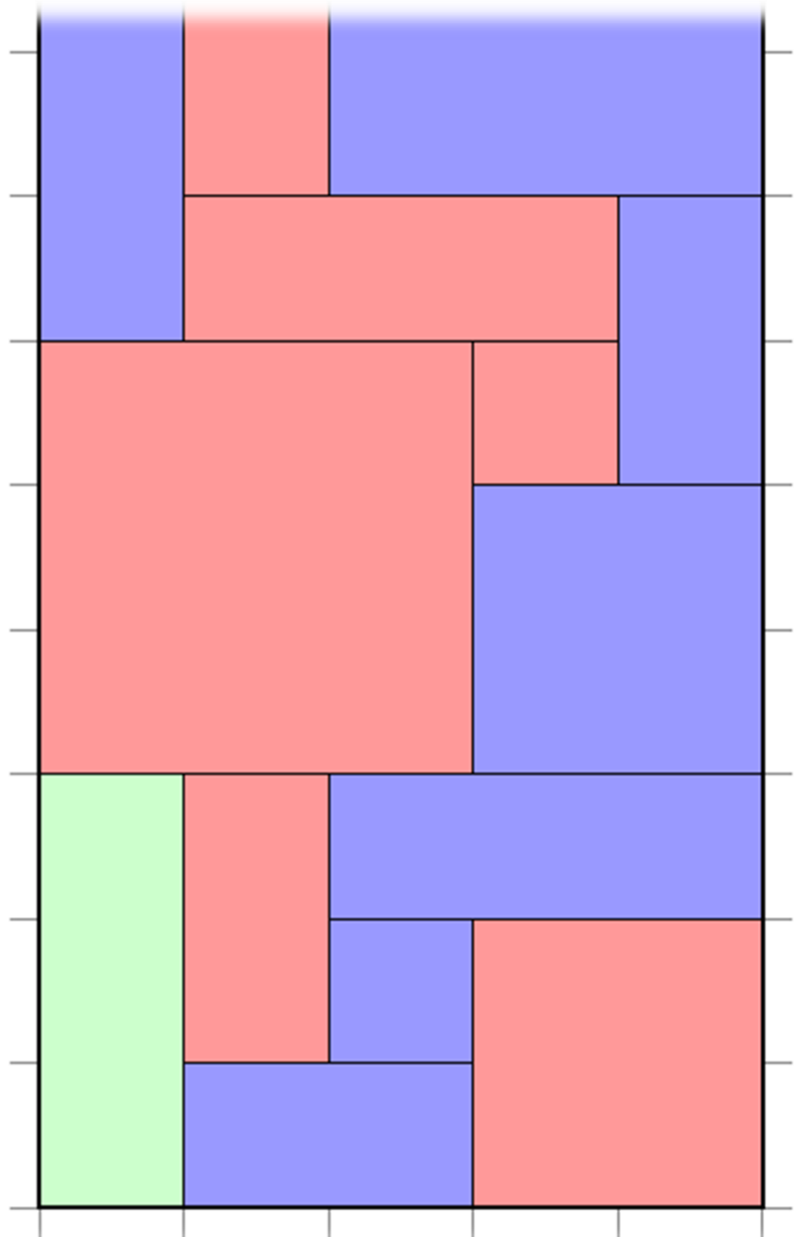


09 MARS 2020



## Rapport de projet

BPO - JAVA



Antoine Després  
Thibault Henrion  
Groupe 106 – BINÔME 1  
2019-2020

# Table des matières

Présentation du projet .....	2
Diagramme UML des classes .....	2
Code des tests unitaires des classes.....	3
<b>Classe JeuDeCartesTest .....</b>	<b>3</b>
<b>Classe ListeCarreauxTest .....</b>	<b>3</b>
<b>Classe FctJeuTest .....</b>	<b>3</b>
<b>Classe MurTest .....</b>	<b>4</b>
Code Java complet.....	5
<b>Énumération TypeCarte.....</b>	<b>5</b>
<b>Énumération TypeErreur .....</b>	<b>5</b>
<b>Classe Carte .....</b>	<b>6</b>
<b>Classe JeuDeCartes .....</b>	<b>7</b>
<b>Classe Carreau .....</b>	<b>9</b>
<b>Classe ListeCarreaux .....</b>	<b>10</b>
<b>Classe Score.....</b>	<b>15</b>
<b>Classe FctJeu .....</b>	<b>15</b>
<b>Classe Mur.....</b>	<b>18</b>
<b>Classe Main .....</b>	<b>24</b>
Bilan de projet .....	26

## Présentation du projet

Ce projet consiste à créer un jeu collaboratif adapté de « *Team Up !* » en langage de programmation Java. Dans cette version, deux joueurs coopèrent pour remplir un mur avec des carreaux rectangulaires de hauteur et largeur variable rouges ou bleus, en suivant certaines règles de disposition. Nous avons notamment dû réaliser l’affichage du mur à chaque coup, sur console.

Le fonctionnement global de notre programme est le suivant :

Une méthode nommée `appelCommande` récupère la saisie de l’utilisateur, et l’envoie à la méthode de placement de carreau si cette dernière est correcte. Cette méthode va ensuite vérifier que ce placement est possible et, le cas échéant, placer ce carreau. Dans tous les cas, le programme renvoie un type d’erreur, le type « CORRECT » signifiant que tout s’est bien passé. En fonction du type d’erreur renvoyé, un affichage sera fait pour informer les carreleurs sur la faute commise. Un système de cartes est également en place. Celles-ci donnent une consigne sur les carreaux pouvant être joués. Il est possible d’en passer une ou de demander l’arrêt de la partie, ce qui affiche le score.

La documentation complète du projet est disponible à l’adresse suivante :

<https://antoinedespres.github.io/tootygroupe1doc/>

## Diagramme UML des classes

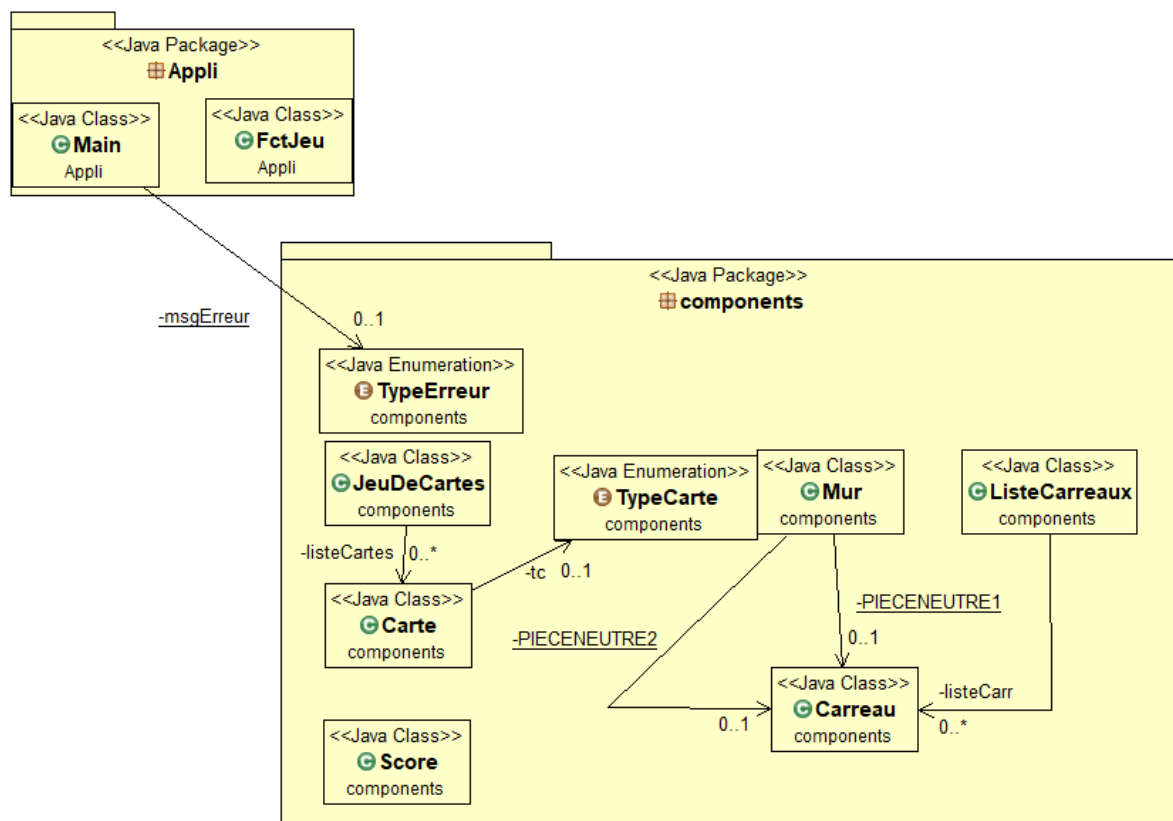


Figure 1 : Diagramme UML des classes avec packages

## Code des tests unitaires des classes

### Classe JeuDeCartesTest

```
package components;

import static org.junit.Assert.*;

import org.junit.Test;

public class JeuDeCartesTest {

    @Test
    public void test() {
        JeuDeCartes j = new JeuDeCartes();
        assertTrue(j.getJeuCartes().size()==33);
    }
}
```

### Classe ListeCarreauxTest

```
package components;

import static org.junit.Assert.*;

import org.junit.Test;

public class ListeCarreauxTest {

    @Test
    public void test() {
        ListeCarreaux p = new ListeCarreaux(true);
        Carte c = new Carte(TypeCarte.TAILLE1);
        assertTrue(p.size()==18);
        // on vérifie qu'un carreau de dimension 2x2 ne fait pas partie de
        ceux de taille 1
        assertFalse(p.carreauDispo(c).contient('d'));
    }
}
```

### Classe FctJeuTest

```
package Appli;

import static org.junit.Assert.*;

import org.junit.Test;
```

```

import components.JeuDeCartes;
import components.ListeCarreaux;

public class FctJeuTest {

    @Test
    public void test() {
        // test des conditions de fin de partie
        ListeCarreaux p = new ListeCarreaux(true); // plein
        ListeCarreaux q = new ListeCarreaux(false); // vide
        JeuDeCartes j = new JeuDeCartes();

        assertTrue(FctJeu.estTerminee(j, p, true));
        assertTrue(FctJeu.estTerminee(j, q, false));
        assertFalse(FctJeu.estTerminee(j, p, false));
    }

}

```

### Classe MurTest

```

package components;

import static org.junit.Assert.*;

import org.junit.Test;

public class MurTest {

    @Test
    public void test() {
        // test des erreurs de placement
        Mur m = new Mur();
        Carreau x = new Carreau(1,3,'x',false);
        ListeCarreaux p = new ListeCarreaux(true);
        TypeErreur t1=m.placerCarreau(p.getCarreau('c'), -1, -1);
        assertTrue(t1==TypeErreur.DEPASSEMENT);
        TypeErreur t2=m.placerCarreau(x, 1, 1);
        assertTrue(t2==TypeErreur.CORRECT);
        TypeErreur t3=m.placerCarreau(p.getCarreau('a'), 1, 4);
        assertTrue(t3==TypeErreur.CLONAGE);
        TypeErreur t4=m.placerCarreau(p.getCarreau('d'),1,4);
        assertTrue(t4==TypeErreur.REPOS_BASE);
        TypeErreur t5 = m.placerCarreau(p.getCarreau('e'), 5, 1);
        assertTrue(t5==TypeErreur.NON_CONTACT);
    }

}

```

## Code Java complet

### Énumération TypeCarte

```
package components;
/**
 * Énumération des types de carreau pouvant être indiqués sur une {@link Carte}
 */
public enum TypeCarte {
    BLEU("bleu"), ROUGE("rouge"), TAILLE1("Taille 1"), TAILLE2("Taille 2"),
    TAILLE3("Taille 3");

    /**
     * La chaîne de caractères devant être affichée
     */
    private final String affichage;

    /**
     * Constructeur du type de carte
     *
     * @param affichageConforme Ce qui doit être affiché
     */
    TypeCarte(final String affichageConforme) {
        this.affichage = affichageConforme;
    }

    @Override
    /**
     * Stocke la chaîne de caractères à afficher
     *
     * @return Le type de carreau à jouer
     */
    public String toString() {
        return this.affichage;
    }
}
```

### Énumération TypeErreur

```
package components;
/**
 * Énumération des erreurs possibles
 */
public enum TypeErreur {
    CORRECT(""),
    DEPASSEMENT("Le carreau dépasse de la zone à carreler"),
    NON_CONTACT("Le carreau ne touche pas un autre carreau"),
    NON_TROUVE("Élément non trouvé"),
    REPOS_BASE("Toute la base du carreau ne repose pas sur le bas de la zone ou d'autres carreaux"),
    CLONAGE("Le carreau clone le côté d'un carreau déjà posé"),
}
```

```

SAISIE("La saisie est incorrecte");

/**
 * La chaîne de caractères devant être affichée
 */
private final String affichage;

/**
 * Constructeur de type d'erreur
 *
 * @param affichageConforme Ce qui doit être affiché
 */
TypeErreur(final String affichageConforme) {
    this.affichage = affichageConforme;
}

@Override
/**
 * Stocke la chaîne de caractères à afficher
 *
 * @return Le type d'erreur
 */
public String toString() {
    return this.affichage + System.LineSeparator();
}
}

```

## Classe Carte

```

package components;
/**
 * Classe Carte.
 *
 * @author Antoine Després
 * @author Thibault Henrion
 * @version 1.0
 */
public class Carte {

    /**
    * Une carte est caractérisée seulement par le type de carreau qu'elle
    oblige à
    * jouer (taille ou couleur)
    */
    private TypeCarte tc;

    /**
    * Constructeur de carte
    *
    * @param tc le type de carte
    */
    public Carte(TypeCarte tc) {
        this.tc = tc;
    }

    /**
    * Accesseur de type de carte

```

```

    *
    * @return tc le type de carte
    */
    public TypeCarte getTypeCarte() {
        return this.tc;
    }
}

```

## Classe JeuDeCartes

```

package components;
import java.util.ArrayList;
import java.util.Collections;

/**
 * Classe JeuDeCartes.
 *
 * @author Antoine Després
 * @author Thibault Henrion
 * @version 1.0
 */
public class JeuDeCartes {
    public static final int NBCARTES = 33;
    public static final int NBCARTESPARCOULEUR = 9;
    public static final int NBCOULEURS = 2;
    public static final int NBCARTESPARTAILLE = 5;
    public static final int NBTAILLES = 3;

    private ArrayList<Carte> listeCartes;

    /**
     * Constructeur de jeu de cartes
     */
    public JeuDeCartes() {
        listeCartes = new ArrayList<>();

        this.remplirPaquet();

        this.mélangerPaquet();
    }

    /**
     * Remplissage du paquet de cartes
     */
    private void remplirPaquet() {
        int i = 0;
        for (; i < NBCARTESPARCOULEUR; ++i)
            this.listeCartes.add(new Carte(TypeCarte.BLEU));
        for (; i < NBCARTESPARCOULEUR * NBCOULEURS; ++i)
            this.listeCartes.add(new Carte(TypeCarte.ROUGE));
        for (; i < NBCARTESPARCOULEUR * NBCOULEURS + NBCARTESPARTAILLE; ++i)
            this.listeCartes.add(new Carte(TypeCarte.TAILLE1));
        for (; i < NBCARTESPARCOULEUR * NBCOULEURS + NBCARTESPARTAILLE +
NBCARTESPARTAILLE; ++i)
            this.listeCartes.add(new Carte(TypeCarte.TAILLE2));
    }
}

```



```

        for (; i < NBCARTES; ++i)
            this.listeCartes.add(new Carte(TypeCarte.TAILLE3));
    }

    /**
     * Tire une carte du paquet
     *
     * @return carteTirée : la carte tirée
     */
    public Carte tirerCarte() {
        Carte carteTirée = getJeuCartes().get(getJeuCartes().size() - 1);
        getJeuCartes().remove(getJeuCartes().size() - 1); // on ne la remet
        pas dans le paquet
        return carteTirée;
    }

    /**
     * Vérifie si le paquet est vide
     *
     * @return true si paquet vide, false sinon
     */
    public boolean estVide() {
        return this.getJeuCartes().isEmpty();
    }

    /**
     * Vide le paquet de cartes
     */
    public void viderPaquet() {
        this.getJeuCartes().clear();
    }

    /**
     * Mélange le paquet de cartes
     */
    public void mélangerPaquet() {
        assert (!this.estVide());
        Collections.shuffle(this.getJeuCartes());
    }

    /**
     * Ajoute une carte en haut du paquet
     *
     * @param newCarte La Carte à insérer dans le jeu de cartes
     */
    public void insererCarte(Carte newCarte) {
        getJeuCartes().add(getJeuCartes().size() - 1, newCarte);
    }

    /**
     * Accesseur du jeu de cartes
     *
     * @return listeCartes le jeu de cartes
     */
    public ArrayList<Carte> getJeuCartes() {
        return listeCartes;
    }
}

```

## Classe Carreau

```
package components;
/**
 * Classe Carreau. Un carreau est caractérisé par une largeur, une hauteur et
 * une lettre.
 *
 * @author Antoine Després
 * @author Thibault Henrion
 * @version 1.0
 */
public class Carreau {
    private int largeur;
    private int hauteur;
    private char lettre;

    /**
     * Constructeur de carreau
     *
     * @param pLargeur Largeur du carreau
     * @param pHauteur Hauteur du carreau
     * @param pLettre Lettre du carreau
     * @param pRouge Booléen définissant si le carreau doit être rouge ou non
     */
    public Carreau(int pLargeur, int pHauteur, char pLettre, boolean pRouge) {
        assert (pLargeur >= 1 && pLargeur <= 3);
        assert (pHauteur >= 1 && pHauteur <= 3);

        if (pRouge) {
            pLettre = Character.toUpperCase(pLettre);
        }
        this.largeur = pLargeur;
        this.hauteur = pHauteur;
        this.lettre = pLettre;
    }

    /**
     * Accesseur de la largeur d'un carreau
     *
     * @return largeur La largeur du carreau
     */
    public int getLargeur() {
        return this.largeur;
    }

    /**
     * Accesseur de la hauteur d'un carreau
     *
     * @return hauteur La hauteur du carreau
     */
    public int getHauteur() {
        return this.hauteur;
    }

    /**
     * Accesseur de la lettre d'un carreau
     *
     */
}
```

```

    * @return lettre La lettre du carreau
    */
    public char getLettre() {
        return this.lettre;
    }

    /**
     * Vérifie si un carreau est rouge. Pour cela, on vérifie si la lettre du
     * carreau est en majuscule.
     *
     * @return True si le carreau est rouge, false sinon
     */
    public boolean estRouge() {
        return Character.isUpperCase(this.lettre);
    }
}

```

## Classe ListeCarreaux

```

package components;
import java.util.*;

/**
 * Classe ListeCarreaux.
 *
 * @author Antoine Després
 * @author Thibault Henrion
 * @version 1.0
 */
public class ListeCarreaux {

    private ArrayList<Carreau> listeCarr;

    private final int COTESMAXCARREAU = 3;
    private final int NBCOULEURS = 2;

    private static final char Lettres[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g',
'h', 'i' };
    private static final int dim[][] = { { 1, 1, 2, 2, 1, 3, 2, 3, 3 }, { 1, 2,
1, 2, 3, 1, 3, 2, 3 } };
    private static final int dimPieceNeutre[][] = { { 1, 3 }, { 3, 1 } };

    /**
     * Constructeur de ListeCarreaux
     *
     * @param remplir Booléen permettant de choisir de remplir ou non la liste
     */
    public ListeCarreaux(boolean remplir) {
        listeCarr = new ArrayList<>();
        if (remplir) {
            this.remplirPaquet();
        }
    }

    /**

```

```

* Accesseur de taille de liste
*
* @return La taille de la liste
*/
public int size() {
    return this.listeCarr.size();
}

/**
* Retire un carreau
*
* @param lettre La lettre du carreau
* @return Un type d'erreur
*/
public TypeErreur retirerCarreau(char lettre) {
    for (int i = 0; i < this.listeCarr.size(); i++) {
        if (this.listeCarr.get(i).getLettre() == lettre) {
            this.listeCarr.remove(i);
            return TypeErreur.CORRECT;
        }
    }
    return TypeErreur.NON_TROUVE;
}

/**
* Remplissage d'une liste de carreaux
*/
private void remplirPaquet() {
    boolean rouge = false;
    for (int j = 0; j < 2; ++j) {
        int k = 0;
        for (char i : Lettres) {
            listeCarr.add(new Carreau(dim[0][k], dim[1][k], i,
rouge));
            k++;
        }
        rouge = true; // mêmes carreaux, en rouge cette fois
    }
}

/**
* Affichage des carreaux pouvant être joués avec la carte tirée
*
* @param carteTiree la carte tirée
* @return carreauxDispo Une liste de carreaux
*/
public ListeCarreaux carreauDispo(Carte carteTiree) {
    ListeCarreaux carreauDispo = new ListeCarreaux(false);
    switch (carteTiree.getTypeCarte()) {
        case BLEU:
            for (Carreau i : this.listeCarr) {
                if (Character.isLowerCase(i.getLettre()))
                    carreauDispo.listeCarr.add(i);
            }
            break;
        case ROUGE:
            for (Carreau i : this.listeCarr) {
                if (Character.isUpperCase(i.getLettre()))
                    carreauDispo.listeCarr.add(i);
            }
    }
}

```

```

        }
        break;
    case TAILLE1:
        for (Carreau i : this.listeCarr) {
            if (i.getHauteur() == 1 || i.getLargeur() == 1)
                carreauDispo.listeCarr.add(i);
        }
        break;
    case TAILLE2:
        for (Carreau i : this.listeCarr) {
            if (i.getHauteur() == 2 || i.getLargeur() == 2)
                carreauDispo.listeCarr.add(i);
        }
        break;
    case TAILLE3:
        for (Carreau i : this.listeCarr) {
            if (i.getHauteur() == 3 || i.getLargeur() == 3)
                carreauDispo.listeCarr.add(i);
        }
        break;
    }
    return carreauDispo;
}

/**
 * Accesseur de carreau
 *
 * @param lettre La lettre correspondant au carreau demandé.
 * @return Le carreau demandé.
 */
public Carreau getCarreau(char lettre) {
    for (int i = 0; i < this.listeCarr.size(); i++) {
        if (this.listeCarr.get(i).getLettre() == lettre) {
            return this.listeCarr.get(i);
        }
    }
    return (new Carreau(0, 0, 'z', true)); // carreau de bug
}

/**
 * Accesseur de la liste de carreaux
 *
 * @return La liste de carreaux
 */
public ArrayList<Carreau> getListeCarreaux() {
    return listeCarr;
}

/**
 * Accesseur de la largeur d'un carreau
 *
 * @return res la largeur du carreau demandé
 */
private int LargeurCarreaux() {
    int res = 0;
    for (Carreau i : this.listeCarr) {
        res += i.getLargeur();
    }
    return res;
}

```

```

}

/**
 * Vérifie si la liste de carreaux est vide ou non.
 *
 * @return True si la liste de carreaux est vide, false sinon.
 */
public boolean estVide() {
    return this.listeCarr.isEmpty();
}

/**
 * Vérifie si la liste de carreaux contient le carreau demandé
 *
 * @param a La lettre correspondant au carreau demandé.
 * @return True si le carreau est dans la liste, false sinon.
 */
public boolean contient(char a) {
    for (Carreau i : this.listeCarr) {
        if (i.getLettre() == a) {
            return true;
        }
    }
    return false;
}

/**
 * Accesseur de la largeur d'un carreau du mur
 *
 * @param lettre La lettre du carreau dont on souhaite connaître la largeur.
 * @param m Le mur
 * @return La largeur du carreau demandé
 */
public static int getLargeurCar(char lettre, Mur m) {
    int j = 0;
    lettre = Character.toLowerCase(lettre);
    for (char i : Lettres) {
        if (i == lettre) {
            return dim[0][j];
        }
        ++j;
    }
    if (lettre == 'x') {
        return dimPieceNeutre[0][m.getNumPieceNeutre()];
    }

    System.exit(2);
    return 0;
}

/**
 * Accesseur de la hauteur d'un carreau du mur
 *
 * @param lettre La lettre du carreau dont on souhaite connaître la hauteur.
 * @param m Le mur
 * @return La hauteur du carreau ou 0 si ce dernier n'existe pas
 */
public static int getHauteurCar(char lettre, Mur m) {
    int j = 0;

```

```

        lettre = Character.toLowerCase(lettre);
        for (char i : Lettres) {
            if (i == lettre) {
                return dim[1][j];
            }
            ++j;
        }
        if (lettre == 'x') {
            return dimPieceNeutre[1][m.getNumPieceNeutre()];
        }

        System.exit(2);
        return 0;
    }

    /**
     * Stockage d'une chaîne de caractères contenant une liste de carreaux
     *
     * @return sb la chaîne de caractères
     */
    public String toString() {
        int largeurPris = 0;
        char[][] tab = new char[COTESMAXCARREAU][this.LargeurCarreaux()];
        for (int i = 0; i < COTESMAXCARREAU; i++) {
            for (int j = 0; j < this.LargeurCarreaux(); j++) {
                tab[i][j] = ' ';
            }
        }
        for (Carreau i : this.listeCarr) {
            for (int j = 0; j < i.getHauteur(); j++) {
                for (int k = 0; k < i.getLargeur(); k++) {
                    tab[tab.length - 1 - j][k + largeurPris] =
i.getLettre();
                }
            }
            largeurPris += i.getLargeur();
        }
        StringBuilder sb = new StringBuilder("");
        for (int i = 0; i < COTESMAXCARREAU; ++i) {
            char derCarac = ' ';
            boolean prem = true;
            for (int j = 0; j < tab[0].length; j++) {
                if (!(derCarac == tab[tab.length - 1][j]) && !prem) {
                    sb.append(" ");
                }
                sb.append(tab[i][j]);
                sb.append(" ");

                prem = false;
                derCarac = tab[tab.length - 1][j];
            }
            sb.append(System.LineSeparator());
        }
        return sb.toString();
    }
}

```

## Classe Score

```
package components;
/**
 * Classe Score.
 *
 * @author Antoine Després
 * @author Thibault Henrion
 * @version 1.0
 */
public class Score {
    private int carteÉcartée;

    /**
     * Stocke la phrase détaillant le score atteint
     *
     * @param s Le score
     * @param p La liste de carreaux
     * @param m Le mur
     * @return Chaîne de caractères détaillant le score
     */
    public String toString(Score s, ListeCarreaux p, Mur m) {
        return (5 * m.getHauteurMin() - p.size() - carteÉcartée + " points ("
+ m.getHauteurMin()
+ " niveaux complets, " + p.size() + " carreaux non
posés, " + carteÉcartée + " cartes écartées)");
    }

    /**
     * Incrémente la valeur de carte écartée. A utiliser lorsque le joueur
écarte
     * une carte
     */
    public void écarter() {
        carteÉcartée++;
    }
}
```

## Classe FctJeu

```
package Appli;
import java.util.Scanner;

import components.Carreau;
import components.Carte;
import components.JeuDeCartes;
import components.ListeCarreaux;
import components.Mur;
import components.Score;
import components.TypeErreur;

/**
 * Classe FctJeu pour l'appel de commande et la détection de fin de partie
 */
```



```

* @author Antoine Després
* @author Thibault Henrion
*/
public class FctJeu {

    /**
     * Vérifie si la partie est terminée ou non
     *
     * @param j Le jeu de carte
     * @param p La liste des carreaux
     * @param stop La condition stop entrée par le joueur
     * @return true si la partie est terminée, false sinon
     */
    public static boolean estTerminee(JeuDeCartes j, ListeCarreaux p, boolean
stop) {
        return j.getJeuCartes().size() == 0 || p.getListeCarreaux().size() ==
0 || stop == true;
    }

    /**
     * Appelle la commande correspondant à la saisie si elle est correcte.
     *
     * @param sc Le scanner
     * @param m Le mur
     * @param CarteTirée La carte tirée
     * @param p La liste des carreaux
     * @param s Le score
     * @param AucunCarreau booléen à true si aucun carreau ne correspond à la
carte
tirée
     */
    public static void appelCommande(Scanner sc, Mur m, Carte CarteTirée,
ListeCarreaux p, Score s,
boolean AucunCarreau) {
        String mot = "";
        if (AucunCarreau) // si aucun carreau jouable avec cette carte, on
force le tour suivant
            mot = "next";
        else
            mot = sc.next(); // sinon, on lit l'entrée de l'utilisateur
        switch (mot) {
            case "next":
                s.écarter(); // écarter la carte et décrémenter le score
                Main.setMsgErreur(TypeErreur.CORRECT); // la saisie est
correcte
                break;
            case "stop":
                Main.setStop(true); // la fonction estTerminee détectera la
demande d'arrêt
                Main.setMsgErreur(TypeErreur.CORRECT);
                break;
            default:
                if (mot.length() > 1) { // il ne s'agit pas d'une seule lettre
!
                Main.setMsgErreur(TypeErreur.SAISIE); // C'est donc une
erreur de saisie
                } else {
                    int absBG, ordBG; // coordonnées à saisir
                    absBG = ordBG = 0;
                }
            }
        }
    }
}

```

```

        if (sc.hasNextInt())
            ordBG = sc.nextInt();
        else {
            Main.setMsgErreur(TypeErreur.SAISIE); // si on ne
saisit pas un int : erreur de saisie
            break;
        }
        if (sc.hasNextInt())
            absBG = sc.nextInt();
        else {
            Main.setMsgErreur(TypeErreur.SAISIE);
            break;
        }
        if (p.carreauDispo(CarteTirée).contient(mot.charAt(0)))
        { // si un carreau correspond...
            Carreau c = p.getCarreau(mot.charAt(0));
            Main.setMsgErreur(m.placerCarreau(c, absBG,
ordBG)); // on récupère le message d'erreur
            if (Main.getMsgErreur() == TypeErreur.CORRECT) {
                p.retirerCarreau(mot.charAt(0)); // si le
placement est CORRECT, on peut retirer le carreau de
                // la liste
            }
        } else {
            Main.setMsgErreur(TypeErreur.SAISIE); // sinon
c'est une erreur de saisie
        }
    }
}
}
}
}

```

## Classe Mur

```
001 package components;
002 import java.util.*;
003
004 /**
005  * Classe Mur.
006  *
007  * @author Antoine Després
008  * @author Thibault Henrion
009  * @version 1.0
010  */
011 public class Mur {
012     private static final int NBPIECENEUTRE = 2; // Deux positions de
013     pièce neutre possibles
014     private static final char LETTREPIECENEUTRE = 'x';
015     private static final Carreau PIECENEUTRE1 = new Carreau(1, 3,
016     LETTREPIECENEUTRE, false); // pièce verticale
017     private static final int[] POSPIECENEUTRE1 = { 1, 5 };
018     private static final Carreau PIECENEUTRE2 = new Carreau(3, 1,
019     LETTREPIECENEUTRE, false); // pièce horizontale
020     private static final int[] POSPIECENEUTRE2 = { 1, 3 };
021
022     /**
023      * Les axes sont en base 10 On peut ainsi savoir quand décaler
024      l'affichage des
025      * coordonnées des lignes
026      */
027     private static final int BASEAXES = 10;
028
029     /**
030      * Numéro de la pièce neutre
031      */
032     private int numPieceNeutre;
033
034     /**
035      * La largeur du mur est constante et définie à 5 unités.
036      */
037     private final static int LARGEUR = 5;
038
039     /**
040      * Nombre de cotés sur un carreau.
041      */
042     private final static int COTESCARREAU = 2;
043
044     /**
045      * Tableau de caractères à deux dimensions.
046      */
047     @SuppressWarnings("unchecked")
048     private ArrayList<Character>[] mur = new ArrayList[LARGEUR];
049
050     /**
051      * Constructeur de {@link Mur}
052      */
053     public Mur() {
054         for (int i = 0; i < LARGEUR; i++) {
055             this.mur[i] = new ArrayList<Character>();
```

```

056         }
057
058     }
059
060     /**
061     * Accesseur du numéro de pièce neutre
062     *
063     * @return Le numéro de la pièce neutre
064     */
065     public int getNumPieceNeutre() {
066         return numPieceNeutre;
067     }
068
069     /**
070     * Place la pièce neutre sur le mur. La position et l'orientation de
071 celle-ci
072 sont déterminées aléatoirement.
073     */
074     public void placerPieceNeutre() {
075         Random piece = new Random(); //orientation aléatoire
076         Random position = new Random(); //position aléatoire
077         if (piece.nextInt(NBPIECENEUTRE) == 0) {
078             this.numPieceNeutre = 0;
079             this.placerCarreau(PIECENEUTRE1,
080 POSPIECENEUTRE1[position.nextInt(2)], 1);
081         } else {
082             this.numPieceNeutre = 1;
083             this.placerCarreau(PIECENEUTRE2,
084 POSPIECENEUTRE2[position.nextInt(2)], 1);
085         }
086     }
087
088     /**
089     * Accesseur de la hauteur maximale.
090     *
091     * @return hauteurMax La hauteur de la ligne remplie la plus haute
092 dans le mur.
093     */
094     public int getHauteurMax() {
095         int hauteurMax = 0;
096         for (ArrayList<Character> i : mur) {
097             if (i.size() >= hauteurMax)
098                 hauteurMax = i.size();
099         }
100         return hauteurMax;
101     }
102
103     /**
104     * Accesseur de la hauteur minimale.
105     *
106     * @return hauteurMin La hauteur de la ligne remplie la plus basse
107 dans le mur.
108     */
109     public int getHauteurMin() {
110         int hauteurMin = this.mur[0].size();
111         for (ArrayList<Character> i : mur) {
112             if (i.size() <= hauteurMin)
113                 hauteurMin = i.size();
114         }

```

```

115         return hauteurMin;
116     }
117
118     /**
119     * Place un carreau sur le {@link Mur}.
120     *
121     * @param c Le carreau
122     * @param absBG Le point d'abscisse en bas à gauche du {@link
123 Carreau}
124     * @param ordBG Le point d'ordonnée en bas à gauche du {@link
125 Carreau}
126     * @return Un type d'erreur
127     */
128     public TypeErreur placerCarreau(Carreau c, int absBG, int ordBG) {
129         absBG--;
130         ordBG--;
131         if (absBG < 0 || ordBG < 0)
132             return TypeErreur.DEPASSEMENT;
133         TypeErreur etatPlacement;
134         if (c.getLettre() == LETTREPIECENEUTRE ||
135 this.placementCorrect(c, absBG, ordBG) == TypeErreur.CORRECT) {
136             etatPlacement = TypeErreur.CORRECT;
137             for (int i = absBG; i < absBG + c.getLargeur(); i++) {
138                 for (int j = 0; j < c.getHauteur(); j++)
139                     this.mur[i].add(c.getLettre());
140             }
141             return etatPlacement;
142         }
143         return this.placementCorrect(c, absBG, ordBG);
144     }
145
146     /**
147     * Vérifie si le placement de carreau est correct
148     *
149     * @param c Le {@link Carreau}
150     * @param absBG L'abscisse du point en bas à gauche du carreau
151     * @param ordBG L'ordonnée du point en bas à gauche du carreau
152     * @return Un type d'erreur
153     */
154     private TypeErreur placementCorrect(Carreau c, int absBG, int ordBG)
155     {
156         if (dépasse(c, absBG, ordBG) == TypeErreur.DEPASSEMENT) {
157             return TypeErreur.DEPASSEMENT;
158         }
159         TypeErreur baseRepose = baseRepose(c, absBG, ordBG);
160         if (baseRepose == TypeErreur.REPOS_BASE)
161             return TypeErreur.REPOS_BASE;
162         if (touche(c, absBG, ordBG, baseRepose) ==
163 TypeErreur.NON_CONTACT)
164             return TypeErreur.NON_CONTACT;
165         if (cloneBord(c, absBG, ordBG) == TypeErreur.CLONAGE)
166             return TypeErreur.CLONAGE;
167
168         return TypeErreur.CORRECT;
169     }
170
171     /**
172     * Vérifie si le carreau dépasse de la zone à carreler
173     *

```

```

174         * @param c      Le {@link Carreau}
175         * @param absBG Le point d'abscisse en bas à gauche du {@link
176 Carreau}
177         * @param ordBG Le point d'ordonnée en bas à gauche du {@link
178 Carreau}
179         * @return Un type d'erreur
180         */
181         private TypeErreur dépasse(Carreau c, int absBG, int ordBG) {
182             if (absBG + c.getLargeur() <= LARGEUR && absBG >= 0 && ordBG
183 >= 0) {
184                 return TypeErreur.CORRECT;
185             }
186             return TypeErreur.DEPASSEMENT;
187         }
188
189         /**
190         * Vérifie si le carreau touche un autre carreau et repose sur une
191 base stable
192         *
193         * @param c      Le {@link Carreau}
194         * @param absBG Le point d'abscisse en bas à gauche du {@link
195 Carreau}
196         * @param ordBG Le point d'ordonnée en bas à gauche du {@link
197 Carreau}
198         * @return Un type d'erreur
199         */
200         private TypeErreur touche(Carreau c, int absBG, int ordBG,
201 TypeErreur baseRepose) {
202             if (baseRepose == TypeErreur.CORRECT && ordBG > 0)
203                 return TypeErreur.CORRECT;
204             if (ordBG == 0) {
205                 boolean touche = false;
206                 for (int i = -1; i < COTESCARREAU + c.getLargeur() - 1
207 && absBG + i < LARGEUR && absBG + i >= 0
208 && !touche; i += 2 + c.getLargeur() - 1) {
209                     if (this.mur[absBG + i].size() >= ordBG &&
210 !this.mur[absBG + i].isEmpty()) {
211                         touche = true;
212                     }
213                 }
214                 if (absBG == 0) {
215                     if (this.mur[absBG + c.getLargeur()].size() >=
216 ordBG && !this.mur[absBG + c.getLargeur()].isEmpty()) {
217                         touche = true;
218                     }
219                 }
220                 if (!touche) {
221                     return TypeErreur.NON_CONTACT;
222                 }
223             }
224             return TypeErreur.CORRECT;
225         }
226
227         /**
228         * Vérifie si le carreau dépasse de la zone à carreler
229         *
230         * @param c      Le {@link Carreau}
231         * @param absBG Le point d'abscisse en bas à gauche du {@link
232 Carreau}

```

```

233         * @param ordBG Le point d'ordonnée en bas à gauche du {@link
234 Carreau}
235         * @return Un type d'erreur
236         */
237         private TypeErreur baseRepose(Carreau c, int absBG, int ordBG) {
238             for (int i = 0; i < c.getLargeur(); i++) {
239                 if (this.mur[absBG + i].size() != ordBG && ordBG != 0)
240 {
241                     return TypeErreur.REPOS_BASE;
242                 } else if (ordBG == 0 && !this.mur[absBG +
243 i].isEmpty()) {
244                     return TypeErreur.REPOS_BASE;
245                 }
246             }
247             return TypeErreur.CORRECT;
248         }
249
250         /**
251         * Vérifie si le carreau clone un autre carreau sur son bord
252 inférieur
253         *
254         * @param c Le Carreau
255         * @param absBG Le point d'abscisse en bas à gauche du {@link
256 Carreau}
257         * @param ordBG Le point d'ordonnée en bas à gauche du {@link
258 Carreau}
259         * @return Un type d'erreur
260         */
261         private TypeErreur cloneBordBas(Carreau c, int absBG, int ordBG) {
262             if (ordBG > 0) {
263                 if
264 (ListeCarreaux.getLargeurCar(this.mur[absBG].get(ordBG - 1), this) ==
265 c.getLargeur()) {
266                     char lettrePré = mur[absBG].get(ordBG - 1);
267                     char lettreSuiv = ' '; // comparaison des lettres
268 sous le carreau
269                     boolean estEgal = true;
270                     for (int i = 1; i < c.getLargeur() && estEgal;
271 i++) {
272                         lettreSuiv = mur[absBG + i].get(ordBG -
273 1);
274                         estEgal = lettrePré == lettreSuiv;
275                     }
276                     if (estEgal)
277                         return TypeErreur.CLONAGE;
278                     else
279                         return TypeErreur.CORRECT;
280                 }
281             }
282             return TypeErreur.CORRECT;
283         }
284
285         /**
286         * Vérifie si le carreau clone un autre carreau sur un côté
287         *
288         * @param c Le carreau
289         * @param absBG Le point d'abscisse en bas à gauche du {@link
290 Carreau}
291

```

```

292     * @param ordBG Le point d'ordonnée en bas à gauche du {@link
293 Carreau}
294     * @param coté Côté à vérifier (gauche ou droite)
295     * @return Un type d'erreur
296     */
297     private TypeErreur cloneBordCote(Carreau c, int absBG, int ordBG,
298 int coté) {
299         int colonneCoté = absBG;
300         if (coté == -1) {
301             colonneCoté += coté;
302             if (colonneCoté < 0) {
303                 return TypeErreur.CORRECT;
304             }
305         } else if (coté == 1) {
306             colonneCoté += c.getLargeur();
307             if (colonneCoté > LARGEUR - 1) {
308                 return TypeErreur.CORRECT;
309             }
310         }
311         if (ordBG + c.getHauteur() > this.mur[colonneCoté].size() ||
312 this.mur[colonneCoté].isEmpty()) {
313             return TypeErreur.CORRECT;
314         }
315         if
316 (ListeCarreaux.getHauteurCar(this.mur[colonneCoté].get(ordBG), this) ==
317 c.getHauteur()) {
318             char lettrePré = mur[colonneCoté].get(ordBG);
319             char lettreSuiv = ' ';
320             boolean estEgal = true;
321             for (int i = 1; i < c.getHauteur() && estEgal; i++) {
322                 lettreSuiv = mur[colonneCoté].get(ordBG + i);
323                 estEgal = lettrePré == lettreSuiv;
324             }
325             if (estEgal)
326                 return TypeErreur.CLONAGE;
327             else
328                 return TypeErreur.CORRECT;
329         }
330         return TypeErreur.CORRECT;
331     }
332
333     /**
334     * Vérifie qu'un carreau ne clone pas le bord d'un carreau autour de
335 lui
336     *
337     * @param c Le carreau
338     * @param absBG Le point d'abscisse en bas à gauche du {@link
339 Carreau}
340     * @param ordBG Le point d'ordonnée en bas à gauche du {@link
341 Carreau}
342     * @return Un type d'erreur
343     */
344     public TypeErreur cloneBord(Carreau c, int absBG, int ordBG) {
345         if (cloneBordBas(c, absBG, ordBG) == TypeErreur.CLONAGE
346             || cloneBordCote(c, absBG, ordBG, -1) ==
347 TypeErreur.CLONAGE
348             || cloneBordCote(c, absBG, ordBG, 1) ==
349 TypeErreur.CLONAGE) {
350             return TypeErreur.CLONAGE;

```



```

351         }
352         return TypeErreur.CORRECT;
353     }
354
355     /**
356     * Affiche le mur complet avec coordonnées.
357     *
358     * @return sb La chaîne de caractères contenant le mur
359     */
360     public String toString() {
361         StringBuilder sb = new StringBuilder("");
362         for (int i = this.getHauteurMax(); i >= 0; --i) {
363             if (this.getHauteurMax() >= BASEAXES - 1 && i + 1 <
364 BASEAXES) // pour éviter un décalage à partir de la ligne 10
365                 sb.append(" ");
366             sb.append(i + 1);
367             sb.append(" ");
368             for (int j = 0; j < LARGEUR; j++) {
369                 if (i < this.mur[j].size()) {
370                     sb.append(this.mur[j].get(i));
371                 } else {
372                     sb.append(" ");
373                 }
374                 sb.append(" ");
375             }
376             sb.append(System.LineSeparator());
377         }
378         if (this.getHauteurMax() >= BASEAXES - 1)
379             sb.append(" 1 2 3 4 5");
380         else
381             sb.append(" 1 2 3 4 5");
382         return sb.toString();
383     }
384 }

```

## Classe Main

```

package Appli;

import java.util.Scanner;

import components.Carte;
import components.JeuDeCartes;
import components.ListeCarreaux;
import components.Mur;
import components.Score;
import components.TypeErreur;

/**
 * Classe Main.
 *
 * @author Antoine Després
 * @author Thibault Henrion
 * @version 1.0
 */
public class Main {

```

```

/**
 * Booléen permettant d'arrêter la partie en saisissant la commande "stop"
 */
private static boolean stop = false;

/**
 * Contient le message d'erreur à traiter
 */
private static TypeErreur msgErreur = TypeErreur.CORRECT;

public static void main(String[] args) {
    Mur m = new Mur();
    m.placerPieceNeutre();
    ListeCarreaux p = new ListeCarreaux(true); // p pour ne pas confondre
1 et 1
    JeuDeCartes j = new JeuDeCartes();
    Score s = new Score();
    Scanner sc = new Scanner(System.in);

    do { // faire tant que la partie n'est pas terminée
        System.out.println(m + System.LineSeparator()); // afficher le
mur
        Carte carteTirée = j.tirerCarte(); // tirer une carte
        System.out.println(carteTirée.getTypeCarte() +
System.LineSeparator()); // afficher son instruction
        ListeCarreaux pTrié = p.carreauDispo(carteTirée); // accès aux
carreaux jouables correspondant
        if (!pTrié.estVide()) { // affichage de ces derniers
            System.out.println(pTrié);
        }
        do { // faire tant que la saisie n'est pas correcte
            FctJeu.appelCommande(sc, m, carteTirée, p, s,
pTrié.estVide());
            System.err.print(msgErreur); // Affichage du message
d'erreur si existant
        } while (msgErreur != TypeErreur.CORRECT);
    } while (!FctJeu.estTerminee(j, p, stop));

    System.out.println(s.toString(s, p, m)); // Affichage du score en fin
de partie
}

/**
 * Accesseur de stop
 */
*
* @return stop
*/
public static boolean getStop() {
    return stop;
}

/**
 * Mutateur de stop
 */
*
* @param sStop la variable stop
*/
public static void setStop(boolean sStop) {
    stop = sStop;
}

```

```

    }

    /**
     * Accesseur du message d'erreur
     *
     * @return msgErreur Le message d'erreur
     */
    public static TypeErreur getMsgErreur() {
        return msgErreur;
    }

    /**
     * Mutateur du message d'erreur
     *
     * @param err Le message d'erreur à stocker
     */
    public static void setMsgErreur(TypeErreur err) {
        msgErreur = err;
    }
}

```

## Bilan de projet

Nous avons trouvé ce premier projet de Java très enrichissant, il nous a permis de bien nous familiariser avec ce nouveau langage et maîtriser les bases ainsi que des composantes de certaines bibliothèques comme les collections et la génération de nombres aléatoires tout en développant nos compétences algorithmiques.

Les principales difficultés rencontrées étaient concentrées dans le contrôle de la validité du placement d'un carreau (notamment le clonage !), les algorithmes nous ayant parfois donné du fil à retordre. Nous avons fait de notre mieux pour respecter le plus possible les consignes précises qui ont été données. Nous avons veillé à ce qu'aucun affichage ne soit en dehors de la méthode principale et que tous les attributs soient privés afin de respecter les bonnes pratiques de programmation.

```

a 4 5
Le carreau clone le côté d'un carreau déjà posé
a 3 4

5
4 e   b
3 e B b a x
2 e B D D x
1 c c D D x
  1 2 3 4 5

bleu

      g g      i i i
d d      g g  h h h i i i
d d  f f f g g  h h h i i i

stop


2 points (3 niveaux complets, 12 carreaux non posés, 1 cartes écartées)

```

Figure 2 : Exécution du programme en console. L'erreur de placement est bien détectée, le calcul du score et son affichage fonctionnent.

## Charte graphique

Java Orange: RGB(237, 139, 0) 

Java Blue : RGB(0, 115, 150) 

Oracle. Logos | Oracle. Consulté le 14 février 2020, à l'adresse  
<https://www.oracle.com/legal/logos.html>

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.