Peer-Review 1: UML

Francesco Re, Ruben Scopacasa, Riccardo Zappa
Gruppo GC13

4 aprile 2022

Valutazione del diagramma UML delle classi del gruppo GC23.

1 Lati positivi

1. La gestione del modello di gioco è ben distribuita tra le varie classi, in quanto ognuna persegue uno e un solo obiettivo. Notiamo l'assenza di classi BLOB.

2. La classe `Game` è la classe che coordina l'esecuzione e lo svolgimento del gioco, ed espone tutti i metodi necessari a interagire con il model: l'utilizzo del pattern Facade ci sembra un'ottima idea.

3. La classe `Bag` gestisce il processo di estrazione casuale degli studenti senza reinserimento, in questo modo diventa facile da testare e riutilizzabile all'interno del progetto, se necessario in futuro.

2 Lati negativi

1. L'enum `Colors`, in quanto rappresenta colori di entità logicamente distinte, andrebbe diviso in `StudentColor` e `TowerColor` sia per una questione di leggibilità del codice che per coerenza dello stesso (e.g. evitare di avere una torre rosa o uno studente grigio).

2. In tutto il progetto viene utilizzata una `HashTable<StudentColor, Integer>` per modellare un contenitore di

studenti. Sarebbe utile creare una classe apposita (e.g. `StudentsSet`) che gestisca tutte le operazioni su questo tipo di insieme, come l'aggiunta e la rimozioni di studenti, con le dovute eccezioni quando necessario.

3. La classe `StudentDisc` potrebbe essere rimossa in quanto non porta nessun reale vantaggio al progetto. L'enum `StudentColor` può essere utilizzato direttamente al posto di `StudentDisc`. Un simile ragionamento vale anche per `ProfessorPawn`.

4. Le carte personaggio richiedono ancora ulteriore sviluppo, potenzialmente utilizzando pattern adatti.

5. La classe `IslandTile` contiene il metodo `sumTowersUnification(towers : int)` che si occupa di unificare le isole. È quindi il metodo chiamante che si deve occupare di controllare che le isole siano compatibili e che non vengano commesse azioni illegali (e.g. unificare isole conquistate da giocatori diversi). Sarebbe più corretto creare un metodo `merge(anotherIsland : Island)` che esegua tale operazione operando i dovuti controlli.
   Sempre in questa classe è presente il campo booleano `hasMotherNature` che memorizza su quale isola è posizionata Madre Natura, ma questa informazione è già presente nella classe `Game` in `motherNaturePosition`. Inoltre ci sembra più oppurtuno memorizzare il proprietario dell'isola come un oggetto `Player`, o alternativamente con il colore delle torri di quel giocatore, piuttosto che con una stringa arbitraria.

6. Nel modello manca la gestione delle monete (modalità esperto) dei giocatori.

7. L'utilizzo di classi per la gestione dei mazzi di carte ci sembra sovraingegnerizzato, in quanto sembra appesantire il progetto senza tangenti benefici. Sicuramente più avanti nello sviluppo sarà più chiaro se questa scelta si rivelerà vantaggiosa o meno.

3 Confronto tra le architetture

Le due architetture condividono molti aspetti e scelte progettuali. Tra tutto spicca il diverso approccio per quanto riguarda i professori, che vengono qui memorizzati nella `SchoolBoard` del giocatore sotto forma di lista piuttosto

che nel `Game` come mappa.
Anche le nuvole vengono gestite diversamente, in questo caso è stata creata una classe apposita `CloudTile` mentre nel nostro progetto una nuvola è rappresentata semplicemente con un contenitore di studenti.
Inoltre, il `Player` delega a `SchoolBoard` la gestione della plancia, mentre la nostra soluzione propone di gestire la plancia direttamente all'interno del `Player`; riteniamo valide entrambe le soluzioni.

4 Nota

Il gruppo ci ha consegnato autonomamente una versione rivisitata e migliorata dell'UML andando a risolvere proattivamente diversi problemi riportati in questa relazione (tra cui i numeri 1, 3, 5[parzialmente]).