Peer-Review 1: UML

Abdallah Alkhetiar, Daniel Bonardi, Caterina Cardenas, Simone Ceccherini

Gruppo AM17

Valutazione del diagramma UML delle classi del gruppo AM26.

# Lati positivi

* Idea di suddividere le carte tra PlayableCard e PlayedCard, in modo da sapere se le carte sono già state piazzate o meno. Quindi risulta anche più facile controllare la correttezza dell’accesso ai dati corretti di una carta, infatti se una carta è stata piazzata è possibile ottenere informazioni riguardanti solo il lato di piazzamento.
* Creazione di una classe Lobby che permette di “registrare” i giocatori prima dell’inizio di una partita. È anche molto utile nel caso si vogliano gestire partite multiple.
* La presenza di un’unica sopraclasse Card, da cui ereditano tutte le altre carte giocabili, permette di usufruire del polimorfismo.

# Lati negativi

* A prima vista risulta molto difficile capire a cosa sono collegate le varie frecce, soprattutto per il fatto che molte si intrecciano tra di loro e a volte sembrano passare dietro ad alcuni blocchi.
* La classe ObjectiveCard eredita da Card, quindi deve contenere l’attributo kingdom (enumeratore) quindi, o viene settato a null oppure deve essere uno dei valori dell’enumerazione. Questo potrebbe avere senso per le carte obbiettivo che hanno un colore particolare, ma non per quelle che attribuiscono punti in base al numero di oggetti posseduti dal giocatore (QUILL, FEATHER, MANUSCRIPT).
* A livello concettuale del gioco, l’albero di ereditarietà delle carte ha senso tranne per il fatto che una carta oro eredita da una carta risorsa. Infatti, pensando ai metodi che possono essere chiamati dai 2 tipi di carte potrebbe aver senso questa implementazione, ma a livello logico lascia perplessi. Ad esempio, il metodo drawCard(…) ritorna una ResourceCard, il che, a prima vista, sembra sbagliato, perché sembra che quel particolare metodo ritorni sempre una carta risorsa, e mai una carta oro, ma ripensando all’albero di ereditarietà ha senso.
* Non c’è alcun metodo che faccia riferimento a come viene scelta e piazzata la carta iniziale, come vengono distribuite le 2 carte obbiettivo tra cui scegliere (nonostante ce ne sia uno che permette di scegliere un obiettivo segreto), la scelta delle pedine colorate e la scelta del primo giocatore ad eseguire un turno.
* Per poter utilizzare i metodi delle sottoclassi di Card, mantenendo il vostro modello, bisognerebbe fare casting, il che non è una buona pratica. Ad esempio: metodo drawCard() in GameMaster ritorna una ResourceCard, ma essendo il tipo statico di tutte le carte Card, bisognerebbe castare alla classe richiesta.
* Non è molto chiara la suddivisione del pattern MVC. Il GameMaster sembra essere un controller molto pesante e la lobby un model molto leggero. Non si riesce bene a capire il ruolo di GameMaster all’interno del pattern.
* Gli enumeratori ObjectiveType e Corner potrebbero essere tolti in favore di una struttura di classi e sottoclassi, in modo da usufruire il più possibile del polimorfismo e rendere più leggibile non solo il codice ma anche il grafico UML.
* Non si capisce la gestione dei turni. Sembrano esserci i metodi per la gestione, ma non gli attributi in cui salvare le informazioni necessarie. Ad esempio, getCurrentPlayer() non sembra poter prendere l’informazione da nessuna parte.
* Alcuni nomi di variabili non sono molto esplicativi, quindi non si capisce quale dovrebbe essere la loro utilità.

# Confronto tra le architetture

* Per quanto riguarda l’implementazione di una classe Lobby è molto utile non solo per la gestione dei giocatori nel prepartita, ma potrebbe anche facilitare l’implementazione di una delle funzionalità avanzate, le partite multiple. L’unica cosa a cui sarebbe meglio prestare attenzione è la creazione di un unico controller con tutti i metodi necessari per gestire le varie partite, quindi il server più pesante e dei client più leggeri, in modo da lasciare anche i controlli più importanti al server, senza doverli effettuare due volte.
* Nel nostro caso potrebbe servire un metodo che controlla se un deck è vuoto, e nel caso si debbano rivelare delle carte sul banco, bisogna aggiungere dei metodi che gestiscono il caso in cui solo uno dei due deck è vuoto.
* Bisogna implementare anche dei controlli per ogni metodo chiamato su una carta per capire se il lato corrente è fronte o retro.