RAPPORT

PROJET BPO

CHRISTIAN GAMO
(102)
MONICA HUYNH

2019
2020

TABLE DES MATIERES

1. Présentation du projet.....p.3

2. Diagramme UML.....p.3

3. Tests unitaires.....p.4-11

4. Code Java du projet.....p.12-32

5. Bilan de projet.....p.32-33

PRESENTATION DU PROJET

Le projet de BPO consiste à réaliser un programme permettant de jouer à une partie de *The Tiler Team* (ou 2T, dit TOOTY). C'est un jeu collaboratif reposant sur une zone de jeu (un mur) et des pièces de jeu (les carreaux), où une équipe de carreleurs coopèrent pour paver (au mieux) un mur. Il y a 9 carreaux de couleur bleu et 9 autres de couleur rouge. Il y a également un paquet de cartes composé de 33 cartes de jeu (9 cartes « Rouge », 9 cartes « Bleu », 5 cartes « Taille 1 », 5 cartes « Taille 2 » et 5 cartes « Taille 3 »). Au cours de ce projet, nous avons ainsi dû réaliser différentes classes au sein de paquets afin d'interpréter ces objets matériels et physiques en des objets informatiques.

DIAGRAMME UML

Nous avons deux packages : `jeu` et `composantDeJeu`. `jeu` utilise le package `composantDeJeu`. La classe `Appli.java` utilise toutes les classes (sauf les énumarations) de ce package.

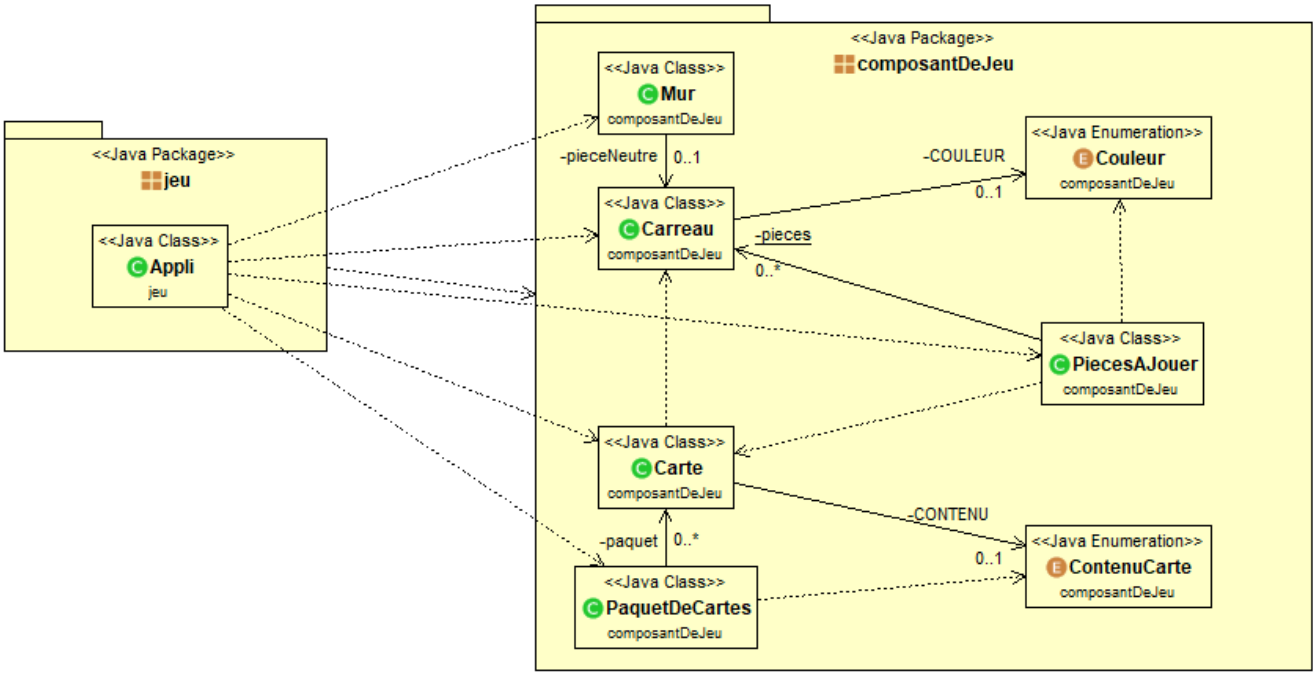

FIGURE 1. DIAGRAMME DE PAQUETAGE

TESTS UNITAIRES

Voici le code Java de nos tests unitaires. Tous se sont exécutés avec succès. Nous avons testé chaque méthode de chaque classe sauf les méthodes simples telles que les getters, les setters, ou les booléens et les constructeurs.

Test Carreau

```
/**
 * @file CarreauTest.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Test jUnit de la classe Carreau
 */
package composantDeJeu;

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

class CarreauTest {
    Mur m = new Mur();
    Carreau valide = new Carreau(1, 1, 'a', Couleur.BLEU);
    Carreau invalide = new Carreau(1, 1, 'A', Couleur.ROUGE);

    @Test
    public void testEstPlacable() {
       int ligneNeutre = 0;
       int colonneNeutre = -1;

       for (int i = 0; i < Mur.getColonne(); ++i) {
           if (m.getTerrain().get(ligneNeutre)[i] == 'x') {
              colonneNeutre = i;
              break;
           }
       }

       assertTrue(colonneNeutre != -1);

       assertTrue(valide.estPlacable(m,
              ligneNeutre + m.getPieceNeutre().getHauteur(), colonneNeutre));
       assertFalse(invalide.estPlacable(m, ligneNeutre, colonneNeutre));

    }

    @Test
    public void testToucheCarreau() {
       int ligneNeutre = 0;
       int colonneNeutre = -1;

       for (int i = 0; i < Mur.getColonne(); ++i) {
           if (m.getTerrain().get(ligneNeutre)[i] == 'x') {
              colonneNeutre = i;
              break;
           }
       }
```

```
    assertTrue(valide.toucheCarreau(m,
           ligneNeutre + m.getPieceNeutre().getHauteur(), colonneNeutre));

    if (colonneNeutre == 0)
        assertFalse(valide.toucheCarreau(m, m.getPieceNeutre().getHauteur(),
               colonneNeutre + m.getPieceNeutre().getLargeur() + 1));
    else {
        assertFalse(invalide.toucheCarreau(m,
               m.getPieceNeutre().getHauteur(), colonneNeutre - 1));
    }
}

@Test
public void testBaseStable() {
    Carreau c = new Carreau(2, 1, 'c', Couleur.BLEU);
    int ligneNeutre = 0;
    int colonneNeutre = -1;

    for (int i = 0; i < Mur.getColonne(); ++i) {
        if (m.getTerrain().get(ligneNeutre)[i] == 'x') {
            colonneNeutre = i;
            break;
        }
    }

    assertTrue(valide.baseStable(m,
           ligneNeutre + m.getPieceNeutre().getHauteur(), colonneNeutre));

    if (colonneNeutre == 0)
        assertFalse(c.baseStable(m, m.getPieceNeutre().getHauteur(),
               colonneNeutre + m.getPieceNeutre().getLargeur() + 1));
    else {
        assertFalse(c.baseStable(m, m.getPieceNeutre().getHauteur(),
               colonneNeutre - 1));
    }
}

@Test // Ce test teste ACarreauAdjacent() et cloneCarreau()
public void testACarreauAdjacent() {
    PiecesAJouer pion = new PiecesAJouer();
    Carreau f = new Carreau(3, 1, 'f', Couleur.BLEU);
    Carreau e = new Carreau(1, 3, 'e', Couleur.BLEU);

    int ligneNeutre = 0;
    int colonneNeutre = -1;
    for (int i = 0; i < Mur.getColonne(); ++i) {
        if (m.getTerrain().get(ligneNeutre)[i] == 'x') {
            colonneNeutre = i;
            break;
        }
    }
    if (m.getPieceNeutre().getHauteur() == 1) {
        assertTrue(f.aCarreauAdjacent(m, ligneNeutre + 1, colonneNeutre,
               pion));
        assertFalse(invalide.aCarreauAdjacent(m, ligneNeutre + 1,
               colonneNeutre, pion));
    } else {
        if (colonneNeutre == 0) {
            assertTrue(e.aCarreauAdjacent(m, ligneNeutre, colonneNeutre + 1,
                   pion));
            assertFalse(invalide.aCarreauAdjacent(m, ligneNeutre,
```

```
                    colonneNeutre + 1, pion));
            } else {
                assertTrue(e.aCarreauAdjacent(m, ligneNeutre, colonneNeutre - 1,
                        pion));
                assertFalse(invalide.aCarreauAdjacent(m, ligneNeutre,
                        colonneNeutre - 1, pion));
            }
        }

    }
}
```

Test Mur

```
/**
 * @file MurTest.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Test jUnit de la classe Mur
 */

package composantDeJeu;


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

import org.junit.jupiter.api.Test;

public class MurTest {
    Mur m = new Mur();
    Carreau c = new Carreau(2, 1, 'c', Couleur.BLEU);
    PiecesAJouer pions = new PiecesAJouer();

    @Test
    public void testPlacerCaroNeutre() {
        int cpt = 0;
        for (int i = 0; i < m.getTerrain().size(); ++i) {
            for (int j = 0; j < Mur.getColonne(); ++j) {
                if (m.getTerrain().get(i)[j] == 'x')
                    ++cpt;
            }
        }
        assertTrue(cpt == 3);
    }

    @Test
    public void testPlacerCarreau() {
        int ligne = -1;
        int colonne = -1;
        for (int i = 0; i < m.getTerrain().size(); ++i) {
            for (int j = 0; j < Mur.getColonne(); ++j) {
                if (!(c.getUtilisé()) && !(c.depasseZone(m, i, j))
                        && (c.estPlacable(m, i, j))
                        && (c.toucheCarreau(m, i, j)) && (c.baseStable(m, i, j))
                        && !(c.aCarreauAdjacent(m, i, j, pions))) {
                    m.placerCarreau(c, i, j);
                    ligne = i;
                    colonne = j;
                }
            }
        }
```

```
        assertTrue(ligne != -1 && colonne != -1);

        for (int k = 0; k < c.getHauteur(); ++k) {
            for (int l = 0; l < c.getLargeur(); ++l) {
                assertTrue(m.getTerrain().get(ligne + k)[colonne + l]
                == c.getLettre());
            }
        }
    }


    @Test
    public void testToString() {
        Mur m = new Mur();
        int ligneNeutre = 0;
        int colonneNeutre = -1;

        String cas1Attendu = " 2           \r\n" +
                             " 1 x x x     \r\n" +
                             "   1 2 3 4 5 ";

        String cas2Attendu = " 4           \r\n" +
                             " 3         x \r\n" +
                             " 2         x \r\n" +
                             " 1         x \r\n" +
                             "   1 2 3 4 5 ";

        String cas3Attendu = " 4           \r\n" +
                             " 3 x         \r\n" +
                             " 2 x         \r\n" +
                             " 1 x         \r\n" +
                             "   1 2 3 4 5 ";

        String cas4Attendu = " 2           \r\n" +
                             " 1     x x x \r\n" +
                             "   1 2 3 4 5 ";

        System.out.print(m.toString());

        for (int i = 0; i < Mur.getColonne(); ++i) {
            if (m.getTerrain().get(ligneNeutre)[i] == 'x') {
                colonneNeutre = i;
                break;
            }
        }

        if (colonneNeutre == 0) {
            if (m.getPieceNeutre().getHauteur() == 1)
                assertEquals(cas1Attendu, m.toString());
            else
                assertEquals(cas3Attendu, m.toString());
        } else {
            if (m.getPieceNeutre().getHauteur() == 1)
                assertEquals(cas4Attendu, m.toString());
            else
                assertEquals(cas2Attendu, m.toString());
        }

    }

}
```

Test Carte

```java
/**
 * @file CarteTest.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Test jUnit de la classe Carte
 */

package composantDeJeu;

import static org.junit.Assert.*;

import org.junit.Test;

public class CarteTest {

	@Test
	public void testConditionCarte() {
		Carte rouge = new Carte(ContenuCarte.ROUGE);
		Carte taille1 = new Carte(ContenuCarte.TAILLE1);
		Carte taille2 = new Carte(ContenuCarte.TAILLE2);
		Carte taille3 = new Carte(ContenuCarte.TAILLE3);
		Carreau carRouge = new Carreau (1,1,'A',Couleur.ROUGE);
		Carreau carBleu = new Carreau(1,1,'a', Couleur.BLEU);
		Carreau carTaille2 = new Carreau (1,2,'b', Couleur.BLEU);
		Carreau carTaille3 = new Carreau (3,1,'F', Couleur.ROUGE);

		assertTrue(rouge.conditionCarte(carRouge));
		assertFalse(rouge.conditionCarte(carBleu));
		assertTrue(taille1.conditionCarte(carRouge));
		assertTrue(taille2.conditionCarte(carTaille2));
		assertTrue(taille3.conditionCarte(carTaille3));
	}

	@Test
	public void testToString() {
		Carte rouge = new Carte(ContenuCarte.ROUGE);
		Carte bleu = new Carte(ContenuCarte.BLEU);
		Carte taille1 = new Carte(ContenuCarte.TAILLE1);
		Carte taille2 = new Carte(ContenuCarte.TAILLE2);
		Carte taille3 = new Carte(ContenuCarte.TAILLE3);

		assertEquals(rouge.toString(),"rouge");
		assertEquals(bleu.toString(),"bleu");
		assertEquals(taille1.toString(),"taille 1");
		assertEquals(taille2.toString(),"taille 2");
		assertEquals(taille3.toString(),"taille 3");

	}

}
```

Test PaquetDeCartes

```java
/**
 * @file PaquetDeCartesTest.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Test jUnit de la classe PaquetDeCartes
 */
```

```java
package composantDeJeu;

import static org.junit.Assert.*;

import org.junit.Test;

public class PaquetDeCartesTest {

    @Test
    public void testPiocher() {
       PaquetDeCartes p = new PaquetDeCartes(
              PaquetDeCartes.getNbCartesCouleur(),
              PaquetDeCartes.getNbCartesTaille());
       String contenu = p.getPaquet().get(p.getPaquet().size() - 1)
              .getContenuCarte();
       Carte pioche = p.piocher();
       assertEquals(contenu, pioche.getContenuCarte());
    }

    @Test
    public void testRetirer() {
       PaquetDeCartes p = new PaquetDeCartes(
              PaquetDeCartes.getNbCartesCouleur(),
              PaquetDeCartes.getNbCartesTaille());
       assertEquals(p.getPaquet().size(), PaquetDeCartes.getMaxCartes());
       p.piocher();
       assertEquals(p.getPaquet().size(), PaquetDeCartes.getMaxCartes() - 1);
    }

    @Test
    public void testPlusDeCartes() {
       PaquetDeCartes pVide = new PaquetDeCartes();
       assertTrue(pVide.getPaquet().size() == 0);
    }

    @Test
    public void testAjouter() {
       PaquetDeCartes p = new PaquetDeCartes(
              PaquetDeCartes.getNbCartesCouleur(),
              PaquetDeCartes.getNbCartesTaille());
       PaquetDeCartes pVide = new PaquetDeCartes();
       Carte pioche = p.piocher();
       pVide.ajouter(pioche);
       assertEquals(pioche.getContenuCarte(),
              pVide.getPaquet().get(0).getContenuCarte());
    }
}
```

Test PiecesAJouer

```java
/**
 * @file PiecesAJouerTest.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Test jUnit de la classe PiecesAJouer
 */

package composantDeJeu;
```

```
import static org.junit.Assert.*;

import org.junit.Test;

public class PiecesAJouerTest {

    @Test
    public void testPlusDeCarreau() {
       PiecesAJouer pion = new PiecesAJouer();

       assertFalse(pion.plusDeCarreau());

       for (int i = 0; i < 18; ++i) {
           pion.getPieces()[i].setUtilisé(true);
       }

       assertTrue(pion.plusDeCarreau());
    }

    @Test
    public void testToString() {
       PiecesAJouer pion = new PiecesAJouer();
       Carte c = new Carte(ContenuCarte.TAILLE1);
       String attendu = "       e            E     \r\n" +
                        "  b    e       B    E     \r\n" +
                        "a b cc e fff A B CC E FFF \r\n";
       assertEquals(pion.toString(c), attendu);

    }
}
```

Test Appli

```
/**
 * @file AppliTest.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Test jUnit de la classe Appli
 */

package jeu;

import static org.junit.Assert.*;

import java.util.*;

import org.junit.Test;

import composantDeJeu.Carte;
import composantDeJeu.Mur;
import composantDeJeu.PaquetDeCartes;
import composantDeJeu.PiecesAJouer;

public class AppliTest {

    @Test
    public void testFinDeJeu() {
       PaquetDeCartes pVide = new PaquetDeCartes();
       PaquetDeCartes p = new PaquetDeCartes(
```

```
                PaquetDeCartes.getNbCartesCouleur(),
                PaquetDeCartes.getNbCartesTaille());
        PiecesAJouer pion = new PiecesAJouer();
        assertTrue(Appli.finDeJeu(pVide, pion));
        assertFalse(Appli.finDeJeu(p, pion));
    }

    @Test
    public void testFinDePartie() {
        Mur m = new Mur();
        PiecesAJouer pion = new PiecesAJouer();
        String phrase = "";
        PaquetDeCartes pVide = new PaquetDeCartes(); // paquet vide, 0 carte
                                                     // écartée
        int score = -18;
        phrase = score + " points (" + 0 + " niveaux complets, " + 18
                + " carreaux non posés, " + pVide.getPaquet().size()
                + " cartes écartées)";
        assertEquals(Appli.finDePartie(m, pion, pVide), phrase);

    }

    @Test
    public void testSaisie() {
        Mur m = new Mur();

        PaquetDeCartes p = new PaquetDeCartes(
        PaquetDeCartes.getNbCartesCouleur(),
        PaquetDeCartes.getNbCartesTaille());
        PaquetDeCartes tasEcarte = new PaquetDeCartes();
        p.mélanger();

        PiecesAJouer pion = new PiecesAJouer();
        pion.trier();

        Carte piocher = p.piocher();
        boolean b1 = false;
        boolean b2 = false;

        String correct = "next";
        String incorrect = "non";
        Scanner sc1 = new Scanner(correct);
        Scanner sc2 = new Scanner(incorrect);

        try {
            b1 = Appli.saisie(m, sc1, pion, piocher, tasEcarte);
            System.out.println("Tu as entré next");
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }

        try {
            b2 = Appli.saisie(m, sc2, pion, piocher, tasEcarte);
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }

        assertTrue(b1);
        assertFalse(b2);
    }

}
```

CODE JAVA DU PROJET

(Enumération) Couleur.java

```
/**
 * @file Couleur.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Type énuméré représentant la couleur possible pour un carreau
 */

package composantDeJeu;
/**
 * Couleur possible pour les pièces à jouer (carreaux)
 */
public enum Couleur {
	/** Couleur bleu */
	BLEU,
	/** Couleur rouge */
	ROUGE;
}
```

(Enumération) ContenuCarte.java

```
/**
 * @file ContenuCarte.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Type énuméré représentant le contenu possible pour une carte de jeu
 */
package composantDeJeu;
/**
 *  Contenu possible pour une carte
 */
public enum ContenuCarte {
	/** Couleur rouge */
	ROUGE,
	/** Couleur bleu */
	BLEU,
	/** Taille 1 */
	TAILLE1,
	/** Taille 2 */
	TAILLE2,
	/** Taille 3 */
	TAILLE3;
}
```

Carreau.java

```
/**
 * @file Carreau.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Type de donnée représentant un carreau de jeu
 */

package composantDeJeu;

/** Type de donnée représentant un carreau de jeu */
```

```java
public class Carreau {
    /** largeur du carreau (en unités) */
    private int largeur;
    /** hauteur du carreau (en unités) */
    private int hauteur;
    /** lettre attribuée au carreau et qui le représente */
    private char lettre;
    /** couleur du carreau */
    private Couleur COULEUR;
    /** indique si le carreau a été posé sur le mur ou non */
    private boolean utilisé = false;

    /**
     * Constructeur
     *
     * @param l      la largeur du carreau (obligatoire)
     * @param h      la hauteur du carreau (obligatoire)
     * @param lettre la lettre attribuée et représentant le carreau
     *               (obligatoire)
     * @param c      couleur de la lettre (optionnel, on ne l'indique pas pour
     *               la pièce neutre)
     * @param piece  le carreau à copier (obligatoire)
     */
    public Carreau(int l, int h, char lettre, Couleur c) {
       this.largeur = l;
       this.hauteur = h;
       this.COULEUR = c;
       this.lettre = lettre;
       if (c == Couleur.ROUGE)
           this.lettre = Character.toUpperCase(lettre);
    }

    public Carreau(int l, int h, char lettre) {
       this.largeur = l;
       this.hauteur = h;
       this.lettre = lettre;
       this.COULEUR = null;
    }

    public Carreau(Carreau piece) {
       assert (piece != null);
       this.largeur = piece.largeur;
       this.hauteur = piece.hauteur;
       this.lettre = piece.lettre;
       this.COULEUR = piece.COULEUR;
    }

    /** Modifie la lettre transmis au constructeur */
    public void setLettre(char c) { // pour l'algorithme de tri, même si c'est
                                    // pas conseillé
       this.lettre = c;
    }

    /** Modifie le booléen d'utilisation du carreau */
    public void setUtilisé(boolean b) {
       this.utilisé = b;
    }

    /** Retourne le booléen qui indique si le carreau a été utilisé ou non */
    public boolean getUtilisé() {
       return this.utilisé;
    }
```

```java
/** Retourne la couleur du carreau */
public Couleur getCouleur() {
   Couleur c = this.COULEUR;
   return c;
}

/** Retourne la hauteur du carreau */
public int getHauteur() {
   return this.hauteur;
}

/** Retourne la largeur du carreau */
public int getLargeur() {
   return this.largeur;
}

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

/**
 * Indique si le carreau est de couleur rouge
 *
 * @return vrai si le carreau est rouge
 */
public boolean estRouge() {
   return this.getCouleur() == Couleur.ROUGE;
}

/**
 * Indique si le carreau est de couleur bleu
 *
 * @return vrai si le carreau est bleu
 */
public boolean estBleu() {
   return this.getCouleur() == Couleur.BLEU;
}

/**
 * Indique si le carreau a une largeur ou hauteur de 1
 *
 * @return vrai si le carreau est de taille 1
 */
public boolean estTaille1() {
   return this.getLargeur() == 1 || this.getHauteur() == 1;
}

/**
 * Indique si le carreau a une largeur ou hauteur de 2
 *
 * @return vrai si le carreau est de taille 2
 */
public boolean estTaille2() {
   return this.getLargeur() == 2 || this.getHauteur() == 2;
}

/**
 * Indique si le carreau a une largeur ou hauteur de 3
 *
 * @return vrai si le carreau est de taille 3
```

```
     */
    public boolean estTaille3() {
       return this.getLargeur() == 3 || this.getHauteur() == 3;
    }

/**
     * Indique si le carreau que l'on veut placer ne supprime pas un autre déja
     * placé sur le mur
     *
     * @param m       le mur, zone de jeu
     * @param ligne   la ligne saisie par l'utilisateur
     * @param colonne la colonne saisie par l'utilisateur
     * @return vrai si le carreau ne supprime pas un autre carreau
     */
    public boolean estPlacable(Mur m, int ligne, int colonne) {
       assert (ligne >= 0 && colonne >= 0);
       int cpt = 0;

       // Si le carreau dépasse du mur il est forcément implacable
       if (depasseZone(m, ligne, colonne)) {
           return false;
       }
       // Vérifie si la ligne où on veut placer
       // le carreau + sa hauteur < nombre de lignes existants du tab
       if (m.getTerrain().size() > ligne + this.getHauteur()) {

           // Vérifie que pour chaque ligne, chaque case que
           // le carreau remplacera est vide
           for (int i = 0; i < this.getHauteur(); ++i) {
              for (int j = 0; j < this.getLargeur(); ++j) {
                  if (m.getTerrain().get(ligne + i)[colonne + j]
                    == Mur.getDefaultCase())
                      ++cpt;
              }
           }

           if (cpt == this.getLargeur() * this.getHauteur())
              return true;
           else {
              return false;
           }
       }

       else // Sinon il ne vérifie que la première ligne
       {
           for (int j = 0; j < this.getLargeur(); ++j) {
              if (m.getTerrain().get(ligne)[colonne + j] == Mur.getDefaultCase())
                  ++cpt;
           }

           if (cpt == this.getLargeur())
              return true;
           else
              return false;
       }
    }

    /**
     * Indique si le carreau dépasse du mur (possède une largeur de 5 unités)
     *
     * @param colonne la colonne saisie par l'utilisateur
     * @return vrai si le carreau dépasse de la zone à carreler
```

```
 */
public boolean depasseZone(Mur m, int ligne, int colonne) { // Figure 8
   assert (ligne >= 0 && colonne >= 0);
   return this.getLargeur() + colonne > Mur.getColonne()
          && ligne < m.getTerrain().size();
}

/**
 * Indique si le carreau touche au moins un carreau déjà posé
 *
 * @param m       le mur à carreler
 * @param ligne   la ligne saisie par l'utilisateur
 * @param colonne la colonne saisie par l'utilisateur
 * @return vrai si le carreau touche un carreau
 */
public boolean toucheCarreau(Mur m, int ligne, int colonne) { // Figure 9
   assert (ligne >= 0 && colonne >= 0);
   boolean b = false;
   if (ligne != 0) { // on regarde en dessous de la pièce
       for (int i = 0; i < this.getLargeur(); ++i) {
          if (m.getTerrain().get(ligne - 1)[colonne + i]
           != Mur.getDefaultCase())
              b = true;
       }
   }
   if (colonne != 0) { // on regarde à gauche de la pièce
       if (m.getTerrain().get(ligne)[colonne - 1] != Mur.getDefaultCase())
          b = true;
   }

   // on regarde à droite de la pièce
   if (colonne + getLargeur() - 1 != Mur.getColonne() - 1) {
       if (m.getTerrain().get(ligne)[colonne + getLargeur()]
       != Mur.getDefaultCase())
          b = true;
   }
   return b;
}

/**
 * Indique si la base du carreau repose sur le bas de la zone à carreler ou
 * sur des carreaux déjà posés
 *
 * @param m       le mur à carreler
 * @param ligne   la ligne saisie par l'utilisateur
 * @param colonne la colonne saisie par l'utilisateur
 * @return vrai si le carreau est stable
 */
public boolean baseStable(Mur m, int ligne, int colonne) { // Figure 10
   assert (ligne >= 0 && colonne >= 0);
   boolean b = false;
   if (ligne == 0)
       b = true;
   else {
       for (int i = 0; i < this.getLargeur(); ++i) {
          if (m.getTerrain().get(ligne - 1)[colonne + i] !=
                 Mur.getDefaultCase())
              b = true;
          else
              return false;
       }
   }
```

```
   return b;
}

/**
 * Indique si un carreau touche un autre carreau avec la même taille
 * d'attribut (largeur ou hauteur) et renvoie si celui-ci le clone
 *
 * @param m       le mur à carreler
 * @param ligne   la ligne saisie par l'utilisateur
 * @param colonne la colonne saisie par l'utilisateur
 * @param p       un tableau de Carreaux (pièces du jeu)
 * @return vrai si le carreau touche un carreau adjacent et le clone
 */
public boolean aCarreauAdjacent(Mur m, int ligne, int colonne,
       PiecesAJouer p) { // Figure 11
   assert (ligne >= 0 && colonne >= 0);
   boolean b = false;
   char stock = Mur.getDefaultCase();
   if (ligne != 0) { // on regarde en dessous de la pièce

       int hauteurMax = 0;
       // stocke la + petite valeur entre taille du terrain ou hauteur du carreau
       if (ligne + getHauteur() > m.getTerrain().size())
          hauteurMax = m.getTerrain().size() - ligne;
       else
          hauteurMax = getHauteur();

       for (int i = 0; i < this.getLargeur(); ++i) {
          // on stocke la lettre du carreau qui se trouve en bas
          stock = m.getTerrain().get(ligne - 1)[colonne + i];
          if (stock == 'x') {
              if (m.getPieceNeutre().getLargeur() == this.getLargeur())
                 b = this.cloneCarreau(m, ligne, colonne, stock,
                        hauteurMax);
              if (b)
                 return b;
          } else {
              for (int j = 0; j < PiecesAJouer.getNbCarreaux(); ++j) {
                 if (stock == p.getPieces()[j].getLettre()) {
                  // on cherche le carreau qui est lié à la lettre
                  // (pièce neutre pas dans le tableau)

                     if(p.getPieces()[j].getLargeur()==this.getLargeur())
                     // on regarde la largeur du carreau
                        b = this.cloneCarreau(m, ligne, colonne, stock,
                               hauteurMax);
                 }
                 if (b)
                     return b;
              }
          }
       }
   }

   if (colonne != 0) { // on regarde à gauche de la pièce
       if (m.getTerrain().get(ligne)[colonne - 1] != Mur.getDefaultCase()) {

          // stocke la + petite valeur entre
          // taille du terrain ou hauteur du carreau
          int hauteurMax = 0;
          if (ligne + getHauteur() > m.getTerrain().size())
              hauteurMax = m.getTerrain().size() - ligne;
```

```
        else
            hauteurMax = getHauteur();

        for (int i = 0; i < hauteurMax; ++i) {

            stock = m.getTerrain().get(ligne + i)[colonne - 1];
            // on stocke la lettre du carreau qui se trouve à gauche

            if (stock == 'x') {
               if (m.getPieceNeutre().getHauteur() == this.getHauteur())
                   b = this.cloneCarreau(m, ligne, colonne, stock, hauteurMax);
               if (b)
                   return b;
            } else {
               for (int j = 0; j < PiecesAJouer.getNbCarreaux(); ++j) {
                   if (stock == p.getPieces()[j].getLettre())
                   // on cherche le carreau qui est lié à la lettre

                      if (p.getPieces()[j].getHauteur() == this.getHauteur())
                      // on regarde la hauteur du carreau
                          b = this.cloneCarreau(m, ligne, colonne,
                                 stock, hauteurMax);
                   if (b)
                      return b;
               }
            }
        }
    }
}

// on regarde à droite de la pièce
if (colonne + getLargeur() - 1 != Mur.getColonne() - 1) {
    if (m.getTerrain().get(ligne)[colonne + getLargeur()]
           != Mur.getDefaultCase()) {

        // stocke la + petite valeur entre
        // taille du terrain ou hauteur du carreau
        int hauteurMax = 0;
        if (ligne + getHauteur() > m.getTerrain().size())
            hauteurMax = m.getTerrain().size() - ligne;
        else
            hauteurMax = getHauteur();
        for (int i = 0; i < hauteurMax; ++i) {
            stock = m.getTerrain().get(ligne + i)[colonne + getLargeur()];
         // on stocke la lettre du carreau qui se trouve à droite
            if (stock == 'x') {
               if (m.getPieceNeutre().getHauteur() == this.getHauteur())
                   b = this.cloneCarreau(m, ligne, colonne, stock, hauteurMax);
               if (b)
                   return b;
            } else {
               for (int j = 0; j < PiecesAJouer.getNbCarreaux(); ++j) {
                   if (stock == p.getPieces()[j].getLettre())
                   // on cherche le carreau qui est lié à la lettre
                      if (p.getPieces()[j].getHauteur() == this.getHauteur())
                      // on regarde la hauteur du carreau

                          b = this.cloneCarreau(m, ligne, colonne,
                                 stock, hauteurMax);
                   if (b)
                      return b;
               }
```

```
            }
         }
      }
   }

   return b;
}

/**
 * Indique si le carreau est un clone d'un autre carreau déjà posé
 *
 * @param m       le mur à carreler
 * @param ligne   la ligne saisie par l'utilisateur
 * @param colonne la colonne saisie par l'utilisateur
 * @param lettre  la lettre qui représente et est attribuée au carreau
 * @param hMax    la hauteur minimale entre le nbr de ligne du mur et la
 *                hauteur de la pièce + la ligne, on le désigne comme hMax
 * @return vrai si le carreau est un clone
 */
public boolean cloneCarreau(Mur m, int ligne, int colonne, char lettre,
      int hMax) {
   assert (ligne >= 0 && colonne >= 0);
   int cptLettreB = 0;
   int cptLettreG = 0;
   int cptLettreD = 0;
   if (ligne != 0) { // pour le bas
      for (int i = 0; i < this.getLargeur(); ++i) {
         if (m.getTerrain().get(ligne - 1)[colonne + i] == lettre)
            cptLettreB++;
         if (cptLettreB == this.getLargeur())
            return true;
      }
   }

   if (colonne != 0) { // on regarde à gauche de la pièce
      for (int i = 0; i < hMax; ++i) {
         if (m.getTerrain().get(ligne + i)[colonne - 1] == lettre)
            cptLettreG++;
         if (cptLettreG == this.getHauteur())
            return true;
      }
   }
   // on regarde à droite de la pièce
   if (colonne + getLargeur() - 1 != Mur.getColonne() - 1) {
      for (int i = 0; i < hMax; ++i) {
         if (m.getTerrain().get(ligne + i)[colonne
               + this.getLargeur()] == lettre)
            cptLettreD++;
         if (cptLettreD == this.getHauteur())
            return true;
      }
   }

   return false;
}

/**
 * Indique si le placement du carreau est valide
 *
 * @param m       le mur à carreler
 * @param ligne   la ligne saisie par l'utilisateur
 * @param colonne la colonne saisie par l'utilisateur
```

```
     * @param pions   un tableau de Carreaux (pièces du jeu)
     * @return vrai si le placement du carreau est valide
     */
    public boolean estValide(Mur m, int ligne, int colonne,
           PiecesAJouer pions) {
       assert (ligne >= 0 && colonne >= 0);
       return  !(this.depasseZone(m, ligne, colonne))
              && (this.estPlacable(m, ligne, colonne))
              && (this.toucheCarreau(m, ligne, colonne))
              && (this.baseStable(m, ligne, colonne))
              && !(this.aCarreauAdjacent(m, ligne, colonne, pions));
    }

}
```

Mur.java

```
/**
 * @file Mur.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Type de donnée représentant la zone de jeu
 */

package composantDeJeu;

import java.util.*;

/** Type de donnée représentant le mur où se déroulera le jeu */
public class Mur {
    /** Nombre de colonnes */
    private static final int COLONNE = 5;
    /** Case par défaut (espace) */
    private static final char DEFAULT_CASE = ' ';
    /** Nombre de positions possibles pour la pièce neutre */
    private static final int nbCasNeutre = 4;
    /** Tableau dynamique contenant tableau de caractères représentant le mur */
    private ArrayList<char[]> terrain;
    /** Carreau neutre */
    private Carreau pieceNeutre;

    /**
     * Constructeur Initialise un mur avec une ligne vide et place le carreau
     * neutre
     *
     */
    public Mur() {
       terrain = new ArrayList<>();
       char[] ligne = new char[getColonne()];
       Arrays.fill(ligne, getDefaultCase());
       terrain.add(ligne);
       placerCaroNeutre();
    }

    /** Retourne le tableau dynamique terrain */
    public ArrayList<char[]> getTerrain() {
       return new ArrayList<char[]>(terrain);
    }

    /** Retourne le carreau neutre */
    public Carreau getPieceNeutre() {
       assert (pieceNeutre != null);
```

```
   Carreau copieNeutre = new Carreau(pieceNeutre);
   return copieNeutre;
}

/** Retourne la constante de la case par défaut (espace) */
public static char getDefaultCase() {
   return DEFAULT_CASE;
}

/** Retourne la constante nombre de colonnes du mur */
public static int getColonne() {
   return COLONNE;
}

/** Place aléatoirement le carreau neutre */
public void placerCaroNeutre() {
   int casMax = nbCasNeutre;
   Random r = new Random();
   int casAléatoire = r.nextInt(casMax);
   Carreau neutreHori = new Carreau(3, 1, 'x');
   Carreau neutreVerti = new Carreau(1, 3, 'x');
   switch (casAléatoire) {
   case 0:
       placerCarreau(neutreHori, 0, 0);
       pieceNeutre = neutreHori;
       break;
   case 1:
       placerCarreau(neutreVerti, 0, 4);
       pieceNeutre = neutreVerti;
       break;
   case 2:
       placerCarreau(neutreVerti, 0, 0);
       pieceNeutre = neutreVerti;
       break;
   case 3:
       placerCarreau(neutreHori, 0, 2);
       pieceNeutre = neutreHori;
       break;
   }
}

/**
 * Place un carreau valide sur le mur
 *
 * @param c       le carreau que l'on veut placer
 * @param lettre  la lettre correspondant au carreau
 * @param ligne   la ligne où l'on veut placer le carreau
 * @param colonne la colonne où l'on veut placer le carreau
 */
public void placerCarreau(Carreau c, int ligne, int colonne) {
   assert (ligne >= 0 && colonne >= 0);
   assert (!c.getUtilisé());
   assert (!c.depasseZone(this, ligne, colonne)); // Condition 3
   if (c.getLettre() != 'x') // Ne le fais pas pour le carreau neutre
       assert (c.toucheCarreau(this, ligne, colonne)); // Condition 4
   assert (c.baseStable(this, ligne, colonne)); // Condition 5
   assert (c.estPlacable(this, ligne, colonne));
   // Ne supprime pas un carreau déja placé
   if (this.terrain.size() <= ligne + c.getHauteur()) {
       int j = ligne + 1 + c.getHauteur();
       for (int i = this.terrain.size(); i < j; ++i) {
          char[] nvLigne = new char[getColonne()];
```

```
                Arrays.fill(nvLigne, getDefaultCase());
                this.terrain.add(nvLigne);
            }
        }

        for (int i = 0; i < c.getHauteur(); ++i) {
            for (int j = 0; j < c.getLargeur(); ++j) {
                this.terrain.get(ligne + i)[colonne + j] = c.getLettre();
            }
        }
        c.setUtilisé(true);

    }

    /**
     * Permet d'afficher le mur
     */
    public String toString() {
        String result = "";
        char c;
        final int DIZAINES = 10;
        for (int i = terrain.size() - 1; i >= 0; --i) {
            if (i + 1 < DIZAINES)
            // en dessous de 10, on rajoute un espace pour l'affichage
                result += getDefaultCase();
            result = result + (i + 1) + getDefaultCase();
            for (int j = 0; j < getColonne(); ++j) {
                c = terrain.get(i)[j];
                result = result + c + getDefaultCase();
            }
            result += System.lineSeparator();
        }

        result = result + getDefaultCase() + getDefaultCase()
                + getDefaultCase();
        for (int i = 1; i <= getColonne(); ++i) {
            result = result + i + getDefaultCase();
        }
        return result;
    }

}
```

Carte.java

```
/**
 * @file Carte.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Type de donnée représentant une carte de jeu
 */
package composantDeJeu;

/** Type de données représentant une carte */
public class Carte {
        /** Contenu d'une carte */
        private ContenuCarte CONTENU;

        /**
         * Constructeur
         *
         * @param c le contenu de la carte (obligatoire)
```

```
 */
public Carte(ContenuCarte c) {
	this.CONTENU = c;
}

/**
 * Renvoie en chaîne de caractère le contenu de la carte
 * @return le contenu d'une carte
 */
public String getContenuCarte() {
	String s = "";
	switch(this.CONTENU) {
	case ROUGE:
		s = "Rouge";
		break;
	case BLEU:
		s = "Bleu";
		break;
	case TAILLE1:
		s = "Taille 1";
		break;
	case TAILLE2:
		s = "Taille 2";
		break;
	case TAILLE3:
		s = "Taille 3";
		break;
	}
	return s;
}

/**
 * Indique si le carreau respecte la condition de la carte piochée
 * @param c le carreau
 * @return vrai si le carreau respecte la condition
 */
public boolean conditionCarte (Carreau c) {
	boolean b = false;
	switch(this.CONTENU) {
	case ROUGE :
		b = c.estRouge();
	break;
	case BLEU :
		b = c.estBleu();
	break;
	case TAILLE1 :
		b = c.estTaille1();
	break;
	case TAILLE2 :
		b = c.estTaille2();
	break;
	case TAILLE3 :
		b = c.estTaille3();
	break;
	}
	return b;
}

/**
 * Renvoie une chaîne de caractère représentant le contenu d'une carte
 */
public String toString() {
```

```
            return getContenuCarte();
        }
}
```

PaquetDeCartes.java

```
/**
 * @file PaquetDeCartes.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Type de donnée représentant un paquet de cartes
 */

package composantDeJeu;

import java.util.*;

/** Type de donnée représentant le paquet de cartes */
public class PaquetDeCartes {
    /** Le nombre de cartes maximal */
    private static final int MAX_CARTES = 33;
    /** Le nombre de cartes indiquant une taille */
    private static final int NB_CARTES_TAILLE = 5;
    /** Le nombre de cartes indiquant une couleur */
    private static final int NB_CARTES_COULEUR = 9;
    /** Tableau dynamique contenant le paquet de cartes */
    private ArrayList<Carte> paquet;

    /**
     * Constructeur Crée un paquet de cartes contenant des cartes de couleur et
     * des cartes de taille
     * Si aucune paramètre n'est rentré un paquet vide sera créé
     *
     * @param nbCouleur Le nombre de cartes indiquant une couleur (optionnel)
     * @param nbTaille  Le nombre de cartes indiquant une taille (optionnel)
     */
    public PaquetDeCartes(int nbCouleur, int nbTaille) {
       this.paquet = new ArrayList<>();
       // Ajout des 18 cartes Couleur
       for (ContenuCarte c : ContenuCarte.values()) {
           if (c == ContenuCarte.ROUGE || c == ContenuCarte.BLEU) {
              for (int i = 0; i < nbCouleur; ++i) {
                  paquet.add(new Carte(c));
              }
           }
       }
       // Ajout des 15 cartes Taille
       for (int i = 0; i < nbTaille; ++i) {
           this.paquet.add(new Carte(ContenuCarte.TAILLE1));
           this.paquet.add(new Carte(ContenuCarte.TAILLE2));
           this.paquet.add(new Carte(ContenuCarte.TAILLE3));
       }
    }

    public PaquetDeCartes() { // Pour le paquet de tas écarté
       this(0,0);
    }

    /** Retourne une copie du paquet de cartes */
    public ArrayList<Carte> getPaquet() {
       return new ArrayList<Carte>(paquet);
```

```
    }

    /** Retourne le nombre maximum de cartes par paquet */
    public static int getMaxCartes() {
       return MAX_CARTES;
    }

    /** Retourne le nombre max de cartes Taille dans un paquet */
    public static int getNbCartesTaille() {
       return NB_CARTES_TAILLE;
    }

    /** Retourne le nombre max de cartes Couleur dans un paquet */
    public static int getNbCartesCouleur() {
       return NB_CARTES_COULEUR;
    }

    /** Mélange le paquet de cartes */
    public void mélanger() {
       Collections.shuffle(this.paquet);
    }

    /**
     * Pioche une carte
     *
     * @return sommet la carte au sommet du paquet
     */
    public Carte piocher() {
       assert (this.paquet.size() > 0);
       // carte au sommet
       Carte sommet = this.paquet.get(this.paquet.size() - 1);
       this.retirer();
       return sommet;
    }

    /** Retire une carte */
    public void retirer() {
       this.paquet.remove(this.paquet.size() - 1);
    }

    /** Ajoute une carte dans le paquet */
    public void ajouter(Carte c) {
       this.paquet.add(c);
    }

    /**
     * Indique si le paquet n'a plus de cartes
     *
     * @return vrai si le paquet n'a plus de cartes
     */
    public boolean plusDeCartes() {
       return this.paquet.size() == 0;
    }

}
```

PiecesAJouer.java

```
/**
 * @file PiecesAJouer.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
```

```
 * @brief Type de donnée représentant les pièces de jeu pour une partie
 */

package composantDeJeu;

/** Type de données représentant les pièces à jouer pour une partie */
public class PiecesAJouer {
    /** Représente le nombre total de carreaux dans une partie */
    private static final int NB_CARREAUX = 18;
    /** Tableau statique de carreaux */
    private static Carreau[] pieces;
    /** nombre de carreleurs */
    private final static int NB_JOUEURS = 2;

    /**
     * Constructeur
     * Initialise et trie le tableau statique à la taille NB_CARREAUX
     */
    public PiecesAJouer() {
       pieces = new Carreau[getNbCarreaux()];
       int cpt = 0;
       for (Couleur c : Couleur.values()) {
           pieces[0 + cpt * PiecesAJouer.NB_CARREAUX
                  / NB_JOUEURS] = new Carreau(3, 3, 'i', c);
           pieces[1 + cpt * PiecesAJouer.NB_CARREAUX
                  / NB_JOUEURS] = new Carreau(3, 2, 'h', c);
           pieces[2 + cpt * PiecesAJouer.NB_CARREAUX
                  / NB_JOUEURS] = new Carreau(2, 3, 'g', c);
           pieces[3 + cpt * PiecesAJouer.NB_CARREAUX
                  / NB_JOUEURS] = new Carreau(3, 1, 'f', c);
           pieces[4 + cpt * PiecesAJouer.NB_CARREAUX
                  / NB_JOUEURS] = new Carreau(1, 3, 'e', c);
           pieces[5 + cpt * PiecesAJouer.NB_CARREAUX
                  / NB_JOUEURS] = new Carreau(2, 2, 'd', c);
           pieces[6 + cpt * PiecesAJouer.NB_CARREAUX
                  / NB_JOUEURS] = new Carreau(2, 1, 'c', c);
           pieces[7 + cpt * PiecesAJouer.NB_CARREAUX
                  / NB_JOUEURS] = new Carreau(1, 2, 'b', c);
           pieces[8 + cpt * PiecesAJouer.NB_CARREAUX
                  / NB_JOUEURS] = new Carreau(1, 1, 'a', c);
           ++cpt;
       }
       this.trier();
    }

    /** Retourne le tableau statique de carreaux */
    public Carreau[] getPieces() {
       Carreau[] copie = pieces.clone();
       return copie;
    }

    /** Retourne le nombre de carreaux jouables dans une partie */
    public static int getNbCarreaux() {
       return NB_CARREAUX;
    }

    /** Modifie l'attribut pieces avec le tableau de Carreau trier */
    public static void setPieces(Carreau[] trier) {
       Carreau[] copie = trier;
       pieces = copie;
    }
```

```java
/**
 * Indique s'il y a encore des carreaux non utilisés
 *
 * @return vrai si tous les carreaux ont été posés
 */
public boolean plusDeCarreau() {
   for (int i = 0; i < PiecesAJouer.getNbCarreaux(); ++i) {
       if (this.getPieces()[i].getUtilisé() == false)
          return false;
   }
   return true;
}

/**
 * Renvoie tous les carreaux non utilisées et qui respecte la carte pioche
 *
 * @param pioche la carte piochée
 * @return une chaîne de caractères représentant les carreaux disponibles et
 *         désignées par la carte pioche
 */
public String toString(Carte pioche) {
   String s = "";
   int maxH = 0; // Hauteur maximale des pièces non utilisées
   for (int i = 0; i < PiecesAJouer.getNbCarreaux(); i++) {
       if (!this.getPieces()[i].getUtilisé()
              && pioche.conditionCarte(this.getPieces()[i])) {
          if (maxH < this.getPieces()[i].getHauteur())
              maxH = this.getPieces()[i].getHauteur();
       }
   }

   while (maxH != 0) {
       for (int i = 0; i < PiecesAJouer.getNbCarreaux(); i++) {
          if (!this.getPieces()[i].getUtilisé()) {
           // condition de la carte
              if (pioche.conditionCarte(this.getPieces()[i])) {
                 if (this.getPieces()[i].getHauteur() >= maxH) {
                     for (int l = 0; l < this.getPieces()[i].getLargeur(); ++l) {
                        s += this.getPieces()[i].getLettre();
                     }
                 } else {
                     for (int l = 0; l < this.getPieces()[i].getLargeur(); ++l) {
                        s += " ";
                     }
                 }
                 s += " ";
              }
          }
       }
       s += System.lineSeparator();
       --maxH;
   }
   return s;
}

/**
 * Trie les pièces à jouer, par ordre croissante et les minuscules avant les
 * majuscules
 */

public void trier() {
   Carreau[] piecesCopie = this.getPieces();
```

```
        char letter = 'a';
        Carreau stock = null;
        for (int k = 0; k < NB_JOUEURS; ++k) {
            for (int j = 0; j < NB_CARREAUX / NB_JOUEURS; ++letter, ++j) {
               for (int i = 0; i < NB_CARREAUX; ++i) {
                   if (piecesCopie[i].getLettre() == letter) {
                      stock = piecesCopie[j + k * NB_CARREAUX / NB_JOUEURS];

                      piecesCopie[j + k * NB_CARREAUX/ NB_JOUEURS]
                      = piecesCopie[i];

                      piecesCopie[i] = stock;
                   }
               }
            }
            letter = 'A';
        }

        PiecesAJouer.setPieces(piecesCopie);
    }
}
```

Appli.java

```
/**
 * @file Appli.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 09/03/2020
 * @brief Programme principal
 */

package jeu;

import java.util.*;
import composantDeJeu.Carreau;
import composantDeJeu.Carte;
import composantDeJeu.Mur;
import composantDeJeu.PaquetDeCartes;
import composantDeJeu.PiecesAJouer;

public class Appli {

    /** booléen contrôlant la boucle de jeu */
    private static boolean boucleDeJeu = false;
    /** points gagnées par niveaux complétés */
    public final static int PTS_NIVEAU = 5;
    /** nombre de carreleurs */
    public final static int NB_JOUEURS = 2;
    /** saisie à entrer pour arrêter la partie */
    public final static String STOP = "stop";
    /** saisie à entrer pour écarter une carte */
    public final static String NEXT = "next";

    public static void main(String[] args) {
       PaquetDeCartes p = new PaquetDeCartes
       (PaquetDeCartes.getNbCartesCouleur(),PaquetDeCartes.getNbCartesTaille());

       PaquetDeCartes tasEcarte = new PaquetDeCartes();
       p.mélanger();
       PiecesAJouer pion = new PiecesAJouer();
```

```
    Mur m = new Mur();

    Scanner saisie = new Scanner(System.in);

    while (!boucleDeJeu) {
        String zone = m.toString();
        /* Etape 1 : Affichage de la zone à carreler */
        System.out.println(zone);

        /* Etape 2 : Tirage de carte */

        Carte piocher = p.piocher();
        piocher.toString();
        System.out.println(piocher);

        /* Etape 3 : Affichage des carreaux disponibles + convenables */
        String pions = pion.toString(piocher);
        System.out.println(pions);

        if (pions.isEmpty()) {
           tasEcarte.ajouter(piocher);
           System.out.println
           ("La carte « " + piocher + " » a été écarté. "
           + "Aucun carreau ne satisfait cette condition.");
        }

        /* Etape 4 : Attendre la saisie du choix du carreleur */
        if (!pions.isEmpty()) {
           boolean b = false;
           // indique la validité de la commande saisie par l'utilisateur
           while (!b) {
               try {
                  b = saisie(m, saisie, pion, piocher, tasEcarte);
               } catch (Exception e) {
                  System.err.println(e.getMessage());
               }
           }
        }

        if (!boucleDeJeu)
           boucleDeJeu = finDeJeu(p, pion);

        /* Fin Etape 4 */
    }

    /* Fin while du jeu */
    String resultat = "";
    resultat = finDePartie(m, pion, tasEcarte);
    System.out.println(resultat);
    saisie.close();
}

/**
 * Lance la saisie utilisateur et son traitement
 *
 * @param m         tableau de chaîne de caractères saisies par
 *                  l'utilisateur
 * @param scan      entrée de l'utilisateur
 * @param pions     le tableau statique des carreaux de la partie
 * @param piocher   la carte piochée
 * @param tasEcarte le paquet de cartes écartées
```

```
 * @return vrai si la commande est valide
 * @throws Exception si la commande est incorrecte
 */
public static boolean saisie(Mur m, Scanner scan, PiecesAJouer pions,
       Carte piocher, PaquetDeCartes tasEcarte) throws Exception {

   String saisie = scan.nextLine();
   boolean b = false;

   if ((saisie.trim().length() == 0) || saisie.isEmpty())
    // Vérifie si l'utilisateur a entré des espaces ou un saut de ligne

       return b;

   if (saisie.equals(STOP)) {
       boucleDeJeu = true;
       b = true;
   } else if (saisie.equals(NEXT)) {
       tasEcarte.ajouter(piocher);
       b = true;


   } else {
       Scanner sc = new Scanner(saisie);
       char lettre = sc.next().charAt(0);

       if (!sc.hasNextInt()) {
          sc.close();
          throw new Exception(
                 "La commande n'existe pas. Veuillez réessayer.");
       }
       int ligne = sc.nextInt() - 1;

       if (!sc.hasNextInt()) {
          sc.close();
          throw new Exception(
                 "La commande n'existe pas. Veuillez réessayer.");
       }
       int colonne = sc.nextInt() - 1;

       if (sc.hasNext()) {
          sc.close();
          throw new Exception(
                 "La commande n'existe pas. Veuillez réessayer.");
       }

       if (!(((lettre >= 'a' || lettre >= 'A') &&
           (lettre <= 'i' || lettre <= 'I')))) {
          sc.close();
          throw new Exception(
                 "La lettre ne correspond à aucun carreau. Veuillez réessayer.");
       }

       if (ligne < 0 || ligne > m.getTerrain().size() || colonne < 0
           || colonne >= Mur.getColonne()) {
          sc.close();
          throw new Exception(
                 "Les coordonnées sont invalides. Veuillez réessayer.");
       }

       for (int j = 0; j < PiecesAJouer.getNbCarreaux(); j++) {
          if (pions.getPieces()[j].getLettre() == lettre) {
```

```
                Carreau c = pions.getPieces()[j];
                if (!(piocher.conditionCarte(c)) || c.getUtilisé()) {
                   sc.close();
                   throw new Exception(
                   "Le carreau ne fait pas partie des carreaux proposés. "
                   + "Veuillez réessayer.");
                }

                if (!(c.estValide(m, ligne, colonne, pions))) {
                   sc.close();
                   throw new Exception(
                          "Le placement du carreau est invalide.
                          + "Veuillez réessayer.");
                }

                m.placerCarreau(c, ligne, colonne);
                b = true;
             }
          }

          sc.close();
      }
      return b;
   }

   /**
    * Indique si une partie est terminée
    *
    * @param p
    * @param pieces
    * @return vrai si le paquet de cartes est vide ou s'il n'y a plus de pièces
    *              à jouer
    */
   public static boolean finDeJeu(PaquetDeCartes p, PiecesAJouer pieces) {
      return p.plusDeCartes() || pieces.plusDeCarreau();
   }

   /**
    * Crée une chaine de caractères contenant le score et son détail
    *
    * @param m      le mur, zone de jeu
    * @param pieces les pièces à jouer
    * @param p      le tas écarté
    * @return phrase, indiquant le score , le nombre de niveaux complétées et
    *                 le nombre de cartes écartées
    */
   public static String finDePartie(Mur m, PiecesAJouer pieces,
          PaquetDeCartes p) {
      String phrase = "";
      int score = 0;
      /* 1) Calculer le nombre de niveaux complétés */
      int cpt = 0;
      int nbNiveauxComplets = 0;
      for (int i = 0; i < m.getTerrain().size(); ++i) {
          for (int j = 0; j < Mur.getColonne(); ++j) {
             if (m.getTerrain().get(i)[j] != Mur.getDefaultCase())
                 ++cpt;
          }
          if (cpt == Mur.getColonne())
             ++nbNiveauxComplets;
          cpt = 0;
      }
```

```
        /* 2) Compter le nombre de carreaux non posés */
        int nbCarreaux = 0;
        for (int i = 0; i < PiecesAJouer.getNbCarreaux(); ++i) {
            if (!pieces.getPieces()[i].getUtilisé())
               ++nbCarreaux;
        }

        /* 3) Compter le nombre de cartes dans le tas écarté */
        score = nbNiveauxComplets * PTS_NIVEAU - (nbCarreaux + p.getPaquet().size());
        phrase = score + " points (" + nbNiveauxComplets + " niveaux complets, "
               + nbCarreaux + " carreaux non posés, "
               + p.getPaquet().size() + " cartes écartées)";

        return phrase;
    }
}
```

BILAN DE PROJET

RETOUR D'EXPERIENCE

Les difficultés rencontrées

Pour ce projet, le début a été le plus difficile. En particulier notre classe Mur représentant le terrain de jeu, nous sommes partis dans des choix assez complexes : `ArrayList<ArrayList<String>>` ou `ArrayList<String>`, des structures de données assez mauvaises tout autant pour le stockage des données que pour l'utilisation des données. Nous avons finalement opté pour un **`ArrayList<char[]>`** en suivant la proposition que nous a conseillé notre professeur. Cette structure est plus facile d'utilisation et permet une structure des données claire. L'ArrayList afin de rajouter des lignes au tableau, et le tableau de char[]de taille 5 qui correspond aux colonnes.

L'IDE Eclipse a été également l'une de nos difficultés durant ce projet, nous n'avons pas eu d'apprentissage particulier du logiciel, nous avons ainsi appris sur le tas sans connaître toutes les fonctionnalités et comment s'en servir réellement : Problèmes de compilation, de partage, incompréhension des erreurs de programmation...

Une autre difficulté rencontrée était la rencontre constante de l'erreur « *Index out of bounds* » lorsque nous modifions ou récupérions des valeurs du Mur incorrects ou inexistantes. Il fallait adapter notre code pour éviter à tout prix ces erreurs. Plus généralement, les méthodes qui permettent de respecter les contraintes quant au placement du carreau dans le mur (**estPlacable()**, **aCarreauAdjacent()**, **cloneCarreau()**) ont été conçues difficilement, nous montrant nos erreurs de programmation. La méthode `saisie()` a été l'une des plus longues à mettre en place, nous sommes partis dans beaucoup de directions différentes et beaucoup d'essais avant d'arriver à notre résultat.

Ce qui est réussi

Premièrement, nous avons déjà réussi à faire fonctionner correctement notre jeu. Nous sommes satisfaits d'avoir réussi à confectionner pour la première fois, un mini-jeu, par nos propres moyens.
Deuxièmement, nous avons mis au clair rapidement quelles classes réalisées. Nous pensons que nous n'avons pas énormément de classes mais qu'elles sont toutes essentielles. Nous avons bien fait attention à ne pas stocker des valeurs que nous pouvions calculer, et également pris des précautions quant à l'encapsulation au niveau des getters et des setters.
Le fait d'utiliser les exceptions dans notre méthode **saisie()** a été une bonne expérience pour apprendre à les manier, de plus cela fonctionne correctement.
On a fait de notre mieux pour éviter d'utiliser des Syso dans le projet, mis à part dans le main, afin d'effectuer des saisies au niveau le plus haut.

VERS DES AMELIORATIONS

En revanche, nous ne sommes pas totalement fiers de notre code. Nous avons du mal à éviter d'écrire trop de caractères ou chiffres magiques, surtout concernant les lettres. L'initialisation du tableau PiecesAJouer est aussi une des déceptions que nous n'avons pas su corriger (création d'un tableau de char pour y mettre les lettres ?).

De plus, en ce qui concerne la méthode **aCarreauAdjacent()** et **cloneCarreau()**, nous avons l'impression d'être redondant et d'avoir une méthode trop longue, qu'elles peuvent être optimiser sans réellement savoir comment.

Les commentaires peuvent sûrement être améliorés afin d'être plus claires et précises pour quiconque souhaitant lire ou modifier notre code.