Rapport

HIEGEL Oleg Groupe 105

– Projet –
‘Crazy Circus’, un jeu de Dominique Ehrhard

Table des matières:

Rapport:

- Présentation de l'application (p.3)
- Diagramme UML (p.4)
- Organisation des tests (p.4-8)
- Bilan de projet (p.9)
- Code complet du projet (p.10-24)
    - Class Podium
    - Class Joueur
    - Class Carte
    - Class Main

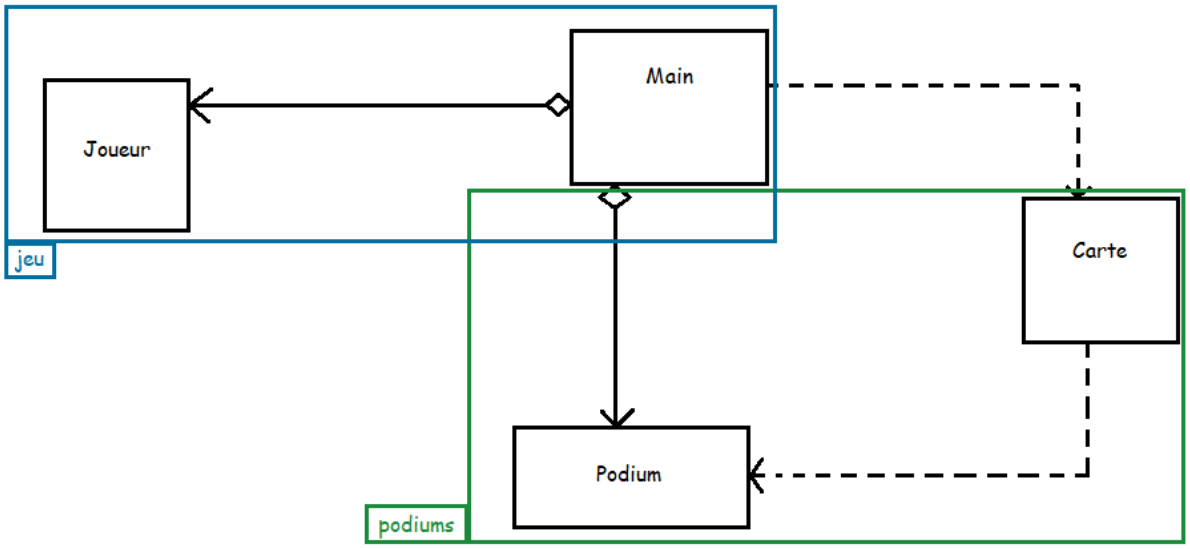
Introduction du projet

Le but de l'application est de recréer le jeu de plateau Crazy Circus où trois animaux : un éléphant, un ours et un lion sont empilés sur un podium.
Les joueurs peuvent tirer une carte montrant une séquence d'empilement d'animaux parmi 24. Le but du jeu est que le joueur donne une chaîne de commandes qui déplace les animaux d'une situation initiale à une situation montrée sur une carte tirée.
Lorsqu'une carte objectif est tirée, elle est défaussée et ne réapparaîtra pas dans la pioche.
Le jeu se termine lorsque les 24 cartes ont été tirées et résolues.
Les joueurs jouent un par un, et peuvent choisir parmi 5 commandes distinctes :

- KI - L'animal se trouvant en haut de la pile du podium bleu saute pour rejoindre le sommet de la pile du podium rouge.
- LO - L'animal se trouvant en haut de la pile du podium rouge saute pour rejoindre le sommet de la pile du podium bleu.
- SO – Les deux animaux se trouvant au sommet des piles des deux podiums échangent leur place.
- NI – L'animal se trouvant en bas de la pile du podium bleu monte et se place en haut de la pile de ce même podium.
- MA – L'animal se trouvant en bas de la pile du podium rouge monte et se place en haut de la pile de ce même podium.

Le premier joueur ayant trouvé une bonne séquence de commandes remporte le tour. Si un joueur donne une mauvaise séquence, il ne peut plus en proposer d'autre pour le reste de ce tour. La situation atteinte après un tour est la situation de départ du tour qui suit. Si, durant un tour, il n'y a plus qu'un joueur à ne pas avoir proposer de séquence, ce dernier remporte le tour. La carte objectif trouvée devient la situation de départ du tour suivant.
Le joueur ayant remporté le plus de tours gagne la partie.

Diagramme UML



Tests unitaires des classes

```
package podiums;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

public class PodiumTest {
    @Test
    public void testAjouter() {
        Podium podium = new Podium();
        podium.ajouter(0);
        podium.ajouter(1);
        podium.ajouter(2);
        assertEquals("[lion, ours, elephant]", podium.toString());
    }

    @Test
    public void testDescendre() {
        Podium podium = new Podium();
        podium.ajouter(0);
        podium.ajouter(1);
        podium.ajouter(2);
        podium.descendre();
        podium.depiler();
        assertEquals("[ours, elephant]", podium.toString());
```

```
    }

    @Test
    public void testIndex() {
        Podium podium = new Podium();
        podium.ajouter(0);
        podium.ajouter(1);
        podium.ajouter(2);
        assertEquals(0, podium.index("lion"));
        assertEquals(1, podium.index("ours"));
        assertEquals(2, podium.index("elephant"));
    }

    @Test
    public void testDepiler() {
        Podium podium = new Podium();
        podium.ajouter(0);
        podium.ajouter(1);
        podium.ajouter(2);
        podium.depiler();
        assertEquals("[lion, ours]", podium.toString());
    }

    @Test
    public void testSommet() {
        Podium podium = new Podium();
        podium.ajouter(0);
        podium.ajouter(1);
        podium.ajouter(2);
        assertEquals("elephant", podium.sommet());
    }

    @Test
    public void testBase() {
        Podium podium = new Podium();
        podium.ajouter(0);
        podium.ajouter(1);
        podium.ajouter(2);
        assertEquals("lion", podium.base());
    }

    @Test
    public void testEstVide() {
        Podium podium = new Podium();
        assertTrue(podium.estVide());
        podium.ajouter(0);
        assertFalse(podium.estVide());
    }
```

```
    @Test
    public void testGetNb() {
        Podium podium = new Podium();
        assertEquals(0, podium.getNb());
        podium.ajouter(0);
        assertEquals(1, podium.getNb());
    }

    @Test
    public void testGetAnimal() {
        Podium podium = new Podium();
        podium.ajouter(0);
        podium.ajouter(1);
        podium.ajouter(2);
        assertEquals("lion", podium.getAnimal(0));
        assertEquals("ours", podium.getAnimal(1));
        assertEquals("elephant", podium.getAnimal(2));
    }
}
```

```
package jeu;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

public class JoueurTest {

    @Test
    public void testGetNomj() {
        Joueur j = new Joueur("Jean");
        assertEquals("Jean", j.getNomj());
    }

    @Test
    public void testSetNbj() {
        String[] noms = {"Jean", "Marie", "Pierre"};
        Joueur.setnbj(noms);
        assertEquals(3, Joueur.getNbj());
    }

    @Test
    public void testJoue() {
        Joueur j = new Joueur("Jean");
        assertFalse(j.ajoue());
        j.joue();
```

```
        assertTrue(j.ajoue());
        j.fini();
        assertFalse(j.ajoue());
    }

    @Test
    public void testPeutJouer() {
        String[] noms = {"Jean", "Marie", "Pierre"};
        Joueur.setnbj(noms);

        Joueur j1 = new Joueur("Jean");
        assertFalse(j1.peut_jouer("Marie"));
        assertTrue(j1.peut_jouer("Jean"));
        j1.joue();
        assertFalse(j1.peut_jouer("Jean"));
    }

    @Test
    public void testScorePlus() {
        Joueur j = new Joueur("Jean");
        assertEquals(0, j.getScore());
        j.scorePlus();
        assertEquals(1, j.getScore());
        j.scorePlus();
        assertEquals(2, j.getScore());
    }

}
```

```
package podiums;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;
import java.util.ArrayList;
import java.util.Arrays;

public class CarteTest {

    private Podium[] podiums;

    @Before
    public void setUp() {
        podiums = new Podium[3];
        for (int i = 0; i < podiums.length; i++) {
            podiums[i] = new Podium();
        }
```

```
    }

    @Test
    public void testInitList() {
        Carte.initList();
        assertEquals(24, Carte.getSize());
        assertFalse(Carte.estVide());
    }

    @Test
    public void testGetCarte() {
        Carte.initList();
        int size = Carte.getSize();
        for (int i = 0; i < size; i++) {
            Carte.getCarte();
            assertEquals(size - i - 1, Carte.getSize());
        }
        assertTrue(Carte.estVide());
    }

    @Test
    public void testInitCartes() {
        Carte.initList();
        int i = 0;
        int j = 1;
        Carte.getCarte();
        Carte.initCartes(podiums, i, j);
        assertFalse(podiums[0].estVide() && podiums[1].estVide());
    }
}
```

Bilan du projet

La SAE a été une expérience enrichissante d'un point de vue de compréhension du langage Java. Cela m'a permis de synthétiser mes connaissances et la SAE m'a permis de travailler les éléments vus en cours en TD et en TP durant la période.
En ce qui concerne les déceptions liées à ce projet, j'aurai aimé avoir le temps d' optimiser le programme sur certains points, comme les différentes cartes et les

conditions du programme principal. J’ai dû faire des compromis pour mener le projet à terme.
La réalisation des tours de hanoï pendant le tp5 a beaucoup aidé à la réalisation de ce projet.
Les difficultés rencontrées étaient, au début, d'implémenter les animaux sur les podiums. J’avais commencé par les mettre dans une classe par eux-mêmes avant de choisir de les garder dans la classe Podium.
L’obstacle suivant était de mettre en place l’affichage des animaux en podiums dans le Main, cela m’a fait perdre du temps à cause d’une erreur que j’avais ratée dans Podium. Pour les cartes, j’ai choisi de les rentrer manuellement et d’en avoir une aléatoire en mélangeant des entiers d’un tableau puis récupérer et éliminer le premier à chaque carte tirée.
Une autre difficultée était de rendre le podium objectif fini en podium initial pour le tour suivant, j’ai essayé la méthode clone, qui me donnait des problèmes de doublons d’animaux, et par peur de ne pas finir à temps j’ai opté pour stocker l’indice de la carte tirée, et de copier les podiums lorsque le tour se finit.
Pour mes réussites pour ce projet, j’ai réussi à faire fonctionner le jeu normalement. Et coder librement par moi-même pour un long programme m'a aidé à mieux comprendre le langage Java en plus des séances de 1h30 de TP/TD.
Pour ce qu’on pourrait améliorer, l’optimisation du code avec plus de fonctions et raccourcir certains passages, je pense notamment aux cartes que j’ai fait manuellement au lieu de les créer avec quelques boucles et toutes les conditions dans mon Main.

Code Complet

```
package podiums;

public class Podium{
  /** Nombre maximal d'animaux qu'un podium peut supporter. */
  private static final int MAX = 3;
```

```java
  /** Animaux supportés par la tour (chacun étant désigné par son nom). */
  private final static String[] ANIMAUX = {"lion","ours","elephant"};
  private String[] animals;
  /** Nombre d'animaux supportés par le podium. */
  private int nb;

  /**
   * Construit un podium vide de tout animal.
   */
  public Podium(){
      nb=0;
      animals = new String[MAX];
  }

  /**
   * @param position indice du nom de l'animal
   * ajoute un animal a la position donnée (0, 1 ou 2)
   * */
  public void ajouter(int position){

      animals[nb++] = ANIMAUX[position];
  }

  /**
   * Méthode pour descendre tous les animaux d'un rang dans le podium
   */
  public void descendre(){
      for(int i=0;i<MAX-1;++i)
          animals[i]=animals[i+1];
  }

  /**
   * @param s nom de l'animal
   * renvoie la position de l'animal (0, 1 ou 2)
   * */
  public int index(String s){
      assert(s!=null);
      for(int i=0;i< MAX;++i)
          if(s.equals(ANIMAUX[i]))
              return i;
      return 0;
  }

  /**
   * Méthode pour afficher les animaux dans un podium
   * @return String avec les animaux d'un podium
   */
  public String toString() {
```

```
        StringBuilder sb = new StringBuilder("[");
        for (int i = 0; i < nb; ++i) {
            if (i != 0)
                sb.append(", ");
            sb.append(animals[i]);
        }
        return sb.toString() + "]";
    }

    /**
     * Méthode pour enlever un animal d'un podium
     */
    public void depiler() {
        assert ! estVide();
        nb--;
    }

    /**
     * Méthode pour avoir l'animal au sommet du podium
     * @return String l'animal au sommet du podium
     */
    public String sommet() {
        assert ! estVide();
        return animals[nb-1];
    }

    /**
     * Méthode pour avoir l'animal a la base du podium
     * @return String l'animal a la base du podium
     */
    public String base(){
        return animals[0];
    }

    /**
     * Méthode pour vérifier si le podium est vide
     * @return true si il n'y a pas d'aniamux dans le podium
     */
    public boolean estVide() {
        return nb == 0;
    }

    /**
     * Méthode pour obtenir le nombre d'animaux dans le podium
     * @return int nb
     */
    public int getNb() {
        return nb;
```

```
    }

    /**
     * Méthode pour obtenir le nom de l'animal à la position demandée
     * @param i position dans le podium
     * @return String nom de l'animal
     */
    public String getAnimal(int i){
        return animals[i];
    }
    /**
     * Méthode pour enlever un animal d'un podium
     */
    public void decNb() {
        nb--;
    }
}
```

```
package jeu;

import java.util.Objects;

public class Joueur {
    //nom du joueur
    private String nomj;
    //score du joueur
    private int score;
    //nombre total de joueurs
    private static int nbj;
    //variable pour savoir si un joueur a joué ou non
    private int ajoue=0;

    /**
     * Constructeur de la classe Joueur
     * @param nom nom du joueur
     */
    public Joueur(String nom){
        nomj=nom;
        score=0;
    }
    public String getNomj(){
        return nomj;
    }

    /**
```

```
    * Méthode pour définir le nombre de joueurs à partir d'un tableau de noms de
joueurs
    * @param arg tableau de String contenant les noms des joueurs
    */
   public static void setnbj(String[] arg){
       nbj = arg.length;
   }

   /**
    * Méthode pour récupérer le nombre de joueurs
    * @return le nombre de joueurs
    */
   public static int getNbj(){
       return nbj;
   }

   /**
    * Méthode pour indiquer qu'un joueur a joué
    */
   public void joue(){
       ajoue=1;
   }

   /**
    * Méthode pour indiquer qu'un joueur a fini de jouer
    */
   public void fini() {
       ajoue=0;
   }

   /**
    * Méthode pour vérifier si un joueur est en train de jouer
    * @return true si le joueur a dèja joué
    */
   public boolean ajoue(){
       return ajoue==1;
   }

   /**
    * Méthode pour récupérer le score du joueur
    * @return score du joueur
    */
   public int getScore(){
       return score;
   }

   /**
    * Méthode pour vérifier si le joueur peut jouer
```

```
    * @param s nom du joueur
    * @return true si le joueur existe et qu'il n'a pas encore joué
    */
   public boolean peut_jouer(String s){
       if(joueurexiste(s))
           return this.ajoue==0;
       return false;
   }

   /**
    * Méthode pour vérifier si le joueur existe
    * @param s nom du joueur
    * @return true si le joueur existe
    */
   public boolean joueurexiste(String s){
       return Objects.equals(nomj, s);
   }

   /**
    * Méthode pour augmenter le score du joueur de 1
    */
   public void scorePlus(){
       score+=1;
   }
}
```

```
package podiums;

import java.util.ArrayList;
import java.util.Collections;


public class Carte {
   // tableau contenant les numéros de cartes disponibles
   private final static int[] cartes =
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24};
   // liste contenant les cartes mélangées
   private static ArrayList<Integer> l;
   // numéro de la dernière carte tirée
   private static int d = 0;

   /**
    * méthode pour initialiser la liste de cartes mélangées
    */

   public static void initList(){
```

```
        l = new ArrayList<>();
        for(int i: cartes)
            l.add(i);
        Collections.shuffle(l);
    }
    /**
     * méthode pour distribuer la prochaine carte de la liste
     */
    public static void getCarte(){
        int c=l.get(0);
        l.remove(0);
        d=c;
    }
    /**
     * méthode pour obtenir le nombre de cartes restantes dans la liste
     */
    public static int getSize(){
        return l.size();
    }
    /**
     * méthode pour vérifier si la liste de cartes est vide
     */
    public static boolean estVide(){
        return l.isEmpty();
    }
    /** @param p : tableau de podiums
     *  @param i : indice du premier podium
     *  @param j : indice du deuxieme podium
     *  méthode pour initialiser les podiums selon le numéro de la dernière carte
distribuée
     */
    public static void initCartes(Podium[] p, int i, int j) {
        switch (d) {
            case 1 -> {
                p[i].ajouter(0);
                p[i].ajouter(1);
                p[i].ajouter(2);
            }
            case 2 -> {
                p[i].ajouter(0);
                p[i].ajouter(1);
                p[j].ajouter(2);
            }
            case 3 -> {
                p[i].ajouter(0);
                p[j].ajouter(1);
                p[j].ajouter(2);
            }
```

```
        case 4 -> {
            p[j].ajouter(0);
            p[j].ajouter(1);
            p[j].ajouter(2);
        }
        case 5 -> {
            p[i].ajouter(1);
            p[i].ajouter(0);
            p[i].ajouter(2);
        }
        case 6 -> {
            p[i].ajouter(1);
            p[i].ajouter(0);
            p[j].ajouter(2);
        }
        case 7 -> {
            p[i].ajouter(1);
            p[j].ajouter(0);
            p[j].ajouter(2);
        }
        case 8 -> {
            p[j].ajouter(1);
            p[j].ajouter(0);
            p[j].ajouter(2);
        }
        case 9 -> {
            p[i].ajouter(1);
            p[i].ajouter(2);
            p[j].ajouter(0);
        }
        case 10 -> {
            p[i].ajouter(1);
            p[i].ajouter(2);
            p[j].ajouter(0);
        }
        case 11 -> {
            p[i].ajouter(1);
            p[j].ajouter(2);
            p[j].ajouter(0);
        }
        case 12 -> {
            p[j].ajouter(1);
            p[j].ajouter(2);
            p[j].ajouter(0);
        }
        case 13 -> {
            p[i].ajouter(0);
            p[i].ajouter(2);
```

```
            p[i].ajouter(1);
        }
        case 14 -> {
            p[i].ajouter(0);
            p[i].ajouter(2);
            p[j].ajouter(1);
        }
        case 15 -> {
            p[i].ajouter(0);
            p[j].ajouter(2);
            p[j].ajouter(1);
        }
        case 16 -> {
            p[j].ajouter(0);
            p[j].ajouter(2);
            p[j].ajouter(1);
        }
        case 17 -> {
            p[i].ajouter(2);
            p[j].ajouter(0);
            p[i].ajouter(1);
        }
        case 18 -> {
            p[i].ajouter(2);
            p[j].ajouter(0);
            p[j].ajouter(1);
        }
        case 19 -> {
            p[i].ajouter(2);
            p[j].ajouter(0);
            p[j].ajouter(1);
        }
        case 20 -> {
            p[j].ajouter(2);
            p[j].ajouter(0);
            p[j].ajouter(1);
        }
        case 21 -> {
            p[i].ajouter(2);
            p[i].ajouter(1);
            p[i].ajouter(0);
        }
        case 22 -> {
            p[i].ajouter(2);
            p[i].ajouter(1);
            p[j].ajouter(0);
        }
        case 23 -> {
```

```
            p[i].ajouter(2);
            p[j].ajouter(1);
            p[j].ajouter(0);
        }
        case 24 -> {
            p[j].ajouter(2);
            p[j].ajouter(1);
            p[j].ajouter(0);
        }


    }
  }
}
```

```
package jeu;


import podiums.Carte;
import podiums.Podium;

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

public class Main {
  /**
   * Déplace l'élément au sommet du podium "de" vers le podium "vers".
   *
   * @param de Le podium de départ.
   * @param vers Le podium d'arrivée.
   */
  private static void bouger(Podium de, Podium vers) {
    assert ! de.estVide();
    int d = de.index(de.sommet());
    de.depiler();
    vers.ajouter(d);
  }
  /**
   * Déplace l'élément à la base du podium "p" sur le dessus du podium
   * et déplace l'élément qui se trouvait au sommet sur la base du podium.
   *
   * @param p Le podium dont on veut déplacer les éléments.
   */
  private static void remonter(Podium p){
    int tmp=p.index(p.base());
    p.descendre();
```

```
        p.depiler();
        p.ajouter(tmp);
    }
    /**
     * Échange les éléments en haut des podiums "p1" et "p2".
     *
     * @param p1 Le premier podium.
     * @param p2 Le deuxième podium.
     */
    private static void echange(Podium p1, Podium p2){
        int tmp=p1.index(p1.sommet());
        p1.depiler();
        p1.ajouter(p2.index(p2.sommet()));
        p2.depiler();
        p2.ajouter(tmp);
    }
    /**
     * Exécute la commande spécifiée par la chaîne de caractères "s"
     * sur les podiums "p".
     *
     * @param p Les podiums sur lesquels on veut exécuter la commande.
     * @param s La commande à exécuter.
     */
    private static void commandes(Podium[] p,String s){
        switch (s) {
            case "KI" -> {
                bouger(p[0], p[1]);
                return;
            }
            case "LO" -> {
                bouger(p[1], p[0]);
                return;
            }
            case "SO" -> {
                echange(p[0], p[1]);
                return;
            }
            case "NI" -> {
                remonter(p[0]);
                return;
            }
            case "MA" -> {
                remonter(p[1]);
                return;
            }
            default -> {
                System.out.println("Commande inconnue");
            }
```

```
    }

  }
  /**
   * Méthode qui exécute une série de commandes pour avancer le jeu.
   * @param p un tableau de 4 Podiums représentant l'état actuel du jeu.
   * @param s une chaîne de caractères représentant les commandes à exécuter.
   * @param Tcommande un entier représentant l'indice de départ des commandes à
exécuter dans la chaîne s.
   */
  public static void action(Podium[] p, String s, int Tcommande){
    int T = 2;
    for(;Tcommande<s.length(); Tcommande += T)
      commandes(p, s.substring(Tcommande, Tcommande + T));
  }
  /**
   * Méthode qui vérifie si les deux premiers Podiums sont identiques aux deux
derniers.
   * @param p un tableau de 4 Podiums représentant l'état actuel du jeu.
   * @return true si les deux premiers Podiums sont identiques aux deux derniers,
false sinon.
   */
  public static boolean reussite(Podium[] p){
    for (int i=0; i<2;++i)
      for(int j=0; j<p[i].getNb();++j) {
        if (!Objects.equals(p[i].getAnimal(j), p[i + 2].getAnimal(j)))
          return false;

      }
    return true;
  }
  /**
   * Méthode qui affiche le nom d'un animal dans un Podium.
   * @param p un Podium contenant l'animal à afficher.
   * @param i l'indice de l'animal à afficher dans le Podium.
   */
  private static void afficheAnimal(Podium p, int i){
    if (!Objects.equals(p.getAnimal(i), "elephant")){
      System.out.print("  ");
    }
    System.out.print(p.getAnimal(i));
    if (!Objects.equals(p.getAnimal(i), "elephant")){
      System.out.print("  ");
    }
  }
  /**
   * Méthode qui vérifie si tous les joueurs sauf un ont joué au moins une fois.
   * @param joueurs un tableau de joueurs représentant les joueurs dans le jeu.
```

```
     * @return true si tous les joueurs sauf un ont joué, false sinon.
     */
    private static boolean dernierJoueur(Joueur[] joueurs){
        int cpt=0;
        for(Joueur j:joueurs)
            if(j.ajoue())
                cpt+=1;
        return cpt == Joueur.getNbj() - 1;
    }
    /**
     * Méthode qui affiche le contenu des Podiums et le menu de commandes.
     * @param podiums un tableau de 4 Podiums représentant l'état actuel du jeu.
     * @param d l'indice de l'animal à afficher dans chaque Podium.
     */
    private static void afficher(Podium[] podiums, int d){
            int cpt=0;
            for(Podium p : podiums){
                cpt+=1;
                if (cpt==3)
                    System.out.print("       ");
                if(p.getNb()>d){
                    afficheAnimal(p,d);
                }
                else
                    System.out.print("       ");
            }

    }
    /**
     * Méthode qui affiche le menu de commandes.
     * @param sl une chaîne de caractères représentant le joueur actuel.
     */
    private static void afficherMenu(String sl){
        System.out.print("    ----    ----    ==>    ----    ----    ");
        System.out.print(sl);
        System.out.println("    BLEU    ROUGE          BLEU    ROUGE   ");
        System.out.println("-----------------------------------------");
        System.out.println("KI : BLEU --> ROUGE NI : BLEU ^");
        System.out.println("LO : BLEU <-- ROUGE MA : ROUGE ^");
        System.out.println("SO : BLEU <-> ROUGE");
    }

    /**
     * Affiche tous les podiums et le menu
     * @param podiums tableau contenant les podiums à afficher
     * @param sl String contenant un saut de ligne
     */
    private static void affichage(Podium[] podiums, String sl){
```

```
    for(int i=2;i>=0;--i){
        System.out.print("   ");
        afficher(podiums,i);
        System.out.print(sl);
    }
    afficherMenu(sl);
}
public static void main(String[] args){
    String sl=System.getProperty("line.separator");

    //Initialiser les joueurs

    if(args.length==0)
        System.out.println("Il n'y a aucun joueur !");
    Joueur.setnbj(args);
    final int TAILLE1=Joueur.getNbj();
    Joueur[] joueurs = new Joueur[TAILLE1];
    for(int i=0;i<TAILLE1;++i){
        joueurs[i]= new Joueur(args[i]);
    }

    Scanner sc = new Scanner(System.in);

    //Initialiser les podiums

    final int TAILLE2 = 4;
    Podium[] podiums = new Podium[TAILLE2];
    podiums[0] = new Podium();
    podiums[1] = new Podium();
    podiums[2] = new Podium();
    podiums[3] = new Podium();

    //Melanger les cartes, puis obtenir la carte de départ

    Carte.initList();
    Carte.getCarte();
    Carte.initCartes(podiums,0,1);

    do {
        Carte.getCarte();
        podiums[2] = new Podium();
        podiums[3] = new Podium();
        Carte.initCartes(podiums,2,3); //tirer les podiums objectif

        do {
            affichage(podiums, sl);
            int numj = 0;
            String s = sc.nextLine();
```

```
            int espaceIndex = s.indexOf(' ');
            String premierePartie = s.substring(0, espaceIndex);
            String deuxiemePartie = s.substring(espaceIndex + 1); //découper la chaine entrée par l'utilisateur en 2 parties : nom du joueur et commandes


            for (int i = 0; i < TAILLE1; ++i) {
                if (Objects.equals(joueurs[i].getNomj(), premierePartie)) { //verifier si le joueur a deja joué
                    if(joueurs[i].peut_jouer(premierePartie))
                        numj = i;
                    else
                        System.out.println("Ce joueur a deja joue");
                }
            }


            if (!dernierJoueur(joueurs) && joueurs[numj].peut_jouer(premierePartie) ) { //le joueur peut jouer
                action(podiums, deuxiemePartie, 0);
                joueurs[numj].joue();
            }
            if(dernierJoueur(joueurs)) { //si c'est le dernier joueur, recuperer son indice
                for (int i = 0; i < Joueur.getNbj(); ++i)
                    if (!joueurs[i].ajoue()) {
                        numj = i;
                    }
            }
            else if (!joueurs[numj].joueurexiste(premierePartie))
                System.out.println("Ce joueur n'existe pas");


            if(reussite(podiums) || dernierJoueur(joueurs)){       //si c'est le dernier joueur ou si la sequence est bonne
                System.out.println("Manche remportee par : "+joueurs[numj].getNomj());
                joueurs[numj].scorePlus();
                podiums[0] = new Podium();
                podiums[1] = new Podium();
                Carte.initCartes(podiums,0,1);
            }
            else
                System.out.println("Sequence incorrecte");

        }while(!reussite(podiums)); //boucle tant que les podiums ne sont pas identiques
        for(Joueur j : joueurs){ //fin de tour, les joueurs peuvent rejouer
            j.fini();
        }
```

```
    }while(!Carte.estVide());

    int scoremax=0;
    for(int i=0;i<TAILLE1;++i){
        if(joueurs[i].getScore() > joueurs[scoremax].getScore()) //recuperer le joueur
avec le plus haut score
            scoremax=i;
    }
    System.out.println(joueurs[scoremax].getNomj() + " a gagne !");

  }
}
```