Peer-Review 1: UML

Tommaso Trabacchin, Melanie Tonarelli, Emanuele Valsecchi, Adem Shehi

Gruppo 35

Valutazione del diagramma UML delle classi del gruppo 25.

Lati positivi

- L'enumerazione "`State`", riferita al `GameModel`, garantisce una buona base per la gestione sia dell'andamento del gioco che della resilienza alle disconnessioni.
- L'utilizzo dello `Stack` per l'assegnazione dei punteggi delle carte obiettivo comune (contenuto nella classe `CommonGoalCardDeck`) è una soluzione vincente in quanto riproduce fedelmente la pila di carte punteggio del gioco e ha un facile utilizzo implementativo.
- Riteniamo che sia una buona idea mantenere per ogni `Player` il proprio stato di gioco attraverso l'enumerazione `PlayerState`. Infatti risulta utile per evidenziare quali azioni possono essere compiute dal giocatore in un determinato istante ed inoltre permette di gestire facilmente eventuali problemi di connessione.

Lati negativi

- Alcuni nomi presenti nell'UML sono poco significativi. Tra di queste vi rientrano:
    - Le classi rappresentanti le carte obiettivo comune che vengono distinte aggiungendo un numero progressivo.
    - `CommonGoalCardDeck`.
    - I metodi contenuti in `PersonalGoalCardBag`.
- Le classi che identificano i `CommonGoal` si presentano in numero molto elevato e probabilmente possono essere accorpare maggiormente. All'interno di alcune di queste viene indicato un frammento di codice specifico che andrebbe piuttosto inserito in una nota aggiuntiva
- Il connettore "composizione" usato tra le "`ItemTile`" e le altre classi è probabilmente da sostituire. Usando questo infatti si sottintende che l'assenza di `Board`, `PersonalGoalCard`, `Bookshelf` e `ItemTileBag` porti all'inesistenza delle `ItemTile` stesse.
- La relazione che intercorre tra la classe `Player` e la coppia di classi `CommonGoalCardDeck` e `CommonGoalCard` è poco chiara. Infatti la classe `Player`, oltre ad avere un attributo "`ArrayList<CommonGoalCard>`", presenta anche un'associazione con "`CommonGoalCardDeck`" che a sua volta presenta una mappa con valori di tipo "`CommonGoalCard`", ciò potrebbe causare un'inconsistenza di dati.
- Nella classe `PersonalGoalCardBag`, interpretato da noi come contenitore di carte obiettivo personale, presenta:
    - metodo "`refill()`", la cui utilità non è chiara; infatti abbiamo assunto che con tale metodo vengano create o inserite nuove carte obiettivo personale, cosa che non è specificata nelle regole di gioco fornite.
    - Un attributo "`availableItemTiles`" di tipo `Stack<ItemTile>` di cui non capiamo la finalità.
- Nella classe "`PersonalGoalCard`" il pattern da verificare viene basato su una matrice di tipi "`ItemTile`" che risulta inconsistente con l'assunzione che le `ItemTile` siano univoche (data la

presenza di un ‘id’ per identificarle). Ciò infatti non rispetta le regole di gioco dato che l’obiettivo personale si basa solamente sulle corrispondenze tra tipi di tessere di gioco.

- Nella classe `Board` è presente un attributo di tipo `Bag` però non presente nel diagramma. Probabilmente il tipo a cui il gruppo voleva riferirsi è “`ItemTileBag`”; se questo fosse il caso, il metodo “`fillBag()`” di quest’ultima classe potrebbe essere omesso in favore di un unico riempimento della sacca nel costruttore della stessa (non ne è infatti previsto uno successivo nelle regole di gioco).

Confronto tra le architetture

Confrontando il diagramma realizzato dal nostro gruppo con quello del gruppo 25, abbiamo riscontrato che una buona parte della loro architettura è simile alla nostra, tuttavia ci sono alcune differenze sostanziali.

La classe `Board` (rappresentante la plancia soggiorno) scelta dal gruppo 25 è strutturata come una matrice, mentre la nostra utilizza un grafo ideato appositamente che emula la struttura effettiva della plancia di gioco.

Nella classe `ModelGame`, i giocatori sono memorizzati in un array circolare (diversamente dall’array lineare utilizzato da noi) che può risultare comodo per la gestione dei turni e dei giocatori successivi al giocatore corrente.

Abbiamo inoltre ritenuto utile l’enumerazione “`State`” usata come attributo di `ModelGame`, dalla quale pensiamo di prendere spunto allo scopo di gestire più agilmente l’evoluzione di gioco.

La quantità di classi usate è sicuramente in numero superiore a quanto ideato dal nostro gruppo. Nella nostra architettura infatti due o più classi presenti nel diagramma del gruppo 25 sono collassate in un’unica classe; è questo il caso della classe `Bag` che abbiamo preferito inserire come attributo della classe rappresentante la “living room board” (mentre è stata tenuta separata nella loro architettura) e anche delle carte che implementano i common goal che nel nostro caso risultano in numero minore.

Infine, un ulteriore aspetto positivo da cui trarremo spunto è l’utilizzo di uno `Stack` di interi per memorizzare i punteggi associati alle carte obiettivo comune e quindi per gestire più facilmente l’assegnazione dei punti secondo le regole di gioco.