Peer-Review 1: UML

Tommaso Crippa, Alessandro Di Maria,
Mattia Brandi, Rodrigo Almandoz Franco
Gruppo IS24-AM42

3 aprile 2024

Valutazione del diagramma UML delle classi del gruppo IS24-AM51.

1 Lati positivi

- L'UML è strutturato molto bene, si individuano le interazioni tra le classi in modo immediato grazie alla sua leggibilità;
- I nomi delle variabili e dei metodi sono autoesplicativi, raramente ci siamo chiesti la funzione di certi elementi;
- L'utilizzo di coordinate per definire la posizione delle carte è appropriato;
- L'utilizzo dell'attributo "bag" nella Board è un modo efficiente per ottenere in modo rapido le risorse inserite senza analizzare ogni volta le carte posizionate;
- La composizione della carta, fatta da quattro attributi di corner diversi, può portare diversi benefici durante la definizione di metodi più complicati;
- La definizione del tipo di ritorno nel metodo addCard() in Board, permette immediatamente di controllare se la posizione di una carta è corretta;

- Le classi Objective sono strutturate molto bene al fine di valutare il numero di punti guadagnati alla fine della partita. In particolare, in "LShapeRequest", l'attributo pattern è definito in modo tale da essere scalabile, considerando ad esempio futuri aggiornamenti in cui vengono aggiunte nuove carte obiettivo di tipo "LShape" con colori diversi.

2 Lati negativi

Di seguito analizzeremo i lati negativi dell'UML:

- Non sono presenti i costruttori delle varie classi. Secondo noi è importante specificarli per avere una visione più chiara di come vengono istanziati gli oggetti;
- Non è chiaro il funzionamento della classe Board:
  - Perché dichiare una mappa, se nel metodo GameBoard (ipotizziamo essere il costruttore), vengono passati gli integer row e col? Perché limitare la dimensione della Board fin dall'inizio?
  - Come posso inserire una carta in una certa posizione, se il metodo addCard non accetta come parametro le coordinate? Mancano, inoltre, i metodi getter e setter della classe BoardCoords.
- Nella classe Game, non vi è una tecnica di memorizzazione né dell'ordine di gioco né degli utenti stessi, ovvero non riusciamo a capire come Game possa avere un riferimento ai Players;
- Non è evidente il funzionamento della classe StarterCard tramite il metodo chooseSide poiché non si capisce come venga scelto il lato con cui viene giocata;
- Non è riportato il metodo shuffle() nelle classi StarterCard e Objective;
- Nella classe Player, riteniamo che il metodo chooseObjective() debba essere implementato nella view e non nel model per poter avere un input esterno;
- Non sono presenti alcuni attributi, metodi (getter e setter) e parametri di ingresso. Manca il riferimento a Board in Player, il riferimento a

Player in Game. Mancano i metodi getter e setter nella classi Corner e BoardCoords. Il metodo getCardPos() dovrebbe ricevere qualche parametro, come anche il metodo verifyObjective() in Objective (forse Board di uno specifico player?).

A luce di quanto detto in precedenza, riteniamo che, chiamando i metodi di Game e quelli accessibili ad esso, non si possa giocare in maniera completa.

3 Confronto tra le architetture

Da un confronto con la nostra architettura si possono notare alcune similarità, come: la definizione di varie classi, l'utilizzo delle coordinate e l'uso della "bag" nella board per memorizzare il numero di risorse utilizzate. Rispetto al nostro model, però, l'implementazione dei corner risulta più compatta pur rimanendo comunque completa, dunque si potrebbe valutare una variazione alla nostra implementazione attuale. Inoltre in questa architettura risultano già descritte e implementate le funzionalità aggiuntive come si può notare dalla classe GameDB per la persistenza e dall'utilizzo di funzioni come disconnected() e isConnected() per la resilienza alle disconnessioni.