JUno

Andrea Musolino 1958512

Relazione per il progetto del corso di
Metodologie di Programmazione
Canale M-Z

Dipartimento di Informatica
Università degli studi Sapienza di Roma

Contents

0.1 Introduzione

Il progetto JUno è stato progettato adottando il pattern MVC come richiesto da specifiche. Il programma segue per intero la logica e le regole del gioco di carte "UNO!"(Regole di UNO!). Di seguito vengono illustrate nel dettaglio le note progettuali riguardanti: Model, View e Controller.

0.2 Model

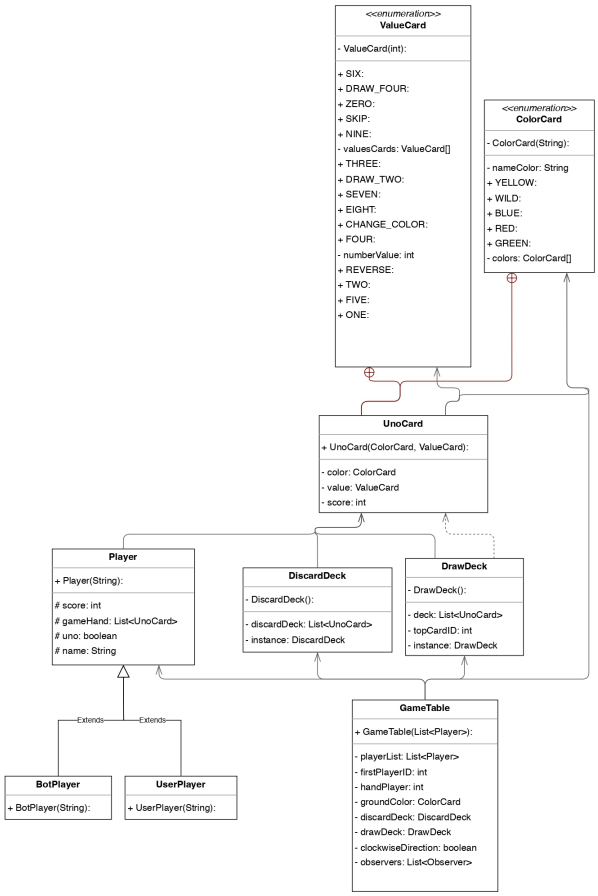

Figure 1: Diagramma UML del Model

Il model è stato progettato seguendo le entità principali del gioco, ovvero: carte, mazzo di carte, giocatori e tavolo da gioco. Le carte vengono modellate dalla classe `UnoCard` che contiene al suo interno due enumerazioni `ValueCard` e `ColorCard`, rispettivamente il valore e il colore che una carta può assumere. I giocatori sono rappresentati dalla classe astratta `Player`, che modella gli aspetti principali di un giocatore generico, ossia: il nome, il punteggio e la mano di gioco. La classe viene a sua volta estesa da due classi concrete che rappresentano

i due tipi di giocatore vale a dire `BotPlayer` e `UserPlayer`. Le classi non reimplementano o aggiungono alcun metodo della classe genitore, servono solo per distinguere il tipo di giocatore: bot o utente. Dopodichè sono state create due classi per rappresentare i due tipi di mazzo: il mazzo di pesca e il mazzo degli scarti. Ognuno ha una classe a sè, rispettivamente il `DrawDeck` e il `DiscardDeck` che implementano i comportamenti principali che assumono i mazzi di carte. Infine il tutto viene incapsulato nella classe principale del model, ovverosia il `GameTable`, che fornisce tutti i metodi che servono alla gestione di una partita di UNO!. I design pattern adottati all'interno del model sono:

- Singleton pattern: adotatto dalle classi `DrawDeck` e `DiscardDeck`, dato che è sufficiente avere una singola istanza di entrambi le classi
- Observer e Observable pattern: come richiesto da specifiche la classe `GameTable` implementa l'interfaccia Observable (riscritta da me dato che è stata deprecata) e rende la classe osservabile da altre entità.

Per quanto riguarda l'utilizzo degli stream non ne ho visto un'eccessiva utilità. Di seguito vengono illustrati gli snippet dei due metodi del `GameTable` in cui vengono adottati:

- Metodo utilizzato per contare il punteggio delle carte rimaste in mano ai giocatori a fine partita

```
public int countScoreCards() {
   return (int)playerList.stream()
            .map(Player::getGameHand)
            .flatMap(List::stream)
            .map(UnoCard::getScore)
            .reduce(0, Integer::sum);
}
```

- Metodo utilizzato per controllare se un giocatore non ha più carte in mano

```
public boolean hasPlayerZeroCard() {
    return playerList().stream().anyMatch(Player::hasEmptyHand);
}
```

0.3 View

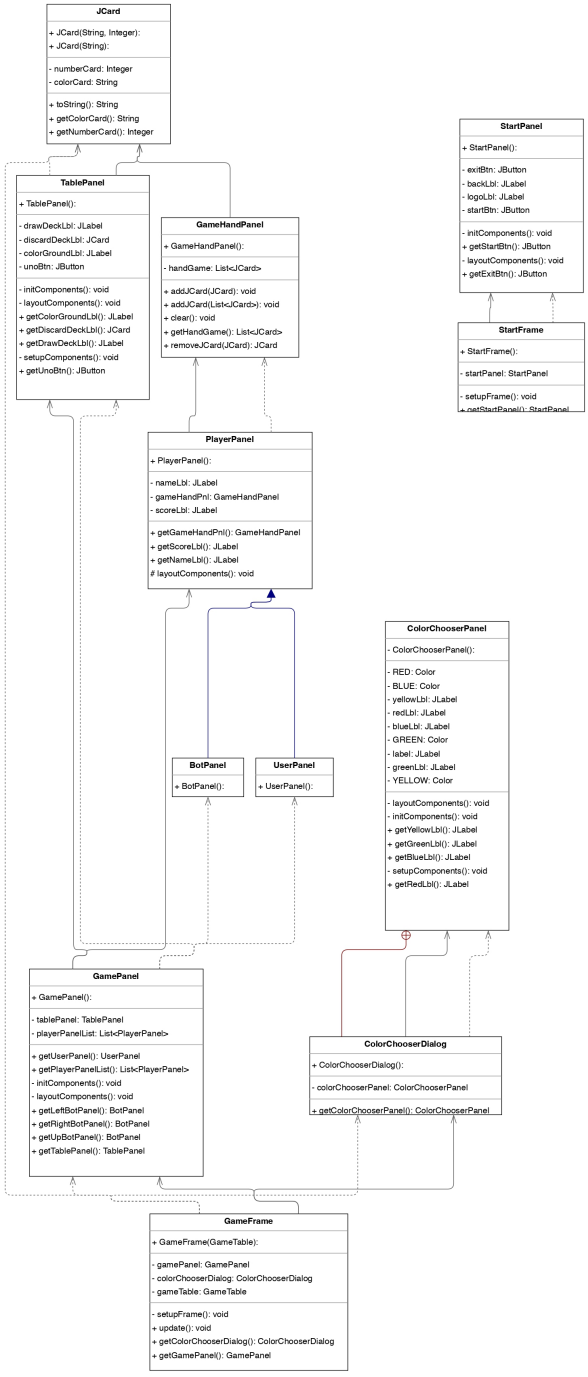

Figure 2: Diagramma UML della View

La View è stata strutturata come segue. Ci sono due frame principali lo `StartFrame` e il `GameFrame`. Lo `StartFrame` è il frame iniziale che viene aperto ad esecuzione del gioco, è composto dal pannello `StartPanel` che contiene i componenti che vengono visualizzati al suo lancio. Il `GameFrame` è il frame principale. All'interno ha un riferimento al pannello `GamePanel` che sarà il pannello con cui interagirà l'utente. Il `GamePanel` al suo interno contiene cinque pannelli.

- Un `TablePanel`: è il pannello che rappresenta il "tavolo da gioco". Al suo interno vi sono:
  - Tre `JLabel` che rappresentano il mazzo di pesca, il mazzo degli scarti e il colore della carta a terra
  - Un `JButton` che rappresenta il bottone da cliccare per dire "UNO!" quando si ha una carta una mano
- Quattro `PlayerPanel`: posizionati agli estremi del `GamePanel`. Ogni pannello conterrà al suo interno:
  - Due `JLabel` per il nome e il punteggio del giocatore
  - Un `GameHandPanel`, ovvero il pannelo che rappresenta la mano di gioco di un giocatore. Al suo interno vi sarà una lista di `JCard` che modellano una generica carta da UNO.

  La classe `PlayerPanel` è astratta e verrà estesa, come nel model, dalle classi concrete `UserPanel` e `BotPanel`.

Inoltre vi è una classe `ColorChooserDialog` che verrà istanziata quando bisognerà scegliere il colore di una carta wild. L'unico design pattern adottato nella view è l'Observer e Observable, come richiesto da specifiche. Viene implementata l'interfaccia `Observer` (anch'essa riscritta da me) dalla classe `GameFrame` e osserva quindi la classe osservabile `GameTable`. Non sono stati adottati gli stream.

0.4 Controller

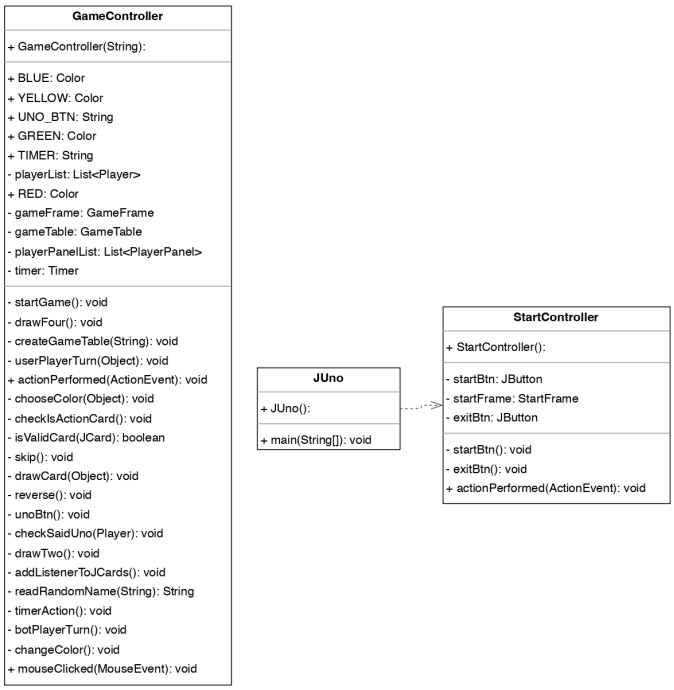

Figure 3: Diagramma UML del Controller

Il controller è diviso in tre classi:`StartController`, `GameController` e `JUno`. Lo `StartController` istanzia lo `StartFrame` ed attiva i listener dei componenti al suo interno. Il `GameController` è la classe principale da cui viene gestita una partita completa di UNO. Come nello `StartController` anche qui vengono attivati i listener ai componenti contenuti nel `GamePanel`. Infine la classe `JUno` conterrà il main da cui verrà eseguito il gioco.