PROJET JAVA

Tracy HONG
Nahean BADAR
Groupe 106

THE GAME - Le Duel

IUT de Paris - Rives de Seine
Université de Paris

Table des matières

Introduction

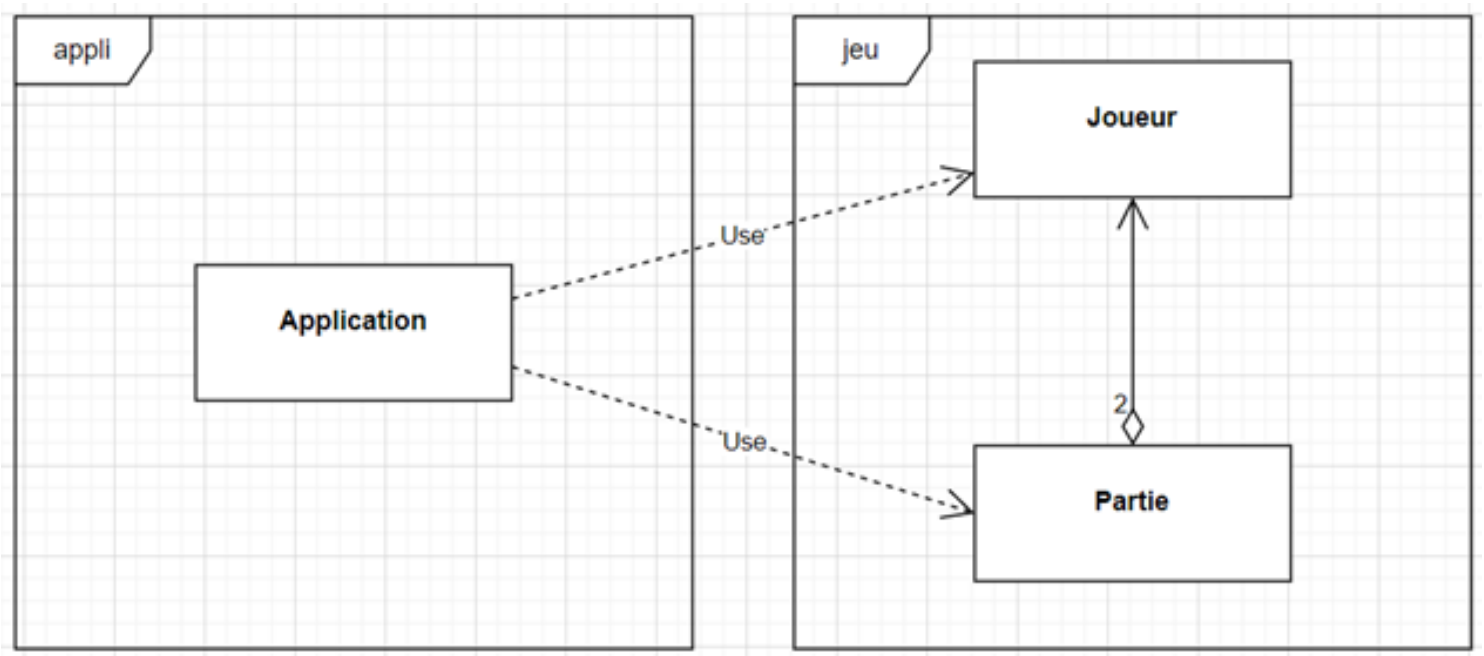
Nous devons programmer un jeu en java 8 qui s'intitule " The game : le duel ". Nous avons deux joueurs : d'une part, on a le joueur NORD et d'autre part il y a le joueur SUD. Il y a au total 60 cartes. La carte n°1 et 60 forment 2 piles où 1 est la pile ascendante et 60 est la pile descendante. Le reste des 58 cartes forme une pioche, qu'on mélange. Ensuite, pour commencer le jeu, chaque joueur pioche 6 cartes. NORD et SUD ont donc six cartes dans leurs mains et il reste 52 dans la pioche.

En ce qui concerne la partie, les deux joueurs jouent chacun leur tour. Il y a un affichage des deux piles pour chaque joueur avec le nombre de cartes en main et celui de la pioche, puis les cartes du joueur courant. Cela permet de lire et vérifier si le coup est possible ou non pour les joueurs.

Au début, c'est le joueur NORD qui commence. Il doit alors poser au moins deux cartes parmi celles qu'il a dans sa main. Pour que le joueur puisse poser sa carte dans sa base, s'il veut poser ses cartes dans la pile ascendante, il faut que celle-ci ait une valeur plus grande que celle qui est déjà posée avant. De plus, il est aussi possible de la poser si la valeur de sa carte est à la dizaine au-dessous. S'il veut les poser dans la pile descendante, c'est l'inverse : la valeur doit être plus petite ou à la dizaine au-dessus. Ensuite, si le joueur veut jouer sur la base adverse, il ne peut poser qu'une seule carte. La valeur de la carte doit être plus petite sur la pile ascendante, et plus grande sur la pile descendante. Il y a également des règles pour piocher les cartes. Si le joueur joue uniquement sur ses propres bases, il pioche alors deux cartes. Sinon, il doit piocher n cartes tel que [nombre de cartes en main] + n soit égale à 6, c'est-à-dire, jusqu'à compléter sa main.

Le jeu se termine lorsqu'un des deux joueurs ne peut plus mettre deux cartes minimums. Si c'est le cas, ce joueur courant perd la partie. Donc finalement, pour gagner la partie du jeu, il faut que le joueur courant ait dans sa main aucune carte.

Diagramme de classe



Explication des classes

Avant de pouvoir structurer nos classes, il était bien important de lire plusieurs fois le sujet. Les instructions mentionnées pour le jeu étaient claires donc on a pris note de tous les détails importants.

On a vraiment hésité sur notre structuration. On pensait faire plein de classe car on pensait que chaque objet devait être une classe comme la pioche, les cartes, [...], mais c'était au début quand nous n'avions pas beaucoup de connaissance en java.

Finalement, on a choisi de créer trois classes : Joueur, Partie et Application.

Tests unitaires

Voici nos tests unitaires. Par manque de connaissance et du temps il y a peu de test.

```java
/**
 * @sujet : Projet BPO : java. Jeu "The GAME - Le Duel"
 * @author Tracy HONG et Nahean BADAR
 * @date : Février - Mars 2021
 */
package Test;
import static org.junit.Assert.*;
import org.junit.Test;
import jeu.Joueur;

public class TestJoueur {

    @Test
    public void test() {
        Joueur NORD=new Joueur();
        Joueur SUD=new Joueur();
        assertTrue(NORD.getCarteEnMain_size()==6);
        assertTrue(SUD.getCarteEnMain_size()==6);

        assertTrue(NORD.getPioche_size()==58);
        assertTrue(SUD.getPioche_size()==58);
    }

    public void testSetter() {
        Joueur J=new Joueur();

        J.setPileAsc(4);
        assertTrue(J.getPileAsc()==4);
        J.setPileDesc(28);
        assertTrue(J.getPileDesc()==28);
    }

    public void testPioche() {
        Joueur J=new Joueur();
        int carteEnMain=J.getCarteEnMain_size();
        J.piocher();
        assertTrue(J.getCarteEnMain_size()>carteEnMain);
    }
    public void testRetirerCarte() {
        Joueur J=new Joueur();
        int carteEnMain=J.getCarteDansMain(0);
        J.retirerCarte(0);
        assertTrue(J.getCarteDansMain(0)!=carteEnMain);
    }
}
```

```java
package Test;
import static org.junit.Assert.*;
import org.junit.Test;
import appli.Application;
import jeu.Joueur;
```

```java
import jeu.Partie;

public class TestPartie {
    @Test
    public void testSaisieSansPile() {
        String[] saisie= new String[]{"1a","2b","3c","4d"};
        int[] saisie1=new int[] {1,2,3,4};
        int[] saisieSansPile=Partie.saisieSansPile(saisie);
        assertEquals(saisieSansPile,saisie1);
    }
}
```

Bilan

La réalisation de ce projet nous a été bénéfique sur plusieurs points.

D'abord, le travail collaboratif fut positif malgré le fait que nous étions à distance pendant ces dernières semaines, nous avons facilement réussi à coder ensemble. Ainsi, le développement du code a été plus efficace et cela nous a permis de nous entraider. En effet, lorsque l'une d'entre nous n'avait pas compris un point, les appels et partages d'écrans sur Discord nous ont été d'une grande utilité.

Pour finir de programmer le jeu complet, nous avions rencontré pas mal de difficulté. L'un de nos principaux problèmes était le manque de connaissance en java. En effet, c'est la première fois qu'on voyait la programmation orienté objet mais au fur et à mesure des cours, on a pu développer et approfondir nos connaissances. Donc pour avancer sur le projet, nous avons fait toutes les recherches nécessaires, tel que savoir comment et quand utiliser le *"static",* comment bien faire une structuration de classe ou encore les fonctions de java, on avait besoin de souvent aller chercher la documentation, par exemple les *ArrayList*.

Le projet nous semblait au début assez conséquent mais on a vite pris l'habitude de découper les spécifications compliquées en plusieurs pour que ça soit plus simple. Par exemple, pour vérifier la saisie, nous avons fait au départ une méthode *verifSaisie* mais on s'est vite rendu compte qu'on pouvait séparer chaque critère en petites méthodes *private* pour rendre la programmation plus simple.

En ce qui concerne les problèmes rencontrés lors du codage, il y'en a eu beaucoup. D'abord, pour la méthode *saisie,* nous avions du mal pour la décomposer et aussi, elle ne retournait pas un tableau avec les caractères pour les piles comme on le voulait. On a trouvé la solution avec le *split.* Aussi, on a choisi d'utiliser la fonction *ParseInt* pour transformer le tableau de String de la saisie avec pile pour avoir un tableau de *int* sans les caractères de pile pour faciliter la vérification par la suite.

Ensuite, pour l'affichage, il y a également eu de nombreuses confusions à propos des variables. Nos variables étaient des références plutôt que de copier les valeurs comme on le voulait. On a pu régler certains soucis en débuguant et en refaisant le code mais parfois ce n'était pas toujours évident. Par exemple, on voulait faire un joueur temporaire pour tester les coups possibles en créant un deuxième constructeur qui permettra de recopier toutes les données du joueur courant : *Joueur JtmpCourant=new Joueur(Jcourant)* . On a pu voir que lorsqu'on met *"="* ça fait un référencement. On a donc eu un souci avec ça dans notre méthode *verifTestPoserCarte*, à cause de ce problème de référence, lorsqu'on voulait retirer une carte de la main du joueur temporaire, elle se retirait aussi sur le vari joueur. On était alors bloqué sur quelques problèmes similaires. C'est alors qu'on vous a sollicité pour nous venir en aide afin. De plus, pour les boucles *for each,* il y avait des complications donc nous avons préféré d'utiliser les boucles *for* « classiques ».

Lorsque nous faisions la vérification, il y avait pas mal de soucis. Tout d'abord, il y avait un problème dans la main (application) mais elle a été vite réglée car le vrai problème venait dans une méthode qu'on avait créé *« inverserJoueur »*. Elle permettait d'inverser les joueurs

à chaque fin de tour mais celui-ci ne fonctionnait pas car il récupérait seulement le joueur courant. Or, le joueur adverse n'était pas modifié. C'est pourquoi on avait décidé de le mettre directement dans le *main.* Ensuite, en ce qui concerne la méthode de vérification saisie il y avait plein de bug donc on a découpé avec des *« if »* pour mieux comprendre en débuggant la fonction. Puis, ailleurs où on avait des difficultés était dans les *"exceptions"* et les *"test unitaire"* car on ne sait pas comment exactement les faire.

Néanmoins, à part cela, nous avons pu finir le jeu à temps et nous sommes très fières car c'est la première fois qu'on a pu compléter un projet en programmation. Malgré les différents types de complications que nous avons pu rencontrer, c'était intéressant de faire ce projet.

Code source complet

```java
/**
 * @sujet : Projet BPO : java. Jeu "The GAME - Le Duel"
 * @author : Tracy HONG et Nahean BADAR
 * @date : Février - Mars 2021
 */
package jeu;

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

public class Joueur {
    // Données ----------------------------------------------------------
---------------------------------
    private int pileAsc;
    private int pileDesc;
    private ArrayList<Integer> pioche;
    private ArrayList<Integer> carteEnMain;

    // Constante --------------------------------------------------------
---------------------------------
    public static final int MAX_MAIN=6;
    private static final int MIN_PIOCHE=2;
    private static final int MAX_PIOCHE=59;


    //Constructeur -----------------------------------------------------
-------------------------------------
    public Joueur() {
        this.pileAsc=1;
        this.pileDesc=60;

        initPioche();
        carteEnMain=new ArrayList<Integer>();
        for(int i=0;i<MAX_MAIN;++i) {
            piocher();
        }
    }
    //Constructeur pour joueur tmp -------------------------------------
---------------------------------------
    /**
     * Constructeur pour joueur tmp
     * @param J : joueur à copier
     */
    public Joueur(Joueur J) {
        this.pileAsc=J.pileAsc;
        this.pileDesc=J.pileDesc;
        pioche = new ArrayList<Integer>(J.pioche);

        carteEnMain=new ArrayList<Integer>(J.carteEnMain);
    }

    // Getter -----------------------------------------------------------
---------------------------------------
    /**
     * récupère la pile ascendante
     * @return la pile ascendante du joueur
```

```
 */
public int getPileAsc() {
    return this.pileAsc;
}
/**
 * récupère la pile descendante
 * @return la pile descendante du joueur
 */
public int getPileDesc() {
    return this.pileDesc;
}
/**
 * récupère le nombre des cartes en main
 * @return le nombre des cartes en main
 */
public int getCarteEnMain_size() {
    return this.carteEnMain.size();
}
/**
 * récupère les cartes en main
 * @return les cartes en main
 */
public ArrayList<Integer> getCarteEnMain() {
    return this.carteEnMain;
}
/**
 * récupère une carte en main à la position i
 * @param i
 * @return la carte en main à la position i
 */
public int getCarteDansMain(int i) {
    return this.carteEnMain.get(i);
}
/**
 * récupère le nombre de carte dans la pioche
 * @return le nombre de carte dans la pioche
 */
public int getPioche_size() {
    return this.pioche.size();
}

//setter
/**
 * modifie la pile ascendante
 * @param carte
 */
public void setPileAsc(int carte) {
    this.pileAsc=carte;
}
/**
 * modifie la pile descendante
 * @param carte
 */
public void setPileDesc(int carte) {
    this.pileDesc=carte;
}

//pioche
/**
 * initialise la pioche
 */
```

```
    private void initPioche() {
        pioche = new ArrayList<Integer>();
        for (int i = MIN_PIOCHE; i <= MAX_PIOCHE; i++) {
           pioche.add(i);
        }
       Collections.shuffle(pioche);
    }
    /**
     * verifier si la pioche est vide
     * @return vrai ou faux
     */
    public boolean piocheIsEmpty() {
        return pioche.isEmpty();
    }
    /**
     * pioche une carte et la met dans la main
     */
    public void piocher() {
        if(!piocheIsEmpty()) {
            int cartePiochée=pioche.remove(0);
            this.carteEnMain.add(cartePiochée);
            Collections.sort(this.carteEnMain);
        }
    }
    /**
     * retire la carte de la main à la position i
     * @param i
     */
    public void retirerCarte(int i) {
        this.carteEnMain.remove(i);
    }
}
```

```java
/**
 * @sujet : Projet BPO : java. Jeu "The GAME - Le Duel"
 * @author : Tracy HONG et Nahean BADAR
 * @date : Février - Mars 2021
 */

package jeu;
import java.util.Arrays;
import java.util.Scanner;


public class Partie {

    //SAISIE
    /**
     * demande la saisie et stocke dans tableau de string avec chaque coup
par case
     * @return tableau de string de la saisie
     */
    private static String[] saisir() {
        @SuppressWarnings("resource")
        Scanner sc = new Scanner(System.in);
        String saisie = sc.nextLine();

        return saisie.split(" ");
    }


    /**
     * converti le tableau de saisie(saisieAvecPile) avec caractere pour
les piles en tableau de int sans caractere (tab_saisieSansPile)
     * @param saisieAvecPile
     * @return tab_saisieSansPile
     */
    public static int[] saisieSansPile(String[] saisie) {
        int coup;
        String[] saisieAvecPile=new String[saisie.length];

        for(coup=0;coup<saisieAvecPile.length;coup++) {
            //saisieAvecPile[coup]=saisieAvecPile[coup].replace("v",
"").replace("^", "").replace("\'","");
            saisieAvecPile[coup]=saisie[coup].replaceAll("[^0-9]", "");
        }

        int[] saisieSansPile=new int[saisieAvecPile.length];
        for(coup=0;coup<saisieAvecPile.length;coup++) {
            saisieSansPile[coup]=Integer.parseInt(saisieAvecPile[coup]);
        }
        return saisieSansPile;
    }
    //SAISIE
    //************************************************************
    /**
     * demande et verifie la saisie
     * @param Jcourant
     * @param Jadverse
     * @return saisieAvecPile : tableau de string de la saisie
     */
    public static String[] saisie(Joueur Jcourant,Joueur Jadverse) {
        System.out.print("> ");
        String[] saisieAvecPile=saisir();
```

```
        int[] saisieSansPile=saisieSansPile(saisieAvecPile);


while(!verifierSaisie(saisieAvecPile,saisieSansPile,Jcourant,Jadverse)) {
            System.out.print("#> ");

            saisieAvecPile=saisir();
            //tab_saisieAvecPile=saisieAvecPile(saisie);
            saisieSansPile=saisieSansPile(saisieAvecPile);
        }
        return saisieAvecPile;
    }



    //VERIFICATION SAISIE

//**********************************************************************
**********************************
    /**
     * verifie si chaque coup dans la saisie a le bon format : 2 chiffres
avec v ou ^ ou v'ou ^' et une seule fois '
     * @param saisie
     * @return vrai ou faux
     */
    private static boolean verifBonFormat(String[] saisie) {
        int cptSaisieAdverse=0; //si cpt = 2 alors le joueur veut poser une
carte 2fois sur la pile adverse
        int cptValide=0;//cpt si le coup est valide

        for(String coup : saisie) {


            if(coup.length()==3 || coup.length()==4) {


if(Character.isDigit(coup.charAt(0))&&Character.isDigit(coup.charAt(1))) {

                    if(coup.charAt(2)==118||coup.charAt(2)==94) {

                        if(coup.length()==4 && coup.charAt(3)==39)
                            cptSaisieAdverse++;
                        cptValide++;
                    }
                }
            }
            if(cptSaisieAdverse>=2) {
                return false;
            }
        }

        if (cptValide==saisie.length)
            return true;
        else return false;
    }
    /**
     * verifie s'il n'y a pas de doublon dans la saisie
     * @param saisie
     * @return vrai ou faux
```

```
     */
    private static boolean verifPasDoublon(int[] saisie) {
        int carteTmp=0;
        for(int i=0;i<saisie.length;i++) {
            carteTmp=saisie[i];
            for(int j=i+1;j<saisie.length;j++) {
                if(carteTmp==saisie[j])
                    return false;
            }
        }
        return true;
    }
    /**
     * verifie si la saisie est croissante
     * @param saisie
     * @return vrai ou faux
     */
    private static boolean verifCroissant(int[] saisie) {
        int[] tabTmp=new int[saisie.length];
        for(int coup=0;coup<saisie.length;coup++) { //for(int
i=0;i<saisie.length;i++)
            tabTmp[coup]=saisie[coup];
        }

        Arrays.sort(tabTmp);

        if(Arrays.equals(tabTmp, saisie))
            return true;


        return false;

    }

    /**
     * verifie si les coups saisis sont dans la main
     * @param saisie
     * @param Jcourant
     * @return vrai ou faux
     */
    private static boolean verifMain(int[] saisie,Joueur Jcourant) {
        /*(int i=1;i<saisie.length;i++) {
            if(!Jcourant.getCarteEnMain().contains(saisie[i]));
                return false;
        }*/
        int cpt=0;
        for(int coup=0;coup<saisie.length;coup++) {
            for(int carte=0;carte<Jcourant.getCarteEnMain_size();carte++) {
                if(Jcourant.getCarteDansMain(carte)==saisie[coup]) {
                    cpt++;
                    continue;
                }
            }
        }
        if(cpt==saisie.length)
            return true;
        else return false;
    }
    /**
     * verifie si les coups de la saisie sont valides
     * @param saisieAvecPile
```

```
     * @param saisieSansPile
     * @param Jcourant
     * @param Jadverse
     * @return vrai ou faux
     */
    private static boolean verifPoserCarte(String[] saisieAvecPile,int[]
saisieSansPile, Joueur Jcourant, Joueur Jadverse) {
        Joueur JtmpCourant=new Joueur(Jcourant);//joueur temporaire pour
poser des cartes temporairement
        Joueur JtmpAdverse=new Joueur(Jadverse);

        for(int coup=0; coup<saisieAvecPile.length;coup++) {
            if(saisieAvecPile[coup].length()==4) { //si il y a 4 caracteres
donc xxx'
                if(saisieAvecPile[coup].charAt(2)==94) {//^
                    if(saisieSansPile[coup]<JtmpAdverse.getPileAsc() ) {
                        JtmpCourant.setPileAsc(saisieSansPile[coup]);
                        for(int
carte=0;carte<JtmpCourant.getCarteEnMain_size();carte++) {

if(saisieSansPile[coup]==JtmpCourant.getCarteDansMain(carte)) {
                                JtmpCourant.retirerCarte(carte);
                                break;
                            }
                        }
                        continue;
                    }
                    else return false;
                }
                else if(saisieAvecPile[coup].charAt(2)==118) {//v
                    if(saisieSansPile[coup]>JtmpAdverse.getPileDesc()) {
                        JtmpCourant.setPileAsc(saisieSansPile[coup]);
                        for(int
carte=0;carte<JtmpCourant.getCarteEnMain_size();carte++) {

if(saisieSansPile[coup]==JtmpCourant.getCarteDansMain(carte)) {
                                JtmpCourant.retirerCarte(carte);
                                break;
                            }
                        }
                        continue;
                    }
                    else return false;
                }
            }
            else {//si il y a 3 caracteres
                if(saisieAvecPile[coup].charAt(2)==94) {//^
                    if(saisieSansPile[coup]>JtmpCourant.getPileAsc() ||
saisieSansPile[coup]==(JtmpCourant.getPileAsc()-10)) {
                        JtmpCourant.setPileAsc(saisieSansPile[coup]);
                        for(int
carte=0;carte<JtmpCourant.getCarteEnMain_size();carte++) {

if(saisieSansPile[coup]==JtmpCourant.getCarteDansMain(carte)) {
                                JtmpCourant.retirerCarte(carte);
                                break;
                            }
                        }
                        continue;
                    }
                    else return false;
```

```
                }
                else if(saisieAvecPile[coup].charAt(2)==118) {//v
                    if(saisieSansPile[coup]<JtmpCourant.getPileDesc() ||
saisieSansPile[coup]==(JtmpCourant.getPileDesc()+10)) {
                        JtmpCourant.setPileAsc(saisieSansPile[coup]);
                        for(int
carte=0;carte<JtmpCourant.getCarteEnMain_size();carte++) {

if(saisieSansPile[coup]==JtmpCourant.getCarteDansMain(carte)) {
                                JtmpCourant.retirerCarte(carte);
                                break;
                            }
                        }
                    }
                    else return false;
                }
            }
        }
        return true;
    }

    //verifier saisie
    /**
     * fait la verification complete de la saisie
     * @param saisieAvecPile
     * @param saisieSansPile
     * @param Jcourant
     * @param Jadverse
     * @return vrai ou faux
     */
    private static boolean verifierSaisie(String[] saisieAvecPile,int[]
saisieSansPile, Joueur Jcourant, Joueur Jadverse) {

/*if(verifCoupSaisi(saisieAvecPile)&&verifPasDoublon(saisieSansPile)

&&verifCroissant(saisieSansPile)&&verifMain(saisieSansPile,Jcourant)

&&verifPoserCarte(saisieAvecPile,saisieSansPile,Jcourant,Jadverse)){
            return true;
        }*/
        //**************//pleins de if pour mieux comprendre quand on debug

        if(verifBonFormat(saisieAvecPile)){
            if(verifPasDoublon(saisieSansPile)) {
                if(verifCroissant(saisieSansPile)) {
                    if(verifMain(saisieSansPile,Jcourant)){

if(verifPoserCarte(saisieAvecPile,saisieSansPile,Jcourant,Jadverse)) {
                                return true;
                            }
                        }
                    }
                }
            }
        }
        return false;

    }

//*************************************************************************
*****
    ////****poser une carte
```

```
    /**
     * pose les cartes sur les piles
     * @param Jcourant
     * @param Jadverse
     * @param saisieAvecPile
     * @param saisieSansPile
     * @return le nombre de cartes posées
     */
    public static int poserCarte(Joueur Jcourant,Joueur Jadverse,
String[]saisieAvecPile, int[]saisieSansPile) {
        int nb_cartesPosees=0;

        for(String coup : saisieAvecPile) {

            if(coup.length()==3) {//dans notre base
                if(coup.charAt(2)==94) {//^
                    Jcourant.setPileAsc(saisieSansPile[nb_cartesPosees]);
                    for(int
carte=0;carte<Jcourant.getCarteEnMain_size();carte++) {

if(saisieSansPile[nb_cartesPosees]==Jcourant.getCarteDansMain(carte)) {
                            Jcourant.retirerCarte(carte);
                            continue;
                        }
                    }
                }
                if(coup.charAt(2)==118) {//v
                    Jcourant.setPileDesc(saisieSansPile[nb_cartesPosees]);
                    for(int
carte=0;carte<Jcourant.getCarteEnMain_size();carte++) {

if(saisieSansPile[nb_cartesPosees]==Jcourant.getCarteDansMain(carte)) {
                            Jcourant.retirerCarte(carte);
                            continue;
                        }
                    }
                }
            }
            else if(coup.length()==4) {//dans base adverse
                    if(coup.charAt(2)==94) {//^

Jadverse.setPileAsc(saisieSansPile[nb_cartesPosees]);
                        for(int
carte=0;carte<Jcourant.getCarteEnMain_size();carte++) {

if(saisieSansPile[nb_cartesPosees]==Jcourant.getCarteDansMain(carte)) {
                                Jcourant.retirerCarte(carte);
                                continue;
                            }
                        }
                    }
                    if(coup.charAt(2)==118) {//v

Jadverse.setPileDesc(saisieSansPile[nb_cartesPosees]);
                        for(int
carte=0;carte<Jcourant.getCarteEnMain_size();carte++) {

if(saisieSansPile[nb_cartesPosees]==Jcourant.getCarteDansMain(carte)) {
                                Jcourant.retirerCarte(carte);
                                continue;
                            }
```

```
                        }
                    }
                }
                nb_cartesPosees++;
            }
            return nb_cartesPosees;
        }

        ////****fin de tour piocher
        /**
         * pioche les cartes à la fin de chaque tour : 2 si on pose sur notre
base et
         * piocher jusqu'à compléter la main si on pose chez l'adversaire
         *
         * @param Jcourant
         * @param saisieAvecPile
         * @return nombre de cartes piochées
         */
        public static int piocherFinTour(Joueur Jcourant,
String[]saisieAvecPile) {
            int nb_cartesPiochees=0;
            boolean poserBaseAdverse=false;
            for(String coup : saisieAvecPile) {
                if(coup.length()==4) {
                    poserBaseAdverse=true;
                    break;
                }
            }
            if(poserBaseAdverse) {
                while(Jcourant.getCarteEnMain_size()!=Joueur.MAX_MAIN) {
                    Jcourant.piocher();
                    if(!Jcourant.piocheIsEmpty())
                        nb_cartesPiochees++;
                }
            }

            else{
                Jcourant.piocher();
                Jcourant.piocher();
                if(!Jcourant.piocheIsEmpty())
                    nb_cartesPiochees=2;

            }
            return nb_cartesPiochees;
        }

}
```

```java
/**
 * @sujet : Projet BPO : java. Jeu "The GAME - Le Duel"
 * @author : Tracy HONG et Nahean BADAR
 * @date : Février - Mars 2021
 */

package appli;



import jeu.Joueur;
import jeu.Partie;

public class Application {
    //LES AFFICHAGES  ------------------------------------------------------
------------------------------------------------------

    /**
     * Affiche le plateau : les bases de chaque joueur et les cartes en
main du joueur courant
     * @param NORD
     * @param SUD
     * @param Jcourant
     */
    private static void afficherPlateau(Joueur NORD, Joueur SUD, Joueur
Jcourant) {

        System.out.println("NORD ^[" + (String.format("%02d",
NORD.getPileAsc()) + "] v["
                + (String.format("%02d", NORD.getPileDesc())
                +"] (m" + NORD.getCarteEnMain_size())
                + "p" + NORD.getPioche_size() + ")"));
        System.out.println("SUD  ^[" + (String.format("%02d",
SUD.getPileAsc()) + "] v["
                + (String.format("%02d", SUD.getPileDesc())
                + "] (m" + SUD.getCarteEnMain_size())
                + "p" + SUD.getPioche_size() + ")"));

        if(Jcourant==NORD) {
            System.out.print("cartes NORD { ");
            for(int carte : Jcourant.getCarteEnMain())
                System.out.print(String.format("%02d",carte) + " ");
            System.out.println("}");
            //System.out.println("cartes NORD
{ "+Jcourant.CarteEnMain_toString().replace("[", "").replace("]",
"").replace(",", "")+ " }");
        }
        if(Jcourant==SUD) {
            System.out.print("cartes SUD { ");
            for(int carte : Jcourant.getCarteEnMain())
                System.out.print(String.format("%02d",carte) + " ");
            System.out.println("}");
            //System.out.println("cartes SUD
{ "+Jcourant.CarteEnMain_toString().replace("[", "").replace("]",
"").replace(",", "")+ " }");
        }
    }
    /**
     * affiche le gagnant
     * @param NORD
```

```
     * @param SUD
     * @param Jcourant
     * @param Jadverse
     */
    private static void afficherGagnant(Joueur NORD, Joueur SUD, Joueur
Jcourant, Joueur Jadverse) {
        String nomCourant,nomAdverse;
        if(Jcourant==NORD) {
            nomCourant="NORD";
            nomAdverse="SUD";
        }
        else {
            nomCourant="SUD";
            nomAdverse="NORD";
        }

        if(partieGagnee(Jcourant)) {
            afficherPlateau(NORD,SUD,Jcourant);
            System.out.println("partie finie, " + nomCourant + " a gagné");

        }
        else if(partiePerdue(Jcourant,Jadverse)) {
            afficherPlateau(NORD,SUD,Jcourant);
            System.out.println("partie finie, " + nomAdverse + " a gagné");
        }
    }


    /**
     * affiche en fin de tour, le nombre de cartes posées et le nombre de
cartes piochées du joueur courant
     * @param Jcourant
     * @param Jadverse
     * @param saisieAvecPile
     * @param saisieSansPile
     */
    private static void afficherFinTour(Joueur Jcourant, Joueur Jadverse,
String[] saisieAvecPile, int[] saisieSansPile) {
        int cartesPiochees=Partie.piocherFinTour(Jcourant, saisieAvecPile);
        int cartesPosees=Partie.poserCarte(Jcourant, Jadverse,
saisieAvecPile, saisieSansPile);

        System.out.println(cartesPosees + " cartes posées,
"+cartesPiochees+" cartes piochées");
    }

    // VERIFICATION DE FIN DE PARTIE ------------------------------------
------------------------------------------------------------
    /**
     * vérifie si la partie est finie
     * @param Jcourant
     * @param Jadverse
     * @return vrai ou faux
     */
    private static boolean finPartie(Joueur Jcourant, Joueur Jadverse) {
        if(partieGagnee(Jcourant)||partiePerdue(Jcourant,Jadverse))
            return true;
        else return false;
    }
```

```
    /**
     * le joueur courant a gagné s'il n'a plus de carte dans sa pioche et
sa main
     * @param Jcourant
     * @return vrai ou faux en fonction du joueur courant
     */
    private static boolean partieGagnee(Joueur Jcourant){
        if(Jcourant.piocheIsEmpty() && Jcourant.getCarteEnMain().isEmpty())
{
            return true;
        }

        return false;
    }
    /**
     * le joueur courant a perdu s'il ne peux pas poser 2 cartes ou plus
     * @param Jcourant
     * @param Jadverse
     * @return vrai ou faux
     */
    private static boolean partiePerdue(Joueur Jcourant,Joueur Jadverse) {
         int cpt=0;
         if(Jcourant.getCarteEnMain_size()<2)
             return true;
         else {
            Joueur JtmpCourant=new Joueur(Jcourant);//joueur temporaire
pour poser des cartes temporairement
            Joueur JtmpAdverse=new Joueur(Jadverse);


            for(int carte=0; carte<Jcourant.getCarteEnMain_size();carte++)
{
                if(cpt>=2)
                    return false;

if(JtmpCourant.getCarteEnMain().get(carte)>JtmpCourant.getPileAsc() ||
JtmpCourant.getCarteEnMain().get(carte)==(JtmpCourant.getPileAsc()-10)) {

JtmpCourant.setPileAsc(JtmpCourant.getCarteEnMain().get(carte));
                    cpt++;
                    continue;
                }


if(JtmpCourant.getCarteEnMain().get(carte)<JtmpCourant.getPileDesc() ||
JtmpCourant.getCarteEnMain().get(carte)==(JtmpCourant.getPileDesc()+10)) {

JtmpCourant.setPileDesc(JtmpCourant.getCarteEnMain().get(carte));
                    cpt++;
                    continue;
                }

if(JtmpCourant.getCarteEnMain().get(carte)<JtmpAdverse.getPileAsc() ) {

JtmpCourant.setPileDesc(JtmpCourant.getCarteEnMain().get(carte));
                    cpt++;
                    continue;
                }

if(JtmpCourant.getCarteEnMain().get(carte)>JtmpAdverse.getPileDesc()) {
```

```
JtmpCourant.setPileDesc(JtmpCourant.getCarteEnMain().get(carte));
                    cpt++;
                    continue;
                }
            }
        }
        if(cpt<2)
            return true;
        return false;
     }



//*************************************************************************
*****************************************
    public static void main(String[] args){


        Joueur NORD = new Joueur();
        Joueur SUD = new Joueur();

        Joueur Jcourant=NORD;
        Joueur Jadverse=SUD;

        String[] saisieAvecPile=new String[Joueur.MAX_MAIN];
        int[] saisieSansPile=new int[Joueur.MAX_MAIN];
        //reflechir a une autre solution car pb inverser joueur et verif
fin partie
        //joueur courant = nord. si Nord gagne puis inverser joueur, sud
continue a jouer or le jeu est censé etre deja fini

        do {
        afficherPlateau(NORD,SUD,Jcourant);
        saisieAvecPile=Partie.saisie(Jcourant,Jadverse);

        saisieSansPile=Partie.saisieSansPile(saisieAvecPile);

        Partie.poserCarte(Jcourant,Jadverse,saisieAvecPile,saisieSansPile);
        afficherFinTour(Jcourant,Jadverse,saisieAvecPile,saisieSansPile);

        if(Jcourant==NORD) {
            Jcourant=SUD;
            Jadverse=NORD;
        }
        else if(Jcourant==SUD) {
            Jcourant=NORD;
            Jadverse=SUD;
        }
        }
        while(!finPartie(Jcourant,Jadverse));

        afficherGagnant(NORD,SUD,Jcourant,Jadverse);
    }
}
```