POLYTECH NICE SOPHIA | UNIVERSITÉ CÔTE D'AZUR

Equipe A – 512 Bank

Présentation du projet

PS5 – Dojo - La main de Poker

Fonctionnalités

Fonctionnalités réalisées	Fonctionnalités non réalisées
Gestion de chacune des mains (Détection et comparaison)	Passage de la main au jeu de poker
Détection des doublons lors de l'entrée	
Affichage de la raison de la victoire	

Détection des doublons

```
Main 1: 3Pi 4Pi 3Co 3Pi VTr
Veuillez entrer des cartes différentes (carte déjà utilisée: 3 de Pique)
```

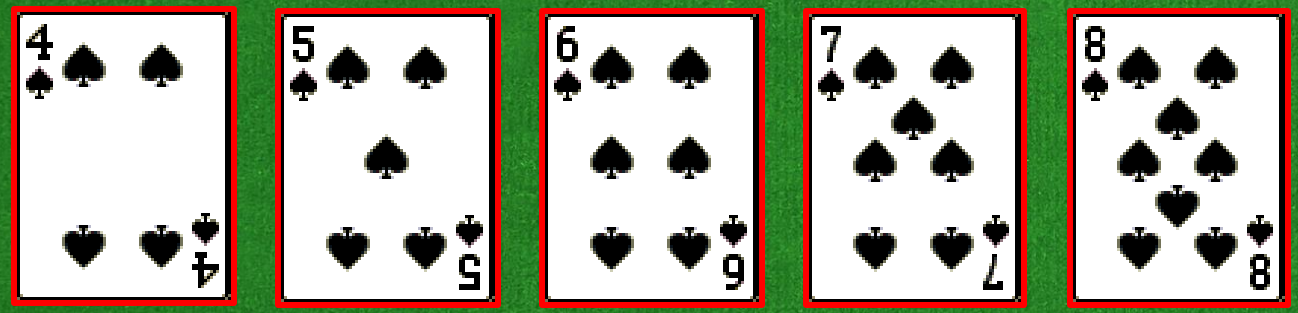
Raison de la victoire

```
Main 1: 3Pi 4Pi 3Co 3Tr VTr
Main 2: 8Pi 5Tr 10Ca 2Co 10Tr
La main 1 gagne avec Brelan de 3
```

Démo

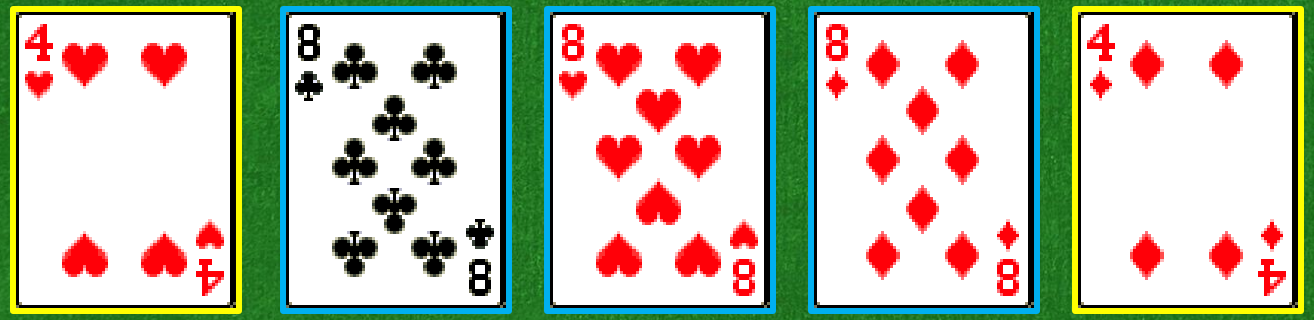
Le cas de la victoire

- Pour commencer, une quinte flush :



= 4Pi 5Pi 6Pi 7Pi 8Pi

- Contre un full

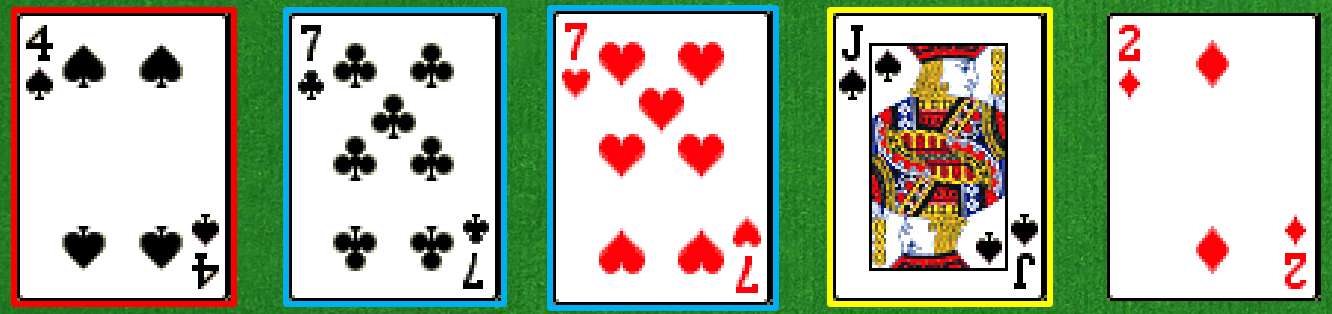


= 4Co 8Tr 8Co 8Ca 4Ca

Démo

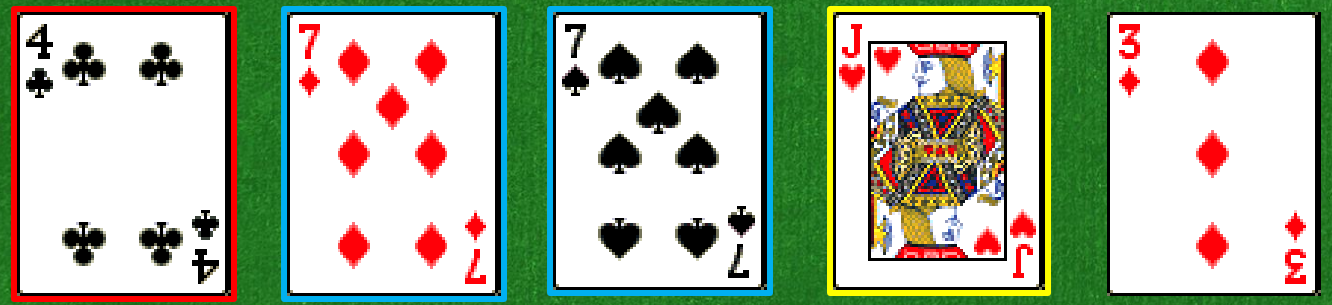
Le cas de l'égalité

- Pour commencer, une paire de 7:



= 4Pi 7Tr 7Co VPi 2Ca

- Contre une autre paire de 7

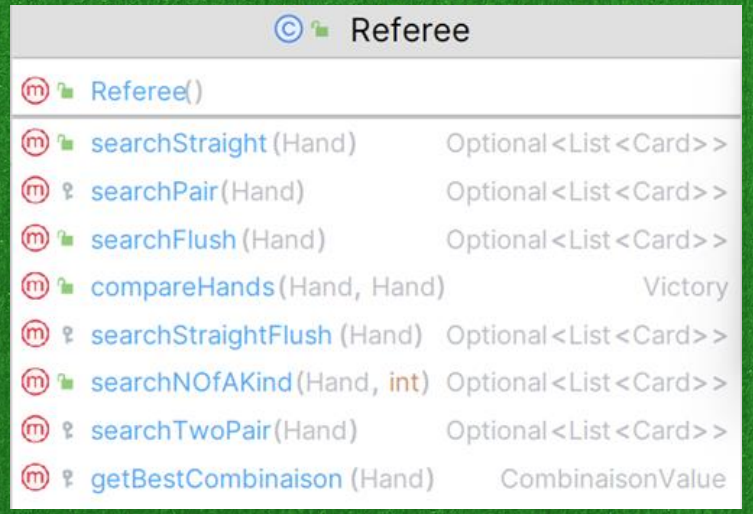


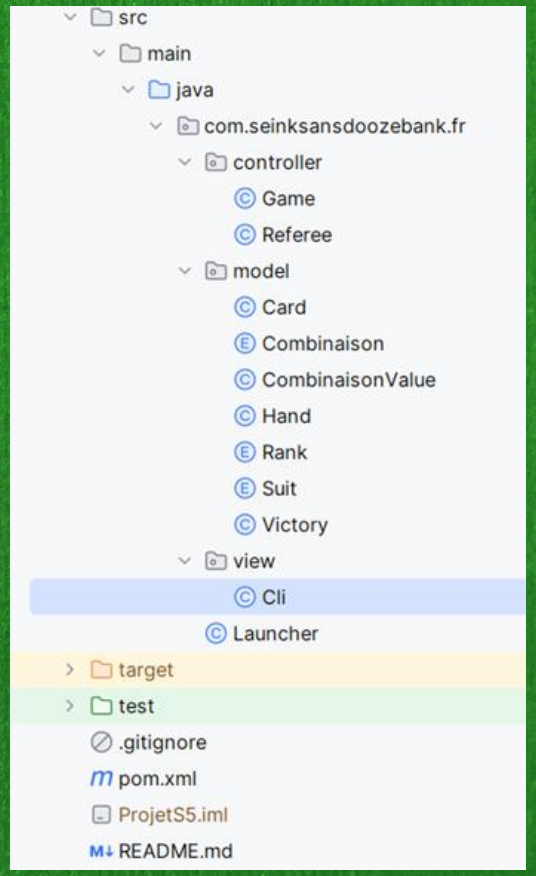
= 4Tr 7Ca 7Pi VCo 3Ca

Forces de notre projet

- Lisibilité du code source
- Architecture du projet
- Enum Rank, Suit

```java
public void run() {
    for (int i = 0; i < this.nbrOfHands; i++) {
        List<String> cardsHand = view.initializeHand( numHand: i + 1);
        while (!checkInput(cardsHand)) {
            cardsHand = view.initializeHand( numHand: i + 1);
        }
        Hand hand = new Hand(cardsHand);
        hands.add(hand);
    }
    Victory winner = referee.compareHands(hands.get(0), hands.get(1));
    if (winner == null) {
        view.displayDraw();
    } else {
        view.displayWinner(winner);
    }
}
```





Axes d'amélioration

Dans le code **source** :

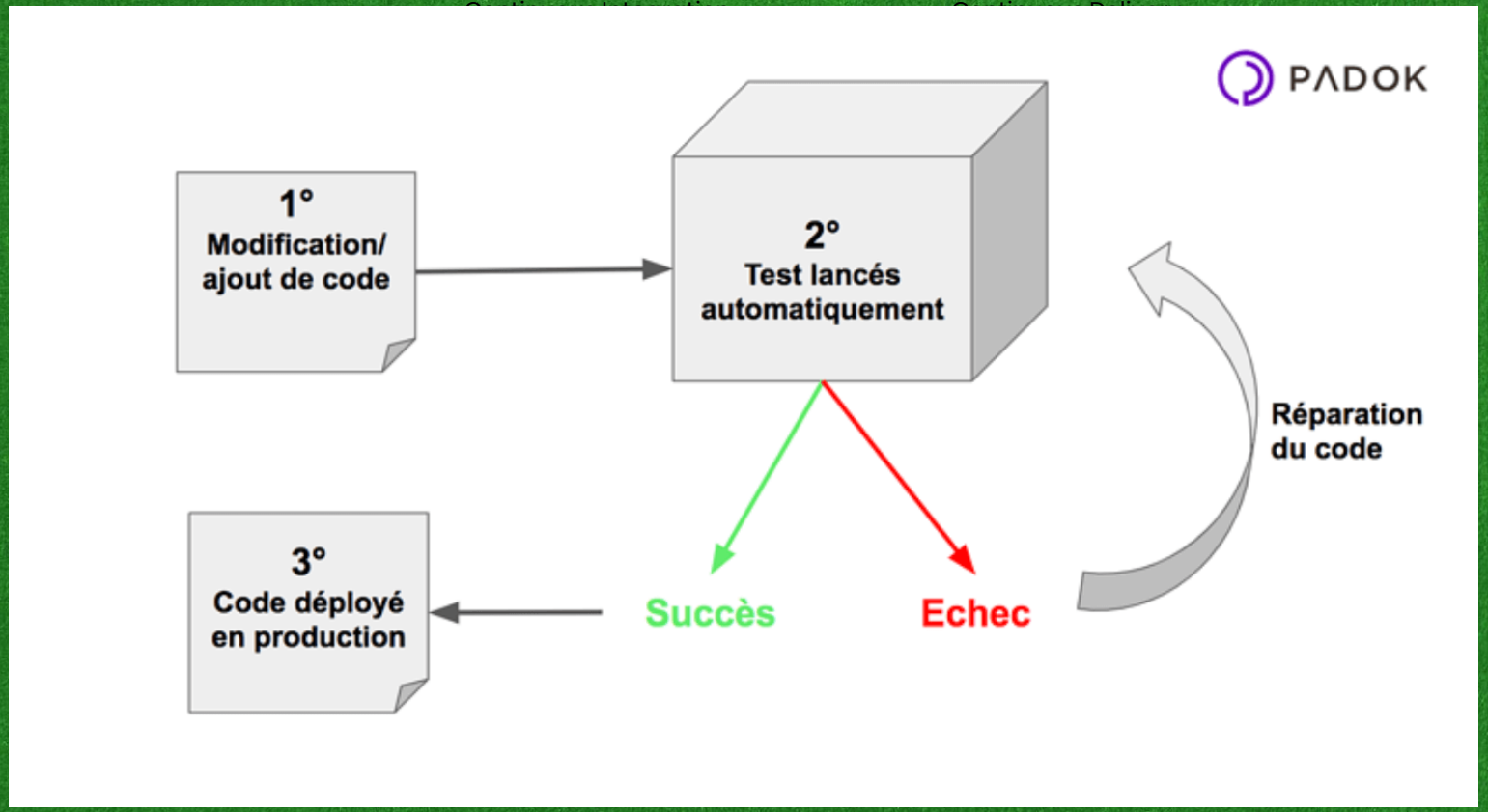
- Améliorer la méthode checkInput pour qu'elle renvoie une liste de cartes
- Mise à jour du code *compareTo*
- Revoir la méthode *compareHand*

Mais aussi dans la partie Test :

- Réorganisation des tests pour chaque combinaison
- Renommage des tests, des variables pour plus de lisibilité.
- Utilisation de la méthode compareTo qui cache deux autres méthodes

Tests et Intégration continue

Code → Build → Test → Release → Deploy



Coverage et Tests

Element	Class, %	Method, %	Line, %
com	91% (11/12)	90% (69/76)	89% (355/396)
seinksansdoozebank	91% (11/12)	90% (69/76)	89% (355/396)
fr	91% (11/12)	90% (69/76)	89% (355/396)
controller	100% (2/2)	92% (13/14)	91% (128/140)
Game	100% (1/1)	80% (4/5)	70% (24/34)
Referee	100% (1/1)	100% (9/9)	98% (104/106)
model	100% (8/8)	96% (53/55)	92% (223/242)
Card	100% (1/1)	100% (8/8)	85% (12/14)
Combinaison	100% (1/1)	60% (3/5)	85% (12/14)
CombinaisonValue	100% (2/2)	100% (17/17)	90% (130/143)
Hand	100% (1/1)	100% (7/7)	95% (23/24)
Rank	100% (1/1)	100% (7/7)	96% (25/26)
Suit	100% (1/1)	100% (7/7)	100% (15/15)
Victory	100% (1/1)	100% (4/4)	100% (6/6)
view	100% (1/1)	50% (3/6)	40% (4/10)
Cli	100% (1/1)	50% (3/6)	40% (4/10)
Launcher	0% (0/1)	0% (0/1)	0% (0/4)

- Points forts
  - Respect du principe Right-BICEP

- **Principe Right-BICEP**
  - Right : est-ce que les résultats sont corrects ?
  - B (Boundary) : est-ce que les conditions aux limites sont correctes ?
  - I (Inverse) : est-ce que l'on peut vérifier la relation inverse ?
  - C (Cross-check) : est-ce que l'on peut vérifier le résultat autrement ?
  - E (Error condition) : est-ce que l'on peut forcer l'occurrence d'erreurs ?
  - P (Performance) : est-ce que les performances sont prévisibles ?

- Points faibles
  - Nommage des variables et des fonctions

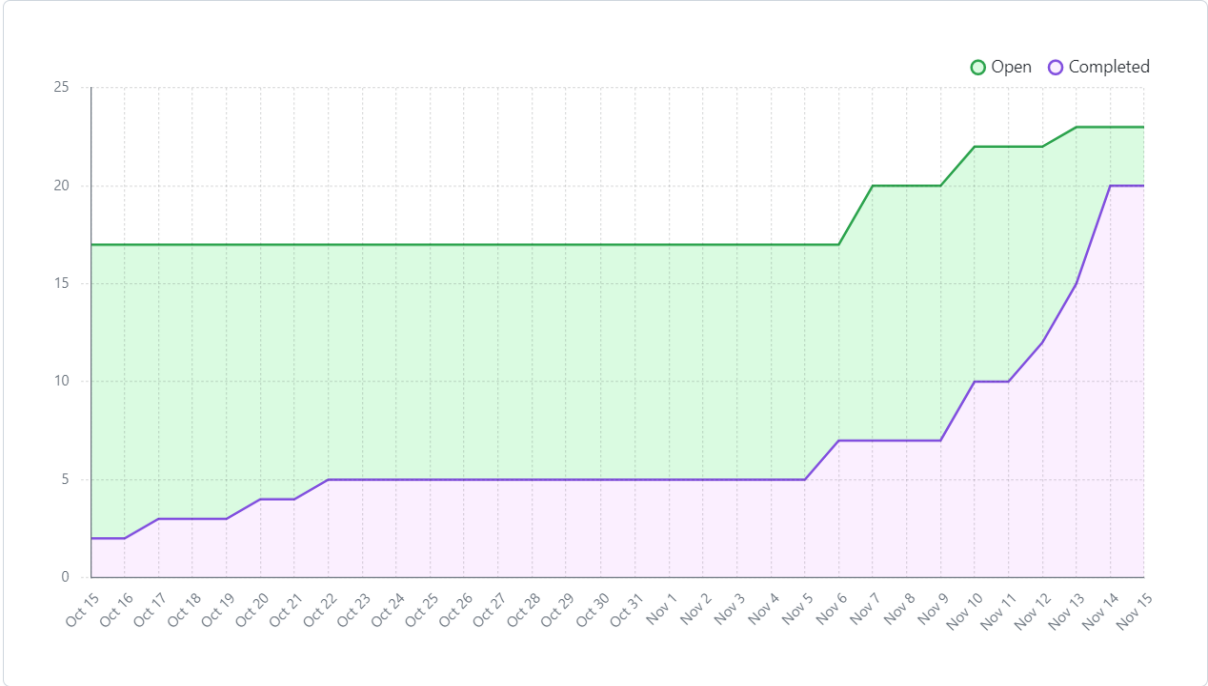
```java
@Test
void compareHandsFirstWin() {
    hand1 = new Hand(new ArrayList<>(List.of("ACa", "2Ca", "4Ca", "5Ca", "6Ca")));
    hand2 = new Hand(new ArrayList<>(List.of("9Co", "2Co", "4Co", "5Co", "6Co")));
    victory = referee.compareHands(hand1, hand2);
    assertEquals(hand1, victory.getHand());
}
```

Organisation du projet

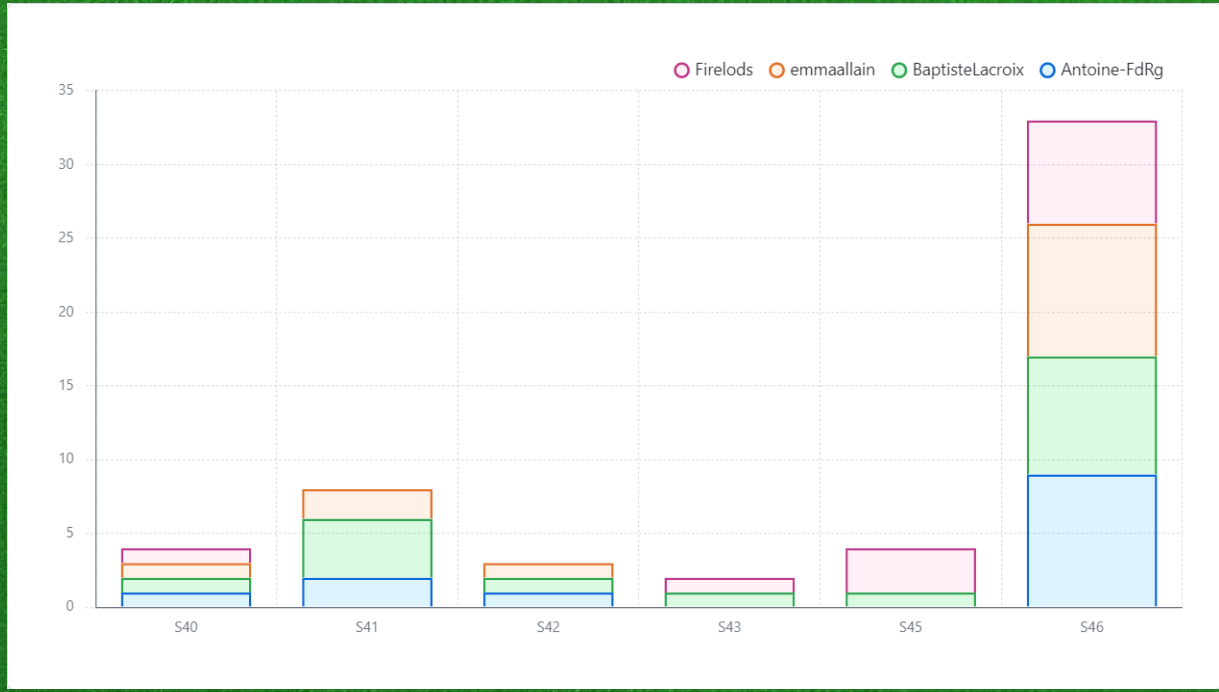
Répartition des issues en fonction des membres de l'équipe et des labels affiliés

Organisation du projet

Évolution du statut des issues
en fonction du temps



Réalisation des issues en fonction des membres
de l'équipe et des différentes semaines



Des questions ?