Rapport de Projet – Crazy Circus

Base de la programmation orienté objet

16 MARS

Yotam WEBER, Yohann PECH | GR109

Université de Paris

Table des matières

I. Introduction

Ce projet de développement orienté objet correspond à notre 3ème travail en groupe en rapport à la programmation, mais il est notre premier en Java.
L'objectif du projet était de réaliser un jeu qui permettrait l'affrontement de dompteurs.

Le jeu est constitué de 2 podiums – l'un rouge, l'un bleu -, 3 animaux – un ours, un éléphant et un lion – ainsi que de 24 cartes objectifs représentants la situation de podiums à obtenir à chaque tour.

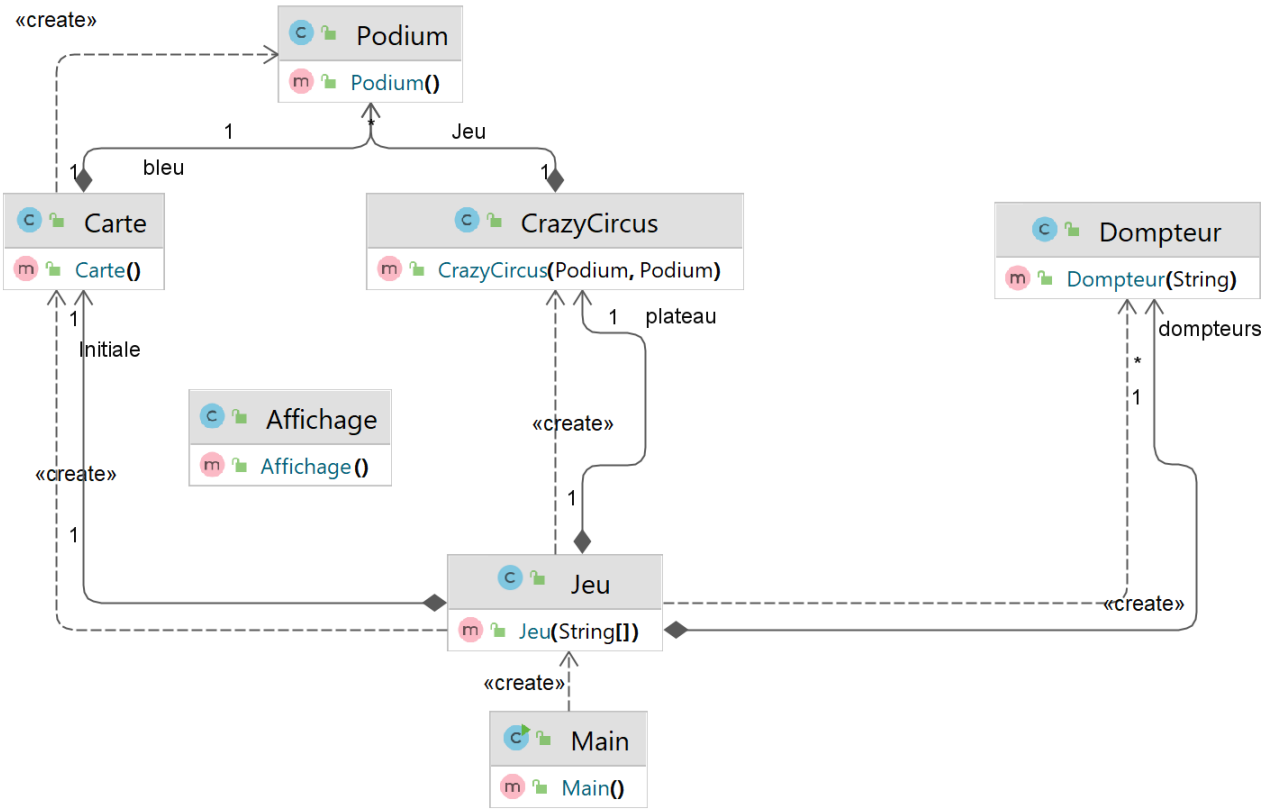
Le but était de former le plus rapidement une combinaison de déplacements bien spécifique de manière à obtenir une situation finale de podium à partir d'une carte piochée.

Un dompteur gagne la manche lorsqu'il trouve en premier la bonne combinaison ; et il récupère la carte. S'il joue et donne une mauvaise combinaison, alors il doit attendre la prochaine manche.
S'il ne reste qu'un joueur a joué, alors il gagne automatiquement la manche ainsi que la carte.

Le jeu se termine lorsque toutes les 24 cartes ont été jouées. Le dompteur ayant obtenu le plus de cartes gagne la partie.

Les différentes combinaisons de déplacements possibles sont :
KI : Permet de déplacer l'animal qui est en haut du podium bleu vers le sommet du podium rouge
LO : Permet de déplacer l'animal qui est en haut du podium rouge vers le sommet du podium bleu
SO : Permet d'inverser les animaux se trouvant sur les sommets des 2 podiums
NI : Permet de déplacer l'animal se trouvant en bas du podium bleu, tout en haut du même podium
MA : Permet de déplacer l'animal se trouvant en bas du podium rouge, tout en haut du même podium

II. Diagramme UML



III. Tests Unitaires des classes

Pour vérifier et tester notre jeu, nous avons crée un code de test unitaire pour :

- Les 24 cartes :

```
public class TestCartes {
    @Test
    public void TestCartes(){
        String []args = {"yoyo","toto"};
        Jeu j = new Jeu(args);
        Carte C = new Carte();
        C.generateCarteObjectif();
        assertFalse(Carte.getCartes().size()==23);
        assertTrue(Carte.getCartes().size()==24);
        Carte C1 = Jeu.CarteAleat();
        Carte C2 = j.piocheAutreCarte(C1);
        assertNotEquals(C1, C2);
        assertFalse(Carte.getCartes().size()==24);
        assertTrue(Carte.getCartes().size()<24);

    }
}
```

- Les dompteurs :

```
public class TestDompteurs {
    @Test
    public void testDompteurPeutJouer(){
        String []args = new String[]{"yoyo", "toto"};
        Jeu j =new Jeu(args);
        assertNotNull(j.getDompteurs("yoyo"));
        assertNotNull(j.getDompteurs("toto"));
        assertNull(j.getDompteurs("popo"));
        assertTrue(j.getDompteurs("yoyo").PeutJouer()&&
j.getDompteurs("toto").PeutJouer());
        j.getDompteurs("yoyo").AJoue();
        j.getDompteurs("toto").AJoue();
        assertFalse(j.getDompteurs("yoyo").PeutJouer() &&
j.getDompteurs("toto").PeutJouer());
        j.getDompteurs("yoyo").setPeutJouerTrue();
        j.getDompteurs("toto").setPeutJouerTrue();
        assertTrue(j.getDompteurs("yoyo").PeutJouer()&&
j.getDompteurs("toto").PeutJouer());
        assertEquals(j.getDompteurs("yoyo").getNomDompteur(),
"yoyo");
        assertNotEquals(j.getDompteurs("yoyo").getNomDompteur(),
"toto");
        assertTrue(j.compareScores(j.getDompteurs("yoyo"),
j.getDompteurs("toto"))==0);
        assertFalse(j.comparePseudos(j.getDompteurs("yoyo"),
j.getDompteurs("toto"))==0);
        j.getDompteurs("yoyo").AJoue();
```

```
        assertNotNull(j.dernierDompteurAJouer());
        j.getDompteurs("toto").AJoue();
        assertNull(j.dernierDompteurAJouer());
        j.getDompteurs("yoyo").AGagne();

assertTrue(j.compareScores(j.getDompteurs("yoyo"),j.getDompteurs("to
to"))==-1);
    }
}
```

- Les poduims :

```
public class TestPodium {
    @Test
    public void testPodium(){
        //Création d'une carte Objectif
        Podium pBObj = new Podium();
        Podium pRObj = new Podium();
        pBObj.getAnimaux().add("OURS");
        pBObj.getAnimaux().add("ELEPHANT");
        pBObj.getAnimaux().add(Affichage.vide);
        pRObj.getAnimaux().add("LION");
        pRObj.getAnimaux().add(Affichage.vide);
        pRObj.getAnimaux().add(Affichage.vide);
        Carte CObj = new Carte();
        CObj.setBleu(pBObj);
        CObj.setRouge(pRObj);

        //Création de 2 podiums
        Podium pB = new Podium();
        Podium pR = new Podium();
        pR.getAnimaux().add("OURS");
        pR.getAnimaux().add("LION");
        pR.getAnimaux().add("ELEPHANT");
        pB.getAnimaux().add(Affichage.vide);
        pB.getAnimaux().add(Affichage.vide);
        pB.getAnimaux().add(Affichage.vide);
        Carte test = new Carte();
        test.setRouge(pR);;
        test.setBleu(pB);

        CrazyCircus Crz = new CrazyCircus(pB, pR);

        assertFalse(Podium.verifComparaisonPodiums(CObj, Crz));
        String combinaison = "KIMALO";
        assertTrue(CrazyCircus.ordreValide(combinaison));
        Crz.deplacer(test, combinaison);
        assertTrue(Podium.verifComparaisonPodiums(test, Crz));

    }
}
```

IV. Code Source du Projet

```java
package main;
/**
 * main
 *
 * @author WEBER Yotam & PECH Yohann
 * @version 2.2 02/03/2023
 */
import affichage.Affichage;
import jeu.Jeu;

public class Main {
    public static void main(String[] args) {

        // On cree le jeu avec les cartes et les dompteurs
        Jeu j = new Jeu(args);
        System.out.println(Affichage.debutJeu());
        j.jouerDompteurs(Jeu.choixDifficulte());

    }
}
```

```java
package affichage;

import jeu.Jeu;
import elements.*;
import dompteur.Dompteur;
import java.util.ArrayList;

/**
 * Enum?re les diff?rents animaux existants
 *
 * @author WEBER Yotam & PECH Yohann
 * @version 2.2 02/03/2023
 */
public class Affichage {

    public static final String border = "----------------------------------
--------\n";

    public static final String vide = "        ";

    /**
     * Permet l'affichage ? la console de la situation de jeu
     *
     * @return un String contenant tout l'affichage du jeu
     */
    public static String displayJeu(Carte carteInit, Carte carteObj) {
        StringBuilder sJeu = new StringBuilder();

        String separator = "  ----  ";

        displayManche(sJeu);

        for (int i = 2; i >= 0; --i) {

            sJeu.append(carteInit.getBleu().getAnimaux().get(i));
            sJeu.append(" ");

            sJeu.append(carteInit.getRouge().getAnimaux().get(i));
            sJeu.append("     ");

            sJeu.append(carteObj.getBleu().getAnimaux().get(i));
            sJeu.append(" ");

            sJeu.append(carteObj.getRouge().getAnimaux().get(i));

            sJeu.append(System.lineSeparator());
        }

        for (int i = 0; i < 2; ++i)
            sJeu.append(separator).append(" ");

        sJeu.append("==>");

        for (int i = 0; i < 2; ++i)
            sJeu.append(" ").append(separator);

        sJeu.append(System.lineSeparator());

        sJeu.append("  BLEU     ROUGE         BLEU     ROUGE");

        sJeu.append(System.lineSeparator());
```

```
        sJeu.append(border);

        sJeu.append("KI : BLEU --> ROUGE    NI : BLEU  ^");
        sJeu.append(System.lineSeparator());
        sJeu.append("LO : BLEU <-- ROUGE    MA : ROUGE ^");
        sJeu.append(System.lineSeparator());
        sJeu.append("SO : BLEU <-> ROUGE");
        sJeu.append(System.lineSeparator().repeat(2));
        sJeu.append(border);
        return sJeu.toString();
    }

    /**
     *
     * @param sJeu le StrinBuilder auquel on va ajouter des ?l?ments
     */
    public static void displayManche(StringBuilder sJeu) {
        sJeu.append(border);
        sJeu.append("                ");
        sJeu.append("MANCHE ").append((23 - Carte.getCartes().size()) + 1);
        sJeu.append(System.lineSeparator());
        if (Carte.getCartes().size() == 23)
            sJeu.append("Nombre de cartes dans la pioche avant la manche :
").append(Carte.getCartes().size() + 1);
        else {
            sJeu.append("Nombre de cartes dans la pioche avant la manche :
").append(Carte.getCartes().size());
        }
        sJeu.append(System.lineSeparator());
        sJeu.append(border);
    }

    /**
     * Permet l'affichage du classement des Dompteurs
     *
     * @return une String contenant l'affichage
     * @see Jeu#ordreDompteurs()
     */
    public static String displayLeaderboard(ArrayList<Dompteur> dompteurs)
{
        StringBuilder sLeaderboard = new StringBuilder();
        sLeaderboard.append(border);
        if (dompteurs.size() != 1) {
            int classement = 1;
            for (Dompteur d : dompteurs) {
                sLeaderboard.append(classement);
                sLeaderboard.append("- ");
                sLeaderboard.append(d.getNomDompteur());
                sLeaderboard.append(" ");
                sLeaderboard.append(d.getScoreDompteur());
                if (d.getScoreDompteur() <= 1)
                    sLeaderboard.append(" carte.");
                else {
                    sLeaderboard.append(" cartes.");
                }
                sLeaderboard.append(System.lineSeparator());
                classement++;
            }

        } else
            sLeaderboard.append("Fin de l'entrainement !");
```

```
        return sLeaderboard.toString();
    }

    /**
     *
     * @param dompteurs
     * @return
     */
    public static String affichageGagnant(String dompteurs) {
        StringBuilder sLeaderboard = new StringBuilder();
        sLeaderboard.append(dompteurs);
        sLeaderboard.append(" a gagne la manche !");
        return sLeaderboard.toString();
    }

    /**
     *
     * @param dompteurs
     * @return
     */
    public static String affichageFinDuJeu(String dompteurs) {
        StringBuilder sLeaderboard = new StringBuilder();
        sLeaderboard.append("Felicitations ! ");
        sLeaderboard.append(dompteurs);
        sLeaderboard.append(" a gagne la partie !");

        return sLeaderboard.toString();
    }

    /**
     *
     * @return
     */
    public static String debutJeu() {
        StringBuilder sb = new StringBuilder();
        sb.append(
                " ######   ########     ###    ######## ##     ##      ######
#### ########   ######   ##     ##  ######  \n");
        sb.append(
                "##    ##  ##     ##   ## ##        ##   ##  ##     ##
##   ##  ##     ## ##    ##  ##     ## ##    ## \n");
        sb.append(
                "##        ##     ##  ##   ##      ##     ####      ##
##  ##     ## ##        ##     ## ##       \n");
        sb.append(
                "##        ########  ##     ##    ##       ##       ##
##  ########  ##        ##     ##  ######  \n");
        sb.append(
                "##        ##   ##   #########   ##        ##       ##
##  ##   ##   ##        ##     ##       ## \n");
        sb.append(
                "##    ##  ##    ##  ##     ##  ##        ##       ##
##   ##  ##    ##  ##    ##  ##     ## ##    ## \n");
        sb.append(
                " ######   ##     ## ##     ## ########    ##        ######
#### ##     ##  ######    #######   ######  \n");

        sb.append(System.lineSeparator());

        sb.append("Demarrage du jeu en cours...\n");
```

```
            sb.append("Saississez vos fouets et la scene est a vous ! ");
            sb.append(System.lineSeparator());

            return sb.toString();
        }
        /**
         * Permet d'ajouter un dompteur au jeu
         *
         * @param nomDeScene le nom du dompteur
         * @return un String d?crivant ce qu'a fait la m?thode
         * @see fctDompteur#getDompteur()
         * @see fctDompteur#dompteurExists(String)
         */
        /*
         * public String addDompteur(String nomDeScene) {
         * fctDompteur fctD = new fctDompteur();
         * if (!(fctD.dompteurExists(nomDeScene))) {
         * fctD.getDompteur().add(new Dompteur(nomDeScene));
         * return "Dompteur <" + nomDeScene + "> ajout?.";
         * } else
         * return "Le dompteur <" + nomDeScene + "> existe d?j?.";
         * }
         */

}
```

```java
package dompteur;
/**
 * Les différents dompteurs
 *
 * @author WEBER Yotam & PECH Yohann
 * @version 2.2 02/03/2023
 */
public class Dompteur {
    private String NomDompteur; // Le nom du dompteur
    private int scoreDompteur; // Le nombre de cartes du dompteur
    private boolean PeutJouer; // Le dompteur peut jouer dans la manche

    /**
     * Constructor
     * @param nomDompteur - le nom du nouveau dompteur
     */
    public Dompteur(String nomDompteur) {
        this.NomDompteur = nomDompteur;
        this.scoreDompteur = 0;
        PeutJouer = true;
    }
/**
 * getNomDompteur - permet d'avoir le nom du compteur
 * @return NomDompteur - le nom du dompteur
 */
    public String getNomDompteur() {
        return NomDompteur;
    }
/**
 * getScoreDompteur - permet d'avoir le nombre de cartes du dompteur
 * @return scoreDompteur - le nombre de cartes du dompteur
 */
    public int getScoreDompteur() {
        return this.scoreDompteur;
    }
/**
 * Le Dompteur courant ne peut plus jouer
 * On met le booleén à false
 */
    public void AJoue() {
        PeutJouer = false;
    }
/**
 * Le dompteur a gangné la manche
 * Il gagne 1 point
 */
    public void AGagne() {
        scoreDompteur++;
    }
/**
 * Le booleen PeutJouer
 * @return PeutJouer
 */
    public boolean PeutJouer() {
        return PeutJouer;
    }
/**
 * Nouvelle Manche, le dompteur peut jouer
 * On met le booleen à true
 */
    public void setPeutJouerTrue() {
```

```
            PeutJouer = true;
        }
    }
```

```java
package elements;

import java.util.ArrayList;
import java.util.Arrays;
import affichage.Affichage;

/**
 * Classe déterminant une Carte. Une carte contient deux Podiums
 *
 * @author WEBER Yotam & PECH Yohann
 * @version 2.2 02/03/2023
 */
public class Carte {

    private Podium bleu; // Le podium BLeu du jeu
    private Podium rouge; // Le podium Rouge du jeu
    private static ArrayList<Carte> cartes; // ArrayList de Carte

    /**
     * Constuctor de Carte
     */
    public Carte() {
        this.bleu = new Podium();
        this.rouge = new Podium();
    }

    /**
     * Génère les 24 cartes possibles du jeu
     * @see Carte
     * @see Podium
     */
    public void generateCarteObjectif() {

        String[][] combinaisons = {
                { "  LION  ", "ELEPHANT", "  OURS  " },
                { "  LION  ", "  OURS  ", "ELEPHANT" },
                { "ELEPHANT", "  LION  ", "  OURS  " },
                { "ELEPHANT", "  OURS  ", "  LION  " },
                { "  OURS  ", "ELEPHANT", "  LION  " },
                { "  OURS  ", "  LION  ", "ELEPHANT" }
        };

        cartes = new ArrayList<>();

        for (String[] combin : combinaisons) {
            Carte tmp1 = new Carte();
            Carte tmp2 = new Carte();

            ArrayList<String> Animaux = new
ArrayList<>(Arrays.asList(combin));
            tmp1.getBleu().getAnimaux().addAll(Animaux);
            for (int i = 0; i < 3; i++)
                tmp1.getRouge().getAnimaux().add(Affichage.vide);

            tmp2.getBleu().getAnimaux().addAll(Animaux.subList(0, 2));
            tmp2.getBleu().getAnimaux().add(Affichage.vide);

            tmp2.getRouge().getAnimaux().addAll(Animaux.subList(2, 3));

            tmp2.getRouge().getAnimaux().add(Affichage.vide);
            tmp2.getRouge().getAnimaux().add(Affichage.vide);
```

```
            Carte carteEchange1 = new Carte();
            Carte carteEchange2 = new Carte();


carteEchange1.getBleu().getAnimaux().addAll(tmp1.getRouge().getAnimaux());

carteEchange1.getRouge().getAnimaux().addAll(tmp1.getBleu().getAnimaux());

carteEchange2.getBleu().getAnimaux().addAll(tmp2.getRouge().getAnimaux());

carteEchange2.getRouge().getAnimaux().addAll(tmp2.getBleu().getAnimaux());

            cartes.add(tmp1);
            cartes.add(carteEchange1);
            cartes.add(tmp2);
            cartes.add(carteEchange2);
        }
    }

    /**
     * getCarte : permet d'obtenir la carte du paquet
     * @see Carte
     * @param numeroCarte la position de la carte dans le paquet
     * @return la carte du paquet
     */
    public static Carte getCarte(int numeroCarte) {
        return cartes.get(numeroCarte);
    }

    /**
     * getCartes : permet d'obtenir le paquuet de cartes
     * @see Carte
     * @return le paquet de cartes
     */
    public static ArrayList<Carte> getCartes() {
        return cartes;
    }

    /**
     * getBleu : permet d'obtenir le podium Bleu
     * @return le podium bleu
     * @see Carte#bleu
     * @see Podium
     */
    public Podium getBleu() {
        return bleu;
    }

    /**
     * setBleu : permet de mettre en place le podium Bleu
     * @param bleu le nouveau podium bleu
     * @see Carte#bleu
     * @see Podium
     */
    public void setBleu(Podium bleu) {
        this.bleu = bleu;
    }

    /**
     * getRouge : permet d'obtenir le podiuù Rouge
```

```
     * @return le podium rouge
     * @see Carte#rouge
     * @see Podium
     */
    public Podium getRouge() {
        return this.rouge;
    }

    /**
     * setRouge : permet de mettre en place la podiuum Rouge
     * @param rouge le nouveau podium rouge
     * @see Carte#rouge
     * @see Podium
     */
    public void setRouge(Podium rouge) {
        this.rouge = rouge;
    }
}
```

```java
package elements;
/**
 * Définition des poduims
 *
 * @author WEBER Yotam & PECH Yohann
 * @version 2.2 02/03/2023
 */
import jeu.CrazyCircus;

import java.util.ArrayList;

public class Podium {
    private ArrayList<String> Podiums; // Un podium - ArrayList de Sting

    /**
     * Constructor de podium
     */
    public Podium() {
        Podiums = new ArrayList<>();
    }

    /**
     * getAnimaux : permet d'obtnir le podium d'Animaux
     * @return le podium d'animaux
     */
    public ArrayList<String> getAnimaux() {
        return this.Podiums;
    }
    /**
     * verifComparaisonPodiums - Permet de verifier si les podiums de la
carte Objectif sont egaux a ceux du plateau
     * @return true s'ils sont egaux, sinon false
     */
    public static boolean verifComparaisonPodiums(Carte Objectif,
CrazyCircus plateau) {
        return
Objectif.getRouge().getAnimaux().equals(plateau.getPodiumRouge().getAnimaux
()) &&

Objectif.getBleu().getAnimaux().equals(plateau.getPodiumBleu().getAnimaux()
);
    }
}
```

```java
package jeu;
/**
 * Définit la base du jeu
 *
 * @author WEBER Yotam & PECH Yohann
 * @version 2.2 02/03/2023
 */
import elements.*;
import affichage.Affichage;
import dompteur.Dompteur;

import java.util.ArrayList;
import java.util.Scanner;

public class Jeu {

    private ArrayList<Dompteur> dompteurs; // ArrayList de Dompteur

    private Carte Initiale; // Carte initiale contenant la situation iniale
à chaque tour

    private Carte Objectif; // Carte objectif qui contient la situation
finale à chaque tour

    private CrazyCircus plateau; // le plateau qui contient les podiums

    /**
     * Constructor de Jeu - permet de creer le jeu( les cartes initiales et
objectifs, inscrit les dompteurs)
     * @param args Tableau contenant tout les dompteurs
     */
    public Jeu(String[] args) {
        dompteurs = new ArrayList<>();
        Initiale = new Carte();
        Objectif = new Carte();
        plateau = new CrazyCircus(Initiale.getBleu(), Initiale.getRouge());
        int compteurDompteurs=0;
        for (String s : args) {
            Dompteur d = new Dompteur(s);
            dompteurs.add(d);
            if(!DompteurUnique(args, s)){
                System.out.println("Erreur dans le systeme, il ne peut pas
y avoir 2 fois le meme dompteur.");
                System.out.println("Relancez le jeu avec tous les prenoms
des dompteurs differents");
                System.exit(0);
            }
            compteurDompteurs++;
        }
        if(compteurDompteurs==1){
            System.out.println("Malheureusement, il ne peut pas y avoir 1
seul dompteur");
            System.out.println("Relancez le jeu avec au moins 2
dompteurs.");
            System.exit(0);
        }else if(compteurDompteurs==0) {
           System.out.println("Aucun dompteur est inscrit dans le jeu.");
            System.out.println("Relancez le jeu avec au moins 2
dompteurs.");
            System.exit(0);
        }
```

```
        Initiale.generateCarteObjectif();
        premierTour();
    }

    /**
     * DompteurUnique - Verifie si un dompteur n'existe pas 2 fois
     * @param args les joueurs
     * @param s le joueur a verifier
     * @return true si le joueur existe 1 seule fois, sinon false
     */
    public static boolean DompteurUnique(String[] args, String s){
        int compteur = 0;
        for(int i = 0; i < args.length; i++){
            if(args[i].equals(s))
                compteur++;
        }
        return compteur==1;
    }

    /**
     * CarteAleat - permet d'obtenir aléatoirement une carte dans le paquet
     * @return la carte tirée
     * @see Carte
     */
    public static Carte CarteAleat() {
        return Carte.getCarte((int) (Math.random() *
Carte.getCartes().size()));
    }

    /**
     * piocheAutreCarte - Pioche une carte objectif différente de celle
entree en paramètre
     * @param tmp Carte
     * @return la carte objectif tiree
     * @see Jeu#CarteAleat()
     * @see Carte#getCartes()
     */
    public Carte piocheAutreCarte(Carte tmp) {
        this.Objectif = CarteAleat();
        while (this.Objectif.equals(tmp)) {
            this.Objectif = CarteAleat();

        }
        Carte.getCartes().remove(Objectif);
        return this.Objectif;
    }

    /**
     * premierTour - Tire les deux premieres cartes du jeu
     * @see Jeu#CarteAleat()
     * @see Jeu#piocheAutreCarte(Carte)
     */
    public void premierTour() {
        this.Initiale = CarteAleat();
        this.Objectif = piocheAutreCarte(this.Initiale);
    }

    /**
     * nouvelleManche - Genere une nouvelle manche
     * @param FinManche indique comment s'est terminee la manche
     * si le dompteur a gagne en proposant une combinaison ou pas
```

```
     * @see Jeu#piocheAutreCarte(Carte)
     * @see Dompteur#setPeutJouerTrue()
     */
    public void nouvelleManche(int FinManche) {
        if (FinManche == 1) {
            this.Initiale = this.Objectif;
        }
        this.Objectif = piocheAutreCarte(this.Initiale);
        for (Dompteur d : dompteurs) {
            d.setPeutJouerTrue();
        }
    }

    /**
     * copyInitiale - Permet de copier la carte initiale pour ensuite
simuler une commande
     * @return copy la copie de la carte initiale
     * @see Carte
     * @see Jeu#Initiale
     */
    public Carte copyInitiale() {
        Carte copy = new Carte();
        for (int i = 0; i < 3; ++i) {

copy.getBleu().getAnimaux().add(this.Initiale.getBleu().getAnimaux().get(i)
);

copy.getRouge().getAnimaux().add(this.Initiale.getRouge().getAnimaux().get(
i));
        }
        return copy;
    }

    /**
     * getDompteurs - Permet de savoir si le dompteur existe
     * @param NomDompteur le nom du dompteur
     * @return le dompteur s'il participe au jeu, un dompteur "null" sinon
     * @see Jeu#dompteurs
     * @see Dompteur
     */
    public Dompteur getDompteurs(String NomDompteur) {
        for (Dompteur d : dompteurs) {
            if (d.getNomDompteur().equals(NomDompteur)) {
                return d;
            }
        }
        return null;
    }

    /**
     * compareScores - Compare le score de deux dompteurs
     * @param d1 le premier dompteur
     * @param d2 le second dompteur
     * @return 1 si le second a un meilleur score, 0 s'ils ont le meme
score, -1 si
     *         le premier a un meilleur score
     */
    public int compareScores(Dompteur d1, Dompteur d2) {
        return Integer.compare(d2.getScoreDompteur(),
d1.getScoreDompteur());
    }
```

```
    /**
     * comparePseudos - Compare les noms de deux dompteurs
     * @param d1 le premier dompteur
     * @param d2 le second dompteur
     * @return int positif si le second se trouve avant alphanumériquement
parlant,
     *         0 s'ils ont le même nom,
     *         int negatif si le premier se trouve avant alphanumériquement
parlant
     */
    public int comparePseudos(Dompteur d1, Dompteur d2) {
        return d1.getNomDompteur().compareTo(d2.getNomDompteur());
    }

    /**
     * ordreDompteurs - Permet le tri des dompteurs en fonction de leur
score puis de leur nom
     */
    public void ordreDompteurs() {
        this.dompteurs.sort((d1, d2) -> {
            if (compareScores(d1, d2) != 0)
                return compareScores(d1, d2);
            else
                return comparePseudos(d1, d2);
        });

    }

    /**
     * dernierDompteurAJouer - Savoir si il reste un seul dompteur qui peut
jouer
     * @return le dernier dompteur qui a encore le droit de jouer, sinon
null
     */
    public Dompteur dernierDompteurAJouer() {
        int cptDompteurRestants = 0;
        for (Dompteur d : dompteurs) {
            if (d.PeutJouer()) {
                cptDompteurRestants++;
            }
        }
        if (cptDompteurRestants == 1) {
            for (Dompteur d : dompteurs) {
                if (d.PeutJouer()) {
                    return d;
                }
            }
        }
        return null;
    }

    /**
     * jouerDompteurs - Gère une partie de jeu à partir du niveau de
difficulte defini tant que le paquet de cartes n'est pas vide
     * @param niveau le niveau de jeu selectionne
     * @see Jeu#dernierDompteurAJouer()
     * @see Jeu#getDompteurs(String)
     * @see Jeu#nouvelleManche(int)
     * @see CrazyCircus#ordreValide(String)
     * @see CrazyCircus#deplacer(Carte, String)
```

```
     * @see Affichage#displayJeu(Carte, Carte)
     * @see Affichage#affichageGagnant(String)
     * @see Affichage#displayLeaderboard(ArrayList)
     * @see Affichage#affichageFinDuJeu(String)
     */
    public void jouerDompteurs(int niveau) {
        do {
            System.out.print(Affichage.displayJeu(this.Initiale,
this.Objectif));
            System.out.println("Entrez votre nom suivi de votre combinaison
: ");
            Scanner sc = new Scanner(System.in);
            String NomDompteur = sc.next();

            // on vérifie si le dompteur existe
            //if (!verifNomDompteurs(NomDompteur)) {
            if(getDompteurs(NomDompteur)==null){

                System.out.println("Le nom du dompteur indique n'existe
pas");
            } else {
                String combinaisonDompteur = sc.next();
                // on vérifie si le dompteur a deja joué ou pas
                if (!getDompteurs(NomDompteur).PeutJouer()) {
                    System.out.println("Vous avez deja joue dans cette
manche, attendez la prochaine manche.");
                } else {
                    // le dompteur n'a pas joué dans cette manche, il
propose sa combinaison
                    Carte testDompteur = copyInitiale();
                    if (niveau == 2 &&
combinaisonDompteur.toUpperCase().contains("SO")) {
                        System.out.print(Affichage.border);
                        System.out.println("La commande <<SO>> est
impossible en mode difficile.");
                    } else {
                        plateau.deplacer(testDompteur,
combinaisonDompteur.toUpperCase());
                        // Mauvaise combinaison
                        if (!Podium.verifComparaisonPodiums(Objectif,
plateau)) {
                            System.out.println(
                                    "Eh non, la combinaison n'est pas la
bonne, vous ne pouvez plus rejouer, attendez la prochaine manche ! ");
                            getDompteurs(NomDompteur).AJoue();
                            // s'il reste un seul dompteur
                            if (dernierDompteurAJouer() != null) {
                                dernierDompteurAJouer().AGagne();
                                System.out.println("Tous les autres joueurs
se sont trompes, "
                                                   +
dernierDompteurAJouer().getNomDompteur() + " remporte la manche ainsi que
la carte !");
                                System.out

.println(Affichage.affichageGagnant(dernierDompteurAJouer().getNomDompteur(
)));
                                nouvelleManche(0);
                            }
                        }
                        // bonne combinaison
```

```
                        else {
                            System.out.println("Bravo, vous avez trouve la
bonne combinaison !"
                                    +
getDompteurs(NomDompteur).getNomDompteur() + " gagne la carte !");
                            getDompteurs(NomDompteur).AGagne();

System.out.println(Affichage.affichageGagnant(getDompteurs(NomDompteur).get
NomDompteur()));
                            nouvelleManche(1);
                        }
                    }
                }

            }

            ordreDompteurs();

System.out.println(Affichage.displayLeaderboard(this.dompteurs));
        } while (!Carte.getCartes().isEmpty());


System.out.println(Affichage.affichageFinDuJeu(this.dompteurs.get(0).getNom
Dompteur()));
    }



    /**
     * choixDifficulte - Permet de choisir la difficulté du jeu
     * @return 1 si le niveau régulier est choisi, 2 si le niveau difficile
est
     *         choisi
     */
    public static int choixDifficulte() {
        System.out.print(Affichage.border);
        System.out.println("NIVEAU DU JEU");
        System.out.println("- Niveau regulier :les joueurs peuvent utiliser
toutes les cartes ordres.");
        System.out.println(
                "- Niveau difficile :les joueurs peuvent utiliser toutes
les cartes ordres sauf la carte << SO >>.");
        System.out.print(Affichage.border);
        Scanner sc = new Scanner(System.in);
        System.out.print("Choisissez le niveau de difficulte du jeu ( R
pour Regulier / D pour Difficile ) : ");
        String niveau = sc.next();
        int tmp;
        switch (niveau.toUpperCase()) {
            case "R":
                tmp = 1;
                System.out.println("Vous avez choisi le niveau regulier");
                break;
            case "D":
                tmp = 2;
                System.out.println("Vous avez choisi le niveau difficile");
                break;
            default:
                tmp = 1;
                System.out.println(
                        "La lettre saisie ne correspond a aucun niveau, le
```

```
niveau regulier est choisi par defaut.");
                break;
        }
        return tmp;

    }

}
```

```java
package jeu;
/**
 * Les éléments du jeu Crazy Circus
 *
 * @author WEBER Yotam & PECH Yohann
 * @version 2.2 02/03/2023
 */
import affichage.Affichage;
import elements.*;

import java.util.ArrayList;

public class CrazyCircus {

    private ArrayList<Podium> Jeu; // Le jeu - ArrayList de Podium
    public static final int BLEU = 0; // Indice correspondant au podium
Bleu
    public static final int ROUGE = 1; // Indice correspondant au podium
Rouge

    /**
     * Constructor de CrazyCircus
     * @param Bleu Le podium Bleu
     * @param Rouge Le podium Rouge
     */
    public CrazyCircus(Podium Bleu, Podium Rouge) {
        Jeu = new ArrayList<>();
        Jeu.add(Bleu);
        Jeu.add(Rouge);
    }

    /**
     * estVide - permet de savoir si le podium courant est vide
     * @param numPod le numero du podium - 0 pour Bleu - 1 pour Rouge
     * @return 1 si le Podium courant est vide, sinon 0
     */
    public boolean estVide(int numPod) {
        return Jeu.get(numPod).getAnimaux().isEmpty();
    }

    /**
     * depiler - permet de dépiler le dernier element du podium et le
remplacer par du vide
     * @param numPod le numero du podium - 0 pour Bleu - 1 pour Rouge
     */
    public void depiler(int numPod) {
        assert !estVide(numPod);
        if (numPod == ROUGE)
            getPodiumRouge().getAnimaux().set(getIndexLastPodium(ROUGE),
Affichage.vide);
        else {
            getPodiumBleu().getAnimaux().set(getIndexLastPodium(BLEU),
Affichage.vide);
        }
    }

    /**
     * getIndexLastPodium - permet d'obtenir l'index du dernier animal du
podium courant
     * @param numPod le numero du podium - 0 pour Bleu - 1 pour Rouge
     * @return la position du dernier animal du podium
```

```
     */
    public int getIndexLastPodium(int numPod) {
        Podium tmp = Jeu.get(numPod);
        for (int i = 0; i < tmp.getAnimaux().size(); ++i)
            if (tmp.getAnimaux().get(i).equals(Affichage.vide))
                return (i - 1);

        return 2;
    }
    /**
     * KI - permet d'effectuer le déplacement KI
     */
    public void KI() {
        assert !estVide(BLEU);
        if (getIndexLastPodium(BLEU) >= 0) {
            getPodiumRouge().getAnimaux().set(getIndexLastPodium(ROUGE) +
1,

getPodiumBleu().getAnimaux().get(getIndexLastPodium(BLEU)));
            depiler(BLEU);
        }
    }

    /**
     * LO - permet d'effectuer le déplacement LO
     */
    public void LO() {
        assert !estVide(ROUGE);
        if (getIndexLastPodium(ROUGE) >= 0) {
            getPodiumBleu().getAnimaux().set(getIndexLastPodium(BLEU) + 1,

getPodiumRouge().getAnimaux().get(getIndexLastPodium(ROUGE)));
            depiler(ROUGE);
        }
    }
    /**
     * SO - permet d'effectuer le déplacement SO
     */
    public void SO() {
        assert !estVide(BLEU) && !estVide(ROUGE);
        if (getIndexLastPodium(BLEU) >= 0 && getIndexLastPodium(ROUGE) >=
0) {
            String tmp =
Jeu.get(BLEU).getAnimaux().get(getIndexLastPodium(BLEU));
            depiler(BLEU);
            LO();
            Jeu.get(ROUGE).getAnimaux().set(getIndexLastPodium(ROUGE) + 1,
tmp);
        }
    }
    /**
     * NI - permet d'effectuer le déplacement NI
     */
    public void NI() {
        assert !estVide(BLEU);
        String tmp = Jeu.get(BLEU).getAnimaux().get(0);
        Jeu.get(BLEU).getAnimaux().remove(0);
        Jeu.get(BLEU).getAnimaux().add(Affichage.vide);
        Jeu.get(BLEU).getAnimaux().set(getIndexLastPodium(BLEU) + 1, tmp);
    }
```

```
    /**
     * MA - permet d'effectuer le déplacement MA
     */
    public void MA() {
        assert !estVide(ROUGE);
        String tmp = Jeu.get(ROUGE).getAnimaux().get(0);
        Jeu.get(ROUGE).getAnimaux().remove(0);
        Jeu.get(ROUGE).getAnimaux().add(Affichage.vide);
        Jeu.get(ROUGE).getAnimaux().set(getIndexLastPodium(ROUGE) + 1,
tmp);
    }

    /**
     * ordreValide - permet de Vérifier si la combinaison du dompteur est
valide
     * @param ordre Combinaison du dompteur
     * @return true si la combinaison ne comporte pas d'erreur, sinon false
     */
    public static boolean ordreValide(String ordre) {
        if (ordre.length() % 2 != 0)
            return false;

        for (int i = 0; i < ordre.length(); i = i + 2) {
            if (!ordre.startsWith("KI", i) && !ordre.startsWith("LO", i)
                    && !ordre.startsWith("SO", i) &&
!ordre.startsWith("NI", i)
                    && !ordre.startsWith("MA", i))
                return false;
        }
        return true;
    }

    /**
     * deplacer - permet de simuler la combinaison saisie par le dompteur
     * @param carte Carte à partir de laquelle on veut simuler l'ordre
     * @param deplacements Combinaison du dompteur
     */
    public void deplacer(Carte carte, String deplacements) {
        Jeu.set(BLEU, carte.getBleu());
        Jeu.set(ROUGE, carte.getRouge());
        if (ordreValide(deplacements)) {
            int LongeurChaineDeplacements = deplacements.length();
            for (int i = 0; i < LongeurChaineDeplacements; i += 2) {
                switch (deplacements.substring(i, i + 2)) {
                    case "KI":
                        KI();
                        break;
                    case "LO":
                        LO();
                        break;
                    case "SO":
                        SO();
                        break;
                    case "NI":
                        NI();
                        break;
                    case "MA":
                        MA();
                        break;
                }
            }
```

```
            }
        }

        /**
         * getPodiumBleu - permet d'obtenir le podium Bleu
         * @return le podium bleu
         */
        public Podium getPodiumBleu() {
            return Jeu.get(BLEU);
        }

        /**
         * getPodiumRouge - permet d'obtenir le podium Rouge
         * @return le podium rouge
         */
        public Podium getPodiumRouge() {
            return Jeu.get(ROUGE);
        }

}
```

V. Bilan de Projet

Pour un troisième projet, c'est dans l'ensemble une réussite car nous avons atteint tous les objectifs que nous nous étions fixés. On a du mal à démarrer car nous ne savions pas comment représenter les cartes, les stockées, représenter les podiums ainsi que les animaux.
Cependant, quelques points nous ont bloqué.
Tout d'abord, la méthode de génération du paquet de cartes a été assez longue a trouvé et à écrire mais nous y sommes arrivés et le résultat est à la hauteur de nos attentes.
Nous sommes aussi très satisfaits de l'affichage du jeu, la façon dont nous avons affiché le jeu le rend agréable à jouer.
Nous ne sommes cependant pas vraiment satisfaits de la manière dont nous avons organisé le programme, l'organisation des classes et des méthodes, on aurait pu mieux faire.
Pour finir, nous sommes tout de même très contents car pour un premier projet en langage orienté objet, nous pensons que c'est une réussite.