Rapport de projet

Bases de la programmation orientée objet

Le Duel

Antoine BANHA
Groupe 102

Logan TANN
Groupe 105

DUT 1ère Année

Université de Paris

Table des matières

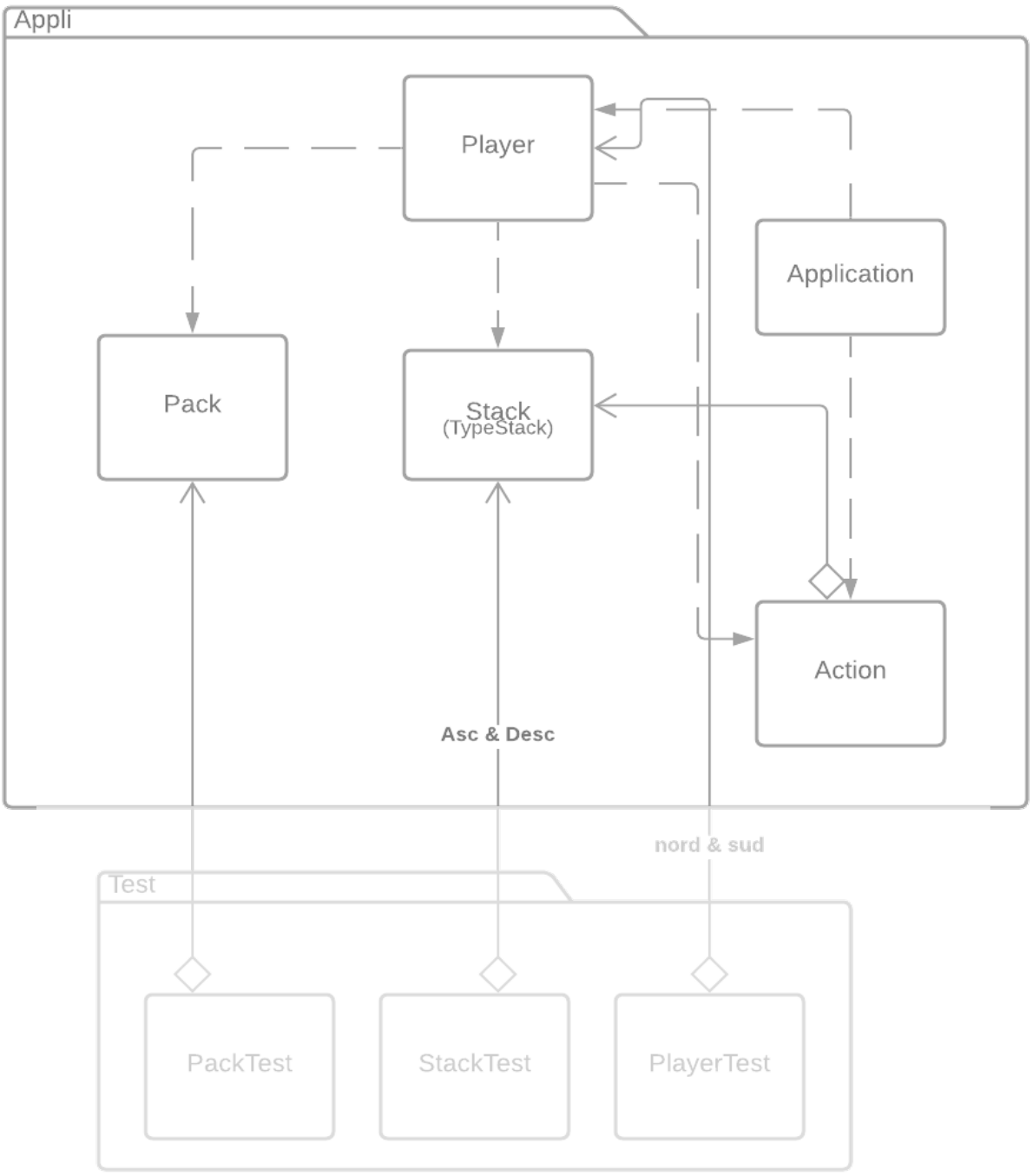
Introduction du projet

Hasbor, une marque de jeux de société/jeux de plateaux, est en pleine crise financière due à la Covid-19. Leurs ventes chutent considérablement et leur chiffre d'affaires de décembre n'a pas été à la hauteur, en particulier sur leur produit phare : Le Duel.

Le Duel est un jeu de cartes où deux joueurs sont munis d'une pioche et deux piles de défausse (nommées piles ascendantes et descendantes). Le joueur peut poser des cartes parmi les deux piles, qui sont soumises à des conditions de poses distinctes, tant chez lui (ce qui le désavantage en limitant ses possibilités d'action dans les futurs coups) que chez l'adversaire (cela offre l'effet contraire à l'adversaire).
Pour pouvoir gagner, il faut que le joueur vide son paquet de cartes, ou que le joueur adverse perde la partie (lorsqu'il ne peut pas jouer plus de deux cartes). Il y a donc une multitude de stratégies possibles, car avantager l'adversaire peut aider le joueur à défausser plus rapidement ses cartes.

Après ce désastre financier et des mois de R&D, ils ont décidé de miser sur une version innovante de ce jeu : Le Duel version numérique ! Le principe est le même mais cette fois-ci, vous jouerez directement sur ordinateur grâce à une interface en ligne de commande, simple, léger et très rapide à programmer, non ? *Hasbor* a également mentionné que le produit doit être fait en Java (car *3 Billions devices runs java* depuis 30 ans) et que celui-ci doit fournir une documentation ainsi que des tests unitaires. Le projet doit être structuré de façon à rendre la vie plus « facile » aux développeurs qui modifieront le programme par la suite. Si la version shell est un succès, *Hasbor* investira sur une application mobile et peut-être un Swing UI pour la version PC !

Diagramme de classes



Antoine BANHA – Logan TANN

Bilan du projet

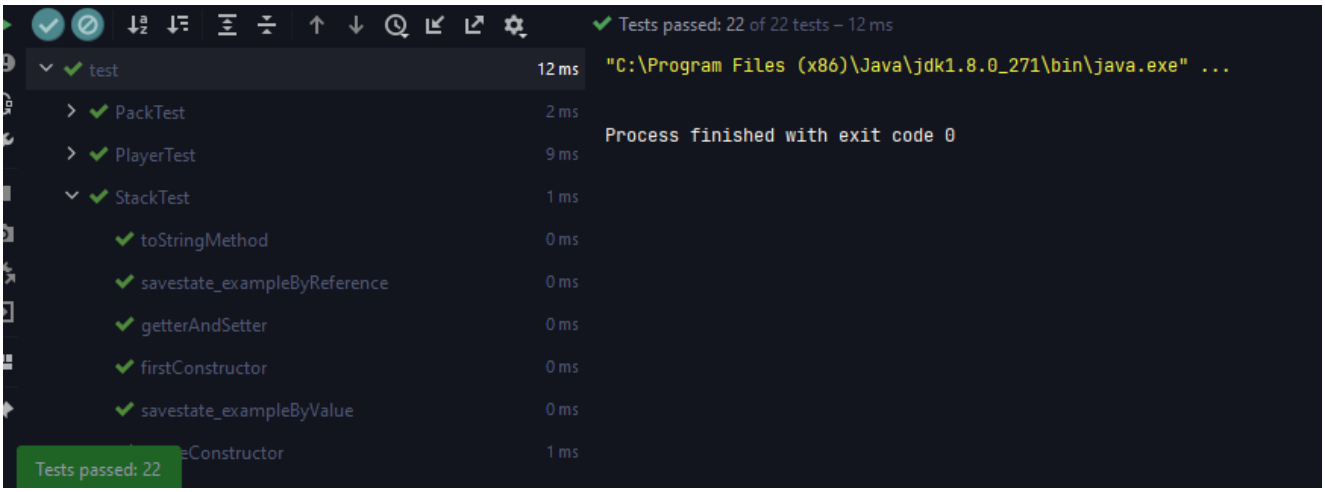
Le module BPO nous a permis de découvrir un langage que nous ne connaissions pas, et pourtant très utilisé dans les entreprises : le Java. Employé dans de nombreuses applications *desktop* multi-plateformes, il est également répandu dans nos chères applications Android ou dans les systèmes embarqués de notre voiture. Nous pratiquions le JavaScript (langage interprété avec des types autonomes et syntaxe assez hasardeuse) ou le C++ (langages compilé avec des types définis où les erreurs de votre msvc ne cesseront de vous hanter), mais le java est un superbe mix avec ces deux opposés et offre donc son lot de potentiel. De plus, ce projet nous a fait découvrir des aspects que nous n'avions jamais pris le temps de pratiquer sur nos projets personnels, à savoir les tests unitaires avec Junit4.

Nous avons fait le choix de séparer les grands axes d'une partie de Duel en différentes classes et utilitaires : Player, Action, Stack et Pack. Ces différentes classes sont instanciées au bon moment et permettent une gestion de partie bien plus simple grâce au concept de méthodes d'instances. Nous avons également porté une attention particulière à la visibilité de nos méthodes et à la qualité de notre code.
Nous ne voyons pas de réelles améliorations possibles, si ce n'est que la création d'une IA ou de parties en réseau permettant de procurer une réelle expérience de jeu. Il est également possible de remarquer l'ajout d'un mode *verbose* pour réaliser nos tests, mais surtout connaître la cause d'une erreur. Cela n'a pas été demandé (*il faut dire que le programme se montre timide envers l'utilisateur et nous avions besoin d'un peu plus d'interactions dans ce monde d'isolation*), mais nous a été primordial pour localiser plus facilement des erreurs de logique.

Pour finir, nous avons utilisé les habituels GitHub pour le VCS, Discord pour la communication et LibreOffice pour le rapport. Nous avons également découvert les environnements Eclipse et IntelliJ Idea qui sont des IDEs riches en fonctionnalités.

Point sur les tests unitaires

Nous avons fait passer à notre programme une grande batterie de tests afin de nous assurer que le code respecte bien nos exigences. Celui-ci passe la totalité des tests, qui vous sont donnés à la page suivante.



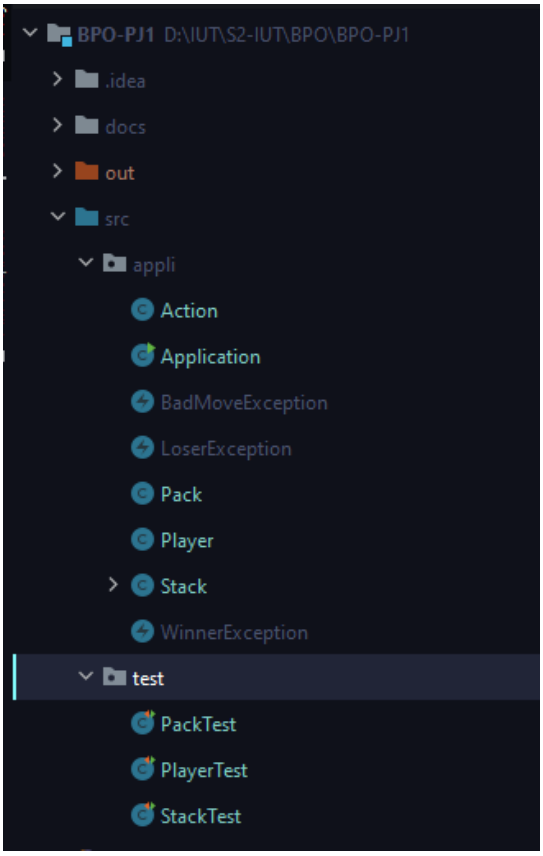
Nous avons accordé une très grande importance à la lisibilité du code de nos tests. Plutôt que de comparer certains tableaux par valeurs, une comparaison du toString() des tableaux avec la chaine de caractère attendue permet de comprendre plus facilement les actions des méthodes.

```
99          @Test
100         public void testPickCards() {
101                 // (init)
102                 setPack(nord, new int[] {7, 8, 9});
103                 nord.setHand(hand2);
104                 assertEquals("[7, 8, 9]", nord.getPack().getPack().toString());
105                 assertEquals("[11, 5]", nord.getHand().toString());
106
107                 assertEquals(2, nord.pickCards(2));
108                 assertEquals("[9]", nord.getPack().getPack().toString());
109                 assertEquals("[11, 5, 7, 8]", nord.getHand().toString());
110
111                 assertEquals(1, nord.pickCards(2));
112                 assertEquals("[]", nord.getPack().getPack().toString());
113                 assertEquals("[11, 5, 7, 8, 9]", nord.getHand().toString());
114         }
115
```

Pour certains tests de méthodes d'affichage (toString etc.), nous avons utilisé des expressions régulières. Même si sur le terrain, le test vérifie uniquement sur des valeurs initiales, le pattern peut être réutilisé pour n'importe quelles valeurs de cartes.

Annexes

Tests unitaires & code source du projet



Structure du projet sur IntelliJ

```java
/**
* PackTest.java
* Classe permettant les tests unitaires des Packs.
* @author Antoine <antoine@jiveoff.fr> & Logan Tann
* @project Projet-BPO
*/

package test;
import appli.Pack;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;

import java.util.LinkedList;

public class PackTest {

private Pack packDefault;

    @Before
    public void setUp() throws Exception {
        packDefault = new Pack();
    }

    @Test
    public void testConstants() {
        assertEquals(1, Pack.DEFAULT_FIRST_CARD);
        assertEquals(60, Pack.DEFAULT_LAST_CARD);
    }

    /**
     * @see testConstants() pour justif valeurs 1 et 60
     */
    @Test
    public void constructors() {
        Pack packLitterals = new Pack(1, 60);

        // arrays defined by the constructor ?
        assertNotNull(packDefault.getPack());
        assertNotNull(packLitterals.getPack());

        // array sorted ?
        LinkedList<Integer> packContent = packLitterals.getPack();
        for (int i = 0; i < 60; i++) {
             assertTrue(packContent.get(i).equals(i + 1));
        }

        // default constructor should create a Pack(1, 60);
        assertArrayEquals(
            packLitterals.getPack().toArray(new Integer[0]),
            packDefault.getPack().toArray(new Integer[0])
        );
    }
```

```java
/**
 * @see constructors() "array sorted ?"
 */
@Test
public void pickCardMethods_and_getPackLength() {
   // 60 cards by default
   assertEquals(60, packDefault.getPackLength());

   // when a card is picked, it returns its value
   assertEquals(3, packDefault.pickCard(2));
   assertEquals(4, packDefault.pickCard(2));
   // picked → the value returned is also removed in the pack.
   assertEquals(60 - 2, packDefault.getPackLength());

   // pickFirstCard = pickCard(0)
   assertEquals(1, packDefault.pickFirstCard());

   // pickLastCard = pickCard( this.getPackLength() - 1)
   assertEquals(60, packDefault.pickLastCard());

   // current pack state : [2 5 6 7 ... 58 59]
}

@Test
public void cloneConstructor() {
    Pack packClone = new Pack(packDefault);
    assertEquals(packDefault.getPack(), packClone.getPack());

    packDefault.pickCard(0);
    packDefault.pickCard(5);
    packDefault.pickCard(0);
    assertNotEquals(packDefault.getPack(), packClone.getPack());

    // can be used for savestate system
    packDefault = packClone;
    assertEquals(packDefault.getPack(), packClone.getPack());
}

/**
 * @see cloneConstructor()
 */
@Test
public void shuffle_and_exists() {
    Pack sorted = new Pack();
   Pack toShuffle = new Pack();
    assertEquals(sorted.getPack(), toShuffle.getPack());

    toShuffle.shuffle();

    assertNotEquals(sorted.getPack(), toShuffle.getPack());
    assertEquals(sorted.getPackLength(), toShuffle.getPackLength());
    for (int card: sorted.getPack()) {
         assertTrue(toShuffle.exists(card));
    }
}
```

```java
    @Test
    public void isEmpty() {
       Pack verySmallPack = new Pack(1, 2); // content : [1 2]
       verySmallPack.pickFirstCard(); // clears it
       verySmallPack.pickLastCard();

       assertTrue(verySmallPack.isEmpty());
       assertEquals(-1, verySmallPack.pickLastCard());
    }
}
```

```java
/**

* PlayerTest.java

* Classe permettant les tests unitaires des Players.
* @author Antoine <antoine@jiveoff.fr> & Logan Tann
* @project Projet-BPO
*/

package test;

import static org.junit.Assert.*;

import java.util.ArrayList;

import org.junit.Before;
import org.junit.Test;

import appli.Player;
import appli.Stack;

public class PlayerTest {

private Player nord, sud;
ArrayList<Integer> hand6, hand2;

@Before
public void setUp() throws Exception {
       nord = new Player("NORD");
       sud = new Player("SUD");

       hand6 = new ArrayList<>();
       hand6.add(4); hand6.add(2); hand6.add(5);
       hand6.add(3); hand6.add(60); hand6.add(1);

       hand2 = new ArrayList<>();
       hand2.add(11); hand2.add(5);
}
```

```java
@Test
public void testPlayer() {
	assertEquals(6, nord.getHand().size());

	assertEquals("NORD", nord.getName());
	assertEquals("SUD", sud.getName());

	Stack asc = nord.getStack(Stack.TypeStack.ASC);
	Stack desc = nord.getStack(Stack.TypeStack.DESC);
	assertEquals(1, asc.getCardOnTop());
	assertEquals(60, desc.getCardOnTop());
}

@Test
public void testToString() {
	String pattern = "(NORD|SUD ) \\^\\[\\d\\d\\] v\\[\\d\\d\\] \\(m\\dp\\d{1,2}+\\)";

	assertTrue(nord.toString().matches(pattern));
	assertTrue(sud.toString().matches(pattern));
}

@Test
public void testHand_toString() {
	String pattern = "cartes \\w+ \\{ (\\d\\d ){2,6}+\\}";
	assertTrue(nord.hand_toString().matches(pattern));
	assertTrue(sud.hand_toString().matches(pattern));
}

@Test
public void testSortHand() {
	// (init)
	nord.setHand(hand6);
	assertEquals("[4, 2, 5, 3, 60, 1]", nord.getHand().toString());

	// test sorted
	nord.sortHand();
	assertEquals("[1, 2, 3, 4, 5, 60]", nord.getHand().toString());
}

@Test
public void testRemoveCardFromHand() {
	// (init)
	nord.setHand(hand6);
	assertEquals("[4, 2, 5, 3, 60, 1]", nord.getHand().toString());

	nord.removeCardFromHand(3);
	assertEquals("[4, 2, 5, 60, 1]", nord.getHand().toString());
}

@Test
public void testCanRemoveFromHand() {
	// (init)
	nord.setHand(hand6);
	assertEquals("[4, 2, 5, 3, 60, 1]", nord.getHand().toString());

	// test method
	assertTrue(nord.canRemoveFromHand(1));
	assertTrue(nord.canRemoveFromHand(60));
	assertFalse(nord.canRemoveFromHand(9));
	assertFalse(nord.canRemoveFromHand(11));
}
```

```java
@Test
public void testPickCards() {
	// (init)
	setPack(nord, new int[] {7, 8, 9});
	nord.setHand(hand2);
	assertEquals("[7, 8, 9]", nord.getPack().getPack().toString());
	assertEquals("[11, 5]", nord.getHand().toString());

	assertEquals(2, nord.pickCards(2));
	assertEquals("[9]", nord.getPack().getPack().toString());
	assertEquals("[11, 5, 7, 8]", nord.getHand().toString());

	assertEquals(1, nord.pickCards(2));
	assertEquals("[]", nord.getPack().getPack().toString());
	assertEquals("[11, 5, 7, 8, 9]", nord.getHand().toString());
}

@Test
public void testHadNoMoreCards() {
	// un joueur n'a pas de paquet vide une fois créé
	assertFalse(nord.hadNoMoreCards());

	// vérif une fois tt vide
	nord.getPack().getPack().clear();
	nord.getHand().clear();
	assertTrue(nord.hadNoMoreCards());
}

@Test
public void testFillCardsComplete() {
	setPack(nord, new int[] {7, 8, 9, 10, 15});
	nord.setHand(hand2);
	assertEquals("[7, 8, 9, 10, 15]", nord.getPack().getPack().toString());
	assertEquals("[11, 5]", nord.getHand().toString());

	// testing usual case
	assertEquals(4, nord.fillCards(6));
	assertEquals("[15]", nord.getPack().getPack().toString());
	assertEquals("[11, 5, 7, 8, 9, 10]", nord.getHand().toString());
	assertEquals(0, nord.fillCards(6));
}

@Test
public void testFillCardsIncomplete() {
	// init
	setPack(nord, new int[] {7, 8, 9});
	nord.setHand(hand2);
	assertEquals("[7, 8, 9]", nord.getPack().getPack().toString());
	assertEquals("[11, 5]", nord.getHand().toString());

	// testing the case where there is 1 card is missing in both hand and pack
	assertEquals(3, nord.fillCards(6));
	assertEquals("[]", nord.getPack().getPack().toString());
	assertEquals("[11, 5, 7, 8, 9]", nord.getHand().toString());
	assertEquals(0, nord.fillCards(6));
}
```

```java
    // HORS TEST
    private void setPack(Player p, int[] values) {
          nord.getPack().getPack().clear();
          for (int i : values) {
                p.getPack().getPack().add(i);
          }
     }
}
```

```java
    /**

     * StackTest.java

     * Classe permettant les tests unitaires des Stacks.
     * @author Antoine <antoine@jiveoff.fr> & Logan Tann
     * @project Projet-BPO
     */

    package test;

    import static org.junit.Assert.*;

    import org.junit.Before;
    import org.junit.Test;

    import appli.Stack;

    public class StackTest {

    private Stack AscValues_Stack, DescValues_Stack;

        @Before
        public void setUp() {
           AscValues_Stack = new Stack(Stack.TypeStack.ASC);
           DescValues_Stack = new Stack(Stack.TypeStack.DESC);

           // (values reset for every tests except firstConstructor)
           AscValues_Stack.addCard(1);
           DescValues_Stack.addCard(60);
        }


    /*
     * METHODS
     */
    @Test
    public void firstConstructor() {
          AscValues_Stack = new Stack(Stack.TypeStack.ASC);
          DescValues_Stack = new Stack(Stack.TypeStack.DESC);
          /** -1 is the fallback value when no cards have been added */
          assertEquals(-1, AscValues_Stack.getCardOnTop());
          assertEquals(-1, DescValues_Stack.getCardOnTop());
    }
```

```java
@Test
public void getterAndSetter() {
	/**  the first time we assign a value */
	AscValues_Stack.addCard(1);
	DescValues_Stack.addCard(60);
	assertEquals(1, AscValues_Stack.getCardOnTop());
	assertEquals(60, DescValues_Stack.getCardOnTop());

	/**  second time we change the value */
	AscValues_Stack.addCard(58);
	DescValues_Stack.addCard(8);
	assertEquals(58, AscValues_Stack.getCardOnTop());
	assertEquals(8, DescValues_Stack.getCardOnTop());
}

@Test
public void toStringMethod() {
	/** by exact values */
	assertEquals("^[01]", AscValues_Stack.toString());
	assertEquals("v[60]", DescValues_Stack.toString());

	AscValues_Stack.addCard(58);
	DescValues_Stack.addCard(8);
	assertEquals("^[58]", AscValues_Stack.toString());
	assertEquals("v[08]", DescValues_Stack.toString());

	/** length should be constant if it follows the correct rules of the game*/
	assertEquals(5, AscValues_Stack.toString().length());
	assertEquals(5, DescValues_Stack.toString().length());
}

@Test
public void CloneConstructor() {
	/** creating and checking clone */
	Stack AscClone = new Stack(this.AscValues_Stack);
	Stack DescClone = new Stack(this.DescValues_Stack);
	assertEquals(1, AscClone.getCardOnTop());   // (checking clone values)
	assertEquals(60, DescClone.getCardOnTop());
	assertEquals("^[01]", AscClone.toString());
	assertEquals("v[60]", DescClone.toString());

	/** editing the original stack shouldn't edit the cloned stack*/
	AscValues_Stack.addCard(58);
	DescValues_Stack.addCard(8);
	assertEquals(1, AscClone.getCardOnTop());
	assertEquals(60, DescClone.getCardOnTop());
}

/*
 * NOT METHODS
 */

private void savestate_example_dummyValues() {
	DescValues_Stack.addCard(44);
	DescValues_Stack.addCard(54);
	AscValues_Stack.addCard(20);
	AscValues_Stack.addCard(23);
	AscValues_Stack.addCard(10);
}
```

```java
    @Test
    public void savestate_exampleByValue() {
           AscValues_Stack.addCard(3);
           DescValues_Stack.addCard(57);

           /**  create savestate :*/
           Stack save_StackASC = new Stack(AscValues_Stack);
           Stack save_StackDESC = new Stack(DescValues_Stack);

           savestate_example_dummyValues();

           /**  load savestate by value :*/
           AscValues_Stack.addCard(save_StackASC.getCardOnTop());
           DescValues_Stack.addCard(save_StackDESC.getCardOnTop());

           assertEquals(3, AscValues_Stack.getCardOnTop());  // (checking if it's success)
           assertEquals(57, DescValues_Stack.getCardOnTop());
    }


    /**
     * Same as savestate_exampleByValue, but with a different load technique
     */
    @Test
    public void savestate_exampleByReference() {
           AscValues_Stack.addCard(3);
           DescValues_Stack.addCard(57);
           Stack save_StackASC = new Stack(AscValues_Stack);
           Stack save_StackDESC = new Stack(DescValues_Stack);
           savestate_example_dummyValues();

           /**  load savestate by reference :*/
                  AscValues_Stack = save_StackASC;
                  DescValues_Stack = save_StackDESC;

           assertEquals(3, AscValues_Stack.getCardOnTop());
           assertEquals(57, DescValues_Stack.getCardOnTop());
    }
}
```

```java
/**

 * Application.java

 * Programme principal du programme.
 * @author Antoine <antoine@jiveoff.fr> & Logan Tann
 * @project Projet-BPO
 */

package appli;

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

public class Application {

    public static boolean VERBOSE = false;

    public static void init_VERBOSE(String[] args) {
        for (String elem: args) {
            if (elem.contains("-v") || elem.contains("--verbose")) {
                VERBOSE = true;
                System.out.println("$I : Mode verbeux activé");
                break;
            }
        }
    }

    public static void main(String[] args) {
        init_VERBOSE(args);

        Player NORD = new Player("NORD");
        Player SUD = new Player("SUD");
        boolean NORD_plays = true;

        boolean isPlaying = true;
        while (isPlaying) {
            // affichage des infos des joueurs :
            System.out.println(NORD);
            System.out.println(SUD);

            try {
                // C'est le tour de nord ou sud :
                if (NORD_plays) {
                    turn(NORD, SUD);
                } else {
                    turn(SUD, NORD);
                }
            } catch (WinnerException thePlayer) {
                System.out.println("partie finie, " + thePlayer.getMessage() + " a gagné");
                isPlaying = false;
            } catch (LoserException thePlayer) {
                String winnerName = thePlayer.toString().equals(NORD.getName()) ? NORD.getName() :
SUD.getName();
                System.out.println("partie finie, " + winnerName + " a gagné");
                isPlaying = false;
            }
```

```
            NORD_plays = !NORD_plays;
        }
    }



    public static void turn(Player me, Player opponent) throws WinnerException, LoserException {
        me.sortHand();
        System.out.println(me.hand_toString());
        boolean showErrorPrompt = false, requestValidMove = true;
        String input;
        Scanner sc = new Scanner(System.in);

        if(!me.canPlay(opponent)) throw new LoserException(me.getName());

        while (requestValidMove) {
            System.out.print((showErrorPrompt) ? "#> " : "> ");
            input = sc.nextLine();

            // Interprétation de l'entrée
            ArrayList<Action> parsedActions = parseInput(input);
            showErrorPrompt = parsedActions.size() < 2;
            if (showErrorPrompt) {
                if (VERBOSE) System.out.println("$E : Syntax error or not enough moves");
                continue;
            }

            // Exécution de l'entrée
            boolean playedEnemy = false;
            me.save();
            opponent.save();

            int movesDone = 0;

            try {
                for (Action action: parsedActions) {
                    playedEnemy = action.handlePlayingInEnemyStack(playedEnemy);
                    action.execute(me, opponent);
                    movesDone++;
                }
            } catch (BadMoveException err) {
                me.restoreSave();
                opponent.restoreSave();
                showErrorPrompt = true;
                if (VERBOSE) System.out.println("$E : " + err.toString());
                continue;
            }

            // Conditions de fin de tour
            if (me.hadNoMoreCards()) {
                throw new WinnerException(me.getName());
            }
            System.out.print(movesDone + " cartes posées");

            if (playedEnemy) {
                System.out.print(", " + me.fillCards(6) + " cartes piochées" + System.lineSeparator());
            } else {
                System.out.print(", " + me.pickCards(2) + " cartes piochées" + System.lineSeparator());
            }
```

```
            // fin du tour
            requestValidMove = false;
        }

    }



    /**
     * Décompose et vérifie l'entrée de l'utilisateur en un tableau d'actions interprétables par le
programme.
     * @param input (String) l'entrée de l'utilisateur
     * @return un ArrayList contenant chaque actions (type Action). Vide si entrée invalide.
     */
    public static ArrayList<Action> parseInput(String input) {
        ArrayList<Action> retval = new ArrayList<>();

        String[] coups = input.split(" ");
        for (String coup : coups) {

            int card;
            try {
                card = Integer.parseInt(coup.substring(0, 2));
            } catch(NumberFormatException | StringIndexOutOfBoundsException e) {
                if (VERBOSE) System.out.println("$E : (syntax) " + e);
                retval.clear();
                return retval;
            }

            try {
                if (coup.charAt(2) != '^') {
                    if (coup.charAt(2) != 'v') {
                        if (VERBOSE) System.out.println("$E : (syntax) the character that precedes the
number have to be ^ or v, got " + coup.charAt(2));
                        retval.clear();
                        return retval;
                    }
                }
                if (coup.length() > 3 && coup.charAt(3) != '’') {
                    if (VERBOSE) System.out.println("$E : (syntax) the second character that precedes
the number have to be ’, got " + coup.charAt(3));
                    retval.clear();
                    return retval;
                }
            } catch(IndexOutOfBoundsException e) {
                if (VERBOSE) System.out.println("$E : (syntax) the character that precedes the number
have to be ^ or v, got nothing");
                retval.clear();
                return retval;
            }

            // si on en est là, c'est que la syntaxe est valide
            Stack.TypeStack type = coup.charAt(2) == '^' ? Stack.TypeStack.ASC : Stack.TypeStack.DESC;
            retval.add(new Action(card, type, coup.length() > 3));
        }
        if (VERBOSE) System.out.println("$I : vous avez joué : " + retval);
        return retval;
    }
}
```

```java
/**

 * Player.java
 * Classe permettant la création et la manipulation des joueurs.
 * @author Antoine <antoine@jiveoff.fr> & Logan Tann
 * @project Projet-BPO
 */

package appli;

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

public class Player {

    /** name : son nom (attendu NORD ou SUD)*/
    private final String name;

    /** pack : la pioche du joueur */
    private Pack pack;
    private Pack pack_save;

    /** stackASC / stackDESC : les deux piles de cartes du joueur */
    private Stack stackASC, stackDESC;
    private Stack stackASC_save, stackDESC_save;

    /** hand : les cartes qui se trouvent dans la main du joueur*/
    private ArrayList<Integer> hand;
    private ArrayList<Integer> hand_save;

    /**
     * Constructeur de l'entité Joueur
     * @param name son joli nom pour l'affichage
     */
    public Player(String name) {
        assert name.length() < 5 : "Le nom ne doit pas excéder 4 caractères. Valeur conseillée : NORD
ou SUD.";
        this.name = name;

        this.pack = new Pack(1, 60);
        this.stackASC = new Stack(Stack.TypeStack.ASC);
        this.stackDESC = new Stack(Stack.TypeStack.DESC);

        this.stackASC.addCard( this.pack.pickCard(0) );
        this.stackDESC.addCard( this.pack.pickLastCard() );

        // à partir du moment où on mélange, prendre la première carte équivaut à prendre une carte du
paquet au hasard
        this.pack.shuffle();

        // génération de la main
        this.hand = new ArrayList<>();
        for(int i = 0; i < 6; ++i) {
            this.hand.add( this.pack.pickFirstCard() ); // < R to L :prendre la première carte et
l'ajouter dans la main
        }
    }
```

```java
public String toString() {
    String retval = String.format("%-5s", this.name);
    retval += stackASC.toString() + " " + stackDESC.toString();
    retval += " (m" + this.hand.size() + "p" + this.pack.getPackLength() + ")";
    return retval;
}

public String getName() {
    return this.name;
}

public String hand_toString() {
    // cartes NORD { 15 20 23 32 41 48 }
    StringBuilder retval = new StringBuilder("cartes ");
    retval.append(this.name);
    retval.append(" { ");
    for (int carte : this.hand) {
        retval.append(carte < 10 ? ("0" + carte) : carte);
        retval.append(" ");
    }
    retval.append("}");
    return retval.toString();
}

public boolean hadNoMoreCards() {
    return this.pack.isEmpty() && this.hand.size() == 0;
}

public int fillCards(int targetCards) {
    int picked = 0;
    while (!this.pack.isEmpty() && this.hand.size() < targetCards) {
        this.pickCardAndAddInHand();
        picked++;
    }
    return picked;
}

public int pickCards(int cardsToPick) {
    int picked = 0;
    while (!this.pack.isEmpty() && picked < cardsToPick) {
        this.pickCardAndAddInHand();
        picked++;
    }
    return picked;
}

private void pickCardAndAddInHand() {
    this.hand.add( this.pack.pickCard(0) );
}

public Stack getStack(Stack.TypeStack type) {
    return (type == Stack.TypeStack.ASC) ? this.stackASC : this.stackDESC;
}

public boolean removeCardFromHand(int cardValue) {
    // cast nécessaire pour éviter confusion avec la surcharge .remove(int index);
    return this.hand.remove((Object) cardValue);
}
```

```
    public boolean canRemoveFromHand(int cardValue) {
        return this.hand.contains(cardValue);
    }

    public void sortHand() {
        Collections.sort(this.hand);
    }

    public void putDown(Player cardSource, Action theAction) throws BadMoveException {
        int card = theAction.getCard();
        if (!cardSource.removeCardFromHand(card)) {
            throw new BadMoveException("La carte du coup " + theAction.toString() + " n'existe même pas
dans votre main...");
        }
        Stack target = this.getStack(theAction.getType());
        target.addCard(card);
    }

    public void save() {
        this.pack_save = new Pack(this.pack);
        this.stackASC_save = new Stack(this.stackASC);
        this.stackDESC_save = new Stack(this.stackDESC);
        this.hand_save = new ArrayList<>(this.hand);
    }
    public void restoreSave() {
        this.pack = this.pack_save;
        this.stackASC = this.stackASC_save;
        this.stackDESC = this.stackDESC_save;
        this.hand = this.hand_save;
    }

    public boolean canPlay(Player you) {
        int canPlaySize = 0;
        boolean playedInAdversaire = false;
        System.out.print("$V : Coups possibles: ");
        for (int c: this.hand) {
            Action test = new Action(c, Stack.TypeStack.ASC, false);
            try {
                test.validMove(this, you);
                System.out.print(test + " ");
                canPlaySize++;
                continue;
            } catch(BadMoveException ignored) {}

            test = new Action(c, Stack.TypeStack.DESC, false);
            try {
                test.validMove(this, you);
                System.out.print(test + " ");
                canPlaySize++;
                continue;
            } catch(BadMoveException ignored) {}

            test = new Action(c, Stack.TypeStack.ASC, true);
            try {
                test.validMove(this, you);
                if(!playedInAdversaire) {
                    System.out.print(test + " ");
                    canPlaySize++;
                    playedInAdversaire = true;
                }
                continue;

        } catch(BadMoveException ignored) {}
```

```
            test = new Action(c, Stack.TypeStack.DESC, true);
            try {
                test.validMove(this, you);
                if(!playedInAdversaire) {
                    System.out.print(test + " ");
                    canPlaySize++;
                    playedInAdversaire = true;
                }
            } catch(BadMoveException ignored) {}
        }
        System.out.println("[" + canPlaySize + "]");
        return canPlaySize > 1;
    }

    /*                    /!\ ATTENTION : /!\
          ces setteurs sont utilisés seulement à des fins de tests unitaires
        */
    public void setPack(Pack pack) {
        this.pack = pack;
    }
    public Pack getPack() {
       return pack;
}

    public void setHand(ArrayList<Integer> hand) {
        this.hand = hand;
    }
public ArrayList<Integer> getHand() {
       return hand;
}

    public void setStackASC(Stack stackASC) {
        this.stackASC = stackASC;
    }
    public void setStackDESC(Stack stackDESC) {
        this.stackDESC = stackDESC;
    }

}
```

```
/**
```

```java
/*
 * Stack.java
 * Classe permettant la création et la manipulation des piles de cartes.
 * @author Antoine <antoine@jiveoff.fr> & Logan Tann
 * @project Projet-BPO
 */

package appli;

public class Stack {

    public enum TypeStack { ASC, DESC }

    /** TypeStack : définit si la pile est ascendante (ASC) ou (DESC) */
    private TypeStack type;


    /** Dans une pile de carte, et dans le contexte de l'exercice, stocker uniquement la dernière carte
est suffisant */
    private int topCard = -1;

    /**
     * Stack : constitue une pile ascendante ou descendante
     * @param type : définit si la pile est ascendante (ASC) ou (DESC)
     */
    public Stack(TypeStack type) {
        this.type = type;
    }

    public Stack(Stack toClone) {
        this.type = toClone.type;
        this.topCard = toClone.topCard;
    }

    public void addCard(int card) {
        this.topCard = card;
    }

    public int getCardOnTop() {
        return this.topCard;
    }

    public String toString() {
        char operator = (type == TypeStack.ASC) ? '^' : 'v';
        return String.format("%c[%02d]", operator, this.getCardOnTop());
    }

}

/**
```

```java
/*
 * Pack.java
 * Classe permettant la création des packs de cartes.
 * @author Antoine <antoine@jiveoff.fr> & Logan Tann
 * @project Projet-BPO
 */

package appli;

import java.util.Collections;
import java.util.LinkedList;

public class Pack {
    public static int DEFAULT_FIRST_CARD = 1;
    public static int DEFAULT_LAST_CARD = 60;

    /**
     * @implNote Usage de linkedList car la plupart des opérations sont les insertions/délétions
     *      au cours d'une partie
     */
    private final LinkedList<Integer> pack;

    /**
     * Crée un nouveau paquet de carte donné la valeur de la première et dernière carte
     * @param min valeur de la première carte
     * @param max valeur de la dernière carte
     * @pre 0 ⩽ min ⩽ max
     * @see Pack() pour appeler ce constructeur avec les valeurs par défaut
     */
    public Pack(int min, int max) {
        assert 0 ⩽ min : "La valeur minimale ne doit pas être négative";
        assert min ⩽ max : "La valeur maximale doit être plus grande que la valeur minimale";

        this.pack = new LinkedList<>();
        for(int i = min; i ⩽ max; ++i) {
            this.pack.add(i);
        }
    }
    /**
     * Crée un nouveau paquet de carte (cartes numérotées de 1 à 60 par défaut)
     */
    public Pack() {
        this(DEFAULT_FIRST_CARD, DEFAULT_LAST_CARD);
    }

    /**
     * Duplique un paquet de cartes
     */
    public Pack(Pack toClone) {
        this.pack = new LinkedList<>(toClone.pack);
    }

    /* Fonctions pour intervenir sur le paquet ---------------- */
```

```java
    /**
     * Mélange le paquet de carte.
     */
    public void shuffle() {
        Collections.shuffle(this.pack);
    }

    /**
     * Récupère une carte dans le paquet (autrement dit, supprime la carte donné sa position + retourne
sa valeur)
     *
     * @param index L'indice de la carte. La carte concernée sera supprimée du paquet.
     * @return La valeur de la carte. Si il n'y a plus de cartes dans le paquet, retourne -1.
     * @implNote On spécifie la carte par son <b>indice</b> et non sa <b>valeur</b> ! On peut tirer
partie (ou PAS...)
     *       de cette méthode d'implémentation.
     * @see this.getPack() pour récupérer le paquet. Cela pourrait être utile pour intervenir par
<b>valeur</b> plutôt
     *      que par son <b>indice</b>.
     */
    public int pickCard(int index) {
        if(this.getPackLength() > 0) {
            int retval = this.pack.get(index);
            this.pack.remove(index);
            return retval;
        }
        return -1;
    }

    /**
     * Raccourci à pickCard() pour récupérer la dernière carte du paquet
     * @return la valeur de la dernière carte du paquet
     */
    public int pickLastCard() {
        // utilisation de la méthode getPackLength plutôt que this.pack.size() au cas où on change de
méthode de stockage
        return this.pickCard( this.getPackLength() - 1);
    }

    /** Raccourci à pickCard() pour récupérer la première carte du paquet
     * @return la valeur de la première carte du paquet
     * @implNote Une fois le paquet mélangé, cette méthode est utile pour piocher une carte au hasard.
     */
    public int pickFirstCard() {
        return this.pickCard(0);
    }

    public boolean isEmpty() {
        return this.pack.isEmpty();
    }

    /* Fonctions pour obtenir des infos sur le paquet ----------------- */

    public LinkedList<Integer> getPack() {
        return this.pack;
    }




    public int getPackLength() {
        return this.pack.size();
    }
```

```
    public boolean exists(int card) {
        return this.pack.contains(card);
    }

}



/**

 * Action.java

* Classe permettant de vérifier l'intégrité des coups et de les exécuter dans les piles renseignées.
 * @author Antoine <antoine@jiveoff.fr> & Logan Tann
 * @project Projet-BPO
 */

package appli;

public class Action {
    private final int card;
    private final Stack.TypeStack type;
    private final boolean playsInEnemyStack;

    /**
     * CLASSE Action : définit une action possible du joueur (tel que poser la carte 4 sur le paquet
adverse).
     * Les méthodes permettent d'effectuer des vérifications de jouabilité. Les arguments de ce
constructeurs
     * constituent les uniques variables privées de la classe
     * @param card      Numéro de la carte source qui sera déplacée
     * @param type      définit dans quel type de pile cible la carte sera posée
     * @param adverse   indique si la pile cible se trouve dans le camp ennemi (et traiter les règles
supplémentaires en
     *                  conséquence)
     */
    Action(int card, Stack.TypeStack type, boolean adverse) {
        this.card = card;
        this.type = type;
        this.playsInEnemyStack = adverse;
    }

    /**
     * UNUSED !! (enfin, seulement dans le mode verbose qui est désactivé durant la release)
     */
    public String toString() {
        StringBuilder retval = new StringBuilder();
        if (this.card < 10) retval.append(0);
        retval  .append(this.card)
                .append( (this.type == Stack.TypeStack.ASC) ? "^" : "v");
        if (this.playsInEnemyStack) retval.append("'");
        return retval.toString();
    }




    public int getCard() {
        return card;
    }
```

```
    public Stack.TypeStack getType() {
        return type;
    }

    public boolean handlePlayingInEnemyStack(boolean currentState) throws BadMoveException {
        if (this.playsInEnemyStack) {
            if (currentState) {
                throw new BadMoveException("Second jeu dans une pile de l'adversaire interdite !");
            } else {
                currentState = true;
            }
        }
        return currentState;
    }

    /**
     * Détecte si le coup peut être joué
     * @param me Le joueur qui exécute le coup
     * @param you Le joueur adverse
     * @throws BadMoveException Si le coup n'est pas jouable. Le coup est donc valide si aucune erreur
n'est jetée.
     */
    public void validMove (Player me, Player you) throws BadMoveException {
        if (this.playsInEnemyStack) {
            String errMsg = "Le coup " + this.toString() + " ne respecte pas la règle « la carte doit
être plus \n"
                    + "RÈGLE adverse», ou bien vous avez joué plus d'une fois la même carte";
            Stack stackToCheck = you.getStack(this.type);

            if (this.type == Stack.TypeStack.ASC && this.card ⩾ stackToCheck.getCardOnTop()) {
                throw new BadMoveException(errMsg.replace("RÈGLE", "petite sur la pile ascendante"));
            }
            if (this.type == Stack.TypeStack.DESC && this.card ⩽ stackToCheck.getCardOnTop()) {
                throw new BadMoveException(errMsg.replace("RÈGLE", "grande sur la pile descendante"));
            }
        } else {
            String errMsg = "Le coup " + this.toString() + " ne respecte pas la règle « la carte doit
être plus \n"
                    + "RÈGLE » ni la règle de la dizaine, ou bien vous avez joué plus d'une \nfois la
même carte";
            Stack stackToCheck = me.getStack(this.type);
            int top = stackToCheck.getCardOnTop();

            if ( (this.type == Stack.TypeStack.ASC && this.card ⩽ top) && this.card ≠ top - 10) {
                throw new BadMoveException(errMsg.replace("RÈGLE", "grande que votre pile
ascendante"));
            }
            if ( (this.type == Stack.TypeStack.DESC && this.card ⩾ top) && this.card ≠ top + 10) {
                throw new BadMoveException(errMsg.replace("RÈGLE", "petite que votre pile
descendante"));
            }
        }
    }




    public void execute(Player me, Player you) throws BadMoveException {
        validMove(me, you);
        // si pas throw ⟺ si le coup + valeur de la carte est jouable :
        if (this.playsInEnemyStack) {
```

```
            you.putDown(me, this);
        } else {
            me.putDown(me,this);
        }
    }
}
```

Exceptions du projet

```
/**
 * BadMoveException.java
 * @author Logan Tann
 */

package appli;

/**
 * BadMoveException est juste un alias de Exception mais je veux spécifier explicitement que l'on
rejette bien la faute
 * au <b>coup</b> du joueur, et non la faute du programmeur
 * Les messages d'erreurs sont utiles uniquement lorsque le mode verbeux est activé.
*/
public class BadMoveException extends Exception {
    public BadMoveException() { super(); }
    public BadMoveException(String message) { super(message); }
    public BadMoveException(String message, Throwable cause) { super(message, cause); }
    public BadMoveException(Throwable cause) { super(cause); }
}


/**
 * LoserException.java
 * @author Logan Tann
 */

package appli;

/**
 * LoserException est juste un alias de Exception mais je veux spécifier explicitement qu'un joueur
perd la partie.
 * Les messages d'erreurs sont utiles uniquement lorsque le mode verbeux est activé.
 */
public class LoserException extends Exception {
    public LoserException() { super(); }
    public LoserException(String message) { super(message); }
    public LoserException(String message, Throwable cause) { super(message, cause); }
    public LoserException(Throwable cause) { super(cause); }
}
```

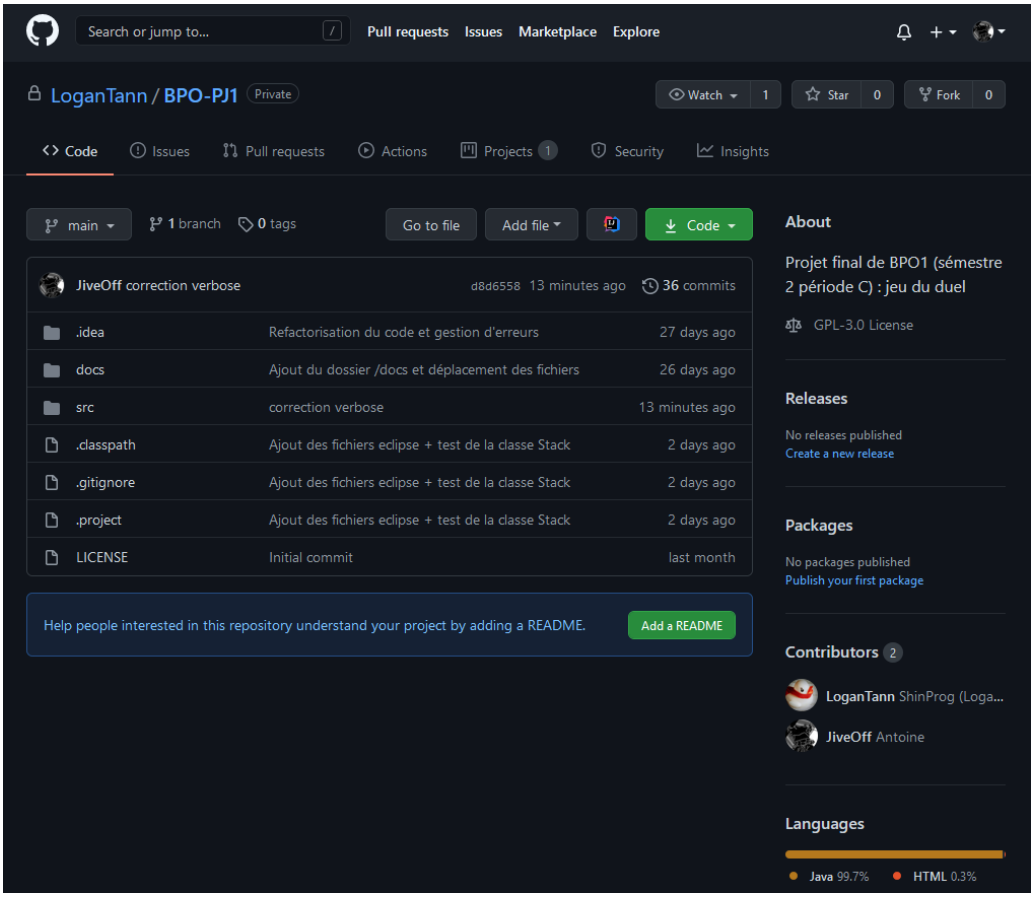
```java
/**
 * WinnerException.java
 * @author Logan Tann
 */

package appli;

/**
 * WinnerException est juste un alias de Exception mais je veux spécifier explicitement qu'un joueur a
gagné la partie.
 * Les messages d'erreurs sont utiles uniquement lorsque le mode verbeux est activé.
 */

public class WinnerException extends Exception {
    public WinnerException() { super(); }
    public WinnerException(String message) { super(message); }
    public WinnerException(String message, Throwable cause) { super(message, cause); }
    public WinnerException(Throwable cause) { super(cause); }
}
```

GitHub du projet

Lien vers notre GitHub de projet :

https://github.com/LoganTann/BPO-PJ1

(sera disponible le jour du rendu)



Fin du rapport, merci d'avoir lu !