KALIAMOORTHY Namodacane 108

CAROUNANITHI Alexandre 107

BUT1 – Projet

R2.01 Développement orienté objets

– Le Crazy Circus –

IUT de Paris - Rives de Seine
Université Paris Cité

Année 2022-2023

Table des matières

1. Introduction du projet

Ce projet consiste à développer un jeu de société populaire des années 2000 « Crazy Circus », en un jeu console. Dans notre projet, le langage utilisé est le langage de programmation orienté objet : le Java. Le but de ce jeu est de devenir le meilleur « dompteur » en domptant 3 animaux (lion, ours blanc, éléphant) répartis sur 2 podiums (rouge et bleu). Pour cela, une situation de départ est tirée au hasard et chaque joueur doit trouver le plus rapidement la bonne séquence d'ordre, qui lui permettra d'arriver à la situation demandée. Pour mieux visualiser la logique de ce jeu, vous retrouverez ci-dessous un exemple d'une manche réalisée.

Les podiums de gauche représentent la situation de départ et les podiums de droite, la situation d'arrivée. Les commandes utilisables sont décrites plus bas.

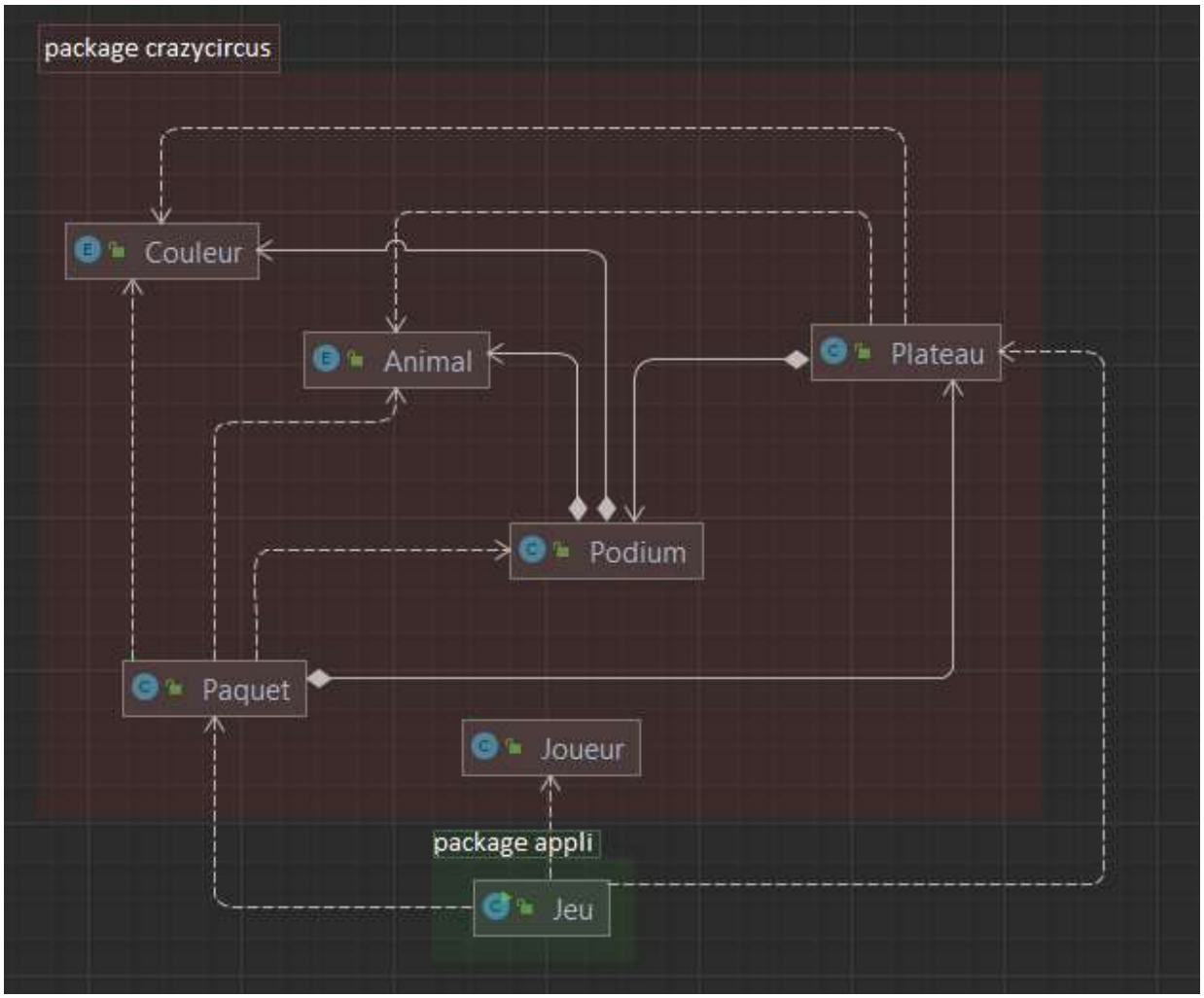
```
  OURS                          LION
  LION     ELEPHANT         ELEPHANT     OURS
  ----       ----     ==>     ----       ----
  BLEU       ROUGE             BLEU       ROUGE

-----------------------------------------
KI : BLEU --> ROUGE     NI : BLEU   ^
LO : BLEU <-- ROUGE     MA : ROUGE  ^
SO : BLEU <-> ROUGE
```

Dans cet exemple, plusieurs séquences permettent d'arriver à la situation finale, dont « KIMALONI » et « SONI ». Plus précisément, la séquence « SONI » est constituée de la commande SO et NI. Ici, SO effectue un échange de podium de l'ours avec l'éléphant et NI remonte l'animal situé en bas du podium bleu, de sorte à ce que le lion se situe en haut et l'éléphant en bas.

Pour finir, ce jeu permet de développer la logique de chacun avec au total une grande combinaisons de séquences et cartes. Que le meilleur dompteur l'emporte !

2. Diagramme UML des classes formant l'application



3. Le code Java des tests unitaires

Tous les jeux de tests effectués fonctionnent parfaitement. Ci-dessous vous retrouverez les différents tests unitaires réalisés : PlateauTest, PaquetTest, JoueurTest.

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

import org.junit.Test;

public class PlateauTest {

    @Test
    public void testEstVide() {
        Plateau p = new Plateau();
        assertTrue(p.estVide(Couleur.BLEU) == true);
    }

    @Test
    public void testKI() {
        Animal elephant = Animal.ELEPHANT;
        Animal ours = Animal.OURS;
        Animal lion = Animal.LION;

        Plateau p1 = new Plateau();
        p1.getPodium(Couleur.BLEU).ajouter(elephant);
        p1.getPodium(Couleur.BLEU).ajouter(lion);
        p1.getPodium(Couleur.ROUGE).ajouter(ours);

        p1.KI();
        assertTrue(p1.getPodium(Couleur.BLEU).getSommet().equals(elephant));
        assertTrue(p1.getPodium(Couleur.ROUGE).getSommet().equals(lion));

    }

    @Test
    public void testLO() {
        Animal elephant = Animal.ELEPHANT;
        Animal ours = Animal.OURS;
        Animal lion = Animal.LION;

        Plateau p1 = new Plateau();
        p1.getPodium(Couleur.ROUGE).ajouter(elephant);
        p1.getPodium(Couleur.ROUGE).ajouter(lion);
        p1.getPodium(Couleur.BLEU).ajouter(ours);

        p1.LO();
```

```
        assertTrue(p1.getPodium(Couleur.ROUGE).getSommet().equals(elephant))
;
        assertTrue(p1.getPodium(Couleur.BLEU).getSommet().equals(lion));
    }

    @Test
    public void testSO() {
        Animal elephant = Animal.ELEPHANT;
        Animal ours = Animal.OURS;
        Animal lion = Animal.LION;

        Plateau p1 = new Plateau();
        p1.getPodium(Couleur.ROUGE).ajouter(elephant);
        p1.getPodium(Couleur.ROUGE).ajouter(lion);
        p1.getPodium(Couleur.BLEU).ajouter(ours);

        p1.SO();
        assertTrue(p1.getPodium(Couleur.ROUGE).getSommet().equals(ours));
        assertTrue(p1.getPodium(Couleur.BLEU).getSommet().equals(lion));
    }

    @Test
    public void testNI() {
        Animal elephant = Animal.ELEPHANT;
        Animal ours = Animal.OURS;
        Animal lion = Animal.LION;

        Plateau p1 = new Plateau();
        p1.getPodium(Couleur.BLEU).ajouter(elephant);
        p1.getPodium(Couleur.BLEU).ajouter(lion);
        p1.getPodium(Couleur.BLEU).ajouter(ours);

        p1.NI();
        assertTrue(p1.getPodium(Couleur.BLEU).getSommet().equals(elephant));
        assertTrue(p1.getPodium(Couleur.BLEU).getAnimal(0).equals(lion));
    }

    @Test
    public void testMA() {
        Animal elephant = Animal.ELEPHANT;
        Animal ours = Animal.OURS;
        Animal lion = Animal.LION;

        Plateau p1 = new Plateau();
        p1.getPodium(Couleur.ROUGE).ajouter(elephant);
        p1.getPodium(Couleur.ROUGE).ajouter(lion);
        p1.getPodium(Couleur.ROUGE).ajouter(ours);

        p1.MA();
```

```
        assertTrue(p1.getPodium(Couleur.ROUGE).getSommet().equals(elephant))
;
        assertTrue(p1.getPodium(Couleur.ROUGE).getAnimal(0).equals(lion));
    }

    @Test
    public void testOrdreCorrect() {
        String ordres1 = "SOLOMANIKI";
        assertTrue(Plateau.ordreCorrect(ordres1) == true);

        String ordres2 = "NAMO";
        assertTrue(Plateau.ordreCorrect(ordres2) == false);
    }

    @Test
    public void testExecuterOrdre() {
        String reponse = "KI";

        Animal elephant = Animal.ELEPHANT;
        Animal ours = Animal.OURS;
        Animal lion = Animal.LION;

        Plateau p1 = new Plateau();
        p1.getPodium(Couleur.BLEU).ajouter(elephant);
        p1.getPodium(Couleur.ROUGE).ajouter(ours);
        p1.getPodium(Couleur.ROUGE).ajouter(lion);

        p1.executerOrdre(reponse);
        assertTrue(p1.estVide(Couleur.BLEU) == true);
        assertTrue(p1.getPodium(Couleur.ROUGE).getSommet() == elephant);
    }

    @Test
    public void testComparer() {

        Animal elephant = Animal.ELEPHANT;
        Animal ours = Animal.OURS;
        Animal lion = Animal.LION;


        Plateau p1 = new Plateau();
        p1.getPodium(Couleur.BLEU).ajouter(elephant);
        p1.getPodium(Couleur.ROUGE).ajouter(ours);
        p1.getPodium(Couleur.ROUGE).ajouter(lion);

        Plateau p2 = new Plateau();
        p2.getPodium(Couleur.BLEU).ajouter(elephant);
        p2.getPodium(Couleur.ROUGE).ajouter(ours);
        p2.getPodium(Couleur.ROUGE).ajouter(lion);
```

```
        assertTrue(p1.comparer(p2) == true);
    }
}
```

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

import java.util.ArrayList;

import org.junit.Test;

public class PaquetTest {

    @Test
    public void testPaquets() {
        ArrayList<Plateau> paquet;

        Animal elephant = Animal.ELEPHANT;
        Animal ours = Animal.OURS;
        Animal lion = Animal.LION;

        paquet = new ArrayList<>();

        Plateau p1 = new Plateau();
        p1.getPodium(Couleur.BLEU).ajouter(elephant);
        p1.getPodium(Couleur.BLEU).ajouter(ours);
        p1.getPodium(Couleur.BLEU).ajouter(lion);
        paquet.add(p1);

        Plateau p2 = new Plateau();
        p2.getPodium(Couleur.BLEU).ajouter(elephant);
        p2.getPodium(Couleur.BLEU).ajouter(ours);
        p2.getPodium(Couleur.BLEU).ajouter(lion);
        paquet.add(p2);

        assertTrue(p1.getPodium(Couleur.BLEU).getPodium().equals(p2.getPodiu
m(Couleur.BLEU).getPodium()));
    }

    @Test
    public void testMelanger() {

        Animal elephant = Animal.ELEPHANT;
        Animal ours = Animal.OURS;
        Animal lion = Animal.LION;
```

```
        Plateau p1 = new Plateau();
        p1.getPodium(Couleur.BLEU).ajouter(elephant);
        p1.getPodium(Couleur.BLEU).ajouter(ours);
        p1.getPodium(Couleur.BLEU).ajouter(lion);

        Paquet paquet = new Paquet();
        paquet.melanger();

        assertTrue(!p1.equals(paquet.getCarte(0)));
    }
}
```

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

import java.util.ArrayList;

import org.junit.Test;

public class JoueurTest {

    @Test
    public void testJoueur() {
        Joueur j = new Joueur("Alexandre");
        assertTrue(j.getNom().equals("Alexandre"));
        assertTrue(j.getScore() == 0);
        assertTrue(j.getADejaJouer() == false);
    }
    @Test
    public void testCreerJoueur() {
        ArrayList<Joueur> joueurs  = new ArrayList<>();
        String nomJoueur = "Alexandre";
        Joueur.creerJoueur(joueurs, nomJoueur);
        assertTrue("Alexandre" == joueurs.get(0).getNom());
    }
    @Test
    public void testVerifJoueur() {
        ArrayList<Joueur> joueurs  = new ArrayList<>();
        String nomJoueur1 = "Alexandre";
        Joueur.creerJoueur(joueurs, nomJoueur1);
        String nomJoueur2 = "Namodacane";
        Joueur.creerJoueur(joueurs, nomJoueur2);
        assertTrue(Joueur.verifJoueur(joueurs, "Alexandre") == true);
        assertTrue(Joueur.verifJoueur(joueurs, "Namodacane") == true);
    }
```

```
    @Test
    public void testAjouterPoint() {
        Joueur j = new Joueur("Alexandre");
        j.ajouterPoint();
        assertTrue(j.getScore() == 1);
    }
    @Test
    public void testSetADejaJoue() {
        Joueur j = new Joueur("Namodacane");
        j.setADejaJouer();
        assertTrue(j.getADejaJouer() == true);
    }
    @Test
    public void testSetAPasJoue() {
        Joueur j = new Joueur("Namodacane");
        j.setAPasJouer();
        assertTrue(j.getADejaJouer() == false);
    }
    @Test
    public void testResetADejaJouer() {
        ArrayList<Joueur> joueurs  = new ArrayList<>();
        String nomJoueur1 = "Alexandre";
        Joueur.creerJoueur(joueurs, nomJoueur1);
        String nomJoueur2 = "Namodacane";
        Joueur.creerJoueur(joueurs, nomJoueur2);
        joueurs.get(0).setADejaJouer();
        joueurs.get(1).setADejaJouer();
        Joueur.resetADejaJouer(joueurs);
        assertTrue(joueurs.get(0).getADejaJouer() == false);
        assertTrue(joueurs.get(1).getADejaJouer() == false);
    }
    @Test
    public void testTriClassment() {
        ArrayList<Joueur> joueurs  = new ArrayList<>();
        String nomJoueur1 = "Alexandre";
        Joueur.creerJoueur(joueurs, nomJoueur1);
        String nomJoueur2 = "Namodacane";
        Joueur.creerJoueur(joueurs, nomJoueur2);
        joueurs.get(1).ajouterPoint();
        Joueur.resetADejaJouer(joueurs);
        Joueur.triClassement(joueurs);
        assertTrue(joueurs.get(0).getNom().equals("Namodacane"));
        assertTrue(joueurs.get(1).getNom().equals("Alexandre"));
    }
}
```

4. Bilan du projet

Selon nous, ce projet nous a été davantage bénéfique pour poser les bases du langage de programmation Java. En effet, il a permis de consolider nos connaissances puisque nous étions face à certaines nouvelles situations que nous n'avons pas encore réalisé. Ce nouveau langage a été assimilé plus aisément grâce aux nombreuses bibliothèques existantes notamment grâce à la bibliothèque Collections et ArrayList.

Également, connaisseurs de ce jeu de société, nous avons pris du plaisir à le développer en un jeu console. Assez simple d'utilisation et de compréhension, ce projet nous est nettement plus accessible et nous l'avons trouvé plus simple que ceux fait précédemment.

En définitive, réaliser ce projet de A à Z nous a été d'une grande utilité puisque nous étions totalement autonomes sur sa réalisation.

a) Difficultés rencontrées

Au cours de ce projet, nous avons fait face à certaines difficultés que nous avons pu surmonter. Par exemple, nous avons eu du mal à trouver la bonne méthode pour réaliser l'affichage (To String). En effet, nous sommes partis de l'idée d'un affiche podium par podium. Seulement, nous nous sommes rendus compte que ce n'était pas la meilleure des méthodes notamment pour les alignements puisque les noms d'animaux ne font pas la même longueur. Ainsi, nous avons réalisé un affichage ligne par ligne, nous permettant de respecter le modèle demandé.

Concernant la comparaison des situations de chaque plateau, nous avons eu une difficulté étant donné que « .equals » ne fonctionne qu'avec des ArrayList. Or, nos podiums ont été défini par un type Podium. C'est pourquoi nous devions faire un « .get » pour avoir l'ArrayList de la classe Podium afin de les comparer entre eux avec « .equals ».

Également, concernant la création des cartes, nous les avons toutes créées une par une à la main, en pensant à toutes les combinaisons possibles. Seulement, cette méthode n'est pas la plus optimisée.

Enfin, trouver les classes nous a pris du temps étant donné qu'il fallait trouver les classes primordiales et utiles afin de ne pas alourdir le code.

b) Ce qui est réussi

Globalement, notre projet répond bien aux attentes demandées pour ce jeu. Le score et l'affichage correspondent bien aux descriptions du jeu. Toutes les commandes fonctionnent correctement ainsi que le système de tour.

c) Ce qui peut être amélioré

Concrètement, nous avons conscience que notre programme peut être amélioré sur divers points. Par exemple, certaines méthodes peuvent être allégées voir fusionnées avec d'autres, certaines peuvent devenir des méthodes d'instances plutôt que des méthodes statiques etc... Également, le programme principal du jeu peut être allégé en y retenant seulement les étapes clés. Concernant la création des cartes, nous pouvons aussi l'améliorer en créant un algorithme qui les génèrent automatiquement.

5. Le code Java complet

```
package Appli;
import crazycircus.*;
import java.util.ArrayList;
import java.util.Scanner;
/**
 * @file jeu.java
 * Projet BUT S2 - Le jeu Crazy Circus
 * @author Namodacane KALIAMOORTHY et Alexandre CAROUNANITHI
 * @version 5 - 15/03/2023
 * @brief Programme principal du jeu : Crazy Circus
 */

public class Jeu {

    public static void main(String[] args) {

        System.out.println("\n================ BIENVENUE SUR CRAZY CIRCUS
================");
        System.out.println("Voici les règles : \nEssayez de trouvez le plus
rapidement possible la bonne combinaison. Bonne chance !\n");
        ArrayList<Joueur> joueurs  = new ArrayList<>();
        int nbJoueur = 0;

        /*Creation des Joueurs*/
        if(args.length < 2) {
            System.out.println("Saisissez au minimum 2 joueurs");
            System.exit(1);
        }
        else {
            for(int i = 0; i<args.length; ++i) {
                String nomJoueur = args[i].toUpperCase();
                if(Joueur.verifJoueur(joueurs, nomJoueur)) {
                    System.out.println("Le joueur " + nomJoueur + " existe
déjà, relancez le jeu !");
                    System.exit(1);
                }else {
                    Joueur.creerJoueur(joueurs, nomJoueur);
                    System.out.println("Joueur " + (i+1) + " : " +
nomJoueur);
                    ++nbJoueur;
                }
            }
            System.out.println("Nombre de participant : " + nbJoueur);
        }
        /*Fin création des joueurs*/
```

```
        /*Début du jeu*/
        System.out.println("\nQue le jeu commence !\n");

        Paquet paquet = new Paquet();
        paquet.melanger();
        System.out.println("Pour répondre, saisissez votre nom puis la bonne
suite d'ordre (exemple : DP KIMALONI)\n");


        for(int i = 0; i<paquet.getSizePaquet()-1; ++i) {
            Plateau.AffichePartie(paquet.getCarte(i), paquet.getCarte(i+1));
//Affiche la partie
            int nbJoueurRestant = joueurs.size();
            do {

                if(nbJoueurRestant != 1) {
                    String nomJoueur = "";
                    String ordreJoueur = "";

                    while(nomJoueur.trim().isEmpty() ||
ordreJoueur.trim().isEmpty()) {

                        System.out.print("Votre réponse : ");
                        @SuppressWarnings("resource")

                        Scanner sc = new Scanner(System.in);

                        nomJoueur = sc.next().toUpperCase().trim();
                        ordreJoueur = sc.nextLine().toUpperCase().trim();

                        //System.out.println(nomJoueur +" " + ordreJoueur);

                        if(nomJoueur.trim().isEmpty() ||
ordreJoueur.trim().isEmpty()) {
                            System.out.println("Veuillez saisir le nom et
ensuite l'ordre !");
                        }

                    }

                    if(!Joueur.verifJoueur(joueurs, nomJoueur))
                        System.out.println("Le joueur n'existe pas !");
                    else {
                        if(joueurs.get(Joueur.getIndice(joueurs,
nomJoueur)).getADejaJouer()) {
                            System.out.println("Vous ne pouvez plus rejouer
pendant cette manche !");
                        }
                        else {
```

```
                            if(!Plateau.ordreCorrect(ordreJoueur)) {
                                --nbJoueurRestant;
                                System.out.println("Votre réponse est
incorrecte, vous ne pouvez plus jouer pendant cette manche !");
                                joueurs.get(Joueur.getIndice(joueurs,
nomJoueur)).setADejaJouer();
                            }
                            else {
                                Plateau pExecuter = paquet.getCarte(i);
                                pExecuter.executerOrdre(ordreJoueur);
                                if(!pExecuter.comparer(paquet.getCarte(i+1))
) {
                                    --nbJoueurRestant;
                                    System.out.println("Votre réponse est
incorrecte, vous ne pouvez plus jouer pendant cette manche !");
                                    joueurs.get(Joueur.getIndice(joueurs,
nomJoueur)).setADejaJouer();
                                }
                                else {
                                    System.out.println("Bravo ! Vous avez
gagnez un point !\n");
                                    joueurs.get(Joueur.getIndice(joueurs,
nomJoueur)).ajouterPoint();
                                    Joueur.resetADejaJouer(joueurs);
                                    break;
                                }
                            }
                        }
                    }


                }
                else {
                    System.out.println("Il ne reste qu'un joueur, il gagne
la manche\n");
                    for(int j = 0; j < joueurs.size(); ++j) {
                        if(!joueurs.get(j).getADejaJouer()) {
                            joueurs.get(j).ajouterPoint();
                        }
                    }
                    Joueur.resetADejaJouer(joueurs);
                    break;
                }
            }while((paquet.getCarte(i).equals(paquet.getCarte(i+1)) ==
false));
        }
        /*Fin du jeu*/
        System.out.println("Fin du jeu ! Merci d'avoir joué");
        Joueur.triClassement(joueurs);
```

```
        Joueur.afficherClassement(joueurs);
        System.out.println("\n==========================================
================");
    }

}
```

```
package crazycircus;
/**
 * @file Plateau.java
 * @brief Entête de la classe Plateau
 */

import java.util.Arrays;

public class Plateau {
    private Podium podiumBleu;
    private Podium podiumRouge;

    /**
     * Construit le plateau composé de deux podiums
     */
    public Plateau() {
        podiumBleu = new Podium(Couleur.BLEU);
        podiumRouge = new Podium(Couleur.ROUGE);
    }

    /**
     * Renvoi le podium en fonction de sa couleur
     * @param c la Couleur du podium
     * @return podiumBleu l'ArrayList du podium bleu
     */
    public Podium getPodium(Couleur c) {
        if (c == Couleur.BLEU)
            return podiumBleu;
        return podiumRouge;
    }

    /**
     * Verifie si le podium est vide
     * @param c la Couleur du podium
     * @return un boolean en fonction de la verification
     */
    public boolean estVide(Couleur c) {
        if (c == Couleur.BLEU)
            return (this.podiumBleu.estVide());
        return (this.podiumRouge.estVide());
```

```
    }

    /**
     * Ajoute au sommet du podium rouge l'animal qui se trouve au sommet du
podium bleu
     * @param p le plateau
     */
    public void KI() {
        if(!podiumBleu.estVide()) {
            podiumRouge.ajouter(podiumBleu.getSommet());
            podiumBleu.retirer(podiumBleu.getSize()-1);
        }
    }
    /**
     * Ajoute au sommet du podium bleu l'animal qui se trouve au sommet du
podium rouge
     * @param p le plateau
     */
    public void LO() {
        if(!podiumRouge.estVide()) {
            podiumBleu.ajouter(podiumRouge.getSommet());
            podiumRouge.retirer(podiumRouge.getSize()-1);
        }
    }
    /**
     * Echange les sommets du podium rouge et du podium bleu
     * @param p le plateau
     */
    public void SO() {
        Animal tmp = podiumRouge.getSommet();
        podiumRouge.retirer(podiumRouge.getSize()-1);
        KI();
        podiumBleu.ajouter(tmp);
    }
    /**
     * Prend l'animal en bas du podium bleu pour le mettre au sommet de ce
même podium
     * @param p le plateau
     */
    public void NI() {
        if(!podiumBleu.estVide()) {
            podiumBleu.ajouter(podiumBleu.getAnimal(0));
            podiumBleu.retirer(0);
        }
    }
    /**
     * Prend l'animal en bas du podium rouge pour le mettre au sommet de ce
même podium
     * @param p le plateau
```

```
     */
    public void MA() {
        if(!podiumRouge.estVide()) {
            podiumRouge.ajouter(podiumRouge.getAnimal(0));
            podiumRouge.retirer(0);
        }
    }
    /**
     * Verifie si l'ordre entrée est correcte
     * @param ordres l'ordre tapé du joueur
     * @return un boolean en fonction de la verification
     */
    public static boolean ordreCorrect(String ordres) {

        String[] ordresValide = {"KI", "LO", "SO", "NI", "MA"};

        for(int i = 0; i<ordres.length()-1; i+=2) {
            String ordre = ordres.substring(i, i+2);
            if(!Arrays.asList(ordresValide).contains(ordre)) {
                return false;
            }
        }
        return true;
    }
    /**
     * Execute l'ordre du joueur
     * @param reponse qui est l'ordre donnée par le joueur
     */
    public void executerOrdre(String reponse) {
        for(int i = 0; i < reponse.length(); i += 2) {
            String ordre = reponse.substring(i, i + 2);
            switch (ordre) {
                case "KI":
                    KI();
                    break;
                case "LO":
                    LO();
                    break;
                case "SO":
                    SO();
                    break;
                case "NI":
                    NI();
                    break;
                case "MA":
                    MA();
                    break;
                default:
                    break;
```

```
            }
        }

    }

    /**
     * Verifie le plateau initiale et le plateau final sont identiques
     * @param pfinal le plateau final
     * @return un boolean en fonction de la verification
     */
    public boolean comparer(Plateau pFinal) {
        return
(this.podiumBleu.getPodium().equals(pFinal.podiumBleu.getPodium()) &&
this.podiumRouge.getPodium().equals(pFinal.podiumRouge.getPodium()));
    }

    /**
     * Affiche la partie donc le plateau initial et le plateau final
     * @param pInitiale le plateau initiale
     * @param pFinal le plateau final
     */
    public static final void AffichePartie(Plateau pInitial, Plateau pFinal)
{

        /* Création des animaux */
        Animal elephant = Animal.ELEPHANT;
        Animal ours = Animal.OURS;
        Animal lion = Animal.LION;

        /* Affichage */
        for(int i = 3; i > 0; i--) {
            if(i <= pInitial.podiumBleu.getSize()) {
                if(pInitial.podiumBleu.getAnimal(i-1).equals(ours)) {
                    System.out.print("  " + pInitial.podiumBleu.getAnimal(i-
1) + "  ");
                }
                if(pInitial.podiumBleu.getAnimal(i-1).equals(lion)) {
                    System.out.print("  " + pInitial.podiumBleu.getAnimal(i-
1) + "  ");
                }
                if(pInitial.podiumBleu.getAnimal(i-1).equals(elephant)) {
                    System.out.print(pInitial.podiumBleu.getAnimal(i-1));
                }
            }
            else {
                System.out.print("        ");
            }

            if(i <= pInitial.podiumRouge.getSize()) {
```

```
                if(pInitial.podiumRouge.getAnimal(i-1).equals(ours)) {
                    System.out.print("   " +
pInitial.podiumRouge.getAnimal(i-1) + "         ");
                }
                else if(pInitial.podiumRouge.getAnimal(i-1).equals(lion)) {
                    System.out.print("   " +
pInitial.podiumRouge.getAnimal(i-1) + "         ");
                }
                else if(pInitial.podiumRouge.getAnimal(i-
1).equals(elephant)) {
                    System.out.print(" " + pInitial.podiumRouge.getAnimal(i-
1) + "        ");
                }
            }
            else {
                System.out.print("                ");
            }

            if(i <= pFinal.podiumBleu.getSize()) {
                if(pFinal.podiumBleu.getAnimal(i-1).equals(ours)) {
                    System.out.print("  " + pFinal.podiumBleu.getAnimal(i-1)
+ "   ");
                }
                else if(pFinal.podiumBleu.getAnimal(i-1).equals(lion)) {
                    System.out.print("  " +pFinal.podiumBleu.getAnimal(i-1)
+ "   ");
                }
                else if(pFinal.podiumBleu.getAnimal(i-1).equals(elephant)) {
                    System.out.print(pFinal.podiumBleu.getAnimal(i-1));
                }
            }
            else {
                System.out.print("        ");
            }

            if(i <= pFinal.podiumRouge.getSize()) {
                if(pFinal.podiumRouge.getAnimal(i-1).equals(ours)) {
                    System.out.println("   " +
pFinal.podiumRouge.getAnimal(i-1));
                }
                else if(pFinal.podiumRouge.getAnimal(i-1).equals(lion)) {
                    System.out.println("   " +
pFinal.podiumRouge.getAnimal(i-1));
                }
                else if(pFinal.podiumRouge.getAnimal(i-1).equals(elephant))
{
                    System.out.println(" " + pFinal.podiumRouge.getAnimal(i-
1));
                }
```

```
            }
            else {
                System.out.println("        ");
            }
        }
        AfficheOrdrePossible();
    }

    /**
     * Affiche les ordres possibles
     */
    public static final void AfficheOrdrePossible() {
        System.out.println("  ----     ----    ==>    ----     ----");
        System.out.println("  BLEU     ROUGE          BLEU     ROUGE\n");
        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\n");
    }
}
```

```
package crazycircus;

/**
* @file Podium.java
* @brief Entête de la classe Podium
*/

import java.util.ArrayList;

public class Podium {
    private ArrayList<Animal> podium;
    Couleur couleur;

    /**
     * Construit l'objet podium correspondant à une couleur
     */
    public Podium (Couleur c) {
        podium = new ArrayList<>();
        couleur = c;
    }

    /**
     * Verifie si le podium est vide
     * @return un boolean estVide
     */
    public boolean estVide() {
```

```
        return (this.podium.isEmpty());
    }

    /**
     * Renvoie la taille du podium
     * @return size la taille du podium
     */
    public int getSize() {
        return podium.size();
    }

    /**
     * Renvoie l'animal à la position donné
     * @param position la position de l'animal
     * @return l'animal à la position
     */
    public Animal getAnimal(int position) {
        return podium.get(position);
    }

    /**
     * Renvoie l'animal au sommet du podium
     * @return l'animal situé au sommet
     */
    public Animal getSommet() {
        return getAnimal(podium.size()-1);
    }

    /**
     * Ajouter l'animal au podium
     * @param e l'animal à ajouter
     */
    public void ajouter(Animal e) {
        podium.add(e);
    }

    /**
     * Retirer l'élément i du podium
     * @param i position i
     */
    public void retirer(int i) {
        podium.remove(i);
    }

    /**
     * Renvoie le podium
     * @return podium
     */
    public ArrayList<Animal> getPodium(){
```

```
        return podium;
    }
}
```

```
package crazycircus;

public enum Couleur{BLEU, ROUGE};
```

```
package crazycircus;

/**
* @file Animal.java
* @brief Entête de la classe Animaux
*/

public enum Animal {
    OURS,
    LION,
    ELEPHANT
}
```

```
package crazycircus;
/**
 * @file Paquets.java
 * @brief Entête de la classe Paquets
 */

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

public class Paquet {
    private ArrayList<Plateau> paquet;

    /**
     * Construit le paquet de 24 cartes mélangé, une carte est un podium
     */
    public Paquet() {

        Animal elephant = Animal.ELEPHANT;
        Animal ours = Animal.OURS;
        Animal lion = Animal.LION;

        paquet = new ArrayList<>();
        Plateau p1 = new Plateau();
```

```
        p1.getPodium(Couleur.BLEU).ajouter(elephant);
        p1.getPodium(Couleur.BLEU).ajouter(ours);
        p1.getPodium(Couleur.BLEU).ajouter(lion);
        paquet.add(p1);

        Plateau p2 = new Plateau();
        p2.getPodium(Couleur.BLEU).ajouter(elephant);
        p2.getPodium(Couleur.BLEU).ajouter(lion);
        p2.getPodium(Couleur.BLEU).ajouter(ours);
        paquet.add(p2);

        Plateau p3 = new Plateau();
        p3.getPodium(Couleur.BLEU).ajouter(lion);
        p3.getPodium(Couleur.BLEU).ajouter(ours);
        p3.getPodium(Couleur.BLEU).ajouter(elephant);
        paquet.add(p3);

        Plateau p4 = new Plateau();
        p4.getPodium(Couleur.BLEU).ajouter(lion);
        p4.getPodium(Couleur.BLEU).ajouter(elephant);
        p4.getPodium(Couleur.BLEU).ajouter(ours);
        paquet.add(p4);

        Plateau p5 = new Plateau();
        p5.getPodium(Couleur.BLEU).ajouter(ours);
        p5.getPodium(Couleur.BLEU).ajouter(elephant);
        p5.getPodium(Couleur.BLEU).ajouter(lion);
        paquet.add(p5);

        Plateau p6 = new Plateau();
        p6.getPodium(Couleur.BLEU).ajouter(ours);
        p6.getPodium(Couleur.BLEU).ajouter(lion);
        p6.getPodium(Couleur.BLEU).ajouter(elephant);
        paquet.add(p6);

        Plateau p7 = new Plateau();
        p7.getPodium(Couleur.ROUGE).ajouter(elephant);
        p7.getPodium(Couleur.ROUGE).ajouter(ours);
        p7.getPodium(Couleur.ROUGE).ajouter(lion);
        paquet.add(p7);

        Plateau p8 = new Plateau();
        p8.getPodium(Couleur.ROUGE).ajouter(elephant);
        p8.getPodium(Couleur.ROUGE).ajouter(lion);
        p8.getPodium(Couleur.ROUGE).ajouter(ours);
        paquet.add(p8);

        Plateau p9 = new Plateau();
        p9.getPodium(Couleur.ROUGE).ajouter(lion);
```

```
        p9.getPodium(Couleur.ROUGE).ajouter(ours);
        p9.getPodium(Couleur.ROUGE).ajouter(elephant);
        paquet.add(p9);

        Plateau p10 = new Plateau();
        p10.getPodium(Couleur.ROUGE).ajouter(lion);
        p10.getPodium(Couleur.ROUGE).ajouter(elephant);
        p10.getPodium(Couleur.ROUGE).ajouter(ours);
        paquet.add(p10);

        Plateau p11 = new Plateau();
        p11.getPodium(Couleur.ROUGE).ajouter(ours);
        p11.getPodium(Couleur.ROUGE).ajouter(elephant);
        p11.getPodium(Couleur.ROUGE).ajouter(lion);
        paquet.add(p11);

        Plateau p12 = new Plateau();
        p12.getPodium(Couleur.ROUGE).ajouter(ours);
        p12.getPodium(Couleur.ROUGE).ajouter(lion);
        p12.getPodium(Couleur.ROUGE).ajouter(elephant);
        paquet.add(p12);

        Plateau p13 = new Plateau();
        p13.getPodium(Couleur.BLEU).ajouter(elephant);
        p13.getPodium(Couleur.BLEU).ajouter(lion);
        p13.getPodium(Couleur.ROUGE).ajouter(ours);
        paquet.add(p13);

        Plateau p14 = new Plateau();
        p14.getPodium(Couleur.BLEU).ajouter(elephant);
        p14.getPodium(Couleur.BLEU).ajouter(ours);
        p14.getPodium(Couleur.ROUGE).ajouter(lion);
        paquet.add(p14);

        Plateau p15 = new Plateau();
        p15.getPodium(Couleur.BLEU).ajouter(lion);
        p15.getPodium(Couleur.BLEU).ajouter(ours);
        p15.getPodium(Couleur.ROUGE).ajouter(elephant);
        paquet.add(p15);

        Plateau p16 = new Plateau();
        p16.getPodium(Couleur.BLEU).ajouter(lion);
        p16.getPodium(Couleur.BLEU).ajouter(elephant);
        p16.getPodium(Couleur.ROUGE).ajouter(ours);
        paquet.add(p16);

        Plateau p17 = new Plateau();
        p17.getPodium(Couleur.BLEU).ajouter(ours);
        p17.getPodium(Couleur.BLEU).ajouter(elephant);
```

```
        p17.getPodium(Couleur.ROUGE).ajouter(lion);
        paquet.add(p17);

        Plateau p18 = new Plateau();
        p18.getPodium(Couleur.BLEU).ajouter(ours);
        p18.getPodium(Couleur.BLEU).ajouter(lion);
        p18.getPodium(Couleur.ROUGE).ajouter(elephant);
        paquet.add(p18);

        Plateau p19 = new Plateau();
        p19.getPodium(Couleur.BLEU).ajouter(elephant);
        p19.getPodium(Couleur.ROUGE).ajouter(lion);
        p19.getPodium(Couleur.ROUGE).ajouter(ours);
        paquet.add(p19);

        Plateau p20 = new Plateau();
        p20.getPodium(Couleur.BLEU).ajouter(elephant);
        p20.getPodium(Couleur.ROUGE).ajouter(ours);
        p20.getPodium(Couleur.ROUGE).ajouter(lion);
        paquet.add(p20);

        Plateau p21 = new Plateau();
        p21.getPodium(Couleur.BLEU).ajouter(lion);
        p21.getPodium(Couleur.ROUGE).ajouter(ours);
        p21.getPodium(Couleur.ROUGE).ajouter(elephant);
        paquet.add(p21);

        Plateau p22 = new Plateau();
        p22.getPodium(Couleur.BLEU).ajouter(lion);
        p22.getPodium(Couleur.ROUGE).ajouter(elephant);
        p22.getPodium(Couleur.ROUGE).ajouter(ours);
        paquet.add(p22);

        Plateau p23 = new Plateau();
        p23.getPodium(Couleur.BLEU).ajouter(ours);
        p23.getPodium(Couleur.ROUGE).ajouter(elephant);
        p23.getPodium(Couleur.ROUGE).ajouter(lion);
        paquet.add(p23);

        Plateau p24 = new Plateau();
        p24.getPodium(Couleur.BLEU).ajouter(ours);
        p24.getPodium(Couleur.ROUGE).ajouter(lion);
        p24.getPodium(Couleur.ROUGE).ajouter(elephant);
        paquet.add(p24);

        melanger();
    }

    /**
```

```
     * Mélange le paquet
     */
    public void melanger() {
        Collections.shuffle(paquet);
    }

    /**
     * Revoi le nombre de carte du paquet
     * @return la taille du paquet
     */
    public int getSizePaquet() {
        return paquet.size();
    }

    /**
     * Pioche une carte du paquet
     * @param indice la position d'une carte
     * @return une carte du paquet à la position indice
     */
    public Plateau getCarte(int indice) {
        return paquet.get(indice);
    }
}
```

```
package crazycircus;
/**
 * @file Joueur.java
 * @brief Entête de la classe Joueur
 */

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;


public class Joueur {
    private String nomJoueur;
    private int scoreJoueur;
    private boolean aDejaJouer;

    /**
     * Construie l'objet joueur avec un nom defini
     * @param nom du joueur
     */
    public Joueur(String nom) {
        this.nomJoueur = nom;
        this.scoreJoueur = 0;
```

```
        this.aDejaJouer=false;
    }
    /**
     * Ajoute le joueur dans le tableau de joueurs
     * @param joueurs l'ArrayList contenant les joueurs
     * @param nomJoueur le nom du joueur a ajouter
     */
    public static void creerJoueur(ArrayList<Joueur> joueurs, String
nomJoueur) {
        Joueur j = new Joueur(nomJoueur);
        joueurs.add(j);
    }
    /**
     * Verifie la presence du joueur dans le tableau de joueurs
     * @param joueurs l'ArrayList contenant les joueurs
     * @param nom le nom du joueur a ajouter
     * @return un boolean en fonction de la verification
     */
    public static boolean verifJoueur(ArrayList<Joueur> joueurs, String nom)
{
        for(int i = 0; i < joueurs.size(); ++i) {
            if(nom.equals(joueurs.get(i).nomJoueur))
                return true;
        }
        return false;
    }
    /**
     * Ajoute 1 point au joueur
     */
    public void ajouterPoint() {
        ++scoreJoueur;
    }
    /**
     * Renvoi le score du joueur
     * @return scoreJoueur du joueur
     */
    public int getScore() {
        return scoreJoueur;
    }
    /**
     * Renvoi le nom du joueur
     * @return nomJoueur du joueur
     */
    public String getNom() {
        return nomJoueur;
    }
    /**
     * Renvoi la situation du joueur
     * @return un boolean aDejaJoueur
```

```
     */
    public boolean getADejaJouer() {
        return aDejaJouer;
    }
    /**
     * Defini le joueur comme joué
     */
    public void setADejaJouer() {
        aDejaJouer = true;
    }
    /**
     *  Defini le joueur comme pas joué
     */
    public void setAPasJouer() {
        aDejaJouer = false;
    }
    /**
     * Renvoi la position/l'indice du joueur dans le tableau de joueur
     * @param joueurs l'ArrayList contenant les joueurs
     * @param nomJ le nom du joueur a ajouter
     * @return l'indice indice du joueur
     */
    public static int getIndice(ArrayList<Joueur> joueurs, String nomJ) {
        int indice = 0;
        for(int i = 0; i < joueurs.size(); ++i) {
            if(nomJ.equals(joueurs.get(i).nomJoueur)){
                indice = i;
                break;
            }
        }
        return indice;
    }
    /**
     * Reinisialise la situation de tout les joueurs à pas jouer
     * @param joueurs l'ArrayList contenant les joueurs
     */
    public static void resetADejaJouer(ArrayList<Joueur> joueurs) {
        for(int i = 0; i < joueurs.size(); i++) {
            joueurs.get(i).setAPasJouer();
        }
    }
    /**
     * Trie le classement par score décroissant et par ordre alphabétique en
cas d’égalité
     * @param joueurs l'ArrayList contenant les joueurs
     */
    public final static void triClassement(ArrayList<Joueur> joueurs) {
        List<Joueur> listeJoueurs = joueurs;
```

```
        listeJoueurs.sort(Comparator.comparing(Joueur::getScore).reversed().
thenComparing(Joueur::getNom));
    }
    /**
     * Affiche le classement final
     * @param joueurs l'ArrayList contenant les joueurs
     */
    public static final void afficherClassement(ArrayList<Joueur>joueurs) {
        System.out.println("\n---------------- CLASSEMENT ----------------
\n");
        for(int i = 0; i < joueurs.size(); i++) {
            if(i == 0) {
                System.out.println((i+1) + "er : " + joueurs.get(i).getNom()
+
                        " - " + joueurs.get(i).getScore() + " points ");
            }
            else {
                System.out.println((i+1) + "ème : " +
joueurs.get(i).getNom() +
                        " - " + joueurs.get(i).getScore() + " point" +
(joueurs.get(i).getScore() <= 1 ? "" : "s"));
            }
        }
    }

}
```