Analyse, conception et implémentation d'une application

Analyse et Conception de Logiciels

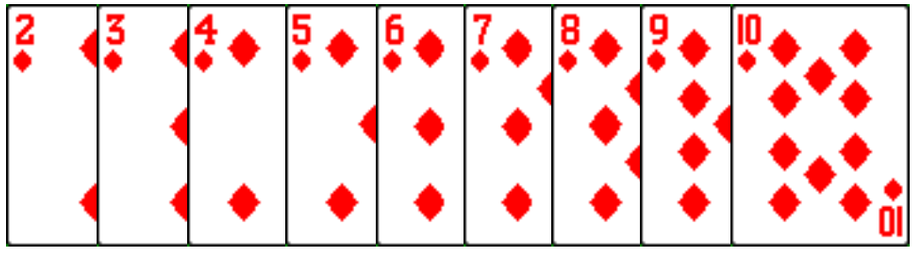
Horatiu CIRSTEA

Application JacquesNoir

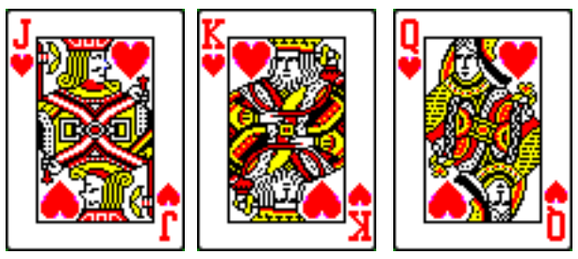
- on souhaite réaliser une application pour jouer au jeu de cartes « JacquesNoir »
- le but du jeu est d'obtenir un maximum de points sans dépasser 21 points
- un joueur joue contre la banque

Valeur cartes

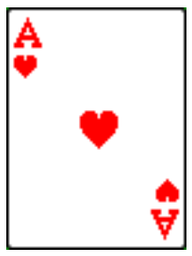
- chaque carte numérotée vaut sa valeur faciale



- chaque figure vaut 10 points



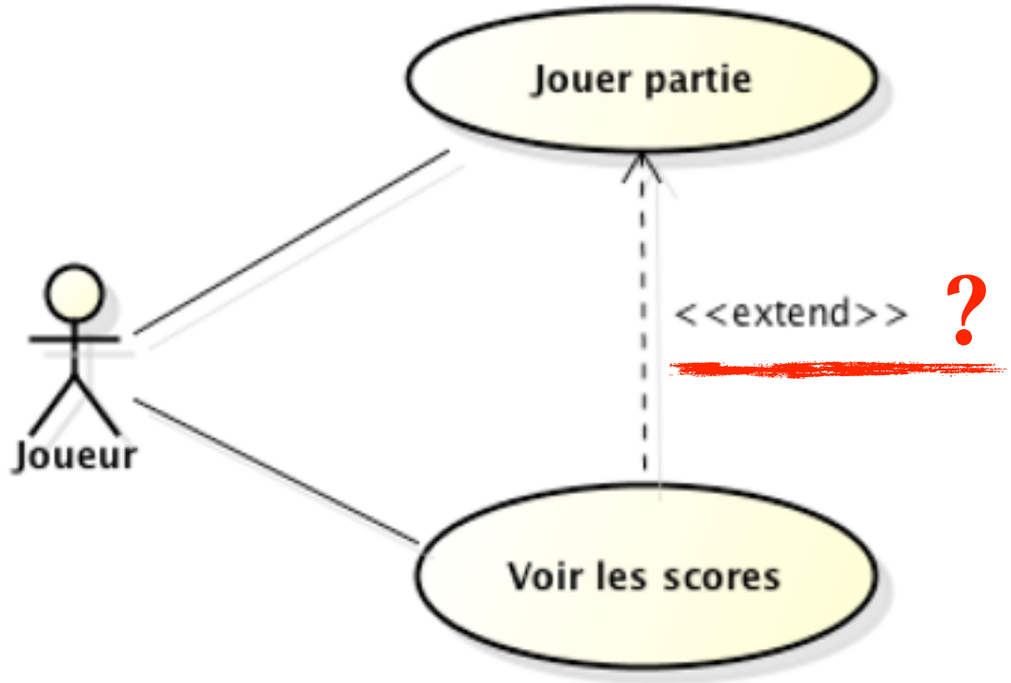
- le As vaut 11 points



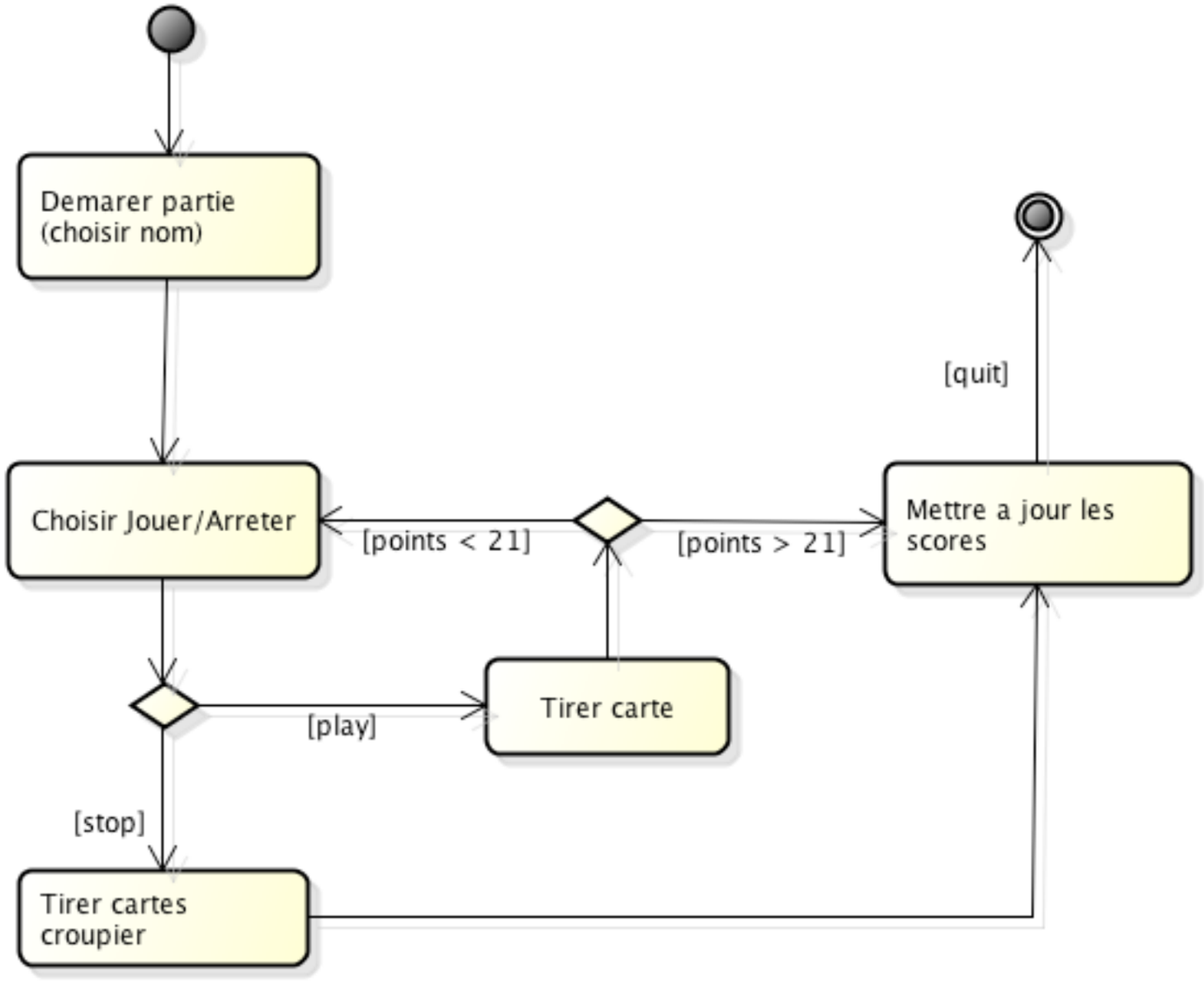
Règles JacquesNoir

- le jouer tire des cartes jusqu'à ce qu'il décide de s'arrêter ou qu'il totalise plus de 21 points
- s'il totalise plus de 21 points, le joueur a perdu
- sinon, la banque tire des cartes jusqu'à ce qu'il dépasse le joueur ou qu'il totalise minimum 21 points
- si la banque a plus de 21 points, il perd
- sinon, si le joueur et la banque ont le même nombre de points, personne gagne
- sinon, le joueur gagne

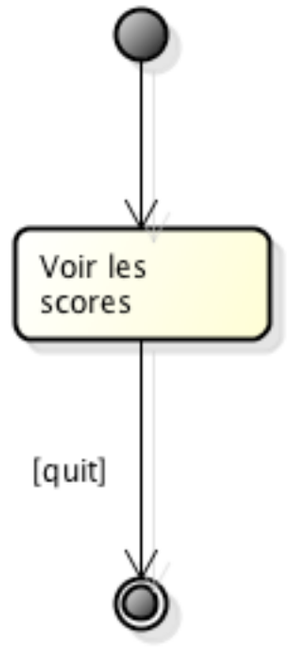
Fonctionnalités JacquesNoir



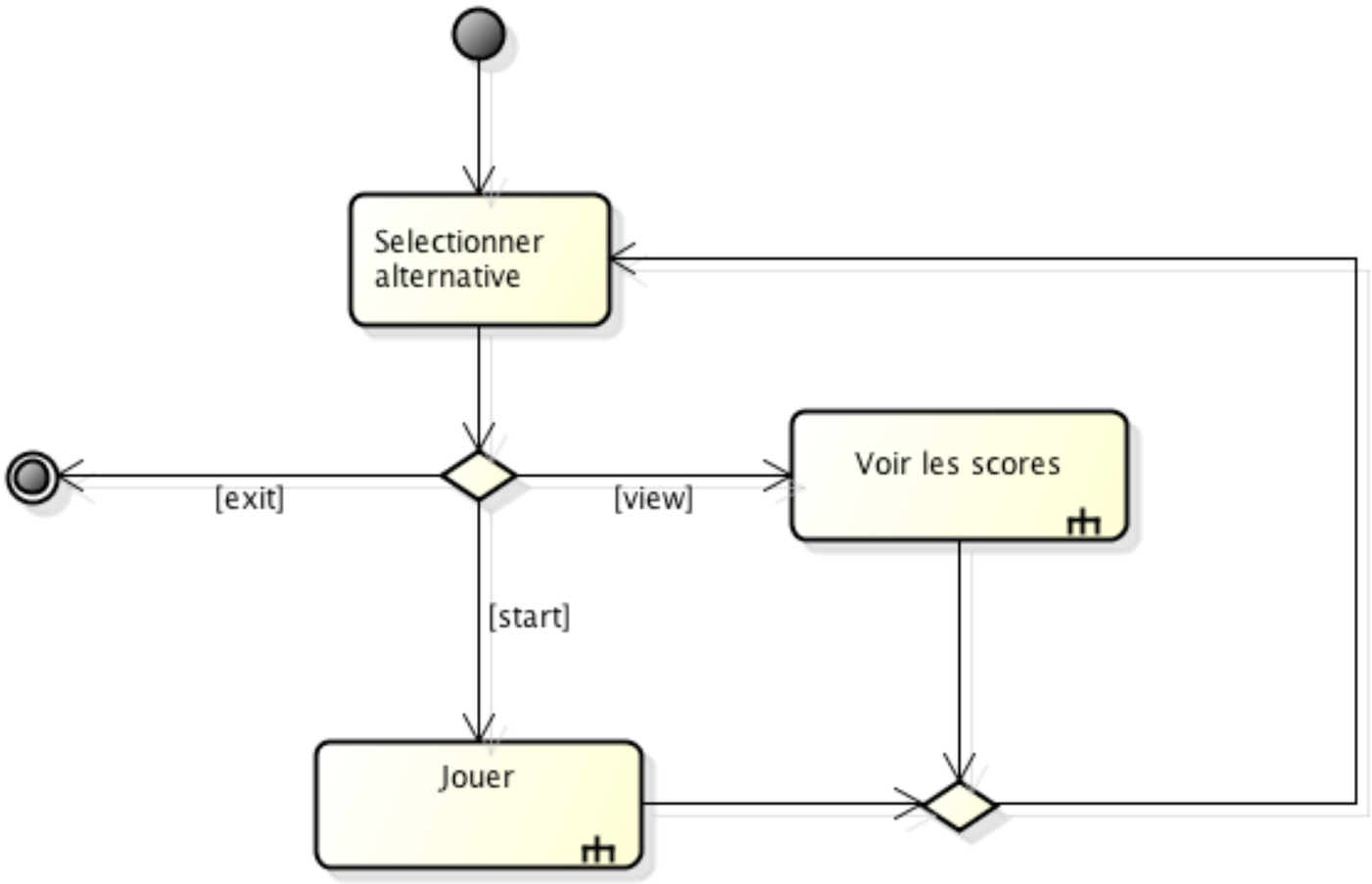
Scénario Jouer partie



Scénario Voir les scores

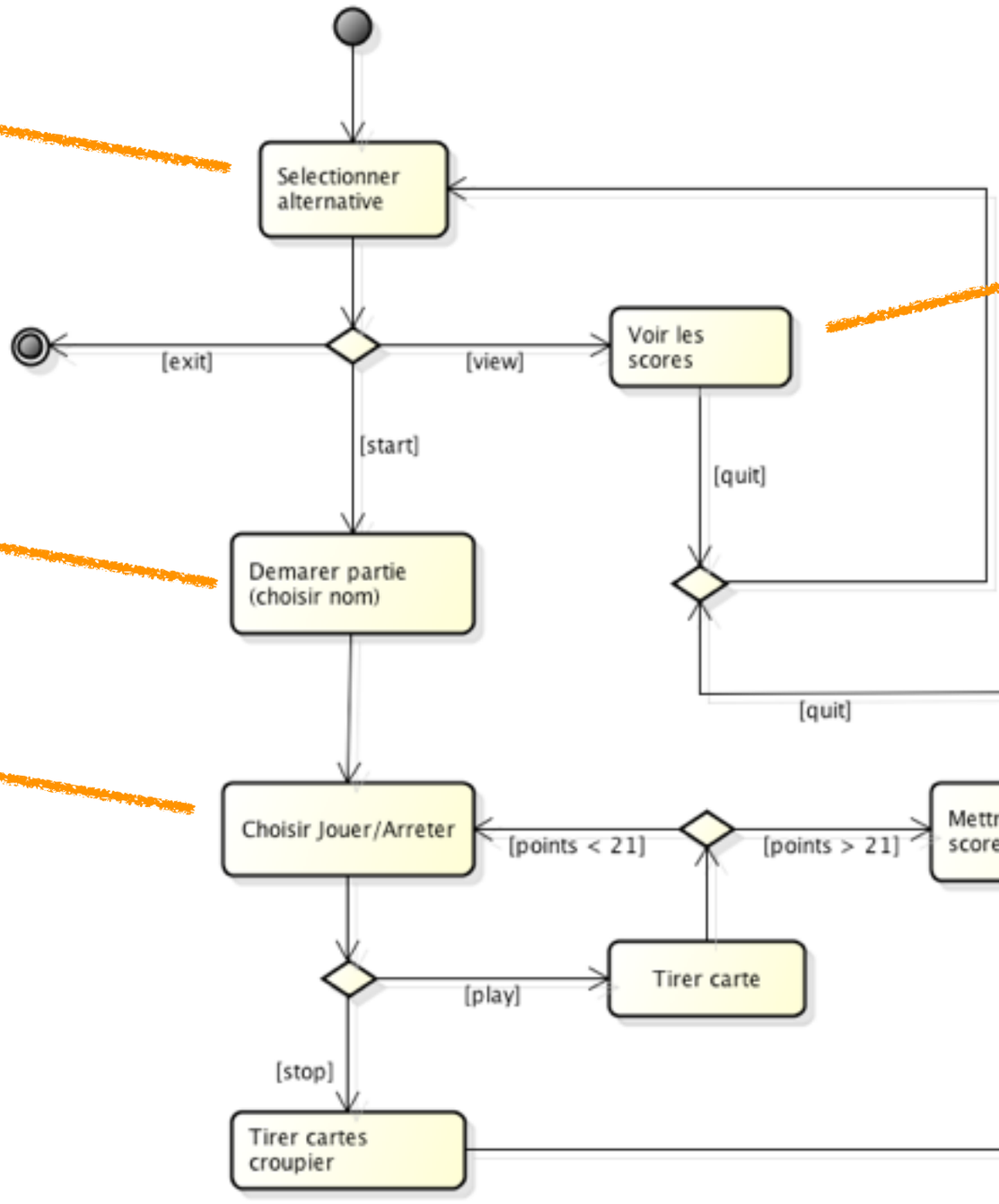


Déroulement JacquesNoir



Déroulement JacquesNoir

```
1: Start
2: See scores
3: Exit
```



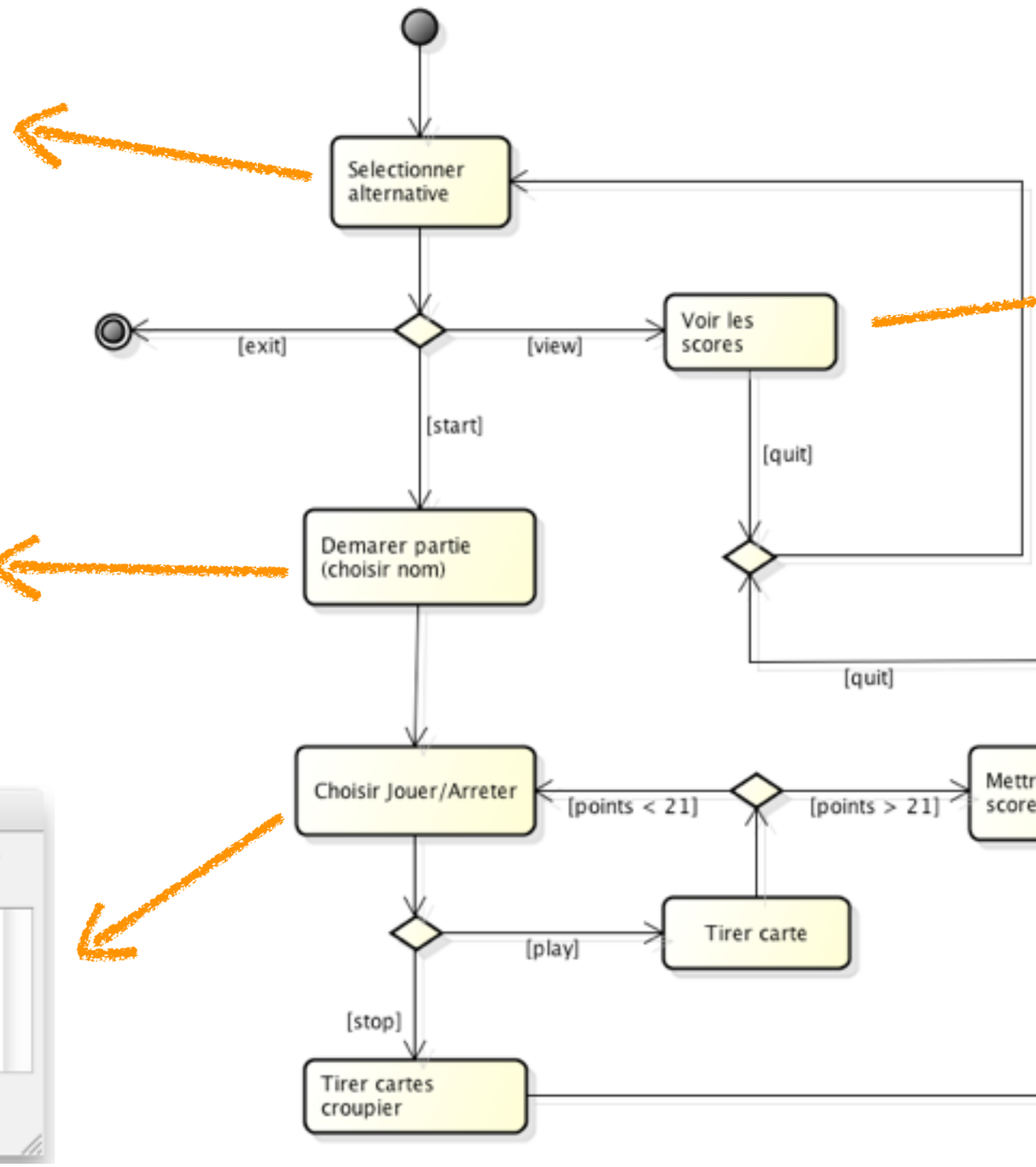

```
James : -1
Vincent : 1
```

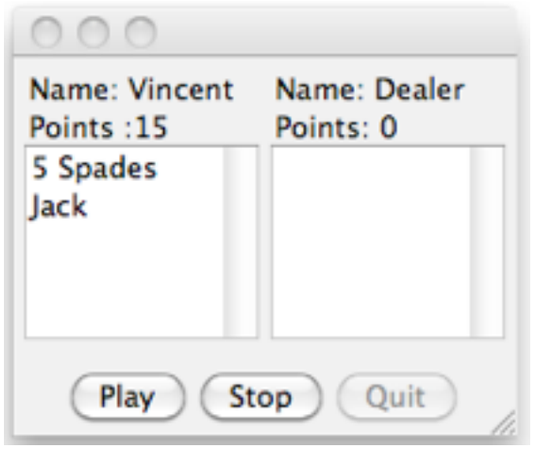
```
Name: James
```

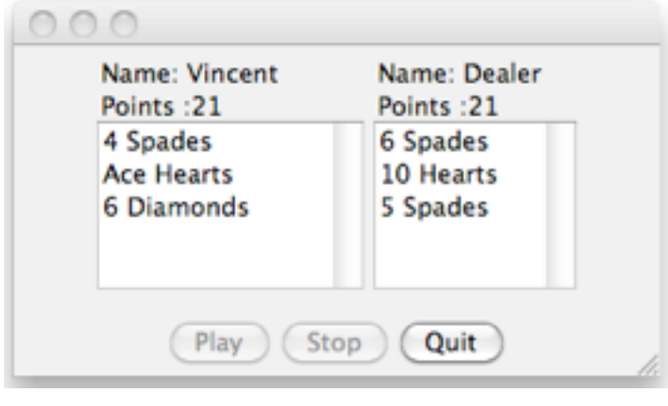
```
Player [3 Spades]
1: Play
2: Stop
```

```
Player [3 Spades, 6 Diamonds, 4 Diamonds, 8 Diamonds] : 21 points
Dealer [3 Hearts, 2 Spades, 4 Hearts, 9 Diamonds, 7 Spades] : 25 points
```

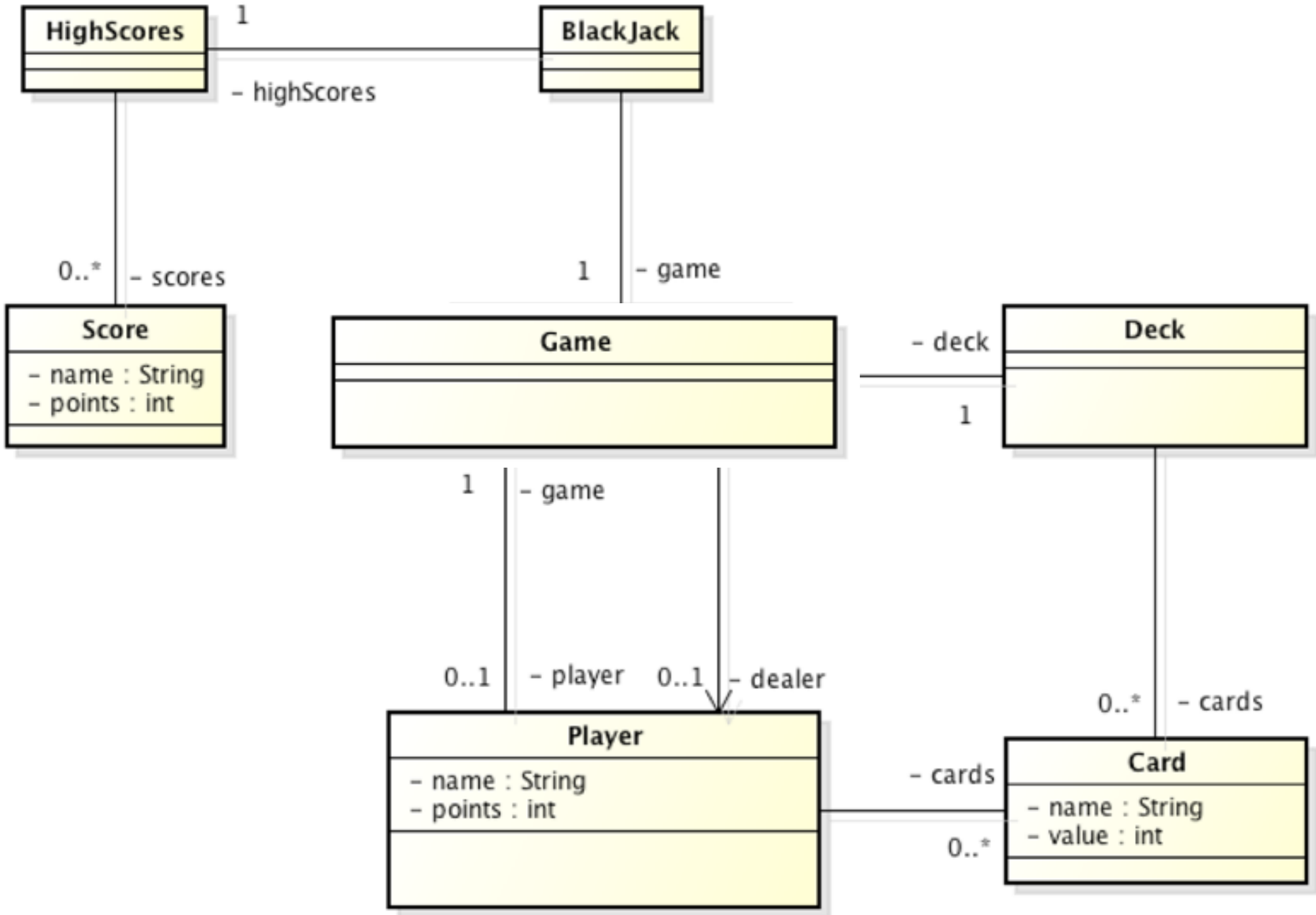
Déroulement JacquesNoir



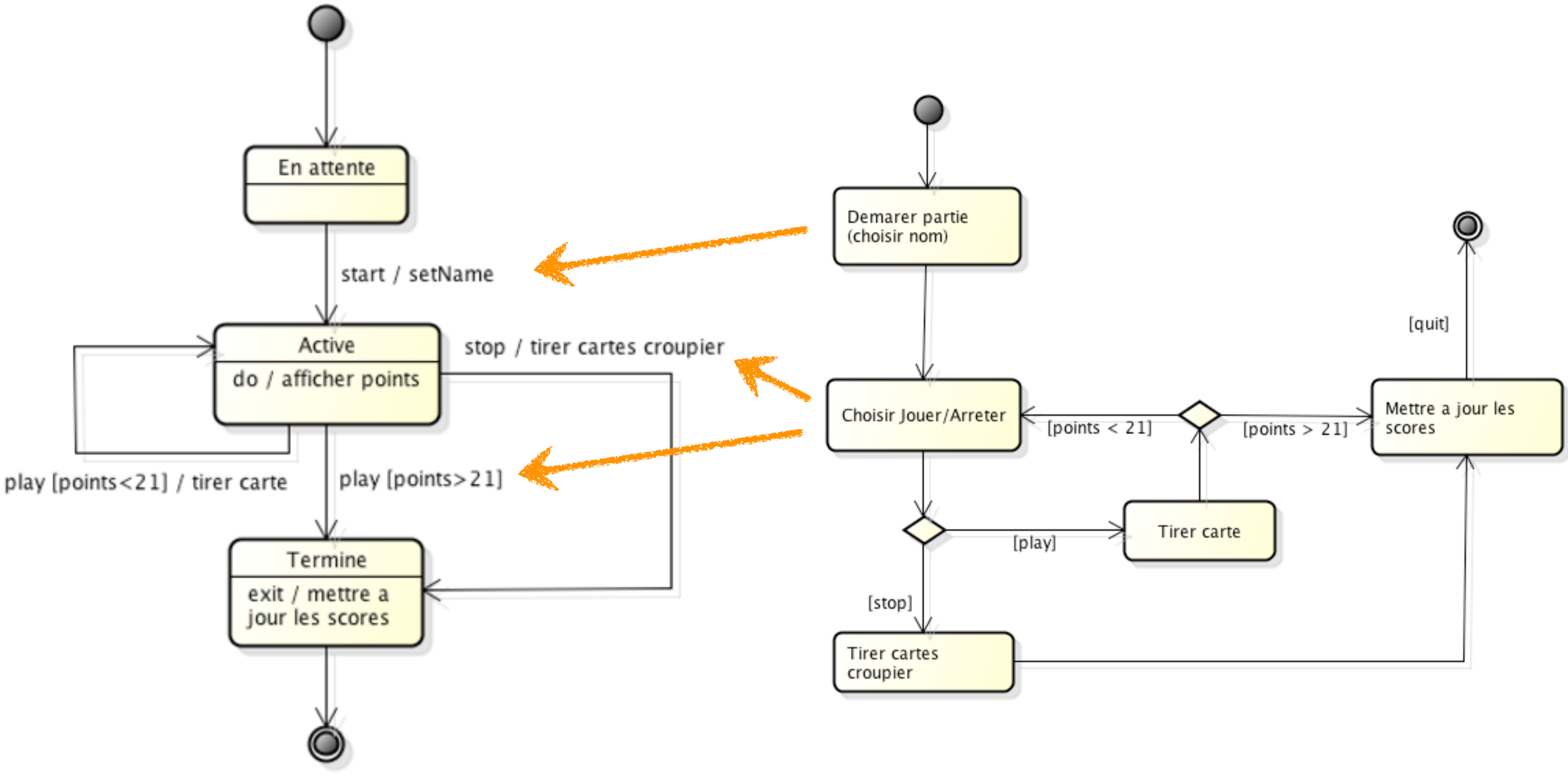




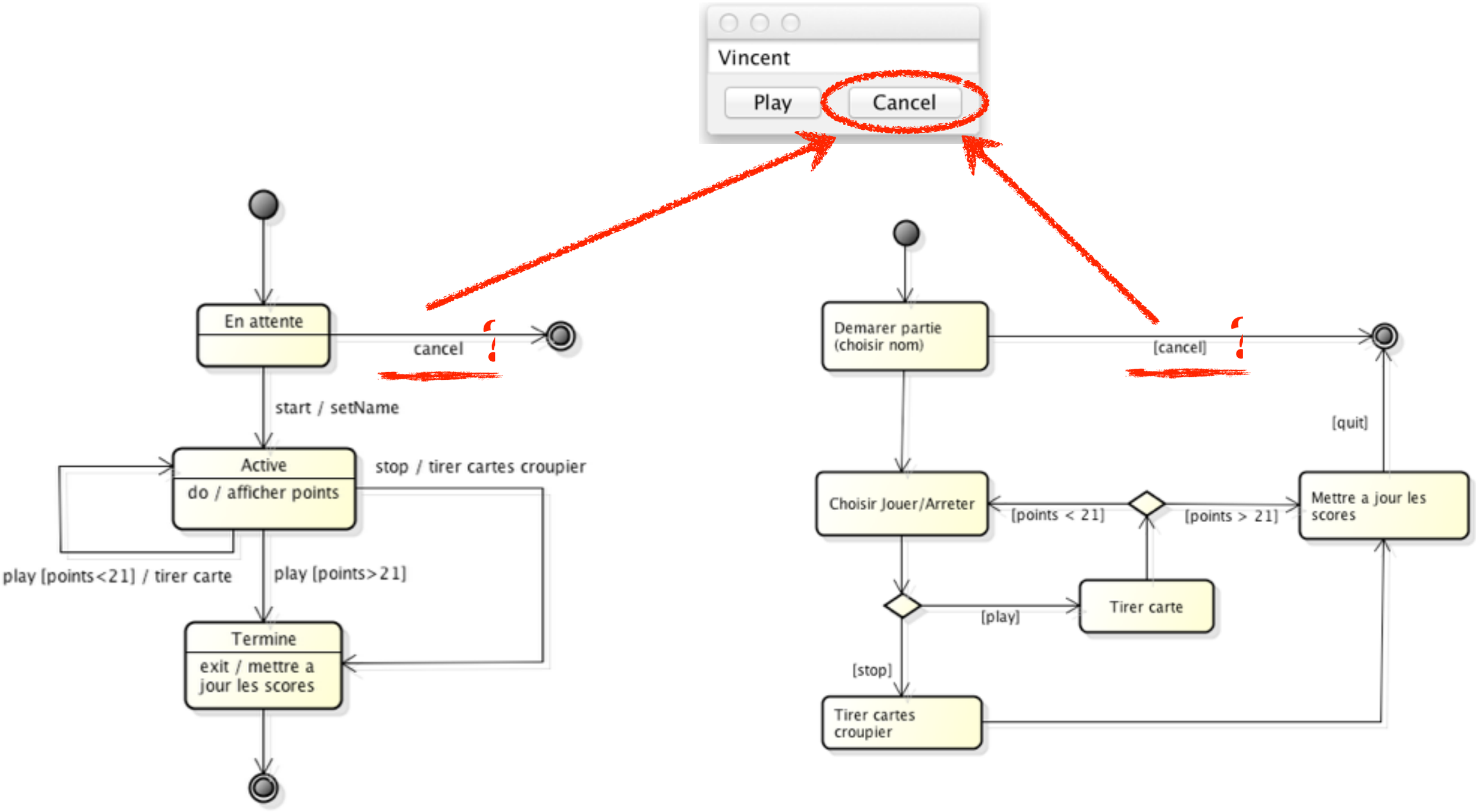
Analyse (statique) du domaine



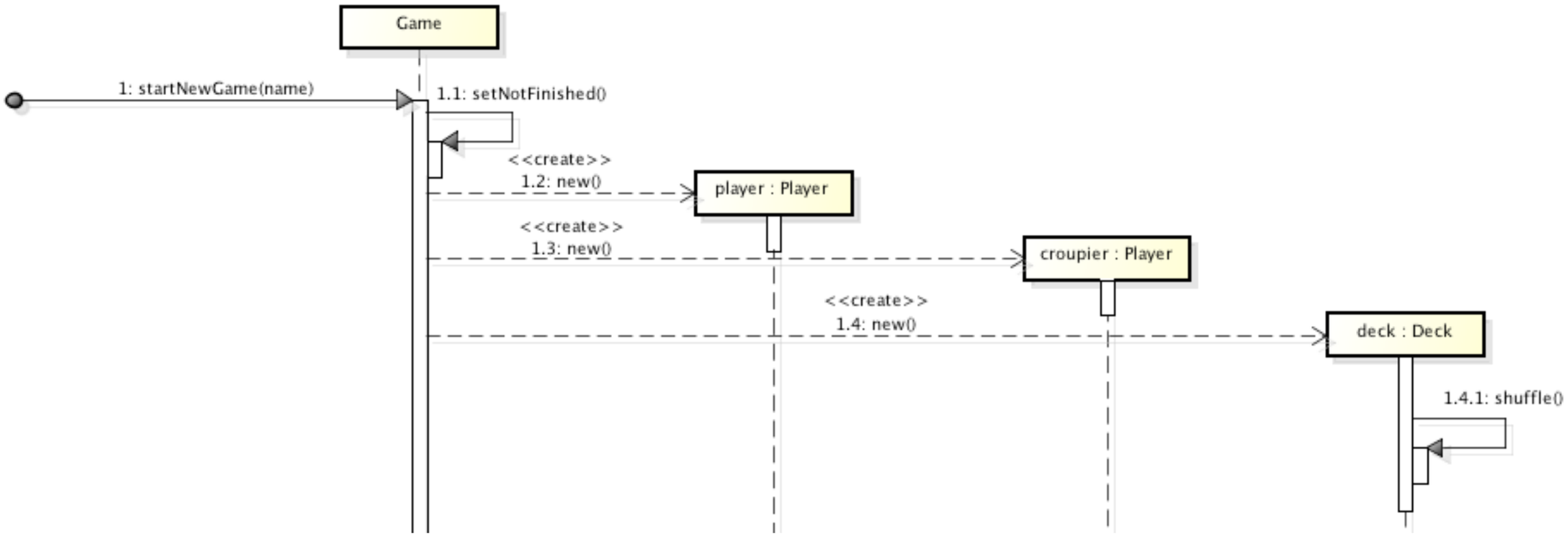
Vérifier cohérence

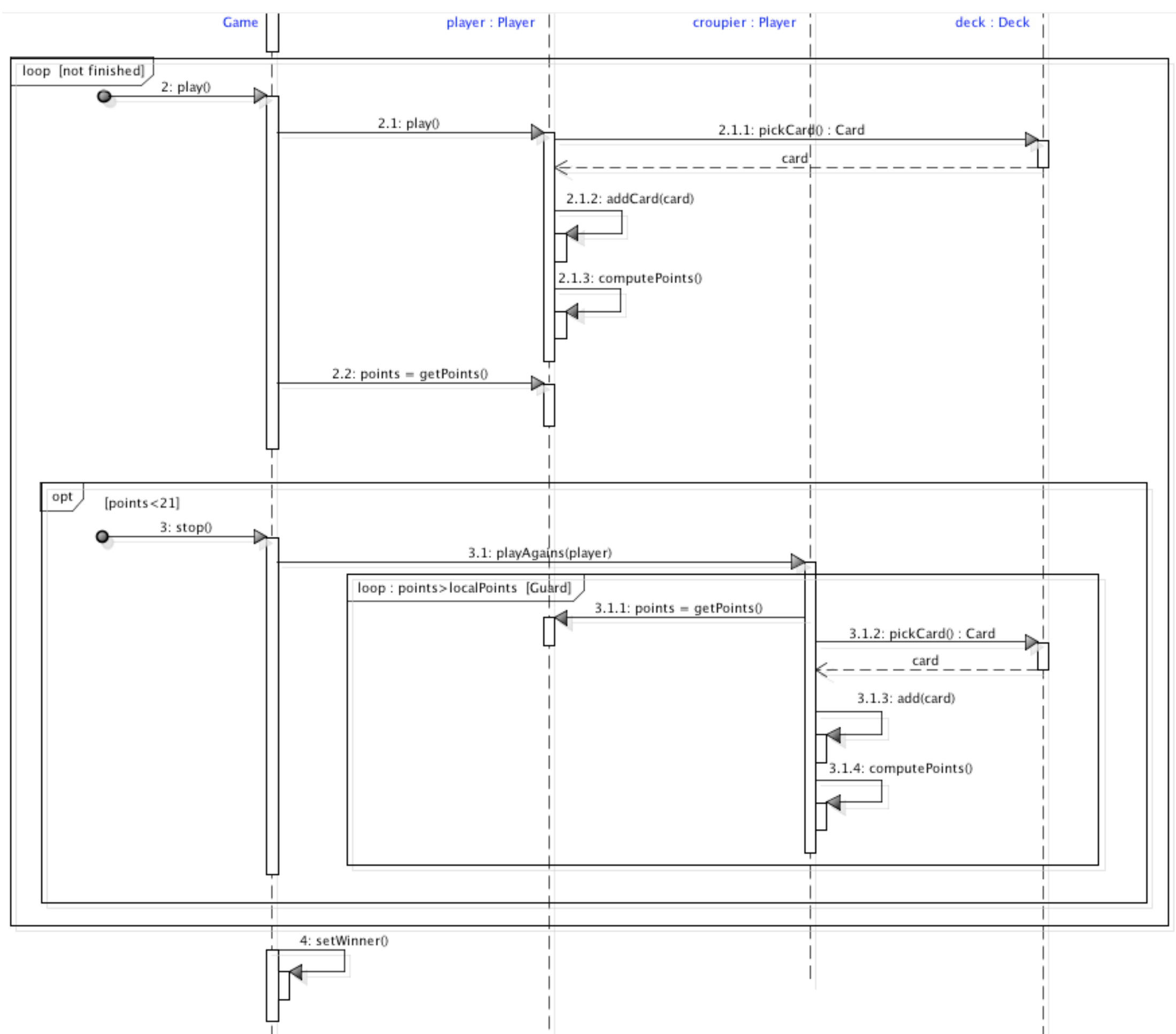


Vérifier cohérence

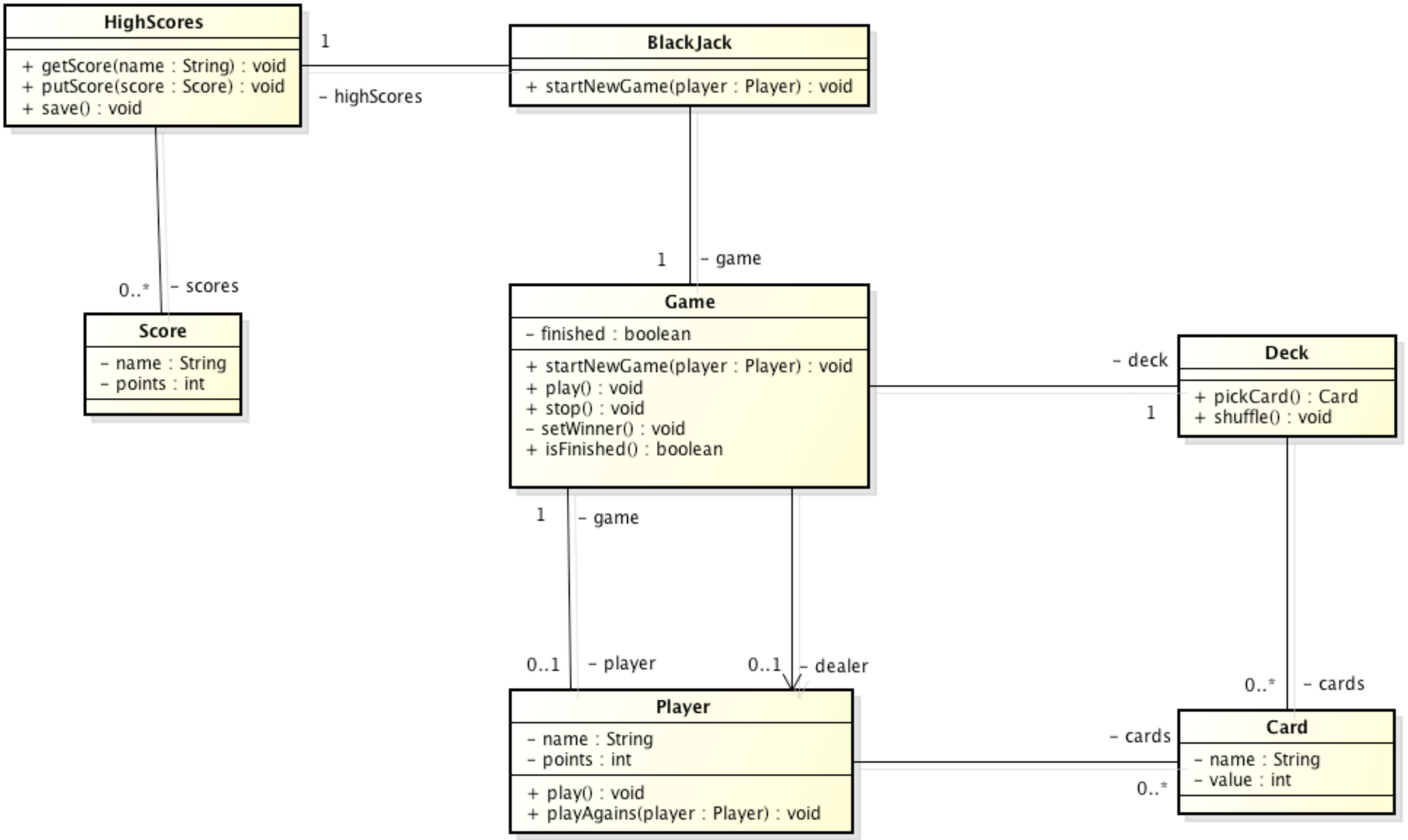


Analyse (dynamique)

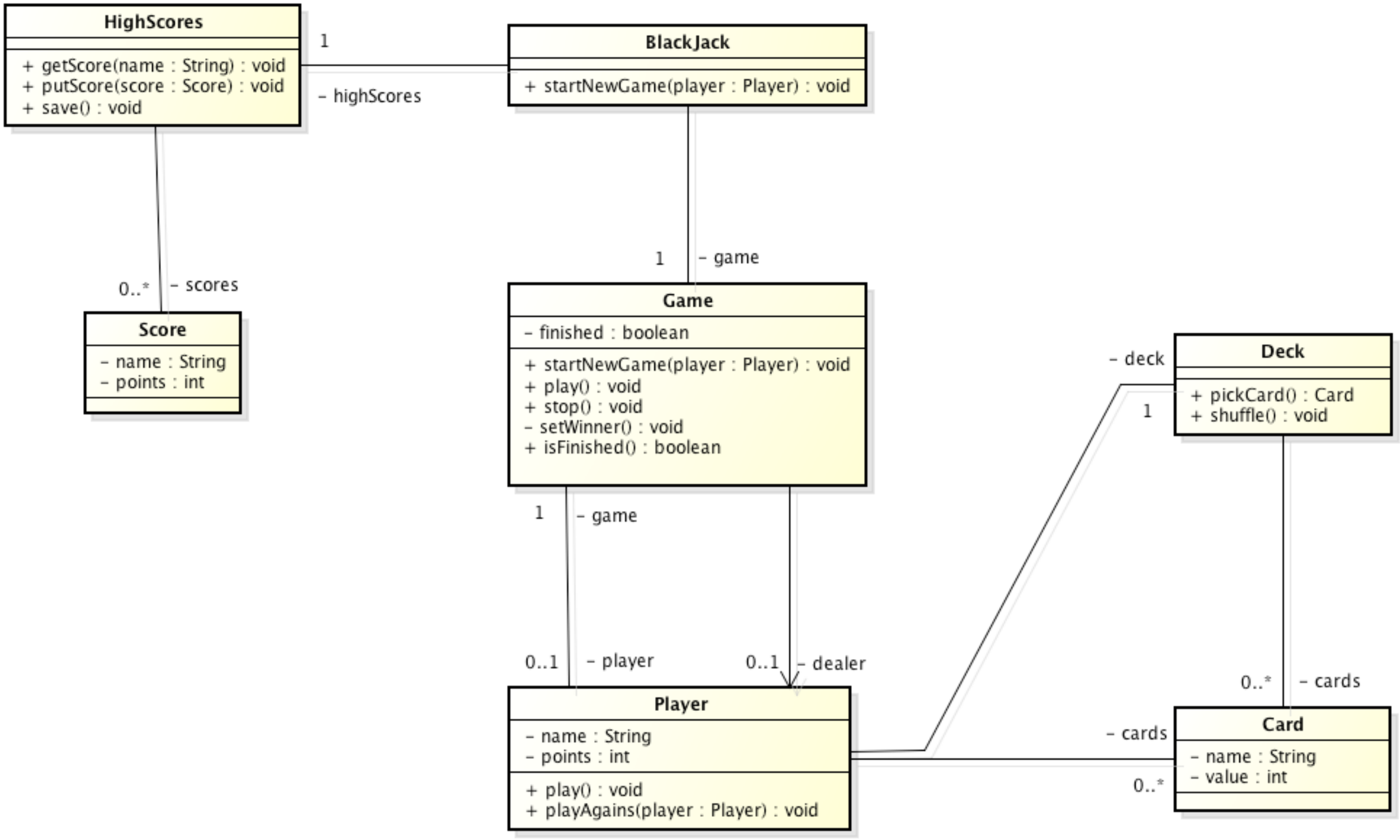



Game
player : Player
croupier : Player
deck : Deck
loop [not finished]
2: play()
2.1: play()
2.1.1: pickCard() : Card
card
2.1.2: addCard(card)
2.1.3: computePoints()
2.2: points = getPoints()
opt
[points<21]
3: stop()
3.1: playAgains(player)
loop : points>localPoints [Guard]
3.1.1: points = getPoints()
3.1.2: pickCard() : Card
card
3.1.3: add(card)
3.1.4: computePoints()
4: setWinner()

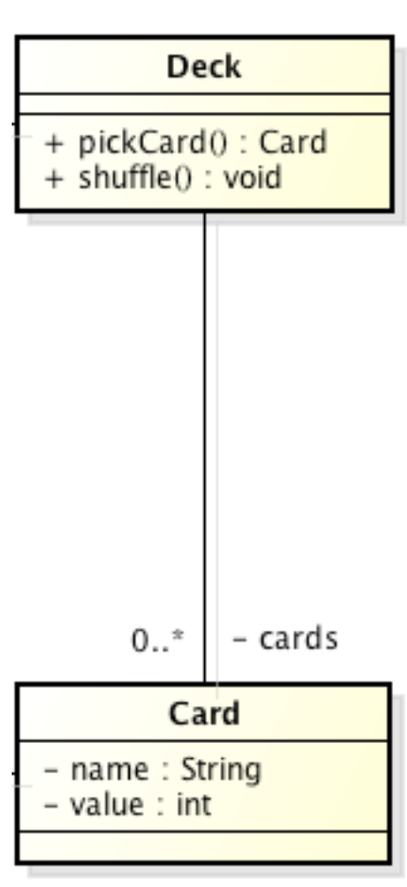
Analyse (statique)



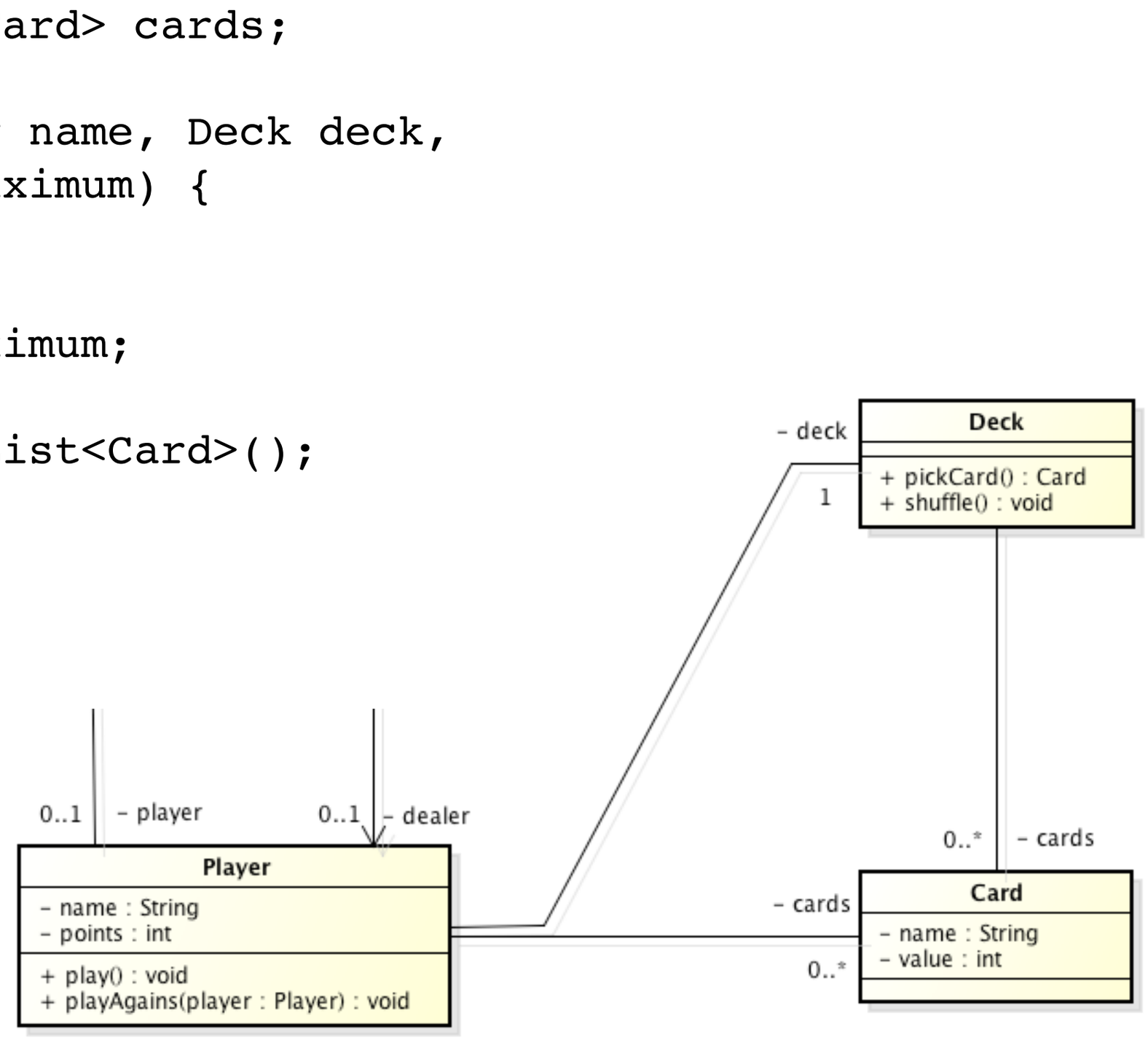
Alternative

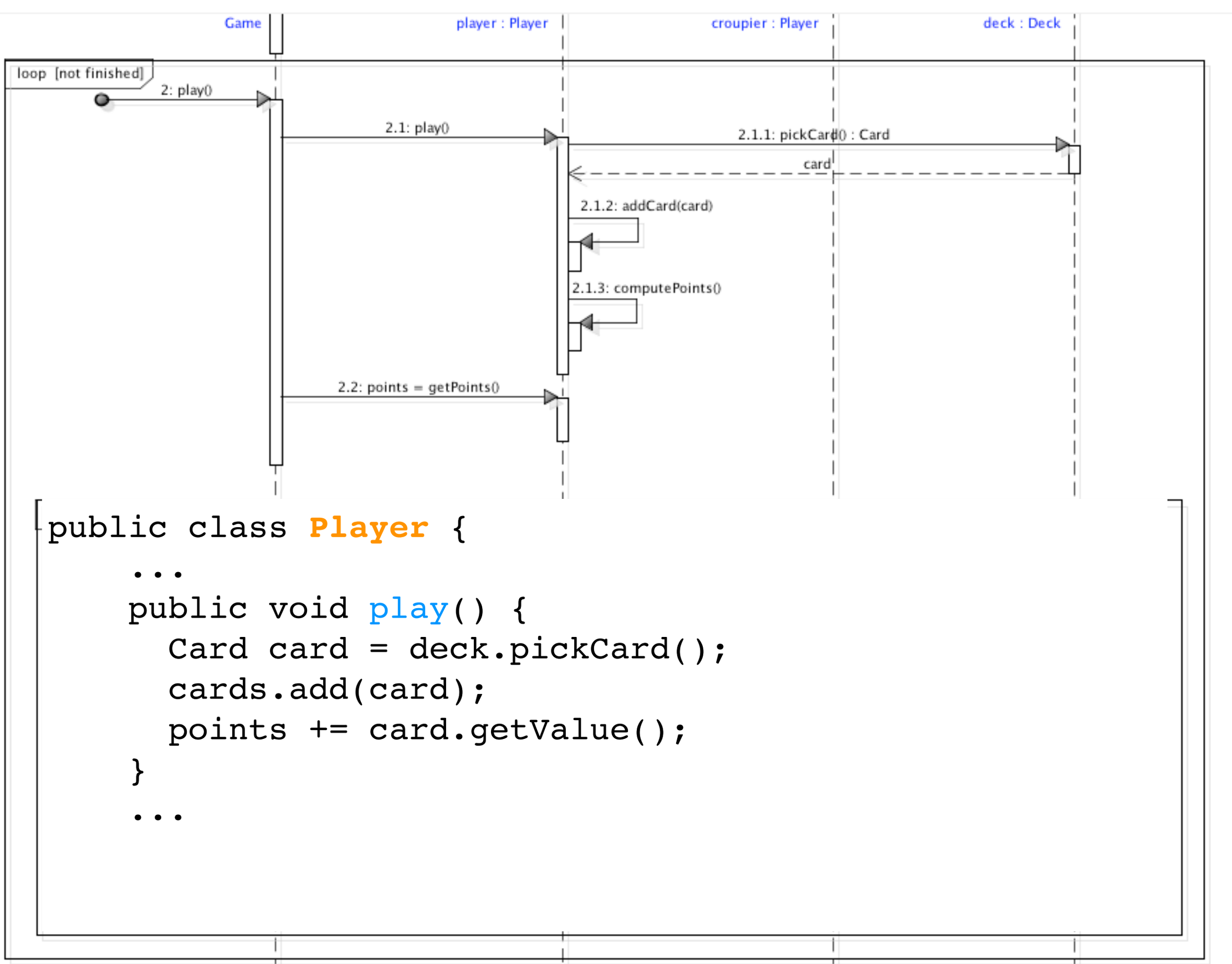


```
public class Deck {
    private static final int SIZE = 32;
    private final List<Card> cards;
    private int nbCards;
    private Random random;
    public Deck() {
      nbCards = SIZE;
      cards = new ArrayList<Card>(SIZE);
      cards.add(new Card("2 Diamonds", 2));
      ...
      cards.add(new Card("Ace Clubs", 11));
    }

    public Card pickCard() {
      int extract = random.nextInt(this.nbCards--);
      return cards.remove(extract);
    }

    public void shuffle() {
      this.random = new Random();
    }
}
```

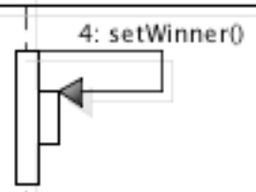


```
public class Player {
    private final String name;
    private int points;
    private final int maximum;  //=Deck.SIZE
    private final Deck deck;
    private final List<Card> cards;

    public Player(String name, Deck deck,
                  int maximum) {
      this.name = name;
      this.points = 0;
      this.maximum = maximum;
      this.deck = deck;
      cards = new ArrayList<Card>();
}
...
```

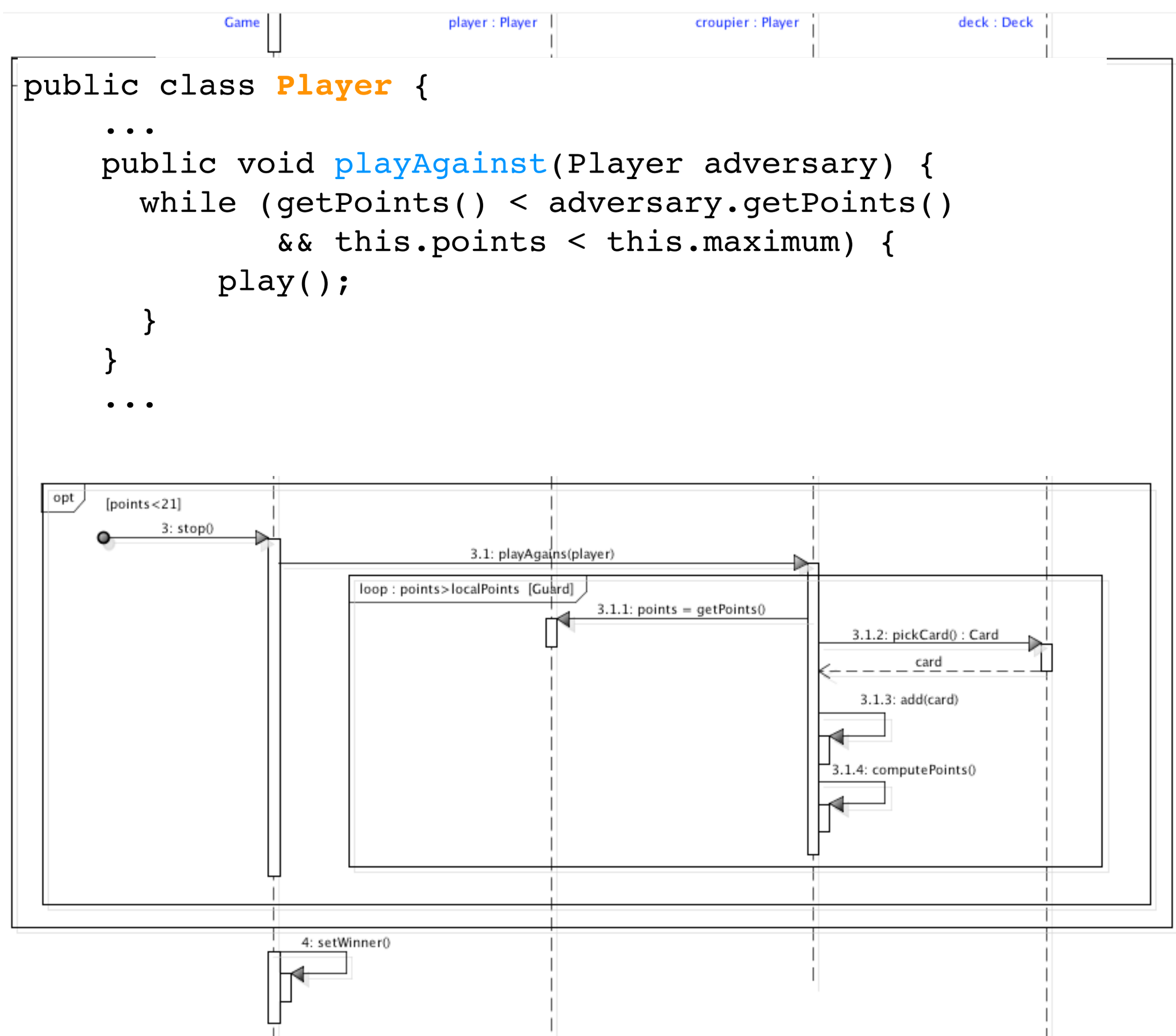


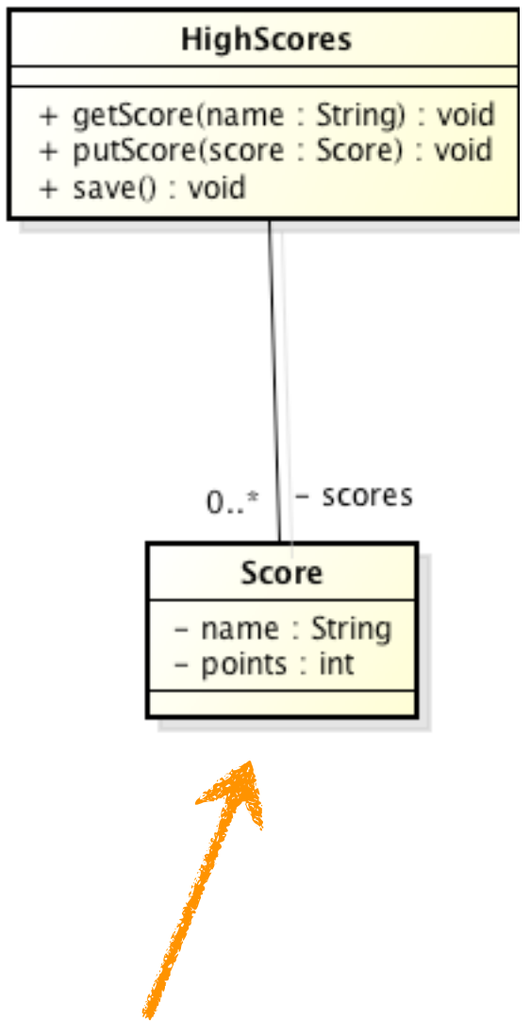



```
public class Player {
    ...
    public void play() {
      Card card = deck.pickCard();
      cards.add(card);
      points += card.getValue();
    }
    ...
```



```
public class Player {
    ...
    public void playAgainst(Player adversary) {
      while (getPoints() < adversary.getPoints()
              && this.points < this.maximum) {
          play();
      }
    }
    ...
```





Immuable

```
public class HighScores  {
    private final Map<String, Integer> scores;
    public HighScores() {
      scores = new HashMap<String, Integer>();
    }

    public List<Score> getScores() {
      List<Score> entries = new ArrayList<Score>();
      for (String name : scores.keySet()) {
        entries.add(new Score(name, scores.get(name)));
      }
      return entries;
    }

    public void updateScore(Score score) {
      int currentPoints=0;
      if (scores.get(score.getName()) != null) {
         currentPoints = scores.get(score.getName());
      }
      scores.put(score.getName(),
      currentPoints+score.getPoints());
    }

    public Integer getScore(String name) {
      return scores.get(name);
    }
```

Déroulement JacquesNoir

```
1: Start
2: See scores
3: Exit
```

```
Name: James
```

```
Player [3 Spades]
1: Play
2: Stop
```

```
BlackJack bj = new BlackJack();
boolean finished = false;
while (!finished) {
    System.out.println("1: Start\n"
                     + "2: See scores\n"
                     + "3: Exit");
    int choice = scan.nextInt();
    switch (choice) {
case 1:
    System.out.print("Name: ");
    String playerName = scan.next();
    if (playerName.compareTo("q") != 0){ // cancel
      Game game = bj.startNewGame(playerName);
      playGame(game);
    }
    break;
case 2:
    ...
```

```
1: Start
2: See scores
3: Exit
```

```
Name: James
```

```
Player [3 Spades]
1: Play
2: Stop
```

```
private static void playGame(Game game) {
    Scanner scan = new Scanner(System.in);
    while (!game.isFinished()) {
        System.out.println("1: Play\n"
                + "2: Stop");
        int choice = scan.nextInt();
        switch (choice) {
          case 1:
            game.play();
            System.out.println("Player " +
                    game.getPlayer().getCards());
            break;
          case 2:
            game.stop();
            System.out.println("Player " + ...);
            System.out.println("Dealer " + ...);
            break;
        }
    }
}
```

```
Player [3 Spades, 6 Diamonds, 4 Diamonds, 8 Diamonds] : 21 points
Dealer [3 Hearts, 2 Spades, 4 Hearts, 9 Diamonds, 7 Spades] : 25 points
```

What next ?

- Interfaces graphiques
- Persistance
- **Refactoring**
  - quelle architecture
  - quels design patterns
  - ...