Peer-Review 1: UML

Pirola Lorenzo, Rabbolini Alessandro, Palazzi Matteo, Poli Matteo
Gruppo GC02

aprile 2024

Valutazione del diagramma UML delle classi del gruppo GC49.

1 Lati positivi

- **Buona gestione delle classi:** le classi sono specifiche, ben autocontenute e gerarchizzate, e bene ottimizzate. Da una rapida occhiata al diagramma UML é subito comprensibile il ragionamento top-down con cui il gruppo si é approcciato al progetto.
- **Ottima rappresentazione di mazzi e carte visibili/giocabili:** la classe ConcreteDeck, rappresentante un mazzo di carte, é stata implementata con lo specifico oggetto astratto tipico Stack<Card>, semplificando ampiamente la gestione dei controlli su metodi e attributi; la gestione delle carte scoperte é stata gestita mediante una classe ulteriore, per permettere allo Stack di simulare tutto ció che resta dovendo utilizzare solo *push* e *pop*.
- **Utilizzo di classi static per calcolo punteggi e inizializzazione mazzi:** il gruppo ha implementato due classi statiche, DeckFactory e ScoreStrategy, permettendo un livello ulteriore di astrazione su questi due specifici task complessi e rendendo l'opera nel complesso piú pulita e comprensibile.

2 Lati negativi

2.1 Criticitá

- **Conflitto tra classe Cell ed enum Cell:** nel progetto sono state inserite una classe ed una enumerazione con il medesimo nome: Cell. Ció puó causare possibili conflitti nella scrittura del codice da sé; inoltre, dopo una piú accurata analisi, é stato notato che l'enum Cell non viene utilizzata in alcun attributo o metodo del progetto (o, se utilizzata, ció é stato fatto ambiguamente con gli oggetti con stesso nome).

- **Assenza gestione orientamento della carta in ResourceCard:** non sono stati individuati attributi e/o metodi per delineare se le carte risorsa/oro siano orientate sul verso fronte o retro.

- **Ridondanze:** i punteggi dei giocatori sono duplicati nelle classi Game e Player; Player.partial_winner e Game.temp_winner sembrano corrispondere allo stesso dato.

2.2 Appunti

- **UML senza molteplicitá esplicite:** le associazioni tra classi non recitano molteplicitá. Ció peró non causa gravi problemi di comprensione poiché gli attributi nelle singole classi permettono di dedurre la molteplicitá.

- **Oggetti Coordinate:** nonostante si faccia riferimento diverse volte ad oggetti di tipo Coordinate, non é stata definita una classe o enumerazione che li gestisca. Presumiamo si tratti di una semplice classe non dissimile a java.awt.Point.

- **Connessione, riconnessione, chat:** queste operazioni, a nostro parere, sono da inserirsi nel controller, essendo parte della funzionalitá di rete del gioco. Nel model si dovrebbe peró comunque tenere traccia della cronologia dei messaggi.

- **Enumerazioni semplificabili:** Orientation puó essere sostituita da una variabile booleana (che dica, ad esempio, se la carta é sul fronte o no); Resource e Item si comportano, di fatto, allo stesso modo, nonostante le regole del gioco facciano una distinzione tra le due; si potrebbe

quindi riassumere le due categorie di risorsa sotto un unico Resource, permettendo inoltre di poter omettere le Optional in Corner ed eliminando casi limite da dover altrimenti controllare (come la co-esistenza di una risorsa e un item nello stesso angolo).

3 Confronto tra le architetture

- **Diverse deleghe a model e controller:** la nostra architettura delega molti compiti al controller che quella in esame ha lasciato al model. Ció prevede vantaggi, come una migliore visione statica del gioco da lato client, e svantaggi, come la maggiore quantitá di tempo necessaria a dover richiedere molte informazioni al controller.

- **Eccezioni:** l'architettura del gruppo GC49 prevede la gestione degli errori e dei casi limite tramite eccezioni, cosa che nel nostro gruppo é stata gestita con metodi "normali" e per via di ció meno specifici e flessibili. Provvederemo ad implementare eccezioni.

- **Differente gestione carte obiettivo:** mentre il nostro progetto include una singola classe QuestCard, con al piú attributi ignorabili e un controllo interno al metodo di risoluzione obiettivo, quello in esame contiene quattro sottoclassi di ObjectiveCard, ognuna con un override al metodo calculateScore(). La soluzione proposta dal gruppo GC49 permette una funzionalitá piú pulita senza attributi inutili, ma piú complessa dal punto di vista strutturale del progetto.