Guillaume de La Grandiere, Luca Randazzo

Structures des Données et Algorithmes fondamentaux

Dossier de développement : recréer le jeu « The Game : Duel » en JAVA

Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

Table des matières

Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

Présentation

Introduction

Le jeu de société est un divertissement très populaire de nos jours. Cependant avec la récente crise sanitaire, il n'est plus possible de se réunir entre amis pour jouer. Il est donc nécessaire de pouvoir créer des versions informatiques de ces jeux pour permette aux gens de continuer à jouer. De plus, un programme pourra effectuer des taches fastidieuses plus rapidement qu'un humain ce qui rendrait les parties plus fluides et limiterais les fautes d'origine humaine. S'il s'agit d'un jeu de logique, un ordinateur pourra plus simplement détecter l'ensemble des combinaisons et indiquer aux joueurs si la partie est terminée.

Objectif du projet

Le but de ce projet est de recréer un jeu de société nommé « The Game : Duel », dans lequel deux joueurs s'affrontent et doivent placer des cartes numérotées jusqu'à ce que l'un des joueurs ne puisse plus jouer ou qu'il ait épuisé toutes ses cartes. The Game : Duel est composé de plusieurs éléments :

- Deux piles de jeu par joueurs : une pile croissante et une pile décroissante ;
- Une pioche par joueur de laquelle ils tireront respectivement leur main

Les cartes sont numérotées pour chaque joueur de 1 à 60, le 1 et le 60 étant présents dès le début de la partie respectivement sur la pile ascendante et la pile descendante. Le reste se trouve dans la pioche des joueurs. En début de partie, chaque joueur se constitue une main en piochant 6 cartes.

Le tour d'un joueur se déroule comme suit :

Le joueur doit jouer au minimum deux cartes sur n'importe quelles piles de la partie en respectant les règles de poses :

- Il ne peut poser au maximum qu'une carte par tour sur une pile adverse
- Il doit poser une carte de valeur strictement supérieure à celle de la carte du dessus de la pile ascendante et inversement pour la pile descendante
- Cependant si la carte correspond à exactement la valeur du dessus de la pile +/- 10 (respectivement pile descendante et ascendante) elle peut aussi être placée
- Pour placer sur la pile adverse les règles sont inversées, il faut donc placer une carte de valeur inférieure sur la pile ascendante et de valeur supérieure sur la pile descendante

Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

Une fois que le joueur a placé ses cartes, il doit alors piocher :

S'il a placé une carte sur une des piles du joueur adverse, il complète sa main à 6 cartes. Si ce n'est pas le cas, il pioche 2 cartes.

Un joueur perd s'il est dans l'incapacité de jouer au minimum deux cartes à son tour et il gagne s'il finit son tour en ayant posé la dernière carte de son deck (pioche + main).

Le programme devra donc être capable de recevoir des commandes utilisateur, de les interpréter, et, dans le cas où elles sont valides, les appliquer.

Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

Organisation du développement

Mode de fonctionnement

L'énoncé ne précisant pas de structure particulière, nous avons décidé de mettre les vérifications les plus simples dans les classes d'objet. Pour les vérifications les plus avancées, nous avons décidé de les intégrer dans le main.

La contrainte se trouvait uniquement sur l'affichage qui devait être sous la forme

```
NORD ^[01] v[60] (m6p52)
SUD  ^[01] v[60] (m6p52)
cartes NORD { 05 23 30 37 56 58 }
> 05^ 58v
2 cartes posées, 2 cartes piochées
NORD ^[05] v[58] (m6p50)
SUD  ^[01] v[60] (m6p52)
cartes SUD  { 03 07 18 21 44 50 }
> 03^' 50v
2 cartes posées, 2 cartes piochées
NORD ^[03] v[58] (m6p50)
SUD  ^[01] v[50] (m6p50)
cartes NORD { 23 30 37 46 54 56 }
>
```

Graphe de dépendance

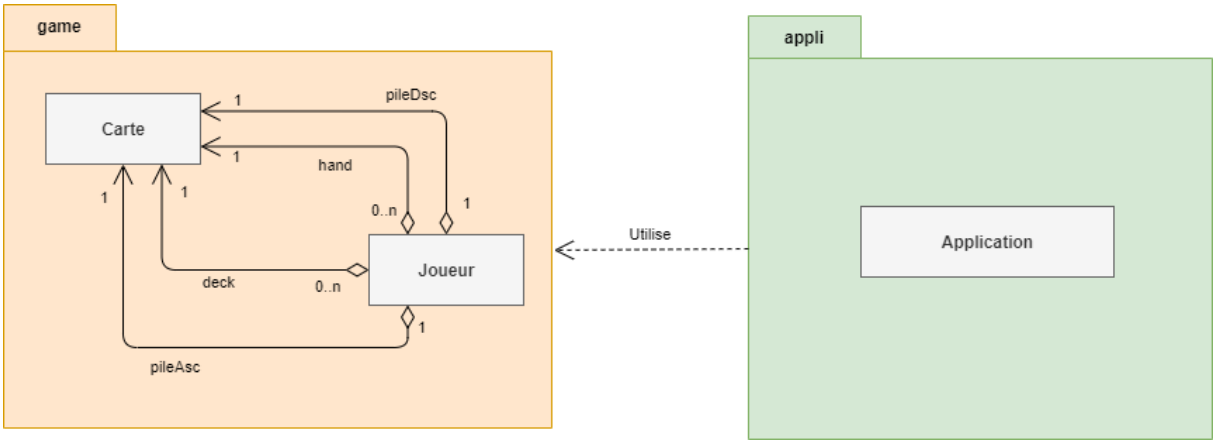
Pour la répartition des classes, nous avons opté pour uniquement deux classes d'objets et une classe d'application.

Les classes objet sont réparties comme suit :

Une classe Carte qui sert à définir une carte et si elle est plaçable ou non sur un plateau.

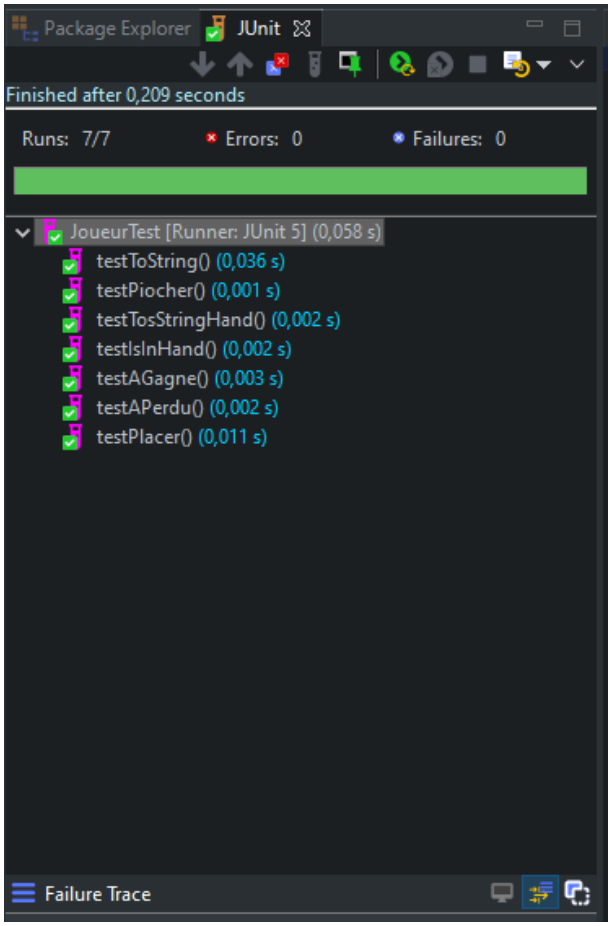
Une classe Joueur qui contient deux ArrayList : respectivement la main (hand) et la pioche (deck), les deux piles pileAsc et pileDsc de type carte qui correspondent à son plateau et le nom du joueur. Nous avons décidé de cette répartition car c'est celle qui crée le moins de difficultés dans l'accès aux différents éléments entre les deux joueurs et qui permet d'éviter des classes fantoches qui ne contiendrait qu'un simple getter et setter.

Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109



Tests unitaires

Nous avons effectué des tests unitaires sur toutes les fonctions des classes du package game. Nous n'avons pas effectué de tests unitaires sur la classe application car elles dépendent d'une saisie de l'utilisateur.



Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

Bilan du projet

Difficultés rencontrées

Les plus grosses difficultés ont été de débugger certaines parties du programme lors des tests unitaires. Ces erreurs ont surtout été dues à des erreurs de réflexion lors de la création de certaines fonctions qui n'avaient pas été vues lors de la première phase de développement. Ces problèmes ont cependant été résolus en relecture par l'autre binôme.

La plus grosse difficulté a été de pouvoir traiter et vérifier correctement plusieurs commandes données dans une seule ligne et de les traiter en continu les uns à la suite des autres.

Les autres difficultés ont pu être évitées grâce à un code structuré et lisible, qui était donc adapté au travail en binôme et qui a permis des relectures faciles.

Conclusion

Ce projet était une bonne introduction aux langages de programmation orienté objet et plus précisément au Java.

Malgré le manque de présentiel nous avons bien communiqué et la mise en place d'un répertoire Github pour partager le code a été efficace.

Les restrictions du projet sur le plan affichage ont été utiles en nous permettant de nous concentrer uniquement sur la partie logique plutôt que sur la problématique du format sous lequel afficher.

Nous nous sommes bien répartis les tâches et le travail a été effectué correctement par les deux membres du groupe. Ces résultats peuvent être dues au fait qu'il s'agisse de notre deuxième projet en binôme avec les mêmes membres ce qui nous a permis d'être bien coordonnés et de repérer plus vite les erreurs que l'autre binôme fait le plus souvent.

Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

Annexe

Classes

Package game

Carte.java

```
package game;

public class Carte {

    private final int valeur; // Valeur de la carte (1-60)

    /*
    [brief] : retourne la valeur de la Carte
    return : renvoie un int
     */
    public int getValeur() {
        return valeur;
    }

    public Carte(int val) {
        this.valeur=val; // Initialisation de la valeur de la carte
    }

    /*
    [brief] : vérifie si la carte est jouable sur l'entièreté des piles
    Joueur [in] : Joueur 1 pour accèder à ses piles
    Joueur [in] : Joueur 2 pour accèder à ses piles
    return : renvoie true si la carte est jouable, false dans le cas contraire
     */
    public boolean estJouable(Joueur j1,Joueur j2) {

        return this.estJouableAsc(j1) || this.estJouableDsc(j1) || this.estJou
ableAscAdv(j2) || this.estJouableDscAdv(j2);
    }

    /*
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

```
    [brief] : vérifie si la carte est jouable sur la pile ascendante d'un joue
ur
    Joueur [in] : pour récupérer la pile ascendante du joueur en question
    return : renvoie true si le coup est jouable, false dans le cas contraire
     */
    public boolean estJouableAsc(Joueur j) {
        return (this.valeur>j.getPileAsc().valeur) || (this.valeur==j.getPileA
sc().valeur-10);
    }

    /*
    [brief] : vérifie si la carte est jouable sur la pile descendante d'un jou
eur
    Joueur [in] : pour récupérer la pile descendante du joueur en question
    return : renvoie true si le coup est jouable, false dans le cas contraire
     */
    public boolean estJouableDsc(Joueur j) {
        return (this.valeur<j.getPileDsc().valeur) || (this.valeur==j.getPileD
sc().valeur+10);
    }

    /*
    [brief] : vérifie si la carte est jouable sur la pile ascendante d'un joue
ur considéré comme l'adversaire
    c'est à dire que la valeur de la carte doit être plus petite que la dernie
re carte posée sur cette pile
    Joueur [in] : pour récupérer la pile ascendante du joueur en question
    return : renvoie true si le coup est jouable, false dans le cas contraire
     */
    public boolean estJouableAscAdv(Joueur j) {
        return (this.valeur<j.getPileAsc().valeur);
    }

    /*
    [brief] : vérifie si la carte est jouable sur la pile descendante d'un jou
eur considéré comme l'adversaire
    c'est à dire que la valeur de la carte doit être plus grande que la dernie
re carte posée sur cette pile
    Joueur [in] : pour récupérer la pile descendante du joueur en question
    return : renvoie true si le coup est jouable, false dans le cas contraire
     */
    public boolean estJouableDscAdv(Joueur j) {
        return (this.valeur>j.getPileDsc().valeur);
    }

    /*
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

```
    [brief] : renvoie une chaîne de caractères correspondant à la valeur de la
 carte
    return : renvoie un String
     */
    public String toString() {

        String val = "";

        if(valeur < 10){ // Si la valeur de la carte est inférieur à 10 on for
mate l'affichage pour qu'un 0 soit présent

            val = "0";

        }

        val += String.valueOf(valeur);

        return val;


    }

}
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

Joueur.java

```
package game;

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

public class Joueur {
    private final ArrayList<Carte> deck; // ArrayList du type Carte correspond
ant à la pioche
    private final ArrayList<Carte> hand; // ArrayList du type Carte correspond
ant à la main
    private Carte pileAsc; // dernière carte posée sur la pile ascendante
    private Carte pileDsc; // dernière carte posée sur la pile descendante
    private final String nom; // nom du joueur
    static Random pioche = new Random();

    public Joueur(String nom) {
        this.deck= new ArrayList<>();
        for(int i=2;i<60;++i) { // Initialisation de la pioche
            this.deck.add(new Carte(i));
        }
        this.hand= new ArrayList<>(); // Initialisation de la main
        this.pileAsc= new Carte(1); // Initialisation de la pile Ascendante
        this.pileDsc= new Carte(60); // Initialisation de la pile Descendante
        this.nom=nom; // Initialisation du nom du joueur à partir du String pa
ssé en paramètre
    }


    /*
    [brief] : Renvoie le nom du joueur
    return : Renvoie un String
     */
    public String getNom(){

        return nom;

    }

    /*
    [brief] : Renvoie la carte sur la pile ascendante du joueur
    return : Renvoie un type Carte
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

```
     */
    public Carte getPileAsc() {

        return pileAsc;
    }

    /*
    [brief] : Renvoie la carte sur la pile descendante du joueur
    return : Renvoie un type Carte
     */
    public Carte getPileDsc() {
        return pileDsc;
    }

    /*
    [brief] : Renvoie la main du joueur sous forme d'une liste de cartes
    return : Renvoie une ArrayList du type Carte
     */
    public ArrayList<Carte> getHand(){
        return hand;
    }

    /*
    [brief] : Renvoie la pioche du joueur sous forme d'une liste de cartes
    return : Renvoie une ArrayList du type Carte
     */
    public ArrayList<Carte> getDeck(){
        return deck;
    }

    /*
    [brief] : Ajoute une carte à la main du joueur
    Carte [in] : Carte à ajouter
     */
    public void ajouterCarte(Carte carte) {
        this.hand.add(carte);
    }

    /*
    [brief] : Renvoie une chaîne de caractères correspondant à
    l'affichage du joueur (piles + nombre de cartes dans la main et dans la pi
oche)
    return : Renvoie un String
     */
    public String toString() {
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

```
        String asc = "";
        String dsc = "";

        if(pileAsc.getValeur() < 10)
            asc += "0";
        if(pileDsc.getValeur() < 10)
            dsc += "0";

        asc += String.valueOf(pileAsc.getValeur());
        dsc += String.valueOf(pileDsc.getValeur());

        return this.nom + " " +
                "^[" + asc + "] " +
                "v[" + dsc + "] " +
                "(m" + this.hand.size() +
                "p" + this.deck.size() + ")";
    }

    /*
    [brief] : Renvoie une chaîne de caractères correspondant à l'affichage de
la main du joueur
    return : Renvoie un String
     */
    public String toStringHand() {

        ArrayList<Carte> listeCroissante  = hand; // ArrayList de Cartes tempo
raire pour afficher la main dans l'ordre croissant

        for(int i = 0 ; i < listeCroissante.size() ; i++){ // Pour trier l'Arr
ayList précédemment créée

            for(int j = i+1 ; j < listeCroissante.size() ; j++){

                if(listeCroissante.get(j).getValeur() < listeCroissante.get(i)
.getValeur()){

                    Carte tmp = new Carte(listeCroissante.get(i).getValeur());
                    listeCroissante.set(i, listeCroissante.get(j));
                    listeCroissante.set(j, tmp);

                }

            }

        }
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

```
        StringBuilder sb= new StringBuilder(); // Utilisation de StringBuilder
 pour la création du String
        sb.append("cartes ");
        sb.append(this.nom);
        sb.append(" { ");
        for(Carte c : listeCroissante) {
            sb.append(c.toString()).append(" ");
        }
        sb.append("}");
        return sb.toString();
    }

    /*
    [brief] : Récupère une carte dans la pioche et l'ajoute à la main du joueu
r (et l'enlève de la pioche)
    int [in] : le nombre de cartes à piocher
    return : Renvoie un String corrrespondant au nombre de cartes piochées
     */
    public String piocher(int nb_cartes) {
        int i;
         // La fonction piocher tire aléatoirement une carte dans la pioche or
donnée
        for(i=0; i<nb_cartes && this.deck.size()>0;++i) {
            Carte carte= deck.get(pioche.nextInt(this.deck.size()));
            hand.add(carte);
            deck.remove(carte);
        }

        return ", " + i +" cartes piochées";

    }

    /*
    [brief] : Pour vérifier si une carte est dans la main du joueur
    Carte [in] : La carte à vérifier
    return : Renvoie true si la carte est présente, false dans le cas contrair
e
     */
    public boolean isInHand(Carte carte) {

        for(Carte c:hand) {

            if(c.getValeur()==carte.getValeur()) {
                return true;
            }
```

```
        }

        return false;
    }

    /*
    [brief] : Pour placer une carte sur une pile (du joueur ou de l'adversaire
)
    en vérifiant ou non la main du joueur
    Joueur [in-out] : sur les piles de quel joueur va-t-on placer la carte
    Carte [in] : carte à placer
    Boolean [in] : true si on joue sur la pile ascendante, false pour la pile
descendante
    Boolean [in] : true si l'on souhaite vérifier si le joueur possède la cart
e dans sa main, false dans le cas contraire
     */
    public void placer(Joueur j, Carte carte, boolean asc, boolean verifHand){

        boolean trouve = false;

        for(int i = 0 ; i < this.hand.size() ; i++){

            if(verifHand) {

                if (this.hand.get(i).getValeur() == carte.getValeur()) {
                    trouve = true;
                    this.hand.remove(i);
                    break;

                }
            }

        }

        if(verifHand && !trouve)
            return;

        if(asc)
            j.pileAsc = carte;
        else
            j.pileDsc = carte;

    }

    /*
    [brief] : Pour verifier si le joueur a perdu
```

```
    Joueur [in] : on inclut l'adversaire afin de pouvoir accèder à ses piles e
t vérifier si le joueur peut jouer dessus
    return : renvoie true si le joueur a perdu, false dans le cas contraire
     */
    public boolean aPerdu(Joueur adv) {
        boolean cartesJouables=true;
        for(Carte i:hand) {
            if(cartesJouables && i.estJouable(this,adv)) {

                Carte temp;

                if(cartesJouables && i.estJouableAsc(this)) {
                    temp=pileAsc;
                    this.placer(this, i, true, false);
                    cartesJouables=this.mainJouabletier2(adv, i,false);
                    this.placer(this, temp, true, false); // Restaure l'état d
e la pile ascendante
                }


                if(cartesJouables && i.estJouableDsc(this)) {
                    temp=pileDsc;
                    this.placer(this, i, false, false);
                    cartesJouables=this.mainJouabletier2(adv, i,false);
                    this.placer(this, temp, false, false); // Restaure l'état
de la pile descendante

                }

                if(cartesJouables && i.estJouableAscAdv(adv)) {
                    temp=adv.pileAsc;
                    this.placer(adv, i, true, false);
                    cartesJouables=this.mainJouabletier2(adv, i,true);
                    this.placer(adv, temp, true, false); // Restaure l'état de
 la pile ascendante de l'adversaire

                }

                if(cartesJouables && i.estJouableDscAdv(adv)) {
                    temp=adv.pileDsc;
                    this.placer(adv, i, false, false);
                    cartesJouables=this.mainJouabletier2(adv, i,true);
                    this.placer(adv, temp, false, false); // Restaure l'état d
e la pile descendante de l'adversaire
                }
```

```
            }
        }
        return cartesJouables;
    }

    /*
    [brief] : Fonction utilisée uniquement dans aPerdu(), elle sert à vérifier
 que le joueur peut bien jouer deux coups
    Joueur [in] : on inclut l'adversaire afin de pouvoir accèder à ses piles e
t vérifier si le joueur peut jouer dessus
    Carte [in] : la carte déjà vérifiée dans la fonction aPerdu(), on l'inclut
 ici pour ne pas la vérifier deux fois
    Boolean [in] : true si le coup testé précédent est sur la base adverse, fa
lse dans le cas contraire,
    cela permet à la fonction aPerdu() de renvoyer false si jamais les deux se
uls coups jouables sont sur les piles adverses
    return : renvoie true si le joueur ne peut jouer aucun coup, false dans le
 cas contraire
     */
    private boolean mainJouabletier2(Joueur adv, Carte i, boolean jouerAdv) {
        for(Carte k:hand) {
            if(i==k) {
                continue;
            }
            if(jouerAdv && (k.estJouableAsc(this) || k.estJouableDsc(this)))
                return false;
            if(!jouerAdv && k.estJouable(this,adv)) {
                return false;
            }
        }
        return true;
    }

    /*
    [brief] : pour vérifier si le joueur a gagné
    return : renvoie true si le joueur a gagné, false dans le cas contraire
     */
    public boolean aGagne() {
        return hand.size()==0 && deck.size()==0;
    }

    /*
    [brief] : pour enlever toutes les cartes de la main du joueur (à des fins
de tests uniquement)
     */
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

```
    public void resetHand(){

        hand.clear();

    }

}
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

Package appli

Application.java

```java
package appli;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

import game.Carte;
import game.Joueur;

public class Application {

    @SuppressWarnings("resource")
    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) {
        Joueur nord = new Joueur("NORD");
        Joueur sud = new Joueur("SUD ");

        Partie(nord, sud);

    }

    /*
    [brief] : Déroulement d'une partie
    Joueur [in-out] : Joueur 1
    Joueur [in-out] : Joueur 2
     */
    private static void Partie(Joueur j, Joueur adv) {

        boolean partieEnCours;

        j.piocher(6);  // Les deux joueurs piochent
        adv.piocher(6);// 6 cartes chacuns

        do {

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

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

            partieEnCours = tourJoueur(j, adv);

            if(!partieEnCours) {
                break;
            }

            System.out.println(j.toString());
            System.out.println(adv.toString());

            partieEnCours = tourJoueur(adv, j);


        } while(partieEnCours);
    }

    /*
    [brief] : tour d'un joueur
    Joueur [in-out] : joueur dont c'est le tour
    Joueur [in-out] : joueur adverse
    return : renvoie true si la partie est encore en cours (donc si aucun joue
ur n'a perdu ni gagné),
    false dans le cas contraire
     */
    private static boolean tourJoueur(Joueur j, Joueur adv) {


        String s;
        boolean coupValide=true;

        System.out.println(j.toStringHand());

        if(j.aPerdu(adv)) {

            System.out.println("Partie finie, " + adv.getNom() + " a gagne");
            return false;
        }

        System.out.print("> ");

        while(true) {

            if(!coupValide) { // Si le coup n'est pas valide on affiche "#> "
au lieu de "> "
                System.out.print("#> ");
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

```
            }
            s = sc.nextLine();
            coupValide = traitement(s, j, adv);
            if(!coupValide)
                continue;

            if(j.aGagne()){
                System.out.println("partie finie, " + j.getNom() + " a gagne")
;
                return false;
            }
            return true;

        }



    }

    /*
    [brief] : Permet de vérifier si une carte est spécifiée plusieurs fois lor
s d'un coup
    String[] [in] : commande liée au coup
    return : renvoie true si il y'a une redondance, false dans le cas contrair
e
     */
    private static boolean checkRedondance(String[] tab){

        ArrayList<Integer> nbs = new ArrayList<>();

        for(String mot : tab){

            nbs.add(Integer.parseInt(String.valueOf(mot.charAt(0)) + String.va
lueOf(mot.charAt(1))));

        }

        Set<Integer> set = new HashSet<>(nbs);

        return set.size() < nbs.size();

    }

    /*
    [brief] : permet de traiter un coup
    String [in] : commande liée au coup
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

```
    Joueur [in-out] : joueur en cours
    Joueur [in-out] : adversaire
    return : renvoie true si le coup est valide, false dans le cas contraire
     */
    private static boolean traitement(String s, Joueur j, Joueur adv) {

        String[] tab = s.split("\\s+");
        Carte tempAsc = j.getPileAsc();
        Carte tempDsc = j.getPileDsc();
        Carte tempAscAdv = adv.getPileAsc();
        Carte tempDscAdv = adv.getPileDsc();


        boolean commandeV;
        boolean jouerAdv;
        boolean completerHand = false;

        commandeV = commandeValide(tab, j, adv);

        j.placer(j, tempAsc, true, false);      // On restaure
        j.placer(j, tempDsc, false, false);     // l'état de
        j.placer(adv, tempAscAdv, true, false); // toutes les
        j.placer(adv, tempDscAdv, false, false);    // piles (adversaire et jo
ueur)

        if (!commandeV) {
            return false;
        }

        for (String mot : tab) {

            int nb = Integer.parseInt(String.valueOf(mot.charAt(0)) + String.v
alueOf(mot.charAt(1)));

            Carte c = new Carte(nb);

            if(mot.length() == 4) { // Si le mot fait 4 caractères c'est que l
'on souhaite jouer sur une des piles de l'adversaire
                jouerAdv = true;
                completerHand = true;
            }

            else {
                jouerAdv = false;
            }
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

```
            placer(String.valueOf(mot.charAt(2)), c, j, adv, jouerAdv, false);

        }

        System.out.print(tab.length + " cartes posées");

        if(completeterHand && j.getDeck().size() > 0) { // Si la main doit être completé (coup sur une pile adverse)
                                                        // et qu'il reste bien des cartes dans la pioche
            if(j.getDeck().size() <= 6 - j.getHand().size()) {     // Si le nombre de cartes dans la pioche est inférieur au
                                                                   // nombre de cartes à piocher alors on pioche toutes
                System.out.println(j.piocher(j.getDeck().size())); // les cartes situées dans la pioche
            }
            else {

                System.out.println(j.piocher(6 - j.getHand().size())); // Sinon on pioche le nombre de cartes nécessaires
            }
  // pour atteindre une main de 6 cartes
        }

        else if(j.getDeck().size() > 0) { // On vérifie bien que la pioche n'est pas vide

            if(j.getDeck().size() < 2) { // Si la pioche est composée d'une seule carte alors on pioche la carte
                System.out.println(j.piocher(1));
            }

            else { // Sinon on pioche deux cartes
                System.out.println(j.piocher(2));
            }

        }

        return true;

    }

    /*
    [brief] : Vérifie que la commande est valide
    String[] [in] : commande à vérifier
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

```
    Joueur [in-out] : joueur en cours
    Joueur [in-out] : joueur adverse
    return : renvoie true si la commande est valide, false dans le cas contrai
re
     */
    private static boolean commandeValide(String[] tab, Joueur j, Joueur adv)
{

        int cpt = 0;
        boolean jouerAdv;

        if(tab.length < 2) // Si la commande est inférieur à 2 coups alors on
renvoie directement false
            return false;  // puisque un tour doit contenir au moins deux coup
s joués

        for (String mot : tab) { // On parcourt la commande, où chaque mot est
 un coup
            if((mot.length() == 3 || mot.length() == 4) && (mot.charAt(2) == '
v' || mot.charAt(2) == '^')){ // On vérifie la syntaxe du mot

                if(mot.length() == 4 && mot.charAt(3) == '\''){ // Si le coup
concerne une pile adverse
                    jouerAdv = true; // Pour signifier que le coup se portera
sur une pile adverse
                    cpt++;
                }
                else if(mot.length() == 4) { // Sinon on renvoie false
                    return false;

                }
                else {
                    jouerAdv = false; // Pour signifier que le coup se portera
 sur une pile du joueur et non de l'adversaire
                }
                if(cpt > 1) { // Si il existe plus d'un coup sur une pile adve
rse, on renvoie false
                    return false;
                }

                String dizaine = String.valueOf(mot.charAt(0)); // On récupère
 la dizaine de la carte à jouer
                String unite = String.valueOf(mot.charAt(1));  // On récupère
l'unité de la carte à jouer
```

```
                if(!(Character.isDigit(unite.charAt(0)) && Character.isDigit(d
izaine.charAt(0)))) { // Si l'unité et la

                      // dizaine ne sont pas
                    return false;
                      // numériques

                      // on renvoie false
                }

                int nb = Integer.parseInt(dizaine + unite); // On converti les
 deux String en un int

                if(nb < 2 || nb > 59) // Si le nombre n'est pas compris entre
2 et 59, on renvoie false
                    return false;

                Carte c = new Carte(nb);

                if(!j.isInHand(c) || !c.estJouable(j, adv)) { // Si la carte n
'est pas jouable ou que le joueur
                    return false;                              // ne possède pa
s la carte dans sa main, on renvoie false
                }

                if(placer(String.valueOf(mot.charAt(2)), c, j, adv, jouerAdv,
true)) // On teste si l'on peut placer la carte

           // sur la pile voulue grâce à la fonction
                    return false;
          // placer et à l'argument test
             }
             else
                return false;
        }

        return !checkRedondance(tab); // On vérifie qu'il n'y ait pas de redon
dance

    }

    /*
    [brief] : placer une carte sur une pile
    String [in] : la pile sur laquelle placer la carte (ascendante ou descenda
nte)
    Carte [in] : la carte à placer
```

```
    Joueur [in-out] : le joueur en cours
    Joueur [in-out] : le joueur adverse
    Boolean [in] : pour savoir si l'on joue sur une pile du joueur ou de l'adv
ersaire
    Boolean [in] : pour savoir si l'on fait un test ou si l'on place
    return : renvoie true si la carte ne peut être posée, renvoie false dans l
e cas contraire
     */
    private static boolean placer(String sens, Carte c, Joueur j, Joueur a, bo
olean adv, boolean test) {

        if(!adv){ // Si on joue sur une pile du joueur

            if (sens.equals("^")) {
                if(c.estJouableAsc(j)) // Si la carte est jouable sur la pile
ascendante on la place
                    j.placer(j, c, true, !test); // On place la carte en vérif
iant ou non la main du joueur (en fonction de test)
                else
                    return true; // sinon on renvoie true
            }
            else {
                if (c.estJouableDsc(j)) // Si la carte est jouable sur la pile
 descendante on la place
                    j.placer(j,c,false, !test); // On place la carte en vérifi
ant ou non la main du joueur (en fonction de test)
                else
                    return true; // sinon on renvoie true

            }

        } else { // Sinon on joue sur une pile adverse

            if (sens.equals("^")) {
                if(c.estJouableAscAdv(a))  // Si la carte est jouable sur la p
ile ascendante adverse on la place
                    j.placer(a,c,true, !test); // On place la carte en vérifia
nt ou non la main du joueur (en fonction de test)
                else
                    return true; // sinon on renvoie true
            }
            else {
                if(c.estJouableDscAdv(a))  // Si la carte est jouable sur la p
ile descendante adverse on la place
                    j.placer(a,c,false, !test); // On place la carte en vérifi
ant ou non la main du joueur (en fonction de test)
```

```
                else
                    return true; // sinon on renvoie true
            }
        }

        return false; // Si tout s'est bien passé on renvoie false

    }

}
```


Guillaume DE LA GRANDIERE groupe 104
Luca RANDAZZO groupe 109

Package tests

CarteTest.java

```java
package tests;

import game.Carte;
import game.Joueur;
import org.junit.jupiter.api.Test;

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

class CarteTest {

    @Test
    void testEstJouableAsc() {

        Joueur nord = new Joueur("NORD");

        Carte c = new Carte(12);

        assertTrue(c.estJouableAsc(nord));

        nord.ajouterCarte(c);

        nord.placer(nord, c, true, true);

        Carte c2 = new Carte(5);

        assertFalse(c2.estJouableAsc(nord));

        Carte c3 = new Carte(2);

        assertTrue(c3.estJouableAsc(nord)); // Test dizaine en dessous



    }

    @Test
    void testEstJouableDsc() {

        Joueur nord = new Joueur("NORD");
```

```
        Carte c = new Carte(40);

        assertTrue(c.estJouableDsc(nord));

        nord.ajouterCarte(c);

        nord.placer(nord, c, false, true);

        Carte c2 = new Carte(52);

        assertFalse(c2.estJouableDsc(nord));

        Carte c3 = new Carte(50);

        assertTrue(c3.estJouableDsc(nord)); // Test dizaine au dessus


    }

    @Test
    void testEstJouableAscAdv() {

        Joueur sud = new Joueur("SUD");

        Carte c = new Carte(12);

        sud.ajouterCarte(c);

        sud.placer(sud, c, true, true);

        Carte c2 = new Carte(14);

        assertFalse(c2.estJouableAscAdv(sud));

        Carte c3 = new Carte(8);

        assertTrue(c3.estJouableAscAdv(sud));



    }

    @Test
    void testEstJouableDscAdv() {
```

```
        Joueur sud = new Joueur("SUD");

        Carte c = new Carte(43);

        sud.ajouterCarte(c);

        sud.placer(sud, c, false, true);

        Carte c2 = new Carte(32);

        assertFalse(c2.estJouableDscAdv(sud));

        Carte c3 = new Carte(53);

        assertTrue(c3.estJouableDscAdv(sud));

    }

    @Test
    void testEstToString() {

        Carte c = new Carte(12);

        assertEquals("12", c.toString());

        Carte c2 = new Carte(9);

        assertEquals("09", c2.toString());

    }
}
```

JoueurTest.java

```
package tests;

import game.*;

import org.junit.jupiter.api.Test;

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

class JoueurTest {

        @Test
        void testToString() {

            Joueur nord = new Joueur("NORD");
            Joueur sud = new Joueur("SUD");

            assertEquals("NORD ^[01] v[60] (m0p58)", nord.toString());
            assertEquals("SUD ^[01] v[60] (m0p58)", sud.toString());

            assertEquals(", 58 cartes piochées", nord.piocher(58));

            assertEquals("NORD ^[01] v[60] (m58p0)", nord.toString());

            nord.placer(nord, new Carte(2), true, true);

            assertEquals("NORD ^[02] v[60] (m57p0)", nord.toString());

            assertEquals(", 58 cartes piochées", sud.piocher(58));

            assertEquals("SUD ^[01] v[60] (m58p0)", sud.toString());

            sud.placer(nord, new Carte(5), false, true);

            assertEquals("SUD ^[01] v[60] (m57p0)", sud.toString());

            assertEquals("NORD ^[02] v[05] (m57p0)", nord.toString());

        }

        @Test
        void testPiocher() {
```

```
        Joueur test= new Joueur("TEST");

        assertEquals(0,test.getHand().size());
        assertEquals(58,test.getDeck().size());

        test.piocher(4);

        assertEquals(4,test.getHand().size());
        assertEquals(54,test.getDeck().size());

        test.piocher(0);


    }

    @Test
    void testIsInHand() {

        Joueur test= new Joueur("TEST");
        Carte ctest= new Carte(15);
        Carte ctest2=new Carte(15);

        test.ajouterCarte(ctest);

        assertTrue(test.isInHand(ctest2));

    }

    @Test
    void testTosStringHand() {

        Joueur test= new Joueur("TEST");
        Carte ctest1= new Carte(15);
        Carte ctest2= new Carte(8);

        test.ajouterCarte(ctest1);

        assertEquals("cartes TEST { 15 }", test.toStringHand());

        test.ajouterCarte(ctest2);

        assertEquals("cartes TEST { 08 15 }", test.toStringHand());
    }

    @Test
```

```
    void testPlacer() {

        Joueur nord = new Joueur("NORD");
        Joueur sud = new Joueur("SUD");
        Carte test= new Carte(15);

        nord.ajouterCarte(test);
        nord.placer(nord, test, true, true);

        assertEquals(15,nord.getPileAsc().getValeur());
        assertFalse(nord.isInHand(test));

        nord.ajouterCarte(test);
        nord.placer(nord, test, false, true);

        assertEquals(15,nord.getPileDsc().getValeur());
        assertFalse(nord.isInHand(test));

        nord.ajouterCarte(test);
        nord.placer(sud, test, true, true);

        assertEquals(15,sud.getPileAsc().getValeur());
        assertFalse(nord.isInHand(test));

        nord.ajouterCarte(test);
        nord.placer(sud, test, false, true);

        assertEquals(15,sud.getPileDsc().getValeur());
        assertFalse(nord.isInHand(test));

    }

    @Test
    void testAPerdu() {

        Joueur nord = new Joueur("NORD");
        Joueur sud = new Joueur("SUD");

        nord.ajouterCarte(new Carte(2));
        nord.ajouterCarte(new Carte(3));
        nord.ajouterCarte(new Carte(4));
        nord.ajouterCarte(new Carte(59));

        assertFalse(nord.aPerdu(sud));

        nord.placer(nord, new Carte(2), false, true);
```

```
            nord.placer(nord, new Carte(4), true, true);

            assertTrue(nord.aPerdu(sud));


            sud.ajouterCarte(new Carte(12));
            sud.ajouterCarte(new Carte(2));
            sud.ajouterCarte(new Carte(58));
            sud.ajouterCarte(new Carte(4));

            sud.placer(sud, new Carte(2), true, true);

            assertTrue(nord.aPerdu(sud));

            sud.placer(sud, new Carte(58), false, true);

            assertTrue(nord.aPerdu(sud));


            sud.placer(sud, new Carte(4), true, true);


            assertFalse(nord.aPerdu(sud));

        }

        @Test
        void testAGagne(){

            Joueur nord = new Joueur("NORD");

            nord.piocher(58);

            nord.resetHand();

            assertTrue(nord.aGagne());

        }


}
```