Peer Review l: UML

Enrico Vento, Federico Valentino, Giacomo Verdicchio

GC61

Valutazione UML (model) del gruppo GC06

3/4/2022

1. Lati Positivi

1. *Estensivo utilizzo delle Enumerations*

   Apprezziamo particolarmente l'uso intelligente delle *enumerations* in funzione di semplificazione nell'identifcare determinate entità, come, per esempio, i vari *wizards*.

2. *Gestione coins*
   Altro lato positivo lo riscontriamo nella gestione dei *coins* all'interno di *school* e non di *player*: in questo modo si semplifica la loro gestione essendo *school* in rapporto uno ad uno con *player*.

3. *Gestione Characters*
   In questo caso la positività dipende fortemente dal tipo di implementazione che verrà scelto in fase di *controller*. Sta di fatto che, gestendo i vari *characters* tramite una classe standard e una semplice *enumeration*, al netto di una implementazione di scelta effetto intelligente a lato *controller*¸ si va ad evitare la creazione di numerosi oggetti *characters* che risulterebbero molto simili. Inoltre, questa soluzione favorisce grandemente l'espandibilità.

4. *GameModel con Possibilità di Istanze Multiple*
   Apprezzabile l'avere un ID che permette di distinguere fra varie istanze di *gameModel,* in funzione della gestione di un contesto multi-partita.

2. Lati Negativi

1. *GameModel Troppo Ricco*
   La gestione della logica risulta estremamente accentrata nella classe di *GameModel*, che contiene metodi relativi a classi dalle funzionalità diversissime. Nella nostra ottica, spostare alcuni pezzi di logica relativi ad ambiti specifici, come ad esempio la gestione dei gruppi di isole, in una classe apposita, strettamente legata alle isole stesse, sarebbe preferibile. Pur capendo la semplicità di utilizzo e di accesso ai metodi derivante dall'approccio corrente, una distribuzione degli stessi che guardi più alle funzionalità comuni risulterebbe sicuramente più leggibile.
   Il risultato è la presenza di una classe che copre da sola funzionalità molto eterogenee; anche non volendo distribuire la logica sulle classi interessate dalla stessa, approccio dal quale deriverebbe una certa complessità di gestione, si sarebbe potuto pensare all'utilizzo di apposite classi di servizio.

2. *Attributo GameState*
   Nella classe *GameModel* risulta un attributo di tipo *GameState*, tipo che però non risulta essere definito in nessun luogo nel modello.

3. *Metodi Gestione Studenti*
   Sempre all'interno di *GameModel* si nota la presenza di vari metodi specifici che consentono di posizionare uno studente precedentemente pescato dal sacchetto su entità differenti: sarebbe stato più utile o definire una classe di servizio a parte che si occupasse di questo, oppure, perlomeno, pensare all'utilizzo dell'*overloading* per sopperire al problema di avere metodi dalla funzionalità molto simile ma con nomi tutti diversi. L'attuale implementazione risulta un po' ridondante.

4. *Gestione NoEntry Tiles*
   La gestione delle tessere divieto crea non poche perplessità; esse vengono istanziate come oggetti (essendo definite in un'apposita classe), ma questi oggetti, pur essendo legati al *GameModel*, non vengono salvati da nessuna parte. Anzi, la struttura che dovrebbe contenere l'informazione sulla loro presenza (o assenza), ovvero l'isola, lo fa non attraverso gli oggetti stessi, ma attraverso una variabile booleana. Alla luce di ciò, il trattare queste entità come veri e propri oggetti sembra essere superfluo.

3. Confronto Architettura

Alla luce di quanto detto, possiamo trarre delle conclusioni e tracciare delle linee di paragone fra l'architettura scelta da noi, e quella proposta dal gruppo revisionato.

Innanzitutto, si nota un approccio completamente diverso rispetto alla gestione della logica di base presente nel model, e nella scelta di quali aspetti del gioco modellare come effettivi oggetti. Nel nostro caso, contrariamente a quanto visto nell'*UML* qui preso in esame, si è optato per distribuire la logica su più classi, accedendoci tramite riferimenti alle stesse salvati nel *GameState*; in secondo luogo, è stata presa la scelta di modellare come veri e propri oggetti le entità di gioco di un certo valore logico o grafico (abbiamo ritenuto inutile modellare Torri e Professori come oggetti a sé, avendo essi un'unica proprietà, ovvero il colore; discorso simile vale per le tessere divieto).
Due approcci differenti, e quasi opposti in alcuni casi, ma ognuno con i suoi pregi e difetti.

Altra rea di grande differenza è la gestione dei *characters*: si è deciso, a differenza di quanto fatto dal gruppo revisionato, di modellarli come oggetti separati (uniti dall'ereditarietà con la classe astratta *CharcaterCard*) e privi di metodi: questi verranno dettagliati a livello di *controller*, e verranno scelti per *overloading* passando il tipo di carta

giocata. Se da un lato nella nostra architettura si denota una corrispondenza, in questo caso, più stretta con l'effettivo oggetto di gioco (cosa che potrebbe potenzialmente semplificare il lavoro a livello grafico), dall'altro si vanno a creare numerosi oggetti molto simili fra loro, e bisogna riconoscere che la soluzione adottata dal gruppo revisionato è molto intelligente in questo senso, e ne terremo conto in futuro.

Un'altra caratteristica da cui possiamo prendere spunto è l'estensivo uso di *enumerations*, che renderebbe più semplice la gestione di determinate entità da noi adoperate; prenderemo questo aspetto sicuramente come esempio.