IUT de Paris - Rives de Seine
Université Paris Cité

DOSSIER DE DÉVELOPPEMENT EN LANGAGE JAVA

Objet : 6-qui-prend !

14 MARS 2022

GROUPES 104, 106 BINOME 75
Anxian ZHANG, Vick YE

Table des matières

I. Présentation de l'application

❖ Introduction du sujet

Le premier projet consistait à développer un interpréteur de commande pour l'organisation d'un tournoi. Le deuxième, un jeu ludique qui est le Minesweeper. Cette fois-ci, nous avons dû programmer en langage Java (version 8) un jeu de cartes qui est le *6-qui-prend !.* Les règles sont identiques à celles du jeu officiel, à une exception près, la partie se termine uniquement lorsque tous les joueurs ne possèdent plus de carte pouvant être jouée.

❖ Précisions du travail à réaliser

Dans un premier temps, notre programme devra générer 104 cartes qui seront mélangées par la suite. Chaque carte possède un numéro unique et un certain nombre de têtes de bœufs (points malus). Puis nous devrons distribuer 10 cartes à chacun des joueurs qui seront enregistrés. Avant que les joueurs ne jouent leurs premières cartes, le programme devra poser initialement 1 carte dans chaque série, soit 4 cartes. Chaque joueur sera appelé par le programme, après la confirmation de sa présence (via une touche du clavier), le contenu des séries et ses cartes lui seront dévoilés, le joueur devra alors choisir une carte qu'il possède. La carte sélectionnée sera posée quasiment automatiquement selon 4 règles :

Règle 1 : Supériorité

Les cartes d'une même série est toujours croissante, la carte ne peut donc être posée qu'à la suite d'une série où sa dernière carte est inférieure à la carte devant être posée.

Règle 2 : Différence la plus faible

La carte peut être posé dans la série que si la différence entre la carte à poser et la dernière carte d'une série est la plus faible parmi les 4.

Règle 3 : Nouveau début de série

Si la carte doit être posés dans une série qui comporte déjà 5 cartes, le joueur ramasse ces cartes et la carte choisie deviendra ainsi la première carte de cette série. On comptera le nombre total de tête de bœufs de toutes les cartes ramassées pour en déterminer les points de pénalité du joueur.

Règle 4 : Infériorité

Si la carte choisie par le joueur est tellement faible qu'il ne peut se poser dans aucune des séries, alors le joueur devra choisir une série à ramasser, puis la carte du joueur deviendra la première de la série choisie. (Des exemples seront apportés dans « Tests unitaires » de la rubrique « Annexes » à la page x).

II. Bilan de projet

❖ Les difficultés rencontrées

Durant le cycle de développement, le plus dur a été la compréhension du sujet. Mais nous avons pu contacter l'enseignant afin qu'il puisse nous éclairer, nous donner des pistes de compréhension.

❖ Le développement

Lors du cycle de développement, nous avons rencontré quelques problèmes. En effet, au sein de la méthode PlaceCardInSerie (de la classe AllSeries), qui permet à un joueur de poser une carte, il a fallu que nous prenions en considération toutes les règles indiquées plus haut, ainsi que le fait que le joueur doit choisir une série lorsque la carte choisie est trop faible pour être posée. Nous avons donc dû nous arranger pour que cette méthode marche correctement même si le joueur ne saisit pas de série et inversement.

Dans la majorité des fonctions toStirng il y avait une petite subtilité. Effectivement, dans la plupart de ces méthodes, nous avons dû mettre les bonnes conditions pour qu'il n'y ait pas de ponctuation en plus ou en moins lors de son exécution, et ce n'était pas évident. C'était surtout le cas pour le toStrinPlayersOxHead (de la classe AllJoueur), cette méthode a pour but de d'afficher le message : « Aucun joueur n'a ramassé de tête de bœufs » si personnes ne récolte de malus dans un tour, sinon le programme doit afficher le nom du joueur concerné suivit du nombre de malus récolté. Par exemple : « Desuwa a ramassé 5 têtes de bœufs ». Le plus compliqué dans la méthode était de trier les joueurs dans l'ordre croissant des têtes de bœufs ramassé l'osque plusieurs personnes doivent en ramasser. Nous sommes restés sur cette partie pas loin de 4 heures.

Erreur d'exécution :

- L'Out of length

Nous avons appris de nos erreurs lors de notre dernier projet. En conséquence, les erreurs rencontrées auparavant ne sont plus d'actualité, mais cela n'exclut pas les nouvelles erreurs ! En effet, toujours dans la méthode PlaceCardInSerie(), nous avons utilisé diverses autres méthodes issues de 2 paquetages différents. Lors des tests, nous avons eu à plusieurs reprises un problème de taille d'ArrayListes, nous prenions toujours une demi-heure pour résoudre le problème. À maintes reprises, l'erreur venait d'une mauvaise comparaison d'une variable dans la méthode.

❖ Les réussites

Dans l'ensemble, nous sommes à peu près passés par toutes les étapes du développement de ce programme, que ce soit sur l'organisation, le développement ou la validation de l'application.

L'organisation :

Avant de commencer à programmer, nous nous sommes consultés à plusieurs reprises afin de discuter du problème fourni. Nous avons d'abord émis des hypothèses sur la conception générale, soit, commencé à construire les classes potentielles pour le projet, mais aussi le « squelette » du programme principal. Puis nous avons prêté attention à leurs conceptions, cette fois-ci un peu plus dans les détails en donnant des bouts de code « potentiels ».

Le développement :

Puis, vient alors la réalisation. Chacun d'entre nous était chargé de programmer une ou plusieurs classe(s), si l'une des deux personnes se trouvait dans une impasse, on se consultait afin d'y remédier. À chaque fin de partie (cf. précision du travail à réaliser), nous faisions une réunion. Ces réunis avaient pour but d'expliquer à l'autre ce qu'untel avait réalisé pour telle(s) commande(s) ou telle(s) modification(s) qu'untel avait apporté au programme.

La validation :

Durant tout le cycle de développement, à chaque fois que nous finissions une partie ou une méthode importante, nous le testions tout de suite pour nous assurer de son bon fonctionnement. Les tests effectués étaient possibles grâce aux méthodes toString. Bien sûr, nous n'excluons pas les System.out qui nous ont permis de visualiser ce qui était stocké dans une ou plusieurs variable(s). Si le(s) résultat(s) est/sont cohérent(s) nous validions la méthode puis passions au codage de la suivante. Quand nous finissions une partie, nous testions la ou les classe(s) en testant divers cas qui nous viennent (voir page x pour les fichiers tests unitaires).

Finalement, nous avons réussi à programmer l'intégralité des méthodes qui nous permettent de jouer à 6-qui-prend ! de A à Z.

❖ Les améliorations

La compréhension du sujet :

Tout au long du cycle, nous avons fréquemment posé des questions à l'enseignant chargé du projet. Après qu'il nous ait répondu et que nous avons revu le sujet, nous avons pu constater que les réponses apportées étaient déjà présentes dans le sujet.

Les tests unitaires (JUnit 4) :

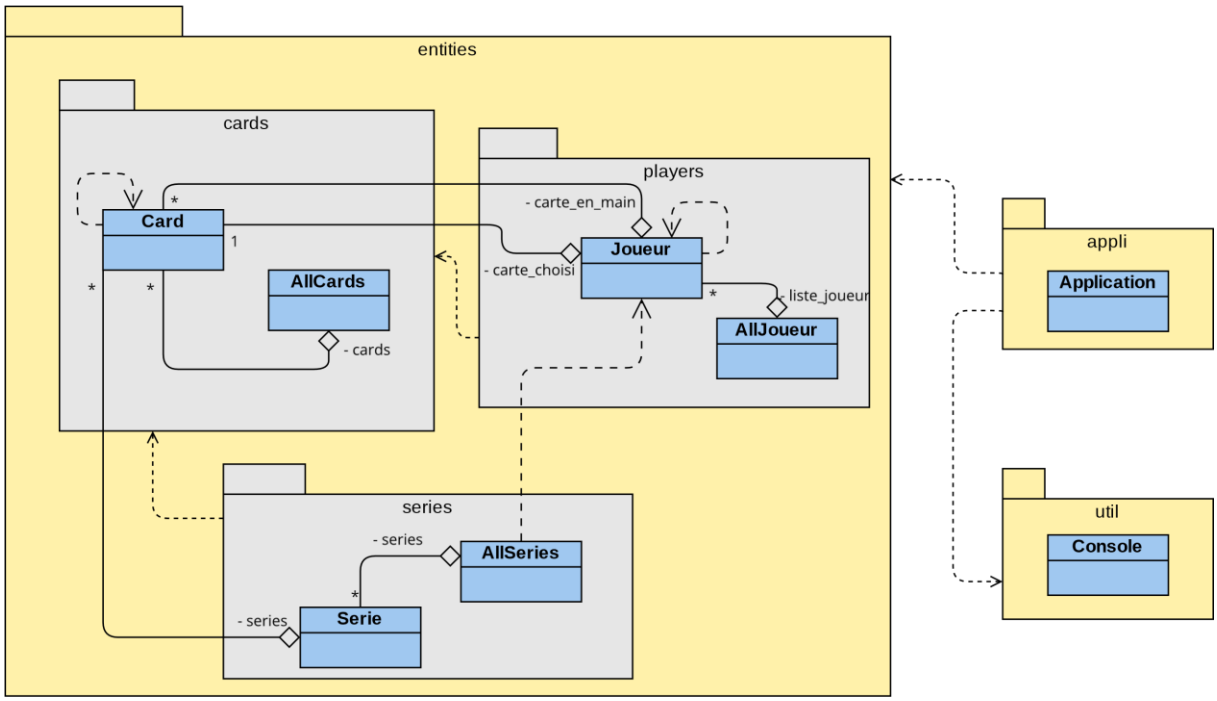
Nous pensons que nos tests ne sont pas très concluants. En effet, dans la plus par des tests, sont effectué simplement avec des comparaisons entre les valeurs calculées par le programme et les valeurs espérés. Donc, nous pensions que nous pouvons mieux faire à l'avenir.

La structuration des fichiers :

Pour ce projet, nous avons dû créer un diagramme UML en représentant les dépendances entre les classes et les packages (voir III). Le problème que nous avions rencontré est la complexité de l'utilisation des classes, en effet les classes unitaires ; « Card », « Joueur » et « Serie », sont utilisé dans respectivement dans « AllCards », « AllJoueur » et « AllSeries ». Et nous pensions que nous pouvions tous les regrouper dans une même classe nommé « Jeu ».

III. Les annexes

❖ Diagramme UML de notre application



❖ Tests JUnit

```
package tests;

import static org.junit.Assert.*;

import entities.cards.AllCards;
import entities.cards.Card;
import org.junit.Test;

import java.util.ArrayList;

public class AllCardsTest {

   /**
    * Vérifit si le maximun de carte à piocher est bien de 104
    */
   @Test
   public void testAllCards(){
       AllCards ac = new AllCards();
       ac.Generate_cards();
       ac.Rand_Card();

       assertFalse(ac.IsEmpty());
       int ctp = 0;
       while (true){
           ac.Piocher();
           ++ctp;
           if (ctp == AllCards.MAX_CARD) {
               assertTrue(ac.IsEmpty());
               return;
           }
           else
               assertFalse(ac.IsEmpty());
       }
   }

   /**
    * Vérification sur les valeurs de 2 cartes
    */
   @Test
   public void testCeation(){
       Card c = new Card(55);
       assertTrue(c.IsEqualsNum(55));
       assertFalse(c.HeadEquals1());
       assertEquals(c.toString(), "55 (7)");

       Card c2 = new Card(3);
       assertEquals(c2.toString(), "3");
       assertTrue(c2.HeadEquals1());
       assertNotEquals(c.getNum_card(), 103);
       assertFalse(c2.IsEqualsNum(7));

       assertFalse(c2.IsEqualsCard(c));
       assertTrue(c2.IsEqualsCard(c2));
   }
```

```
    /**
     * Test du bon fonctionnement de la somme
     */
    @Test
    public void testSumToal(){
        int total_malus = 0;
        ArrayList<Card> list_card = new ArrayList<>();
        list_card.add(new Card(55));
        list_card.add(new Card(80));
        list_card.add(new Card(1));

        for (Card c : list_card)
            total_malus = c.SumTotal(total_malus);

        assertEquals(total_malus, 11);
    }

    /**
     * Test si la comparaison de carte fonctionne bien
     */
    @Test
    public void testCompare() {
        Card c1 = new Card(9), c2 = new Card(26);
        assertEquals(Card.CompareCard(c1, c2), -17);
    }

    /**
     * Test si la carte recherché correspond bien à celle
     * que nous avaons déclaré
     */
    @Test
    public void testFindCard(){
        Card c = new Card(50);
        assertEquals(c.FindCard(50), c);
        assertNull(c.FindCard(55)); // car 50 != 55
        assertEquals(c.toStringPoserCard(),"Pour poser la carte 50, ");
    }

    /**
     * Test de déclaration d'une carte supérieur à 104
     */
    @Test
    public void testUpperNum(){
        Card c = new Card(105);
    }

    /**
     * Test de déclaration d'une carte inférieur à 1
     */
    @Test
    public void testLowerNum(){
        Card c = new Card(0);
    }
}
```

```java
package tests;

import entities.cards.AllCards;
import entities.cards.Card;
import entities.players.AllJoueur;
import entities.players.Joueur;
import org.junit.Test;

import java.util.ArrayList;

import static org.junit.Assert.*;
public class AllJoueurTest {

   /**
    * Test les valeurs lors de la création de nouveaux joueurs.
    */
   @Test
   public void testAllJoueur() {
       // Test de la méthode IsBellowNbJoueurs()
       AllJoueur aj = new AllJoueur();
       assertTrue(aj.IsBellowNbJoueurs());

       AllCards paquet = new AllCards();
       paquet.Generate_cards();
       aj.addJoueur("John",paquet);
       aj.addJoueur("Paul",paquet);
       aj.addJoueur("George",paquet);
       aj.addJoueur("Ringo",paquet);

       // Tests d'ajout de joueurs dans une instance de type AllJoueur.
       assertEquals(aj.getJoueur(0).toStringPlayerName(), "(John)");
       assertEquals(aj.getJoueur(1).toStringPlayerName(), "(Paul)");
       assertEquals(aj.getJoueur(2).toStringPlayerName(), "(George)");
       assertEquals(aj.getJoueur(3).toStringPlayerName(), "(Ringo)");

       // Test du nombre de joueurs ajoutés.
       assertFalse(aj.IsBellowNbJoueurs());
       assertEquals(aj.getNbjoueurs(), 4);

       assertEquals(aj.Remerciment(),
                       "Les 4 joueurs sont John, Paul, George et Ringo. " +
                       "Merci de jouer à 6 qui prend !");

       assertTrue(aj.getJoueur(0).HasCard(8));
       assertTrue(aj.getJoueur(1).HasCard(14));
       assertFalse(aj.getJoueur(2).HasCard(5));
       assertFalse(aj.getJoueur(3).HasCard(100));

       assertEquals(aj.getJoueur(0).toString(),
               "- Vos cartes : 1, 2, 3, 4, 5 (2), 6, 7, 8, 9, 10 (3)");
       assertEquals(aj.getJoueur(1).toString(),
               "- Vos cartes : 11 (5), 12, 13, 14, 15 (2)," +
                       " 16, 17, 18, 19, 20 (3)");
       assertEquals(aj.getJoueur(2).toString(),
               "- Vos cartes : 21, 22 (5), 23, 24, 25 (2)," +
                       " 26, 27, 28, 29, 30 (3)");
```

```
    assertEquals(aj.getJoueur(3).toString(),
                    "- Vos cartes : 31, 32, 33 (5), 34, 35 (2)," +
                    " 36, 37, 38, 39, 40 (3)");
}

/**
 * Tests d'initialisation d'une nouvelle instance de type Joueur.
 */
@Test
public void testInitJoueur() {
    // Test de création d'une nouvelle instance de type Joueur.
    Joueur j = new Joueur("Bojji");
    assertEquals(j.toStringPlayerName(), "(Bojji)");
    assertEquals(j.toString(), "- Vos cartes : ");
    assertEquals(j.toStringOxHeadPerTurn(),
            "Bojji a ramassé 0 tête de boeufs");
    assertEquals(j.toStringFinalScore(),
            "Bojji a ramassé 0 tête de boeufs");

    AllCards paquet = new AllCards();
    paquet.Generate_cards();
    j.PiocheCards(paquet);

    // Test la distribution des cartes à un joueur.
    for (int i = 0; i < 10; ++i) {
        assertTrue(j.HasCard(i + 1));
    }

    /* Test la chaîne de caractères contenant la liste de cartes
     * en main d'un joueur;*/
    assertEquals(j.toString(),
            "- Vos cartes : 1, 2, 3, 4, 5 (2), 6, 7, 8, 9, 10 (3)");
}

/**
 * Test de la méthode qui retrouve un joueur qui possède une carte spécifique.
 */
@Test
public void testFindPlayer() {
    AllJoueur aj = new AllJoueur();
    AllCards ac = new AllCards();
    ac.Generate_cards();

    aj.addJoueur("Michel", ac);
    aj.addJoueur("Jean", ac);
    assertFalse(aj.IsBellowNbJoueurs());
    assertEquals(aj.getNbjoueurs(), 2);

    assertEquals(aj.FindPlayer(new Card(6)).toStringPlayerName(),
            "(Michel)");
    assertEquals(aj.FindPlayer(new Card(19)).toStringPlayerName(),
            "(Jean)");
}
```

```
/**
 * Tests concernant toutes les valeurs relatives aux têtes de bœufs
 * desjoueurs.
 */
@Test
public void testOxHead() {
    AllJoueur aj = new AllJoueur();
    AllCards ac = new AllCards();
    ac.Generate_cards();
    aj.addJoueur("Michel", ac);
    aj.addJoueur("Jean", ac);

    /* Test d'affectation de têtes de boeufs aux joueurs d'une
     * instance de type AllJoueur.*/
    aj.getJoueur(0).GatherOxHead(3);
    aj.getJoueur(1).GatherOxHead(2);
    assertEquals(aj.getJoueur(1).getOxHeadPerTurn(), 2);
    assertEquals(aj.getJoueur(0).getOxHeadPerTurn(), 3);

    assertEquals(aj.getJoueur(0).toStringOxHeadPerTurn(),
            "Michel a ramassé 3 têtes de boeufs");
    assertEquals(aj.toStringPlayersOxHead(),
            "Jean a ramassé 2 têtes de boeufs" + "\n" +
            "Michel a ramassé 3 têtes de boeufs");

    /* Test de réinitialisation de toutes les têtes de boeufs
     * ramassées à chaque tour par un joueur.*/
    aj.InitGamersOxHeadPerTurn();
    assertEquals(aj.getJoueur(0).getOxHeadPerTurn(), 0);
    assertEquals(aj.getJoueur(1).getOxHeadPerTurn(), 0);

    /* Test de trie par le nombre de têtes de boeufs ramassées
     * au total par chaque joueur.*/
    aj.OrderByTotalOxHead();
    assertEquals(aj.getJoueur(1).toStringPlayerName(), "(Michel)");
    assertEquals(aj.getJoueur(0).toStringPlayerName(), "(Jean)");
    aj.getJoueur(0).GatherOxHead(8);
    aj.getJoueur(1).GatherOxHead(1);

    assertEquals(aj.toStringPlayersOxHead(),
            "Michel a ramassé 1 tête de boeufs" + "\n" +
            "Jean a ramassé 8 têtes de boeufs");

    aj.OrderByTotalOxHead();
    assertEquals(aj.getJoueur(0).toStringPlayerName(), "(Michel)");
    assertEquals(aj.getJoueur(1).toStringPlayerName(), "(Jean)");

    /* Test la chaîne de charactères retournée contenant le
     * score final d'un joueur.*/
    assertEquals(aj.getJoueur(1).toStringFinalScore(),
            "Jean a ramassé 10 têtes de boeufs");

    /* Test la chaîne de charactères retournée contenant le
     * score final de tout les joueurs.*/
    assertEquals(aj.toStringFinalPlayersOxHead(), "** Score final" + "\n"
            + "Michel a ramassé 4 têtes de boeufs" + "\n"
            + "Jean a ramassé 10 têtes de boeufs");
}
```

```
/**
 * Test de trie des joueurs par ordre alphabétique.
 */
@Test
public void testOrderByName() {
    AllJoueur aj = new AllJoueur();
    AllCards ac = new AllCards();
    ac.Generate_cards();
    aj.addJoueur("Michel", ac);
    aj.addJoueur("Jean", ac);
    aj.addJoueur("Michel",ac);
    assertEquals(aj.getJoueur(0).toStringPlayerName(), "(Michel)");
    assertEquals(aj.getJoueur(1).toStringPlayerName(), "(Jean)");
    assertEquals(aj.getJoueur(2).toStringPlayerName(), "(Michel)");

    aj.OrderByGamerName();
    assertEquals(aj.getJoueur(0).toStringPlayerName(), "(Jean)");
    assertEquals(aj.getJoueur(1).toStringPlayerName(), "(Michel)");
    assertEquals(aj.getJoueur(2).toStringPlayerName(), "(Michel)");

    //Test de comparaison de deux joueurs à leurs noms.
    assertTrue(Joueur.compareNom(aj.getJoueur(0),aj.getJoueur(1))<0);
    assertTrue(Joueur.compareNom(aj.getJoueur(1),aj.getJoueur(0))>0);
    assertEquals(Joueur.compareNom(aj.getJoueur(1), aj.getJoueur(1)), 0);
}

/**
 * Test des méthodes toStringChooseSerie(),
 * HasPlayedCard() et RemoveCard()
 */
@Test
public void testToStringChooseSerie() {
    Joueur j = new Joueur("John");
    AllCards paquet = new AllCards();
    paquet.Generate_cards();
    j.PiocheCards(paquet);
    assertTrue(j.HasCard(10));

    /* Test la chaîne retournée demandant à un joueur de choisir
     * une série à vider pour poser une carte.*/
    Card c = j.FindCard(10);
    assertEquals(j.toStringChooseSerie(c),
                    "Pour poser la carte 10, " +
                    "John doit choisir la série qu'il va ramasser.");

    // Vérifie si un joueur possède la carte qu'il veut jouer.
    assertTrue(j.HasPlayedCard(c));

    //Test si la carte joué a bien été retiré de la main du joueur.
    j.RemoveCard(c);
    assertFalse(j.HasCard(10));
    assertEquals(j.toString(),
            "- Vos cartes : 1, 2, 3, 4, 5 (2), 6, 7, 8, 9");
}
```

```
    /**
     * Test de la chaîne de charactères retournée par toStringPassage().
     */
    @Test
    public void testToStringPassage() {
        AllJoueur aj = new AllJoueur();
        AllCards paquet = new AllCards();
        paquet.Generate_cards();
        aj.addJoueur("Luck", paquet);
        aj.addJoueur("Rob", paquet);

        aj.getJoueur(0).HasCard(5);
        aj.getJoueur(1).HasCard(13);
        ArrayList<Card> lc = new ArrayList<>();
        lc.add(aj.getJoueur(1).FindCard(13));
        lc.add(aj.getJoueur(0).FindCard(5));
        assertEquals(aj.toStringPassage(lc,true),
                "Les cartes 13 (Rob) et 5 (Luck) vont être posées.");
        assertEquals(aj.toStringPassage(lc,false),
                "Les cartes 13 (Rob) et 5 (Luck) ont été posées.");
    }
}
```

```
package tests;

import entities.cards.AllCards;
import entities.cards.Card;
import entities.players.Joueur;
import entities.series.AllSeries;
import org.junit.Test;

import static org.junit.Assert.*;

public class AllSeriesTest {

    /**
     * Test répectivement à chaque saut de ligne:
     *  - la création de series
     *  - l'affichages des series
     *  - et si l'ajout d'une carte est possible ou pas
     */
    @Test
    public void testAllSeriesInit() {
        AllCards packet = new AllCards();
        packet.Generate_cards();
        Card c = packet.Piocher();
        AllSeries as = new AllSeries(packet);
        for (int i = 1; i <= AllSeries.MAX_SERIES; ++i)
            assertFalse(as.getSerie(i).IsFull());

        assertEquals(as.toString(),
                     "- série n° 1 : 2" + "\n" +
                     "- série n° 2 : 3" + "\n" +
                     "- série n° 3 : 4" + "\n" +
                     "- série n° 4 : 5 (2)");

        assertFalse(as.CanPlaceCard(c));
```

```
        c = packet.Piocher();
        assertTrue(as.CanPlaceCard(c));
    }

    /**
     * test la différence entre une carte et la dernière
     * carte d'une série mais aussi le placement des cartes
     * et la vérification des malus récolté si le joueur
     * ajoute une 6eme carte à une serie qui en posède deja 5
     *
     * dans ce test on considère que les cartes ajouté, sont
     * du même paquet et que les carte ajouté sont des cartes
     * que le joueur posède
     */
    @Test
    public void TestDifference (){
        AllCards packet = new AllCards();
        packet.Generate_cards();

        // test d'ajout de carte
        AllSeries as = new AllSeries(packet);
        for (int i = 1; i <= AllSeries.MAX_SERIES; ++i){
            packet.Piocher();
            as.getSerie(i).addCarte(packet.Piocher());
        }
        as.getSerie(1).addCarte(new Card(87));
        as.getSerie(2).addCarte(new Card(60));
        as.getSerie(3).addCarte(new Card(55));
        as.getSerie(4).addCarte(new Card(22));
        assertEquals(as.toString(),
                "- série n° 1 : 1, 6, 87" + "\n" +
                     "- série n° 2 : 2, 8, 60 (3)" + "\n" +
                     "- série n° 3 : 3, 10 (3), 55 (7)" + "\n" +
                     "- série n° 4 : 4, 12, 22 (5)");

        // test de placement de carte
        Joueur j = new Joueur("Aqua");
        as.PlaceCardInSerie(new Card(90), j, null);
        as.PlaceCardInSerie(new Card(43), j, null);
        as.PlaceCardInSerie(new Card(101), j, null);
        assertEquals(as.toString(),
                  "- série n° 1 : 1, 6, 87, 90 (3), 101" + "\n" +
                       "- série n° 2 : 2, 8, 60 (3)" + "\n" +
                       "- série n° 3 : 3, 10 (3), 55 (7)" + "\n" +
                       "- série n° 4 : 4, 12, 22 (5), 43");

        // test si une 6eme cartes est posé
        as.PlaceCardInSerie(new Card(104), j, null);
        assertEquals(as.toString(),
                  "- série n° 1 : 104" + "\n" +
                       "- série n° 2 : 2, 8, 60 (3)" + "\n" +
                       "- série n° 3 : 3, 10 (3), 55 (7)" + "\n" +
                       "- série n° 4 : 4, 12, 22 (5), 43");
        assertEquals(j.getOxHeadPerTurn(), 7);
    }
}
```

❖ **Code Source**

```
package entities.cards;

/**
* Brief: Classe Card, permettant de manipuler une
*        carte
* @author Anxian Zhang, Vick Ye
* @version 3
* @since 08/03/2022
*/
public class Card {
   private final int num_card;
   private final int num_ox_head;

   /**
    * Constructeur permettant de créer de nouvelles
    * instances de Card.
    * @param num le numéro qui sera attribué à la carte
    * @see #Generate_ox_head()
    */
   public Card(int num) throws RuntimeException{
       if (num <= 0 || num > AllCards.MAX_CARD)
           throw new RuntimeException("Not Allowed value");
       this.num_card = num;
       this.num_ox_head = this.Generate_ox_head();
   }

   /**
    * Permet l'accès au numéro de la carte.
    * @return le numéro de la carte
    */
   public int getNum_card() {
       return this.num_card;
   }

   /**
    * Compare 2 cartes afin de déduir si la première
    * possède un numéro inférieur, égal ou supèrieur
    * à celui du deuxième.
    * @param c1 carte n°1
    * @param c2 carte n°2
    * @return la differénce entre les 2 cartes
    */
   public static int CompareCard(Card c1, Card c2){
       return c1.num_card - c2.num_card;
   }


    /**
    * Génére le nombre têtes de boeufs correspondant à
    * la carte créée.
    * @return le nombre de tête de boeufs
    */
   private int Generate_ox_head() {
```

```
    if (this.num_card == 55)
        return 7;
    else if (this.num_card % 10 == 0)
        return 3;
    else if (this.num_card % 5 == 0 && this.num_card % 10 != 0)
        return 2;
    else if (this.num_card % 11 == 0)
        return 5;
    else
        return 1;
}

/**
 * Renvoie le numéro de la carte et son nombre de
 * têtes de boeufs si celui-ci est différent de 1,
 * sinon renvoie uniquement le numéro de la carte.
 * @return la chaîne de caractères
 * @see #HeadEquals1()
 */
public String toString() {
    if (!this.HeadEquals1())
        return this.num_card + " (" + this.num_ox_head + ")";
    return Integer.toString(this.num_card);
}

/**
 * Indique si le nombre de têtes de boeufs est égal à 1.
 * @return vrai si c'est le cas, sinon faux
 */
public boolean HeadEquals1(){
    return this.num_ox_head == 1;
}

/**
 * Renvoie une chaîne indiquant la carte à poser.
 * @return la chaîne de caractère correspondant à
 *           la carte à poser
 */
public String toStringPoserCard(){
    return "Pour poser la carte " + this.num_card + ", ";
}

/**
 * Trouve une carte dont le numéro correspond
 * à un numéro spécifique, si elle existe.
 * @param num le numéro
 * @return la carte si elle existe, sinon
 *           un ensemble vide
 * @see #IsEqualsCard(Card)
 */
public Card FindCard (int num){
    if (IsEqualsNum(num))
        return this;
    return null;
}
```

```
    /**
     * Vérifie si le numéro d'une carte correspond
     * à un numéro spécifique.
     * @param num le numéro spécifique
     * @return vrai si c'est le cas, sinon faux
     */
    public boolean IsEqualsNum(int num){
        return this.num_card == num;
    }

    /**
     * Vérifie si le numéro d'une carte est égal à
     * celui d'une deuxième.
     * @param c la deuxième carte
     * @return vrai si oui, sinon faux
     */
    public boolean IsEqualsCard(Card c){
        return this.num_card == c.num_card;
    }

    /**
     * Incrémente le nombre de têtes de boeufs d'une
     * carte à la somme totale des têtes de boeufs.
     * @param total la somme totale des têtes de boeufs
     * @return la somme totale des têtes de boeufs +
     *           les têtes de boeufs de la carte
     */
    public int SumTotal(int total){
        return this.num_ox_head + total;
    }
}
```

```
package entities.cards;

import java.util.ArrayList;
import java.util.Random;

/**
* Brief: Classe AllCards, permettant de manipuler un
*        paquet de cartes.
* @author Anxian Zhang, Vick Ye
* @version 3
* @since 08/03/2022
*/
public class AllCards {
    public static final int MAX_CARD = 104;
    private final ArrayList<Card> cards;

    /**
     * Constructeur permettant de créer de nouvelles
     * instances de type AllCards.
     */
    public AllCards() {
        this.cards = new ArrayList<>();
    }
```

```
    /**
     * Génére toutes les cartes constituant un nouveau paquet
     * @see #Rand_Card()
     */
    public void Generate_cards() {
        for (int i = 0; i < MAX_CARD; ++i) {
            this.cards.add(new Card(i + 1));
        }
    }

    /**
     * Mélange les cartes générées.
     */
    public void Rand_Card() {
        Random position = new Random();
        int new_position;
        for (int i = 0; i < MAX_CARD; ++i) {
            new_position = position.nextInt(MAX_CARD);
            Card tmp = this.cards.get(i);
            this.cards.set(i ,this.cards.get(new_position));
            this.cards.set(new_position , tmp);
        }
    }

    /**
     * Retire une carte du paquet lorsqu'elle est distribuée à
     * un joueur.
     * @return la carte retirée
     */
    public Card Piocher(){
        assert (!IsEmpty());
        Card c = this.cards.get(0);
        this.cards.remove(c);
        return c;
    }

    /**
     * Vérifie si le paquet est vide.
     * @return vrai si c'est le cas, sinon faux
     */
    public boolean IsEmpty(){
        return this.cards.size() == 0;
    }
}
```

```
package entities.players;

import java.util.ArrayList;

import entities.cards.*;

/**
* Brief: Classe Joueur, permettant de manipuler un
*        joueur.
* @author Anxian Zhang, Vick Ye
* @version 7
* @since 09/03/2022
*/
```

```
public class Joueur {
   private static final int MAX_CARTES_MAIN = 10;
   private final String nom_joueur;
   private int total_ox_head;
   private int ox_head_per_turn;
   private Card carte_choisi;
   private final ArrayList<Card> cartes_en_main;

   /**
    * Constructeur permettant de créer de nouvelles
    * instances de types Joueur.
    * @param nom le  nom du joueur
    */
   public Joueur(String nom) {
       this.nom_joueur = nom;
       this.total_ox_head = 0;
       this.ox_head_per_turn = 0;
       this.carte_choisi = null;
       this.cartes_en_main = new ArrayList<>();
   }

   /**
    * Distribue 10 cartes à un joueur depuis un
    * paquet de cartes.
    * @param packet le paquet de cartes
    * @see #RangeCartes()
    * @see AllCards#Piocher()
    */
   public void PiocheCards(AllCards packet) {
       assert(this.cartes_en_main.size() < 10 && this.total_ox_head == 0);

       for (int i = 0; i < MAX_CARTES_MAIN; ++i) {
           this.cartes_en_main.add(packet.Piocher());
       }
       this.RangeCartes();
   }

   /**
    * Range les cartes d'un joueur par ordre croissant
    * selon le numéro de chaque carte.
    * @see Card#CompareCard(Card, Card)
    */
   private void RangeCartes(){
       this.cartes_en_main.sort(Card::CompareCard);
   }

   /**
    * Permet l'accès au nom d'un joueur.
    * @return le nom du joueur
    */
   public String getNomJoueur() {
       return this.nom_joueur;
   }
```

```
/**
 * Vérifie si le numéro de la carte entrée existe
 * dans les cartes en main d'un joueur.
 * @param num le numéro de carte donné en entrée
 * @return vrai si la carte existe, sinon faux
 * @see Card#IsEqualsNum(int)
 */
public boolean HasCard(int num){
    for (Card c : this.cartes_en_main)
        if (c.IsEqualsNum(num)) {
            this.carte_choisi = c;
            return true;
        }
    return false;
}

/**
 * Vérifie si un joueur possède une carte
 * spécifique parmis ses cartes en main.
 * @param c la carte spécifique
 * @return vrai si c'est le cas, sinon faux
 */
public boolean HasPlayedCard(Card c){
    return this.carte_choisi.IsEqualsCard(c);
}

/**
 * Permet de connaître le nom d'un joueur.
 * @return une chaîne de caractères contenant le nom du joueur
 */
public String toStringPlayerName(){
    return "(" + this.nom_joueur + ")";
}

/**
 * Cherche, depuis les cartes en main d'un joueur,
 * la carte qui correspond au numéros de la carte
 * donnée en entrée.
 * @param num le numéro de la carte donné en entrée
 * @return la carte du joueur correspondante
 * @see Card#FindCard(int)
 */
public Card FindCard(int num){
    Card card;
    for (Card c : this.cartes_en_main) {
        card = c.FindCard(num);
        if (card != null)
            return card;
    }
    return null;
}

/**
 * Affecte le nombre de malus récolté par un joueur à
 * chaque tour au nombre total de malus récolté par
 * ce-denier durant la partie en cours.
 * @param ox_head_per_turn les points de malus du joueur par tour
 */
public void GatherOxHead(int ox_head_per_turn) {
```

```
        this.ox_head_per_turn = ox_head_per_turn;
        this.total_ox_head += this.ox_head_per_turn;
    }

    /**
     * Initialise le nombre de têtes de boeufs ramassé par un joueur à chaque tour
à 0.
     */
    public void InitOxHeadPerTurn(){
        this.ox_head_per_turn = 0;
    }

    /**
     * Retire une carte qui se trouve dans la main
     * d'un joueur.
     * @param c la carte à retirer
     */
    public void RemoveCard(Card c){
        this.cartes_en_main.remove(c);
    }

    /**
     * Permet de connaître la série à ramasser, pour pouvoir
     * poser une carte.
     * @param c la carte à poser
     * @return la chaîne de caractères indiquant la série à ramasser
     */
    public String toStringChooseSerie(Card c){
        return c.toStringPoserCard() + this.nom_joueur +
                " doit choisir la série qu'il va ramasser.";
    }

    /**
     * Permet à un joueur de visualiser ses cartes en main
     * @return la chaîne de caratères spécifiant les cartes d'un joueur
     * @see Card#toString()
     */
    public String toString(){
        StringBuilder sb = new StringBuilder("- Vos cartes : ");

        for (int idx = 0; idx < this.cartes_en_main.size(); ++idx) {
            sb.append(this.cartes_en_main.get(idx).toString());
            if (idx+1 != this.cartes_en_main.size())
                sb.append(", ");
        }
        return sb.toString();
    }

    /**
     * Permet de connaître le nombre de têtes de boeufs ramassé
     * par un joueur à la fin d'un tour.
     * @return le nombre de têtes de boeufs.
     */
    public int getOxHeadPerTurn() {
        return this.ox_head_per_turn;
    }
```

```
/**
 * Compare le nom de deux joueurs afin de déterminer si le nom du premier
 * précède, est postérieur ou est identique à celui du deuxième.
 * @param j1 : le premier joueur.
 * @param j2 : le deuxième.
 * @return 0 lorsque les noms sont identiques.
 * Un nombre positif lorsque le nom du premier précède celui du deuxième.
 * Un nombre négatif lorsque le nom premier est postérieur à ceuli du
 * deuxième.
 */
public static int compareNom(Joueur j1, Joueur j2) {
    return j1.nom_joueur.compareTo(j2.nom_joueur);
}

/**
 * Compare le nombre total de têtes de boeufs accumulé lors de la partie
 * de deux joueurs afin de déterminer si le premier en a plus, moins ou si
 * les deux possèdent le même nombre de têtes de boeufs.
 * @param j1 : le premier joueur.
 * @param j2 : le deuxième.
 * @return 0 lorsque le nombre est égal et le nom des joueurs identiques.
 * Un nombre positif lorsque le premier en possède plus que le deuxième ou
 * lorsque le nombre est égal mais le nom du premier précède celui du
 * deuxième.
 * Un nombre négatif lorsque le premier en possède moins que le deuxième ou
 * lorsque le nombre est égal mais le nom du premier esr postérieur à celui
 * du deuxième.
 * @see #compareNom(Joueur, Joueur)
 */
public static int compareNbTotalOxHead(Joueur j1, Joueur j2){
    if (j1.total_ox_head == j2.total_ox_head)
        return compareNom(j1, j2);
    return j1.total_ox_head - j2.total_ox_head;
}

/**
 * Permet de connaître le nombre de têtes de boeufs ramassé par un joueur.
 * @return le nombre de têtes de boeufs ramassé par le joueur.
 */
public String toStringOxHeadPerTurn() {
    StringBuilder sb = new StringBuilder(this.nom_joueur);
    sb.append(" a ramassé ").append(this.ox_head_per_turn);
    if(this.ox_head_per_turn>1) {
        return sb.append(" têtes de boeufs").toString();
    }
    return sb.append(" tête de boeufs").toString();
}

/**
 * Permet de connaître le nombre total de têtes de boeufs ramassé par un
 * joueur durant toute la partie.
 * @return le nombre total de têtes de boeufs ramassé par un joueur
 */
public String toStringFinalScore(){
    StringBuilder sb = new StringBuilder(this.nom_joueur);
    sb.append(" a ramassé ").append(this.total_ox_head);
    if(this.total_ox_head>1) {
        return sb.append(" têtes de boeufs").toString();
    }
```

```
        return sb.append(" tête de boeufs").toString();
    }
}
```

```
package entities.players;

import entities.cards.*;

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

/**
* Brief: Classe AllJoueur, permettant de manipuler
*        la liste de joueurs présent dans la partie.
* @author Anxian Zhang, Vick Ye
* @version 6
* @since 09/03/2022
*/
public class AllJoueur {
    private final ArrayList<Joueur> liste_joueur;
    private static final int MIN_JOUEURS = 2;
    private int nb_joueurs = 0;

    /**
     * Constructeur permettant de créer de nouvelles
     * instances de type Alljoueur.
     */
    public AllJoueur(){
        this.liste_joueur = new ArrayList<>();
    }

    /**
     * Vérifie si le nombre total de joueur initialisé
     * est strictement inférieur à MIN JOUEURS (soit 2).
     * @return vrai si c'est le cas, faux sinon
     */
    public boolean IsBellowNbJoueurs(){
        return this.nb_joueurs < MIN_JOUEURS;
    }

    /**
     * Initialise tous les joueurs puis distribue 10
     * cartes à chacun.
     * @param nomJ le nom du joueur, pour l'initialisation
     *             d'un nouveau joueur
     * @param packet le paquet pour initialiser les
     *               cartes en main du joueur initialisé
     * @see Joueur#PiocheCards(AllCards)
     */
    public void addJoueur(String nomJ, AllCards packet) {
        this.liste_joueur.add(new Joueur(nomJ));
        this.liste_joueur.get(this.nb_joueurs).PiocheCards(packet);
        ++this.nb_joueurs;
    }
```

```
/**
 * Cherche un joueur en fonction d'une carte jouée.
 * @param c la carte jouée
 * @return le joueur à trouver
 * @see Joueur#HasCard(int)
 */
public Joueur FindPlayer(Card c){
    Joueur j = null;

    for (int i = 0; i < this.nb_joueurs; ++i){
        if (this.liste_joueur.get(i).HasCard(c.getNum_card()))
            j = this.liste_joueur.get(i);
    }
    return j;
}

/**
 * Permet l'accès au nombre de joueurs enregistré
 * lors de l'initialisation de la liste de joueurs.
 * @return le nombre de joueur
 */
public int getNbjoueurs() {
    return this.nb_joueurs;
}

/**
 * Retourne l'inième joueur enregistré lors de
 * l'initialisation de la liste de joueurs.
 * @param i indice du joueur
 * @return l'inième joueur
 */
public Joueur getJoueur(int i) {
    return this.liste_joueur.get(i);
}

/**
 * Retourne la chaîne de caractères permettant de visualiser
 * les joueurs qui ont été enregistrés dans le programme,
 * tout en les remerciant.
 * @return le message de remerciement
 */
public String Remerciment() {
    StringBuilder sb = new StringBuilder();

    sb.append("Les ").append(this.nb_joueurs).append(" joueurs sont ");
    for(int i = 0; i < this.nb_joueurs; ++i) {
        if (i+1 == this.nb_joueurs)
            sb.append(" et ");
        else if( i > 0)
            sb.append(", ");
        sb.append(this.liste_joueur.get(i).getNomJoueur());
    }
    sb.append(". Merci de jouer à 6 qui prend !");

    return sb.toString();
}
```

```
    /**
     * Retourne la chaîne de caractères permettant de visualiser
     * l'ordre de passage des joueurs en fonction des cartes qu'ils
     * ont joué. Si le choix de la série est à true alors le programme
     * ajoute à la fin: "vont être posées.", sinon "ont été posées.",
     * puis retire la carte de la main du joueur.
     * @param cartesJouees les cartes jouées par les joueurs
     * @param cs choix de la serie, true/false
     * @return l'ordre de passage des joueurs
     * @see Joueur#toStringPlayerName()
     * @see Joueur#HasPlayedCard(Card)
     */
    public String toStringPassage (ArrayList<Card> cartesJouees, Boolean cs){
        StringBuilder sb = new StringBuilder("Les cartes ");
        for(int i = 0; i < cartesJouees.size(); ++i) {
            if(i+1 == cartesJouees.size())
                sb.append(" et ");
            else if (i > 0)
                sb.append(", ");
            sb.append(cartesJouees.get(i).getNum_card());
            for(int j = 0; j < cartesJouees.size(); ++j) {
                if(this.liste_joueur.get(j).HasPlayedCard(cartesJouees.get(i))) {
                    sb.append("
").append(this.liste_joueur.get(j).toStringPlayerName());
                }
            }
        }
        if (cs)
            sb.append(" vont être posées.");
        else
            sb.append(" ont été posées.");

        return sb.toString();
    }

    /**
     * Initialise le nombre de têtes de boeufs ramassé par tous les joueurs
     * à chaque tour à 0.
     *
     * @see Joueur#InitOxHeadPerTurn()
     */
    public void InitGamersOxHeadPerTurn() {
        for (int i = 0; i < this.nb_joueurs; ++i) {
            this.liste_joueur.get(i).InitOxHeadPerTurn();
        }
    }

    /**
     * Trie les joueurs de la partie par ordre alphabétique selon leur nom.
     */
    public void OrderByGamerName(){
        this.liste_joueur.sort(Joueur::compareNom);
    }
```

```
    /**
     * Trie les joueurs en fonctions de toutes les têtes de boeufs ramassées
     * durant la partie.
     */
    public void OrderByTotalOxHead(){
        this.liste_joueur.sort(Joueur::compareNbTotalOxHead);
    }

    /**
     * Permet de connaître le nombre de têtes de boeufs ramassé par chaque
     * joueur lorsqu'il n'est pas nul lors d'un tour. Sinon, renvoie un
     * message par défaut.
     * @return le message adapté selon le cas.
     */
    public String toStringPlayersOxHead() {
        StringBuilder sb = new StringBuilder();
        int tmp = 0;
        ArrayList<Integer> joueursOxHead = GetAllPlayerOxHead();

        //cas 1: Aucun joueur ne ramasse de tête de boeufs.
        if(joueursOxHead.size()==0) {
            sb.append("Aucun joueur ne ramasse de tête de boeufs.");
            return sb.toString();
        }

        //cas 2: renvoie les joueurs suivis de leurs nombre de malus s'il est
supérieur à 0.
        for(int i = 0; i < joueursOxHead.size(); ++i) {
            for(int j = 0; j < getNbjoueurs(); ++j) {
                if(liste_joueur.get(j).getOxHeadPerTurn() == joueursOxHead.get(i))
{
                    ++tmp;
                    sb.append(liste_joueur.get(j).toStringOxHeadPerTurn());
                    if(tmp < joueursOxHead.size())
                        sb.append("\n");
                }
            }
        }
        return sb.toString();
    }

    /**
     * Récolte toutes les têtes de boeufs ramassées par chaque joueur
     * puis les trie.
     * @return la liste de têtes de boeufs récoltées triée
     */
    private ArrayList<Integer> GetAllPlayerOxHead (){
        ArrayList<Integer> list = new ArrayList<>();
        for (Joueur joueur : this.liste_joueur) {
            if (joueur.getOxHeadPerTurn() != 0)
                list.add(joueur.getOxHeadPerTurn());
        }
        Collections.sort(list);
        return list;
    }
```

```
    /**
     * Permet de connaître le nombre de têtes de boeufs ramassé par chaque
     * joueur à la fin de la partie.
     * @return le message contenant le nombre de têtes de boeufs
     *           corespondant à chaque joueur.
     * @see #OrderByGamerName()
     * @see Joueur#toStringFinalScore()
     * @see Joueur#compareNom(Joueur, Joueur)
     */
    public String toStringFinalPlayersOxHead() {
        StringBuilder sb = new StringBuilder("** Score final");
        sb.append("\n");
        for (int i = 0; i < this.nb_joueurs; ++i) {
            for (int j = 0; j < this.nb_joueurs; ++j) {
                if (i != j) {
                    if (Joueur.compareNom(this.liste_joueur.get(i),
                            this.liste_joueur.get(j)) == 0)
                        this.OrderByGamerName();
                }
            }
        }
        for (int i = 0; i < this.nb_joueurs; ++i) {
            sb.append(this.liste_joueur.get(i).toStringFinalScore());
            if(i<nb_joueurs-1)
                sb.append("\n");
        }
        return sb.toString();
    }
}
```

```
package entities.series;

import java.util.ArrayList;

import entities.cards.*;

/**
* Brief: Classe Serie, permettant de manipuler une série de cartes.
* @author Anxian Zhang, Vick Ye
* @version 5
* @since 09/03/2022
*/
public class Serie {
    private final ArrayList<Card> serie;
    private final int num_serie;
    private static int ctp_num_serie = 0;
    public static final int MAX_CARDS_SERIE = 5;

    /**
     * Constructeur permettant de créer de nouvelles instances
     * de type série, puis affecte une carte en début de série.
     * @param packet le paquet qui sera utisé pour
     *               l'affectation de la première carte d'une
     *               série.
     * @see #InitSeries(AllCards)
     */
    public Serie(AllCards packet) throws RuntimeException{
        if (ctp_num_serie == 4)
          ctp_num_serie = 0;
```

```
    this.serie = new ArrayList<>();
    this.num_serie = ++ctp_num_serie;
    InitSeries(packet);
}

/**
 * Initialise une série créée en plaçant
 * une carte à son début.
 * @param packet le paquet utilisé pour l'initialisation
 * @see AllCards#Piocher()
 */
private void InitSeries(AllCards packet){
    this.serie.add(packet.Piocher());
}

/**
 * Ajoute une carte à une série.
 * @param c la carte à ajouter
 */
public void addCarte(Card c) {
    this.serie.add(c);
}

/**
 * Retourne le nombre de cartes qu'une série possède.
 * @return le nombre de cartes
 */
public boolean IsFull() {
    return this.serie.size() == MAX_CARDS_SERIE;
}

/**
 * Fait la somme de toutes les têtes de boeufs d'une série,
 * puis la vide.
 * @return la sommes des têtes de boeufs de la série
 * @see Card#SumTotal(int)
 */
public int GatherOxHeadSerie(){
    int total = 0;
    for (Card card : this.serie) {
        total = card.SumTotal(total);
    }
    this.serie.clear();
    return total;
}

/**
 * Retourne une chaîne spécifiant les cartes d'un série
 * @return les cartes de la série
 * @see Card#toString()
 */
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("- série n° ").append(this.num_serie).append(" : ");
    for (int i = 0; i < this.serie.size(); ++i) {
        sb.append(this.serie.get(i).toString());
        if (i+1 != this.serie.size())
            sb.append(", ");
```

```
        }
        return sb.toString();
    }

    /**
     * Permet de connaître la différence entre une carte
     * spécifique et une carte de la série.
     * @param c la carte spécifique
     * @return la diférence les deux cartes
     * @see Card#CompareCard(Card, Card)
     */
    public int GetDifference(Card c) {
        return Card.CompareCard(c, this.serie.get(this.serie.size()-1));
    }
}
```

```
package entities.series;

import entities.cards.*;
import entities.players.Joueur;

import java.util.ArrayList;

public class AllSeries {
    private final ArrayList<Serie> series;
    public static final int MAX_SERIES = 4;

    /**
     * Constructeur permettant de créer de nouvelles
     * instances de type Series.
     * @param packet le paquet de cartes qui sera utilisé pour
     *               l'initilisation de la liste de séries
     * @see #CreateSeries(AllCards)
     */
    public AllSeries(AllCards packet){
        this.series = new ArrayList<>();
        this.CreateSeries(packet);
    }

    /**
     * Crée les 4 séries de cartes avant d'initiliser chaque
     * série avec une carte du paquet.
     * @param packet le paquet de cartes utilisé pour
     *               l'initialisation d'une série
     */
    private void CreateSeries(AllCards packet){
        for(int i = 0; i < MAX_SERIES; ++i)
            this.series.add(new Serie(packet));
    }
```

```
/**
 * Concatène toutes les séries.
 * @return les seriés concaténées
 * @see Serie#toString()
 */
public String toString() {
    StringBuilder sb = new StringBuilder();
    for(int i = 0; i < MAX_SERIES; ++i) {
        sb.append(this.series.get(i).toString());
        if (i + 1 != MAX_SERIES)
            sb.append("\n");
    }
    return sb.toString();
}

/**
 * Place la carte choisie dans une série si elle ne contient pas
 * déjà 5 cartes, sinon les têtes de boeufs des cartes se trouvant
 * dans la série sont affectées au joueur qui a choisi la carte.
 *
 * @param c la carte choisie
 * @param j le joueur
 * @see #CalculateMinValues(Card)
 * @see #SearchSerie(int, Card)
 * @see Joueur#GatherOxHead(int)
 * @see Joueur#RemoveCard(Card)
 * @see Serie#IsFull()
 * @see Serie#addCarte(Card)
 * @see Serie#GatherOxHeadSerie()
 */
public void PlaceCardInSerie(Card c, Joueur j, Serie s){
    if (s == null){
        int min_value = CalculateMinValues(c);
        s = SearchSerie(min_value, c);
        if (!s.IsFull())
            s.addCarte(c);
        else{
            j.GatherOxHead(s.GatherOxHeadSerie());
            s.addCarte(c);
        }
    }
    else {
        j.GatherOxHead(s.GatherOxHeadSerie());
        s.addCarte(c);
    }
    j.RemoveCard(c);
}

/**
 * Vérifie si un numéro spécifique se trouve dans l'intervalle [1;4].
 * @param num le numéro spécifique
 * @return vrai si c'est le cas, sinon faux
 */
public boolean HasSerie(int num){
    return num >= 1 && num <= 4;
}
```

```
/**
 * Trouve la série où sera placé la carte lorsque
 * la difference calculée entre la carte choisie et la
 * dernière carte carte d'une série est égale à la
 * valeur minimun calculer avant l'appel de cette méthode.
 * @param value la valeur minimale caculée avant l'appel
 *              de cette méthode
 * @param c la carte choisie
 * @return la série correspondante
 * @see Serie#GetDifference(Card)
 */
private Serie SearchSerie(int value, Card c){
    Serie s = null;
    for (int i = 0; i < MAX_SERIES; ++i){
        if (this.series.get(i).GetDifference(c) == value)
            s = this.series.get(i);
    }
    return s;
}

/**
 * Indique si une carte peut être posée dans l'une des séries
 * @param c la carte
 * @return true si la différence minimale parmis toutes les
 *         series est positif (donc jouable), sinon false
 * @see #CalculateMinValues(Card)
 */
public boolean CanPlaceCard(Card c){
    return CalculateMinValues(c) > 0;
}

/**
 * Calcule la valeur minimale entre la carte choisie et
 * la dernière carte de chaque série, afin de retourner
 * la valeur la plus petite, si elle est positive, qui sera
 * utilisée pour le placement future de la carte dans
 * une série, sinon renvoie un entier relatif.
 * @param c la carte choisie
 * @return le minmun s'il est positif sinon renvoie
 *         un entier relatif
 * @see #SearchMinValue(ArrayList)
 * @see Serie#GetDifference(Card)
 */
private int CalculateMinValues (Card c){
    ArrayList<Integer> list_diffrences = new ArrayList<>();
    for (int i = 0; i < MAX_SERIES; ++i)
        list_diffrences.add(this.series.get(i).GetDifference(c));

    return SearchMinValue(list_diffrences);
}

/**
 * Cherche la valeur la plus petite parmis les 4 différences
 * calculées.
 * @param liste la liste des différences calculées
 * @return le minmun s'il est positif sinon renvoie
 *         un entier relatif
 * @see #SearchMaxValue(ArrayList)
 */
```

```
    private int SearchMinValue(ArrayList<Integer> liste){
        int min = SearchMaxValue(liste);
        for (Integer integer : liste)
            if (integer > 0)
                if (integer < min)
                    min = integer;
        return min;
    }

    /**
     * Cherche la valeur maximale d'une liste de différences
     * calculées antérieurement.
     * @param liste la liste de différences
     * @return la valeur la plus grande
     */
    private int SearchMaxValue(ArrayList<Integer> liste){
        int max = liste.get(0);
        for (int i = 1; i < liste.size(); ++i)
            if(liste.get(i) > 0)
                if (liste.get(i) > max)
                    max = liste.get(i);
        return max;
    }

    /**
     * Permet d'accéder à l'inième série de la liste de séries.
     * @param i l'indice de la série
     * @return l'inième série
     */
    public Serie getSerie(int i) {
        return this.series.get(i-1);
    }
}
```

```
package appli;

import entities.cards.*;
import entities.players.*;
import entities.series.*;

import static util.Console.*;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;

/**
* Brief: L'application qui nous permet de jouer
*        une partie entre 2 et 10 joueurs inclus
* @author Anxian Zhang, Vick Ye
* @version 10
* @since 09/03/2022
*/
public class Application{
    public static int MAX_TOUR = 10;
```

```
/**
 * Enregistre tous les joueurs qui seront lus dans
 * le fichier config.txt, puis distribue 10 cartes
 * à chacun d'entre eux.
 * @param aj la liste des joueurs
 * @param packet le paquet qui sera utilisé
 */
public static void EnregistrementJoueurs(AllJoueur aj, AllCards packet){
    try {
        File f = new File("config.txt");
        Scanner sc_file = new Scanner(f);
        while(sc_file.hasNextLine()){
            while(sc_file.hasNextLine()){
                if (sc_file.hasNext()) {
                    String nomJ = sc_file.nextLine();
                    aj.addJoueur(nomJ, packet);
                    /* si nb_joueur > 10 le programme s'arrete
                     * car celui-ci ne peut plus piocher dans
                     * un paquet étant vide */
                }
            }
        }

        if (aj.IsBellowNbJoueurs()) {
            System.err.print("There is not enough player,");
            System.err.print(" please try it again.");
            System.exit(0);
        }

        System.out.println(aj.Remerciment());
        sc_file.close();
    } catch (FileNotFoundException e) {
        System.err.println("File cannot be found.");
        System.exit(0);
    }
}

/**
 * Demande au joueur de saisir un entier.
 * Si cet entier se trouve parmis les numéros
 * de cartes possédés, sort de la boucle, sinon
 * le programme redemande de saisir une valeur correcte.
 * @param j le joueur
 * @return la carte saisie
 */
public static Card InputCardJoueur(Joueur j){
    Scanner sc = new Scanner(System.in);
    do {
        if (sc.hasNextInt()){
            int n = sc.nextInt();
            if (j.HasCard(n))
                return j.FindCard(n);
        }
        else
            sc.next();
        System.out.print("Vous n'avez pas cette carte, ");
        System.out.print("saisissez votre choix : ");
    }while(true);
}
```

```
/**
 * Demande à un joueur de saisir un entier.
 * si cet entier se trouve parmis les numéros
 * de séries, sort de la boucle, sinon
 * le programme redemande de saisir une valeur correcte.
 * @param as a listes des séries
 * @return la série sélectionnée
 */
public static Serie InputSerieJoueur(AllSeries as){
    Scanner sc = new Scanner(System.in);
    do {
        if (sc.hasNextInt()){
            int n = sc.nextInt();
            if (as.HasSerie(n))
                return as.getSerie(n);
        }
        else
            sc.next();
        System.out.print("Ce n'est pas une série valide, ");
        System.out.print("saisissez votre choix : ");
    }while(true);
}

public static void main(String[] args){
    int tour = 0; // nombre de tour passé
    AllCards packet = new AllCards();
    packet.Generate_cards();
    packet.Rand_Card();
    AllSeries as = new AllSeries(packet);
    AllJoueur aj = new AllJoueur();

    // les cartes enregistré à chaque tour de table
    ArrayList<Card> cartes_jouer = new ArrayList<>();

    Card choix; // la carte choisis par un joueur
    Serie s; //serie choisie si la carte ne rentre dans 0 serie
    EnregistrementJoueurs(aj, packet);

    do {
        ++tour;
        /* choix de carte pour chaque joueur */
        for (int i = 0; i < aj.getNbjoueurs(); ++i) {
            System.out.println("A " + aj.getJoueur(i).getNomJoueur() +
                    " de jouer.");
            pause();
            System.out.println(as);
            System.out.println(aj.getJoueur(i));
            System.out.print("Saisissez votre choix : ");
            choix = InputCardJoueur(aj.getJoueur(i));
            cartes_jouer.add(choix);
            clearScreen();
        }

        // placement des cartes
        cartes_jouer.sort(Card::CompareCard);
        for (Card c : cartes_jouer) {
            if (as.CanPlaceCard(c))
                as.PlaceCardInSerie(c, aj.FindPlayer(c), null);
```

```
                else {
                    System.out.println(aj.toStringPassage(cartes_jouer,
                            true));
                    System.out.println(aj.FindPlayer(c).
                            toStringChooseSerie(c));
                    System.out.println(as);
                    System.out.print("Saisissez votre choix : ");
                    s = InputSerieJoueur(as);
                    as.PlaceCardInSerie(c, aj.FindPlayer(c), s);
                }
            }

            System.out.println(aj.toStringPassage(cartes_jouer, false));
            System.out.println(as);
            cartes_jouer.clear();
            System.out.println(aj.toStringPlayersOxHead());
            aj.InitGamersOxHeadPerTurn();
        }while (tour != MAX_TOUR);

        aj.OrderByTotalOxHead();
        System.out.println(aj.toStringFinalPlayersOxHead());
    }
}
```