R2.01 - Développement Orienté Objet

– Projet –

Le ’3 SPOT GAME’ de Edward de Bono

SOUMAH Ali et LEMARQUIS Bryan Jones 112

Table des Matières

1 Introduction

Le projet

Le jeu se déroule sur un plateau composé de neuf carrés, dont trois sont désignés par un point, appelés les "spots". Chaque joueur, dans ce jeu à deux participants, contrôle une pièce colorée, tandis qu'une pièce blanche reste neutre. Le but est d'atteindre 12 points, mais avec une particularité : si l'adversaire a obtenu au moins 6 points, alors celui qui atteint 12 points devient le perdant. Cela introduit une dynamique stratégique unique, obligeant les joueurs à décider s'ils doivent marquer des points, forcer l'adversaire à en marquer, ou éviter d'en marquer.

Le projet consiste à créer un programme permettant à deux joueurs de s'affronter dans ce jeu. Le programme doit gérer l'intégralité de la partie, afficher le plateau à chaque tour, suivre les points des joueurs et détecter automatiquement la fin de la partie, en annonçant la couleur du gagnant.

Interface utilisateur

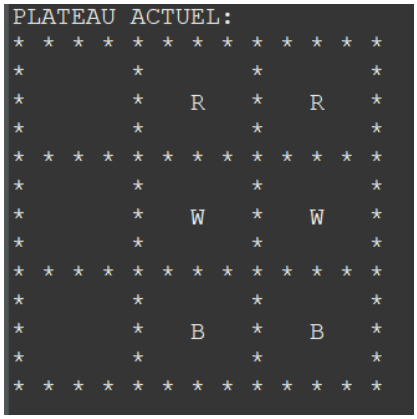


Figure 1 : Etat actuel du plateau

A chaque tour, l'état actuel du plateau est affiché (voir figure 1). Pour simplifier le programme le choix de la position d'une pièce se fait à l'aide de numéros de destinations. Un deuxième plateau avec les destinations possibles est affiché après l'état actuel du plateau (voir figure 2). Si un joueur entre une destination indisponible celui-ci sera alerté avec un message

d’erreur et devra entrer une nouvelle destination. A la fin de chaque tour les points de chaque joueur est affiché. Enfin lorsque la partie se termine, le programme affiche la couleur du gagnant.

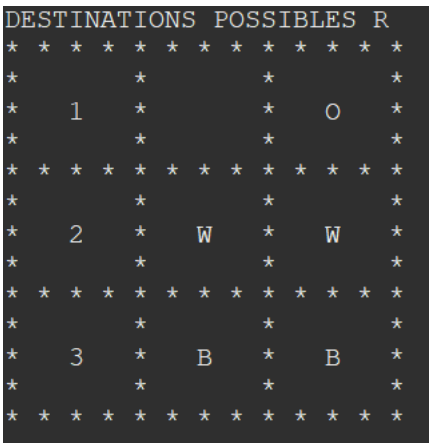
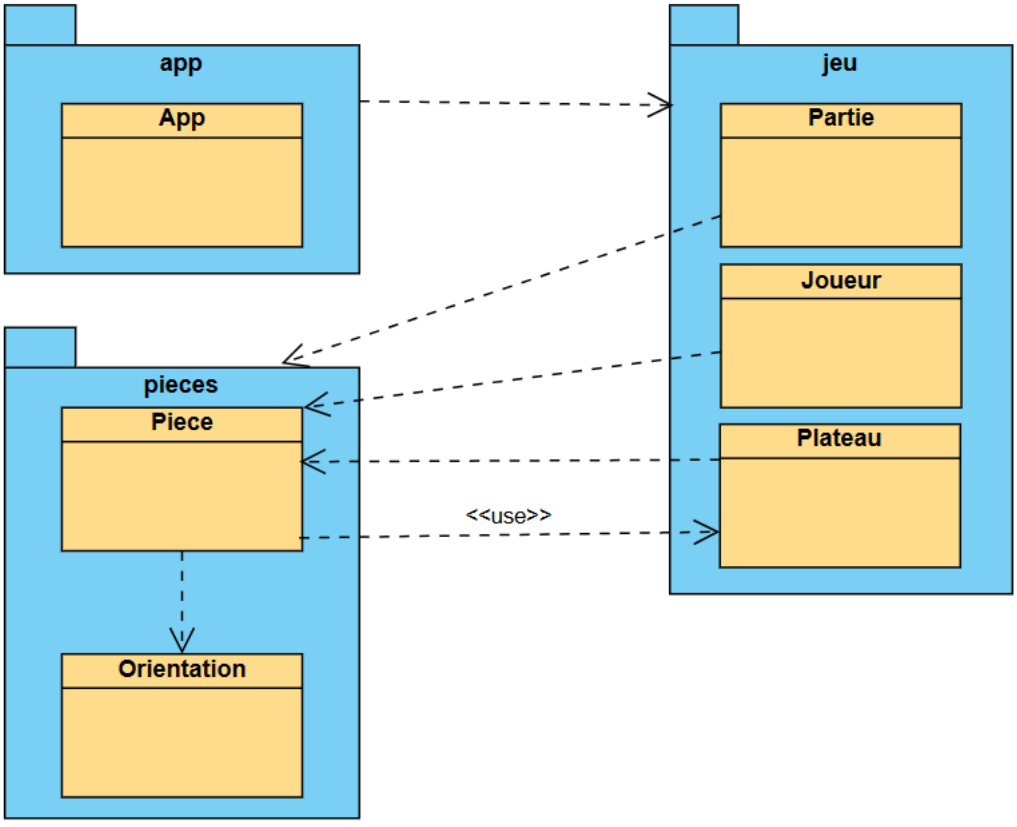


Figure 2 : Plateau avec les destinations possibles

2 Diagramme UML



3 Bilan du projet

Points positifs

- Dans l'ensemble, le code est bien structuré et bien organisé.
- La logique des méthodes est claire et facile à suivre.
- La séparation des responsabilités est bien respectée, chaque méthode faisant une tâche spécifique.

Difficultés rencontrées

La seule difficulté a été le choix de la structure du programme avant même de commencer à coder. Quelles classes utiliser ? Comment représenter le plateau ? Prioriser l'affichage ou le fonctionnement ? Ces questions nous ont permis de savoir clairement vers où nous nous dirigeons en codant.

Améliorations

- Les noms des variables pourraient être encore plus descriptifs. Par exemple, au lieu de nx et ny dans la méthode bouger de la classe Piece, on pourrait utiliser nouvellePositionX et nouvellePositionY pour plus de clarté.

4 Annexes

Tests Unitaires

Seules les méthodes publiques sont testées ci-dessous. Tous les tests s'exécutent avec succès.

Tests Piece

```java
import static org.junit.Assert.*;

import org.junit.Test;

import jeu.Plateau;
import pieces.Orientation;
import pieces.Piece;

public class PieceTest {

    @Test
    public void testOccupe() {
        Piece mapiece = new Piece('T', 0, 0, Orientation.HORIZONTAL);
        assertTrue(mapiece.occupe(0, 0));
    }

    @Test
    public void testBouger() {
        Piece mapiece = new Piece('T', 0, 1, Orientation.HORIZONTAL);
        Plateau monplateau = new Plateau(3, 3);
        monplateau.ajouter(mapiece);
        //Deplacement vers une destination horizontale
        mapiece.bouger(1, monplateau);
        assertTrue(mapiece.occupe(0, 0));
        assertTrue(mapiece.getOrientation() == Orientation.HORIZONTAL);
        //Deplacement vers une destination verticale
        mapiece.bouger(6, monplateau);
        assertTrue(mapiece.occupe(1, 2));
        assertTrue(mapiece.getOrientation() == Orientation.VERTICAL);
        //Deplacement vers une case ayant deux orientations possibles
        mapiece.bouger(3, monplateau);
        assertTrue(mapiece.occupe(1, 0));
        assertTrue(mapiece.getOrientation() == Orientation.VERTICAL);
        mapiece.bouger(6, monplateau);
        mapiece.bouger(4, monplateau);
        assertTrue(mapiece.occupe(1, 0));
        assertTrue(mapiece.getOrientation() == Orientation.HORIZONTAL);
    }

    @Test
    public void testEstDestniationValide() {
        Piece mapiece = new Piece('T', 0, 1, Orientation.HORIZONTAL);
        Plateau monplateau = new Plateau(3, 3);
```

```
        int nbDestinations = 11;
        monplateau.ajouter(mapiece);
        mapiece.printDestinations(monplateau);

        // Verification de chaque destination possible sur le plateau
        for(int i = 1; i < nbDestinations+1; ++i) {
            assertTrue(mapiece.estDestinationValide(i, monplateau));
        }

        //Verification des destinations invalides
        assertFalse(mapiece.estDestinationValide(0, monplateau));
        assertFalse(mapiece.estDestinationValide(-1, monplateau));
        assertFalse(mapiece.estDestinationValide(12, monplateau));
    }

}
```

Tests Plateau

```
import static org.junit.Assert.*;

import org.junit.Test;

import jeu.Plateau;
import pieces.Orientation;
import pieces.Piece;

public class PlateauTest {

    @Test
    public void testOccupant() {
        int xInitial = 0; int yInitial = 1;
        Plateau monplateau = new Plateau(3, 3);
        Piece mapiece = new Piece('T', xInitial, yInitial, Orientation.HORI-
ZONTAL);
        monplateau.ajouter(mapiece);

        //Verification que la piece occupe bien sa position initiale sur le
plateau
        assertTrue(monplateau.occupant(xInitial, yInitial) == mapiece);

        //Verification que les autres emplacements du plateau ne sont occupes
par aucune piece
        for(int x = 0; x < monplateau.getLargeur(); ++x) {
            for(int y = 0; y < monplateau.getHauteur(); ++y) {
                if(x != xInitial && y != yInitial)
```

```
                    assertTrue(monplateau.occupant(x, y) == null);
                }
            }
        }

}
```

Tests Partie

```
import static org.junit.Assert.*;

import org.junit.Test;

import jeu.Joueur;
import jeu.Partie;

public class PartieTest {

    @Test
    public void testEstTerminee() {
        Partie mapartie = new Partie();
        //Verification qu'une partie n'est pas terminee a son initialisation
        assertFalse(mapartie.estTerminee());

        //Verification que la partie est terminee lorsqu'un joueur a atteint
le maximum de points
        mapartie.setPointsJoueur(Joueur.MAX_POINTS, mapartie.getJoueurUn());
        mapartie.setPointsJoueur((Joueur.MAX_POINTS/2)-1, mapartie.get-
JoueurDeux());
        assertTrue(mapartie.estTerminee());

        //Verification que la partie n'est pas terminee lorsqu'un qu'aucun
joueur n'a atteint le maximum de points
        mapartie.setPointsJoueur(Joueur.MAX_POINTS-1, mapartie.getJoueu-
rUn());
        mapartie.setPointsJoueur(Joueur.MAX_POINTS-2, mapartie.getJoueur-
Deux());
        assertFalse(mapartie.estTerminee());
    }

    @Test
    public void testGagnant() {
        Partie mapartie = new Partie();
        //Verification qu'il n'y a pas de gagnant au debut de la partie
        assertTrue(mapartie.gagnant() == null);
```

```
        //Simulation d'une victoire du joueur deux (cas de victoire 1)
        mapartie.setPointsJoueur(Joueur.MAX_POINTS, mapartie.getJoueurUn());
        mapartie.setPointsJoueur((Joueur.MAX_POINTS/2)-1, mapartie.get-
JoueurDeux());
        assertTrue(mapartie.gagnant() == mapartie.getJoueurDeux());

        //Simulation d'une victoire du joueur un (cas de victoire 2)
        mapartie.setPointsJoueur(Joueur.MAX_POINTS, mapartie.getJoueurUn());
        mapartie.setPointsJoueur(Joueur.MAX_POINTS-1, mapartie.getJoueur-
Deux());
        assertTrue(mapartie.gagnant() == mapartie.getJoueurUn());

    }


}
```

Code des classes

Orientation.java

```
/**
 * Represente les orientations possibles des pieces sur un plateau de jeu.
 * Les orientations peuvent etre horizontale ou verticale.
 */
public enum Orientation {
    HORIZONTAL,
    VERTICAL
}
```

Piece.java

```
package pieces;
import java.util.ArrayList;

import jeu.Plateau;

/**
 * La classe Piece modelise une piece sur un plateau de jeu.
 */
public class Piece {
    private char couleur;
    private int x, y; // Position de la piece sur le plateau
```

```
    private Orientation orientation;

    /**
     * Constructeur de la classe Piece.
     * @param c la couleur de la piece.
     * @param x la coordonnee x de la position de la piece sur le plateau.
     * @param y la coordonnee y de la position de la piece sur le plateau.
     * @param o l'orientation de la piece.
     */
    public Piece(char c, int x, int y, Orientation o) {
        couleur = c; orientation = o;
        this.x = x; this.y = y;
    }

    public char getCouleur() {return couleur;}

    public Orientation getOrientation() {return orientation;}

    /**
     * Verifie si la piece occupe une position specifique sur le plateau.
     * @param x la coordonnee x de la position à verifier.
     * @param y la coordonnee y de la position à verifier.
     * @return vrai si la piece occupe la position specifiee, sinon faux.
     */
    public boolean occupe(int x, int y) {
        boolean res = false;

        if(this.x == x && this.y == y) res = true;
        else if(orientation == Orientation.HORIZONTAL && this.x == x && this.y
== y-1) res = true;
        else if(orientation == Orientation.VERTICAL && this.x-1 == x && this.y
== y) res = true;
        else res = false;

        return res;
    }

    /**
     * Deplace la piece vers une destination specifiee sur le plateau.
     * @param numDest le numero de la destination.
     * @param p le plateau sur lequel se deplace la piece.
     */
    public void bouger(int numDest, Plateau p) {
        ArrayList<int[]> destinations = getDestinations(p);
        int nx = destinations.get(numDest-1)[0];
        int ny = destinations.get(numDest-1)[1];

        if(aDeuxOrientations(destinations.get(numDest-1), p) && numDest < des-
tinations.size()) {
```

```
            if(destinations.get(numDest-1)[0] == destinations.get(numDest)[0]
&& destinations.get(numDest-1)[1] == destinations.get(numDest)[1]) this.orien-
tation = Orientation.VERTICAL;
            else this.orientation = Orientation.HORIZONTAL;
        }else {
             if(p.occupant(x, y) == null || (p.occupant(x, y) == this &&
(this.x != nx || this.y != ny))) {
                if(ny+1 < p.getLargeur() && (p.occupant(nx, ny+1) == null) ||
p.occupant(nx, ny+1) == this) this.orientation = Orientation.HORIZONTAL;
                else if(nx-1 >= 0 && (p.occupant(nx-1, ny) == null || p.occu-
pant(nx-1, ny) == this)) this.orientation = Orientation.VERTICAL;
            }
        }

        this.x = nx; this.y = ny;
    }

    /**
     * Verifie si une destination a deux orientations possibles.
     * @param destination les coordonnees de la destination à verifier.
     * @param p le plateau sur lequel se deplace la piece.
     * @return vrai si la destination a deux orientations possibles, sinon
faux.
     */
    private boolean aDeuxOrientations(int[] destination, Plateau p) {
        int nbOrientations = 0;
        ArrayList<int[]> destinations = getDestinations(p);
        for(int[] dest : destinations) {
            if(dest[0] == destination[0] && dest[1] == destination[1]) nbO-
rientations++;
        }
        return nbOrientations == 2;
    }

    /**
     * Obtient les destinations possibles pour la piece sur le plateau.
     * @param p le plateau sur lequel se deplace la piece.
     * @return une liste de tableau d'entiers representant les coordonnees des
destinations possibles.
     */
    private ArrayList<int[]> getDestinations(Plateau p){
        ArrayList<int[]> destinations = new ArrayList<>();

        for(int x = 0; x < p.getHauteur(); ++x) {
            for(int y = 0; y < p.getLargeur(); ++y) {
                if(p.occupant(x, y) == null || (p.occupant(x, y) == this &&
(this.x != x || this.y != y))) {
                    int[] position = {x, y};
                    if(x-1 >= 0) {
```

```
                        if((p.occupant(x-1, y) == null) || (p.occupant(x-1, y)
== this)) destinations.add(position);
                    }
                    if(y+1 < p.getLargeur()) {
                        if((p.occupant(x, y+1) == null)||(p.occupant(x, y+1)
== this)) destinations.add(position);
                    }
                }
            }
        }

        return destinations;
    }

    /**
     * Verifie si une destination est valide.
     * @param d le numero de la destination à verifier.
     * @param p le plateau sur lequel se deplace la piece.
     * @return vrai si la destination est valide, sinon faux.
     */
    public boolean estDestinationValide(int d, Plateau p) {
        ArrayList<int[]> destinations = getDestinations(p);
        if (d > destinations.size() || d <= 0) return false;
        return true;
    }

     /**
     * Obtient le numero de destination pour une position donnee.
     * @param d les coordonnees de la position à verifier.
     * @param p le plateau sur lequel se deplace la piece.
     * @return le numero de destination, ou -1 si la position n'est pas une
destination valide.
     */
    private int numDestination(int[] d, Plateau p) {
        ArrayList<int[]> destinations = getDestinations(p);
        for(int[] destination : destinations) {
            if(destination[0] == d[0] && destination[1] == d[1]) {
                return destinations.indexOf(destination)+1;
            }
        }
        return -1;
    }

    /**
     * Affiche les destinations possibles pour la piece sur le plateau.
     * @param p le plateau sur lequel se deplace la piece.
     */
    public void printDestinations(Plateau p) {
        StringBuilder res = new StringBuilder();
```

```
        res.append("* * * * * * * * * * * * *\n");
        for (int x = 0; x < p.getLargeur(); ++x) {
            res.append("*   *   *   *\n");
            for(int y = 0; y < p.getHauteur(); ++y) {
                int[] position = {x, y};
                int numDestination = numDestination(position, p);
                Piece occupant = p.occupant(x, y);

                if(numDestination != -1) {
                    if(aDeuxOrientations(position, p)) res.append("*  "+num-
Destination+"-"+(numDestination+1)+"  ");
                    else res.append("*   "+numDestination+"   ");
                }
                else if(this.x == x && this.y == y) res.append("*       ");
                else if((occupant == this || occupant == null) && y == Pla-
teau.COLONNE_SPOTS) res.append("*   O   ");
                else if(occupant == this) res.append("*       ");
                else if(occupant != null) res.append("*   "+occupant.getCou-
leur()+"   ");
                else res.append("*       ");
            }
            res.append("*");
            res.append("\n");
            res.append("*   *   *   *\n");
            res.append("* * * * * * * * * * * * *\n");
        }
        System.out.println(res);
    }

    public String toString() {return ""+couleur;}
}
```

Joueur.java

```
package jeu;
import java.util.Scanner;

import pieces.Piece;

public class Joueur {
    public final static int MAX_POINTS = 12;
    int points;
    Piece piece;

    /**
```

```
     * Constructeur de la classe Joueur.
     * @param piece la piece contrôlee par le joueur.
     */
    public Joueur(Piece piece) {
        points = 0; this.piece = piece;
    }

    /**
     * Augmente les points du joueur en fonction de la position de sa piece
sur le plateau.
     * @param p la partie en cours.
     */
    private void augmenterPoints(Partie p) {
        for(int i = 0; i < p.getPlateau().getHauteur(); ++i) {
            if(p.getPlateau().occupant(i, Plateau.COLONNE_SPOTS) == piece)
points++;
        }
    }

    /**
     * Permet au joueur de jouer son tour.
     * @param p la partie en cours.
     */
    public void jouer(Partie p){
        Scanner entree = new Scanner(System.in);

        System.out.println("PLATEAU ACTUEL:");
        System.out.println(p.getPlateau().toString());
        System.out.println("DESTINATIONS POSSIBLES "+piece.getCouleur());
        piece.printDestinations(p.getPlateau());

        System.out.println("Joueur "+piece.getCouleur()+" veuillez deplacer
votre piece");
        int destination = entree.nextInt();

        while(!piece.estDestinationValide(destination, p.getPlateau())) {
            System.out.println("DESTINATION INVALIDE ! Joueur "+piece.getCou-
leur()+" veuillez entrer une destination valide");
            destination = entree.nextInt();
        }
        piece.bouger(destination, p.getPlateau());

        System.out.println("PLATEAU ACTUEL:");
        System.out.println(p.getPlateau().toString());
        System.out.println("DESTINATIONS POSSIBLES "+p.getPieceNeutre().get-
Couleur());
        p.getPieceNeutre().printDestinations(p.getPlateau());
```

```
        System.out.println("Joueur "+piece.getCouleur()+" veuillez deplacer la
piece neutre");
        destination = entree.nextInt();

        while(!p.getPieceNeutre().estDestinationValide(destination, p.getPla-
teau())) {
            System.out.println("DESTINATION INVALIDE ! Joueur "+piece.getCou-
leur()+" veuillez entrer une destination valide pour la piece neutre");
            destination = entree.nextInt();
        }
        p.getPieceNeutre().bouger(destination, p.getPlateau());
        augmenterPoints(p);
        System.out.println("PLATEAU ACTUEL:");
        System.out.println(p.getPlateau().toString());
    }

    public String toString() {
        return piece.toString();
    }
}
```

Plateau.java

```
package jeu;
import java.util.ArrayList;

import pieces.Piece;

public class Plateau {
    public final static int COLONNE_SPOTS = 2;
    private int largeur, hauteur;
    private ArrayList<Piece> pieces;

    /**
     * Constructeur de la classe Plateau.
     * @param l la largeur du plateau.
     * @param h la hauteur du plateau.
     */
    public Plateau(int l, int h) {
        assert(l > 0 && h > 0);
        largeur = l; hauteur = h;
        pieces = new ArrayList<Piece>();
    }

    /**
     * Ajoute une piece au plateau.
```

```
     * @param p la piece à ajouter.
     */
    public void ajouter(Piece p) {pieces.add(p);}

    /**
     * Retourne la piece occupant une position specifique sur le plateau.
     * @param x la coordonnee x de la position.
     * @param y la coordonnee y de la position.
     * @return la piece occupant la position specifiee, ou null si la position
est vide.
     */
    public Piece occupant(int x, int y) {
        for(Piece p : pieces) {
            if(p.occupe(x, y)) return p;
        }
        return null;
    }

    public int getLargeur() { return largeur;}

    public int getHauteur() { return hauteur;}

    public String toString() {
        StringBuilder res = new StringBuilder();
        res.append("* * * * * * * * * * * * *\n");
        for(int x = 0; x < largeur; ++x) {
            res.append("*   *   *   *\n");
            for(int y = 0; y < hauteur; ++y) {
                Piece occupant = occupant(x, y);
                if(occupant == null && y == COLONNE_SPOTS) res.ap-
pend("*   O   ");
                else if(occupant == null) res.append("*       ");
                else res.append("*   "+occupant.getCouleur()+"   ");
            }
            res.append("*");
            res.append("\n");
            res.append("*   *   *   *\n");
            res.append("* * * * * * * * * * * * *\n");
        }
        return res.toString();
    }
}
```

Partie.java

```
package jeu;

import pieces.*;

/**
 * La classe Partie represente une partie de jeu avec deux joueurs et un pla-
teau de jeu.
 */
public class Partie {
    private Joueur joueurUn, joueurDeux;
    private Piece pieceRouge, pieceNeutre, pieceBleue;
    private Plateau plateau;

    public Partie() {
        pieceRouge = new Piece('R', 0, 1, Orientation.HORIZONTAL);
        pieceNeutre = new Piece('W', 1, 1, Orientation.HORIZONTAL);
        pieceBleue = new Piece('B', 2, 1, Orientation.HORIZONTAL);
        joueurUn = new Joueur(pieceRouge);
        joueurDeux = new Joueur(pieceBleue);
        plateau = new Plateau(3, 3);
        plateau.ajouter(pieceRouge);
        plateau.ajouter(pieceNeutre);
        plateau.ajouter(pieceBleue);
    }

    /**
     * Verifie si la partie est terminee en fonction des points des joueurs.
     * @return vrai si la partie est terminee, sinon faux.
     */
    public boolean estTerminee() {
        boolean ok = false;
        if(joueurUn.points == Joueur.MAX_POINTS && joueurDeux.points <
Joueur.MAX_POINTS/2
                ||
                joueurDeux.points == Joueur.MAX_POINTS && joueurUn.points <
Joueur.MAX_POINTS/2) {
            ok = true;
        }else if(joueurUn.points == Joueur.MAX_POINTS && joueurDeux.points >=
Joueur.MAX_POINTS/2
                ||
                joueurDeux.points == Joueur.MAX_POINTS && joueurUn.points >=
Joueur.MAX_POINTS/2) {
            ok = true;
        }
        return ok;
    }
```

```
    /**
     * Affiche le gagnant de la partie.
     */
    public Joueur gagnant() {
        Joueur gagnant = null;

        if(joueurUn.points == Joueur.MAX_POINTS && joueurDeux.points <
Joueur.MAX_POINTS/2) gagnant = joueurDeux;
        else if(joueurDeux.points == Joueur.MAX_POINTS && joueurUn.points <
Joueur.MAX_POINTS/2) gagnant = joueurUn;
        else if(joueurUn.points == Joueur.MAX_POINTS && joueurDeux.points <
joueurUn.points) gagnant = joueurUn;
        else if(joueurDeux.points == Joueur.MAX_POINTS && joueurUn.points <
joueurDeux.points) gagnant = joueurDeux;

        return gagnant;
    }

    /**
     * Affiche les points des deux joueurs.
     */
    public void afficherPoints() {
        System.out.println("Joueur R: "+joueurUn.points+" points | Joueur B:
"+joueurDeux.points+" points");
    }

    public void setPointsJoueur(int pts, Joueur j) {
        j.points = pts;
    }

    public Plateau getPlateau() {return plateau;}

    public Piece getPieceNeutre() {return pieceNeutre;}

    public Joueur getJoueurUn() {return joueurUn;}

    public Joueur getJoueurDeux() {return joueurDeux;}
}
```

App.java

```
package app;

import jeu.*;

public class App {
    public static void main(String[] args) {
        Partie partie = new Partie();
        while(!partie.estTerminee()) {
            partie.getJoueurUn().jouer(partie);
            partie.getJoueurDeux().jouer(partie);
            partie.afficherPoints();
        }
        System.out.println("Le gagnant est joueur "+partie.gagnant().toS-
tring());
    }
}
```