Peer-Review 1: UML

Davide Ciamacco, Andrea Callari, Simone Checchia, Federica Currò Dossi

Gruppo AM03

Valutazione del diagramma UML delle classi del gruppo AM12.

Lati positivi

- **Scalabilità carte obiettivo**
  L'utilizzo della classe 'PatternObjectiveCard', usata per verificare carte obiettivo "pattern", permette di poter estendere facilmente il gioco con nuove carte della medesima tipologia (disposizione di tre carte) senza l'utilizzo di nuove classi.

- **Interfaccia sicura**
  Adottare la scelta di imporre a void il tipo di ritorno dei metodi della classe 'GameModel', la quale si interfaccia con il Controller, impedisce di accedere ai metodi modificatori degli oggetti.

- **Chiarezza e sintesi**
  L'UML risulta prevalentemente chiaro e autoesplicativo rappresentando una buona base su cui modellare il gioco. La presenza di molteplici metodi, che supportano lo svolgimento di una partita, consente di cogliere a pieno il flusso di esecuzione di quest'ultima. È inoltre evidente la distinzione tra Model e Controller.

Lati negativi

- **Parità**
  Non è specificato come viene gestita la situazione di parità tra più giocatori, ovvero non si tiene conto del criterio per cui il giocatore che ha realizzato più carte obiettivo viene decretato vincitore.

- **Angoli vuoti**
  Dopo una prima lettura dell'UML risultava poco chiara la gestione dei corner vuoti. Solo dopo esserci confrontati con il gruppo abbiamo colto la loro idea di implementazione.

- **Carte scoperte**
  All'interno della classe 'DrawTable' è netta la distinzione riguardo la tipologia delle carte svelate, due di tipo 'GoldCard' e due 'ResourceCard'. Con questa scelta non verrà rispettato il regolamento nel caso uno dei due mazzi dovesse terminare. (Slack)

- **Coordinate**
  All'interno delle classi 'GameCard' e 'Match' è presente un riferimento ad oggetti di tipo 'Coordinate', classe che non viene definita all'interno dell'UML. Supponendo che sia un oggetto contenente una coppia di interi 'x' e 'y', sarebbe più coerente utilizzare questa classe in tutto l'UML.

- **Classe astratta**
  Reputiamo che la classe ‘GameCard’ debba essere abstract poiché nessuna carta avrà tipo dinamico ‘GameCard’.

- **Refuso**
  Il tipo di ritorno del metodo ‘getCardsInHand’ presente nella classe ‘Match’ dovrebbe essere una lista di int anziché una lista di String per coerenza con gli altri metodi.

Confronto tra le architetture

A valle di un attento confronto tra i due UML, è stato possibile riscontrare diverse analogie. Tra queste le scelte relative alla progettazione e gestione del giocatore e del suo tavolo di gioco. In entrambe viene fatto uso di una matrice per le carte giocate e di una struttura per verificare in modo pratico ed efficiente la presenza degli oggetti sul tavolo da gioco individuale.

Sono comunque presenti delle differenze tra i due Model. Una delle principali è la gestione del lato su cui viene giocata la carta, nel nostro Model rappresentato in modo più sintetico come attributo, mentre in quello analizzato viene rappresentato come classe.
Diversa è anche l’implementazione dei deck pensati come un array di carte, invece di creare una classe dedicata come nel nostro Model.
Infine, è presente una differenza sulla verifica delle classi obiettivo di tipo pattern: è stata preferita una soluzione più flessibile piuttosto che tramite ereditarietà come nel nostro caso.

Diversi dettagli sono stati apprezzati e saranno utilizzati come spunto per migliorare il nostro UML. Tra questi l’attributo, presente nella classe ‘GameCard’, relativo alle coordinate spaziali su cui una carta viene posizionata e l’attributo ‘flagObj’ che tiene conto se una ‘GameCard’ è stata già considerata per un medesimo obiettivo.
Ottimo anche l’approccio orientato ad uno sviluppo atomico delle funzionalità del gioco tramite metodi specifici. Per esempio il metodo ‘getPlaceablePositions(x: int, y: int)’ che restituisce le celle disponibili intorno a un coppia di coordinate. Di questo metodo potrebbe risultare utile implementare una sua generalizzazione, ritornando tutte le celle nella ‘PlayerBoard’ su cui è possibile posizionare una carta.
Infine, sono condivisibili le scelte strutturali come la classe ‘Corner’, la cui semplicità ne permette il riuso, e la presenza di una unica classe (‘Match’) incaricata della partita, agevolando di fatto l’implementazione.