Peer-Review 1: UML

Riccardo Inghilleri, Laura Daniela Maftei, Manuela Merlo
Gruppo 35

Valutazione del diagramma UML delle classi del gruppo 34.

Aspetti valutati

Analizzando l'UML ci siamo soffermati in particolare sui seguenti aspetti:

- Classi scelte
- Attributi e metodi definiti
- Logica applicativa inserita nel model

1 Lati positivi

Per quando riguarda le classi scelte, abbiamo trovato principalmente 3 lati positivi.

Gli studenti : una delle scelte che reputiamo migliore è il non aver definito un oggetto per lo studente che, per come viene utilizzato, si riduce al solo attributo Colore. La gestione negli oggetti che possiedono Studenti attraverso l'uso di contatori è a nostro parere molto efficiente, facendo sempre attenzione alla corrispondeza tra i diversi colori durante lo spostatemento metaforico dello studente da un oggetto all'altro.

Gli Handler : la suddivisione della gestione dei singoli oggetti attraverso Handler specifici, limitano la dimensione delle singole classi rendendo il model distribuito.

Le carte speciali : La gestione degli effetti speciali nel model potrebbe essere una buona idea. La validità di questa scelta sta tutta nell'implementazione (che purtroppo per molte carte non è stata riportata). I principali problemi che il nostro gruppo ha riscontrato nella gestione di questa parte della logica applicativa nel model sono:

- la gestione degli effetti delle carte in termini di durata e momento in cui vengono giocate. Infatti, per alcune carte l'effetto è immediato (lo spostamento di uno studente) per altre invece permane dal momento in cui viene scelta fino alla fine del turno del giocatore (le carte che modificano il calcolo dell'influenza).
- la gestione dell'effetto della carta avente i divieti che deve essere gestito nonostante la carta sia stata scelta in un altro turno.
- la necessità da parte delle diverse carte di parametri diversi.

Esempi: la carta che calcola l'influenza senza tenere conto di un colore scelto dall'utente necessita di un paramentro in ingresso. In generale le carte che permettono movimenti aggiuntivi degli studenti necessitano della posizione di partenza, di arrivo e dei colori scelti.
Dunque il solo metodo `useEffect()` secondo noi non è sufficiente.

2 Lati negativi

Durante l'analisi dell'UML abbiamo riscontrato alcuni difetti che in generale riguardano :

- La mancanza dei costruttori nelle Classi
- La mancanza in molte classi di Getter e Setter
  Esempio : `getNickname()` nel Player
- La mancanza di alcuni parametri nei metodi
  Esempio : calculateInfluence() non ha alcun parametro di ritorno e nemmeno un parametro in ingresso per capire in quale isola vada calcolata l'influenza.

Per quanto riguarda gli aspetti negativi:

Le isole : è stata utilizzata una classe astratta chiamata *Island* che secondo noi posside degli attributi che sono specifici per il Gruppo di isole e che quindi non dovrebbero essere ereditati dalla singola isola.
Secondo noi, per come è stata realizzata, la suddivisione in NormalIsland e IslandUnion non è necessaria. L'IslandUnion non è una classe a sè stante, bensì è una composizione di oggetti di tipo NormalIsland di cui possiede i riferimenti, che mantengono una propria indipendenza. Il ruolo contenitore si nota in particolare durante il calcolo dell'influenza sull'IslandUnion:

1. Accesso alla singola isola
2. Calcolo dell'influenza
3. Unione dei singoli risultati

Per noi la possibile soluzione è definire un singolo oggetto isola che possa comportarsi da singola isola (attributo `NumTower` =1) oppure da Gruppo di isole (attributo `NumTower` $>$ 1). Quindi durante il gioco, nel caso in cui due isole andassero unificate, incorporare gli elementi di un'isola nell'altra. Potrebbe essere mantenuta anche la suddivisione presentata, creando un nuovo oggetto di tipo IslandUnion e riunendo gli attributi di tutte le isole da unificare.
In questo modo il metodo `getInfluence()` utilizzato per Normalisland potrebbe essere riutilizzato anche per IslandUnion.

Gli Handler : nonostante la suddivisione degli Handler renda i singoli oggetti meno elaborati, la gestione del gioco complessivo è difficile. Rimanendo sull'esempio della `getinfluence()`, l'isola deve:

1. accedere al Player che gli viene passato come parametro
2. accedere alla Board
3. verificare quali professori sono presenti
4. calcolare l'influenza

Avere una gestione centralizzata, per quanto possa rendere il codice più complesso, permette di avere i paramentri per i calcolo dell'inlfuenza tutti nello stesso posto.

ExpertMode : secondo noi sono stati mischiati aspetti appartenenti alla modalità base e la modalità esperta.
Esempio: Player, nonostante non possa avere sempre delle monete o della infleunza aggiuntiva, ha 2 attributi di tipo `coins` e `additionalInfluence()`.

Metodi Ridondanti : Alcuni metodi, con dei paramentri in ingresso, potrebbero essere tolti: Esempio: Board possiede due metodi,
`setProf(professorColor : Color)` e `removeProf(professorColor : Color)` che potrebbe essere distinti con un paramentro di tipo `boolean`.

GameSettings : nonostante l'idea di avere dei paramentri di setting sia buona per poter gestire vari controlli durante il gioco, oppure per gestire azioni automatiche come il riempimento delle isole, forse l'idea poteva essere realizzata meglio:

1. `numTowerPerPlayer` dovrebbe essere un array, stessa cosa vale per `numStudentsInHall`.

2. `numCloud` non è un paramentro variabile, quindi non è necessario.
3. Due parametri interessanti da aggiungere per un controllo di terminazione del gioco potrebbero essere `numIsland` e `numStudent`.
4. Tra i metodi è stato aggiunto `getNumPlayer()` che però non c'è tra gli attributi.

Deck : Dato che le carte assistente non possono essere rigiocate, forse non è necessario tenere traccia delle carte scelte nei turni precedenti. É sufficiente salvare l'ultima carta.

CardName : Non è necessario un Id poichè ad ogni enum viene assegnato un indice di default.

3 Confronto tra le architetture

Complessivamente le due architetture:

Si assomigliano : per quanto riguarda la scelta della maggior parte delle classi e la gestione di molti metodi, come lo spostamento degli studenti gestiti dagli handler;

Differiscono :

- per la gestione della logica del gioco, soprattuto quella in modalità expert.
  Infatti, mentre il nostro gruppo ha preferito gestire la logica delle carte speciali con uno `Strategy Patter` a livello controller, il gruppo 34 ha optato per l'estensione di una singola Classe Character a livello Model.
  Inoltre, la modifica dello stato delle classi rappresentanti elementi atomici, è gestita dal gruppo 34 da piccoli handler, mentre noi abbiamo scelto un'unica classe centralizzata che modificasse tutti gli elementi facenti parte del tavolo da gioco così che avesse la visione di insieme.
- per le dipendenze tra classi. Infatti, l'aver scelto uno schema distribuito, ha portato ad una diversa aggregazione degli elementi. Dal nostro punto di vista, il Player, rappresenta una Classe il cui compito è salvare lo stato di un Client e dunque non si occupa di modificare la scuola che gli appartiene.

I pregi che pensiamo di poter integrare nel nostro progetto sono:

1. L'utilizzo degli studenti tramite contatori. Data la semplicità della classe e l'indistinguibilità tra oggetti è una scelta implementativa migliore.
2. L'utilizzo di una classe che raccolga lo stato generale del Game per poter eseguire in modo più efficiente il check sulle condizioni di terminazione e parametri di default per le fasi di setup dei turni, senza dover accere ad ogni singolo oggetto.