Peer-Review 1: UML

Maddalena Panceri, Alessandro Ruzza,
Gabriele Santandrea, Flavio Villa
Gruppo 33

Aprile 2024

Valutazione del diagramma UML delle classi del gruppo 23.

1 Lati positivi

1.1 Generale

Ottima l'ampia visione di progettazione della *View* e del *Network*. Specialmente quest'ultimo è stato strutturato rispetto agli esempi presentati a esercitazione.

Ottimo anche l'uso di *design pattern*, quali *factory pattern* per l'istanziazione dei mazzi e *strategy pattern* per la valutazione degli obiettivi e delle gold card (eccetto alcuni accorgimenti).

1.2 Elementi strutturali

Corretto l'uso di **Game** come classe fulcro, che racchiude gli elementi chiave della struttura di gioco (*Table, Player, ecc..*).

Ingegnosa la strutturazione del *Player*, principalmente per l'inclusione dell'attributo **status** che controlla la connessione o disconnessione del client.

2 Lati negativi

2.1 Generale

Innanzitutto risalta all'occhio l'utilizzo delle composizioni. Queste sono utilizzate in maniera scorretta in molteplici casi: le carte, per esempio, esistono come entità a sé, in quanto compongono più classi differenti.

Dopodiché si è notata l'assenza di dettagli su alcuni attributi e metodi poco comprensibili dall'UML (alcuni segnalati in seguito).

Alcune frecce di aggregazione e composizione sono nella direzione errata (Token –> Scoreboard; DeckFactory –> Deck).

Si è notata la presenza della classe Menu. Poiché non ne è stato spiegato l'utilizzo o la funzione, non si hanno abbastanza elementi per decidere se la sua inclusione nel *Model* sia corretta o meno.

CentralCards è una classe ritenuta superflua, in quanto i suoi componenti sono assimilabili dalla **Table**.

2.2 Board

La mappa in **Board** che implementa l'area su cui giocare le carte ha una sola coordinata. Inoltre la carta **starter** non ha coordinate. Non si comprende come sono integrate le due informazioni.

2.3 Player

- Sia player sia scoreboard hanno l'informazione riguardante il punteggio del giocatore. Non se ne comprende l'utilità;
- Non è stata annotata alcuna funzione che permetta l'accesso alla mano del giocatore;
- Non è stato spiegato come viene identificato il primo giocatore. Inoltre nessun token sulla scoreboard può essere nero;

2.4 Objective

- Le due sottoclassi **SharedObjective** e **PersonalObjective** non hanno alcuna differenza rispetto alla sovraclasse, o almeno non è stata specificata, perciò tale divisione si ritiene errata;

- L'interfaccia **Objective** funziona anche per le **GoldCard**. In tal modo si potrebbe associare un obiettivo a una **GoldCard** (in via potenziale) e viceversa;
- **PlacementCard** corrisponde all'attributo **points** di **ResourceCard**;
- **pointsGained** prende **Objective** come parametro (non se ne capisce il motivo);
- l'attributo **points** sembra duplicato su **Objective** e **ObjectiveCard** e **GoldCard** (ereditato da **ResourceCard**).

2.5 Cards

- Si è notata la duplicazione di elementi tra **ResourceCard** e **StarterCard** (e.g. *flip()*, **isFront**, ...)
- Il metodo *cover()* non è spiegato. Il dubbio sorto è nella gestione dei piazzamenti delle carte sui diversi corner e la relativa copertura di risorse;
- L'ereditarietà tra **GoldCard** e **ResourceCard** permetterebbe di usare una **GoldCard** come **ResourceCard** (per esempio inserirla in **ResourceDeck**);
- Da quello che si ricava dall'UML, la classe **Deck** restituisce **ResourceCard**, quindi un metodo che vuole piazzare una **GoldCard** non ha modo di chiamare i metodi della sottoclasse (**getCost** e **getObj**). Si suppone che la funzione *place()* sia ridefinita in **GoldCard** e che chiami le funzioni in questione (che dovrebbero allora essere private).
- il costo delle **GoldCard** è un **Set<Resource>**; questo è errato in quanto il costo può avere più risorse dello stesso tipo (e.g. 5 farfalle);
- Un'ultima nota riguarda le funzioni *getFront()* e *isFlipped()*: sembrano avere la stessa funzione.

3 Confronto tra le architetture

L'uso della classe centrale **Game** è un principale punto di forza dell'UML del gruppo 23, che dimostra lungimiranza verso la funzione addizionale di partite multiple sullo stesso server. Prenderemo spunto da questo.

Similmente, l'utilizzo di uno *status* per indicare la disconnessione del **Player** mira all'implementazione della funzione di resilienza alla disconnessione, o comunque alla notifica di disconnessione di un giocatore. Questo elemento manca nel nostro *Model* e pertanto lo includeremo.

Anche il package *Network* può migliorare il nostro UML, ponendo le basi per la futura definizione dei protocolli di comunicazione.