Peer-Review 1: UML

Moretti Valentina, Franzè Lorenzo, Nonino Lara

Gruppo 22

Valutazione del diagramma UML delle classi del gruppo 21.

Lati positivi

Osservando il diagramma UML si può subito notare che il model è stato realizzato tramite una classe principale GameModel composizione delle classi characterCard, Bag, Cloud, Player ed Island. Questa struttura del diagramma si oppone a quella di un unico blocco monolitico rappresentante una classe troppo ricca di metodi e attributi. Riteniamo che questa struttura distribuita sia una buona scelta di progetto in uno stile di programmazione ad oggetti in quanto realizza un'ottima ripartizione del carico lavorativo alle diverse classi.

Per la gestione degli studenti è stata utilizzata una Map<House, int> che riteniamo sia la struttura più adatta in quanto comoda per il conteggio degli studenti in base al loro colore e alla gestione correlata dell'incremento o rimozione di essi.

Non è stata utilizzata una classe rappresentante la singola isola. Island, infatti, è stata utilizzata per rappresentare un gruppo di isole che si aggregano durante il gioco. Quando si uniscono due isole si incrementa il valore dell'attributo numTowers. Infatti, non e necessario gestire il colore delle Towers che si sommano in quanto due isole adiacenti che si uniscono hanno lo stesso valore dell'attributo rappresentante il colore delle torri.

Come all'interno di Island, anche in Dashboard non è stata utilizzata un'entità Tower, ma si è preferito l'uso dell'attributo towerColor (tipo di enumerazione) e l'attributo numTowers per il conteggio delle stesse. Riteniamo questa una buona scelta perché questi due attributi sono le uniche informazioni necessarie e sarebbe eccessiva la costituzione di una classe apposita Tower.

Lati negativi

Per l'implementazione dei personaggi si è scelto l'utilizzo di una classe astratta da cui ereditano le 12 carte personaggio. Riteniamo che questa sia una scelta valida in quanto tutte le carte personaggio posseggono gli attributi cost, firstUse, inUse ed i relativi metodi. Tuttavia gli attributi houseMap e noEntryTile sono propri soltanto di alcune di queste sottoclassi.
Si sarebbe potuto implementare questo costrutto omettendo dalla classe characterCard gli attributi noEntryTile ed houseMap. Questi sarebbero stati propri soltanto delle classi che effettivamente ne fanno uso.

L'attributo islandList all'interno della classe GameModel non è necessario in quanto si è utilizzata la "freccia" di aggregazione tra le due classi (Island e GameModel). Questa ridondanza è impropria in UML.

Non è stata inserita una struttura per il corretto conteggio delle monete dei i giocatori.

Nell’enumerazione Card, si è dimenticato di elencare il nome dei 10 elementi.

Le isole si aggregano durante il gioco quindi il loro numero totale diminuisce progressivamente. Manca dunque un metodo per la rimozione di un elemento dalla lista IslandList di GameModel necessaria nel momento dell’unione di queste.

In più classi si è dimenticato il metodo per la rimozione degli studenti dalla Mappa houseMap.

Confronto tra le architetture

Una prima differenza tra gli elaborati può essere individuata nella scelta della realizzazione delle Assistant Cards. Il gruppo 21 ha utilizzato un’enumerazione per le 10 Card: riteniamo questa una scelta valida. Il gruppo 22 preferisce tuttavia mantenere la propria scelta di creare una classe apposita con i relativi attributi (turnOrder e movementsMotherNature).

Come è stato fatto dal gruppo 21, anche il nostro gruppo, una prima scelta progettuale aveva portato il nostro gruppo a realizzare le 12 carte personaggio con l’ereditarietà da una <<abstract class>> Character. Tuttavia questa scelta è stata poi abbandonata perché si è preferito gestire i diversi effetti delle carte personaggio nel Controller piuttosto che nel Model.

Visto il frequente uso della Map<House, Int> in diverse classi (situazione presente nei diagrammi UML di entrambi i gruppi), il nostro gruppo ha trovato comodo, oltre che stilisticamente più elegante, definire una classe a parte che ridefinisse questa mappa. Questa classe rappresenta la mappa con l’aggiunta di metodi che effettuano operazioni specifiche sulla mappa stessa in modo più facile ed immediato. In questo modo risulta più comodo e ordinato effettuare chiamate a questi metodi dalle altre classi invocando i metodi della classe rappresentante la mappa al posto di lavorare direttamente sulla mappa. Si può evitare così di scrivere codice ripetuto per effettuare determinate operazioni che sono necessarie di frequente.