Compte-rendu du projet "Échecs Martiens"

AUBERT-BÉDUCHAUD Julien, BOURGES Nicolas

Groupe 4-1

1 Introduction

A travers ce mini-projet, nous avons abordé le problème de la programmation objet de manière ludique en ayant pour sujet la création du jeu de plateau "échec martiens" dont les règles sont présentes à cette adresse. La problématique étant de simuler le jeu au moyen d'un diagramme de classe pré-défini, nous avons suivi cette règle, sauf pour deux méthodes de la classe Jeu, que nous détaillerons par la suite.
Afin de réaliser ce projet, nous avons utilisé le langage Java au moyen du logiciel de développement Eclipse, Github afin de mettre le travail et commun ainsi le logiciel TeXstudio pour la conception de ce document.

2 Les classes

2.1 La classe Case

Cette classe comporte deux attributs, le joueur ainsi que le pion qui la compose. Celle-ci va servir à créer les cases de notre aire de jeu.
On y trouve un constructeur et cinq méthodes :

```
public Case(){
   this.joueur = null;
   this.pion = null;
}
```

Ce constructeur détermine la valeur par défaut de la case, à savoir, pas de joueur ni de pion.

```
public boolean estLibre(){
   return (this.pion == null);
}
```

Renvoie si oui ou non la case est libre. On regarde donc si la valeur pion est nulle.

```
public Joueur getJoueur(){
   return this.joueur;
}
```

On renvoie le joueur de la case.

```
public Pion getPion(){
   return this.pion;
}
```

On renvoie le pion de la case.

```
public void setJoueur(Joueur joueur){
   this.joueur = joueur;
}
```

On affecte à l'attribut joueur le joueur donné en entrée.

```
public void setPion(Pion pion){
   this.pion = pion;
}
```

On affecte à l'attribut pion le pion donné en entrée.

2.2 La classe Coordonnee

Cette classe comporte deux attributs entiers, une coordonnée X et une coordonnée Y. Elle va servir à stocker les coordonnées des pions dans des listes.
On trouve un constructeur et deux accesseurs :

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

On associe aux attributs x et y les entiers entrés x et y.

```
public int getX(){
   return this.x;
}
```

On retourne l'attribut X.

```
public int getY(){
   return this.y;
}
```

On retourne l'attribut Y.

2.3 La classe Plateau

Cette classe permet de créer le plateau de jeu, une matrice de cases possédant les attributs tailleHorizontale et tailleVerticale, permettant de définir une aire de jeu.

On y trouve un constructeur, trois accesseurs et deux méthodes :

```
public Plateau(int saTailleHorizontale, int saTailleVerticale){
   this.tailleHorizontale = saTailleHorizontale;
   this.tailleVerticale = saTailleVerticale;
   this.plateau = new Case[this.tailleHorizontale][this.tailleVerticale];
}
```

Les tailles horizontales et verticales du plateau sont données en entrée. Celle-ci permettent la création de la matrice de cases "plateau" qui possède ces dimensions.

```
public Case[][] getCases(){
   return this.plateau;
}
```

Permet de retourner le plateau de jeu.

```
public int getTailleHorizontale(){
   return this.tailleHorizontale;
}
```

Permet de retourner la taille horizontale du plateau.

```
public int getTailleVerticale(){
   return this.tailleVerticale;
}
```

Permet de retourner la taille verticale du plateau.

```
public void initialiser(){
   //Initialisation des cases du plateau
   for (int i = 0; i < this.tailleHorizontale; i++){
      for (int j = 0; j < this.tailleVerticale; j++){
         this.plateau[i][j] = new Case();
      }
   }
   //Emplacement des pions par defaut
   //Partie Haute
   this.plateau[0][0].setPion(new GrandPion());
   this.plateau[1][0].setPion(new GrandPion());
   this.plateau[0][1].setPion(new GrandPion());
   this.plateau[2][0].setPion(new MoyenPion());
   this.plateau[1][1].setPion(new MoyenPion());
   this.plateau[0][2].setPion(new MoyenPion());
```

```
        this.plateau[2][1].setPion(new PetitPion());
        this.plateau[2][2].setPion(new PetitPion());
        this.plateau[1][2].setPion(new PetitPion());
        //Partie Basse
        this.plateau[3][7].setPion(new GrandPion());
        this.plateau[2][7].setPion(new GrandPion());
        this.plateau[3][6].setPion(new GrandPion());
        this.plateau[1][7].setPion(new MoyenPion());
        this.plateau[2][6].setPion(new MoyenPion());
        this.plateau[3][5].setPion(new MoyenPion());
        this.plateau[1][6].setPion(new PetitPion());
        this.plateau[1][5].setPion(new PetitPion());
        this.plateau[2][5].setPion(new PetitPion());
    }
```

Cette méthode permet d'initialiser le plateau de jeu. Tout d'abord, pour chaque instance de la matrice, on va y associer une case. Celle-ci est par défaut vide et non-attribuée, lié à la façon dont elle a été construite dans Case().
Ensuite, on va attribuer aux cases prédéfinie leurs pions correspondant, comme expliqué dans les règles du jeu.

```
    //Cf programme, les caracteres unicode semblent ne pas marcher dans un lstlisting
        LaTeX, ces derniers sont ici remplaces par "?"
       public String toString(){
          String chaine = "";
          chaine += "????????????\n";
          for (int j=0; j < this.tailleVerticale; j++){
             chaine += "?";
             for (int i=0; i < this.tailleHorizontale; i++){
                if (this.plateau[i][j].getPion() == null){
                   chaine +=" ";
                }else{
                   chaine += this.plateau[i][j].getPion().toString();
                }
                if (i != 3){
                   chaine += "?";
                }else{
                   chaine += "?";
             }
          }
          if (j == 7){
             chaine += "\n????????????\n";
          }else if (j == 3){
             chaine += "\n????????????\n";
          }else{
             chaine += "\n????????????\n";
          }
       }
       return chaine;
    }
```

Cette méthode consiste en l'affichage du plateau de jeu. On commence par créer une chaine de caractères initialisée par le haut du plateau, à laquelle on va par la suite ajouter le contenu. A chaque ligne, on débute par le cadre du plateau, ensuite cette ligne prend la chaine " " si il n'y a pas de pion, sinon le symbole de celui-ci. On détecte ensuite si la ligne est finie pour fermer le plateau. Les retours à la ligne engendre une ligne de séparation qui diffère selon l'emplacement du plateau.

2.4 La classe Pion

Il s'agit de la super-classe qui va permettre de créer différents types de pions. On doit importer le type Liste.
On y trouve deux méthodes :

```
public abstract int getScore();
```

Méthode abstraite permettant d'obtenir la valeur du score du pion.

```
public abstract Liste getDeplacement(int coordDepartX, int coordDepartY, int
    coordArriveeX, int coordArriveeY);
```

Méthode abstraite permettant d'obtenir la liste de déplacements d'un pion allant de son point de départ à celui d'arrivée.

2.5 La classe PetitPion

Sous-classe de Pion, permet de définir les déplacement du pion de petite taille et sa valeur. L'import d'une Liste y est nécessaire.
On y trouve deux accesseurs et un toString :

```
public Liste getDeplacement(int coordDepartX, int coordDepartY, int coordArriveeX,
    int coordArriveeY ){
   Liste deplacements = new Liste();
   if ((Math.abs(coordArriveeX - coordDepartX) == 1) && (Math.abs(coordArriveeY -
       coordDepartY) == 1)){
      deplacements.add(new Coordonnee(coordArriveeX,coordArriveeY));
   }
   return deplacements;
}
```

Le pion se déplaçant diagonalement et d'une case, on cherche à savoir si la différence de coordonnées est de 1 pour X et Y. Pour cela, on utilise Math.abs() permettant d'obtenir la valeur absolue, ce qui permet de trouver l'écart entre le départ et l'arrivée sans tenir compte de sa direction dans le plateau.

```
public int getScore(){
   return 1;
}
```

On retourne la valeur du pion directement sous forme d'entier soit 1 point.

```
//Cf programme, les caracteres unicode semblent ne pas marcher dans un lstlisting
    LaTeX, ces derniers sont ici remplaces par "?"
public String toString(){
   String chaine;
   chaine = "?";
   return chaine;
}
```

On redéfinie la fonction toString() qui sera appelée par Plateau. De ce fait, au lieu d'afficher une case vide, on affichera le caractère " ?" dans la case.

2.6 la classe GrandPion

Sous-classe de Pion, permet de définir les déplacement du pion de grande taille et sa valeur. Le déplacement y est déterminé par 3 accesseurs vérifiant si déplacement horizontal, vertical ou diagonal est possible. Ces derniers renvoient le cas échéant une liste de déplacement. Ces trois méthodes seront ensuite testées par getDeplacement(). On doit importer Liste, celle-ci étant présente.
On y trouve cinq accesseurs et un toString :

```
protected Liste getCheminHorizontal(int coordDepartX, int coordDepartY, int
    coordArriveeX, int coordArriveeY ){
   Liste deplacements = new Liste();
   if (coordArriveeX-coordDepartX != 0 && coordArriveeY-coordDepartY == 0){
      int compteur = Math.abs(coordArriveeX-coordDepartX);
      if (coordArriveeX < coordDepartX){
         for(int i=1;i<=compteur;i++){
            deplacements.add(new Coordonnee(coordDepartX-i,coordDepartY));
         }
      }
      if (coordArriveeX > coordDepartX){
         for(int i=1;i<=compteur;i++){
            deplacements.add(new Coordonnee(coordDepartX+i,coordDepartY));
         }
      }
   }
   return deplacements;
}
```

On vérifie si le pion se déplace de n cases horizontalement, si c'est le cas, on stocke la différence entre l'arrivée et le départ dans une variable "compteur" qui va permettre de déterminer les coordonnées parcourues. Dans le cas ou la coordonnée d'arrivée est supérieure à celle de départ,

on ajouter à la liste de déplacement toutes les valeurs permettant d'arriver à l'arrivée, via l'incrémentation. Dans le cas opposé, ces valeurs décrémentent.

```
protected Liste getCheminVertical(int coordDepartX, int coordDepartY, int
    coordArriveeX, int coordArriveeY ){
   Liste deplacements = new Liste();
   if (coordArriveeX-coordDepartX == 0 && coordArriveeY-coordDepartY != 0){
      int compteur = Math.abs(coordArriveeY-coordDepartY);
      if (coordArriveeY < coordDepartY){
         for(int i=1;i<=compteur;i++){
            deplacements.add(new Coordonnee(coordDepartX,coordDepartY-i));
         }
      }
      if (coordArriveeY > coordDepartY){
         for(int i=1;i<=compteur;i++){
            deplacements.add(new Coordonnee(coordDepartX,coordDepartY+i));
         }
      }
   }
   return deplacements;
}
```

Même principe que précédemment, mais pour les coordonnées verticales.

```
protected Liste getCheminDiagonal(int coordDepartX, int coordDepartY, int
    coordArriveeX, int coordArriveeY ){
   Liste deplacements = new Liste();
   if (coordArriveeX - coordDepartX == coordArriveeY - coordDepartY &&
       coordArriveeX - coordDepartX != 0){
      int compteur = Math.abs(coordArriveeX-coordDepartX);
      if (coordArriveeX < coordDepartX && coordArriveeX < coordArriveeY){
         for(int i=1;i <= compteur;i++){
            deplacements.add(new Coordonnee(coordDepartX-i,coordDepartY-i));
         }
      }
      if (coordArriveeX < coordDepartX && coordArriveeX > coordArriveeY){
         for(int i=1;i <= compteur;i++){
            deplacements.add(new Coordonnee(coordDepartX-i,coordDepartY+i));
         }
      }
      if (coordArriveeX > coordDepartX && coordArriveeX < coordArriveeY){
         for(int i=1;i <= compteur;i++){
            deplacements.add(new Coordonnee(coordDepartX+i,coordDepartY-i));
         }
      }
      if (coordArriveeX > coordDepartX && coordArriveeX > coordArriveeY){
         for(int i=1;i <= compteur;i++){
            deplacements.add(new Coordonnee(coordDepartX+i,coordDepartY+i));
         }
      }
   }
```

```
        return deplacements;
    }
```

On vérifie que les coordonnées parcourues par le pion sont les mêmes pour l'axe horizontal et vertical mais également qu'il y a eu un déplacement. S'ensuit alors la détection du mouvement et la mise dans la liste de celui-ci.

```
    public Liste getDeplacement(int coordDepartX, int coordDepartY, int coordArriveeX,
        int coordArriveeY ){
       Liste deplacements = new Liste();
       if (coordArriveeX-coordDepartX != 0 && coordArriveeY-coordDepartY == 0){
          deplacements = this.getCheminHorizontal(coordDepartX, coordDepartY,
              coordArriveeX, coordArriveeY);
       }
       if (coordArriveeX-coordDepartX == 0 && coordArriveeY-coordDepartY != 0){
          deplacements = this.getCheminVertical(coordDepartX, coordDepartY,
              coordArriveeX, coordArriveeY);
       }
       if (coordArriveeX - coordDepartX == coordArriveeY - coordDepartY &&
           coordArriveeX - coordDepartX != 0){
          deplacements = this.getCheminDiagonal(coordDepartX, coordDepartY,
              coordArriveeX, coordArriveeY);
       }
       return deplacements;
    }
```

On vérifie pour chacune des trois méthodes précédentes sa possibilité. Si l'une d'entre elle marche elle modifie la liste et les autres ne modifieront pas. Si aucune de ces méthodes ne marche, la liste sera nulle. On retourne ensuite la liste.

```
    public int getScore(){
       return 3;
    }
```

Retourne la valeur du score du grand pion, soit 3.

```
    //Cf programme, les caracteres unicode semblent ne pas marcher dans un lstlisting
        LaTeX, ces derniers sont ici remplaces par "?"
    public String toString(){
       String chaine;
       chaine = "?";
       return chaine;
    }
```

On redéfinie la fonction toString() qui sera appelée par Plateau. De fait, au lieu d'afficher une case vide, on affichera le caractère " ?" dans la case, symbolisant le grand pion.

2.7 La classe MoyenPion

Cette classe est une sous-classe de GrandPion. En effet, le pion moyen fonctionne comme le grand pion, à la différence qu'il ne se déplace qu'au maximum de deux cases. Avant d'appeler la méthode getDeplacement de GrandPion, il faut donc vérifier si le déplacement demandé est de 2 cases maximum. Une liste étant utilisée, l'import de Liste est nécessaire.
On y trouve 2 accesseurs, une méthode et un toString.

```
    public boolean longueurOk(int coordDepartX, int coordDepartY, int
        coordArriveeX, int coordArriveeY){
      if (Math.abs(coordArriveeX - coordDepartX) <= 2 && coordArriveeY -
          coordDepartY == 0){
        return true;
      }else if (coordArriveeX - coordDepartX == 0 && Math.abs(coordArriveeY -
          coordDepartY) <= 2){
        return true;
      }else if (Math.abs(coordArriveeX - coordDepartX) == Math.abs(coordArriveeY -
          coordDepartY) && Math.abs(coordArriveeX - coordDepartX) <= 2){
        return true;
      }
      return false;
    }
```

Cette méthode permet de vérifier les trois possibilités de déplacements et leur longueur. Si on à un déplacement de deux cases horizontalement uniquement, on renvoie true car la longueur est possible, de même pour le cas vertical. Pour le cas diagonal, on vérifie si le pion se déplace au maximum de deux cases à la fois horizontalement et verticalement. Si oui, alors la longueur est correcte. Les valeurs absolues permettent de vérifier les déplacements pouvant avoir lieux dans différents sens.
Si aucune de ces conditions n'est vérifiée, alors on renvoie false, la longueur n'étant pas correcte.

```
  public Liste getDeplacement(int coordDepartX, int coordDepartY, int
      coordArriveeX, int coordArriveeY){
    Liste deplacements = new Liste();
    if (this.longueurOk(coordDepartX, coordDepartY, coordArriveeX, coordArriveeY)
        == true){
      this.deplacements = super.getDeplacement(coordDepartX, coordDepartY,
          coordArriveeX, coordArriveeY);
    }
    return deplacements;
  }
```

L'accesseur appelle la méthode getDeplacement() de GrandPion si la longueur est correcte. On renvoie donc une liste de coordonnées de déplacement.

```
  public int getScore(){
    return 2;
```

```
    }
```

Retourne la valeur du score du pion de taille moyenne, soit 2.

```
    //Cf programme, les caracteres unicode semblent ne pas marcher dans un lstlisting
        LaTeX, ces derniers sont ici remplaces par "?"
    public String toString(){
       String chaine;
       chaine = "?";
       return chaine;
    }
```

On redéfinie la fonction toString() qui sera appelée par Plateau. De ce fait, au lieu d'afficher une case vide, on affichera le caractère " ?" dans la case.

2.8 La classe Joueur

Cette classe définie les joueurs de la partie. Ces derniers ont un pseudo, qui est une chaine de caractères, et une liste de pions capturés. L'import de Liste est donc nécessaire.
On y trouve trois accesseurs et trois méthodes :

```
    public Joueur(java.lang.String pseudo){
       this.pseudo = pseudo;
       this.pionsCaptures = new Liste();
    }
```

On a en entrée une chaine de caractère qui sera déterminé par le joueur. Celle-ci servira de pseudo au joueur. De plus, la liste des pions capturés du joueur est au début nulle.

```
    public Liste getPionsCaptures(){
       return this.pionsCaptures;
    }
```

On retourne la liste des pions capturés du joueur.

```
    public java.lang.String getPseudo(){
       return this.pseudo;
    }
```

On retourne le pseudonyme du joueur.

```
    public void ajouterPionCapture(Pion pion){
```

```
        this.pionsCaptures.add(pion);
    }
```

On ajoute à la liste de capture du joueur un pion donné en entrée.

```
    public int getNbPionsCaptures(){
        if (this.pionsCaptures == null){
            return 0;
        }
        return this.pionsCaptures.size();
    }
```

On retourne la taille de la liste de capture du joueur. La méthode size() ne pouvant calculer une liste vide, si celle-ci l'est, on retourne 0 à la place.

```
    public boolean equals(java.lang.Object o){
        if (o instanceof Joueur){
            return this.pseudo.equals(((Joueur)o).pseudo);
        }
        return false;
    }
```

Avec cette méthode, on vérifie si le pseudo du joueur est égal à celui d'un autre. En entré, on a l'objet que l'on souhaite comparer, objet que l'on cast comme joueur, permettant de comparer le nom du joueur à celui en entrée. On retourne true si les noms sont égaux, sinon false. Cette méthode sera utile pour comparer les joueurs durant la sélection de pseudonyme et pour déterminer si le joueur capture son propre pion.

```
    public int calculerScore(){
        int score;
        score = 0;
        if (this.getNbPionsCaptures() != 0){
            for (int i = 0; i < this.pionsCaptures.size(); i++){
                score += ((Pion)this.pionsCaptures.get(i)).getScore();
            }
        }
        return score;
    }
```

Via cette méthode, on vérifie le score d'un joueur. Tout d'abord, si le nombre de pions capturés du joueur est égal à 0, on retourne un score de 0. Sinon, on va pour chaque objet de la liste, que l'on cast comme pion, récupérer le score de celui-ci via getScore().

2.9 La classe Jeu

Cette classe contient la base de ce qui compose une partie d'échecs martiens. Celle-ci possède les attributs joueur1 et 2, les joueurs de la partie, un plateau mais également les coordonnées du pion traversant le plateau et changeant de camp afin d'empêcher son déplacement au tour suivant comme mentionnée dans les règles. L'importation de Liste est nécessaire, des listes servant à améliorer la lisibilité ayant été utilisées.
Néanmoins, deux des méthodes présentes dans le diagramme de classes n'ont pas été utilisées. Il s'agit des méthodes Jouer() et InitialiserJoueur(). Ayant utilisé l'outil scanner pour les entrées de la classe TestJeu, nous ne savions pas comment implémenter l'initialisation que nous y avons effectué au sein de ces deux méthodes. La structure de ces méthodes est présente dans la classe mais ne possèdent pas de contenu.
De ce fait, on trouve dans cette classe un constructeur, cinq méthodes ainsi qu'un toString().

```
public Jeu(Joueur sonJoueur1, Joueur sonJoueur2){
   this.joueur1 = sonJoueur1;
   this.joueur2 = sonJoueur2;
   this.plateau = null;
}
```

les deux joueurs prennent la valeur que l'on donnera en entrée via le clavier. Le plateau est pour l'instant vide.

```
public void initialiserPlateau(){
   this.plateau = new Plateau(4,8);
   this.plateau.initialiser();
   for (int i = 0; i < this.plateau.getTailleHorizontale(); i++){
      for (int j = 0; j < this.plateau.getTailleVerticale()/2; j++){
         plateau.getCases()[i][j].setJoueur(joueur1);
      }
   }
   for (int i = 0; i < this.plateau.getTailleHorizontale(); i++){
      for (int j = this.plateau.getTailleVerticale()/2; j <
          this.plateau.getTailleVerticale(); j++){
         this.plateau.getCases()[i][j].setJoueur(joueur2);
      }
   }
}
```

On commence par définir l'espace de jeu, soit un plateau de 4x8. On appelle ensuite l'initialisation faite dans la classe plateau afin de disposer les pions par défaut. Après cela, on va définir les zones des deux joueurs, le premier obtenant la partie haute du plateau et le second la partie basse.

```
private boolean deplacementPossible(int coordDepartX, int coordDepartY, int
    coordArriveeX, int coordArriveeY, Joueur joueur){
```

```
            //PRECONDITION 1
            if(coordDepartX < 0 || coordDepartX > 3 || coordDepartY < 0 ||
                coordDepartY > 7){
               return false;
            }
            if(coordArriveeX < 0 || coordArriveeX > 3 || coordArriveeY < 0 ||
                coordArriveeY > 7){
               return false;
            }

            //variables permettant d'alleger le code des lignes suivantes
            int i = 0;
            Case raccourciCaseD = this.plateau.getCases()[coordDepartX][coordDepartY];
            Case raccourciCaseA=this.plateau.getCases()[coordArriveeX][coordArriveeY];

            //check de l'existence du pion
            if(raccourciCaseD.getPion() == null){
               System.out.println("Veuillez deplacer un pion");
               return false;
            }

            //PRECONDITION 2
            if(raccourciCaseD.getJoueur().equals(joueur) == false){
              System.out.println("Ce pion ne vous appartient pas");
              return false;
            }

            //PRECONDITION 3
            if(raccourciCaseD.equals(raccourciCaseA) == true &&
                raccourciCaseA.estLibre() == false){
               System.out.println("Vous ne pouvez pas capturer votre propre pion");
               return false;
            }

            //variables permettant d'alleger le code des lignes suivantes
            Liste raccourciDeplacementD = raccourciCaseD.getPion()
            .getDeplacement(coordDepartX,coordDepartY,coordArriveeX,coordArriveeY);
            Coordonnee RGetI = ((Coordonnee)raccourciDeplacementD.get(i));

            //PRECONDITION 4
            if (raccourciDeplacementD.size() > 1){
               for (int i=0; i < raccourciDeplacementD.size(); i++ ){
                  if (this.plateau.getCases()[RGetI.getX()][RGetI.getY()].estLibre() ==
                      false){
                     System.out.println("Impossible de sauter un autre pion");
                    return false;
                  }
               }
            }

            if (raccourciDeplacementD.size() == 0) {
               System.out.println("Ce mouvement est impossible.");
               return false;
```

```
            }
            //PRECONDITION 5
            if (this.pionBloque != null){
               if (this.pionBloque.getX() == coordDepartX && this.pionBloque.getY() ==
                    coordDepartY){
                  System.out.println("Vous ne pouvez pas deplacer ce pion tout de
                      suite.");
                  return false;
               }
            }
            return true;
         }
```

Cette méthode permet de vérifier les cinq pré-conditions au déplacement d'un pion. Si celui-ci est possible, on retourne true, sinon false.
4 variables de lisibilité sont présentes ainsi que l'initialisation de i, nécessaire à une d'elles.
Cette méthode est divisé par pré-conditions, soit :

1. Le pion doit être et rester dans l'aire de jeu. On vérifie donc que les pion de départ et d'arrivée respectent la taille de l'aire de jeu, sinon on renvoie false.
2. Après cela on vérifie qu'il existe bel et bien un pion sur la case.
3. On vérifie que le pion déplacé appartienne au joueur. Pour cela, on compare le joueur de la case au joueur actif. Si ils sont différents, on renvoie false.
4. On vérifie si le joueur ne tente pas de capturer son propre pion. Pour cela on compare le joueur de la case de départ à celle d'arrivée. Si le joueur est similaire, alors on renvoie false.
5. Permet de vérifier si le pion ne saute pas un autre pion avant l'arrivée. Vérifie également si le déplacement fourni est disponible.
   On va pour chaque coordonnées de la liste de déplacements vérifier si celles-ci représentent une case occupée. Si une paire de ces coordonnées est occupée par un pion, on retourne false.
6. On cherche à savoir si le pion que l'on cherche à déplacer n'a pas été changé de camp au tour précédant, empêchant ainsi le déplacement. Pour cela, on utilise une variable temporaire, "pionBloque" qui sera modifié dans la méthode déplacer() du Jeu. Si les coordonnées de départ correspondent à celles étant bloquées, on retourne false.

```
   public boolean deplacer(int coordDepartX, int coordDepartY, int coordArriveeX, int
       coordArriveeY, Joueur joueur){
      if (this.deplacementPossible(coordDepartX, coordDepartY, coordArriveeX,
          coordArriveeY, joueur) == false){
         return false;
      }else{
         Pion pionDeplace =
             this.plateau.getCases()[coordDepartX][coordDepartY].getPion();
         this.plateau.getCases()[coordDepartX][coordDepartY].setPion(null);

         if (this.plateau.getCases()[coordArriveeX][coordArriveeY].estLibre() ==
             false){
            joueur.ajouterPionCapture(this.plateau.getCases()
```

```
            [coordArriveeX][coordArriveeY].getPion());
            this.plateau.getCases()[coordDepartX][coordDepartY].setPion(null);
            this.plateau.getCases()[coordArriveeX][coordArriveeY].setPion(pionDeplace);
         }else{
            this.plateau.getCases()[coordArriveeX][coordArriveeY].setPion(pionDeplace);
         }

         if (this.plateau.getCases()[coordDepartX][coordDepartY].getJoueur()
         .equals(this.plateau.getCases()[coordArriveeX][coordArriveeY].getJoueur())
             == false){
         this.pionBloque = new Coordonnee(coordArriveeX, coordArriveeY);
         }else{
            this.pionBloque = null;
         }
      }
      return true;
   }
```

Cette méthode effectue le déplacement des pions et renvoie true si il a été effectué, false sinon. On va d'abord vérifier si celui-ci est possible. De fait, si la méthode précédente renvoie false, on n'effectue pas de déplacement et on renvoi false. Sinon, on continue le déplacement.
On va d'abord stocker le pion déplacé dans une variable temporaire "pionDeplace" et le supprimer du plateau.
Ensuite, si la coordonnée d'arrivée est occupée, on va ajouter le pion sur la case à la liste des pions capturés du joueur puis le supprimer. On ajoute après cela le pion que l'on déplace sur la case.
Si la coordonnée n'est pas occupée, alors on se contente de placer le pion sur la case.

Après cela, on va effectuer une vérification concernant le pion, si celui-ci change de camp durant ce tour, alors on ajoute ses coordonnées d'arrivée dans pionBloque, que l'on utilisera dans la fonction précédente. Si il ne change pas de camp, on attribut à pionBloque la valeur nulle.
On retourne finalement true pour montrer que le déplacement à bien eu lieu.

```
    public Joueur joueurVainqueur(){
       Joueur gagne = null;
       if (this.joueur1.calculerScore() < this.joueur2.calculerScore()){
          gagne = joueur2;
       }else if (this.joueur1.calculerScore() > this.joueur2.calculerScore()){
          gagne = joueur1;
       }else if (this.joueur1.calculerScore() == this.joueur2.calculerScore()){
           gagne = null;
       }
       return gagne;
    }
```

Par cette méthode, on détermine le joueur qui remporte la partie. Si le score d'un des deux joueur est supérieur à l'autre, alors on retourne ce joueur. Si ils sont égaux, alors on retourne null.

```
    public boolean arretPartie(){
```

```
        int jouerJ1 = 0;
        Coordonnee pionDeplacableJ1 = null;
        int jouerJ2 = 0;
        Coordonnee pionDeplacableJ2 = null;

        for (int i = 0; i < this.plateau.getTailleHorizontale(); i++){
           for (int j = 0; j < this.plateau.getTailleVerticale()/2; j++){
              if (this.plateau.getCases()[i][j].estLibre() == false){
                 jouerJ1 ++;
                 pionDeplacableJ1 = new Coordonnee(i,j);
              }
           }
        }

        for (int i = 0; i < this.plateau.getTailleHorizontale(); i++){
           for (int j = this.plateau.getTailleVerticale()/2; j <
               this.plateau.getTailleVerticale(); j++){
              if (this.plateau.getCases()[i][j].estLibre() == false){
                 jouerJ2 ++;
                 pionDeplacableJ2 = new Coordonnee(i,j);
              }
           }
        }

        if (jouerJ1 == 1 && pionBloque != null){
           if (this.pionBloque.getX() == pionDeplacableJ1.getX() &&
               this.pionBloque.getY() == pionDeplacableJ1.getY()){
              pionDeplacableJ1 = null;
              jouerJ1 = 0;
           }
        }

        if (jouerJ2 == 1 && pionBloque != null){
           if (this.pionBloque.getX() == pionDeplacableJ2.getX() &&
               this.pionBloque.getY() == pionDeplacableJ2.getY()){
              pionDeplacableJ2 = null;
              jouerJ2 = 0;
           }
        }

        if (jouerJ1 == 0 && jouerJ2 == 0){
            return true;
         }
            return false;
      }
```

On cherche avec cette méthode à déterminer si oui ou non un joueur peut encore effectuer un déplacement. Pour cela, on utilise deux entier, "jouerJ1" et "jouerJ2", qui représentent le nombre de pions que les joueurs respectifs peuvent déplacer. A cela s'ajoute deux coordonnées, "pionDeplacableJ1" et "pionDeplacableJ2" qui serviront à vérifier si le seul pion dont disposent les joueurs peut se déplacer.
On va donc vérifier dans tout l'espace de jeu d'un joueur le nombre de pions disponibles. La paire

de coordonnées ne servira que si il ne reste qu'un pion disponible.
Après cela, on vérifie pour les deux joueurs le cas où il ne reste qu'un pion dans leur espace de jeu et on vérifie si celui-ci est bloqué, lié au changement de camp. Si c'est le cas, alors on passe le nombre de pion déplaçable du joueur à 0.
Si les deux joueurs ne peuvent déplacer de pions, alors on retourne true, sinon on retourne false.

```
public String toString(){
   String chaine = plateau.toString();
   chaine += "\nJoueur " + joueur1.getPseudo() + " : " + joueur1.calculerScore()
       +"\n";
   chaine += "\nJoueur " + joueur2.getPseudo() + " : " + joueur2.calculerScore()
       +"\n";
   return chaine;
}
```

On redéfini ici la méthode toString(). On appelle l'affichage du plateau auquel on ajoute les joueurs et leur score.

2.10 La classe TestJeu

Il s'agit de la classe permettant de tester le jeu. Afin d'utiliser les entrées du joueur, on importe la fonction scanner via "import java.util.Scanner;"
Voici le déroulement du programme :

```
public static Scanner input = new Scanner(System.in);
```

On défini la variable saisie au clavier, nommé ici input.

```
Joueur joueur1;
Joueur joueur2;
System.out.println("Entrez le pseudo du premier joueur");
String nom = input.nextLine();
joueur1 = new Joueur(nom);
System.out.println("Entrez le pseudo du deuxieme joueur");
do{
   nom = input.nextLine();
   joueur2 = new Joueur(nom);
   if (joueur2.equals(joueur1)){
      System.out.println("Les pseudos doivent etre differents, entrez a nouveau le
          pseudo du deuxieme joueur");
   }
}
while (joueur2.equals(joueur1) == true);
```

Dans le main, on commence par initialiser les joueurs. On entre le pseudo du joueur 1 puis celui du joueur 2. Le dowhile permet de vérifier si les deux pseudos sont différents. Tant qu'ils sont similaires, on redemande le pseudo du joueur 2.

```
Jeu jeu = new Jeu(joueur1, joueur2);
jeu.initialiserPlateau();
System.out.print(jeu.toString());
```

Après cela, on créé le jeu et on initialise le plateau, puis on l'affiche.

```
int departX, departY, arriveeX, arriveeY;
do{
   do{
      System.out.print("\nJoueur : " + joueur1.getPseudo() + "\n" + "Coordonnee
          depart X: ");
      departX = input.nextInt();
      System.out.print("Coordonnee depart y: ");
      departY = input.nextInt();
      System.out.print("Coordonnee arrivee x: ");
      arriveeX = input.nextInt();
      System.out.print("Coordonnee arrivee y: ");
      arriveeY = input.nextInt();
   }while(jeu.deplacer(departX, departY, arriveeX, arriveeY, joueur1) == false);
   System.out.print(jeu.toString());
   if (jeu.arretPartie() == true){
      break;
   }
   do{
      System.out.print("\nJoueur : " + joueur2.getPseudo() + "\n" + "Coordonnee
          depart X: ");
      departX = input.nextInt();
      System.out.print("Coordonnee depart y: ");
      departY = input.nextInt();
      System.out.print("Coordonnee arrivee x: ");
      arriveeX = input.nextInt();
      System.out.print("Coordonnee arrivee y: ");
      arriveeY = input.nextInt();
   }while(jeu.deplacer(departX, departY, arriveeX, arriveeY, joueur2) == false);
   System.out.print(jeu.toString());
}while (jeu.arretPartie()==false);
```

On déclare les coordonnées à saisir. Ensuite, tant que le jeu n'est pas terminé, on va demander les coordonnées au joueur 1 tant que celles-ci ne sont pas correctes, puis afficher le jeu. Si le joueur gagne après son déplacement, on arrête la boucle. On fait de même pour le second joueur.

```
System.out.print("Partie terminee ! Vainqueur: " +
    jeu.joueurVainqueur().getPseudo());
input.close();
```

Finalement, quand la partie est finie, on renvoie le vainqueur. On ferme la plage de saisie de l'utilisateur également.

3 Conclusion

Ce travail fut très intéressant car illustrant bien le principe de la programmation objet. Il nous a permis de mieux cerner son fonctionnement et de mieux maitriser le langage Java. Également, ce fut une opportunité d'utiliser le langage LaTeXpour la conception du compte-rendu.

Notre mini-projet semble fonctionnel, néanmoins, celui-ci peut présenter encore quelques bugs sur certains mouvements. Celui-ci effectue environs 95% des déplacements possibles avec certains engendrant des erreurs n'étant pas censée être affichée. Ce problème reste heureusement minime, les tests effectuée n'ayant pas engendré de crash du programme.

Ce projet, bien que difficile, fut utile nous permettant de développer tant nos compétences en programmation que la mise en place de situations pré-définies via diagramme de classe, même si nous avons pris quelques libertés avec deux méthodes et fut ainsi très enrichissant.