AMON Pauline
AUBRY Gatien
Groupe AA

Rapport de Projet Java

Trinôme

Introduction

Dans le cadre des TP du semestre 4, nous avons pour objectif de réaliser un projet en binôme parmi la liste des sujets présentés dans le document fourni au début du semestre. L'objectif de se projet est d'apprendre à réaliser un logiciel comportant une interface graphique en Java.

Nous avons choisi de réaliser le Projet du Trinôme. Ce projet consiste à reproduire un jeu de plateau donc le but consiste à capturer des pions adverses où à réunir des pions dans une zone. Ce jeu se joue à deux sur un cadrillage de 11x11 cases.

D'abord nous faisons le listing des unités de traitement puis le test unitaires des principales méthodes.

La première partie de ce projet a été réalisé à l'aide du logiciel Eclipse. Nous avons ensuite utilisé NetBeans.

Table des matières

Analyse du sujet

Après lecture du sujet « Trinôme » et avant de commencer à nous pencher sur la programmation, nous avons souligné et résumé les points importants pour en obtenir une liste de tâches qu'il nous sera nécessaire d'effectuer :

-Un Menu de démarrage dans lequel il est possible de choisir entre trois choix (lancer une nouvelle partie, lancer une nouvelle partie spéciale, charger une partie en, cours)

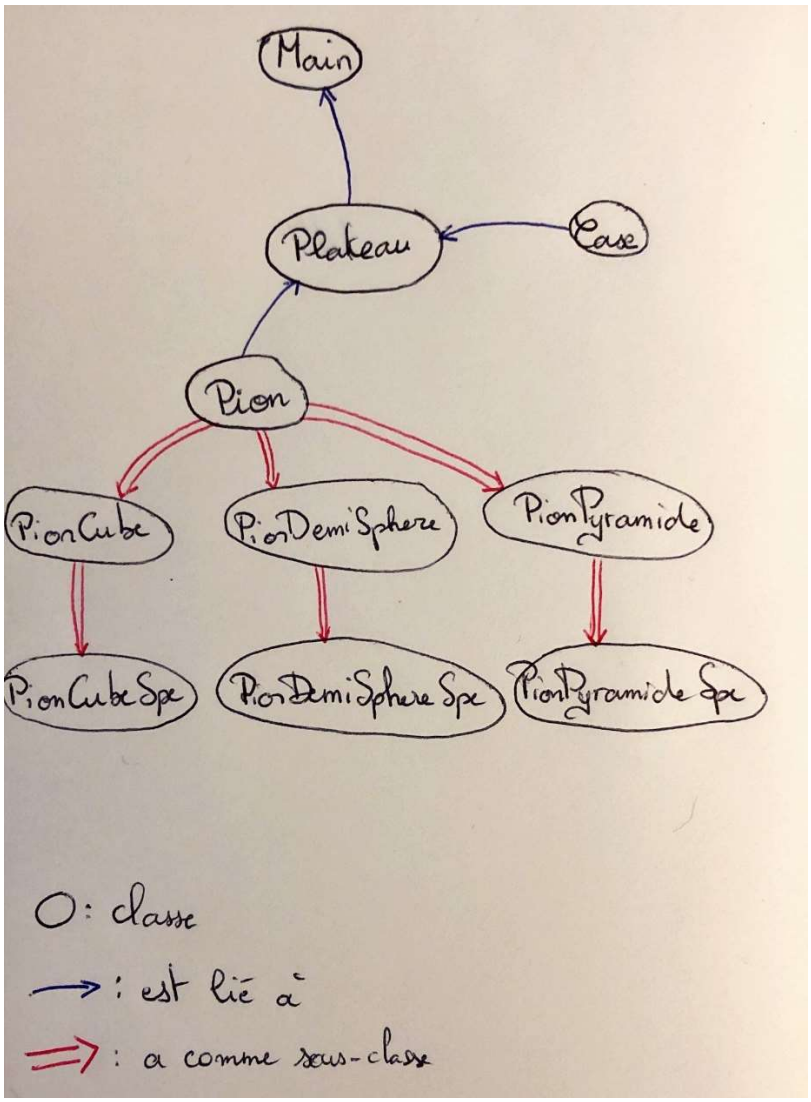
-Un plateau de 11x11 cases sur lequel se déroule le jeu consistant à y déplacer des pièces

-Des Pions, pièces du plateau divisés en différents types (pyramide, cube et demi-sphère) ayant des versions spéciales

-Des règles de déplacement et de Capture, permettant des interactions entre les pions et le plateau.

-Des conditions de fin de partie, permettant de mettre fin au jeu et de désigner un vainqueur.

-Un système de sauvegarde des données d'une partie afin de pouvoir la relancer plus tard.



1 Schéma des relations des classes du programme entre elles

Listing des unités de traitement

L'utilité de ce programme

Il y a sur la version actuelle 10 classes différentes :

- La classe AUBRY_AMON_Trinome contenant le programme principal affichant un menu donnant un choix à faire

- La classe Plateau qui permet de créer un plateau et de jouer une partie

- La classe Case. Les cases sont des entités contenues dans le plateau qui possèdent une couleur et une équipe.

- La classe Pion. Les pions sont des entités contenues dans le plateau, capables de se déplacer.

- Les classes PionPyramide, PionCube et PionDemiSphere, sous-classes de la classe Pion. Leurs entités ont des spécificités différentes les unes des autres (déplacement vertical et horizontal ou diagonal et la possibilité de capturer des pions adverses).

- Les classes PionPyramideSpe, PionCubeSpe et PionDemiSphereSpe étendant respectivement les classes PionPyramide, PionCube et PionDemiSphere. Leurs entités sont des variantes des précédentes et possèdent des spécificités supplémentaires, telles que le joker, l'incapacité de reculer ou parcourir une plus longue distance en capturant un pion.

Test unitaire des principales méthodes

Chacune de ces méthodes sont testées à partir du Main. Chaque test est effectué plusieurs fois. Si l'issue d'un de ces tests est une erreur, alors c'est ce résultat qui sera présenté. Dans le cas contraire, un résultat obtenu sera présenté. Les tests sont effectués classe par classe et non dans l'ordre chronologique des méthodes utilisées pour jouer une partie.

1. L'utilisateur entre les données du mauvais type (quelque soit la fonction)

On s'attend à une erreur.

```
Round 1. C'est au tour du Joueur 1.
Saisissez les coordonnées du pion que vous souhaitez déplacer (x puis y, 1:1 en haut à gauche):
rentrez -1 -1 pour sauvegarder
abc
Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:939)
        at java.base/java.util.Scanner.next(Scanner.java:1594)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
        at Plateau.joue(Plateau.java:140)
        at Plateau.PvPsansGraphismes(Plateau.java:120)
        at AUBRY_AMON_Trinome.JoueurVSJoueurVIAConsole(AUBRY_AMON_Trinome.java:31)
        at AUBRY_AMON_Trinome.main(AUBRY_AMON_Trinome.java:6)
```

On obtient bien l'erreur attendue. Cette erreur ne sera normalement plus présente lors du passage à une interface graphique. En effet, les scanners seront remplacés par des boutons.

2. Plateau

Initialisation d'une partie

Fonctions testées : *main, constructeur de la classe Plateau, tableauPionInit, tableauPionSpeInit*

On cherche vérifier que les parties s'initialisent correctement. Pour se faire, on sélectionne des choix d'un menu réalisé dans le Main. Ce menu permettra par la suite de mener au test de la plupart des fonctions principales ainsi que de la réalisation d'une partie complète.

```
Bienvenue. Choisissez une option
1. Jouer une partie
2. Jouer une partie spéciale
3. Charger une partie
4. Reprendre la partie en cours
5. Règles
0. Quitter
```

On s'attend à observer l'affichage d'un plateau initialisé par la fonction *affichageMatricielDesPions*. On vérifiera le contenu du tableau (emplacement des pions, équipes, ect…)

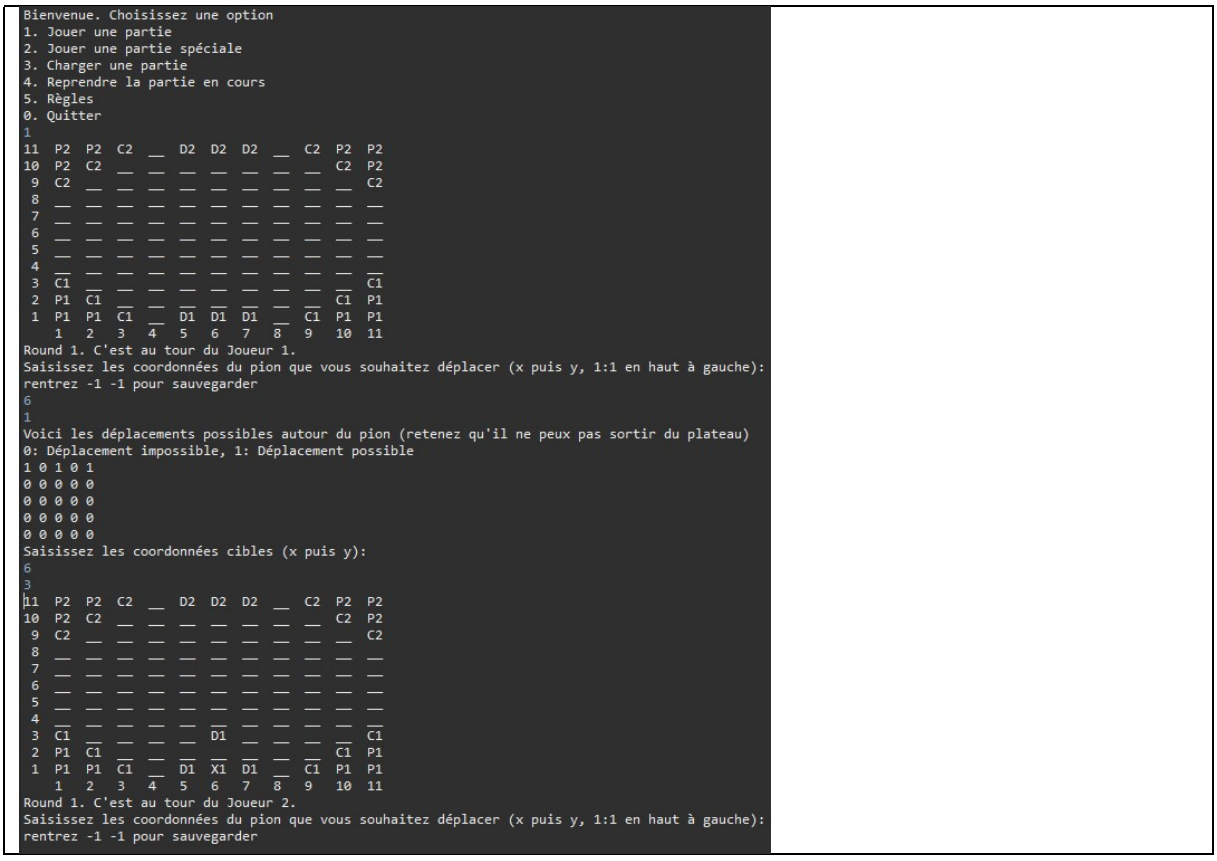
```
1
11  P2  P2  C2  __  D2  D2  D2  __  C2  P2  P2
10  P2  C2  __  __  __  __  __  __  C2  P2
 9  C2  __  __  __  __  __  __  __  __  C2
 8  __  __  __  __  __  __  __  __  __  __
 7  __  __  __  __  __  __  __  __  __  __
 6  __  __  __  __  __  __  __  __  __  __
 5  __  __  __  __  __  __  __  __  __  __
 4  __  __  __  __  __  __  __  __  __  __
 3  C1  __  __  __  __  __  __  __  __  C1
 2  P1  C1  __  __  __  __  __  __  C1  P1
 1  P1  P1  C1  __  D1  D1  D1  __  C1  P1  P1
    1   2   3   4   5   6   7   8   9   10  11
Round 1. C'est au tour du Joueur 1.
Saisissez les coordonnées du pion que vous souhaitez déplacer (x puis y, 1:1 en haut à gauche):
rentrez -1 -1 pour sauvegarder
2
11  T2  P2  C2  __  D2  S2  D2  __  C2  P2  T2
10  P2  R2  __  __  __  __  __  __  __  R2  P2
 9  C2  __  __  __  __  __  __  __  __  __  C2
 8  __  __  __  __  __  __  __  __  __  __  __
 7  __  __  __  __  __  __  __  __  __  __  __
 6  __  __  __  __  __  __  __  __  __  __  __
 5  __  __  __  __  __  __  __  __  __  __  __
 4  __  __  __  __  __  __  __  __  __  __  __
 3  C1  __  __  __  __  __  __  __  __  __  C1
 2  P1  R1  __  __  __  __  __  __  __  R1  P1
 1  T1  P1  C1  __  D1  S1  D1  __  C1  P1  T1
    1   2   3   4   5   6   7   8   9   10  11
Round 1. C'est au tour du Joueur 1.
Saisissez les coordonnées du pion que vous souhaitez déplacer (x puis y, 1:1 en haut à gauche):
rentrez -1 -1 pour sauvegarder
```

On observe qu'un plateau est bien initialisé et qu'il contient des pions. Grâce aux vérifications sur les pions, on peut conclure que les plateaux sont correctement initialisés (pions placés au bon endroit).

Déplacement d'un pion sur le tableau

Fonction testée : *deplacement*

On s'attend à que le pion change de place après exécution de la fonction et affichage de la représentation console du plateau.

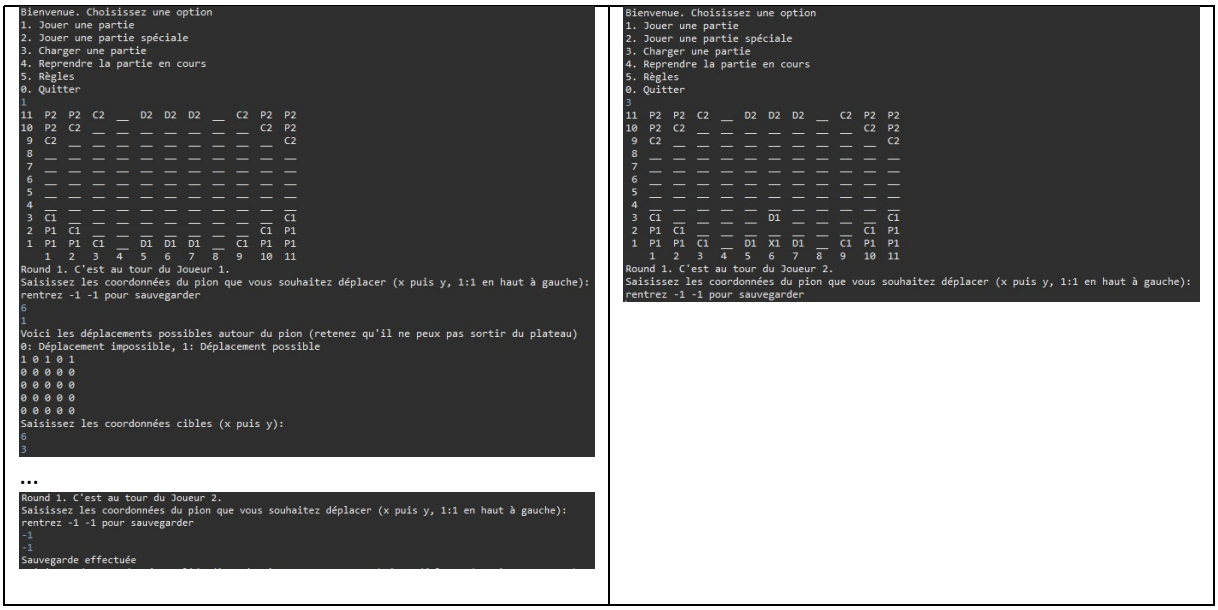


On observe bien le déplacement et la capture est bien prise en compte dans le cas d'une pyramide spéciale.

Sauvegarde et chargement

Fonctions testées : *save, load*

Nous cherchons à enregistrer une partie en cours et à la charger après une itération du programme.



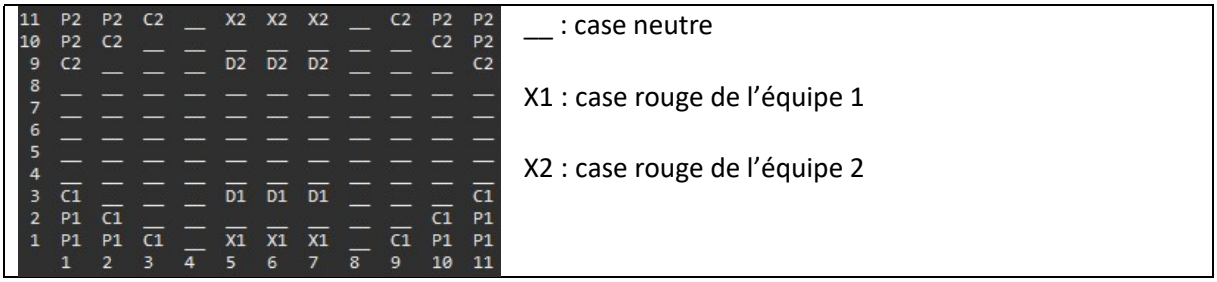
La sauvegarde et la recharge s'effectuent correctement.

3. Case

Initialisation des cases

Fonction testée : *tableauCaseInit*

On cherche à vérifier que des cases rouges sont bien présentes sur le plateau



On observe que les cases sont bien placées et qu'ils appartiennent à leurs équipes respectives

4. Pion et ses sous-classes

Création d'un pion

Fonctions testées : *constructeurs de Pion, toString (Pion)*

On cherche à vérifier que chaque pion, lorsqu'il est créé, contient les bonnes informations selon sa place. Pour ce faire, on se sert de la représentation graphique de ses pions sur le plateau.

On obtient les résultats suivants après observation du plateau :

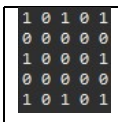
P1 : Pion Pyramide de l'équipe 1	T1 : Pion Pyramide Spécial de l'équipe 1
P2 : Pion Pyramide de l'équipe 2	T2 : Pion Pyramide Spécial de l'équipe 2
C1 : Pion Cube de l'équipe 1	R1 : Pion Cube Spécial de l'équipe 1
C2 : Pion Cube de l'équipe 2	R2 : Pion Cube Spécial de l'équipe 2
D1 : Pion Demi-Sphère de l'équipe 1	S1 : Pion Demi-Sphère Spéciale de l'équipe 1
D2 : Pion Demi-Sphère de l'équipe 2	S2 : Pion Demi-Sphère Spéciale de l'équipe 2

Les pions sont correctement créés et appartiennent tous à l'équipe qui leur a été assignée.

Zones de déplacement des Pions

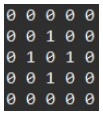
Fonction testée : *ouPeutJouer (valeur des tableaux)*

On cherche ici à vérifier les zones de déplacement des pions lorsque qu'aucun obstacle n'est présent.

<table>
<tr>
<td>


Zone obtenue pour un Pion Demi-Sphère

Zone obtenue pour un Pion Pyramide



Zone obtenue pour un Pion Cube
</td>
<td>

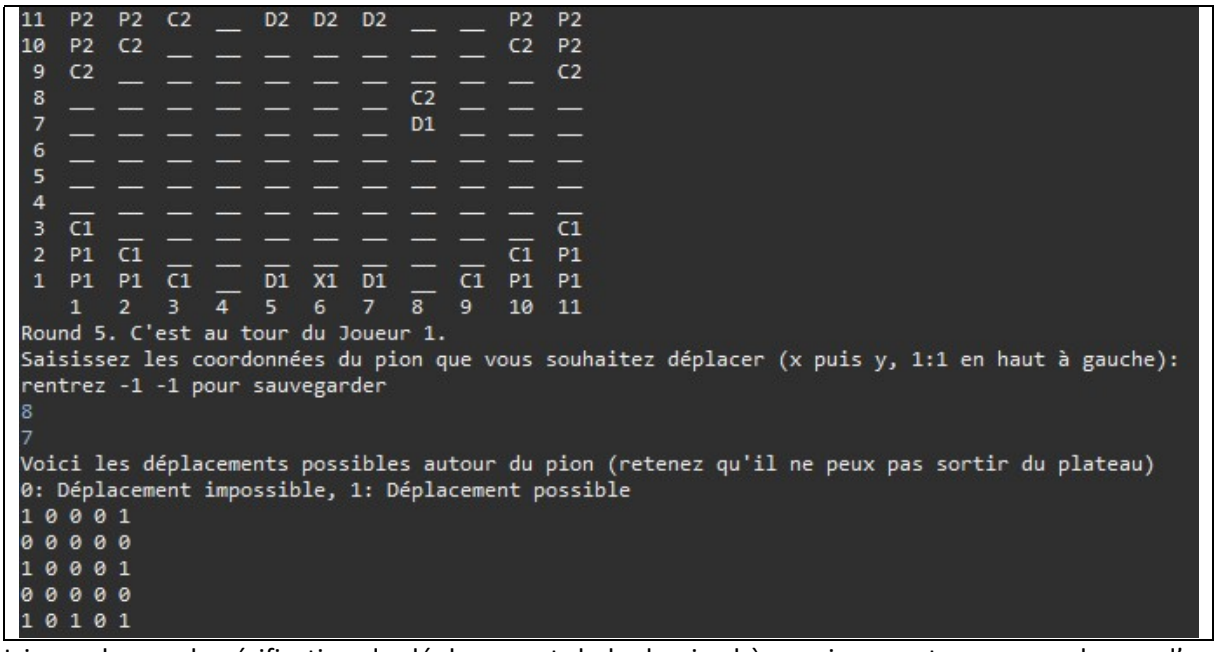

Zones obtenues pour un Pion Pyramide Spécial respectivement de l'équipe 1 et 2 dont le déplacement n'est pas bloqué
</td>
</tr>
</table>

On obtient bien un affichage des matrices intégrées dans chaque pion.

Déplacements spéciaux

Fonction testée : *ouPeutJouer (redéfinitions des sous-classes, if et boucles for)*

On s'attend à observer des contraintes et possibilités comme expliquées dans les règles du jeu.



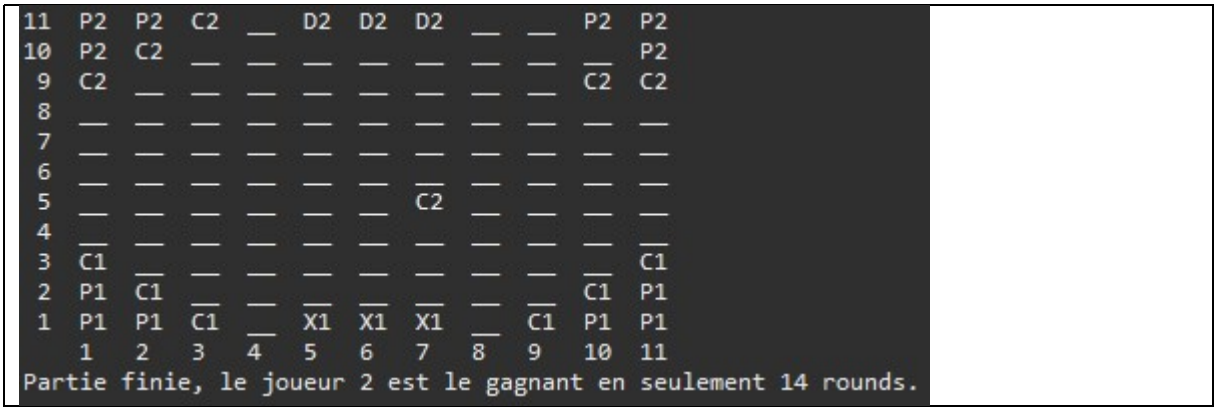
Ici, on observe la vérification du déplacement de la demi-sphère qui ne peut passer au-dessus d'un autre pion. Les déplacements autorisés affichés correspondent.

D'autres tests sur le joker du pion cube spécial, la capture spéciale de la pyramide et les conditions pour que la demi-sphère spéciale puisse reculer ont été réalisés avec des observations similaires. On en conclue que la fonction *ouPeutJouer* est correcte.

Fin de Partie

Fonction testée : *continue*

La fonction *continue* est un booléen qui permet de savoir si la partie continue ou non. On cherche à vérifier si l'état du booléen correspond à celui du plateau.



Ici le joueur 1 a perdu toutes ses demi-sphères, son opposant à gagné. Nous avons également testé la victoire du joueur 1 ainsi que la victoire par occupation des cases rouges adverses.

Conclusion

Nous avons déjà codé un programme qui permet de jouer une partie selon les règles classiques ou leur variante à travers la console. Cette version nous permet de tester les différentes fonctions qui seront utilisées par l'interface graphique.
Notre objectif actuel est de mettre en place l'interface graphique, ce qui implique de traiter la gestion des images que nous utiliserons. Le rendu final étant purement graphique et géré à l'aide de boutons, nous n'avons pas jugé nécessaire de rattraper les erreurs de type d'entrée de la console.
Parmi améliorations possibles, il y a par exemple :
- Diverses petites optimisations du programme de façon à améliorer la lisibilité de celui-ci et sa rapidité d'exécution
- La gestion des erreurs de type d'entrée afin de finaliser la jouabilité en mode console

Annexe :

Programme

Le Programme se divise en plusieurs documents qui seront affichés dans l'ordre suivant :

- AUBRY_AMON_Trinome
- Plateau
- Case
- Pion
- PionPyramide
- PionPyramideSpe
- PionCube
- PionCubeSpe
- PionDemiSphere
- PionDemiSphereSpe

```java
import java.util.Scanner;

public class AUBRY_AMON_Trinome {

    public static void main String[] args) {
        JoueurVSJoueurVIAConsole(args);
    }

    public static void JoueurVSJoueurVIAConsole String[] args) {
        //Main de vérification des fonctions
        Scanner sc = new Scanner(System.in);
        Plateau plateau;
        int choix = -1;
        while(choix != 0) {
            System.out.println("Bienvenue. Choisissez une option");
            System.out.println("1. Jouer une partie");
            System.out.println("2. Jouer une partie spéciale");
            System.out.println("3. Charger une partie");
            System.out.println("4. Reprendre la partie en cours");
            System.out.println("5. Règles");
            System.out.println("0. Quitter");

            choix = sc.nextInt();
            switch (choix) {
            case 0:
                System.out.println("Au revoir");
                break;
            case 1:
                plateau = new Plateau(false);
                plateau.PvPsansGraphismes();
                choix = 0;
                break;
            case 2:
                plateau = new Plateau(true);
                plateau.PvPsansGraphismes();
                choix = 0;
                break;
            case 3:
                plateau = new Plateau(false);
                plateau.Load("001");
                plateau.PvPsansGraphismes();
                choix = 0;
                break;
            case 4:
                plateau = new Plateau(false);
                plateau.Load("000");
                plateau.PvPsansGraphismes();
                choix = 0;
                break;
            case 5:
                System.out.println("Préparation : \r\n"
                        + "PREPARATION\r\n"
                        + "Placez le plateau de jeu au centre de la table.\r\n"
                        + "Disposez les pions comme sur la figure 1.\r\n"
                        + "Le joueur rouge commence, puis les joueurs jouent à tour de
rôle.");
                System.out.println("Tour de jeu : \r\n"
                        + "A son tour de jeu, un joueur doit déplacer l'un de ses pions.\r
\n"
                        + "- Les pyramide se déplacent en diagonal d'une case.\r\n"
                        + "- Les Cubes se déplacent verticalement ou horizontalement d'une
case.\r\n"
```

```
60                          + "- Les sphères se déplacent verticalement, horizontalement ou en
  diagonale de deux cases.\r\n"
61                          + "Capture:\r\n"
62                          + "La capture se fait par remplacement, c'est à dire par un pion se
  déplaçant dans une case occupée par un pion adverse. Attention, seul les pyramides et les
  cubes peuvent effectuer des captures. Les demi-sphères ne peuvent pas effectuer de capture,
  par contre elles peuvent être capturées. Un pion capturé est définitivement éliminé du
  plateau de jeu.");
63                  System.out.println("Fin de partie :\r\n"
64                          + "1ère possibilité: Le premier joueur qui parvient à placer 3
  différentes pièces (une demi-sphère + un cube, + une pyramide l'emporte) sur les 3 cases
  rouges de l'adversaire remporte la partie.\r\n"
65                          + "Ci-contre le joueur vert a remporter la partie.\r\n"
66                          + "2ème possibilité: Si un joueur parvient à capturer tous les pions
  d'une sorte de son adversaire (ex: les 6 cubes) il remporte la partie immédiatement.\r\n"
67                          + "\r\n"
68                          + "Variante (3ème possibilité):\r\n"
69                          + "Se sont les trois demi-sphères qui doivent occuper les cases de
  départ des trois demi-sphères adverses.");
70                  break;
71              default:
72                  System.out.println("Merci de rentrer un choix valide");
73              }
74          }
75          sc.close();
76      }
77  }
```

```java
import java.io.BufferedReader;

public class Plateau {
    private Pion[][] tableauPion; //[y][x] Matrice qui permet de situer les pions sur le plateau
    private Case[][] tableauCase; //[y][x] Matrice qui permet de situer les cases sur le Plateau
    private int round; //Compteur du nombre de tours de la partie
    private boolean j1DoitJouer; //Indique quel joueur doit jouer : true = Joueur1, false = Joueur2
    private int gagnant;//définit le gagnant: 0:Pas encore de gagnant, 1:J1 et 2:J2
    private Scanner sc;//scanner associé au plateau

    public Plateau(boolean spe) {
        //Initialisation des variables numériques
        this.round = 0;
        this.j1DoitJouer = true;
        //Initialisation des tableaux
        if(spe) {
            this.tableauPion = this.tableauPionSpeInit();
        } else {
            this.tableauPion = this.tableauPionInit();
        }
        this.tableauCase = this.tableauCaseInit();
        //Initialisation du scanner
        this.sc = new Scanner(System.in);
    }

    //Fonctions Principales
    public boolean continuer() {
        //Fonction qui permet de vérifier si la partie doit continuer
        //Retourne Vrai s'il n'y a pas encore de gagant
        //Vérification du nombre de pions restants par type et équipe
        int[][] tab = {{0, 0, 0},{0, 0, 0}};
        for(int x = 0; x <11; x++) {
            for(int y = 0; y <11; y++) {
                if(this.tableauPion[y][x] != null) {
                    if (this.tableauPion[y][x].getType()-1 < 3) {
                        tab[this.tableauPion[y][x].getEquipe()-1][this.tableauPion[y][x].getType()-1]++;
                    } else {
                        tab[this.tableauPion[y][x].getEquipe()-1][this.tableauPion[y][x].getType()-4]++;
                    }


                }
            }
        }
        for(int i = 0; i <2; i++) {
```

```
49              for(int j = 0; j <3; j++) {
50                  if(tab[i][j] == 0) {
51                      this.gagnant = 2-i;
52                      return false;
53                  }
54              }
55          }
56          //Vérification des cases rouges des deux équipes
57          int[][] tab2 = {{0, 0, 0},{0, 0, 0}};
58          //Equipe 1
59          for(int x=4; x < 7; x++) {
60              if(this.tableauPion[0][x] == null) {
61                  break;
62              } else {
63                  if(this.tableauPion[0][x].getEquipe()==1 &&
  this.tableauPion[0][x].getType()-1 < 3) {
64                      tab2[1][this.tableauPion[0][x].getType()-1]++;
65                  } else if (this.tableauPion[0][x].getEquipe()==1 &&
  this.tableauPion[0][x].getType()-1 < 3){
66                      tab2[1][this.tableauPion[0][x].getType()-4]++;
67                  }
68              }
69          }
70          //Equipe 2
71          for(int x=4; x < 7; x++) {
72              if(this.tableauPion[10][x] == null) {
73                  break;
74              } else {
75                  if(this.tableauPion[10][x].getEquipe()==1) {
76                      tab2[0][this.tableauPion[10][x].getType()-1]++;
77                  } else if (this.tableauPion[10][x].getEquipe()==1 &&
  this.tableauPion[10][x].getType()-1 < 3){
78                      tab2[0][this.tableauPion[10][x].getType()-4]++;
79                  }
80              }
81          }
82          if(tab2[1][0] == 1 && tab2[1][1] == 1 && tab2[1][2] == 1) {
83              this.gagnant = 2;
84              return false;
85          } else if(tab2[0][0] == 1 && tab2[0][1] == 1 && tab2[0][2] == 1) {
86              this.gagnant = 1;
87              return false;
88          }
89          return true;
90      }
91
92      public void Deplacement(int x1, int y1, int x2, int y2){
93          //Fonction qui permet de déplacer un Pion sur le tableau des Pions
94          if (this.tableauPion[y1][x1].getType() == 4) {
95              System.out.println("ouiiii");
```

```
 96             if (y2-y1 == 2 && x2-x1 == 2){
 97                 this.tableauPion[y1+1][x1+1] = null;
 98             } else if (y2-y1 == 2 && x2-x1 == -2){
 99                 this.tableauPion[y1+1][x1-1] = null;
100             } else if (y2-y1 == -2 && x2-x1 == 2){
101                 this.tableauPion[y1-1][x1+1] = null;
102             } else if (y2-y1 == -2 && x2-x1 == -2){
103                 this.tableauPion[y1-1][x1-1] = null;
104             }
105         }
106         this.tableauPion[y2][x2] = this.tableauPion[y1][x1];
107         this.tableauPion[y1][x1] = null;
108     }
109
110     //Procédures d'une partie sans interface graphique
111     public void PvPsansGraphismes() {
112         //Fonction qui permet de jouer ou de reprendre une partie
113         this.affichageMatricielDesPions();
114         while (this.continuer()) {
115             this.Save();
116             if (this.j1DoitJouer) {
117                 this.round++;
118                 this.joue(1);
119                 this.j1DoitJouer = false;
120             } else {
121                 this.joue(2);
122                 this.j1DoitJouer = true;
123             }
124             this.verifDesJokers();
125             this.affichageMatricielDesPions();
126         }
127         System.out.println("Partie finie, le joueur "+this.gagnant+" est le
    gagnant en seulement "+this.round+" rounds.");
128     }
129
130     public void joue(int joueur) {
131         //Fonction qui permet à un joueur de jouer
132         //1: Choix du pion à déplacer
133         System.out.println("Round "+this.round+". C'est au tour du Joueur
    "+joueur+".");
134         System.out.println("Saisissez les coordonnées du pion que vous
    souhaitez déplacer (x puis y, 1:1 en haut à gauche):");
135         System.out.println("rentrez -1 -1 pour sauvegarder");
136         int x1 = this.sc.nextInt()-1;
137         int y1 = this.sc.nextInt()-1;
138         int[][] tab;
139         while(x1 < 0 || y1 < 0 || x1 > 10 || y1 > 10 ||
    this.tableauPion[y1][x1] == null || this.tableauPion[y1][x1].getEquipe() !=
    joueur) {
140             if (x1 == -2 && y1 == -2) {
```

```
141                 this.Save("001");
142             }
143             System.out.println("Saisissez des coordonnées valide d'un pion à vous que vous souhaitez déplacer (x puis y, 1:1 en bas à gauche):");
144             System.out.println("rentrez -1 -1 pour sauvegarder");
145             x1 = this.sc.nextInt()-1; y1 = this.sc.nextInt()-1;
146         }
147         //2: Affichage des possibilités et sélection de la case cible
148         boolean next = true;
149         int x2;
150         int y2;
151         while(next) {
152             tab = this.tableauPion[y1][x1].ouPeutJouer(x1, y1);
153             System.out.println("Voici les déplacements possibles autour du pion (retenez qu'il ne peux pas sortir du plateau)");
154             System.out.println("0: Déplacement impossible, 1: Déplacement possible");
155             this.afficher(tab);
156             System.out.println("Saisissez les coordonnées cibles (x puis y):");
157             x2 = this.sc.nextInt()-1; y2 = this.sc.nextInt()-1;
158 
159             while (!(0<=x2 && x2<=11 && 0<=y2 && y2<=11)) {
160                 if (0<=y2-y1+2 && y2-y1<3 && 0<=x2-x1+2 && x2-x1<3) {
161                     if (this.valeurTab(tab, y2-y1+2, x2-x1+2) == 1 && Math.abs(x2-x1) <= 2  && Math.abs(y2-y1) <= 2) {
162                         break;
163                     }
164                 }
165                 if(x2 < 0 && y2 < 0 && x2 > 10 && y2 > 10) {
166                     if (this.tableauPion[y2][x2] != null) {
167                         if (this.tableauPion[y2][x2].getEquipe() == (this.j1DoitJouer? 1:2)) {
168                             break;
169                         }
170                     }
171                 }
172                 System.out.println("Saisissez des coordonnées cibles valides (x puis y):");
173                 x2 = this.sc.nextInt()-1; y2 = this.sc.nextInt()-1;
174             }
175 
176             if(this.valeurTab(tab, y2-y1+2, x2-x1+2) == 1 && Math.abs(x2-x1) <= 2  && Math.abs(y2-y1) <= 2) {
177                 this.Deplacement(x1, y1, x2, y2);
178                 next = false;
179             } else if(this.tableauPion[y2][x2] != null) {
180                 if(this.tableauPion[y2][x2].getEquipe() == (this.j1DoitJouer? 1:2)) {
181                     System.out.println("Changement de Pion");
```

```
182                     x1 = x2; y1 = y2;
183                 }
184             }
185         }
186     }
187
188     private int valeurTab(int[][] tab, int y, int x) {
189         //Fonction qui permet de savoir si un pion a le droit de se
    déplacer quelque part
190         if (y < 0 || x <0 || y>4 || x>4) {
191             return 0;
192         } else {
193             return tab[y][x];
194         }
195     }
196
197     public void afficher(int[][] tab) {
198         //Fonction qui affiche les déplacements possibles pour un pion
199         for(int i=0; i<tab.length; i++) {
200             for(int j=0; j<tab.length; j++) {
201                 System.out.print(tab[tab.length-1-i][j]+" ");
202             }
203             System.out.println();
204         }
205     }
206
207     private void affichageMatricielDesPions() {
208         //Fonction qui affiche une représentation du plateau
209         for(int i=0; i<11; i++) {
210             System.out.print((11-i < 10 ? " " : "")+(11-i)+" ");
211             for(int j=0; j<11; j++) {
212                 if(this.tableauPion[10-i][j] != null) {
213                     System.out.print(" "+this.tableauPion[10-i]
    [j].toString()+this.tableauPion[10-i][j].getEquipe()+" ");
214                 } else {
215                     System.out.print(" "+this.tableauCase[10-i]
    [j].toString()+" ");
216                 }
217             }
218             System.out.println();
219         }
220         System.out.println("    1   2   3   4   5   6   7   8   9   10
    11");
221     }
222
223     private void verifDesJokers() {
224         //Fonction qui vérifie l'état des jokers des pions du plateau
225         for(int x=0; x<5; x++) {
226             for(int y=0; y<5; y++) {
227                 if (this.tableauPion[y][x] != null) {
```

```
228                     if (this.tableauPion[y][x].getType() == 5) {
229                         this.tableauPion[y][x].verifJoker(x,y);
230                     }
231                 }
232             }
233         }
234     }
235
236     //Aides au constructeur
237     public Pion[][] tableauPionInit() {
238         //Fonction qui retourne un tableau des pions initialisé
239
240         Pion[][] pions = new Pion[11][11];
241         ////Placement des pions verts
242         //Première ligne
243         pions[0][0] = new PionPyramide(1, this);    pions[0][1] = new
    PionPyramide(1, this);    pions[0][2] = new PionCube(1, this);
244         pions[0][4] = new PionDemiSphere(1, this); pions[0][5] = new
    PionDemiSphere(1, this); pions[0][6] = new PionDemiSphere(1, this);
245         pions[0][8] = new PionCube(1, this);        pions[0][9] = new
    PionPyramide(1, this);    pions[0][10] = new PionPyramide(1, this);
246         //Deuxième ligne
247         pions[1][0] = new PionPyramide(1, this);    pions[1][1] = new
    PionCube(1, this);
248         pions[1][9] = new PionCube(1, this);        pions[1][10] = new
    PionPyramide(1, this);
249         //Troisième ligne
250         pions[2][0] = new PionCube(1, this);
251         pions[2][10] = new PionCube(1, this);
252
253         ////Placement des pions rouges
254         //Première ligne
255         pions[10][0] = new PionPyramide(2, this);    pions[10][1] = new
    PionPyramide(2, this);    pions[10][2] = new PionCube(2, this);
256         pions[10][4] = new PionDemiSphere(2, this); pions[10][5] = new
    PionDemiSphere(2, this); pions[10][6] = new PionDemiSphere(2, this);
257         pions[10][8] = new PionCube(2, this);        pions[10][9] = new
    PionPyramide(2, this);    pions[10][10] = new PionPyramide(2, this);
258         //Deuxième ligne
259         pions[9][0] = new PionPyramide(2, this);    pions[9][1] = new
    PionCube(2, this);
260         pions[9][9] = new PionCube(2, this);        pions[9][10] = new
    PionPyramide(2, this);
261         //Troisième ligne
262         pions[8][0] = new PionCube(2, this);
263         pions[8][10] = new PionCube(2, this);
264
265         return pions;
266     }
267
```

```
268     public Pion[][] tableauPionSpeInit() {
269         //Fonction qui retourne un tableau des pions initialisé avec des pions spéciaux
270 
271         Pion[][] pions = new Pion[11][11];
272         ////Placement des pions verts
273         //Première ligne
274         pions[0][0] = new PionPyramideSpe(1, this);    pions[0][1] = new PionPyramide(1, this);       pions[0][2] = new PionCube(1, this);
275         pions[0][4] = new PionDemiSphere(1, this);     pions[0][5] = new PionDemiSphereSpe(1, this); pions[0][6] = new PionDemiSphere(1, this);
276         pions[0][8] = new PionCube(1, this);           pions[0][9] = new PionPyramide(1, this);       pions[0][10] = new PionPyramideSpe(1, this);
277         //Deuxième ligne
278         pions[1][0] = new PionPyramide(1, this);       pions[1][1] = new PionCubeSpe(1, this);
279         pions[1][9] = new PionCubeSpe(1, this);        pions[1][10] = new PionPyramide(1, this);
280         //Troisième ligne
281         pions[2][0] = new PionCube(1, this);
282         pions[2][10] = new PionCube(1, this);
283 
284         ////Placement des pions rouges
285         //Première ligne
286         pions[10][0] = new PionPyramideSpe(2, this);   pions[10][1] = new PionPyramide(2, this);      pions[10][2] = new PionCube(2, this);
287         pions[10][4] = new PionDemiSphere(2, this);    pions[10][5] = new PionDemiSphereSpe(2, this);pions[10][6] = new PionDemiSphere(2, this);
288         pions[10][8] = new PionCube(2, this);          pions[10][9] = new PionPyramide(2, this);      pions[10][10] = new PionPyramideSpe(2, this);
289         //Deuxième ligne
290         pions[9][0] = new PionPyramide(2, this);       pions[9][1] = new PionCubeSpe(2, this);
291         pions[9][9] = new PionCubeSpe(2, this);        pions[9][10] = new PionPyramide(2, this);
292         //Troisième ligne
293         pions[8][0] = new PionCube(2, this);
294         pions[8][10] = new PionCube(2, this);
295 
296         return pions;
297     }
298 
299     public Case[][] tableauCaseInit() {
300         //Fonction qui retourne un tableau des cases initialisé
301 
302         Case[][] cases = new Case[11][11];
303         //Coloration des cases noires et blanches
304         for (int x=0; x<11; x++) {
305             //Coloration des colonnes paires
306             for (int y=0; y<11 && x % 2 == 0; y++) {
```

```
307                 if (y % 2 == 0) {
308                     cases[y][x] = new Case("black", 0);
309                 } else {
310                     cases[y][x] = new Case("white", 0);
311                 }
312 
313             //Coloration des colones impaires
314             for (int y=0; y<11 && x % 2 == 1; y++) {
315                 if (y % 2 == 0) {
316                     cases[y][x] = new Case("white", 0);
317                 } else {
318                     cases[y][x] = new Case("black", 0);
319                 }
320 
321             }
322         //Coloration des cases rouges
323         for (int y=4; y<7; y++) {
324             cases[0][y] = new Case("red", 1);
325             cases[10][y] = new Case("red", 2);
326         }
327         return cases;
328     }
329 
330     //Sauvegarde
331     public void Save() {
332         //Fonction de sauvegarde automatique
333         try{
334             FileWriter fich = new FileWriter("Save_Nb_000.txt");
335             fich.write(this.toString());
336             fich.close();
337         } catch (IOException ex) {
338             System.out.println("Sauvegarde Error");
339         }
340     }
341     public void Save(String nb) {
342         //Fonction de sauvegarde manuelle
343         try{
344             FileWriter fich = new FileWriter("Save_Nb_"+nb+".txt");
345             fich.write(this.toString());
346             fich.close();
347             System.out.println("Sauvegarde effectuée");
348         } catch (IOException ex) {
349             System.out.println("Sauvegarde Error");
350         }
351 
352     }
353     //Chargement
354     public void Load(String nb) {
355         //Fonction qui permet de charger une partie
356         try{
```

```
357             FileReader fich = new FileReader("Save_Nb_"+nb+".txt");
358             BufferedReader br = new BufferedReader(fich);
359             String str;
360             String[] a = br.readLine().split(":");
361             this.round = Integer.valueOf(a[0]);
362             this.j1DoitJouer = Boolean.valueOf(a[1]);
363 
364             String[] b;
365             this.tableauPion = new Pion[11][11];
366             for (int y = 0; y < 11; y++) {
367                 for (int x = 0; x < 11; x++) {
368                     if ((str = br.readLine()) != null) {
369                         b = str.split(":");
370                         if (!b[0].isEmpty()) {
371                             switch(Integer.valueOf(b[0])){
372                             case 0:
373                                 this.tableauPion[y][x] = new
    Pion(Integer.valueOf(b[1]), this);
374                                 break;
375                             case 1:
376                                 this.tableauPion[y][x] = new
    PionPyramide(Integer.valueOf(b[1]), this);
377                                 break;
378                             case 2:
379                                 this.tableauPion[y][x] = new
    PionCube(Integer.valueOf(b[1]), this);
380                                 break;
381                             case 3:
382                                 this.tableauPion[y][x] = new
    PionDemiSphere(Integer.valueOf(b[1]), this);
383                                 break;
384                             case 4:
385                                 this.tableauPion[y][x] = new
    PionPyramideSpe(Integer.valueOf(b[1]), this);
386                                 break;
387                             case 5:
388                                 this.tableauPion[y][x] = new
    PionCubeSpe(Integer.valueOf(b[1]), this);
389                                 break;
390                             case 6:
391                                 this.tableauPion[y][x] = new
    PionDemiSphereSpe(Integer.valueOf(b[1]), this);
392                                 break;
393                             default:
394                                 break;
395                             }
396                         }
397                     }
398                 }
399             }
```

```
400             fich.close();
401             if (this.j1DoitJouer) {
402                 this.round--;
403             }
404         } catch (IOException ex) {
405             System.out.println("Loading Error");
406         }
407     }
408 
409     //Accès aux informations
410     public Pion[][] getTableauPion() {return this.tableauPion;}
411     public Case[][] getTableauCase() {return this.tableauCase;}
412     public boolean J1DoitJouer() {return this.j1DoitJouer;}
413     @Override
414     public String toString() {
415         //Fonction qui convertit un plateau en String prêt à être
    sauvegardé
416         String txt = ""+this.round+":"+this.j1DoitJouer+"\n";
417         for (int y = 0; y < 11; y++) {
418             for (int x = 0; x < 11; x++) {
419                 if (this.tableauPion[y][x] != null) {
420                     txt += this.tableauPion[y][x].toSave();
421                 }
422                 txt += "\n";
423             }
424         }
425         return txt;
426     }
427 }
```

```java
public class Case {
    private String couleur; //Définit la couleur d'une case
    private int equipe; //L'équipe de la case (0 neutre, 1 equipe1 et 2 equipe2)

    public Case(String couleur, int equipe) {
        //Constructeur d'une case
        this.equipe = equipe;
        this.couleur = couleur;
    }

    //Accès aux informations
    public int getEquipe() {return this.equipe;}
    @Override
    public String toString() {
        if(this.couleur.equals("red")) {
            return "X"+this.equipe;
        } else {
            return "__";
        }
    }
}
```

```java

public class Pion implements Cloneable{
    //private Image image;
    protected String imageLien;//Lien vers le fichier de l'image du pion
    protected int equipe;//A qui appartient le pion
    protected int type;//Assigne un type au pion: 1 pour pyramide, 2 pour cube, 3 pour demisphere, ect...
    protected Plateau plateau;
    protected int [][] tabMove;

    public Pion(int equipe, Plateau plateau) {
        //Constructeur de la classe
        this.imageLien="image d'erreur";
        this.equipe = equipe;
        this.type=0;
        final int [][] tab = {{0,0,0,0,0},{0,0,0,0,0},{0,0,0,0,0},{0,0,0,0,0},{0,0,0,0,0}};
        this.tabMove = Pion.cloneTab(tab);
        this.plateau = plateau;
    }

    public static int [][] cloneTab(int [][] tab) {
        //Fonction qui permet de copier un tableau à deux dimensions
        int length = tab.length;
        int[][] tab2 = new int[length][tab[0].length];
        for (int i = 0; i < length; i++) {
            System.arraycopy(tab[i], 0, tab2[i], 0, tab[i].length);
        }
        return tab2;
    }

    public int[][] ouPeutJouer(int posX, int posY){
        //Fonction qui retourne un tableau représentant les déplacements possibles autour d'un pion
        //Redéfinie dans les sous-classes
        int [][] tab = Pion.cloneTab(this.tabMove);

        ////Retrait des déplacements impossibles:
        //En cas de sortie de plateau
        for(int x=0; x<5; x++) {
            for(int y=0; y<5; y++) {
                if(0 > posY+y-2 || posY+y-2 > 10 ||  0 > posX+x-2 || posX+x-2 > 10) {
                    tab[y][x] = 0;
                }
            }
        }
        //En cas de présence d'un pion allié
        for(int x=0; x<5; x++) {
            for(int y=0; y<5; y++) {
                if(0 <= posY+y-2 && posY+y-2 < 11 &&  0 <= posX+x-2 && posX+x-2 < 11) {
                    if(this.plateau.getTableauPion()[posY+y-2][posX+x-2] != null) {
                        if(this.plateau.getTableauPion()[posY+y-2][posX+x-2].getEquipe() == this.equipe) {
                            tab[y][x] = 0;
                        }
                    }
                }
            }
        }
        if(this.plateau.getTableauPion()[posY][posX].getType() != 5 && (this.plateau.getTableauCase()[posY][posX].getEquipe() == (this.plateau.J1DoitJouer()? 2 : 1))) {
            int[][] tabVide = {{0,0,0,0,0},{0,0,0,0,0},{0,0,0,0,0},{0,0,0,0,0},{0,0,0,0,0}};
```

```
58              return tabVide;
59          }
60          return tab;
61      }
62
63      //Accès aux informations
64      public int getEquipe() { return this.equipe;}
65      public int getType() { return this.type;}
66      public String toSave() {
67          //Renvoie les données de sauvegarde du pion en txt
68          String txt = "";
69          txt += this.type;
70          txt += ":";
71          txt += this.equipe;
72          return txt;
73      }
74      @Override
75      public String toString() {
76          //Permet d'afficher une représentation du Pion lors d'une partie
77          if(this.type == 1) {
78              return "P";
79          } else if(this.type == 2) {
80              return "C";
81          } else if(this.type == 3){
82              return "D";
83          } else if(this.type == 4) {
84              return "T";
85          } else if(this.type == 5) {
86              return "R";
87          } else {
88              return "S";
89          }
90      }
91
92      //Overrides
93      public void verifJoker(int x, int y) {}
94  }
```

```java
public class PionPyramide extends Pion{

    public PionPyramide int equipe, Plateau plateau) {
        //Constructeur de la classe
        super(equipe, plateau);
        this.imageLien="lienPyramide";
        this.type = 1;
        final int [][] tab = {{0,0,0,0,0},{0,1,0,1,0},{0,0,0,0,0}, {0,1,0,1,0},{0,0,0,0,0}};
        this.tabMove = Pion.cloneTab(tab);
    }

    @Override
    public int[][] ouPeutJouer(int posX, int posY){
        //Fonction qui retourne un tableau représentant les déplacements possibles autour d'un pion
        //Présente ici en cas de modification future
        int [][] tab = super.ouPeutJouer(posX, posY).clone();

        return tab;
    }
}

```

```java
public class PionPyramideSpe extends PionPyramide{

    public PionPyramideSpe int equipe, Plateau plateau) {
        //Constructeur de la classe
        super(equipe, plateau);
        this.imageLien="lienPyramideSpe";
        final int [][] tab = {{0,0,0,0,0},{0,1,0,1,0},{0,0,0,0,0},{0,1,0,1,0},{0,0,0,0,0}};
        this.tabMove = Pion.cloneTab(tab);
        this.type = 4;
    }

    @Override
    public int[][] ouPeutJouer(int posX, int posY){
        //Fonction qui retourne un tableau représentant les déplacements possibles autour d'un pion
        int [][] tab = super.ouPeutJouer(posX, posY).clone();

        //Vérification des possibilités de sauter par dessus un Pion
        if(0<=posX-1 && posX-1<12 && 0<=posY-1 && posY-1<12) {
            if(this.plateau.getTableauPion()[posY-1][posX-1] != null) {
                if(this.plateau.getTableauPion()[posY-1][posX-1].getEquipe() == (this.plateau.J1DoitJouer()? 2 : 1)) {
                    tab[1][1] = 1;
                }
            }
        }
        if(0<=posX+1 && posX+1<12 && 0<=posY-1 && posY-1<12) {
            if(this.plateau.getTableauPion()[posY-1][posX+1] != null) {
                if(this.plateau.getTableauPion()[posY-1][posX+1].getEquipe() == (this.plateau.J1DoitJouer()? 2 : 1)) {
                    tab[1][3] = 1;
                }
            }
        }
        if(0<=posX-1 && posX-1<12 && 0<=posY+1 && posY+1<12) {
            if(this.plateau.getTableauPion()[posY+1][posX-1] != null) {
                if(this.plateau.getTableauPion()[posY+1][posX-1].getEquipe() == (this.plateau.J1DoitJouer()? 2 : 1)) {
                    tab[3][1] = 1;
                }
            }
        }
        if(0<=posX+1 && posX+1<12 && 0<=posY+1 && posY+1<12) {
            if(this.plateau.getTableauPion()[posY+1][posX+1] != null) {
                if(this.plateau.getTableauPion()[posY+1][posX+1].getEquipe() == (this.plateau.J1DoitJouer()? 2 : 1)) {
                    tab[3][3] = 1;
                }
            }
        }
        return tab;
    }
}
```

```java

public class PionCube extends Pion{

    public PionCube(int equipe, Plateau plateau) {
        //Constructeur de la classe
        super(equipe, plateau);
        this.imageLien="lienCube";
        final int [][] tab = {{0,0,0,0,0},{0,0,1,0,0},{0,1,0,1,0}, {0,0,1,0,0},{0,0,0,0,0}};
        this.tabMove = Pion.cloneTab(tab);
        this.type = 2;

    }

    @Override
    public int[][] ouPeutJouer(int posX, int posY){
        //Fonction qui retourne un tableau représentant les déplacements possibles autour d'un pion
        //Présente ici en cas de modification future
        int [][] tab = super.ouPeutJouer(posX, posY).clone();

        return tab;
    }
}

```

```java

public class PionCubeSpe extends PionCube{
    private boolean sousJoker;

    public PionCubeSpe(int equipe, Plateau plateau) {
        //Constructeur de la classe
        super(equipe, plateau);
        this.imageLien="lienCubeSpe";
        final int [][] tab = {{0,0,0,0,0},{0,0,1,0,0},{0,1,8,1,0},{0,0,1,0,0},{0,0,0,0,0}};
        this.tabMove = Pion.cloneTab(tab);
        this.sousJoker = true;
        this.type = 5;
    }

    public void verifJoker(int x, int y) {
        //Fonction qui permet de vérifier l'état du Joker d'un PionCube Spécial et de le modifier si besoin
        if (this.sousJoker) {
            if (!this.checkConditionsJoker(x, y)) {
                this.sousJoker = !this.sousJoker;
            }
        } else {
            if (!((x==0 && 3<y && y<7 && this.equipe == 1) || (x==10 && 3<y && y<7 && this.equipe == 2))) {
                if (this.checkConditionsJoker(x, y)) {
                    this.sousJoker = true;
                }
            }
        }
    }

    private boolean checkConditionsJoker(int x, int y) {
        //Vérification des quatres diagonales
        int compt1 = 0;
        int compt2 = 0;
        for (int i=0; x+i<11 && y+i <11; i++) {
            if(this.plateau.getTableauPion()[y+i][x+i] != null) {
                if(this.plateau.getTableauPion()[y+i][x+i].getEquipe() == this.equipe) {
                    compt1++;
                }
            }
        }
        for (int i=0; x+i<11 && 0 <= y-i; i++) {
            if(this.plateau.getTableauPion()[y+i][x+i] != null) {
                if(this.plateau.getTableauPion()[y-i][x+i].getEquipe() == this.equipe) {
                    compt2++;
                }
            }
        }
        for (int i=0; 0 <= x-i && y+i <11; i++) {
            if(this.plateau.getTableauPion()[y+i][x+i] != null) {
                if(this.plateau.getTableauPion()[y+i][x-i].getEquipe() == this.equipe) {
                    compt2++;
                }
            }
        }
        for (int i=0; 0 <= x-i && 0 <= y-i; i++) {
            if(this.plateau.getTableauPion()[y+i][x+i] != null) {
                if(this.plateau.getTableauPion()[y-i][x-i].getEquipe() == this.equipe) {
                    compt1++;
                }
            }
        }
```

```
61          }
62          if(compt1 > 2 || compt2 > 2) {
63              return true;
64          } else {
65              return false;
66          }
67      }
68
69      public int[][] ouPeutJouer(int posX, int posY){
70          int [][] tab = super.ouPeutJouer(posX, posY).clone();
71
72          if(!this.sousJoker && (this.plateau.getTableauCase()[posY][posX].getEquipe() ==
    (this.plateau.J1DoitJouer()? 2 : 1))) {
73              int[][] tabVide = {{0,0,0,0,0},{0,0,0,0,0},{0,0,0,0,0},{0,0,0,0,0},{0,0,0,0,0}};
74              return tabVide;
75          }
76          return tab;
77      }
78  }
```

```java

public class PionDemiSphere extends Pion

    public PionDemiSphere int equipe, Plateau plateau) {
        //Constructeur de la classe
        super(equipe, plateau);
        this.imageLien="lienDemiSphere";
        final int [][] tab = {{1,0,1,0,1} ,{0,0,0,0,0},{1,0,0,0,1}, {0,0,0,0,0} ,{1,0,1,0,1}};
        this.tabMove = Pion.cloneTab(tab);
        this.type = 3;
    }

    public int[][] ouPeutJouer(int posX, int posY){
        //Fonction qui retourne un tableau représentant les déplacements possibles autour d'un pion
        int [][] tab = super.ouPeutJouer(posX, posY).clone();

        //Ajout de l'impossibilité de capturer un Pion (la demiSphere ne peut pas capturer)
        for(int x=0; x<5; x++) {
            for(int y=0; y<5; y++) {
                if(0 <= posY+y-2 && posY+y-2 < 11 &&  0 <= posX+x-2 && posX+x-2 < 11) {
                    if(this.plateau.getTableauPion()[posY+y-2][posX+x-2] != null) {
                        if(this.plateau.getTableauPion()[posY+y-2][posX+x-2].getEquipe() != this.equipe) {
                            tab[y][x] = 0;
                        }
                    }
                }
            }
        }
        //Ajout de l'impossibilité de sauter par dessus un Pion
        //Haut
        if(0 <= posY+1 && posY+1 < 11) {
            if(this.plateau.getTableauPion()[posY+1][posX] != null) {
                tab[4][2] = 0;
            }
        }
        //Droit
        if(0 <= posX-1 && posX-1 < 11) {
            if(this.plateau.getTableauPion()[posY][posX-1] != null) {
                tab[2][4] = 0;
            }
        }
        //Gauche
        if(0 <= posX+1 && posX+1 < 11) {
            if(this.plateau.getTableauPion()[posY][posX+1] != null) {
                tab[2][0] = 0;
            }
        }
        //Bas
        if(0 <= posY-1 && posY-1 < 11) {
            if(this.plateau.getTableauPion()[posY-1][posX] != null) {
                tab[0][2] = 0;
            }
        }
        //Haut-Droit
        if(0 <= posY+1 && posY+1 < 11 &&  0 <= posX-1 && posX-1 < 11) {
            if(this.plateau.getTableauPion()[posY+1][posX+1] != null) {
                tab[4][4] = 0;
            }
        }
        //Haut-Gauche
```

```
61          if( 0 <= posY+1 && posY+1 < 11 &&  0 <= posX-1 && posX-1 < 11) {
62              if(this.plateau.getTableauPion()[posY+1][posX-1] != null) {
63                  tab[4][0] = 0;
64              }
65          }
66          //Bas-Droite
67          if( 0 <= posY-1 && posY-1 < 11 &&  0 <= posX+1 && posX+1 < 11) {
68              if(this.plateau.getTableauPion()[posY-1][posX+1] != null) {
69                  tab[0][4] = 0;
70              }
71          }
72          //Bas-Gauche
73          if( 0 <= posY-1 && posY-1 < 11 &&  0 <= posX-1 && posX-1 < 11) {
74              if(this.plateau.getTableauPion()[posY-1][posX-1] != null) {
75                  tab[0][0] = 0;
76              }
77          }
78          return tab;
79      }
80  }
```

```java
public class PionDemiSphereSpe extends PionDemiSphere{

    public PionDemiSphereSpe int equipe, Plateau plateau) {
        //Constructeur de la classe
        super(equipe, plateau);
        this.imageLien="lienDemiSphereSPE";
        final int [][] tab = {{1,0,1,0,1},{0,0,0,0,0},{1,0,0,0,1},{0,0,0,0,0},{1,0,1,0,1}};
        this.tabMove = Pion.cloneTab(tab);
        this.type = 6;
    }

    public int[][] ouPeutJouer(int posX, int posY){
        //Fonction qui retourne un tableau représentant les déplacements possibles autour d'un pion
        int [][] tab = super.ouPeutJouer(posX, posY).clone();
        //Vérification des possibilités de déplacement
        boolean pasPossible = true;

        if(this.equipe == 1) {
            for(int x=0; x<5 && pasPossible; x++) {
                for(int y=2; y<5 && pasPossible; y++) {
                    if(tab[y][x] == 1) {
                        pasPossible = false;
                    }
                }
            }
            //Retrait du recul s'il est possible d'avancer
            if(!pasPossible) {
                for(int x=0; x<5; x++) {
                    for(int y=0; y<2; y++) {
                        tab[y][x] = 0;
                    }
                }
            }
        } else {
            for(int x=0; x<5 && pasPossible; x++) {
                for(int y=0; y<3 && pasPossible; y++) {
                    if(tab[y][x] == 1) {
                        pasPossible = false;
                    }
                }
            }
            //Retrait du recul s'il est possible d'avancer
            if(!pasPossible) {
                for(int x=0; x<5; x++) {
                    for(int y=3; y<5; y++) {
                        tab[y][x] = 0;
                    }
                }
            }
        }
        return tab;
    }
}
```