Rapport de projet DOO : 6 qui prend

Nils Mittelhockamp et Louis Masson

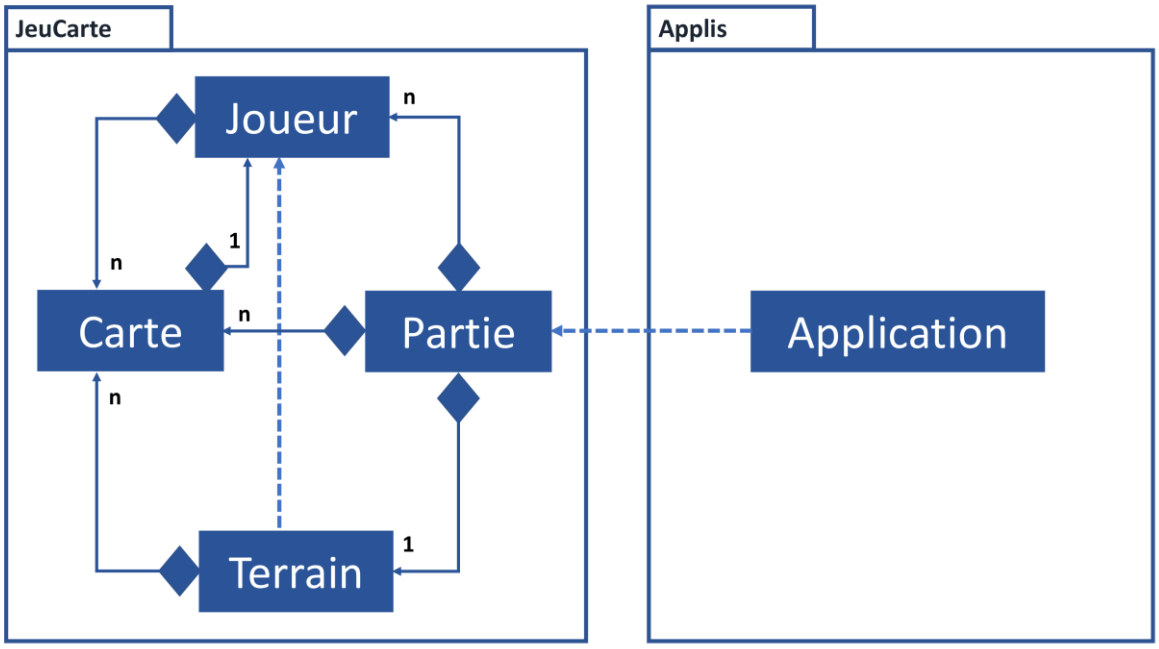
Tables des matières

I. Cahier des charges

Notre mission lors de ce projet était de produire un programme permettant à des joueurs de jouer une partie de ***6 qui prend***. Nous devions concevoir ce programme en Java 1.8 avec les notions vues lors de cette période. Plusieurs contraintes nous étaient imposées pour la réalisation du jeu :

- Les joueurs doivent jouer à travers une console
- Le programme doit effectuer des pauses avant chaque saisie d'un coup pour s'assurer que le joueur est bien présent. L'écran doit être effacer après chaque coup pour éviter la triche.
- Le programme doit faire attention aux saisies des joueurs et leurs signaler leurs erreurs
- Les différents joueurs sont déclarés dans un fichier de configuration
- Les sorties du programme doivent respectés une syntaxe stricte
- Le programme doit respecter les règles du 6 qui prend, mis à part pour la longueur de la partie qui est limité à 10 tours
- Le jeu de caractères doit être en `ISO-8859-1`

II. Diagramme UML



III. Code des différentes classes

Carte

```
package JeuCarte;

/**
 * La classe Carte permet de creer des cartes correspondant au 6 qui
prend
 * Elle contient le numéro d'une carte
 * son nombre de tetes de boeufs
 * et la reférence du joueur qui l'a possède
 * @author Louis Masson et Nils Mittelhockamp
 */
public class Carte {
     private int numero=0;
    private int teteBoeufs=0;
    private Joueur j;

    /**
     * Construit une carte à partir d'un numéro
     * @param numero Le numéro de la carte
     */
    public Carte(int numero) {
     //Verifie que le numéro passé est bien valide
     assert(numero>0 && numero<Terrain.getNBCARTES());
        this.numero=numero;
        //Initialise le nombre de têtes de boeufs de la carte
```

```
        this.initialiserTeteBoeufs();
        j=null;
    }

    /***
     * Renvoi la réference du joueur d'une carte
     * @return j Le joueur possédent la carte
     */
    public Joueur getJoueur() {
          return this.j;
      }

    /***
     * Attribut un joueur à une carte
     * @param j Le joueur de la carte
     */
    public void setJoueur(Joueur j) {
          this.j=j;
      }

    /***
     * Renvoi les têtes de boeufs d'une carte
     * @return teteBoeufs Le nombre de têtes de boeufs d'une carte
     */
    public int getTetesBoeufs() {
          return teteBoeufs;
      }

    /***
     * Initialise le nombre de têtes de boeufs d'une carte
     */
    private void initialiserTeteBoeufs()
    {
     //Initialise à l'aide du numéro de la carte
     int b;
           if (numero%5==0 && numero%10!=0) { //Carte se terminant
par un 5
                if (numero==55) //Si le numéro est égal à 55 ?
                     b=7;
            else
                b=2; }
         else if(numero%10==0) //Si la carte se terminant par un 0
           b=3;
         else if(numero%11==0) //Si la carte possède deux chiffres
égaux
           b=5;
         else
           b=1;
```

```
            this.teteBoeufs=b;

    }

    /***
     * Renvoi le numéro d'une carte
     * @return numero Le numéro de la carte
     */
    public int getNumero() {
          return numero;
     }
}
```

Joueur

```
package JeuCarte;

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

/**
 * La classe Joueur permet de créer un joueur
 * Elle contient son nom,
 * son nombre de tetes de boeufs,
 * le nombre maximum de cartes par joueur
 * et une ArrayList de Carte representant sa main
 * @author Louis Masson et Nils Mittelhockamp
 */

public class Joueur {
     private String nomJoueur;
     private int nbTetesBoeufs;
     private static final int NBCARTESJOUEUR=10;
     //ArrayList de cartes representant la main d'un joueur
     private ArrayList<Carte> main = new ArrayList<Carte>();

     /***
      * Constructeur d'un joueur, on initialise son nom à null et
ses points à 0
      */
     public Joueur(String nomJoueur)
     {
          this.nomJoueur=nomJoueur;
          nbTetesBoeufs=0;
     }

     /***
      * Permet d'obtenir le nom d'un Joueur
```

```java
	 * @return nomJoueur le nom du joueur en question sous forme de String
	 */
	public String getNomJoueur() {
		return nomJoueur;
	}

	/***
	 * Permet d'obtenir le nombre de têtes de boeufs d'un joueur
	 * @return le nombre de têtes de boeufs d'un joueur
	 */
    public int getTetesBoeufs() {
        return this.nbTetesBoeufs;
    }

    /***
     * Permet d'obtenir le nombre maximum de cartes qu'un joueur peut avoir en main
     * @return La constante correspondante
     */
	public static int getNbcartes() {
		return NBCARTESJOUEUR;
	}

	/***
	 * Ajoute des têtes de boeufs à un joueur
	 * @param p Le nombre de têtes de boeufs à ajouter
	 */
	public void ajoutTetesBoeuf(int p) {
		nbTetesBoeufs+=p;
	}

	/***
	 * Permet d'initialiser la main d'un Joueur à partir d'un terrain
	 * @param t Le terrain pour lequel il faut piocher dans le paquet (deck)
	 */
	public void initialiserMain(Terrain t)
	{
		for (int i=0 ; i<NBCARTESJOUEUR;i++)
		{
			main.add(t.piocher()); //On prend la carte du dessus du deck, on l'attribue au joueur
                                   //et on supprime la carte du deck
		}

		trierMain(); //On trie la main du joueur
```

```
	}

	/***
	 * Trie la main d'un joueur
	 */
	private void trierMain()
	{
		Carte tmp;
		for (int j=0;j<main.size();++j)
		{
			for(int i=1;i<main.size()-j;++i)
			{
				if(main.get(i).getNumero()<main.get(i-
1).getNumero()) //Si une carte supérieur possède un indice inferieur
				{
					tmp=main.get(i);
					main.set(i, main.get(i-1));
					main.set(i-1, tmp);
				}
			}
		}

	}

	/***
	 * Supprime une carte donnée dans la main d'un joueur
	 * @param c La carte à supprimer
	 */
	private void supprimerCarte(Carte c)
	{
		for (int i=0;i<main.size();++i) //Parcourt la main
		{
			if (main.get(i).getNumero()==c.getNumero())
				main.remove(i);
		}
	}

	/***
	 * Vérifie si un joueur a une carte donnée dans sa main
	 * @param numero Le numero de la carte à chercher
	 * @return Une valeure booléene correspondant à si le joueur a
cette carte ou non
	 */
	public boolean aCetteCarte(int numero) {
		for(Carte c : main)
			if(c.getNumero()==numero)
				return true;

		return false;
```

```
    }

    /***
     * Permet à un joueur de jouer
     * @param terrain Le terrain sur lequel jouer
     * @return c La carte jouée
     */
    public Carte joue(Terrain terrain) {

        System.out.println("A "+this.getNomJoueur()+" de jouer.");
        util.Console.pause();
        //Affiche le terrain et les cartes du joueur
        System.out.print(terrain);
        System.out.println(cartes());


        System.out.print("Saisissez votre choix : ");
        Scanner scanner = new Scanner(System.in);


      int numeroCarte=0;

      do {
         //Essai de récupérer les choix du joueur
          try {
              numeroCarte=scanner.nextInt();
              //vérifie que le jour possède bien la carte jouée
              if (!this.aCetteCarte(numeroCarte))
              {
                    System.out.print("Vous n'avez pas cette carte,
saisissez votre choix : ");
              }

          } catch(InputMismatchException e) {
              System.out.print("Vous n'avez pas cette carte,
saisissez votre choix : ");
              scanner.nextLine();
          }
      //Appelle une fonction qui vérifie que le joueur a bien
cette carte
      }while(!this.aCetteCarte(numeroCarte));

      util.Console.clearScreen();

      //Supprime la carte jouée par le joueur
         Carte c = new Carte(numeroCarte);
         this.supprimerCarte(c);

         return c;
```

```
    }

    /***
     * Renvoi une chaine de caractères correspondant à l'affichage
attendu pour les cartes d'un joueur
     * @return s La chaine de caractères
     */
    public String cartes()
    {
        StringBuilder s = new StringBuilder();
        s.append("- Vos cartes : ");
        for (int i=0; i<main.size(); ++i)
        {
            s.append(main.get(i).getNumero());
            if (main.get(i).getTetesBoeufs()>1)
                s.append(" ("+main.get(i).getTetesBoeufs()+")");
            if (i<main.size()-1)
                s.append(", ");
        }

        return s.toString();
    }

    /***
     * Renvoi la main d'un joueur, est utilisé seulement pour la
classe de tests
     * @return main La main du joueur
     */
    public ArrayList<Carte> getMain()
    {
        return main;
    }
}
```

Terrain

```
package JeuCarte;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * La classe Terrain permet de creer un terrain de jeu correspondant
au 6 qui prend
 * Elle contient le nombre de cartes par partie
 * le nombre de series
```

```java
 * le nombre de cartes maximum par serie
 * une ArrayList de cartes correspondant au deck de la partie
 * un tableau d'ArrayList de Carte representant les différentes
series
 * et une HashMap utilisant les joueurs comme clés comportant les
têtes de boeufs accumulées durant le dernier tour
 * @author Louis Masson et Nils Mittelhockamp
 */
public class Terrain {

	private static final int NBCARTES=105;
	private static final int NBSERIES=4;
	private static final int NBCARTESERIES=5;
	private ArrayList<Carte> deck= new ArrayList<Carte>();
	private ArrayList<Carte>[] series = new ArrayList[NBSERIES];
	private HashMap<Joueur, Integer> pointsDuTour = new
HashMap<Joueur, Integer>();

	/***
	 * Constructeur d'un terrain,
	 * on crée le deck et on initialise chaque serie avant de les
trier
	 */
	public Terrain() {
		creationDeck();
		for (int i=0; i<NBSERIES;++i)
		{
			series[i]=new ArrayList<Carte>();
			series[i].add(deck.get(deck.size() -1));
			deck.remove(deck.size() -1 );
		}
		trierSeries();
	}

	/***
	 * Initialise un deck
	 */
	private void creationDeck() {
		//Le deck doit etre vide
		assert(deck.size()==0);
		//Ajoute les cartes du jeu au deck
		for(int i=1; i<NBCARTES;i++) {
			deck.add(new Carte(i));
		}
		//Melange le deck
		Collections.shuffle(deck);
	}

	/***
```

```
	 * Permet de piocher une carte du deck
	 * @return La carte piochée
	 */
	public Carte piocher() {
		//Le deck doit contenir des cartes
		assert(deck.size()>0);
		Carte c=deck.get(deck.size()-1);
		deck.remove(deck.size()-1); //supprime la carte piochée du
deck
		return c;
	}

	/***
	 * Verifie si une carte donnée peut être ajoutée sur les series
du terrain
	 * @param c La carte à ajouter
	 * @return Une valeure booléenne qui indique si la carte peut
etre ajoutée
	 */
	private boolean carteEmpilable(Carte c) {

		for(int i=0; i<NBSERIES;++i)
		{
			//verifie pour chaque serie si la carte peut etre
empilée
			if (series[i].get(series[i].size()-
1).getNumero()<c.getNumero())
				return true;
		}

		return false;
	}

	/***
	 * Permet à un joueur de choisir une serie à ramasser
	 * @return tmp La serie à ramasser
	 */
	private int choisirSerie() {
		Scanner scan = new Scanner(System.in);
		int tmp=0;
		System.out.print("Saisissez votre choix : ");
		do {
			//Essai de récuperer l'entrée du joueur
			try {
				tmp=scan.nextInt();
				if(tmp>NBSERIES || tmp<1)
					System.out.print("Ce n'est pas une série
valide, saisissez votre choix : ");
```

```
                    //En cas d'erreur
                } catch(InputMismatchException e) {
                    System.out.print("Ce n'est pas une série
valide, saisissez votre choix : ");
                    scan.nextLine();
                }
            }while(tmp>NBSERIES || tmp<1);

            return tmp;
      }

      /***
       * Trie les series entre elles par rapport à leur carte la plus
grande
       *
       */
      private void trierSeries()
      {

            Carte tmp;
            //Double boucle permettant de trier les series par rapport
à leur carte la plus grande

            for (int j=0;j<NBSERIES;++j)
            {
                  for(int i=1;i<NBSERIES-j;++i)
                  {
                        if(series[i].get(0).getNumero()<series[i-
1].get(0).getNumero())
                        {
                              tmp=series[i].get(0);
                              series[i].set(0, series[i-1].get(0));
                              series[i-1].set(0, tmp);

                        }
                  }
            }

      }

      /***
       * Ajoute les cartes jouées par les joueurs au terrain
       * @param cartesJouees Une ArrayList de cartes comportant les
cartes jouées
       */
      public void ajouterCarteTerrain(ArrayList<Carte> cartesJouees)
{
```

```
            //Tableau comportant les différences entre une carte et
chaque série
            int[] differencesSeries = new int[NBSERIES];
            int teteBoeufs=0;
            int indiceSerie;

            for(int i=0;i<cartesJouees.size();i++) {
                  teteBoeufs=0; //On réinitialise les têtes de boeufs
à 0
                  //Verifie que la carte est ajoutable au terrain
                  if (!carteEmpilable(cartesJouees.get(i)))
                  {

      System.out.println(cartesVontEtrePosees(cartesJouees));

                        //Si la carte n'est pas jouable, on demande au
joueur de la carte de ramasser une série
                        System.out.println("Pour poser la carte
"+cartesJouees.get(i).getNumero()+", "+

      cartesJouees.get(i).getJoueur().getNomJoueur()+" doit choisir
la série qu'il va ramasser.");
                        System.out.print(toString());
                        //Apppelle la fonction pour choisir une serie
                        int numeroSerie = choisirSerie();

                        //On compte les tetes de boeufs de la serie
                        for(int k=0;k<series[numeroSerie-
1].size();k++){
                              teteBoeufs+= series[numeroSerie-
1].get(k).getTetesBoeufs();
                        }
                        //On ajoute les tetes de boeufs au joueur

      cartesJouees.get(i).getJoueur().ajoutTetesBoeuf(teteBoeufs);

      pointsDuTour.put(cartesJouees.get(i).getJoueur(), teteBoeufs);

      //cartesJouees.get(i).getJoueur().afficherRamasserSerie(teteBoe
ufs);

                        //On reinitialise la serie
                        series[numeroSerie-1].clear();
                        series[numeroSerie-1].add(cartesJouees.get(i));
                  }

                  else
                  {
                        for (int j=0; j<NBSERIES;++j)
```

```
                        {
                              //On calcule la différence entre la carte
et les différentes series
                              differencesSeries[j]=
cartesJouees.get(i).getNumero() - series[j].get(series[j].size()-
1).getNumero();
                        }

                        //On récupere l'indice de la serie la plus
proche de la carte

      indiceSerie=serieLaPlusProche(differencesSeries);

                        //On vérifie si la serie la lpus proche est
déjà pleine
                        if(series[indiceSerie].size()==NBCARTESERIES) {

                              //Si oui, on calcule les têtes de boeufs
de la série
                              for(int l=0;l<NBCARTESERIES;++l)
                              {
                                    teteBoeufs+=
series[indiceSerie].get(l).getTetesBoeufs();
                              }
                              //On lui ajoute ces têtes de boeufs et on
réinitialise la série

      cartesJouees.get(i).getJoueur().ajoutTetesBoeuf(teteBoeufs);

      pointsDuTour.put(cartesJouees.get(i).getJoueur(), teteBoeufs);

      //cartesJouees.get(i).getJoueur().afficherRamasserSerie(teteBoe
ufs);
                              series[indiceSerie].clear();

                        }

                        //On ajoute la carte jouée
                        series[indiceSerie].add(cartesJouees.get(i));
                  }
            }
      }

      /***
       * Renvoi une chaine de caractère correpsondant à l'affichage
des cartes allant être jouées
       * @param cartesJouees Les cartes jouées durant le tour
       * @return s les cartes jouées sous forme de chaine de
caractères
```

```
	 */
	private String cartesVontEtrePosees(ArrayList<Carte>
cartesJouees) {
		StringBuilder s = new StringBuilder();
		s.append("Les cartes ");
		for (int i=0; i<cartesJouees.size(); ++i){
			//Affichage spécifique pour la dernière carte
			if (i==cartesJouees.size()-1)
				s.append(cartesJouees.get(cartesJouees.size()-
1).getNumero()+" ("+cartesJouees.get(cartesJouees.size()-
1).getJoueur().getNomJoueur()+") vont être posées.");
			else
			{
				s.append(cartesJouees.get(i).getNumero()+"
("+cartesJouees.get(i).getJoueur().getNomJoueur()+")");
				if (i==cartesJouees.size()-2)
					s.append(" et ");
				else
					s.append(", ");
			}
		}

		return s.toString();
	}

	/***
	 * Calcule la série la plus proche d'une carte
	 * @param differencesSeries Un tableau comportant la différence
entre la carte et chaque série
	 * @return indiceMinimum L'indice de la serie la plus proche de
cette carte est jouable
	 * (carte la plus grande de la serie inferieure à la carte
jouée)
	 */
	private int serieLaPlusProche(int[] differencesSeries){

		//Cette différence sera forcement mise à jour étant donné
que la plus grande carte du jeu est 105
		int differenceMinimum=NBCARTES+1;
		int indiceMinimum=0;
		for (int i=0; i<NBSERIES;++i)
		{
			if (differencesSeries[i]>0 &&
differencesSeries[i]<differenceMinimum)
			{
				indiceMinimum=i;
				differenceMinimum=differencesSeries[i];
			}
		}
```

```
            return indiceMinimum;
        }

        /***
         * Renvoi l'affichage demandée des cartes d'une serie
         * @param serie La serie à afficher
         * @return s Un String contenant les données à afficher
         */
        private String cartesSerie(ArrayList<Carte> serie) {
           StringBuilder s = new StringBuilder();
           for(int i=0; i<serie.size();++i)
           {
               s.append(serie.get(i).getNumero());
               if(serie.get(i).getTetesBoeufs()>1)
                   s.append(" ("+ serie.get(i).getTetesBoeufs()+")");

               if(i<serie.size()-1)
                   s.append(", ");

           }
           return s.toString();
       }

        /***
         * Renvoi l'affichage demandée d'un terrain
         * @return s Un String contenant les données à afficher
         */
        @Override
        public String toString() {

            StringBuilder s = new StringBuilder();
            for(int i=1;i<=NBSERIES;i++) {
                 s.append("- série n° "+i+" : ");
                 s.append(cartesSerie(series[i-
1])+System.getProperty("line.separator"));
            }
            return s.toString();
        }

        /***
         * Renvoi le nombre de cartes dans le jeu
         * @return NBCARTES Le nombre de cartes maximum
         */
        public static int getNBCARTES() {
            return NBCARTES;
        }

        /***
```

```
	 * Renvoi le deck du terrain, seulement utilisé pour la classe
de tests
	 * @return deck L'ArrayList comportant les cartes du deck
	 */
	public ArrayList<Carte> getDeck() {

		return deck;
	}

	/***
	 * Renvoi la HashMap comportant les têtes de boeufs accumulés
durant le dernier tour pour chaque joueur en ayant ramassé
	 * @return pointsDuTour La Hashmap
	 */
	public HashMap<Joueur, Integer> getPointsDuTour() {
		return pointsDuTour;
	}

}
```

Partie

```
package JeuCarte;

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

/**
 * La classe Partie permet de creer une partie
 * Elle contient un terrain,
 * une ArrayList de joueurs,
 * une ArrayList de cartes representant les cartes jouées à chaque
tour,
 * le nombre de joueurs
 * @author Louis Masson et Nils Mittelhockamp
 */
public class Partie {
	//le terrain de la partie
	private Terrain terrain = new Terrain();
	//les joueurs de la partie
	private ArrayList<Joueur> joueurs= new ArrayList<Joueur>();
	//les cartes jouees à chaque tour
	private ArrayList<Carte> cartesJouees = new ArrayList<Carte>();
	int nbJoueurs;

	/***
	 * Constructeur de partie
```

```
	 */
	public Partie() {
		//Recupere les differents joueurs depuis le fichier de
configuration
		this.recuperationJoueurs();
		for(Joueur j : joueurs)
			//Initialise la main de chaque joueur à partir d'un
terrain
			j.initialiserMain(this.terrain);
	}

	/***
	 * Recupere les differents joueurs depuis le fichier de
configuration
	 * et les sauvegardes dans l'ArrayList de joueurs
	 */
	public void recuperationJoueurs()
		{
		    FileReader fichierConfig;

		    //essai d'ouvrir le fichier de configuartion
		     try {
		          fichierConfig = new FileReader("config.txt");
		          Scanner flux = new Scanner(fichierConfig);
		         while (flux.hasNext())
		         {
		          //crée et ajoute un nouveau joueur
		          joueurs.add(new Joueur(flux.nextLine()));
		          ++nbJoueurs;
		         }
		         flux.close();

		     //si l'ouverture du fichier echoue
		     } catch (FileNotFoundException e) {
		          e.printStackTrace();
		     }
		}

	/***
	 * Permet de jouer un tour
	 */
	public void prochainTour() {
		terrain.getPointsDuTour().clear(); //On supprime les
points accumulés au tour précedent
		cartesJouees.clear(); //On vide l'ArrayList des cartes
jouées

		for(Joueur j : joueurs)
		{
```

```
                //Chaque joueur joue une carte et l'ajoute aux
cartes jouees
                cartesJouees.add(j.joue(terrain));
                cartesJouees.get(cartesJouees.size()-
1).setJoueur(j);
          }
          //On trie les cartes jouées et on les ajoutent au terrain
          trierCartesJouees();
          terrain.ajouterCarteTerrain(cartesJouees);

     }

     /***
      * Renvoi une chaine de caractère correpsondant à l'affichage
des cartes jouées
      * @return s les cartes jouées sous forme de chaine de
caractères
      */
     public String cartesJouees() {
          StringBuilder s = new StringBuilder();
          s.append("Les cartes ");
          for (int i=0; i<cartesJouees.size(); ++i){
                //Affichage spécifique pour la dernière carte
                if (i==cartesJouees.size()-1)
                     s.append(cartesJouees.get(cartesJouees.size()-
1).getNumero()+" ("+cartesJouees.get(cartesJouees.size()-
1).getJoueur().getNomJoueur()+") ont été
posées."+System.getProperty("line.separator"));
                else
                {
                     s.append(cartesJouees.get(i).getNumero()+"
("+cartesJouees.get(i).getJoueur().getNomJoueur()+")");
                     if (i==cartesJouees.size()-2)
                          s.append(" et ");
                     else
                          s.append(", ");
                }
          }

          return s.toString();
     }

     /***
      * Renvoi une chaine de caractère correpsondant à l'affichage
des joueurs de la partie
      * @return s les joueurs de la partie sous forme de chaine de
caractères
      */
     public String joueursParties() {
```

```
            StringBuilder s = new StringBuilder();
            s.append("Les "+nbJoueurs+" joueurs sont ");
            for (int i=0; i<nbJoueurs;++i)
            {     if(i<nbJoueurs-2)
                        s.append(joueurs.get(i).getNomJoueur()+", ");
                  else if(i==nbJoueurs-2)
                        s.append(joueurs.get(i).getNomJoueur()+" ");

                  else
                        s.append("et "+joueurs.get(i).getNomJoueur()+".
");
            }
            s.append("Merci de jouer à 6 qui prend !");
            return s.toString();
      }

      /***
       * Renvoi une chaine de caractère correpsondant à l'affichage
des scores de la partie
       * @return s les scores final sous forme de chaine de
caractères
       */
      public String scores() {
            trierScores();
            StringBuilder s = new StringBuilder();
            s.append("** Score
final"+System.getProperty("line.separator"));
            for (Joueur j: joueurs)
            {
                  s.append(j.getNomJoueur()+" a ramassé
"+j.getTetesBoeufs()+ " tête de
boeufs"+System.getProperty("line.separator"));
            }

            return s.toString();

      }

      /***
       * Trie les scores de la partie
       */
      private void trierScores() {
            Joueur j;
            //Double boucle pour parcourir l'ArrayList de joueurs et
comparer ses valeurs
            for (int i=0;i<joueurs.size();++i)
            {
                  for(int k=1;k<joueurs.size()-i;++k)
                  {
```

```
	if(joueurs.get(k).getTetesBoeufs()<joueurs.get(k-
1).getTetesBoeufs())
                        {
                              j=joueurs.get(k);
                              joueurs.set(k, joueurs.get(k-1));
                              joueurs.set(k-1, j);
                        }

                        //En cas d'égalité des têtes de boeufs, trie
par ordre alphabétique
                        else if
(joueurs.get(k).getTetesBoeufs()==joueurs.get(k-1).getTetesBoeufs()
                                    &&
((joueurs.get(k).getNomJoueur().compareTo(joueurs.get(k-
1).getNomJoueur())<0)))
                        {
                              j=joueurs.get(k);
                              joueurs.set(k, joueurs.get(k-1));
                              joueurs.set(k-1, j);
                        }
                  }
            }

	}

	/***
	 * Trie les cartes jouées
	 */
	private void trierCartesJouees()
	{
            Carte tmp;
            //Double boucle pour parcourir l'ArrayList de cartes et
comparer les cartes jouees
            for (int j=0;j<cartesJouees.size();++j)
            {
                  for(int i=1;i<cartesJouees.size()-j;++i)
                  {
                        //Si une carte superieur possède un indice
inferieur

	if(cartesJouees.get(i).getNumero()<cartesJouees.get(i-
1).getNumero())
                        {
                              tmp=cartesJouees.get(i);
                              cartesJouees.set(i, cartesJouees.get(i-
1));
                              cartesJouees.set(i-1, tmp);
```

```
                    }
                }
            }
        }

        /***
         * Renvoi le terrain
         * @return terrain le terrain de la partie
         */
        public Terrain getTerrain() {
            return terrain;
        }

        /***
         * Renvoi le nombre de joueurs, seulement utilsié dans les
tests
         * @return nbJoueurs Le nombre de joueurs de la partie
         */
        public int getNbJoueurs() {
            return nbJoueurs;
        }

        /***
         * Renvoi un String qui correspond à l'affichage attendu pour
afficher les points ramassés durant un tour
         * @return s Le String correspondant
         */
        public String afficherPointsRamasses() {
            StringBuilder s = new StringBuilder();

            //Si aucun joueur n'a ramassé de têtes de boeufs
            if(terrain.getPointsDuTour().size()==0)
                s.append("Aucun joueur ne ramasse de tête de
boeufs."+System.getProperty("line.separator"));
            else
            {
                //pour chaque joueur present dans la HashMap
               for(Joueur j: terrain.getPointsDuTour().keySet()){
                s.append(j.getNomJoueur()+" a ramassé
"+terrain.getPointsDuTour().get(j)+" tête de
boeufs"+System.getProperty("line.separator"));
               }
            }

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

Application

```
package Applis;

import JeuCarte.Partie;
import JeuCarte.Joueur;

/**
* Programme permettant de jouer à une partie de 6 qui prend
* @author Louis Masson et Nils Mittelhockamp
*/

public class Application {

	/***
	 * Le programme principal permettant de jouer une partie de 6
qui prend
	 * @param args Arguments de la fonction principal
	 */
	public static void main(String[] args) {
		//Créer une nouvelle partie
		Partie partie = new Partie();
		//Affiche les joueurs de la partie
		System.out.println(partie.joueursParties());

		for(int i=0; i<Joueur.getNbcartes();++i)
		{
			//Effectue un tour
			partie.prochainTour();
			//Affiche les cartes jouées durant ce tour
			System.out.print(partie.cartesJouees());
			//Affiche le terrain de jeu
			System.out.print(partie.getTerrain().toString());
			//Affiche les points rammassés durant ce tour par
les joueurs
			System.out.print(partie.afficherPointsRamasses());
		}

		//Affiche le résultat de la partie
		System.out.print(partie.scores());
	}
}
```

IV. Tests unitaires

Test de la classe Carte

```
package Test;

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

```java
import org.junit.jupiter.api.Test;

import JeuCarte.Carte;

/***
 * Permet de tester le fonctionnement de la classe carte
 * @author Nils
 *
 */
class CarteTest {

	@Test
	void main() {
		Carte c1 = new Carte(3);
		Carte c2= new Carte(55);
		Carte c3 = new Carte(10);
		Carte c4 = new Carte(22);
		Carte c5 = new Carte(65);

		//Verifie la valeur attribuée de tetes de boeufs aux
cartes
		assertEquals(1, c1.getTetesBoeufs());
		assertEquals(7, c2.getTetesBoeufs());
		assertEquals(3, c3.getTetesBoeufs());
		assertEquals(5, c4.getTetesBoeufs());
		assertEquals(2, c5.getTetesBoeufs());

	}
   }
```

Test de la classe Joueur

```java
package Test;

import static org.junit.Assert.assertEquals;

import org.junit.jupiter.api.Test;

import JeuCarte.Joueur;
import JeuCarte.Terrain;

/***
 * Classe vérifiant le fonctionnement de la classe Joueur
 * @author Nils
 *
 */
class JoueurTest {

	@Test
```

```
	void main() {

		Joueur joueur = new Joueur("Louis");
		Terrain t = new Terrain();
		joueur.initialiserMain(t);
		//On vérifie que le joueur a bien reçu le bon nombre de
cartes

	assertEquals(joueur.getMain().size(),Joueur.getNbcartes());
		//On vérifie que les cartes ont bien été piochées du deck
		assertEquals(t.getDeck().size(), Terrain.getNBCARTES()-5-
Joueur.getNbcartes());


		/*On teste la méthode aCetteCarte
		 * la méthode doit retourner true 10 fois
		 */
		int cartes=0;
		for (int i=1; i<Terrain.getNBCARTES();++i)
		{
			if(joueur.aCetteCarte(i))
				++cartes;
		}
		//La variable carte doit avoir été incrémenté 10 fois
		assertEquals(cartes, Joueur.getNbcartes());
	}
  }
```

Test de la classe Terrain

```
package Test;

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

import org.junit.jupiter.api.Test;
import JeuCarte.Terrain;

/***
 * Permet de tester le fonctionnement de la classe terrain
 * @author Nils
 *
 */
class TerrainTest {
	@Test
	void piocher() {
		Terrain t = new Terrain();

		//On souhaite vérifier qu'aucun point ait été attribué
		assertEquals(t.getPointsDuTour().size(),0);
```

```
            for(int i=0;i<Terrain.getNBCARTES()-5;++i)
            {
                  t.piocher();
            }
            //Vérifie que le deck a bien été vidée et donc que les
cartes peuvent bien etre piochées
            assertEquals(t.getDeck().size(),0);

      }
   }
```

Test de la classe Partie

```
package Test;

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

import org.junit.jupiter.api.Test;

import JeuCarte.Partie;

/***
 * Permet de tester le fonctionnement de la classe partie
 * @author Nils
 *
 */
class PartieTest {

      private final int NBJOUEURS=4;
      @Test
      void nbJoueurs() {
            Partie p = new Partie();
            //Vérifie qu'une partie comporte bien le nombre de joueurs
donné dans le fichier de configuration
            //après création
            assertEquals(p.getNbJoueurs(),NBJOUEURS);

            //Aucun joueur doit ramasser de tetes de boeufs au début
de la partie
            assertEquals("Aucun joueur ne ramasse de tête de
boeufs."+System.getProperty("line.separator"),p.afficherPointsRamass
es());
      }
}
```

V. Difficultés rencontrées

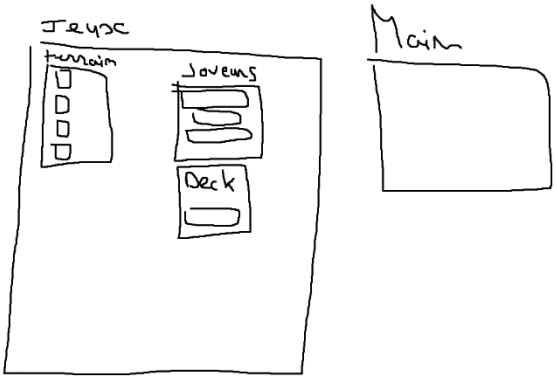
Nous avons rencontré quelques difficultés lors de la conception de notre programme.

Utilisation de la Programmation Orienté Objet

Afin de répondre correctement aux contraintes du cahier des charges, nous devions utiliser la programmation orientée objet. La question que nous nous sommes posés est comment modéliser le jeu sous forme d'objets ?

La POO étant une notion nouvelle pour nous, nous avons décidé de schématiser le jeu et les différents objets qui le composent avant de commencer à coder.

Schéma représentant les objets de notre programme



Nous avons représenté le jeu par une classe 'Partie' dans notre code. Cette classe est composé d'un objet 'Terrain' représentant le terrain de jeu, et d'objets 'Joueur' représentant les joueurs de la partie. Les classes Terrain et Joueur ont quant à elle besoin d'objets 'Carte', correspondant aux cartes du 6 qui prend. La classe 'Partie' est quant à elle utilisée dans le programme principal, permettant de faire évoluer le programme plus facilement, en ajoutant une interface graphique ou en permettant de jouer plusieurs parties simultanément par exemple et en ayant à modifier que très légèrement la classe Partie.

```
public class Partie {
	//le terrain de la partie
	private Terrain terrain = new Terrain();
	//les joueurs de la partie
	private ArrayList<Joueur> joueurs= new ArrayList<Joueur>();
	//les cartes jouees chaque tour
	private ArrayList<Carte> cartesJouees = new ArrayList<Carte>();
	int nbJoueurs;
```

Stockage des joueurs

Après avoir créé la classe Joueur, nous nous sommes demandé où nous allions stocker les joueurs de la partie. En effet, au début nous voulions stocker l'ArrayList de joueurs dans la classe Terrain, pensant que les joueurs appartiennent d'une certaine manière au terrain. Ce n'est qu'après réflexion que

nous nous sommes dit qu'il serait plus judicieux et logique de conserver les joueurs de la partie dans la classe Partie.

Gestion des erreurs

Le cahier des charges nous imposait de vérifier les entrées des joueurs et d'éviter les erreurs de saisies. Nous devions éviter qu'un joueur joue une carte qu'il n'a pas ou qu'il entre une chaine de caractères au lieu d'un entier par exemple.

Pour cela, nous avons utilisé le **try-catch** de Java permettant d'attraper l'exception levé lorsque l'entrée est invalide. En combinant cette structure de contrôle avec une boucle **while**, nous avons pu contrôler l'entrée des joueurs. Voici les instructions pour la saisie d'un coup par exemple :

```
do {
          //Essai de récupérer les choix du joueur
           try {
               numeroCarte=scanner.nextInt();
               //vérifie que le jour possède bien la carte jouée
               if (!this.aCetteCarte(numeroCarte))
               {
                    System.out.println("Vous n'avez pas cette
carte, saisissez votre choix :");
               }

            } catch InputMismatchException e) {
               System.out.println("Vous n'avez pas cette carte,
saisissez votre choix :");
               scanner.nextLine();
     //Appelle une fonction qui vérifie si le joueur a bien la carte
     joué    }
        }while(!this.aCetteCarte(numeroCarte));
```

Le problème de cette solution est que la fonction 'aCetteCarte' est appelée deux fois.

Stockage des points ramassés durant un tour

Dû à aux contraintes imposées dans le cahier des charges vis-à-vis de l'affichage, il était nécessaire de stocker les points accumulés par chaque joueur durant un tour afin de pouvoir les afficher au moment convenu. Pour cela, nous avons utilisé une HashMap dans la classe terrain qui, à un joueur, nous donne une valeur correspondant aux têtes de bœufs ramassés durant ce tour. Cette HashMap est réinitialisé au début de chaque tour.

```
private HashMap<Joueur, Integer> pointsDuTour = new HashMap<Joueur,
Integer>();
```

Cette HashMap est ensuite appelée dans une méthode de la classe partie pour afficher les points accumulés durant ce tour.

```
	public String afficherPointsRamasses() {
		StringBuilder s = new StringBuilder();

		//Si aucun joueur n'a ramassé de têtes de boeufs
		if(terrain.getPointsDuTour().size()==0)
			s.append("Aucun joueur ne ramasse de têtes de
boeufs."+System.getProperty("line.separator"));
		else
		{
			//pour chaque joueur present dans la HashMap
		   for(Joueur j: terrain.getPointsDuTour().keySet()){
			s.append(j.getNomJoueur()+" a ramassée
"+terrain.getPointsDuTour().get(j)+" têtes de
boeufs"+System.getProperty("line.separator"));
		   }
		}

		return s.toString();
	}
```

VII. Améliorations possibles

Les améliorations possibles seraient d'ajouter une interface graphique au 6 qui prend, permettant ainsi aux joueurs de s'immiscer complètement dans le jeu. Pour cela, il faudrait corriger quelques imperfections dans le code du programme. En effet, certaines méthodes dans les classes Partie et Joueur gèrent elle-même l'affichage au lieu de renvoyer une chaine de caractères pouvant être traiter. Nous n'avons pas encore réussi à remédier à ce problème car ces méthodes ont besoin des entrées/sorties pour effectuer leurs taches, par exemple permettre à un joueur de jouer et vérifier cette entrée.

```
	public Carte joue(Terrain terrain) {

		System.out.println("A "+this.getNomJoueur()+" de jouer.");
		util.Console.pause();
		//Affiche le terrain et les cartes du joueur
		System.out.print(terrain);
		System.out.println(cartes());


		System.out.println("Saisissez votre choix :");
		Scanner scanner = new Scanner(System.in);
```

Une fois ce problème corrigé, nous serions en mesure de produire une interface graphique en réutilisant le code. La manière dont nous avons conçu le programme nous permettrait dans le futur d'effectuer plusieurs parties simultanément en réutilisant notre classe Partie et en adaptant notre programme principal. Il serait alors judicieux de développer une base de données dans laquelle stocker les différents joueurs permettant de créer un classement global entre tous joueurs par exemple.

Une autre amélioration possible serait l'optimisation des tris du code. En effet, au cours du jeu nous avons souvent besoin de trier des éléments (des cartes, des scores, des séries…). Nous avons créé des méthodes respectives pour trier les éléments étant donné que les données et les structures sont différentes, mais il serait judicieux de factoriser ce code afin de créer une seule méthode/ fonction permettant de trier ces éléments de types différents, étant donné que la majorité des tris du code se font sur des entiers (le numéro d'une carte, le score d'un joueur, la carte d'une série).

```
private void trierMain()
        {
private void trierScores() {
private void trierCartesJouees()
        {
        }
```

Une autre façon de résoudre le problème serait d'utiliser des méthodes fournis par Java.