Peer-Review 1: UML

<Luca Bigatti>, <Alessandro Bacchio>, <Alessandro Bertelli>,<Davide Ali>

Gruppo <23>

Valutazione del diagramma UML delle classi del gruppo <32>.

Lati positivi

- Chiarezza:
  L'organizzazione del diagramma è tale per cui il modello UML risulta chiaro ed autoesplicativo, l'obiettivo verso l'implementazione appare ben delineato e dettagliato.
  Appare evidente la rigorosa organizzazione e la separazione chiara nel model in sottosezioni (modelPlayer, modelCard ecc.) che rende il diagramma di facile comprensione.

- Pattern Strategy:
  Nota di merito sicuramente per l'utilizzo di pattern strategy per orchestrare integralmente e ottimamente la parte relativa alla gestione del punteggio del singolo giocatore.
  Apprezzabile nell'implementazione delle strategie il fatto di ritornare un coefficiente moltiplicativo che servirà poi per l'effettivo conteggio dei punti ottenuti.

- Interfaccia Game:
  Interessante l'implementazione di una interfaccia di gioco da esporre direttamente al controller al fine di ottenere una maggiore chiarezza, una netta flessibilità e una manutenibilità durante lo sviluppo software.

In generale si nota un buon approccio iniziale verso il diagramma UML della parte di Model, che sicuramente permetterà di realizzare una buona implementazione del progetto assegnato.

Lati negativi

Come detto in precedenza il diagramma è ben realizzato, tuttavia riportiamo i seguenti suggerimenti riguardo piccoli difetti notati, nella speranza che possano aiutare il gruppo ad una maggiore integrazione, volta ad una dettagliata implementazione:

- Mancanza di un ID del match:
  L'aggiunta dell'attributo ID risulta utile per diversi fattori; permette l'identificazione unica della partita di gioco, aiutando il server a gestire più richieste da parte del client, permettendo inoltre di tenere traccia dei giocatori connessi. Suggeriamo quindi una semplice aggiunta del suddetto attributo nella classe Match.
- Sottoclassi di Card:

  La presenza dell'unica sottoclasse NonObjectiveCard rende magari più veloce l'implementazione ma ne aumenta la difficoltà in termini di leggibilità e comprensibilità, poiché non vi è una netta distinzione tra oggetti di tipo diverso (carta risorsa, carta oro, ecc ecc).

Confronto tra le architetture

Le due architetture, nonostante la diversità grafica del class-diagram, risultano simili nell'idea di base che ruota attorno al flusso di gioco e nella dichiarazione delle classi con i relativi attributi/metodi.

Entrambi i gruppi hanno preso la decisione di gestire parte della classe relative alle carte (in particolare le implementazioni degli obiettivi) mediante una pattern-strategy per garantire una maggiore flessibilità del codice e una semplificazione di esso. Il nostro gruppo ha deciso di adottare la seguente modalità anche per la creazione della carte, in modo tale da garantire la riusabilità del codice per un'azione giudicata ripetitiva.

La gestione degli angoli delle carte è stata trattata in modo diverso nei due modelli UML: nel nostro caso, abbiamo gestito ciò attraverso una classe dedicata, dove ogni carta avrà i riferimenti dei suoi angoli. Nel modello revisionato, invece, si è scelto di gestire esplicitamente il contenuto/esistenza degli angoli tramite un enumeratore che identifica lo stato dell'angolo.

Osserviamo che la gestione delle risorse contenute nelle carte è simile e strutturata utilizzando una semplice enumerazione, con la sottile distinzione che il nostro gruppo ha due enum separate per distinguere tra "oggetti" e "risorse".

L’idea sull’implementazione delle carte piazzate dai singoli giocatori è anch’essa analoga, basandosi sul tracciamento delle posizioni mediante coordinate tramite la creazione di una classe apposita. L’unica differenza è l’utilizzo da parte nostra del supporto di un’ulteriore matrice di carte, una struttura che abbiamo giudicato chiara per lo sviluppo del progetto.

La pianificazione di tutta la parte riguardante carte e deck è affine alla nostra idea ancora una volta, con lievi dissomiglianze. Il nostro gruppo ha deciso di realizzare diverse classi per le diverse tipologie di deck presenti nel gioco (in base alla tipologia di carte che essi gestiscono) e gestirle mediante polimorfismo con una classe astratta, stessa tecnica adottata per lo sviluppo della classe “card”. A nostro giudizio tutto ciò favorisce una struttura gerarchica chiara e organizzata nel codice, mediante ereditarietà si gestisce al meglio il comportamento e le caratteristiche della classe astratta, riducendo la duplicazione del codice e promuovendo la coerenza nell'applicazione. L’altro gruppo ha avuto un approccio più sintetico nella gestione della classe “deck”, decisione comunque lodevole in base alla loro idea di sintesi e chiarezza del diagramma; la loro gestione della carte è invece affidata ad un pattern-strategy coinciso, rendendo il codice più chiaro e modulare, separando la logica delle carte.

In conclusione entrambi i diagrammi UML riteniamo siano stati eseguiti con accuratezza e siano ben pensati per svolgere le funzioni primarie del progetto, presentando lieve differenze solamente nell’organizzazione del codice e mantenendo forti analogie nella logica implementativa.