Peer-Review 1: UML

Carlo Sgaravatti, Jacopo Maria Polverini, Luca Romano

Gruppo GC34

Valutazione del diagramma UML delle classi del gruppo GC44.

Lati positivi

I lati positivi che abbiamo constatato dall'UML del progetto del gruppo 44 sono:

- La classe astratta Initializer e le sue sottoclassi perché rende flessibile l'inizializzazione del gioco (in base al numero di giocatori e alla difficoltà di gioco) tramite l'utilizzo del pattern strategy.
- La classe Field perché tiene in memoria lo stato della partita separatamente dal Game così da poterlo recuperare in caso di interruzione e perché permette di separare lo stato della partita dalle regole del gioco (in modo da poter inviare facilmente le informazioni sulla partita alle view dei client).
- In generale, il diagramma sembra essere completo per la parte di regole semplificate, ovvero dovrebbe essere in grado di coprire tutte situazioni di gioco (tranne le carte personaggio che non sono ancora implementate).

Lati negativi

I lati negativi che abbiamo constatato dall'UML del progetto del gruppo 44 sono:

- Professor, Tower e MotherNature non necessitano di essere implementate con una classe ma possono essere inserite come semplici attributi (usando dei tipi primitivi) all'interno delle classi a cui possono appartenere per facilitarne la gestione.
- La gestione dei turni di gioco e la fase in cui il giocatore attivo si trova (TurnPhase) si trovano in classi diverse. Nonostante queste classi siano comunque accessibili dal ModelHandlerSimple, sarebbe più semplice avere questi due aspetti concentrati in un unico posto, il che potrebbe anche aiutare a capire più facilmente quando il turno di un giocatore è finito e se il giocatore che ha richiesto di fare una certa azione sia autorizzato a farla (cioè se sia il suo turno e se l'azione richiesta sia quella corretta rispetto alla fase del turno).
- Non c'è un vero e proprio controller; viene gestito tutto dalla classe principale del model ModelHandlerSimple, che fa anche da controller. Di conseguenza, il controller viene visto come parte del model, il che non rispetta il pattern MVC. Inoltre, tutto lo stato del gioco viene salvato direttamente in tale classe, che è associata a diverse classi del model. Questo può rendere difficile l'interazione tra controller e model; sarebbe più opportuno che il controller potesse accedere al model solo tramite una classe per avere una visione più ad alto livello del model.
- ModelHandlerSimple viene salvato come attributo della classe Game, che da quanto dichiarato dovrebbe fare parte della view. Dunque, la view ha diretto accesso al controller, il che non rispetta il pattern MVC in quanto è il controller a dover osservare la view e a dover inviare le informazioni alla view.
- Le classi ProfessorTable, StudentTable, Entrance e Garrison possono essere raggruppate all'interno della classe School; questo per permettere una più facile modifica dei loro valori.

- Anche se i Character non sono ancora implementati, la classe CharacterDeck è assorbibile all'interno di FieldExpert. In particolare, all'interno di FieldExpert c'è già una lista di Character, dunque la classe CharacterDeck presenta una ridondanza.
- L'UML presenta la maggior parte delle associazioni con la freccia rivolta al contrario.

A fronte di questi aspetti, riteniamo di poter dare i seguenti suggerimenti, che potrebbero aiutare a migliorare l'architettura:

- Dividere le classi in package, sarebbe opportuno avere un package per il model, uno per il controller e uno per la view. Questo per poter separare al meglio ciascun componente e per non confondere le funzioni a cui deve adempiere ciascun componente.
- Nonostante non siano ancora implementati, pensiamo che utilizzare il pattern strategy per implementare i Character non sia attuabile in quanto i diversi personaggi hanno effetti molto diversi tra di loro e quindi i metodi che li implementano difficilmente possono essere riferibili alla stessa interfaccia (cioè non possono avere la stessa segnatura in quanto potrebbero avere dei parametri molto diversi tra loro)
- Alcune classi *Expert (in particolare FieldExpert e ModelHandlerExpert) potrebbero essere difficili da usare in quanto aggiungono dei metodi che non sono chiamabili se il tipo statico è la classe non Expert. Di conseguenza, per chiamare tali metodi è necessario utilizzare l'instanceof e il casting.

Confronto tra le architetture

Abbiamo trovato molto interessante l'idea della classe Field per mettere lo stato del gioco in una classe separata e per salvare lo stato della partita. Questo aspetto è gestito in maniera più corretta rispetto a quanto pensato da noi; dunque, prenderemo spunto da questa idea per migliorare la nostra architettura.