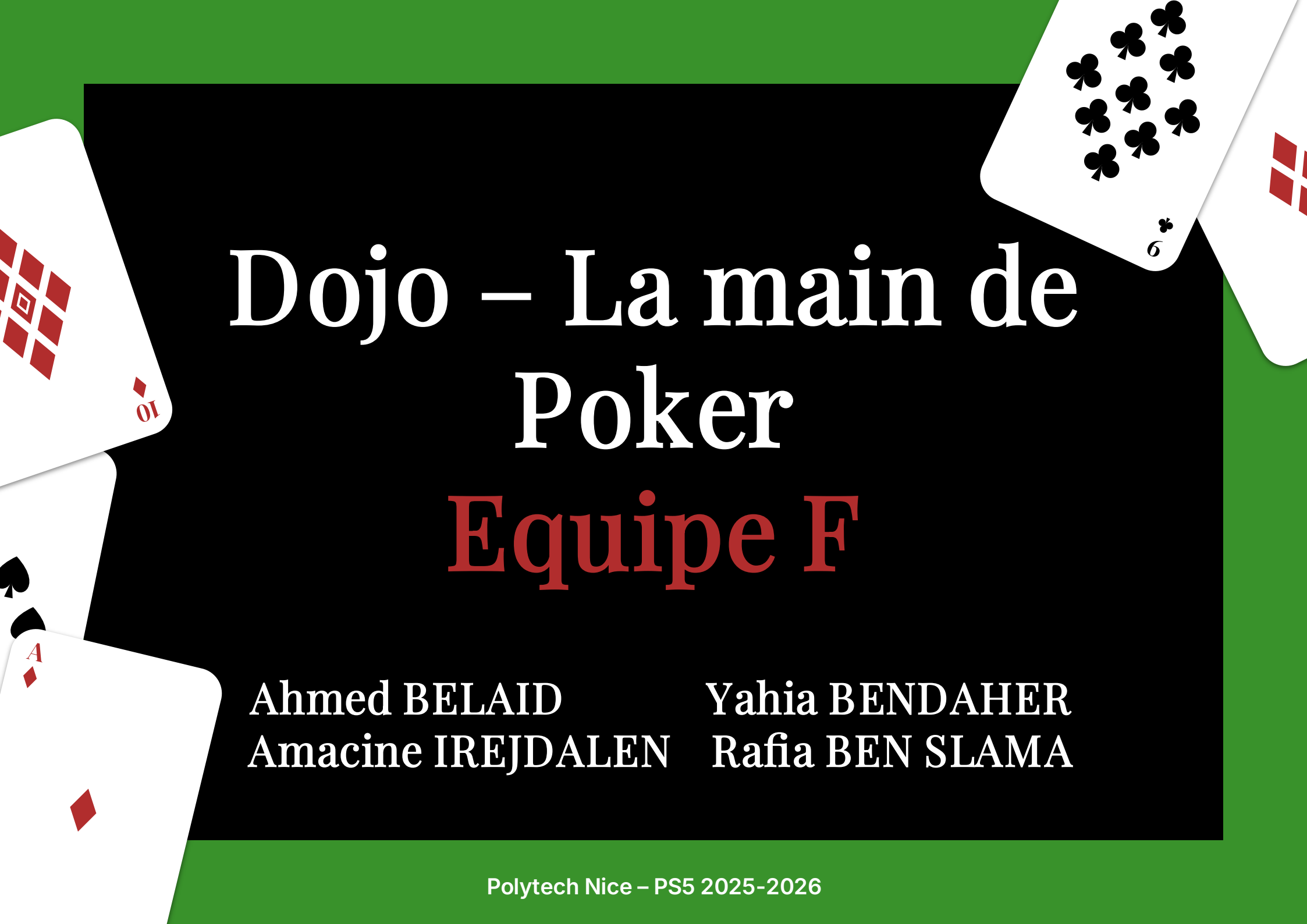


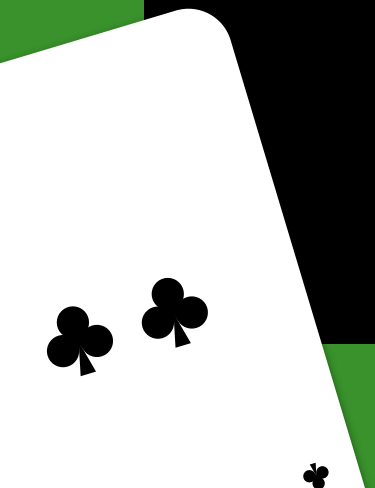
Dojo – La main de Poker

Equipe F

Ahmed BELAID
Yahia BENDAHER
Amacine IREJDALEN
Rafia BEN SLAMA

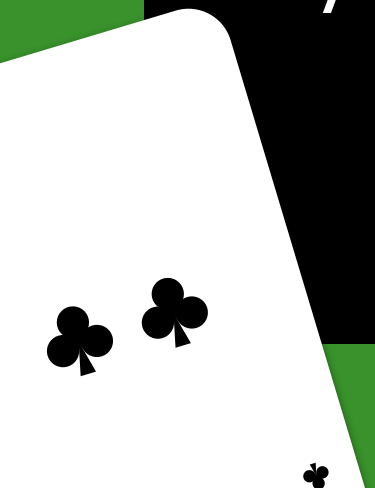
Polytech Nice – PS5 2025-2026

Sommaire



01

Fonctionnalités réalisées / non réalisées



Fonctionnalités réalisées

L’ensemble des objectifs prévus pour ce projet a été atteint :

- Analyse et validation des cartes
- Validation complète d'une main (format, doublons, 5 cartes)
- Évaluation de la combinaison de Poker
- Comparaison de deux mains

Fonctionnalités non réalisées

- Messages d'erreur encore incomplets lors de la validation des cartes
- Améliorations possibles sur l'ergonomie de l'affichage

Exemple :

```
Main 1: 10Ca 10Co 8Tr 7pi 16Co
Valeur '1' invalide (attendu : 2..10, V, D, R, A)
```


02
Qualité du code et abstractions

Points forts

- Parsing isolé avec CardSeperator
- Architecture claire
- Résultat structuré avec HandEvaluation
  - ↠ Rang + valeurs + couleur dans un objet cohérent
- Comparaison séparée dans HandComparator
- Refactorisation réussie depuis PokerRules
  - ↠ Code plus modulaire, chacun a pu travailler en parallèle

```java
public static HandEvaluation evaluateHand(Hand hand) {  18 usages  Yahia BENDAHER +2
    List<Card> cards = hand.getCards();
    List<Integer> sortedValues = getSortedCardValues(cards);

    HandEvaluation royalFlush = evaluateRoyalFlush(cards);
    if (royalFlush != null) return royalFlush;

    HandEvaluation straightFlush = evaluateStraightFlush(cards, sortedValues);
    if (straightFlush != null) return straightFlush;

    HandEvaluation fourOfKind = evaluateFourOfKind(cards, sortedValues);
    if (fourOfKind != null) return fourOfKind;

    HandEvaluation fullHouse = evaluateFullHouse(cards);
    if(fullHouse != null) return fullHouse;

    HandEvaluation flush = evaluateFlush(cards, sortedValues);
    if (flush != null) return flush;

    HandEvaluation straight = evaluateStraight(cards, sortedValues);
    if (straight != null) return straight;

    HandEvaluation threeOfKind = evaluateThreeOfKind(cards, sortedValues);
    if (threeOfKind != null) return threeOfKind;

    HandEvaluation twoPair = evaluateTwoPair(cards, sortedValues);
    if (twoPair != null) return twoPair;

    HandEvaluation pair = evaluatePair(cards, sortedValues);
    if (pair != null) return pair;

    return new HandEvaluation(HandRank.HIGH_CARD, sortedValues);
}
```

Points faibles / à améliorer

- Trop de responsabilités dans HandEvaluation
  ↠ tri, evaluate les règles, kickers, nombreux if
- Responsabilités mal séparées
- Code statique difficile à modulariser
- Textes “en dur” dans HandComparator
- Prochaine étape évidente : interface Rule
  ↠ Pour alléger HandEvaluation et clarifier le design

```java
private static String buildResultString(String winner, HandEvaluation eval) {  4 usages  rafiabenslama +3
    HandRank rank = eval.getRank();
    List<Integer> values = eval.getValues();

    int highVal = values.get(0);
    String symbol = Value.getNameFromNumber(highVal);

    switch (rank) {
        case HIGH_CARD:{
            return winner + " gagne avec carte la plus élevée : " + symbol;
        }

        case PAIR:{
            return winner + " gagne avec paire de " + symbol;
        }

        case TWO_PAIR:{
            int lowVal = values.get(1);
            String lowSymbol = Value.getNameFromNumber(lowVal);
            return winner + " gagne avec double paire de " + symbol + " et " + lowSymbol;
        }

        case THREE_OF_A_KIND:{
            return winner + " gagne avec brelan de " + symbol;
        }

        case STRAIGHT:{
            return winner + " gagne avec une suite, carte la plus haute : " + symbol;
        }
```


03
Confiance dans les tests

Confiance dans les tests

- Tests unitaires couvrent toutes les règles du poker
- Structure du code qui facilite les tests
- Cas critiques testés
- Affichage clair grâce à la gestion des erreurs

↠ Si une main est invalide, on indique exactement quelle valeur ou quelle couleur pose un problème

Element	Class, %	Method, %	Line, %	Branch, %
fr.pns.poker	92% (24/26)	91% (67/73)	87% (353/403)	80% (190/235)
Main	0% (0/1)	0% (0/1)	0% (0/6)	100% (0/0)
utils	0% (0/1)	0% (0/1)	0% (0/17)	0% (0/8)
IOUtils	0% (0/1)	0% (0/1)	0% (0/17)	0% (0/8)
rules	100% (9/9)	100% (9/9)	94% (82/87)	88% (76/86)
TwoPairsRule	100% (1/1)	100% (1/1)	92% (13/14)	90% (9/10)
ThreeOfKindRule	100% (1/1)	100% (1/1)	100% (7/7)	100% (6/6)
StraightRule	100% (1/1)	100% (1/1)	94% (16/17)	88% (16/18)
StraightFlush	100% (1/1)	100% (1/1)	100% (8/8)	87% (7/8)
RoyalFlushRule	100% (1/1)	100% (1/1)	83% (5/6)	75% (6/8)
PairRule	100% (1/1)	100% (1/1)	100% (7/7)	100% (6/6)
FullHouseRule	100% (1/1)	100% (1/1)	92% (13/14)	87% (14/16)
FourOfKindRule	100% (1/1)	100% (1/1)	100% (7/7)	100% (6/6)
FlushRule	100% (1/1)	100% (1/1)	85% (6/7)	75% (6/8)
model	100% (6/6)	93% (30/32)	92% (115/125)	76% (51/67)
Value	100% (1/1)	100% (7/7)	95% (39/41)	88% (15/17)
HandRank	100% (1/1)	100% (4/4)	100% (14/14)	100% (0/0)
Hand	100% (1/1)	71% (5/7)	78% (18/23)	70% (14/20)
Color	100% (1/1)	100% (6/6)	86% (19/22)	66% (8/12)
CardSeperator	100% (1/1)	100% (1/1)	100% (16/16)	77% (14/18)
Card	100% (1/1)	100% (7/7)	100% (9/9)	100% (0/0)
exception	100% (6/6)	100% (7/7)	100% (7/7)	100% (0/0)
PokerException	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
InvalidValueException	100% (1/1)	100% (2/2)	100% (2/2)	100% (0/0)
InvalidHandException	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
InvalidColorException	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
InvalidCardFormatException	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
DuplicateCardException	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
evaluator	100% (3/3)	91% (21/23)	92% (149/161)	85% (63/74)
HandEvaluation	100% (1/1)	90% (18/20)	94% (110/117)	93% (43/46)
HandComparator	100% (2/2)	100% (3/3)	88% (39/44)	71% (20/28)

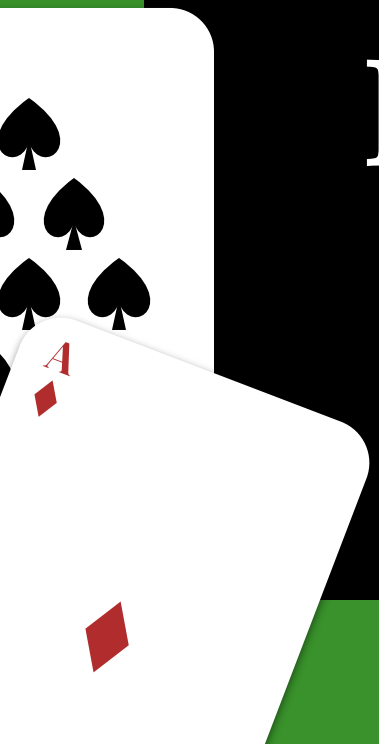
Confiance dans les tests

Limite : Absence de vérification “jeu réel”

```
Main 1: 10Ca 10Co 8Tr 7Pi 3Co
Main 2: 9Ca 9Pi 8Co 7Tr 3Co
Main 1 gagne avec paire de 10
```

04

Répartition du travail



Répartition du travail

Travail en slices qui nous a permis de tous toucher à l'ensemble du projet :

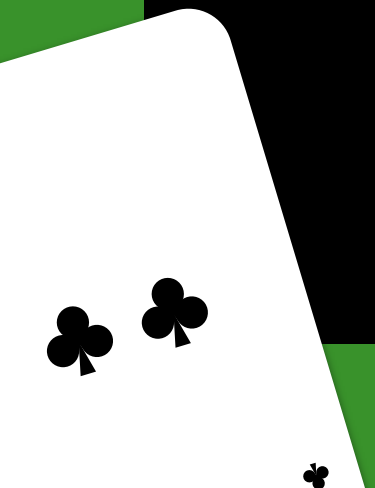
- Implémentation des règles
- Ecriture de tests unitaires
- Comparaison des mains

Responsabilités spécifiques

Amacine : Gestion des exceptions
Yahia : Refactoring global
Rafia : Règles avancées
Ahmed : Consolidation des règles et des tests

05

Démo



Merci pour votre attention !

Avez-vous des questions ?