Peer-Review 1: UML

Giovanni Manfredi, Mattia Martelli, Sebastiano Meneghin
Gruppo 27

Valutazione del diagramma UML delle classi del gruppo 26

Lati positivi

Abbiamo trovato nell'utilizzo del pattern *facade* una soluzione molto elegante, in quanto semplifica l'accesso ai dati contenuti nel model e nasconde la loro effettiva implementazione a terze parti, permettendo agilmente modifiche strutturali. Abbiamo inoltre apprezzato la bassa frammentazione in classi di concetti adiacenti, che permette dunque di avere uno schema molto compatto, pur possedendo un elevato grado di chiarezza all'atto della comprensione dello stesso.

Lati negativi

L'utilizzo di una struttura dati dedicata per il salvataggio dell'effetto correntemente attivo sembra superfluo, poiché non è un dato, bensì un'azione riguardante lo stato globale. Medesimo discorso vale per l'attributo `winner`, in quanto la partita termina una volta individuato il vincitore, portando ad un'eliminazione dei dati.

A giudicare dagli attributi della classe `Character`, sembrerebbe esserci un fraintendimento riguardante le regole di gioco: parrebbe che il costo di ogni carta aumenti ad ogni utilizzo, mentre dalle regole si evince che questo dovrebbe accadere solo al primo utilizzo. Per quanto riguarda la gestione delle sottoclassi, è necessario prestare attenzione ai metodi non specificati nella classe padre, in quanto questi non saranno visibili staticamente.

Per come descritta, la classe `Island` non favorisce l'unificazione delle isole, in quanto questa sembrerebbe essere delegata a terzi. Inoltre, l'utilizzo di attributi interi per gestire la presenza delle *no entry tile* potrebbe generare errori, potendo essercene al massimo una per isola. Suggeriamo dunque di sostituirli con dei booleani. L'approccio adottato per indicare la posizione della pedina *mother nature*, ovvero quello di renderla un puntatore all'isola su cui si trova, potrebbe risultare di difficile gestione ogniqualvolta si necessita un suo spostamento, in quanto si lega al discorso della gestione dell'unificazione delle isole. Infine, l'attributo `influences` sembrerebbe rappresentare informazioni parzialmente sovrapponibili con l'attibuto `students`, generando potenzialmente errori legati a dati contrastanti.

La funzione `addStudents` presente in `Cloud`, avendo un numero fisso di parametri, potrebbe generare problemi nel caso di partite a tre giocatori, poiché il numero di studenti è differente rispetto alle altre tipologie.

La classe `Bag`, rappresentante il sacchetto in cui sono contenuti gli studenti, non permette di aumentare a runtime i contatori al suo interno presenti: ciò rende impossibile l'implementazione di una carta personaggio.

La fusione dei concetti *SchoolBoard* e *Player* parrebbe essere incompleta: la *DiningRoom* è difatti a sé stante, divisa in una classe `DiningTable` istanziata cinque volte, una per tavolata, dalla classe `Player`. Ci sembra che ciò crei problemi di gestione facilmente risolvibili implementando questo concetto come una mappa di tipo `StudentColor` $\rightarrow$ `int`, eliminando la necessità di avere una classe separata per ogni contatore.

Confronto tra le architetture

Per quanto riguarda la gestione di strutture dati complesse, il loro approccio sembra concentrarsi sui tipi `List` e `HashMap`. In generale, sembra collocarsi su un livello di astrazione più elevato rispetto al nostro, che si basa su `Array`: ci sembrava superfluo utilizzare delle mappe quando possiamo facilmente avere coerenza sfruttando la numerazione intrinseca dell'enumeratore `StudentColor`, mentre abbiamo preferito non utilizzare liste in quanto abbiamo trovato comodo basarci sugli indici posizionali. Ci ha fatto però riflettere sull'effettiva bontà del nostro approccio, che potrebbe essere in certi punti aggiornato in favore di soluzioni più eleganti.

Infine, il loro model risulta sotto certi aspetti più corposo del nostro, implementando anche funzioni che riguardano la logica del gioco. Noi abbiamo preferito delegare tutta la logica al controller, rendendolo molto complesso, in favore di un model limitato a funzioni di rappresentazione dei dati. Stiamo pensando di fondere i due approcci per certe funzionalità, aggiungendo al nostro model funzioni che potrebbero alleggerire il carico delle altre componenti del progetto.