Projet d'algorithmique: Awale

Jeremy Wagemans Sebastien Lebon

16 décembre 2014

Table des matières

Chapitre 1

Introduction

Afin d'appliquer les méhodologies et les notions enseignées en cours d'Algorithmique, nous avions pour objectif d'implémenter, en java et par groupe de deux, une version jouable du jeu de société africain, Awalé.

En effet, celui-ci fait partie de la famille des "Mancala", catégorie regroupant les jeux de type "compter et capturer". Il se compose d'un plateau de douze trous répartis en deux rangées égales, contenant chacune quatre graines. Le but est de capturer un maximum de graines au cours d'une partie.
Il existe une multitude de variantes du jeu, dépendantes de leur région d'origine. C'est la raison pour laquelle les règles du jeu nous ont été imposées.

Au terme du projet, nous avons du délivrer une version java du jeu Awalé en parfaite adéquation avec un cahier des charges et répondant à des critères de qualité stricts. Ce rapport permet donc d'exposer de manière précise son fonctionnement ainsi que les différentes étapes de son développement et est structuré comme suit.
Dans un premier temps, le manuel d'utilisateur du jeu sera présenté. Les étapes du fonctionnement du jeu et ses diverses fonctionnalités y seront rigoureusement détaillées.
Ensuite, chacune des méthodes des classes Etat et PartieAwale, nous ayant permis de porter le jeu en Java, seront développées donnant un aperçu global de l'implémentation.
Enfin, nous exposerons les divers tests qui nous ont permis d'atteindre le niveau de fiabilité et de qualité nécessaire à la soumission du projet.

Chapitre 2

Manuel utilisateur du jeu

2.1 Début d'une partie

Au lancement de *PartieAwale.java*, deux modes de jeu s'offrent à vous :

- Le premier permet de jouer seul contre l'ordinateur.
- Le second permet de faire une partie contre un autre joueur.

```
************************************************************

Bienvenue dans le jeu Awalé

************************************************************

1. Faire une partie solo
2. Faire une partie multijoueurs

* Veuillez choisir votre mode de jeu:
```

Pour introduire votre choix, entrez le numéro associé au mode de jeu souhaité puis tapez sur la touche *[ENTRER]*.
Si vous décidez de démarrer une partie en mode multijoueur, vous devrez entrer le nom des deux joueurs avant de commencer le jeu.

2.2 Interface du plateau de jeu

```
************** Manche 1 - A Juliette de jouer **************
```

Le numéro de la manche ainsi que le nom du joueur sont indiqués à chaque tour.

```
Votre adversaire => |4| |4| |4| |4| |4| |4|      Captures: 0
                    -----------------------
            Vous => |4| |4| |4| |4| |4| |4|      Captures: 0
```

Le plateau est toujours représenté de façon à ce que le joueur de la manche ait sa propre rangée de trou en face de lui. Un chiffre entouré par deux barres verticales représente un trou et le nombre de graines qu'il contient (durant la première manche, les trous de chaque joueur contiennent quatre graines chacun). Le score des joueurs est affiché à la droite de leur rangée respective.
Rappel : le jeu tourne toujours dans le sens des aiguilles d'une montre.

```
Vos jokers encore disponibles:  - Le classement des trous les plus efficaces
                                - Le nombre de captures pour chaque trou
```

Ce tableau vous indique le nombre de jokers qu'il vous reste à disposition (pour plus d'informations, veuillez vous référer à la section *2.4 Utiliser un joker*).

```
#### Menu de jeu:

1. Jouer un coup
2. Afficher l'historique de la partie et jouer un coup
3. Utiliser un joker et jouer un coup

* Que voulez-vous vous faire (1-3):
```

Pour utiliser le menu, veuillez introduire un nombre compris entre 1 et 3 en fonction de l'action désirée et appuyez sur la touche *[ENTRER]*.

Actions possibles :

- Choix 1 - Jouer un coup (Voir section *2.3 Jouer un coup*).
- Choix 2 - Afficher l'historique de la partie et jouer un coup (cette action permet d'afficher un historique complet de chaque manche depuis le début de la partie).
- Choix 3 - Utiliser un joker et jouer un coup (Voir section *2.4 Utiliser un joker*).

2.3 Jouer un coup

```
#### Jouer un coup:

Votre adversaire => |4|  |4|  |4|  |4|  |4|  |4|      Captures: 0
                    ---------------------------
            Vous => |4|  |4|  |4|  |4|  |4|  |4|      Captures: 0
                     ^    ^    ^    ^    ^    ^
  Numéro du trou =>  1    2    3    4    5    6

* Quel trou choisissez-vous? (1-6)
```

La rangée de votre adversaire est toujours affichée au dessus de la votre. La ligne *"Numéro du trou"*, quant à elle, vous indique quelle touche appuyer pour jouer le trou pointé.

Pour jouer un trou, veuillez introduire le numéro du trou (compris entre 1 et 6) et appuyez sur la touche *[ENTRER]*.

Après avoir sélectionné le trou et avoir joué le coup, le tour se finit et vous passez la main à votre adversaire.

2.4 Utiliser un joker

Chaque joueur bénéficie de deux jokers à usage unique.

```
#### Utiliser un joker:

1. Jouer un coup sans utiliser de joker
2. Montrez-moi le classement des trous les plus efficaces
3. Montrez-moi le nombre de captures pour chaque trou

* Quel joker choisissez-vous?
```

Le jeu vous propose les jokers suivants :

- Afficher le nombre de graines capturées pour chaque trou.
- Afficher le classement des trous les plus efficaces.

Une fois le joker utilisé, le jeu vous demandera de jouer un coup. Veuillez introduire le numéro du trou (compris entre 1 et 6) que vous souhaité jouer et appuyez sur la touche *[ENTRER]*.

2.4.1 Afficher le nombre de graines capturées pour chaque trou

```
#### Prédiction du nombre de captures pour chaque trou:

     Votre adversaire => |2| |2| |2| |2| |2| |2|     Captures: 0
                         -------------------------
                 Vous => |1| |3| |5| |7| |9| |11|    Captures: 0
                          ^   ^   ^   ^   ^   ^
   Nombres de captures => 3   6   9   12  15  18
```

Ce joker vous permet de prédire le nombre de graines capturées pour chaque trou joué. Les chiffres situés sur la ligne *"Nombres de captures"* vous indiquent le nombre de graines capturées après avoir joué le trou pointé.

2.4.2 Afficher le classement des trous les plus efficaces

```
#### Classement des trous les plus efficaces :


Votre adversaire => |2| |2| |2| |2| |2| |2|     Captures: 0
                    -----------------------
            Vous => |1| |3| |5| |7| |9| |11|    Captures: 0
                     ^   ^   ^   ^   ^   ^
     Classement =>  6   5   4   3   2   1
```

Ce joker vous permet de déterminer quel trou vous permet de capturer le plus de graines. Les chiffres situés sur la ligne *"Classement"* vous indiquent l'efficacité de chaque trou. Par conséquent, le trou indiqué par le chiffre 1 vous permet de capturer le plus de graines une fois joué et le trou indiqué par le chiffre 6 celui qui vous permet de capturer le moins de graines.

2.5 Fin d'une partie

```
************************************************************

Fin du jeu: Juliette a gagné!

************************************************************
```

Ce message indique la fin du jeu et vous affiche le nom du gagnant.

Chapitre 3

La classe Etat

3.1 Classe de base et attributs

```
public class Etat {
    int n;
    int[] plateau;
    int[] captures;

    final static int NBR_TROUS = 12;
    final static int NBR_ADVERSAIRES = 2;

    ...

}
```

3.2 Constructeurs

```
// Constructeur de base
public Etat() {
    this.n = 0;

    this.plateau = new int[NBR_TROUS];
    for( int i = 0; i < this.plateau.length; i++ )
        this.plateau[i] = 4;

    this.captures = new int[NBR_ADVERSAIRES];
    for( int i = 0; i < this.captures.length; i++ )
        this.captures[i] = 0;
}
```

```java
// Constructeur pour tours n > 0
public Etat(Etat etat) {
    if( etat == null ) throw new IllegalArgumentException("Etat invalide");

    this.n = etat.n;

    this.plateau = new int[NBR_TROUS];
    for( int i = 0; i < plateau.length; i++ )
        this.plateau[i] = etat.plateau[i];

    this.captures = new int[NBR_ADVERSAIRES];
    for( int i = 0; i < captures.length; i++ )
        this.captures[i] = etat.captures[i];
}
```

```java
// Constructeur pour les tests
public Etat(int n, int[] plateau, int[] captures) {
    if( n < 0 ) throw new IllegalArgumentException("Nombre de tours incorrect");
    else if( plateau == null || plateau.length > NBR_TROUS ) throw new
        IllegalArgumentException("Plateau invalide ou taille du tableau > 12");
    else if( captures == null || captures.length > NBR_ADVERSAIRES ) throw new
        IllegalArgumentException("Captures invalide ou taille du tableau > 2");

    this.n = n;
    this.plateau = plateau;
    this.captures = captures;

}
```

3.3 Méthode toString()

```java
public String toString(){

    String plateauNord = ""; // Chaine qui représente le contenu du plateau côté
        nord
    String plateauSud = ""; // Chaine qui représente le contenu du plateau côté
        sud

    for( int i=0; i < NBR_TROUS; i++ ) {
        if( i < 6 )
            plateauNord += plateau[i] + ", ";
        else
            plateauSud = plateau[i] + ", " + plateauSud;
    }

    String repTextuelle = captures[0] + "\n"; // Représentation textuelle de l'
        Etat

    repTextuelle += plateauNord + "\n";

    repTextuelle += plateauSud + "\n";

    repTextuelle += captures[1];

    return repTextuelle;

}
```

3.4 Méthode egrene(int t)

```
/**
* Méthode permettant de récupérer les graines du trou t et d'égrener chaque trou
    (suivant t) d'une graine jusqu'á ce qu'il n'y en ait plus.
* Cette méthode modifie les attributs de l'objet courant.
* @param t position du trou duquel on prend les graines á égréner.
* @return int la position du dernier trou égréné
*/
public int egrene(int t) {

    if( t < 0 || t > NBR_TROUS-1 ) throw new IllegalArgumentException("Numero du
        trou invalide");

    // On récupère les graines contenues dans le trou t
    int nbGraines = plateau[t];
    plateau[t] = 0;

    // Si t vaut NBR_TROUS - 1, position vaut 0 car on passe chez l'adversaire et
        on boucle le tour
    int position = (t == NBR_TROUS-1) ? 0 : t+1;

    while( nbGraines > 0 ) {

        // Si le nombre de graines est supérieur ou égal au nombre de trou, on
            passe le trou duquel
        // on a récupéré les graines
        if( position != t ) {
            plateau[position] += 1;
            nbGraines--;
        }

        position++;

        // Une fois qu'on arrive á la fin du plateau, on retourne á la case 0
        if( position == NBR_TROUS )
            position = 0;

    }

    return (position == 0) ? NBR_TROUS-1 : position-1;

}
```

3.5 Méthode estNourri()

```
/**
* Méthode permettant de savoir si l'adversaire est nourri. Il est nourri si un de
     ses trous contient au moins une graine.
* Cette méthode ne modifie pas les attributs de l'objet courant.
* @return boolean retourne true si l'adversaire est nourri et false s'il ne l'est
     pas.
*/
public boolean estNourri() {

    int position = 0;
    if( n%2 == 0 )
        position = NBR_TROUS/2;
    int borneSup = position + (NBR_TROUS/2);

    int nbGraines = 0;

    // Teste pour chaque trou s'il y a une graine. S'arrête dès qu'on en trouve
        une.
    while( position < borneSup && nbGraines == 0 ) {

        if( plateau[position] > 0 )
            nbGraines = plateau[position];

        position++;

    }

    return nbGraines > 0;
}
```

3.6 Méthode capture(int t)

```
/**
* Méthode permettant de capturer les graines de l'adversaire (si un trou contient
    2 ou 3 graines) après égrénage
* Cette méthode modifie les attributs de l'objet courant.
* @param t numéro du trou á partir duquel on fait nos captures
* @return int retourne le nombre de graines capturées
*/
public int capture(int t) {

    if( t < 0 || t > NBR_TROUS-1 ) throw new IllegalArgumentException("Numéro du
        trou invalide");

    int nbGraines = 0;

    int position = t;

    int borneMinimale = 0;
    if( n%2 == 0 )
        borneMinimale = NBR_TROUS/2;

    int borneMaximale = borneMinimale + (NBR_TROUS/2);

    // Si position est chez l'adversaire (< borneMaximale && >= borneMinimale) et
        que position contient 2 ou 3 graines
    // On capture
    while( ((position >= borneMinimale) && (position < borneMaximale)) &&
    (plateau[position] == 2 || plateau[position] == 3) ) {

        nbGraines += plateau[position];
        plateau[position] = 0;
        position--;

    }

    return nbGraines;
}
```

3.7 Méthode estAffame(int t)

```
/**
* Méthode permettant de savoir si l'adversaire est affamé après égrénage.
* Cette méthode ne modifie pas les attributs de l'objet courant.
* @param t numéro duquel on égrène
* @return boolean retourne true si l'adversaire est affamé.
*/
public boolean estAffame(int t) {

    if( t < 0 || t > NBR_TROUS-1 ) throw new IllegalArgumentException("Numéro du
        trou invalide");

    Etat tmp_etat = new Etat(this);

    int positionDerniereGraine = tmp_etat.egrene(t);
    tmp_etat.capture(positionDerniereGraine);

    return !(tmp_etat.estNourri());

}
```

3.8 Méthode estLegal(int t)

```
/**
* Méthode permettant de savoir si le trou t peut être joué dans l'état courant.
* Cette méthode ne modifie pas les attributs de l'objet courant.
* @param t numéro du trou á tester.
* @return boolean retourne true si.
*/
public boolean estLegal(int t) {

    if( t < 0 || t > NBR_TROUS-1 ) throw new IllegalArgumentException("Numéro du
        trou invalide");

    // Teste si t ne se trouve pas du côté du joueur
    if( ((n%2 == 0) && (t >= NBR_TROUS/2 && t <= NBR_TROUS-1)) || ((n%2 != 0) &&
        (t >= 0 && t <= (NBR_TROUS/2)-1)) )
        return false;

    // Teste s'il n'y a pas de graine dans le trou
    if( plateau[t] == 0 )
        return false;

    // Teste si l'adversaire est nourri après égrénage
    Etat tmp_etat = new Etat(this);
    int positionDerniereGraine = tmp_etat.egrene(t);
    return tmp_etat.estNourri();

}
```

3.9 Méthode joue(int t)

```
/**
* Méthode permettant de jouer un coup depuis le trou t.
* Cette méthode ne modifie pas les attributs de l'objet courant.
* @param t numéro du trou duquel le joueur joue son coup.
* @return instance de la classe Etat représentant l'Etat après le coup.
*/
public Etat joue(int t) {

    if( !estLegal(t) ) throw new IllegalArgumentException("Coup invalide");

    Etat nouvelEtat = new Etat(this);
    int positionDerniereGraine = nouvelEtat.egrene(t); // On égrène ápd de t

    if( !nouvelEtat.estAffame(t) )
        nouvelEtat.captures[n%2] += nouvelEtat.capture(positionDerniereGraine);
            // On capture les graines de l'adversaire

    nouvelEtat.n += 1; // On passe au tour suivant

    return nouvelEtat;

}
```

3.10 Méthode estTerminee()

```
/**
* Méthode permettant de savoir si la partie est terminée ou pas.
* Cette méthode ne modifie pas les attributs de l'objet courant.
* @return boolean retourne vrai si la partie est terminée.
*/
public boolean estTerminee() {

    // Si le nombre de captures >= 25
    if( captures[0] >= 25 || captures[1] >= 25 )
        return true;

    // Si l'adversaire est nourri
    if( estNourri() )
        return false;

    int i = (n%2)*(NBR_TROUS/2);
    int borneSup = (n%2 == 0) ? NBR_TROUS/2 : NBR_TROUS;
    int grainesNecessaires = ((NBR_TROUS/2)-1);

    // Vérifie s'il n'y a plus moyen de nourrir l'adversaire
    boolean estTerminee = true;
    while( i < borneSup && estTerminee ) {

        if(plateau[i] > grainesNecessaires)
            estTerminee = false;
        i++;
        grainesNecessaires--;

    }

    return estTerminee;

}
```

3.11 Méthode pieceCapturees()

```
/**
* Méthode permettant d'obtenir un tableau de telle sorte que pour chaque indice i
     du tableau,
* t[i] représente le nombre de graines capturées lorsque le trou i est joué.
* Si le trou i ne peut être joué, t[i] sera -1.
* Cette méthode ne modifie pas les attributs de l'objet courant.
* @return int[] le tableaut représentant le nombre de graines capturées pour
    chaque trou.
*/
public int[] piecesCapturees() {

    int[] tableauPiecesCapturees = new int[NBR_TROUS];

    for( int i = 0; i < tableauPiecesCapturees.length; i++ ){
        if( estLegal(i) ) {
            tableauPiecesCapturees[i] = joue(i).captures[n%2] - this.captures[n
                %2]; // On soustrait les captures de l'etat courant avec celles du
                 nouvel état afin d'enregistrer les pieces capturees
        } else {
            tableauPiecesCapturees[i] = -1;
        }
    }

    return tableauPiecesCapturees;

}
```

3.12 Méthode classementDesTrous()

```
/**
* Méthode permettant d'obtenir un tableau contenant les indices i tel que si i
    est joué, il y aura plus de captures que si i+1 était joué.
* Cette méthode ne modifie pas les attributs de l'objet courant.
* @return int[] le tableau représentant le classement des captures.
*/
public int[] classementDesTrous() {

    int[] tableauPiecesCapturees = piecesCapturees();
    int[] tableauClassementDesTrous = new int[NBR_TROUS]; // Contiendra le
        classement des indices
    int[] tmp_tableau = new int[NBR_TROUS]; // Contiendra une copie de
        tableauPiecesCapturees sans les trous injouables

    // Recopie tous les trous jouables dans tmp_tableau et leur indice dans
        tableauClassementDesTrous
    int nbTrousJouables = 0;
    for( int i = 0; i < tableauPiecesCapturees.length; i++ ) {

        if( tableauPiecesCapturees[i] > -1 ) {
            tmp_tableau[nbTrousJouables] = tableauPiecesCapturees[i];
            tableauClassementDesTrous[nbTrousJouables] = i;
            nbTrousJouables++;
        }

    }

    // Tri par sélection des tableaux tmp_tableau & tableauClassementDesTrous
    for( int i = 0; i < nbTrousJouables - 1; i++ ) {

        int indiceMin = i;

        for( int j = i+1; j < nbTrousJouables; j++ ) {
            if( tmp_tableau[indiceMin] < tmp_tableau[j] ) indiceMin = j;
        }

        // tri dans tmp_tableau
        int tmp_value = tmp_tableau[i];
        tmp_tableau[i] = tmp_tableau[indiceMin];
        tmp_tableau[indiceMin] = tmp_value;

        // tri dans tableauClassementDesTrous
        tmp_value = tableauClassementDesTrous[i];
        tableauClassementDesTrous[i] = tableauClassementDesTrous[indiceMin];
        tableauClassementDesTrous[indiceMin] = tmp_value;
    }

    // Optimisation de la taille de tableauClassementDesTrous
    tmp_tableau = new int[nbTrousJouables];
    for( int i = 0; i < tmp_tableau.length; i++ )
        tmp_tableau[i] = tableauClassementDesTrous[i];

    tableauClassementDesTrous = tmp_tableau;

    return tableauClassementDesTrous;

}
```

Chapitre 4

La classe PartieAwale

4.1 Classe de base et attributs

```
public class PartieAwale {

    private static int n;
    private static Etat[] etats;

    private static int modeDeJeu;
    private static String[] joueurs;
    private static boolean[] jokers;

    public static java.util.Scanner scanner = new java.util.Scanner(System.in);

    ...

}
```

4.2 Méthode main(String args[])

```
public static void main(String args[]) {

    // Initialisation des joueurs
    joueurs = new String[2];

    // Initialisation des jokers
    jokers = new boolean[4];
    for(int i=0; i < jokers.length; i++)
        jokers[i] = true; // Un joker est encore disponible quand il vaut true

    // Initialisation du tableau d'etats
    etats = new Etat[40];

    selectionnerModeDeJeu();

    // Debut du match
    System.out.println("\n
        ************************************************************");
    System.out.println("Début du match: " + joueurs[0] + " vs " + joueurs[1]);
    System.out.println("
        ************************************************************");
    System.out.println("\n");

    etats[n] = new Etat();

    boolean estTerminee = false;
    while( !estTerminee ) {

        // En mode solo: tour du joueur - en mode multi : tours des joueurs
        if(modeDeJeu == 2 || ((n%2) == 0 && modeDeJeu == 1)) {
            System.out.println("************** Manche " + (n+1) + " - A " +
                joueurs[n%2] + " de jouer **************\n");

            afficherPlateauComplet();

            afficherMenu();
        }

        // En mode solo: tour de l'ordinateur
        if(modeDeJeu == 1 && (n%2) == 1) {
            jouerCoupIA();
        }

        n++;
        estTerminee = etats[n].estTerminee();

    }

    // Fin du jeu
    System.out.println("
        ************************************************************");
    System.out.println("Fin du jeu: " + ( (etats[n].captures[0] > etats[n].
        captures[1]) ? joueurs[0] : joueurs[1] ) + " a gagne!");
    System.out.println("
        ************************************************************");
    System.out.println("\n");

}
```

4.3 Méthode selectionnerModeDeJeu()

```
/**
* Méthode permettant d'afficher les différents modes de jeu disponible et d'en sé
    lectionner un.
* @return void
*/
public static void selectionnerModeDeJeu() {

    System.out.println("
        ************************************************************");
    System.out.println("Bienvenue dans le jeu Awale");
    System.out.println("
        ************************************************************\n");

    System.out.println("1. Faire une partie solo");
    System.out.println("2. Faire une partie multijoueurs");
    System.out.println("\n* Veuillez choisir votre mode de jeu:");

    int choix = scanner.nextInt();
    while( choix < 1 || choix > 2 ) {
        System.out.println("Mode de jeu incorrect. Veuillez-choisir votre mode de
            jeu:");
        choix = scanner.nextInt();
    }

    if( choix == 1 ) {
        joueurs[0] = "vous";
        joueurs[1] = "ordinateur";
        modeDeJeu = 1;
    } else {
        modeDeJeu = 2;
        System.out.println("* Veuillez introduire le nom du joueur 1:");
        joueurs[0] = scanner.next();
        System.out.println();

        System.out.println("* Veuillez introduire le nom du joueur 2:");
        joueurs[1] = scanner.next();
        System.out.println();
    }
}
```

4.4 Méthode afficherMenu()

```
/**
* Méthode permettant d'afficher le menu du jeu et d'exécuter une action (jouer un
     coup / afficher l'historique / utiliser un joker)
* @return void
*/
public static void afficherMenu() {

    System.out.println("\n#### Menu de jeu :\n");

    System.out.println("1. Jouer un coup");
    System.out.println("2. Afficher l'historique de la partie et jouer un coup");

    // On affiche la possibilité de jouer un joker si au moins un joker est
        disponible
    boolean peutUtiliserUnJoker = false;
    if( (n%2 == 0 && (jokers[0] == true || jokers[1] == true)) || (n%2 == 1 && (
        jokers[2] == true || jokers[3] == true)) ){
        System.out.println("3. Utiliser un joker et jouer un coup");
        peutUtiliserUnJoker = true;
    }

    System.out.println("\n* Que voulez-vous vous faire (1-3):");
    int choix = scanner.nextInt();

    while( (choix < 1 || choix > 3) || (choix == 3 && !peutUtiliserUnJoker) ) {
        if(choix == 3 && !peutUtiliserUnJoker)
            System.out.println("\n* Vous n'avez plus de joker disponible. Que
                voulez-vous faire? (1-2)");
        else
            System.out.println("\n* Le numéro de l'action ne peut être compris qu
                'entre 1 et 3. Quel voulez-vous faire?");

        choix = scanner.nextInt();
    }

    switch(choix) {
        case 1:
            jouerUnCoup();
            break;
        case 2:
            afficherDeroulement();
            break;
        case 3:
            utiliserUnJoker();
            break;
    }

}
```

4.5 Méthode afficherPlateauComplet()

```
/**
* Méthode permettant d'afficher le plateau complet (plateau / captures / jokers
    disponibles) dans la console de façon user friendly.
* @return void
*/
public static void afficherPlateauComplet() {

    String plateau ="";

    plateau += plateauVisuel() + "\n";

    plateau += "";

    if( (n%2 == 0 && jokers[0] == true) || (n%2 == 1 && jokers[2] == true) ) {
        plateau += "\nVos␣jokers␣encore␣disponibles:␣␣-␣Le␣nombre␣de␣captures␣
            pour␣chaque␣trou\n";

        if((n%2 == 0 && jokers[1] == true) || (n%2 == 1 && jokers[3] == true))
            plateau += "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣Le␣classement␣des␣trous
                ␣les␣plus␣efficaces";

    } else if( (n%2 == 0 && jokers[1] == true) || (n%2 == 1 && jokers[3] == true)
        ){
        plateau += "\nVos␣jokers␣encore␣disponibles:␣␣-␣Le␣classement␣des␣trous␣
            les␣plus␣efficaces";
    }

    System.out.println(plateau);

}
```

4.6 Méthode plateauVisuel()

```
/**
* Méthode permettant d'obtenir une String contenant la version user friendly du
    plateau.
* @return String - le plateau
*/
public static String plateauVisuel() {

    String plateau = "";

    plateau += "Votre adversaire => ";

    for( int i = ( (n%2 == 0) ? 6 : 0 ); i < ( (n%2 == 0) ? 12 : 6 ); i++ )
        plateau += "| " + etats[n].plateau[i] + "| ";

    plateau += "     Captures: " + ( (n%2 == 0) ? etats[n].captures[1] : etats[n].
        captures[0] ) + "\n";
    plateau += "                                  ----------------------\n";
    plateau += "              Vous => ";

    for( int i = ( (n%2 == 0) ? 5 : 11 ); i >= (n%2)*6; i-- )
        plateau += "| " + etats[n].plateau[i] + "| ";

    plateau += "     Captures: " + etats[n].captures[n%2];

    return plateau;

}
```

4.7 Méthode jouerUnCoup()

```
/**
* Méthode permettant de demander au joueur quel trou jouer et le jouer si celui-
    ci est jouable.
* Incrémente n et ajoute une nouvelle instance d'Etat á etats
* @return void
*/
public static void jouerUnCoup() {

    System.out.println("\n#### Jouer un coup:\n");

    System.out.println(plateauVisuel());

    String coupVisuel = "";
    coupVisuel += "                    ^   ^   ^   ^   ^   ^\n";
    coupVisuel += "   Numéro du trou =>  1   2   3   4   5   6\n";

    System.out.println(coupVisuel);

    System.out.println("* Quel trou choisissez-vous? (1-6)");

    // On joue le trou uniquement si joue(int t) de la classe Etat ne renvoit
        aucune exception
    boolean estLegal = false;
    while(!estLegal) {

        int trou = scanner.nextInt();

        if(n%2 == 0)
            trou = (6 - trou) + 1;
        else
            trou = (12 - trou) + 1;

        try {
            // On double la taille d'etats si le tableau est rempli
            if(n+1 == etats.length)
                etats = agrandirTableau(etats);

            etats[n+1] = etats[n].joue(trou - 1);

            estLegal = true;
        } catch (Exception e) {
            System.out.println("* Le trou sélectionné est invalide. Quel trou
                voulez-vous jouer?");
        }

    }

}
```

4.8 Méthode jouerCoupIA()

```
/**
* Méthode permettant de jouer le coup le plus efficace en mode solo.
* Incrémente n et ajoute une nouvelle instance d'Etat á etats
* @return void
*/
public static void jouerCoupIA() {

    System.out.println("**********Manche " + (n+1) + " L'ordinateur a joué
        son coup**********\n");

    // On cherche le trou le plus efficace
    int trou = etats[n].classementDesTrous()[0];

    // On double la taille d'etats si le tableau est rempli
    if(n+1 == etats.length)
        etats = agrandirTableau(etats);

    etats[n+1] = etats[n].joue(trou);

    System.out.println("* L'ordinateur a joué son coup. Faites [ENTER] pour
        continuer");
    try {
        System.in.read();
    } catch(Exception e) { }

}
```

4.9 Méthode utiliserUnJoker()

```
/**
* Méthode permettant de demander au joueur quel joker il veut utiliser, l'
    utiliser et ensuite jouer un coup.
* @return void
*/
public static void utiliserUnJoker() {

    System.out.println("\n#### Utiliser un joker:\n");

    // On vérifie quels jokers sont encore disponibles
    boolean capturesDispo = false;
    if( (n%2 == 0) && (jokers[0] == true) || (n%2 == 1) && (jokers[2] == true) )
        capturesDispo = true;

    boolean classementDispo = false;
    if( (n%2 == 0) && (jokers[1] == true) || (n%2 == 1) && (jokers[3] == true) )
        classementDispo = true;

    // Affichage du menu de sélection sur base des jokers disponibles
    System.out.println("1. Jouer un coup sans utiliser de joker");
    if( capturesDispo ) {

        System.out.println("2. Montrez-moi le nombre de captures pour chaque trou
            ");
        if( classementDispo )
            System.out.println("3. Montrez-moi le classement des trous les plus
                efficaces");

    } else if( classementDispo )
        System.out.println("2. Montrez-moi le classement des trous les plus
            efficaces");

    System.out.println("\n* Quel joker voulez-vous utiliser?");
    int choix = scanner.nextInt();

    while( choix < 1 || choix > 3 || (capturesDispo && !classementDispo && choix
        == 3) || (!capturesDispo && classementDispo && choix == 3) ||
        (!capturesDispo && !classementDispo && (choix == 2 || choix == 3)) ) {

        System.out.println("Le joker sélectionné est invalide. Quel joker voulez-
            vous utiliser?");
        choix = scanner.nextInt();
    }

    switch(choix) {
        case 1:
            break;
        case 2:
            if(capturesDispo) afficherPredictions();
            else afficherClassement();
            break;
        case 3:
            afficherClassement();
            break;
    }
    // On joue un coup
    jouerUnCoup();
}
```

4.10 Méthode afficherDeroulement()

```
/**
* Méthode permettant d'afficher l'historique du jeu dans la console.
* @return void
*/
public static void afficherDeroulement() {

    System.out.println("\n#### Historique de la partie:\n");

    for(int i=0; i <= n; i++) {
        System.out.println("**** " + i);
        System.out.println(etats[i] + "\n");
    }

    // On joue un coup
    jouerUnCoup();

}
```

4.11 Méthode afficherPredictions()

```
/**
* Méthode permettant d'afficher dans la console une prédiction du nombre de
    captures pour chaque trou.
* @return void
*/
public static void afficherPredictions() {

    System.out.println("\n#### Prédiction du nombre de captures pour chaque trou
        :\n");

    int[] piecesCapturees = etats[n].piecesCapturees();

    String predictions = "";
    predictions += plateauVisuel() + "\n";

    predictions += "                    ^   ^   ^   ^   ^   ^\n";
    predictions += "Nbres de captures => ";

    for(int i = ((n%2 == 0) ? 5 : 11); i >= (n%2)*6; i--)
        predictions += ((piecesCapturees[i] == -1) ? "X" : piecesCapturees[i]) +
            "   ";

    System.out.println(predictions);

    // Ce joker n'est plus disponible pour le joueur
    if(n%2 == 0)
        jokers[0] = false;
    else
        jokers[2] = false;

}
```

4.12 Méthode afficherClassement()

```
/**
* Méthode permettant d'affiche dans la console un classement des trous les plus
    efficaces.
* @return void
*/
public static void afficherClassement() {

    System.out.println("\n#### Classement des trous les plus efficaces :\n");

    int[] classementDesTrous = etats[n].classementDesTrous();

    // On initialise classementParIndice avec un tableau rempli de 99 pour
        trouver
    // les trous injouables par la suite
    int[] classementParIndice = new int[6];
    for(int i = 0; i < classementParIndice.length; i++)
        classementParIndice[i] = 99;


    // On trie le classement pour s'adapter á l'affichage des trous
    for(int i = 0; i < classementDesTrous.length; i++) {
        if(n%2 == 0)
            classementParIndice[classementDesTrous[i]] = i;
        else
        classementParIndice[classementDesTrous[i]-6] = i;
    }

    // Affichage du classement en dessous du plateau
    String classement = "";
    classement += plateauVisuel() + "\n";
    classement += "                    ^   ^   ^   ^   ^   ^\n";
    classement += "       Classement => ";

    for(int i = 5; i >= 0; i--) {
        if(classementParIndice[i] == 99)
            classement += "X   ";
        else
            classement += (classementParIndice[i]+1) + "   ";
    }

    System.out.println(classement);

    // Ce joker n'est plus disponible pour le joueur
    if(n%2 == 0)
        jokers[1] = false;
    else
        jokers[3] = false;

}
```

4.13 Méthode agrandirTableau(Etat[] tableau)

```
/**
* Methode permettant de doubler la taille d'un tableau d'int
* @param Etat[] tableau dont la taille doit être doublée
* @return Etat[] tableau dont la taille a été doublée
*/
public static Etat[] agrandirTableau(Etat[] tableau) {

    Etat[] nouveauTableau = new Etat[tableau.length * 2];

    for( int i = 0; i < tableau.length; i++ )
        nouveauTableau[i] = tableau[i];

    return nouveauTableau;

}
```

Chapitre 5

Les tests

Les tests suivants représentent l'ensemble des possibilités à traiter dans le jeu. Chaque test a été éffectué pour les deux rangées du plateau (Nord et Sud) dans la classe TestEtat *(Voir Annexe C)*.

5.1 Test de la méthode egrene()

n° test	t (paramètre)	plateau (avant)	plateau (après)	t (renvoyé)	Exception	commentaire (éventuel)
1	-1	4 0 0 0 0 0 0 0 0 0 0 0	4 0 0 0 0 0 0 0 0 0 0 0	/	OUI	trou non jouable
2	12	4 0 0 0 0 0 0 0 0 0 0 0	4 0 0 0 0 0 0 0 0 0 0 0	/	OUI	trou non jouable
3	2	4 0 0 0 0 0 0 0 0 0 0 0	4 0 0 0 0 0 0 0 0 0 0 0	2	/	
4	0	4 0 0 0 0 0 0 0 0 0 0 0	0 1 1 1 1 0 0 0 0 0 0 0	4	/	
5	10	0 0 0 0 0 0 0 4 0 0 0 0	1 1 1 0 0 0 1 0 0 0 0 0	2	/	
6	1	0 14 0 0 0 0 0 0 0 0 0 0	1 0 2 2 2 1 1 1 1 1 1 1	4	/	Règle n°6
7	11	0 0 0 0 0 0 1 0 0 0 0 0	1 0 0 0 0 0 0 0 0 0 0 0	0	/	

5.2 Test de la méthode estNourri()

n° test	n	plateau (avant)	Adversaire	Boolean renvoyé	commentaire (éventuel)
1	0	4 0 0 0 0 0 0 0 0 0 0 0	joueur Sud	False	
2	1	4 0 0 0 0 0 0 0 0 0 0 0	joueur Nord	TRUE	
3	0	0 0 0 0 0 0 4 0 0 0 0 0	joueur Sud	TRUE	
4	1	0 0 0 0 0 0 4 0 0 0 0 0	joueur Nord	False	

5.3 Test de la méthode capture(int t)

n° test	n	t (parametre)	plateau (avant)	plateau (après)	nbr graines capturées (renvoyé)	Exception	commentaire (éventuel)
1	1	1	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	/	
2	0	9	0 0 0 0 0 0 0 0 2 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	2	/	
3	1	5	2 3 2 3 2 3 2 2 2 2 3 2	0 0 0 0 0 0 2 2 2 2 3 2	15	/	
4	0	11	2 2 2 2 2 2 2 2 2 5 2 2	2 2 2 2 2 2 2 2 2 5 0 0	4	/	
5	0	5	2 2 2 2 2 2 2 2 2 5 2 2	2 2 2 2 2 2 2 2 2 5 2 2	0	/	Capture dans son plateau
6	1	2	2 1 2 2 2 2 2 2 2 5 2 2	2 1 0 2 2 2 2 2 2 5 2 2	2	/	
7	0	-1	4 0 0 0 0 0 0 0 0 0 0 0	4 0 0 0 0 0 0 0 0 0 0 0	/	OUI	trou non jouable
8	1	5	2 2 3 2 3 2 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	14	/	Affame

5.4 Test de la méthode estAffame(int t)

n° test	n	t (parametre)	plateau (avant)	plateau (après)	Boolean renvoyer	Exception	commentaire (éventuel)
1	0	0	1 0 0 0 0 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0 0 0 0 0	True	/	
2	1	6	0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 1 0	True	/	
3	1	11	0 0 0 0 0 0 1 0 0 0 0 0	1 0 0 0 0 0 0 0 0 0 0 0	False	/	
4	0	5	0 0 0 0 0 1 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 1	False	/	
6	0	-1	4 0 0 0 0 0 0 0 0 0 0 0	4 0 0 0 0 0 0 0 0 0 0 0	/	OUI	trou non jouable

5.5 Test de la méthode estLegal(int t)

n° test	n	t (parametre)	plateau (avant)	plateau (après)	Boolean optenu	Exception	commentaire (éventuel)
1	1	5	0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	False	/	
2	0	1	1 0 0 0 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0 0 0 0	False	/	
3	0	5	1 1 1 1 1 1 0 0 0 0 0 0	1 1 1 1 1 1 0 0 0 0 0 0	True	/	
4	1	6	1 1 1 1 1 1 0 0 0 0 0 1	1 1 1 1 1 1 0 0 0 0 0 1	True	/	
5	0	10	0 0 0 0 0 0 0 5 0 0 0 0	0 0 0 0 0 0 0 5 0 0 0 0	False	/	
6	0	-1	1 0 0 0 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0 0 0 0	/	oui	

5.6 Test de la méthode joue(int t)

n° test	n	t (parametre)	plateau (avant)	plateau Obtenu	Exception	commentaire (éventuel)
1	1	5	0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	oui	trou non jouable
2	0	-1	1 0 0 0 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0 0 0 0	oui	trou non jouable
3	1	6	0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 1	oui	trou non jouable
4	0	5	0 0 0 0 0 1 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 1	/	
5	1	10	0 1 0 0 0 0 0 0 1 0 0 0	0 1 0 0 0 0 0 1 0 0 0 0	/	
6	1	11	0 0 0 0 0 0 1 0 0 0 0 0	1 0 0 0 0 0 0 0 0 0 0 0	/	

5.7 Test de la méthode estTerminee()

n° test	n	Score nord	Score Sud	plateau (avant)	plateau (après)	Boolean optenu	Exception	commentaire (éventuel)
1	1	18	18	1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1	False	/	
2	1	23	25	0 0 0 0 0 0 1 0 0 0 0 0	0 0 0 0 0 0 1 0 0 0 0 0	True	/	
3	0	10	10	5 4 3 2 1 0 0 0 0 0 0 0	5 4 3 2 1 0 0 0 0 0 0 0	True	/	
4	0	10	09	0 0 0 0 0 0 5 4 3 2 2 0	0 0 0 0 0 0 5 4 3 2 2 0	False	/	
5	1	23	23	0 0 0 0 0 0 0 0 1 0 0 1	0 0 0 0 0 0 0 0 1 0 0 1	True	/	

5.8 Test de la méthode piecesCapturees()

n° test	n	plateau (avant)	plateau optenu	Exception	commentaire (éventuel)
1	1	0 0 0 0 0 0 0 0 0 0 0 0	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	/	
2	1	1 1 1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 2 -1 -1 -1 -1 -1 -1	/	
3	0	11 10 9 8 7 6 1 1 1 1 1 1	12 12 12 12 12 12 -1 -1 -1 -1 -1 -1	/	
5	1	0 0 0 0 0 0 12 0 0 0 0 0	-1 -1 -1 -1 -1 -1 2 -1 -1 -1 -1 -1	/	
6	1	1 1 1 1 1 1 1 1 1 1 1 1	-1 -1 -1 -1 -1 -1 2 0 0 0 0 0	/	

5.9 Test de la méthode classementDesTrous()

n° test	n	plateau (avant)	plateau optenu	Exception	commentaire (éventuel)
1	1	0 0 0 0 0 0 0 0 0 0 0 0	/	/	
2	1	1 1 1 1 1 1 1 0 0 0 0 0	11,	/	
3	1	1 1 1 1 1 1 5 4 8 4 3 2	9, 11, 10, 8, 6, 7	/	
4	0	11 9 7 5 3 0 1 1 1 1 1 1	0, 1, 2, 3, 4,	/	

Chapitre 6

Conclusion

A l'issue de deux semaines de travail intensif, nous pouvons affirmer que ce premier projet d'algorithmique s'est terminé sans encombre et dans les délais. Nous avons réalisé une solution répondant parfaitement au cahier des charges, élégante techniquement et garantissant les critères d'évolutivité.

Afin de mener à bien et de façon optimale le développement du jeu, nous avons consacré un point d'honneur à l'organisation de celui-ci. Dès lors, nous nous sommes tourné vers plusieurs outils collaboratifs afin d'optimiser le travail en commun et la communication au sein de notre petite équipe. Nous avons donc décidé de travailler avec la solution de versionage et de développement collaboratif de Github. Grâce à elle, nous oeuvrions chacun de notre côté sur deux branches différentes et y transférions nos modifications au fûr et à mesure. Par conséquent, l'outil nous a permis de coder simultanément sans créer de conflits entre nos différentes versions. Cependant, n'ayant jamais travaillé avec un tel système auparavant, nous avons éprouvé quelques difficultés à l'intégrer à notre flux de travail dans un premier temps. Une fois tous les concepts acquis, nous avons pu utiliser l'outil au maximum de ses possibilités et pouvons attester de sa facilité d'utilisation et de son efficacité au sein d'un projet en équipe.

Ensuite, nous avons accordé beaucoup d'importance au soucis du détails aussi bien au niveau du code que de l'expérience utilisateur. Outre le fait que nous voulions délivrer une solution évolutive, nous avons développé une version du jeu à la fois performante, élégante et simple à utiliser, objectifs assez difficiles à atteindre de prime abord puisqu'elle devait tourner intégralement dans la console java. De ce fait, une série de fonctionnalités supplémentaires ont été rajoutée à la version de base afin d'en augmenter le plaisir de jeu. En effet, nous avons implémenté un système de joker, rendant le jeu plus amusant et plus compliqué à la fois, ainsi qu'un mode de jeu solo permettant d'affronter l'ordinateur en duel.
De plus, nous avons débuté une version graphique du jeu mais les délais approchants combiné au fait que nous n'avions pas les compétences suffisantes en la matière nous ont contraint d'abandonner l'idée et de nous concentrer sur la version en console.

Suite au développement de ce jeu, nous avons également pu dégager plusieurs opportunités de développement futures. Un module de sauvegarde de partie serait une fonctionnalité plus qu'intéressante, permettant aux utilisateurs de continuer leurs parties à n'importe quel moment. De surcroit, nous pensons que la possibilité de jouer contre ses amis en ligne rendrait le jeu encore

plus attractif et amusant. Enfin, une version mobile du jeu permettrait d'élargir drastiquement la base d'utilisateurs potentiels.

En définitive, nous avons eu l'opportunité, grâce à ce projet, d'améliorer et d'approfondir nos connaissances en java ainsi qu'à nous familiariser aux bonnes pratiques du langage. Nous avons également pu appliquer l'ensemble des savoir-faire acquis en cours d'algorithmique et d'introduction à l'orienté objet. C'est donc avec une certaine fierté que nous délivrons ce premier projet aujourd'hui.

Annexe A

Etat.java

```
//IPL, algorithmique (I1010), projet awale
//
// Q1, 2014
//
// Nom et prenom de l'etudiant 1 : WAGEMANS Jeremy
// Nom et prenom de l'etudiant 2 : LEBON Sebastien

public class Etat {
        int n;
        int[] plateau;
        int[] captures;

        final static int NBR_TROUS = 12;
        final static int NBR_ADVERSAIRES = 2;

        // Constructeur de base
        public Etat() {
                this.n = 0;

                this.plateau = new int[NBR_TROUS];
                for( int i = 0; i < this.plateau.length; i++ )
                        this.plateau[i] = 4;

                this.captures = new int[NBR_ADVERSAIRES];
                for( int i = 0; i < this.captures.length; i++ )
                        this.captures[i] = 0;
        }

        // Constructeur pour tours n > 0
        public Etat(Etat etat) {
                if( etat == null ) throw new IllegalArgumentException("Etat␣
                    invalide");

                this.n = etat.n;

                this.plateau = new int[NBR_TROUS];
                for( int i = 0; i < plateau.length; i++ )
                        this.plateau[i] = etat.plateau[i];

                this.captures = new int[NBR_ADVERSAIRES];
                for( int i = 0; i < captures.length; i++ )
                        this.captures[i] = etat.captures[i];
        }
```

```
// Constructeur pour les tests
public Etat(int n, int[] plateau, int[] captures) {
        if( n < 0 ) throw new IllegalArgumentException("Nombre de tours
            incorrect");
        else if( plateau == null || plateau.length > NBR_TROUS ) throw
            new IllegalArgumentException("Plateau invalide ou taille du
            tableau > 12");
        else if( captures == null || captures.length > NBR_ADVERSAIRES )
            throw new IllegalArgumentException("Captures invalide ou
            taille du tableau > 2");

        this.n = n;
        this.plateau = plateau;
        this.captures = captures;

}

public String toString(){

        String plateauNord = ""; // Chaine qui représente le contenu du
            plateau côté nord
        String plateauSud = ""; // Chaine qui représente le contenu du
            plateau côté sud

        for( int i=0; i < NBR_TROUS; i++ ) {
                if( i < 6 )
                        plateauNord += plateau[i] + ", ";
                else
                        plateauSud = plateau[i] + ", " + plateauSud;
        }

        String repTextuelle = captures[0] + "\n"; // Représentation
            textuelle de l'Etat

        repTextuelle += plateauNord + "\n";

        repTextuelle += plateauSud + "\n";

        repTextuelle += captures[1];

        return repTextuelle;

}

/**
* Méthode permettant de récupérer les graines du trou t et d'égrener
    chaque trou (suivant t) d'une graine jusqu'á ce qu'il n'y en ait plus.
* Cette méthode modifie les attributs de l'objet courant.
* @param t position du trou duquel on prend les graines á égréner.
* @return int la position du dernier trou égréné
*/
public int egrene(int t) {

        if( t < 0 || t > NBR_TROUS-1 ) throw new IllegalArgumentException
            ("Numero du trou invalide");

        // On récupère les graines contenues dans le trou t
        int nbGraines = plateau[t];
        plateau[t] = 0;
```

```
        // Si t vaut NBR_TROUS - 1, position vaut 0 car on passe chez l'
            adversaire et on boucle le tour
        int position = (t == NBR_TROUS-1) ? 0 : t+1;

        while( nbGraines > 0 ) {

                // Si le nombre de graines est supérieur ou égal au
                    nombre de trou, on passe le trou duquel
                // on a récupéré les graines
                if( position != t ) {
                        plateau[position] += 1;
                        nbGraines--;
                }

                position++;

                // Une fois qu'on arrive á la fin du plateau, on retourne
                     á la case 0
                if( position == NBR_TROUS )
                        position = 0;

        }

        return (position == 0) ? NBR_TROUS-1 : position-1;

}

/**
* Méthode permettant de savoir si l'adversaire est nourri. Il est nourri
    si un de ses trous contient au moins une graine.
* Cette méthode ne modifie pas les attributs de l'objet courant.
* @return boolean retourne true si l'adversaire est nourri et false s'il
    ne l'est pas.
*/
public boolean estNourri() {

        int position = 0;
        if( n%2 == 0 )
                position = NBR_TROUS/2;
        int borneSup = position + (NBR_TROUS/2);

        int nbGraines = 0;

        // Teste pour chaque trou s'il y a une graine. S'arrête dès qu'on
             en trouve une.
        while( position < borneSup && nbGraines == 0 ) {

                if( plateau[position] > 0 )
                        nbGraines = plateau[position];

                position++;

        }

        return nbGraines > 0;
}

/**
* Méthode permettant de capturer les graines de l'adversaire (si un trou
    contient 2 ou 3 graines) après égrénage
```

```
* Cette méthode modifie les attributs de l'objet courant.
* @param t numéro du trou á partir duquel on fait nos captures
* @return int retourne le nombre de graines capturées
*/
public int capture(int t) {

        if( t < 0 || t > NBR_TROUS-1 ) throw new IllegalArgumentException
            ("Numéro du trou invalide");

        int nbGraines = 0;

        int position = t;

        int borneMinimale = 0;
        if( n%2 == 0 )
                borneMinimale = NBR_TROUS/2;

        int borneMaximale = borneMinimale + (NBR_TROUS/2);

        // Si position est chez l'adversaire (< borneMaximale && >=
            borneMinimale) et que position contient 2 ou 3 graines
        // On capture
        while( ((position >= borneMinimale) && (position < borneMaximale)
            ) &&
        (plateau[position] == 2 || plateau[position] == 3) ) {

                nbGraines += plateau[position];
                plateau[position] = 0;
                position--;

        }

        return nbGraines;
}

/**
* Méthode permettant de savoir si l'adversaire est affamé après égrénage.
* Cette méthode ne modifie pas les attributs de l'objet courant.
* @param t numéro duquel on égrène
* @return boolean retourne true si l'adversaire est affamé.
*/
public boolean estAffame(int t) {

        if( t < 0 || t > NBR_TROUS-1 ) throw new IllegalArgumentException
            ("Numéro du trou invalide");

        Etat tmp_etat = new Etat(this);

        int positionDerniereGraine = tmp_etat.egrene(t);
        tmp_etat.capture(positionDerniereGraine);

        return !(tmp_etat.estNourri());

}

/**
* Méthode permettant de savoir si le trou t peut être joué dans l'état
    courant.
* Cette méthode ne modifie pas les attributs de l'objet courant.
* @param t numéro du trou á tester.
```

```
     * @return boolean retourne true si.
     */
    public boolean estLegal(int t) {

            if( t < 0 || t > NBR_TROUS-1 ) throw new IllegalArgumentException
                ("Numéro du trou invalide");

            // Teste si t ne se trouve pas du côté du joueur
            if( ((n%2 == 0) && (t >= NBR_TROUS/2 && t <= NBR_TROUS-1)) || ((n
                %2 != 0) && (t >= 0 && t <= (NBR_TROUS/2)-1)) )
                    return false;

            // Teste s'il n'y a pas de graine dans le trou
            if( plateau[t] == 0 )
                    return false;

            // Teste si l'adversaire est nourri après égrénage
            Etat tmp_etat = new Etat(this);
            int positionDerniereGraine = tmp_etat.egrene(t);
            return tmp_etat.estNourri();

    }

    /**
     * Méthode permettant de jouer un coup depuis le trou t.
     * Cette méthode ne modifie pas les attributs de l'objet courant.
     * @param t numéro du trou duquel le joueur joue son coup.
     * @return instance de la classe Etat représentant l'Etat après le coup.
     */
    public Etat joue(int t) {

            if( !estLegal(t) ) throw new IllegalArgumentException("Coup
                invalide");

            Etat nouvelEtat = new Etat(this);
            int positionDerniereGraine = nouvelEtat.egrene(t); // On égrène á
                pd de t

            if( !nouvelEtat.estAffame(t) )
                    nouvelEtat.captures[n%2] += nouvelEtat.capture(
                        positionDerniereGraine); // On capture les graines de
                        l'adversaire

            nouvelEtat.n += 1; // On passe au tour suivant

            return nouvelEtat;

    }

    /**
     * Méthode permettant de savoir si la partie est terminée ou pas.
     * Cette méthode ne modifie pas les attributs de l'objet courant.
     * @return boolean retourne vrai si la partie est terminée.
     */
    public boolean estTerminee() {

            // Si le nombre de captures >= 25
            if( captures[0] >= 25 || captures[1] >= 25 )
                    return true;
```

```
        // Si l'adversaire est nourri
        if( estNourri() )
                return false;

        int i = (n%2)*(NBR_TROUS/2);
        int borneSup = (n%2 == 0) ? NBR_TROUS/2 : NBR_TROUS;
        int grainesNecessaires = ((NBR_TROUS/2)-1);

        // Vérifie s'il n'y a plus moyen de nourrir l'adversaire
        boolean estTerminee = true;
        while( i < borneSup && estTerminee ) {

                if(plateau[i] > grainesNecessaires)
                        estTerminee = false;
                i++;
                grainesNecessaires--;

        }

        return estTerminee;

}

/**
* Méthode permettant d'obtenir un tableau de telle sorte que pour chaque
    indice i du tableau,
* t[i] représente le nombre de graines capturées lorsque le trou i est
    joué.
* Si le trou i ne peut être joué, t[i] sera -1.
* Cette méthode ne modifie pas les attributs de l'objet courant.
* @return int[] le tableaut représentant le nombre de graines capturées
    pour chaque trou.
*/
public int[] piecesCapturees() {

        int[] tableauPiecesCapturees = new int[NBR_TROUS];

        for( int i = 0; i < tableauPiecesCapturees.length; i++ ){
                if( estLegal(i) ) {
                        tableauPiecesCapturees[i] = joue(i).captures[n%2]
                            - this.captures[n%2]; // On soustrait les
                            captures de l'etat courant avec celles du
                            nouvel état afin d'enregistrer les pieces
                            capturees
                } else {
                        tableauPiecesCapturees[i] = -1;
                }
        }

        return tableauPiecesCapturees;

}

/**
* Méthode permettant d'obtenir un tableau contenant les indices i tel que
     si i est joué, il y aura plus de captures que si i+1 était joué.
* Cette méthode ne modifie pas les attributs de l'objet courant.
* @return int[] le tableau représentant le classement des captures.
*/
public int[] classementDesTrous() {
```

```
                int[] tableauPiecesCapturees = piecesCapturees();
                int[] tableauClassementDesTrous = new int[NBR_TROUS]; //
                    Contiendra le classement des indices
                int[] tmp_tableau = new int[NBR_TROUS]; // Contiendra une copie
                    de tableauPiecesCapturees sans les trous injouables

                // Recopie tous les trous jouables dans tmp_tableau et leur
                    indice dans tableauClassementDesTrous
                int nbTrousJouables = 0;
                for( int i = 0; i < tableauPiecesCapturees.length; i++ ) {

                        if( tableauPiecesCapturees[i] > -1 ) {
                                tmp_tableau[nbTrousJouables] =
                                    tableauPiecesCapturees[i];
                                tableauClassementDesTrous[nbTrousJouables] = i;
                                nbTrousJouables++;
                        }

                }

                // Tri par sélection des tableaux tmp_tableau &
                    tableauClassementDesTrous
                for( int i = 0; i < nbTrousJouables - 1; i++ ) {

                        int indiceMin = i;

                        for( int j = i+1; j < nbTrousJouables; j++ ) {
                                if( tmp_tableau[indiceMin] < tmp_tableau[j] )
                                    indiceMin = j;
                        }

                        // tri dans tmp_tableau
                        int tmp_value = tmp_tableau[i];
                        tmp_tableau[i] = tmp_tableau[indiceMin];
                        tmp_tableau[indiceMin] = tmp_value;

                        // tri dans tableauClassementDesTrous
                        tmp_value = tableauClassementDesTrous[i];
                        tableauClassementDesTrous[i] = tableauClassementDesTrous[
                            indiceMin];
                        tableauClassementDesTrous[indiceMin] = tmp_value;
                }

                // Optimisation de la taille de tableauClassementDesTrous
                tmp_tableau = new int[nbTrousJouables];
                for( int i = 0; i < tmp_tableau.length; i++ )
                        tmp_tableau[i] = tableauClassementDesTrous[i];

                tableauClassementDesTrous = tmp_tableau;

                return tableauClassementDesTrous;

        }
}
```

Annexe B

PartieAwale.java

```
//IPL, algorithmique (I1010), projet awale
//
// Q1, 2014
//
// Nom et prenom de l'etudiant 1 : WAGEMANS Jeremy
// Nom et prenom de l'etudiant 2 : LEBON Sebastien
public class PartieAwale {

    private static int n;
    private static Etat[] etats;

    private static int modeDeJeu;
    private static String[] joueurs;
    private static boolean[] jokers;

    public static java.util.Scanner scanner = new java.util.Scanner(System.in);

    public static void main(String args[]) {

        // Initialisation des joueurs
        joueurs = new String[2];

        // Initialisation des jokers
        jokers = new boolean[4];
        for(int i=0; i < jokers.length; i++)
            jokers[i] = true; // Un joker est encore disponible quand il vaut
                true

        // Initialisation du tableau d'etats
        etats = new Etat[40];

        selectionnerModeDeJeu();

        // Debut du match
        System.out.println("\n
            ************************************************************");
        System.out.println("Début du match: " + joueurs[0] + " vs " + joueurs[1])
            ;
        System.out.println("
            ************************************************************");
        System.out.println("\n");

        etats[n] = new Etat();
```

```
        boolean estTerminee = false;
        while( !estTerminee ) {

            // En mode solo: tour du joueur - en mode multi : tours des joueurs
            if(modeDeJeu == 2 || ((n%2) == 0 && modeDeJeu == 1)) {
                System.out.println("************* Manche " + (n+1) + " - A " +
                    joueurs[n%2] + " de jouer *************\n");

                afficherPlateauComplet();

                afficherMenu();
            }

            // En mode solo: tour de l'ordinateur
            if(modeDeJeu == 1 && (n%2) == 1) {
                jouerCoupIA();
            }

            n++;

            estTerminee = etats[n].estTerminee();

        }

        // Fin du jeu
        System.out.println("
            ************************************************************");
        System.out.println("Fin du jeu: " + ( (etats[n].captures[0] > etats[n].
            captures[1]) ? joueurs[0] : joueurs[1] ) + " a gagne!");
        System.out.println("
            ************************************************************");
        System.out.println("\n");

}

    /**
    * Méthode permettant d'afficher les différents modes de jeu disponible et d'
        en sélectionner un.
    * @return void
    */
    public static void selectionnerModeDeJeu() {

        System.out.println("
            ************************************************************");
        System.out.println("Bienvenue dans le jeu Awale");
        System.out.println("
            ************************************************************\n");

        System.out.println("1. Faire une partie solo");
        System.out.println("2. Faire une partie multijoueurs");
        System.out.println("\n* Veuillez choisir votre mode de jeu:");

        int choix = scanner.nextInt();
        while( choix < 1 || choix > 2 ) {
            System.out.println("Mode de jeu incorrect. Veuillez-choisir votre
                mode de jeu:");
            choix = scanner.nextInt();
        }

        if( choix == 1 ) {
```

```
            joueurs[0] = "vous";
            joueurs[1] = "ordinateur";
            modeDeJeu = 1;
        } else {
            modeDeJeu = 2;
            System.out.println("* Veuillez introduire le nom du joueur 1:");
            joueurs[0] = scanner.next();
            System.out.println();

            System.out.println("* Veuillez introduire le nom du joueur 2:");
            joueurs[1] = scanner.next();
            System.out.println();
        }
    }

    /**
    * Méthode permettant d'afficher le menu du jeu et d'exécuter une action (
        jouer un coup / afficher l'historique / utiliser un joker)
    * @return void
    */
    public static void afficherMenu() {

        System.out.println("\n##### Menu de jeu:\n");

        System.out.println("1. Jouer un coup");
        System.out.println("2. Afficher l'historique de la partie et jouer un
            coup");

        // On affiche la possibilité de jouer un joker si au moins un joker est
            disponible
        boolean peutUtiliserUnJoker = false;
        if( (n%2 == 0 && (jokers[0] == true || jokers[1] == true)) || (n%2 == 1
            && (jokers[2] == true || jokers[3] == true)) ){
            System.out.println("3. Utiliser un joker et jouer un coup");
            peutUtiliserUnJoker = true;
        }

        System.out.println("\n* Que voulez-vous vous faire (1-3):");
        int choix = scanner.nextInt();

        while( (choix < 1 || choix > 3) || (choix == 3 && !peutUtiliserUnJoker) )
            {
            if(choix == 3 && !peutUtiliserUnJoker)
                System.out.println("\n* Vous n'avez plus de joker disponible. Que
                    voulez-vous faire? (1-2)");
            else
                System.out.println("\n* Le numéro de l'action ne peut être
                    compris qu'entre 1 et 3. Quel voulez-vous faire?");

            choix = scanner.nextInt();
        }

        switch(choix) {
            case 1:
                jouerUnCoup();
                break;
            case 2:
                afficherDeroulement();
                break;
            case 3:
```

```
                utiliserUnJoker();
                break;
        }

    }

    /**
    * Méthode permettant d'afficher le plateau complet (plateau / captures /
        jokers disponibles) dans la console de façon user friendly.
    * @return void
    */
    public static void afficherPlateauComplet() {

        String plateau ="";

        plateau += plateauVisuel() + "\n";

        plateau += "";

        if( (n%2 == 0 && jokers[0] == true) || (n%2 == 1 && jokers[2] == true) )
            {
            plateau += "\nVos␣jokers␣encore␣disponibles:␣␣-␣Le␣nombre␣de␣captures
                ␣pour␣chaque␣trou\n";

            if((n%2 == 0 && jokers[1] == true) || (n%2 == 1 && jokers[3] == true)
                )
                plateau += "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣Le␣classement␣des␣
                    trous␣les␣plus␣efficaces";

        } else if( (n%2 == 0 && jokers[1] == true) || (n%2 == 1 && jokers[3] ==
            true) ){
            plateau += "\nVos␣jokers␣encore␣disponibles:␣␣-␣Le␣classement␣des␣
                trous␣les␣plus␣efficaces";
        }

        System.out.println(plateau);

    }

    /**
    * Méthode permettant d'obtenir une String contenant la version user friendly
        du plateau.
    * @return String - le plateau
    */
    public static String plateauVisuel() {

        String plateau = "";

        plateau += "Votre␣adversaire␣=>␣";

        for( int i = ( (n%2 == 0) ? 6 : 0 ); i < ( (n%2 == 0) ? 12 : 6 ); i++ )
            plateau += "|" + etats[n].plateau[i] + "|␣";

        plateau += "␣␣␣␣Captures:␣" + ( (n%2 == 0) ? etats[n].captures[1] : etats
            [n].captures[0] ) + "\n";
        plateau += "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣——————————————————————————\n";
        plateau += "␣␣␣␣␣␣␣␣␣␣␣␣Vous␣=>␣";

        for( int i = ( (n%2 == 0) ? 5 : 11 ); i >= (n%2)*6; i-- )
            plateau += "|" + etats[n].plateau[i] + "|␣";
```

```
        plateau += "␣␣␣␣Captures:␣" + etats[n].captures[n%2];

        return plateau;

    }

    /**
    * Méthode permettant de demander au joueur quel trou jouer et le jouer si
        celui-ci est jouable.
    * Incrémente n et ajoute une nouvelle instance d'Etat á etats
    * @return void
    */
    public static void jouerUnCoup() {

        System.out.println("\n####␣Jouer␣un␣coup:\n");

        System.out.println(plateauVisuel());

        String coupVisuel = "";
        coupVisuel += "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣^␣␣␣^␣␣␣^␣␣␣^␣␣␣^␣␣␣^\n";
        coupVisuel += "␣␣␣Numéro␣du␣trou␣=>␣1␣␣␣2␣␣␣3␣␣␣4␣␣␣5␣␣␣6\n";

        System.out.println(coupVisuel);

        System.out.println("*␣Quel␣trou␣choisissez-vous?␣(1-6)");

        // On joue le trou uniquement si joue(int t) de la classe Etat ne renvoit
            aucune exception
        boolean estLegal = false;
        while(!estLegal) {

            int trou = scanner.nextInt();

            if(n%2 == 0)
                trou = (6 - trou) + 1;
            else
                trou = (12 - trou) + 1;

            try {
                // On double la taille d'etats si le tableau est rempli
                if(n+1 == etats.length)
                    etats = agrandirTableau(etats);

                etats[n+1] = etats[n].joue(trou - 1);

                estLegal = true;
            } catch (Exception e) {
                System.out.println("*␣Le␣trou␣sélectionné␣est␣invalide.␣Quel␣trou
                    ␣voulez-vous␣jouer?");
            }

        }

    }

    /**
    * Méthode permettant de jouer le coup le plus efficace en mode solo.
    * Incrémente n et ajoute une nouvelle instance d'Etat á etats
    * @return void
```

```
*/
public static void jouerCoupIA() {

    System.out.println("********** Manche " + (n+1) + " L'ordinateur a
        joué son coup **********\n");

    // On cherche le trou le plus efficace
    int trou = etats[n].classementDesTrous()[0];

    // On double la taille d'etats si le tableau est rempli
    if(n+1 == etats.length)
        etats = agrandirTableau(etats);

    etats[n+1] = etats[n].joue(trou);

    System.out.println("* L'ordinateur a joué son coup. Faites [ENTER] pour
        continuer");
    try {
        System.in.read();
    } catch(Exception e) { }

}

/**
* Méthode permettant de demander au joueur quel joker il veut utiliser, l'
    utiliser et ensuite jouer un coup.
* @return void
*/
public static void utiliserUnJoker() {

    System.out.println("\n#### Utiliser un joker:\n");

    // On vérifie quels jokers sont encore disponibles
    boolean capturesDispo = false;
    if( (n%2 == 0) && (jokers[0] == true) || (n%2 == 1) && (jokers[2] == true
        ) )
        capturesDispo = true;

    boolean classementDispo = false;
    if( (n%2 == 0) && (jokers[1] == true) || (n%2 == 1) && (jokers[3] == true
        ) )
        classementDispo = true;


    // Affichage du menu de sélection sur base des jokers disponibles
    System.out.println("1. Jouer un coup sans utiliser de joker");
    if( capturesDispo ) {

        System.out.println("2. Montrez-moi le nombre de captures pour chaque
            trou");
        if( classementDispo )
            System.out.println("3. Montrez-moi le classement des trous les
                plus efficaces");

    } else if( classementDispo )
        System.out.println("2. Montrez-moi le classement des trous les plus
            efficaces");

    System.out.println("\n* Quel joker voulez-vous utiliser?");
    int choix = scanner.nextInt();
```

```
        while( choix < 1 || choix > 3 || (capturesDispo && !classementDispo &&
            choix == 3) || (!capturesDispo && classementDispo && choix == 3) ||
            (!capturesDispo && !classementDispo && (choix == 2 || choix == 3)) )
                {

            System.out.println("Le joker sélectionné est invalide. Quel joker
                voulez-vous utiliser?");
            choix = scanner.nextInt();

        }

        switch(choix) {
            case 1:
                break;
            case 2:
                if(capturesDispo) afficherPredictions();
                else afficherClassement();
                break;
            case 3:
                afficherClassement();
                break;
        }

        // On joue un coup
        jouerUnCoup();

    }

    /**
    * Méthode permettant d'afficher l'historique du jeu dans la console.
    * @return void
    */
    public static void afficherDeroulement() {

        System.out.println("\n#### Historique de la partie:\n");

        for(int i=0; i <= n; i++) {
            System.out.println("**** " + i);
            System.out.println(etats[i] + "\n");
        }

        // On joue un coup
        jouerUnCoup();
    }

    /**
    * Méthode permettant d'afficher dans la console une prédiction du nombre de
        captures pour chaque trou.
    * @return void
    */
    public static void afficherPredictions() {

        System.out.println("\n#### Prédiction du nombre de captures pour chaque
            trou:\n");

        int[] piecesCapturees = etats[n].piecesCapturees();

        String predictions = "";
        predictions += plateauVisuel() + "\n";
```

```
        predictions += "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣^␣␣␣^␣␣␣^␣␣␣^␣␣␣^␣␣␣^\n";
        predictions += "Nbres␣de␣captures␣=>␣";

        for(int i = ((n%2 == 0) ? 5 : 11); i >= (n%2)*6; i--)
            predictions += ((piecesCapturees[i] == -1) ? "X" : piecesCapturees[i
                ]) + "␣␣␣";

        System.out.println(predictions);

        // Ce joker n'est plus disponible pour le joueur
        if(n%2 == 0)
            jokers[0] = false;
        else
            jokers[2] = false;
    }

    /**
    * Méthode permettant d'affiche dans la console un classement des trous les
        plus efficaces.
    * @return void
    */
    public static void afficherClassement() {

        System.out.println("\n####␣Classement␣des␣trous␣les␣plus␣efficaces␣:\n");

        int[] classementDesTrous = etats[n].classementDesTrous();

        // On initialise classementParIndice avec un tableau rempli de 99 pour
            trouver
        // les trous injouables par la suite
        int[] classementParIndice = new int[6];
        for(int i = 0; i < classementParIndice.length; i++)
            classementParIndice[i] = 99;


        // On trie le classement pour s'adapter á l'affichage des trous
        for(int i = 0; i < classementDesTrous.length; i++) {
            if(n%2 == 0)
                classementParIndice[classementDesTrous[i]] = i;
            else
            classementParIndice[classementDesTrous[i]-6] = i;
        }

        // Affichage du classement en dessous du plateau
        String classement = "";
        classement += plateauVisuel() + "\n";
        classement += "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣^␣␣␣^␣␣␣^␣␣␣^␣␣␣^␣␣␣^\n";
        classement += "␣␣␣␣␣␣␣Classement␣=>␣";

        for(int i = 5; i >= 0; i--) {
            if(classementParIndice[i] == 99)
                classement += "X␣␣␣";
            else
                classement += (classementParIndice[i]+1) + "␣␣␣";
        }

        System.out.println(classement);

        // Ce joker n'est plus disponible pour le joueur
```

```
        if(n%2 == 0)
            jokers[1] = false;
        else
            jokers[3] = false;

        }

    /**
    * Methode permettant de doubler la taille d'un tableau d'int
    * @param Etat[] tableau dont la taille doit être doublée
    * @return Etat[] tableau dont la taille a été doublée
    */
    public static Etat[] agrandirTableau(Etat[] tableau) {

        Etat[] nouveauTableau = new Etat[tableau.length * 2];

        for( int i = 0; i < tableau.length; i++ )
           nouveauTableau[i] = tableau[i];

       return nouveauTableau;

    }

}
```

Annexe C

JeuDeTests.java

```
public class TestEtat {

   public static java.util.Scanner scanner = new java.util.Scanner(System.in);

   public static void main(String args[]){

      System.out.println("\nBienvenue dans le debugger Awale");

      int choice = 1;

      do {

         System.out.println("\n-------------------------------\n");

         System.out.println("1. egrene()");
         System.out.println("2. estNourri()");
         System.out.println("3. capture(t)");
         System.out.println("4. estAffame(t)");
         System.out.println("5. estLegal(t)");
         System.out.println("6. joue(t)");
         System.out.println("7. estTerminee()");
         System.out.println("8. piecesCapture()");
         System.out.println("9. classementDesTrous()");

         System.out.println();
         System.out.println("Veuillez sélectionner la méthode á exécuter: (tapez
             0 pour sortir)");
         choice = scanner.nextInt();

         switch(choice) {
            case 1:
               testEgrene();
               break;
            case 2:
               testEstNourri();
               break;
            case 3:
               testCapture();
               break;
            case 4:
               testEstAffame();
               break;
            case 5:
               testLegal();
```

```
                break;
            case 6:
                testJoue();
                break;
            case 7:
                testEstTerminee();
                break;
            case 8:
                testPiecesCapturees();
                break;
            case 9:
                testClassementDesTrous();
                break;


        }

    } while(choice != 0);

}

    // Test egrene()*********************
public static void testEgrene() {

    System.out.println("\nRésultat des tests:");
    System.out.println("----------------------\n");


    int[] c = {0,0};

      // test 1                          //egrene a partir d'un trou non jouable
          (-1)

    System.out.println("Test no1: ");
    int[] p1 = {4,0,0,0,0,0,0,0,0,0,0,0,0};
    Etat e1 = new Etat(0,p1,c);

    try{
        int trou = e1.egrene(-1);
        System.out.println("Plateau apres egrenage: \n"+ e1);
        System.out.println("Trou renvoye: "+trou);
    }
    catch (IllegalArgumentException e){
        System.out.println("IllegalArgumentException "+ e.toString());
    }

    System.out.println();

      // test 2                         //egrene a partir d'un trou non jouable
          (12)
    System.out.println("Test no2: ");

    int[]p2 = {4,0,0,0,0,0,0,0,0,0,0,0,0};
    Etat e2 = new Etat(0,p2,c);
    try{
        int trou = e2.egrene(12);
        System.out.println("Plateau apres egrenage: \n"+ e2);
        System.out.println("Trou renvoye: "+trou);
    }
    catch (IllegalArgumentException e){
```

```
    System.out.println("IllegalArgumentException "+ e.toString());
}

System.out.println();

  // Test 3                //joue un trou vide
System.out.println("Test no3: ");

int[] p3 = {4,0,0,0,0,0,0,0,0,0,0,0};
Etat e3 = new Etat(0,p3,c);
try{
    int trou = e3.egrene(2);
    System.out.println("Plateau apres egrenage: \n"+ e3);
    System.out.println("Trou renvoye: "+trou);
}
catch (IllegalArgumentException e){
    System.out.println("IllegalArgumentException "+ e.toString());
}

System.out.println();

  // test 4                //joue un trou et egrene
System.out.println("Test no4: ");

int[] p4 = {4,0,0,0,0,0,0,0,0,0,0,0};
Etat e4 = new Etat(0,p4,c);
try{
    int trou = e4.egrene(0);
    System.out.println("Plateau apres egrenage: \n"+ e4);
    System.out.println("Trou renvoye: "+trou);
}
catch (IllegalArgumentException e){
    System.out.println("IllegalArgumentException "+ e.toString());
}

System.out.println();

  // test 5            //egrene depassant le trou 11
System.out.println("Test no5: ");

int[] p5 = {0,0,0,0,0,0,0,0,0,0,4,0};
Etat e5 = new Etat(1,p5,c);
try{
    int trou = e5.egrene(10);
    System.out.println("Plateau apres egrenage: \n"+ e5);
    System.out.println("Trou renvoye: "+trou);
}
catch (IllegalArgumentException e){
    System.out.println("IllegalArgumentException "+ e.toString());
}

System.out.println();

  // test 6            //test regle numero 6
System.out.println("Test no6: ");

int[] p6 = {0,14,0,0,0,0,0,0,0,0,0,0};
Etat e6 = new Etat(0,p6,c);
try{
    int trou = e6.egrene(1);
```

```
            System.out.println("Plateau apres egrenage: \n"+ e6);
            System.out.println("Trou renvoye: "+trou);
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }

        // test 7              //test l'egrenage a partir du 11 eme trou
        System.out.println("Test no7: ");

        int[]p7 = {0,0,0,0,0,0,0,0,0,0,0,1};
        Etat e7 = new Etat(0,p7,c);
        try{
            int trou = e7.egrene(11);
            System.out.println("Plateau apres egrenage: \n"+ e7);
            System.out.println("Trou renvoye: "+trou);
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }
    }

    // Test testEstNourri()******************
    public static void testEstNourri() {
        System.out.println("\nRésultat des tests:");
        System.out.println("----------------------\n");


        int[] c = {0,0};

          // Test 1                                  //verifie si le joueur Sud est
              nourri
        System.out.println("Test no1: ");
        int[] p1 = {4,0,0,0,0,0,0,0,0,0,0,0,0};
        Etat e1 = new Etat(0,p1,c);

        try{
            boolean etatNourri = e1.estNourri();
            System.out.println("Réponse estNourri(): " + etatNourri);
            System.out.println("Plateau apres estNourri(): \n"+ e1);
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }

        System.out.println();

          // Test 2                               //verifie si le joueur nord est
              nourri
        System.out.println("Test no2: ");
        int[] p2 = {4,0,0,0,0,0,0,0,0,0,0,0,0};
        Etat e2 = new Etat(1,p2,c);

        try{
            boolean etatNourri = e2.estNourri();
            System.out.println("Réponse estNourri(): " + etatNourri);
            System.out.println("Plateau apres estNourri(): \n"+ e2);
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
```

```
    }

    System.out.println();

      // Test 3                    //verifie si le joueur Sud est nourri
    System.out.println("Test no3: ");
    int[] p3 = {0,0,0,0,0,0,0,0,0,0,0,4};
    Etat e3 = new Etat(0,p3,c);

    try{
       boolean etatNourri = e3.estNourri();
       System.out.println("Réponse estNourri(): " + etatNourri);
       System.out.println("Plateau apres estNourri(): \n"+ e3);
    }
    catch (IllegalArgumentException e){
       System.out.println("IllegalArgumentException "+ e.toString());
    }

    System.out.println();

      // Test 4                    //verifi si le joueur Nord est nourri
    System.out.println("Test no4: ");
    int[] p4 = {0,0,0,0,0,0,0,0,0,0,0,4};
    Etat e4 = new Etat(1,p4,c);

    try{
       boolean etatNourri = e4.estNourri();
       System.out.println("Réponse estNourri(): " + etatNourri);
       System.out.println("Plateau apres estNourri(): \n"+ e4);
    }
    catch (IllegalArgumentException e){
       System.out.println("IllegalArgumentException "+ e.toString());
    }

    System.out.println();

}

// Test testCapture()**********************
public static void testCapture() {

    System.out.println("\nRésultat des tests:");
    System.out.println("-----------------------\n");


    int[] c = {0,0};

      // Test 1                          //plateau vide
    System.out.println("Test no1: ");
    int[] p1 = {0,0,0,0,0,0,0,0,0,0,0,0};
    Etat e1 = new Etat(1,p1,c);

    try{
       int capture = e1.capture(1);
       System.out.println("Plateau apres capture: \n"+ e1);
       System.out.println("Nombre capture: "+capture);
    }
    catch (IllegalArgumentException e){
       System.out.println("IllegalArgumentException "+ e.toString());
    }
```

```
    System.out.println();

    // Test 2                //joueur Nord capture le trou 9
    System.out.println("Test no2: ");
    int[] p2 = {0,0,0,0,0,0,0,0,0,2,0,0};
    Etat e2 = new Etat(0,p2,c);

    try{
       int capture = e2.capture(9);
       System.out.println("Plateau apres capture: \n"+ e2);
       System.out.println("Nombre capture: "+capture);
    }
    catch (IllegalArgumentException e){
       System.out.println("IllegalArgumentException "+ e.toString());
    }

    System.out.println();

  // Test 3                      //joueur nord affame joueur sud
    System.out.println("Test no3: ");
    int[] p3 = {3,2,3,2,3,2,2,2,2,2,3,2};;
    Etat e3 = new Etat(1,p3,c);

    try{
       int capture = e3.capture(5);
       System.out.println("Plateau apres capture: \n"+ e3);
       System.out.println("Nombre capture: "+capture);
    }
    catch (IllegalArgumentException e){
       System.out.println("IllegalArgumentException "+ e.toString());
    }

    System.out.println();



   // Test 4                //joueur nord capture a partir du trou 11. 2 de
       capture car trou precedent est de 5 voir regle 4
    System.out.println("Test no4: ");
    int[] p4 = {2,2,2,2,2,2,2,2,2,5,2,2};;
    Etat e4 = new Etat(0,p4,c);

    try{
       int capture = e4.capture(11);
       System.out.println("Plateau apres capture: \n"+ e4);
       System.out.println("Nombre capture: "+capture);
    }
    catch (IllegalArgumentException e){
       System.out.println("IllegalArgumentException "+ e.toString());
    }

    System.out.println();

     // Test 5                              //joueur nord capture a partir du
         trou 5. Ne peut capturer renvoi 0
    System.out.println("Test no5: ");
    int[] p5 = {2,2,2,2,2,2,2,2,2,5,2,2};;
    Etat e5 = new Etat(0,p5,c);
```

```
        try{
            int capture = e5.capture(5);
            System.out.println("Plateau apres capture: \n"+ e5);
            System.out.println("Nombre capture: "+capture);
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }

        System.out.println();

         // Test 6                                  //joueur sud capture a partir du
             trou 2.
        System.out.println("Test no6: ");
        int[] p6 = {2,1,2,2,2,2,2,2,2,5,5,2};;
        Etat e6 = new Etat(1,p6,c);

        try{
            int capture = e6.capture(2);
            System.out.println("Plateau apres capture: \n"+ e6);
            System.out.println("Nombre capture: "+capture);
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }

        System.out.println();

        // test 7                               //trou non jouable
        System.out.println("Test no7: ");
        int[] p7 = {4,0,0,0,0,0,0,0,0,0,0,0};
        Etat e7 = new Etat(0,p7,c);

        try{
            int capture = e7.capture(-1);
            System.out.println("Plateau apres capture: \n"+ e7);
            System.out.println("Nombre capture: "+capture);
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }

        System.out.println();

    // test 8                                 //joueur sud affame joueur nord

        System.out.println("Test no8: ");
        int[] p8 = {2,2,3,2,3,2,0,0,0,0,0,0};
        Etat e8 = new Etat(1,p8,c);

        try{
            int capture = e8.capture(5);
            System.out.println("Plateau apres capture: \n"+ e8);
            System.out.println("Nombre capture: "+capture);
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }

        System.out.println();
```

```
}

// Test testEstAffame()********************
public static void testEstAffame() {
   System.out.println("\nRésultat des tests:");
   System.out.println("----------------------\n");


   int[] c = {0,0};


     // Test 1        //test si joueur nord est affame
   System.out.println("Test no1: ");
   int[] p1 = {1,0,0,0,0,0,0,0,0,0,0,0};
   Etat e1 = new Etat(0,p1,c);

   try{
      boolean affame = e1.estAffame(1);
      System.out.println(" Plateau apres verification estAffame : \n"+ e1);
      System.out.println("Est affame: "+affame);
   }
   catch (IllegalArgumentException e){
      System.out.println("IllegalArgumentException "+ e.toString());
   }

   System.out.println();

     // Test 2               //test si joueur nord est affame
   System.out.println("Test no2: ");
   int[] p2 = {0,0,0,0,0,0,1,0,0,0,0,0};
   Etat e2 = new Etat(1,p2,c);

   try{
      boolean affame = e2.estAffame(6);
      System.out.println("Plateau apres verification estAffame : \n"+ e2);
      System.out.println("Est affame: "+affame);
   }
   catch (IllegalArgumentException e){
      System.out.println("IllegalArgumentException "+ e.toString());
   }

   System.out.println();
   // Test 3                    //test si joueur Sud est affame
   System.out.println("Test no3: ");
   int[] p3 = {0,0,0,0,0,1,0,0,0,0,0,0};
   Etat e3 = new Etat(0,p3,c);

   try{
      boolean affame = e3.estAffame(5);
      System.out.println("Plateau apres verification estAffame : \n"+ e3);
      System.out.println("Est affame: "+affame);
   }
   catch (IllegalArgumentException e){
      System.out.println("IllegalArgumentException "+ e.toString());
   }
```

```
        System.out.println();
        //Test 4              //test si joueur Sud est affame
        System.out.println("Test no4: ");
        int[] p4 = {0,0,0,0,0,0,0,0,0,0,0,1};
        Etat e4 = new Etat(1,p4,c);

        try{
            boolean affame = e4.estAffame(11);
            System.out.println("Plateau apres verification estAffame : \n"+ e4);
            System.out.println("Est affame: "+affame);
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }

        System.out.println();

         // test 5                          //trou non jouable (-1)

        System.out.println("Test no5: ");
        int[] p5 = {4,0,0,0,0,0,0,0,0,0,0,0};
        Etat e5 = new Etat(0,p5,c);

        try{
            boolean affame = e5.estAffame(-1);
            System.out.println("Plateau apres verification estAffame :  \n"+ e5);
            System.out.println("Est affame: "+affame);
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }

        System.out.println();
    }

    // Test testLegal()*********************
    public static void testLegal(){
        System.out.println("\nRésultat des tests:");
        System.out.println("-----------------------\n");


        int[] c = {0,0};


          // Test 1                  //plateau vide
        System.out.println("Test no1: ");
        int[] p1 = {0,0,0,0,0,0,0,0,0,0,0,0};
        Etat e1 = new Etat(1,p1,c);

        try{
            boolean legal = e1.estLegal(5);
            System.out.println("Plateau apres verification estLegal  : \n"+ e1);
            System.out.println("Est Legal: "+legal);
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }

        System.out.println();
```

```
  // Test 2              //joueur joue un trou qui ne nourrit pas le
      joueur sud
System.out.println("Test no2: ");
int[] p2 = {1,0,0,0,0,0,0,0,0,0,0,0};
Etat e2 = new Etat(0,p2,c);

try{
   boolean legal = e2.estLegal(1);
   System.out.println("Plateau apres verification estLegal: \n"+ e2);
   System.out.println("Est Legal: "+legal);
}
catch (IllegalArgumentException e){
   System.out.println("IllegalArgumentException "+ e.toString());
}

System.out.println();
// Test 3            //joueur Nord joue un trou qui nourrit le joeur sud
System.out.println("Test no3: ");
int[] p3 = {1,1,1,1,1,1,0,0,0,0,0,0};
Etat e3 = new Etat(0,p3,c);

try{
   boolean legal = e3.estLegal(5);
   System.out.println("Plateau apres verification estLegal: \n"+ e3);
   System.out.println("Est Legal: "+legal);
}
catch (IllegalArgumentException e){
   System.out.println("IllegalArgumentException "+ e.toString());
}

System.out.println();

// Test 4               // joueur nord est nourri, joueur sud peu jouer le
    trou.
System.out.println("Test no4: ");
int[] p4 = {1,1,1,1,1,1,1,0,0,0,0,0};
Etat e4 = new Etat(1,p4,c);

try{
   boolean legal = e4.estLegal(6);
   System.out.println("Plateau apres verification estLegal: \n"+ e4);
   System.out.println("Est Legal: "+legal);
}
catch (IllegalArgumentException e){
   System.out.println("IllegalArgumentException "+ e.toString());
}

  // Test 5          // joueur nord joue un trou non jouable
System.out.println("Test no5: ");
int[] p5 = {0,0,0,0,0,0,0,0,0,0,5,0};
Etat e5 = new Etat(0,p5,c);

try{
   boolean legal = e5.estLegal(10);
   System.out.println("Plateau: \n"+ e5);
   System.out.println("Est Legal: "+legal);
}
catch (IllegalArgumentException e){
   System.out.println("IllegalArgumentException "+ e.toString());
}
```

```
        System.out.println();

        // Test 6          // exception
        System.out.println("Test no6: ");
        int[] p6 = {1,0,0,0,0,0,0,0,0,0,0,0};
        Etat e6 = new Etat(0,p6,c);

        try{
           boolean legal = e5.estLegal(-1);
           System.out.println("Plateau apres verification estLegal: \n"+ e6);
           System.out.println("Est Legal: "+legal);
        }
        catch (IllegalArgumentException e){
           System.out.println("IllegalArgumentException "+ e.toString());
        }

        System.out.println();


    }

    // Test testJoue()***************
    public static void testJoue(){

        System.out.println("\nRésultat des tests:");
        System.out.println("----------------------\n");


        int[] c = {0,0};



          // Test 1                                    //joue un trou non jouable.
        System.out.println("Test no1: ");
        int[] p1 = {0,0,0,0,0,0,0,0,0,0,0,0};
        Etat e1 = new Etat(1,p1,c);

        try{
           Etat joue = e1.joue(5);
           System.out.println("Plateau: \n"+ e1);
           System.out.println("Plateau apres verification est joue:\n"+ joue);
        }
        catch (IllegalArgumentException e){
           System.out.println("IllegalArgumentException "+ e.toString());
        }


          // Test 2              //trou hors tableau
        System.out.println("Test no2: ");
        int[] p2 = {1,0,0,0,0,0,0,0,0,0,0,0};
        Etat e2 = new Etat(0,p2,c);

        try{
           Etat joue = e2.joue(-1);
           System.out.println("Plateau: \n"+ e2);
           System.out.println("Plateau apres verification est joue: \n"+ joue);
        }
        catch (IllegalArgumentException e){
           System.out.println("IllegalArgumentException "+ e.toString());
```

```
}

        // Test 3                //Joueur joue un trou qui ne nourrit pas
            le joueur sud
System.out.println("Test no3: ");
int[] p3 = {0,0,0,0,0,0,1,0,0,0,0,0};
Etat e3 = new Etat(6,p3,c);

try{
   Etat joue = e3.joue(1);
   System.out.println("Plateau: \n"+ e3);
   System.out.println("Plateau apres verification est joue: \n"+ joue);
}
catch (IllegalArgumentException e){
   System.out.println("IllegalArgumentException "+ e.toString());
}

        // Test 4                //Joueur joue un trou qui nourrit le
            joueur sud
System.out.println("Test no4: ");
int[] p4 = {0,0,0,0,0,1,0,0,0,0,0,0};
Etat e4 = new Etat(0,p4,c);

try{
   Etat joue = e4.joue(5);
   System.out.println("Plateau: \n"+ e4);
   System.out.println("Plateau apres verification est joue: \n"+ joue);
}
catch (IllegalArgumentException e){
   System.out.println("IllegalArgumentException "+ e.toString());
}

        // Test 5
System.out.println("Test no5: ");
int[] p5 = {0,1,0,0,0,0,0,0,0,0,1,0};
Etat e5 = new Etat(1,p5,c);

try{
   Etat joue = e5.joue(10);
   System.out.println("Plateau: \n"+ e5);
   System.out.println("Plateau apres verification est joue: \n"+ joue);
}
catch (IllegalArgumentException e){
   System.out.println("IllegalArgumentException "+ e.toString());
}
        // Test 6                //Joueur joue un trou qui nourrit le
            joueur nord
System.out.println("Test no6: ");
int[] p6 = {0,0,0,0,0,0,0,0,0,0,0,1};
Etat e6 = new Etat(1,p6,c);

try{
   Etat joue = e6.joue(11);
   System.out.println("Plateau: \n"+ e6);
   System.out.println("Plateau apres verification est joue: \n"+ joue);
}
catch (IllegalArgumentException e){
   System.out.println("IllegalArgumentException "+ e.toString());
}
```

```
}



// Test  testEstTerminee()**********************
public static void  testEstTerminee(){

   System.out.println("\nRésultat des tests:");
   System.out.println("-----------------------\n");


   int[] c = {18,18};


     // Test 1                //jeu pas termine
   System.out.println("Test no1: ");
   int[] p1 = {1,1,1,1,1,1,1,1,1,1,1,1};
   Etat e1 = new Etat(1,p1,c);

   try{
      boolean terminee = e1.estTerminee();
      System.out.println("Plateau apres verification est termine: \n"+ e1);
      System.out.println("termine: "+ terminee);
   }
   catch (IllegalArgumentException e){
      System.out.println("IllegalArgumentException "+ e.toString());
   }

     // Test 2                //jeu termine joueur sud a 25 point
   c[0] = 23;
   c[1] = 25;
   System.out.println("Test no2: ");
   int[] p2 = {0,0,0,0,0,0,0,0,0,0,0,1};
   Etat e2 = new Etat(1,p2,c);

   try{
      boolean terminee = e2.estTerminee();
      System.out.println("Plateau apres verification est termine: \n"+ e2);
      System.out.println("termine: "+ terminee);
   }
   catch (IllegalArgumentException e){
      System.out.println("IllegalArgumentException "+ e.toString());
   }

            // Test 3            //jeu termine le joueur nord ne peut pas
                nourrir le joueur Sud
   c[0] = 10;
   c[1] = 10;
   System.out.println("Test no3: ");
   int[] p3 = {5,4,3,2,1,0,0,0,0,0,0,0};
   Etat e3 = new Etat(0,p3,c);

   try{
      boolean terminee = e3.estTerminee();
      System.out.println("Plateau apres verification est termine: \n"+ e3);
      System.out.println("termine: "+ terminee);
   }
   catch (IllegalArgumentException e){
```

```
        System.out.println("IllegalArgumentException "+ e.toString());
    }
    // Test 4                    //jeu non termine le joueur Sud a au moins un
        cas qui peut le nourrir.
    c[0] = 10;
    c[1] = 9;
    System.out.println("Test no4: ");
    int[] p4 = {0,0,0,0,0,0,5,4,3,2,2,0};
    Etat e4 = new Etat(0,p4,c);

    try{
        boolean terminee = e4.estTerminee();
        System.out.println("Plateau apres verification est termine: \n"+ e4);
        System.out.println("termine: "+ terminee);
    }
    catch (IllegalArgumentException e){
        System.out.println("IllegalArgumentException "+ e.toString());
    }

      // Test 5                 //jeu termine le joueur Sud ne peut pas nourrir
          le joueur nord qui est affame
    c[0] = 23;
    c[1] = 23;
    System.out.println("Test no5: ");
    int[] p5 = {0,0,0,0,0,0,1,0,0,1,0,0};
    Etat e5 = new Etat(1,p5,c);

    try{
        boolean terminee = e5.estTerminee();
        System.out.println("Plateau apres verification est termine: \n"+ e5);
        System.out.println("termine: "+ terminee);
    }
    catch (IllegalArgumentException e){
        System.out.println("IllegalArgumentException "+ e.toString());
    }


}




// Test testPiecesCapturees()*****************
public static void testPiecesCapturees(){
    System.out.println("\nRésultat des tests:");
    System.out.println("----------------------\n");


    int[] c = {0,0};


      // Test 1                 //plateau vide
    System.out.println("Test no1: ");
    int[] p1 = {0,0,0,0,0,0,0,0,0,0,0,0};
    Etat e1 = new Etat(0,p1,c);

    try{
        int[] lesPiecesCapturees = e1.piecesCapturees();
        System.out.println("Plateau apres verification piecesCapturees(): \n"+
            e1);
```

```
    System.out.println("pieceCapturer: \n"+ afficherPlateau(
        lesPiecesCapturees));
}
catch (IllegalArgumentException e){
    System.out.println("IllegalArgumentException "+ e.toString());
}

  // Test 2            //joueur nord
System.out.println("Test no2: ");
int[] p2 = {1,1,1,1,1,1,1,1,1,1,1,1};
Etat e2 = new Etat(0,p2,c);

try{
    int[] lesPiecesCapturees = e2.piecesCapturees();
    System.out.println("Plateau apres verification piecesCapturees(): \n"+
        e2);
    System.out.println("pieceCapturer: \n"+ afficherPlateau(
        lesPiecesCapturees));
}
catch (IllegalArgumentException e){
    System.out.println("IllegalArgumentException "+ e.toString());
}

          // Test 3            //joueur nord affame sud
System.out.println("Test no3: ");
int[] p3 = {11,10,9,8,7,6,1,1,1,1,1,1};
Etat e3 = new Etat(0,p3,c);

try{
    int[] lesPiecesCapturees = e3.piecesCapturees();
    System.out.println("Plateau apres verification piecesCapturees(): \n"+
        e3);
    System.out.println("pieceCapturer: \n"+ afficherPlateau(
        lesPiecesCapturees));
}
catch (IllegalArgumentException e){
    System.out.println("IllegalArgumentException "+ e.toString());
}


          // Test 4            //joueur sud
System.out.println("Test no4: ");
int[] p4 = {0,0,0,0,0,0,0,0,0,0,0,12};
Etat e4 = new Etat(1,p4,c);

try{
    int[] lesPiecesCapturees = e4.piecesCapturees();
    System.out.println("Plateau apres verification piecesCapturees(): \n"+
        e4);
    System.out.println("pieceCapturer: \n"+ afficherPlateau(
        lesPiecesCapturees));
}
catch (IllegalArgumentException e){
    System.out.println("IllegalArgumentException "+ e.toString());
}
  // Test 5            //joueur Sud
System.out.println("Test no2: ");
int[] p5 = {1,1,1,1,1,1,1,1,1,1,1,1};
Etat e5 = new Etat(1,p5,c);
```

```
        try{
            int[] lesPiecesCapturees = e5.piecesCapturees();
            System.out.println("Plateau apres verification piecesCapturees(): \n"+
                e5);
            System.out.println("pieceCapturer: \n"+ afficherPlateau(
                lesPiecesCapturees));
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }









    }

    // Test testClassementDesTrous()******************
    public static void testClassementDesTrous(){
        System.out.println("\nRésultat des tests:");
        System.out.println("----------------------\n");


        int[] c = {0,0};

          // Test 1                //plateau vide aucune case jouable
        System.out.println("Test no1: ");
        int[] p1 = {0,0,0,0,0,0,0,0,0,0,0,0};
        Etat e1 = new Etat(1,p1,c);

        try{
            int[] leClassementDesTrous = e1.classementDesTrous();
            System.out.println("Plateau avant verification classementDesTrous(): \n"
                + e1);
            System.out.println("Plateau apres verification classementDesTrous(): \n"
                + afficherPlateau(leClassementDesTrous));
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }


          // Test 2                //joueur sud
        System.out.println("Test no2: ");
        int[] p2 = {1,1,1,1,1,1,0,0,0,0,0,1};
        Etat e2 = new Etat(1,p2,c);

        try{
            int[] leClassementDesTrous = e2.classementDesTrous();
            System.out.println("Plateau avant verification classementDesTrous(): \n"
                + e2);
            System.out.println("Plateau apres verification classementDesTrous(): \n"
                + afficherPlateau(leClassementDesTrous));
        }
        catch (IllegalArgumentException e){
            System.out.println("IllegalArgumentException "+ e.toString());
        }
```

```
        // Test 3                  //joueur sud
      System.out.println("Test no3: ");
      int[] p3 = {1,1,1,1,1,1,2,3,4,8,4,5};
      Etat e3 = new Etat(1,p3,c);

      try{
         int[] leClassementDesTrous = e3.classementDesTrous();
         System.out.println("Plateau avant verification classementDesTrous(): \n"
             + e3);
         System.out.println("Plateau apres verification classementDesTrous(): \n"
             + afficherPlateau(leClassementDesTrous));
      }
      catch (IllegalArgumentException e){
         System.out.println("IllegalArgumentException "+ e.toString());
      }

               // Test 4                  //joueur sud
      System.out.println("Test no4: ");
      int[] p4 = {11,9,7,5,3,0,1,1,1,1,1,1};
      Etat e4 = new Etat(0,p4,c);

      try{
         int[] leClassementDesTrous = e4.classementDesTrous();
         System.out.println("Plateau avant verification classementDesTrous(): \n"
             + e4);
         System.out.println("Plateau apres verification classementDesTrous(): \n"
             + afficherPlateau(leClassementDesTrous));
      }
      catch (IllegalArgumentException e){
         System.out.println("IllegalArgumentException "+ e.toString());
      }


   }

 // methode pour afficher plateau**************
   public static String afficherPlateau(int[] plateau){

      String plateauNord = ""; // Chaine qui represente le contenu du plateau
          nord
      String plateauSud = ""; // Chaine qui represente le contenu du plateau sud

      for(int i=0; i < plateau.length; i++) {
         if(i < 6)
            plateauNord += plateau[i] + ", ";
         else
            plateauSud = plateau[i] + ", " + plateauSud;
      }

      String repTextuelle = plateauNord + "\n";

      repTextuelle += plateauSud + "\n";

      return repTextuelle;

   }

}
```