Peer-Review 1: UML

Francesco Arcuri, Giulia Prosio, Carlo Ronconi
Gruppo 7

5 aprile 2022

Valutazione del diagramma UML delle classi del gruppo 6.

1 Lati positivi

Esaminando il grafico UML del progetto elaborato dal gruppo 6, la nostra attenzione è stata catturata da alcuni punti e aspetti di forza. Tra questi troviamo l'utilizzo di classi più piccole e specifiche che si interfacciano direttamente alla classe GameModel (nonché classe principale del model) in modo da poter essere coordinate con facilità. Alcune di queste sono collegate tramite classi che si possono definire "ponte", come ad esempio PlayerHandler. Quest'ultima contiene una lista di Player e altri attributi che identificheranno il numero di maghi, nonché giocatori, e anche quale mago dovrà fare la sua mossa nel turno corrente. Un ulteriore esempio è IslandHandler che contiene una lista di Island. A parer nostro, si sono rivelate funzionali perché il loro ruolo è quello di agevolare la gestione del gruppo (Player e Island) ad ogni turno e non invece il singolo.

Focalizzando il nostro interesse sulle liste e sull'utilizzo che ne è stato fatto, possiamo osservare che nella classe Cloud è stato inserito un attributo di tipo ArrayList e non un array con dimensione prefissata, come nel nostro caso. Così come in altre classi del progetto, all'interno della classe TeacherHandler è stata utilizzata l'interfaccia Map<key,value>. In questo caso la key è una classe Enum, a differenza del value che non è rappresentato da un'ennesima classe Enum, come fatto da noi, ma direttamente dal riferimento al player che gode dei benefici che il professore offre.

Secondo la nostra visione, un'altra interpretazione corretta è stata data alla gestione delle torri sulle isole. Queste vengono viste di "proprietà" di un player e lo si capisce dall'attributo "islandOwner" di tipo Player definito. Per questo motivo non è necessario definire una classe Enum contenente costanti che definiscono i colori delle torri.

2 Lati negativi

Analizzando l'UML proposto dal gruppo 6 abbiamo anche osservato alcuni elementi nella struttura a nostro avviso migliorabili. In particolare la gestione della classe Island ci sembra scomoda: avendo la classe IslandHandler come array delle isole ogni volta che avviene un'unificazione è necessario eliminare una delle isole dall'array in modo che venga "inglobata" dall'altra a formare un'isola unica, il che può creare problemi di gestione. In questo modo anche tenere conto di dove si trovi la posizione di Madre Natura diventa complicato.

Inoltre per quanto riguarda la classe Board, la StudentEntrance è trattata come ArrayList, per cui però la gestione di accesso e modifica degli elementi risulta costosa, soprattutto in un contesto in cui ciò avviene ad ogni turno, quando il giocatore sceglie tre pedine in posizioni casuali da disporre tra le isole e la student room. Abbiamo pensato che alternativamente, visto che hanno trattato l'attributo studentRoom come una mappa, potrebbero usare questo approccio anche per la studentEntrance. Infine non hanno ancora implementato in questa prima versione dell'UML l'alternativa di gioco a quattro giocatori.

3 Confronto tra le architetture

Nell'analisi dell'architettura proposta dal gruppo 6, abbiamo trovato in generale una struttura comparabile a quella da noi proposta:

- la scelta di non creare una classe per studenti e professori, perché il loro unico scopo nel gioco è quello di fare da segnaposto ed essere contati, ma quella di creare invece una enum con i colori degli studenti/professori

- l'utilizzo di una mappa per il conteggio sia del numero di pedine studente per ogni colore nelle varie posizioni del gioco, che della posizione dei professori, centralizzata della classe GameModel
- la presenza di una classe centrale GameModel, che fa da punto di accesso agli aspetti principali rappresentati dal package Model
- la presenza di elementi comuni a tutti i giocatori (Cloud, Bag, Islands) e quella di una board personale per ogni giocatore.

Pur essendo presenti somiglianze nella logica adottata per la rappresentazione del model, scendendo in maggiori dettagli si possono trovare numerose differenze.

Nella gestione della mappa dei professori abbiamo trovato nella soluzione dell'altro gruppo una miglioria che apporteremo al nostro schema: anziché mappare il colore dei professori con un'altra enum (ProfessorPosition) contenente le possibili posizioni di tali pedine, visto che tali posizioni sono effettivamente solo i giocatori, mapparlo con i giocatori (PlayerTool) della partita. Altra differenza che abbiamo trovato, di carattere più generale, è la presenza di classi più piccole e numerose, in luogo di classe grandi e complesse come le nostre. Ci siamo resi conto iniziando a scrivere il codice del Model che il vantaggio di scorporare le classi e renderle più specializzate è che diventano più gestibili, mantenibili e più facilmente modificabili. Abbiamo quindi deciso di modificare la nostra architettura e scorporare alcune classi, come ad esempio quella di PlayerTools, che avevamo inizialmente pensato come comprendente sia la scuola, che il mazzo di carte assistente, che le informazioni sul Player.

Un'altra differenza marcata che abbiamo trovato nel confronto delle due architetture è quella della gestione delle isole, per la quale si rimanda al paragrafo precedente, in quanto ci è sembrato che la nostra soluzione portasse più vantaggi di quella adottata dall'altro gruppo. Abbiamo trovato nell'architettura del gruppo 6 la classe character necessaria per lo sviluppo di funzionalità avanzate, e ci siamo riproposti di aggiungerla nella nostra soluzione, seppur in maniera differente: creando una classe genitore Card, dalla quale discendano sia le carte AssistantCard che quelle CharacterCard.

In seguito, un elenco riassuntivo di tutte le modifiche che ci siamo riproposti di adottare in seguito all'analisi della soluzione del gruppo 6, comprendente anche dettagli per i quali non è necessaria una discussione ulteriore:

1. modifica del nome della classe Game in GameModel per maggiore chiarezza

2. modifica della mappatura adottata nella ProfessorMap, passando da <PawnColor, ProfessorPosition> a <PawnColor, Player>

3. introduzione di più classi intermedie semplici in luogo delle poche classi complesse, in particolare suddividendo in classi più piccole le due macro-classi PlayerTools e SharedBoard

4. sostituzione dell'attributo enum TowerColor in IslandTile con un riferimento diretto al Player proprietario dell'isola, per rendere più semplice risalire da un'isola al suo occupante

5. rendere l'attributo clouds in SharedBoard una List invece di un array, per permettere di avere 2, 3 o 4 cloud (a seconda del numero di giocatori)

6. aggiunta di carte Character come estensione della classe Card, da cui ereditano anche le AssistantCard

In conclusione, l'analisi della architettura adottata da un altro gruppo ci è stata molto utile perché, se da un lato ci ha confortati nel trovare molte somiglianze con la logica con la quale si è scelto di rappresentare il modello del gioco, dall'altra ci ha permesso di ispirarci ad alcune idee, per adattarle alla nostra soluzione e migliorarla.