Peer-Review 1: UML

<Riccardo Domingo Compagnoni>, <Eleonora Fidelia Chiefari>, <Michele D'Elia>, <Emanuele Dossi>

Gruppo <AM28>

Valutazione del diagramma UML delle classi del gruppo <AM37>.

Lati positivi

A livello generale osserviamo un'ottima gestione delle gerarchie tra classi: Game funge da contenitore di player e board, che a loro volta permettono di accedere alle common card e ai personal target. Ciò semplifica la gestione delle fasi di gioco e della sequenza di invocazione dei metodi per modificare lo stato della partita.
Riteniamo inoltre la classe Utils molto comoda per evitare la duplicazione degli stessi metodi in diverse parti del codice.
Un ulteriore elemento che favorisce la chiarezza è la nomenclatura, che è autoesplicativa; in questo modo il funzionamento del modello risulta facilmente comprensibile.

Lati negativi

Abbiamo trovato superflue le seguenti classi:

1. TileSlot, che aggiunge un ulteriore passaggio per accedere alla Tile dalla Library, senza però aggiungere metodi o funzionalità che non sarebbero disponibili semplicemente memorizzando la libreria come matrice di Tile.
2. CommonDeck, poiché contiene solo un ArrayList di CardCommonTarget e un metodo getter. Suggeriamo dunque di rimpiazzare questa classe con un attributo privato. Inoltre, crediamo sia più ragionevole memorizzare il deck in Game, perché, nonostante il mazzo sia visivamente rappresentato sulla plancia, non ha alcuna connessione logica con la classe Board. Dato che tutti i comandi del controller passano da Game, questo spostamento faciliterebbe anche l'accesso al metodo getScoringToken quando il Player finisce il turno.
3. PersonalDeck, per gli stessi motivi del mazzo di carte comuni.
4. ScoringToken, perché riteniamo che uno Stack<Integer> svolgerebbe le stesse funzioni.

Inoltre, dato che le diverse CommonCard sono differenziate mediante enumerazione, presumiamo che il metodo getScoringToken contenga gli algoritmi di ogni classe, anche se di fatto solo uno sia quello utile. Questo problema potrebbe essere risolto utilizzando l'ereditarietà, anche se neanche questa sarebbe una soluzione perfetta, dato che porterebbe ad appesantire notevolmente l'UML.
Per quanto riguarda la nomenclatura dell'enum CommonList, riteniamo che sia possibile utilizzare dei nomi più corti e quindi più maneggevoli, ma che consentano ugualmente di identificare efficacemente la carta che rappresentano. Ad esempio "FOUR_FULL_ROWS_WITH_MAX_THREE_ DIFFERENT_TYPES" potrebbe essere rimpiazzato con "FOUR_ROWS".

Confronto tra le architetture

La nostra architettura è complessivamente simile a quella del gruppo revisionato, sia nella nomenclatura che nelle relazioni tra classi. Si distingue invece per i seguenti elementi: la mancanza di alcune classi come Library, PersonalDeck, CommonDeck e Utils; il collegamento delle CommonCard al Game anziché alla Board; la differenziazione delle CommonCard mediante ereditarietà.
In generale, nel modello del gruppo revisionato abbiamo notato una maggiore separazione degli elementi in diverse classi, che potrebbe facilitare la fase di testing. Tuttavia, come riportato negli

aspetti negativi, crediamo che alcune classi risultino ridondanti o superflue. Abbiamo invece trovato ottima l'idea di astrarre nella classe Utils i metodi ricorrenti; valuteremo quindi le modifiche da apportare al nostro progetto per adottare una strategia simile.