Peer-Review 1: UML

Simone Rodari, *Cristian Summa*, *Mathias Rodigari*, *Gabriele Pignataro*

Gruppo *GC42*

Valutazione del diagramma UML delle classi del gruppo *GC32*.

Lati positivi

- Ottima gestione dell'import delle carte tramite *CardFactory*
- Ottima idea la divisione tra *Observable* e *ViewObservable* ed i rispettivi *Observer* e *ViewObserver:* semplificherà non di poco l'implementazione delle View quando giungerà il momento.
- Buona la gestione interna di *DrawableDeck* tramite la *Map*, ottima anche per futuri upgrade.
- Buona l'idea di definire una *enum* per tutti gli stati che il *Game* può assumere.

Lati negativi

- Uso eccessivo di array (invece di strutture dati dinamiche e già fornite di tutti i metodi necessari già ben ottimizzati) e soprattutto di dimensione *hardcoded* (ad esempio dimensione prestabilita di 4 elementi).
- Alcune strutture dati interne sono esposte, come nel caso di *center* in *StartingCard* e *players* in *Game*
    - In generale non è buona pratica in OOP esporre l'implementazione interna degli attributi di classe, ed è preferibile fornire solamente i metodi per interagire con i dati al loro interno, piuttosto che ritornare al chiamante l'intera struttura che li contiene: ad esempio invece di *Game.getPlayers()* che ritorna la queue sarebbe preferibile avere solo i metodi (già presenti) *getFirstPlayer()*, *getCurrentPlayer()* e, nel caso fosse necessario, affiancargli un *getPlayer(int)* in cui il parametro *int* indica la posizione del giocatore da ritornare.
- *BoardDeck* usa degli oggetti di tipo *ResourceCard* e *GoldCard* per contenere le carte posizionate sul tavolo (scoperte): questo diventa un problema nel momento in cui un mazzo finisce, perché in quel caso le carte scoperte devono comunque essere 4, indipendentemente dal loro tipo (vedi Slack, #2 Requirements, 21/03, 15:03, Giuseppe Laguardia).

    - Suggerimento: usare oggetti di tipo *PlayingCard*, in modo che sia *ResourceCard* che *GoldCard* possano essergli assegnati.

- Rischio di incorrere in problemi con la gestione degli obiettivi: una singola classe costringe ad inserire tutti i metodi di controllo lì dentro (con anche potenziali problemi di aggiornabilità futura) e costringe a mantenere tutte le informazioni in ognuno degli obiettivi, anche dove non servono.

    - Inoltre, se in futuro si dovrà aggiungere un obiettivo che richiede un parametro aggiuntivo, ad esempio un *tertiaryType*, la sua aggiunta richiede di recuperare tutti i punti in cui il costruttore è già stato usato e di aggiungere questo nuovo parametro in tutte le sue chiamate.
    - Infine, l'aver specificato in due parametri i *primaryType* e *secondaryType* può rendere scomoda l'introduzione di eventuali obiettivi con più di due colori. Potrebbe essere preferibile utilizzare una struttura dati dinamica per contenere i tipi delle carte da usare per gli obiettivi di posizionamento.

- *PlayingCard* contiene un array con dimensione fissa di 4 *Corner*: non è prevista la possibilità di avere angoli particolari sul retro della carta, cosa che va bene per le carte attualmente presenti nel gioco, ma potrebbe comportare problemi per potenziali aggiornamenti futuri.

- I *Corner* tengono conto della loro posizione sulla carta mediante degli *int*. Sarebbe consigliato utilizzare perlomeno una *enum* per rendere più chiaro quale numero è assegnato ad ogni angolo.

- *StartingCard* usa un *array* di *boolean* per indicare se al suo centro si trova una risorsa di un certo tipo. È chiaro il perché sia stato fatto, tuttavia in un'ottica di espandibilità questo potrebbe portare problemi se in futuro si volesse introdurre una carta con, ad esempio, 2 o più risorse dello stesso tipo al centro: in quel caso il valore *true* non basterebbe per indicare quante ce ne sono.

    - Una modifica semplice e relativamente indolore potrebbe essere quella di usare degli *int* invece dei *boolean*: 0 se non c'è quella risorsa, 1, 2, …, n se ce ne sono. **N.B:** rimarrebbe comunque il limite di usare un array di dimensione statica, che rappresenta un ulteriore problema di *future upgradability* come già evidenziato al primo punto.

- Il fatto che *DrawableDeck* si chiami così anche se contiene più mazzi in una *Map* può essere fuorviante. Sarebbe preferibile usare il plurale.

Confronto tra le architetture

- La gestione dello stato di gioco tramite i valori della *enum GameState* è un'ottima idea, da cui molto probabilmente prenderemo spunto quando dovremo approcciare la gestione della

comunicazione *Client-Server*.

- La gestione del *BoardDeck*, nonostante l'appunto fatto sopra, è migliore della nostra, che invece avevamo diviso in una serie di oggetti simili ma specifici per tipo: la situazione descritta dove un mazzo finisce e le carte scoperte devono venire tutte dallo stesso mazzo può essere gestita meglio da questa architettura rispetto alla nostra.