IUT de Paris - Rives de Seine
Université Paris Cité

Rapport du Projet R2.01

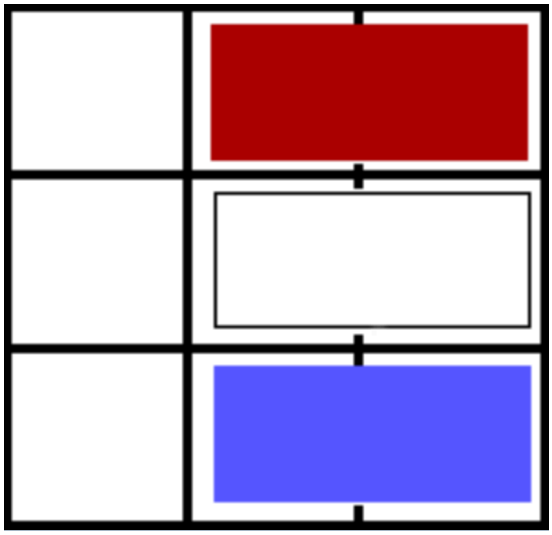
Jeu du “3 Spot Game” en Java

Amsan SUTHARSAN
Rayan MERI
Groupe 109-103

TABLE DES MATIÈRES

Présentation du projet

Explication du jeu :



Le “3 Spot Game” est un jeu de réflexion assez minimaliste créé par Edward de Bono.
Deux joueurs sont nécessaires pour commencer une partie.

Sur un plateau de 9 cases sont posées 3 pièces de couleur rouge, blanche et bleue. Chacune de ces pièces a une longueur de 2 cases.
3 “Spots” sont respectivement placés sur les 3 cases de la colonne de droite du plateau.
Les deux joueurs choisissent chacun une pièce entre la rouge et la bleue. La pièce blanche est neutre.

Les joueurs jouent chacun leur tour, ils commencent par déplacer leur pièce entre la rouge et la bleue avant de déplacer la pièce blanche. Les pièces peuvent être placées à l’horizontale ou à la verticale selon la situation, ce qui compte c’est qu’au minimum une case de la pièce soit placée dans une nouvelle case du plateau.
Pour gagner des points, il faut que le joueur place sa pièce sur les “Spots” de la colonne de droite. Pour une case de la pièce qui cache un spot, c’est un point en plus pour le joueur à la fin du tour.

Pour gagner, un joueur doit avoir un score de 12 tout en faisant en sorte que son adversaire ait un score supérieur à 6. Si ce n’est pas le cas, le joueur qui atteint 12 points a perdu.

Adaptation du jeu en Java :

Nous avons créé une Classe Java différente pour chacun des éléments qui composent le jeu. Nous avons par exemple une classe Plateau, Pièce, Joueur, Spot ou Position.
Chacune de ces classes à ses propres variables, constantes et méthodes.

Certaines classes sont plus importantes que d’autres comme notre classe Element qui permet de spécifier n’importe quel objet présent sur le plateau, comme une pièce, un Spot, une position possible etc.

Nous avons adapté le plateau de 9 cases du jeu en utilisant des caractères étoiles (*). Les pièces sont remplacées par les lettres correspondantes, (‘R’ pour Red, ‘W’ pour White et ‘B’ pour Blue). Pour les Spots, nous les avons représentés par le caractère ‘O’.

Nous laissons la possibilité aux joueurs de choisir leurs pièces en utilisant un scanner. Le premier joueur choisit Rouge ou Bleu et la pièce qui n'a pas été choisie est attribuée au deuxième joueur.

Comme le jeu l'exige, il faut que chaque joueur déplace sa pièce puis la pièce blanche. Pour se faire, nous avons mis les positions disponibles pour la pièce actuelle qui doit être déplacée par le joueur. Nous avons représenté ces différentes positions par des nombres qui sont présents dans les cases où la pièce peut être déplacée.

Tout comme le choix des pièces, nous avons créé un scanner pour permettre aux joueurs de choisir la destination qu'ils souhaitent depuis une entrée clavier, tout en faisant en sorte d'afficher un message d'erreur, qui n'interrompt pas la partie, si le joueur entre au clavier une mauvaise chose.

Pour que les joueurs s'y retrouvent, nous avons représenté le tableau de 9 cases, deux fois. Une fois à gauche pour voir où sont les pièces actuellement et une fois à droite pour voir les destinations possibles.

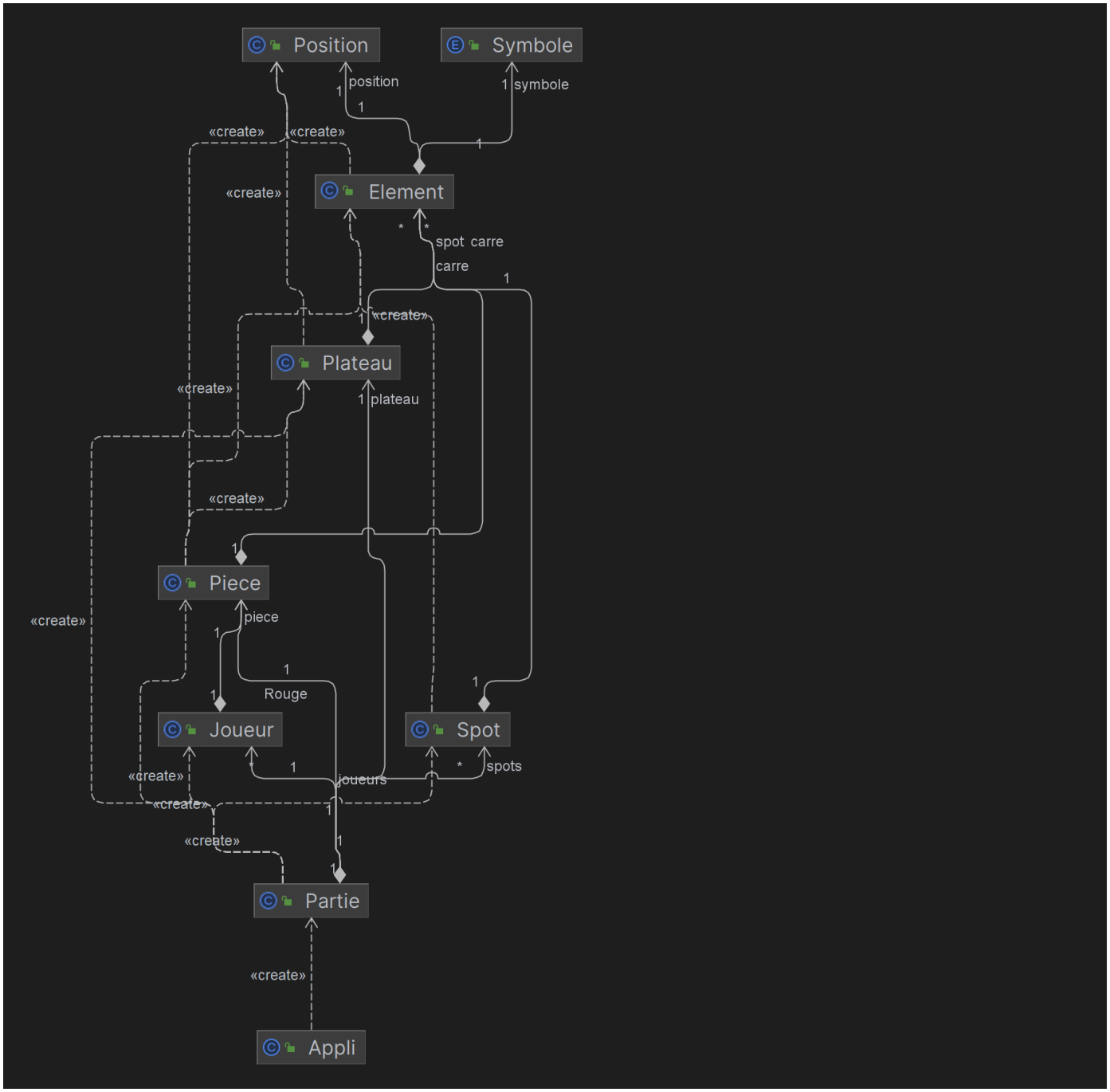
```
Tour du Joueur 1 :
Deplacement de votre pièce rouge :

Plateau :                       Destinations possibles :
* * * * * * * * * * * * *      * * * * * * * * * * * * *
*       *       *       *      *       *       *       *
*       *   R   *   R   *      *   1   *       *   0   *
*       *       *       *      *       *       *       *
* * * * * * * * * * * * *      * * * * * * * * * * * * *
*       *       *       *      *       *       *       *
*       *   W   *   W   *      *   2   *   W   *   W   *
*       *       *       *      *       *       *       *
* * * * * * * * * * * * *      * * * * * * * * * * * * *
*       *       *       *      *       *       *       *
*       *   B   *   B   *      *   3   *   B   *   B   *
*       *       *       *      *       *       *       *
* * * * * * * * * * * * *      * * * * * * * * * * * * *
```

Des méthodes ont été créées dans la Classe Joueur pour comptabiliser les points de chacun des joueurs. Lorsque la position d'un carré d'une pièce est à la même position qu'un spot, cela ajoute des points.

Le jeu se déroule en plusieurs tours. A chacun de ces tours, il y a un récapitulatif des points, ce qui permet à chacun des joueurs d'élaborer une stratégie en fonction de la situation.

DIAGRAMME UML DES CLASSES



TESTS UNITAIRES DES CLASSES

Package tests :

ElementTest.java :

```
package SpotGame.tests;


import SpotGame.structure.Element;
import SpotGame.utils.Symbole;
import org.junit.Assert;
import org.junit.Test;


public class ElementTest {
   private final Element element = new Element(0, 0,
Symbole.Vide);


   @Test
   public void test_Deplacement() {
       element.setPosition(3, 4);
       Assert.assertEquals(3, element.getLigne());
       Assert.assertEquals(4, element.getColonne());
   }
}
```

PieceTest.java :

```
package SpotGame.tests;



import SpotGame.structure.Partie;
```

```
import SpotGame.structure.Spot;

import SpotGame.utils.Position;

import org.junit.Assert;

import org.junit.Test;


import java.util.ArrayList;


public class PieceTest {

   private final Partie partie = new Partie();


   @Test

   public void test_Obstacle(){

       // La piece rouge se rencontre elle-même

       Assert.assertTrue(partie.getPieceRouge()


.Ne_rencontre_pas_de_obstacle(partie.getPieceRouge().getPosition(0
).getLigne(),partie.getPieceRouge().getPosition(0).getColonne(),pa
rtie));

       // Elle ne rencontre rien

       Assert.assertTrue(partie.getPieceRouge()

               .Ne_rencontre_pas_de_obstacle(0,0,partie));
//Nouvelle partie donc il n'y a donc rien à cette position

       // Elle rencontre un spot

       Assert.assertTrue(partie.getPieceRouge()


.Ne_rencontre_pas_de_obstacle(Partie.getSpots()[0].getSpot().getLig
ne(), Spot.COLONNE_OCCUPE,partie));

       // Elle rencontre la pièce bleue

       Assert.assertFalse(partie.getPieceRouge()


.Ne_rencontre_pas_de_obstacle(partie.getPieceBleue().getPosition(0
).getLigne(),partie.getPieceBleue().getPosition(0).getColonne(),pa
rtie));

       // Elle rencontre la pièce blanche
```

```
        Assert.assertFalse(partie.getPieceRouge()

.Ne_rencontre_pas_de_obstacle(partie.getPieceBlanche().getPosition
(0).getLigne(),partie.getPieceBlanche().getPosition(0).getColonne(
),partie));
    }

    @Test
    public void test_Nouvelle_position() {
        // Les deux carrés de la piece rouge n'ont pas bougé (on
reprend sa position actuelle).
        Assert.assertFalse(partie.getPieceRouge()
                .Est_au_moins_dans_une_nouvelle_case(new
Position[]{
                        new
Position(partie.getPieceRouge().getPosition(0).getLigne(),partie.g
etPieceRouge().getPosition(0).getColonne()),
                        new
Position(partie.getPieceRouge().getPosition(1).getLigne(),partie.g
etPieceRouge().getPosition(1).getColonne())
                }));
        // Seul le premier carré est dans une nouvelle position (on
décale la piece d'un carré à gauche (colonne-1)).
        Assert.assertTrue(partie.getPieceRouge()
                .Est_au_moins_dans_une_nouvelle_case(new
Position[]{
                        new
Position(partie.getPieceRouge().getPosition(0).getLigne(),partie.g
etPieceRouge().getPosition(0).getColonne()-1),
                        new
Position(partie.getPieceRouge().getPosition(1).getLigne(),partie.g
etPieceRouge().getPosition(1).getColonne()-1)
                }));
        // Seul le deuxième carré est dans une nouvelle position
(on pivote la piece verticalement de sorte que le deuxième carré
se trouve en dessous du premier).
        Assert.assertTrue(partie.getPieceRouge()
```

```
                .Est_au_moins_dans_une_nouvelle_case(new
Position[]{
                        new
Position(partie.getPieceRouge().getPosition(0).getLigne(),partie.g
etPieceRouge().getPosition(0).getColonne()-1),
                        new
Position(partie.getPieceRouge().getPosition(0).getLigne()-1,partie
.getPieceRouge().getPosition(0).getColonne())
                }));
        // Les deux carrés de la piece rouge ont bougé (on la
déplace sur la piece blanche).
        Assert.assertTrue(partie.getPieceRouge()
                .Est_au_moins_dans_une_nouvelle_case(new
Position[]{
                        new
Position(partie.getPieceBlanche().getPosition(0).getLigne(),partie
.getPieceBlanche().getPosition(0).getColonne()-1),
                        new
Position(partie.getPieceBlanche().getPosition(1).getLigne(),partie
.getPieceBlanche().getPosition(1).getColonne()-1)
                }));
    }
    @Test
    public void test_Placement_verticale_horizontale() {
        // Les éléments de la partie n'ont pas bougé, on va donc
vérifier les positions par défaut de la pièce rouge et bleue
        // RAPPEL : aucune des fonctions appelée ne déplace les
pièces.
        int nombres_de_destinations_attendues = 3;

        // La pièce rouge à trois destinations si c'est la première
de la partie à bouger.
        Assert.assertEquals(3,partie.getPieceRouge()
                        .Destinations_possibles(partie).size()
                );
        // La pièce bleu à trois destinations si c'est la première
de la partie à bouger.
```

```
        Assert.assertEquals(3,partie.getPieceBleue()
                .Destinations_possibles(partie).size()
        );


    }


    @Test
    public void test_Position_avec_deux_destination() {
        // Les éléments de la partie n'ont pas bougé
        // La piece bleu à trois destinations si c'est la première
de la partie à bouger,
        // dont deux a la meme position en bas à gauche, nous
allons vérifier ce cas.
        ArrayList <Position[]> destinationsPossibles =
partie.getPieceBleue().Destinations_possibles(partie); // Les
destinationsPossibles des deux carrés de la pièce, on teste le
premier carré [0] :

Assert.assertTrue(destinationsPossibles.get(1)[0].A_la_meme_positio
n_que(destinationsPossibles.get(2)[0])); // La deuxième
destination et la troisième doivent être à la même position


    }




}
```

JoueurTest.java :

```
package SpotGame.tests;


import SpotGame.structure.Joueur;
```

```java
import SpotGame.structure.Partie;

import SpotGame.structure.Spot;

import SpotGame.utils.Position;

import org.junit.Assert;

import org.junit.Test;


public class JoueurTest {

   private final Partie partie = new Partie();

   private final Joueur joueur1Test = new
Joueur(partie.getPieceRouge());

   private final Joueur joueur2Test = new
Joueur(partie.getPieceBleue());


   @Test

   public void test_Condition_de_victoire(){


       // Les deux manières de gagner du joueur 1

       joueur1Test.setScore(5);

       joueur2Test.setScore(12);

       Assert.assertTrue(joueur1Test.A_gagne_contre(joueur2Test));
// Il a moins de six points et l'adversaire a douze points

       joueur1Test.setScore(12);

       joueur2Test.setScore(6);

       Assert.assertTrue(joueur1Test.A_gagne_contre(joueur2Test));
// Il a moins de douze points et l'adversaire a six points

   }


   @Test

   public void test_Gagne_des_points(){


       Assert.assertEquals(Joueur.SCORE_INITIAL,
joueur1Test.getScore()); // Le joueur n'a pas de point au debut
```

```
        joueur1Test.getPiece().Deplacer_vers(new Position[] { // On
déplace la piece du joueur sur deux spots

                new
Position(Partie.getSpots()[0].getSpot().getLigne(),
Spot.COLONNE_OCCUPE),

                new
Position(Partie.getSpots()[1].getSpot().getLigne(),Spot.COLONNE_OCC
UPE)

        });

        joueur1Test.A_sa_piece_sur_un_spot(); // ajoute des points

        Assert.assertEquals(2, joueur1Test.getScore()); // Le joueur
doit avoir deux points




    }

}
```

PlateauTest.java :

```
package SpotGame.tests;


import SpotGame.structure.*;

import SpotGame.utils.Position;

import SpotGame.utils.Symbole;

import org.junit.Assert;

import org.junit.Test;


import java.util.ArrayList;


public class PlateauTest {

    private final Partie partie = new Partie();


    @Test
```

```
   public void test_Affichage_de_destinations_a_la_meme_position()
{

       // On prend le cas où la pièce bleue commence, car elle a
deux destinations dans le même carré à la position (2,0).

       int Ligne_destinations_a_la_meme_position = 2;

       int Colonne_destinations_a_la_meme_position = 0;


       // On prend le début de la fonction Déplacement de la
classe pièce (comme si on l'avait appelé avec la pièce bleue).

       ArrayList<Element> Carre_avec_destinations = new
ArrayList<>();

       ArrayList<Position[]> Destinations_possibles;

       Destinations_possibles =
partie.getPieceBleue().Destinations_possibles(partie);


partie.getPieceBleue().Ajouter_destinations(Destinations_possibles
, Carre_avec_destinations);

       Plateau Destination = new Plateau(Carre_avec_destinations);


Destination.Ajouter_les_autres_elements_sauf(partie.getPieceBleue(
), partie);

       String Carre_plateau_destination =
Destination.Element_a_la_positon(Ligne_destinations_a_la_meme_posi
tion,Colonne_destinations_a_la_meme_position);


       // On regarde

       Assert.assertTrue(Carre_plateau_destination.length() > 1);
// Vérifie que le String envoyé a une taille supérieure à 1, si
c'est le cas ça veut dire qu'elle affiche bien deux destinations
séparées d'un tiré.

   }


   @Test

   public void test_Piece_par_dessus_les_spots() {

       // Les éléments de la partie n'ont pas bougé (position par
défaut).
```

```
        // On regarde que c'est bien le symbole de la pièce rouge
qui s'affiche devant les spots :

Assert.assertEquals(Symbole.Rouge.toString(),partie.getPlateau()

.Element_a_la_positon(partie.getPieceRouge().getPosition(1).getLig
ne(),partie.getPieceRouge().getPosition(1).getColonne())); //On
met la position du deuxième carré de la piece rouge



Assert.assertNotEquals(Symbole.Spot.toString(),partie.getPlateau()

.Element_a_la_positon(Partie.getSpots()[0].getSpot().getLigne(),Spo
t.COLONNE_OCCUPE)); // On met la position d'un spot




    }
}
```

PartieTest.java :

```
package SpotGame.tests;


import SpotGame.structure.Partie;

import SpotGame.utils.Symbole;

import org.junit.Assert;

import org.junit.Test;

import java.io.ByteArrayInputStream;

import java.io.InputStream;



public class PartieTest {

    private final Partie partie = new Partie();
```

```
   @Test
   public void test_Creation_de_joueur(){
       // Choisie la couleur rouge de la piece à la place du
joueur
       String couleur = "R\n";
       InputStream in = new
ByteArrayInputStream(couleur.getBytes());
       System.setIn(in);

       // On lance le debut du jeu qui demande la couleur de la
piece au joueur 1
       partie.Debut_de_jeu();

Assert.assertEquals(Symbole.Rouge.toString(),partie.getJoueur(0).ge
tPiece().toString()); // Le joueur 1 doit avoir la piece rouge

Assert.assertEquals(Symbole.Bleue.toString(),partie.getJoueur(1).ge
tPiece().toString()); // Le joueur 2 doit avoir la piece bleue
       System.setIn(System.in);
   }

   @Test
   public void test_Recommencer() {
       // Saisie Oui à la place des joueurs
       String choix = "Oui\n";
       InputStream in = new
ByteArrayInputStream(choix.getBytes());
       System.setIn(in);

       Assert.assertTrue(partie.Recommencer()); // On a saisie
"Oui"

       System.setIn(System.in);
```

```
    }
}
```

CODE JAVA COMPLET

Package utils :

Position.java :

```java
package SpotGame.utils;


/** Represente une position avec le numero de la ligne et colonne
*/
public class Position {
    /** La position N ligne
     */
    private int ligne;
    /** La position N colonne
     */
    private int colonne;


    /** Crée une position
     * @param ligne Sur quelle ligne se situe l'objet?
     * @param colonne Sur quelle colonne se situe l'objet?
     */
    public Position(int ligne, int colonne) {
        this.ligne = ligne;
        this.colonne = colonne;
    }


    // Getter et Setter :
```

```
    public int getLigne() {
        return ligne;
    }
    public int getColonne() {
        return colonne;
    }

    public void setPosition(int ligne,int colonne){
        this.ligne = ligne;
        this.colonne = colonne;
    }
    public boolean A_la_meme_position_que(Position position) {
        return this.ligne == position.ligne && this.colonne ==
position.colonne;
    }
}
```

Symbole.enum :

```
package SpotGame.utils;
public enum Symbole {

    // Attributs du symbole :

    /** Les différents symboles disponibles pour le jeu
     */
    Rouge("R"), Bleue("B"), Blanche("W"), Spot("O"),
Destination("X"),Vide(" ");

    /** Le String qui permet d'avoir un symbole spécifique
     */
    private final String symbole;
```

```
    /** Crée un symbole
     * @param symbole spécifie un symbole particulier
     */
    Symbole(String symbole) {
        this.symbole = symbole;
    }

    // toString :
    public String toString() {
        return symbole;
    }

    // Getter :
    public String getName() {
        return name();
    }

}
```

Package structure :

Appli.java :

```
package SpotGame.structure;

public class Appli {

    public static void main(String[] args) {

```

```
        // Message de présentation pour les deux joueurs :

System.out.println("\n**********************************************
**********************************");
        System.out.println("3 SPOT GAME");
        System.out.println("Projet codé en JAVA par : ");
        System.out.println("Amsan SUTHARSAN (109) et Rayan MERI
(103)");
        System.out.println("Règles du jeu :
http://jeuxstrategieter.free.fr/Jeu_des_3_croix_complet.php");

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


        // Création d'une nouvelle partie
        Partie partie = new Partie();
        Piece Piece_blanche = partie.getPieceBlanche(); // Variable
qui stocke la piece blanche


        // On commence le jeu
        partie.Debut_de_jeu();


        while (!partie.Est_Fini()) { // Tant que la partie n'est
pas terminée
            for (int id = 0; id < Partie.NOMBRES_DE_JOUEUR; ++id) {
// Pour chaque joueur
                Joueur joueur = partie.getJoueur(id); // Variable
qui stocke le joueur qui joue
                Piece Piece_joueur = joueur.getPiece(); // Stocke
sa pièce


                // Début du tour :
                System.out.println("Tour du Joueur " + (id + 1) + "
: "); // On indique quel joueur doit jouer
```

```
                // Déplacement des deux pièces, grâce à la fonction
Déplacement
                Piece_joueur.Deplacement(partie); // Déplacement de
la pièce du joueur
                Piece_blanche.Deplacement(partie); // Déplacement
de la pièce blanche



                // On augmente les points :
                joueur.A_sa_piece_sur_un_spot();



                // On affiche le score des deux joueurs :
                partie.Afficher_les_scores();


                // Vérifie à la fin du tour si le joueur a rempli
la condition de victoire

partie.setFin(joueur.A_rempli_la_condition_de_victoire()); //
Actualise la variable 'Fin' de la partie


                // Si la partie est terminée :
                if (partie.Est_Fini()) {
                    System.out.println("Le joueur "+ (id + 1) + " a
atteint 12 point !");
                    System.out.println("Plateau :");
                    partie.getPlateau().Afficher();
                    partie.Le_gagnant_est(); // On affiche le
gagnant
                    if (partie.Recommencer()){ // On vérifie si les
joueurs veulent rejouer
                        partie = new Partie(); // On réinitialise
la partie
```

```
                        partie.Debut_de_jeu(); // On commence le
jeu
                    }
                    break; // Permet de sortir de la boucle
                }
            }
        }
        System.out.print("Merci d'avoir joué !"); // Message de
remerciement, qui s'affiche lorsque la partie est terminée
    }
}
```

Element.java :

```
package SpotGame.structure;

import SpotGame.utils.Position;
import SpotGame.utils.Symbole;

/** Représente un élément du plateau
*/
public class Element {

    // Attributs de l'élément :
    private final Position position;
    private final Symbole symbole;

    /** Cree un élément
     * @param ligne la ligne où il se trouve sur le plateau
     * @param colonne la colonne où il se trouve sur le plateau
     * @param symbole le symbole qui le représentera sur le plateau
     */
```

```java
    public Element(int ligne, int colonne, Symbole symbole) {
        this.position = new Position(ligne,colonne);
        this.symbole = symbole;
    }

    // ToString :
    /**
     *@return le symbole à afficher
     */
    public String toString(){return symbole.toString();}
    // Getter et Setter:
    /**
     *@return le symbole de l'élément
     */
    public Symbole getSymbole() { return symbole; }
    public int getColonne() { return position.getColonne(); }
    public int getLigne() { return position.getLigne(); }
    public Position getPosition() { return position; }

    /** Change la position
     *@param ligne valeur de la ligne
     *@param colonne valeur de la colonne
     */
    public void setPosition(int ligne, int colonne) {
position.setPosition(ligne,colonne); } // Change la position de la
piece avec celle donné


}
```

Spot.java :

```java
package SpotGame.structure;

import SpotGame.utils.Symbole;

/** Représente un spot sur le plateau
*/
public class Spot {

   // Constante :
   public static final int COLONNE_OCCUPE = 2;
   // Attribut du Spot :
   private final Element spot;

   /** Crée un spot
    * @param ligne la ligne où il se trouve sur le plateau
    * @param colonne la colonne où il se trouve sur le plateau
    */
   public Spot(int ligne, int colonne){
       spot = new Element(ligne,colonne, Symbole.Spot);
   }

   // Getter :
   public Element getSpot() {
       return spot;
   }

   // toString :
   public String toString(){
       return Symbole.Spot.toString();
   }
```

```
}
```

Piece.java :

```
package SpotGame.structure;


import SpotGame.utils.Position;
import SpotGame.utils.Symbole;


import java.util.ArrayList;
import java.util.Objects;


/** Représente une piece sur le plateau
*/
public class Piece {


   // Constantes :
   private static final int PREMIER_CARRE = 0;
   private static final int DEUXIEME_CARRE = 1;
   private static final int NOMBRES_DE_CARRES_DE_LA_PIECE = 2;
   Element[] carre = new Element[NOMBRES_DE_CARRES_DE_LA_PIECE]; //
La pièce est constituée de deux carrés (éléments).


   /** Crée une piece
    * @param ligne la ligne avec laquelle il se trouve le premier
carré sur le plateau
    * @param colonne la colonne avec laquelle il se trouve le
premier carré sur le plateau
    * @param symbole le symbole qui le représentera sur le plateau
    */
   public Piece(int ligne, int colonne, Symbole symbole) {
```

```
        carre[PREMIER_CARRE] = new Element(ligne, colonne, symbole)
;

        carre[DEUXIEME_CARRE] = new Element(ligne, colonne + 1,
symbole) ; // Le deuxième carré se situe à droite du premier
(colonne de droite)

    }


    // ToString :

    public String toString() { return
carre[PREMIER_CARRE].toString();} // Les deux carrés ont le même
symbole


    /**

     * @return le nom du symbole

     */

    public String SymboletoString() { return
carre[PREMIER_CARRE].getSymbole().getName().toLowerCase(); }


    // Getter et Setter:


    /**

     * @param i L'indice du carré

     * @return la position du carré

     */

    public Position getPosition(int i) { return
carre[i].getPosition(); }

    public Element[] getCarre() { return carre; }


    public void Deplacer_vers(Position[] destinationChoisie) {

        for (int i = 0; i < NOMBRES_DE_CARRES_DE_LA_PIECE; ++i)


this.carre[i].setPosition(destinationChoisie[i].getLigne(),
destinationChoisie[i].getColonne());

    }
```

```
    // Fonctions :


    /** S'occupe du déplacement de la pièce
     * @param partie partie dans laquelle la pièce doit se déplacer
     */
    public void Deplacement(Partie partie) {
        // Message au joueur, affiche "la " ou "votre" si on parle de la piece du joueur ou non :
        System.out.println("Déplacement de " + (Objects.equals(toString(), Symbole.Blanche.toString()) ? "la" : "votre") + " pièce " + this.SymboletoString() + " : \n");


        // On crée les variables :
        ArrayList<Element> Carre_avec_destinations = new ArrayList<>(); // Stock l'emplacement des destinations en tant qu'éléments du jeu
        ArrayList<Position[]> Destinations_possibles; // Stock les destinations possibles des deux carrés de la pièce


        // On les remplie :
        Destinations_possibles = Destinations_possibles(partie);
        Ajouter_destinations(Destinations_possibles, Carre_avec_destinations);


        Plateau plateau = new Plateau(Carre_avec_destinations); // On ajoute les destinations en premier ce qui permet aux destinations de s'afficher devant les spots
        plateau.Ajouter_les_autres_elements_sauf(this, partie);


        // Affichage au joueur :
        partie.getPlateau().Afficher_destination(plateau);


        // On demande au joueur de choisir une destination :
```

```
        System.out.print("Veuillez choisir la destination de la
pièce " + SymboletoString() + " parmi celles proposées : ");
        int Choix_joueur =
Partie.Choix_destination(Destinations_possibles.size()); //
Fonction qui scanne, évite les erreurs de saisie


        // On déplace la pièce :


this.Deplacer_vers(Destinations_possibles.get(Choix_joueur));
    }


    /** Ajoute seulement les positions du premier carré en tant
qu'élément, pour l'affichage
     * @param destinationsPossibles les destinations possibles de la
pièce
     * @param carreAvecDestinations Liste où on ajoute les
destinations
     */
    public void Ajouter_destinations(ArrayList<Position[]>
destinationsPossibles, ArrayList<Element> carreAvecDestinations) {
        for (Position[] destinationsPossible :
destinationsPossibles) {
            int ligne = destinationsPossible[0].getLigne();
            int colonne = destinationsPossible[0].getColonne();
            carreAvecDestinations.add(new Element(ligne, colonne,
Symbole.Destination));
        }
    }


    /** Crée une liste avec toutes les destinations possibles de la
pièce
     * @param partie partie dans laquelle la pièce doit se déplacer
     * @return La liste avec les destinations possibles de la pièce
dans le plateau
     */
```

```
   public ArrayList<Position[]> Destinations_possibles(Partie
partie) {

       ArrayList<Position[]> destination = new ArrayList<>();

       // La position du premier carré :

       for (int ligne = 0; ligne < Plateau.NOMBRES_DE_LIGNES;
++ligne)

           for (int colonne = 0; colonne <
Plateau.NOMBRES_DE_COLONNES; ++colonne) {

               if (this.Ne_rencontre_pas_de_obstacle(ligne,
colonne, partie)) {

                   // On regarde d'abord si il peut se placer
verticalement :

                   if (this.Peut_se_placer_verticalement(ligne,
colonne, partie)) {

                       Position[] Nouvelle_destination = {

                               new Position(ligne, colonne), // La
position du premier carré

                               new Position(ligne - 1, colonne)};
// La position du deuxième carré, ligne - 1, car il se place en
haut du premier carré

                       if
(this.Est_au_moins_dans_une_nouvelle_case(Nouvelle_destination)) {

                           destination.add(Nouvelle_destination);

                       }

                   }

                   // Puis horizontalement, car cela permet
d'afficher la destination verticale puis horizontale ( ex : 1-2 )

                   if (this.Peut_se_placer_horizontalement(ligne,
colonne, partie)) {

                       Position[] Nouvelle_destination = {

                               new Position(ligne, colonne), // La
position du premier carré

                               new Position(ligne, colonne + 1)};
// La position du deuxième carré, colonne + 1, car il se place à
droite du premier carré

                       if
(this.Est_au_moins_dans_une_nouvelle_case(Nouvelle_destination)) {
```

```
                            destination.add(Nouvelle_destination);
                        }
                    }
                }
            }
        return destination;
    }

    /** Regarde à partir de la position du premier carré si le
deuxième carré peut se placer verticalement
     * @param ligne La ligne du premier carré de la pièce
     * @param colonne La colonne du premier carré de la pièce
     * @param partie partie dans laquelle la pièce doit se déplacer
     * @return true si le deuxième carré de la pièce peut se placer
en haut du premier carré
     */
    private boolean Peut_se_placer_verticalement(int ligne, int
colonne, Partie partie) {
        // Le deuxième carré doit être dans le plateau et pas sur
un obstacle
        return this.Ne_rencontre_pas_de_obstacle((ligne - 1),
colonne, partie) && (ligne - 1) >= 0; // Ligne - 1 = carré en haut
    }

    /** Regarde à partir de la position du premier carré si le
deuxième carré peut se placer horizontalement
     * @param ligne La ligne du premier carré de la pièce
     * @param colonne La colonne du premier carré de la pièce
     * @param partie partie dans laquelle la pièce doit se déplacer
     * @return true si le deuxième carré de la piece peut se placer à
droite du premier carré
     */
    private boolean Peut_se_placer_horizontalement(int ligne, int
colonne, Partie partie) {
```

```
        // Le deuxième carré doit être dans le plateau et pas sur
un obstacle
        return this.Ne_rencontre_pas_de_obstacle(ligne, (colonne +
1), partie) && (colonne + 1) < Plateau.NOMBRES_DE_COLONNES;//
Colonne + 1 = carré de droite
    }


    /** Regarde si la position donnée ne correspond pas à la position
actuelle de la pièce
     * @param Nouvelle_destination position à comparer
     * @return true s'il a au moins un carré à une position
différente
     */
    public boolean Est_au_moins_dans_une_nouvelle_case(Position[]
Nouvelle_destination) {
        for (int i = 0; i < NOMBRES_DE_CARRES_DE_LA_PIECE; ++i) {
            if
(!this.getPosition(i).A_la_meme_position_que(Nouvelle_destination[
i]))
                return true;
        }
        return false;
    }


    /** Regarde à la position donnée s'il n'y à pas déjà d'autres
pièces que lui-même
     * @param partie partie, on regarde les autres éléments
     * @return true si à la position donnée, il n'y à rien, un spot
ou la pièce elle-même
     */
    public boolean Ne_rencontre_pas_de_obstacle(int ligne, int
colonne, Partie partie) {
        String Element_rencontrer =
partie.getPlateau().Element_a_la_positon(ligne, colonne); //On
stocke l'élément rencontré
```

```
        return Objects.equals(Element_rencontrer,
Symbole.Vide.toString()) || Objects.equals(Element_rencontrer,
this.toString()) || Objects.equals(Element_rencontrer,
Symbole.Spot.toString());
    }
}
```

Joueur.java :

```
package SpotGame.structure;

/** Représente un joueur de la partie
*/
public class Joueur {

    // Constantes :
    public static final int SCORE_INITIAL = 0;
    private static final int SCORE_MINIMAL_REQUIS = 6;
    private static final int SCORE_VICTOIRE = 12;

    // Attributs du joueur :
    private final Piece piece;
    private int score;

    /** Crée un joueur
     * @param piece la pièce qu'il possède
     */
    public Joueur(Piece piece) {
        this.piece = piece;
        score = SCORE_INITIAL;
    }
```

```java
    // ToString :
    /**
     *@return le score du joueur
     */
    public String toString(){ return Integer.toString(score); }

    // Getter et Setter:
    public Piece getPiece() { return piece; }
    public int getScore() {
        return score;
    }
    public void setScore(int score) {
        this.score = score;
    }

    // Fonctions :

    /** Ajoute un point pour chaque carré de la pièce du joueur qui se trouve sur un spot
     */
    public void A_sa_piece_sur_un_spot() {
        for (Element carre : piece.getCarre()) {
            if (carre.getColonne() == Spot.COLONNE_OCCUPE) { // Si un carré se trouve dans la meme colonne que les spots
                ++score; // On ajoute un point
            }
        }

    }

    /** Vérifie si le joueur a rempli la condition de victoire
     * @return true si le joueur a un score de 12 point ou plus
```

```
     */
    public boolean A_rempli_la_condition_de_victoire() {
        return score >= SCORE_VICTOIRE;
    }

    /** Vérifie que le joueur a atteint le pallier des 6 points
     * @return true si le joueur a au moins 6 points
     */
    public boolean A_au_moins_six_points() {
        return score >= SCORE_MINIMAL_REQUIS;
    }

    /** Vérifie si le joueur appelé gagne contre le joueur en
paramètre
     * @return true si il gagne
     */
    public boolean A_gagne_contre(Joueur adversaire) {
        // le joueur peut gagner de DEUX manières :
        // 1. Il a 12 point ou plus et l'adversaire en a au moins 6
        // 2. Il a strictement moins de 6 points lorsque le joueur
adverse atteint 12 points
        return this.A_rempli_la_condition_de_victoire() &&
adversaire.A_au_moins_six_points() ||
!this.A_au_moins_six_points();
    }
}
```

Plateau.java :

```
package SpotGame.structure;


import SpotGame.utils.Position;
```

```java
import SpotGame.utils.Symbole;


import java.util.ArrayList;
import java.util.Arrays;
import java.util.Objects;


/** Représente le plateau de la partie
*/
public class Plateau {


   // Constantes :
   public static final int NOMBRES_DE_LIGNES = 3;
   public static final int NOMBRES_DE_COLONNES = 3;


   // Attributs du plateau :


   /** Représente chaque élément du plateau, (les carrés vides ne sont pas comptés)
    */
   ArrayList<Element> carre = new ArrayList<>();


   /** Cela permet d'afficher des valeurs aux destinations
    */
   private int Compteur_destination = 1;


   /** Crée un nouveau plateau
    * @param partie les éléments de la partie sont ajoutés au plateau vide
    */
   public Plateau(Partie partie){
       // Les pièces de la partie sont ajoutés en premier :
       Placer(partie.getPieceRouge());
```

```
        Placer(partie.getPieceBlanche());
        Placer(partie.getPieceBleue());
        // Les spots après ce qui permet aux pièces de s'afficher sur les spots
        Placer_spots();

    }

    /** Crée un nouveau plateau
     * @param destinations les destinations données sont ajoutées au plateau vide
     */
    public Plateau(ArrayList<Element> destinations){
        carre.addAll(destinations);
    }

    // Placer les éléments dans le plateau :
    private void Placer_spots() {
        for (Spot spot : Partie.getSpots())
            carre.add(spot.getSpot());
    }

    private void Placer(Piece piece) {
        carre.addAll(Arrays.asList(piece.getCarre()));
    }

    // Affichage du plateau

    /** Affiche le plateau de neuf carrés avec ses éléments
     */
    public void Afficher() {
        String toString = this.toString();
```

```
        System.out.println(toString);
    }


    // Création du quadrillage en étoiles (*)


    /**Création du plateau en étoiles (*) avec les éléments du
plateau
     * @return Le plateau avec ses éléments
     */
    public String toString() {
        StringBuilder sb = new StringBuilder();


        sb.append("* * * * * * * * * * * * * *\n"); // On ajoute le
dessus
        for (int ligne = 0; ligne < NOMBRES_DE_LIGNES; ligne++) {
            sb.append("*       *       *       *\n"); // On ajoute
les étoiles entre le dessus de la ligne (dessus des 3 carrés) et
les éléments
            sb.append("*");
            for (int colonne = 0; colonne < NOMBRES_DE_COLONNES;
colonne++) {
                    sb.append("   ");
                    sb.append(Element_a_la_positon(ligne,colonne));
// On ajoute tous les élements de la ligne
                    sb.append("   *");
            }
            // Ajoute à chaque fois le bas de la ligne
            sb.append("\n");
            sb.append("*       *       *       *\n");
            sb.append("* * * * * * * * * * * * * *\n");
        }
        return sb.toString();
    }
```

```
   /** Affiche deux plateaux côte à côte, le premier avec ses
éléments et le deuxième avec les destinations

    * @param Destination le deuxième plateau à afficher, il doit
contenir les destinations

    */

   public void Afficher_destination(Plateau Destination) {

       String toString = this.toString(Destination);

       System.out.println(toString);

   }


   /**Création des deux plateaux en étoiles (*) avec les éléments du
plateau étant que premier tableau et le deuxième avec les
destinations

    * @return Les deux plateaux côte à côte crées

    */

   private String toString(Plateau Destination) { // Cette
fonction reprend le fonctionnement de toString

       StringBuilder sb = new StringBuilder();

       sb.append("Plateau :                              Destinations
possibles : \n"); // Affiche aux joueurs à quoi correspond chaque
plateau

       sb.append("* * * * * * * * * * * * *          * * * * * * *
* * * * * *\n"); // Le dessus des plateaux (un espace de 10 entre
les plateaux)

       for (int ligne = 0; ligne < NOMBRES_DE_LIGNES; ligne++) {

           sb.append("*       *       *       *           *       *
*       *\n");

           sb.append("*");

           // On ajoute d'abord tous les élements de la ligne du
plateau de la partie :

           for (int colonne = 0; colonne < NOMBRES_DE_COLONNES;
colonne++) {

               sb.append("   ");

               sb.append(Element_a_la_positon(ligne,colonne));

               sb.append("   *");
```

```
            }
            sb.append("           *");
            // Puis ajoute tous les élements de la ligne du plateau
avec les destinations :
            for (int colonne = 0; colonne < NOMBRES_DE_COLONNES;
colonne++) {
                sb.append(" ");
                String Carre_plateau_destination =
Destination.Element_a_la_positon(ligne,colonne); // L'intérieur du
carré
                if (Carre_plateau_destination.length() > 1){
                    sb.append(Carre_plateau_destination); // Pas
d'espaces car l'intérieur du carré contient deux destinations
                }
                else {
                    sb.append("  "); // Espaces qui centrent la
destination dans le carré
                    sb.append(Carre_plateau_destination);
                    sb.append("  ");
                }
                sb.append(" *");
            }
            // Ajoute à chaque fois le bas de la ligne pour chaque
plateau
            sb.append("\n");
            sb.append("*       *       *       *           *       *
*       *\n");
            sb.append("* * * * * * * * * * * * *           * * * * *
* * * * * * * * *\n");
        }
        return sb.toString();
    }

    /** Regarde s'il y a deux destinations à la même position donnée
```

```
    * @return true si il y en a deux
    */
   private boolean A_la_meme_position(int ligne, int colonne) {
       int nb_destination_trouve = 0; // Compteur
       for (Element element : carre)
           // S'il y a un élément à la position donnée, et qu'il
s'agit d'une destination
           if (element.getPosition().A_la_meme_position_que(new
Position(ligne,colonne)) && element.getSymbole() ==
Symbole.Destination){
               ++nb_destination_trouve; // On augmente le compteur
               if (nb_destination_trouve == 2) return true; //
Quand il y'a deux destinations
           }
           else if (nb_destination_trouve == 1) // S'il n'y a
qu'une destination (les destinations avec la même position sont
placées à côté dans l'ArrayList carré)
               return false ; // Il y'a une destination
       return false; // Il n'y a pas de destination
   }

   /** La fonction transforme en String l'élément à la position
donnée, élément ou destination(s) (ex : 1-2).
    * @return le string qui sera à l'intérieur du carré à la
position donnée, un espace est renvoyé si le carré est vide
    */
   public String Element_a_la_positon(int ligne, int colonne) {
       for (Element element : carre)
           if (element.getPosition().A_la_meme_position_que(new
Position(ligne,colonne))) { // S'il y a un élément à la position
donnée
               if (element.getSymbole() == Symbole.Destination) {
// On regarde si c'est une destination
                   if (A_la_meme_position(ligne,colonne))// S'il y
en a deux
```

```
                        return Compteur_destination++ +" - "+
Compteur_destination++; // On les renvoie côte à côte séparées
d'un tiré (-)
                    else return
Integer.toString(Compteur_destination++); // Il y'a qu'une
destination à la même position
                }
                else { // Ce n'est pas une destination
                    return element.toString();
                }
            }
        return Symbole.Vide.toString(); // Il n'y a pas d'éléments
à la position donnée
    }


    /** Permets d'afficher aux joueurs les destinations d'une pièce
sans la pièce dans le plateau
     * @param Piece_a_exclure la pièce à ne pas afficher
     * @param partie celle en cours
     */
    public void Ajouter_les_autres_elements_sauf(Piece
Piece_a_exclure,Partie partie) {
        Piece[] pieces =
{partie.getPieceRouge(),partie.getPieceBlanche(),partie.getPieceBl
eue()};


        for (Piece piece : pieces){
            if (!Objects.equals(piece.toString(),
Piece_a_exclure.toString()))
                Placer(piece);
        }
        Placer_spots();
    }
}
```

Partie.java :

```java
package SpotGame.structure;


import SpotGame.utils.Symbole;


import java.util.Objects;
import java.util.Scanner;


/** Représente une partie
*/
public class Partie {


   // Constante :
   public static final int NOMBRES_DE_JOUEUR = 2;


   // Attributs de la partie :
   private final Plateau plateau;
   private final Joueur[] joueurs = new Joueur[NOMBRES_DE_JOUEUR];
   private final Piece Rouge;
   private final Piece Blanche;
   private final Piece Bleue;
   private static final Spot[] spots = new Spot[]{ // Les spots
présents dans toutes les parties et qui ne sont jamais modifiés
(static)
           new Spot(0, Spot.COLONNE_OCCUPE),
           new Spot(1, Spot.COLONNE_OCCUPE),
           new Spot(2, Spot.COLONNE_OCCUPE)
   };
   private boolean Fin; // Boolean qui indique si la partie est
terminée ou non
```

```java
    /** Crée une nouvelle partie
     */
    public Partie() {
        Fin = false;
        // La position initiale des 3 pièces :
        Rouge = new Piece(0, 1, Symbole.Rouge);
        Blanche = new Piece(1, 1, Symbole.Blanche);
        Bleue = new Piece(2, 1, Symbole.Bleue);

        // On initialise le plateau, grâce aux éléments de la partie :
        plateau = new Plateau(this);

        // Les joueurs sont initialisés en dehors du constructeur, car ils doivent choisir leur pièce
    }

    // Getter et Setter :
    public Joueur getJoueur(int i) { return joueurs[i]; } // Renvoie le joueur i demandé
    public static Spot[] getSpots() { return spots; }
    public Piece getPieceRouge() { return Rouge; }
    public Piece getPieceBlanche() { return Blanche; }
    public Piece getPieceBleue() { return Bleue; }
    public Plateau getPlateau() { return plateau; }
    public void setFin(boolean Fin) { this.Fin = Fin; } // Modifie le statut de la partie
    public boolean Est_Fini() { return Fin; } // Renvoie le statut de la partie

    // Fonctions :

    /** Commence le jeu, avec une interface pour les joueurs
```

```
    */
   public void Debut_de_jeu() {
       // On annonce le debut :
       System.out.println("Debut du jeu : ");
       System.out.println("Bonjour Joueur 1 et Joueur 2 !");

       plateau.Afficher(); // On affiche le plateau pour permettre
aux joueurs de le visualiser
       System.out.print("Joueur 1 choisissez la couleur de votre
pièce, Tapez 'R' ou 'B' : ");

       String Couleur_choisie = Dilemme(Rouge.toString(),
Bleue.toString()); // Le choix du joueur

       // Attribution des couleurs selon le choix du Joueur 1 :
       if (Objects.equals(Couleur_choisie, Rouge.toString())) {
// Si le joueur 1 prend la piece rouge
           System.out.println("Joueur 2 la couleur de votre pièce
sera donc " + Bleue.SymboletoString() + " !");
           joueurs[0] = new Joueur(Rouge);
           joueurs[1] = new Joueur(Bleue);
       } else {
           System.out.println("Joueur 2 la couleur de votre pièce
sera donc " + Rouge.SymboletoString() + " !");
           joueurs[0] = new Joueur(Bleue);
           joueurs[1] = new Joueur(Rouge);
       }
       System.out.println("Que la partie commence !\n");

System.out.println("**********************************************
*********************************");

   }
```

```
   public void Afficher_les_scores() {

System.out.println("\n**********************************************
**********************************");
       for (int id = 0; id < NOMBRES_DE_JOUEUR; ++id) { // Pour
chaque joueur
           System.out.println("Joueur " + (id + 1) + " vous avez "
+ joueurs[id].toString() + " points"); // On affiche son nombre de
points
       }

System.out.println("**********************************************
**********************************\n");     }


   /**Demande à un joueur de faire un choix, évite les erreurs de
saisie
    * @param DESTINATION_MAX Le nombre de destinations max possibles
    * @return Le choix du joueur
    */
   public static int Choix_destination(int DESTINATION_MAX) {
       Scanner scanner = new Scanner(System.in);
       int Choix_joueur;
       do {
           while (!scanner.hasNextInt()) { // Tant que la valeur
entrée n'est pas un nombre entier
               // Message d'erreur :
               System.out.println("\nEntrée non valide, veuillez
entrer un nombre entier");
               System.out.print("Veuillez entrer une destination
correcte : ");
               scanner.next(); // On vide le scanner
           }
           Choix_joueur = scanner.nextInt() - 1; // -1, car
utilisé dans une liste, qui commence à 0 contrairement à
l'affichage qui commence à 1
```

```
            if (!(DESTINATION_MAX > Choix_joueur && Choix_joueur >= 0)) { // Vérifie si la valeur stockée est entre 1 et le nombre de destinations max
                // Message d'erreur :
                System.out.println("\nCette destination n'existe pas, recommencez");
                System.out.print("Veuillez entrer une destination correcte : ");
            }
        } while (!(DESTINATION_MAX > Choix_joueur && Choix_joueur >= 0)); // Tant que la valeur entrée ne correspond pas à une destination
        return Choix_joueur;
    }

    /**Demande aux joueurs de faire un choix parmi 2 options, évite les erreurs de saisie
     * @param Choix_1 Le premier choix
     * @param Choix_2 Le deuxième choix
     * @return Le choix du joueur tout en majuscule
     */
    public static String Dilemme(String Choix_1, String Choix_2) {
        Scanner scanner = new Scanner(System.in);
        String Choix_joueur;
        do {
            while (scanner.hasNextInt()) { // Tant que la valeur saisie est un entier naturel
                // Message d'erreur :
                System.out.print("\nEntrée non valide, veuillez entrer '" + Choix_1 + "' ou '" + Choix_2 + "': ");
                scanner.next(); // On vide le scanner
            }
            Choix_joueur = scanner.next().toUpperCase(); // On stocke la valeur saisie, on la met en majuscule
```

```
            if (!Choix_joueur.equals(Choix_1.toUpperCase()) &&
!Choix_joueur.equals(Choix_2.toUpperCase())) { // Vérifie que la
valeur saisie fait partie des 2 options

                // Message d'erreur :

                System.out.print("\nEntrée non valide, veuillez
entrer '" + Choix_1 + "' ou '" + Choix_2 + "': ");

            }

        } while (!(Choix_joueur.equals(Choix_1.toUpperCase()) ||
Choix_joueur.equals(Choix_2.toUpperCase())));  // Tant que la
valeur saisie ne fait pas partie des 2 options

        return Choix_joueur;


    }


    /** Fonction qui annonce le gagnant

     */

    public void Le_gagnant_est() {


        if (joueurs[0].A_gagne_contre(joueurs[1])) { // Si le
joueur 1 gagne contre le joueur 2

            System.out.println("Joueur 1 vous avez gagné la partie
avec votre pièce "+joueurs[0].getPiece().SymboletoString()+"!");

            System.out.println("Joueur 2 vous gagnerez la prochaine
fois ^^");

        } else { // Sinon

            System.out.println("Joueur 2 vous avez gagné la partie
avec votre pièce "+joueurs[1].getPiece().SymboletoString()+"!");

            System.out.println("Joueur 1 vous gagnerez la prochaine
fois ^^");

        }

    }


    /** Demande aux joueurs s'ils veulent recommencer

     * @return true si oui

     */
```

```
   public boolean Recommencer() {


        System.out.print("Voulez-vous recommencer ? Tapez \"Oui\"
ou \"Non\": ");
        return Objects.equals(Dilemme("Oui", "Non"), "OUI"); // On
renvoie leur choix
   }
}
```

BILAN

Ce projet demeure beaucoup plus simple que le précédent. En effet, nous avons chacun eu beaucoup plus de mal pour le jeu du LEXICON en C++, dans un premier temps parce que le projet en lui même est plus simple mais le langage Java l'est également par rapport au C++.

Nous trouvons tous les deux que le langage Java est beaucoup plus facile à maîtriser sachant qu'il partage de nombreuses choses avec le C++ comme la syntaxe des opérandes tout en supprimant les choses les plus difficiles comme la gestion de données manuelles avec les pointeurs ou l'allocation dynamique par exemple.

Ce que la réalisation du projet nous a apporté :

La réalisation du projet nous a bien évidemment permis de nous améliorer en Java, langage que nous avons, tous les deux, jamais utilisé auparavant.
Nous sommes donc maintenant beaucoup plus expérimentés qu'avant concernant la programmation orientée objet.

La réalisation de ce projet nous a également aidé à améliorer notre capacité à mener à bien un projet en équipe. En effet, nous devions nous coordonner pour avancer ensemble sur le même projet, nous avons donc fourni les efforts nécessaires dans la communication et l'esprit d'équipe pour y arriver.

Les problèmes rencontrés tout au long de la conception du projet:

Nous avons pris du temps avant de trouver une structure qui nous convenait vraiment. Nous l'avons donc modifié plusieurs fois, ce qui nous a fait perdre pas mal de temps.

L'affichage du plateau n'étant pas fourni avec le sujet, nous avons donc dû le faire nous même, ce qui n'a pas été une mince affaire.

La spécificité du jeu qui a été la plus difficile à intégrer dans le projet pour nous, est la capacité de certaines pièces d'être placées soit horizontalement ou verticalement dans une

même position selon la situation. Couplé à l'affichage qui change quand cette situation se produit, c'est sans doute cette partie où on a eu le plus de mal.

Finalement, on utilisait chacun une version de Windows différente (Windows 10 et Windows 11) ce qui rendait le partage de notre projet sur discord plus difficile, car le PC sous Windows 10 le reconnaissait comme un virus suite à son téléchargement.