileInputStream("../data/" +
fileName));
      while (dataIn.available() > 0) {
        liste.add(dataIn.readLine());
      }
    }
    catch (FileNotFoundException e) {
      System.out.println("RWFile readFile() - Fichier non trouvé : " + fileName);
    }
    finally {
      return liste;
    }
  }

  /**
    * Ecrit toutes les informations utiles concernant la partie en cours dans le fichier
donné.
    * @param fileName le nom du fichier dans lequel nous voulons écrire
    * @param joueurs la liste des joueurs dans la partie
    * @param barrieres la liste des barrières déjà jouées sur le plateau
    * @param tour le numéro du dernier tour joué
    * @param dernierJoueur le joueur ayant sauvegardé (Donc celui qui jouera lors de la
reprise de la partie)
    */
  public static void writeFile(String fileName, ArrayList<Joueur> joueurs,
ArrayList<Barriere> barrieres, int tour, Joueur dernierJoueur) {
    try {
      if (fileName == null) {
        throw new Exception("RWFile writeFile() - Le nom du fichier doit exister");
      }
      else if (joueurs == null) {
        throw new Exception("RWFile writeFile() - La liste des joueurs doit exister");
      }
      else if (tour < 0) {
        throw new Exception("RWFile writeFile() - Le numéro d'un tour doit être positif");
      }
      else if (dernierJoueur == null) {
        throw new Exception("RWFile writeFile() - Le dernier joueur ayant joué doit
exister");
      }
      else {
        DataOutputStream dataOut = new DataOutputStream(new FileOutputStream ("../data/" +
fileName));

        for (Joueur j : joueurs) {
```

```
          dataOut.writeUTF(j.getNom());
          if (j.isHumain()) {
            dataOut.writeUTF(" H ; ");
          }
          else {
            IA ia = (IA)j;
            dataOut.writeUTF(" IA " + ia.getDifficulte() + " ; ");
          }
        }
        dataOut.writeUTF("\n");

        for (Joueur j : joueurs) {
          Pion p = j.getPion();
          Coordonnee coord = p.getCoordonnee();
          dataOut.writeUTF(p.getCouleur() + " " + coord.getX1() + " " + coord.getY1() + " ;
");
        }
        dataOut.writeUTF("\n");

        for (Joueur j : joueurs) {
          ArrayList<Barriere> lesBarrieres = j.getBarrieres();
          dataOut.writeUTF(String.valueOf(lesBarrieres.size()));
        }

        for (Barriere b : barrieres) {
          Coordonnee coord = b.getCoordonnee();
          dataOut.writeUTF(b.getCouleur() + " " + coord.getX1() + " " + coord.getY1() + " "
+ coord.getX2() + " " + coord.getY2());
        }
        dataOut.writeUTF("\n");

        dataOut.writeUTF(tour + " ; " + String.valueOf(dernierJoueur.getNumero()));
        dataOut.close();
      }
    }
    catch(FileNotFoundException e) {
      System.err.println("WriteFile - Fichier non trouvé : " + fileName);
    }
    catch (Exception e) {
      System.err.println(e.getMessage());
    }
  }

}
```