Peer-Review 1: UML

Luca Botti, Alessandro Cogollo, Mattia Colbertaldo
Gruppo AM56

3 aprile 2022

Valutazione del diagramma UML delle classi del gruppo AM29

1 Lati positivi

- La logica di gioco descritta nel class diagram è fedele alla logica di gioco reale. Ciò permette di progettare il software seguendo dinamiche facilmente simulabili; inoltre, la fedeltà al gioco migliora la lettura e scrittura del codice, facilitando la comprensione dell'architettura a più sviluppatori, rispettando buone prassi dell'ingegneria del software.
- La scelta di mantenere quasi tutte le informazioni all'interno della classe GAME semplifica l'impacchettamento delle informazioni, rendendo più agevole l'invio al controller.
- L'uso di interfacce (*es. influenceCalculator*) permette una scelta dinamica delle strategie di gioco, semplici o avanzate.

2 Lati negativi

- Come già evidenziato, la logica del model risulta molto fedele alla logica di gioco reale, presentando sia aspetti positivi che negativi. Le classi STUDENT e PROFESSOR vengono istanziate senza alcun metodo o attributo, riflettendo a pieno la presenza di pedine nella plancia di gioco, ma non rappresentano una soluzione ottimizzata. Una buona soluzione potrebbe essere il modellare gli STUDENT tramite Array di integer, in

cui ogni cella è associata ad un colore. Analoghe considerazioni si possono fare a proposito delle scuole e, ad esempio, della TowerTable, che funge unicamente da "deposito" di Tower.

- Anche la centralizzazione di buona parte della logica del model all'interno di game può essere contestabile, in quanto effettivamente semplifica la gestione dei messaggi verso il controller, ma complica la scrittura della classe. Questo approccio simile a quello procedurale tende a delegare solo poche funzioni alle restanti classi, caratteristica che invece è fondante nella programmazione Object Oriented.

- La lobby essendo una classe deputata all'aggregazione dei giocatori (precedente quindi all'istanziamento del game) andrebbe modellata nel controller, in quanto al model viene affidata unicamente la gestione degli oggetti e delle logiche di gioco.

- Le interfacce fixed e placeable non sono autoesplicative, e la comprensione delle loro funzionalità risulta complessa, oltre che l'ipotesi della loro effettiva implementazione: alcune classi (*es.* student) sono sia *fixed* che *movable*, rendendo poco chiari i meccanismi di associazione della classe con l'interfaccia.

3 Confronto tra le architetture

L'uso di GameInterface è interessante in quanto permette di esporre all'esterno del model solo le funzionalità che possono effettivamente essere usate dal controller; questa scelta è stata valutata anche dal nostro gruppo durante la stesura dell'UML, optando alla fine per l'uso di indicatori di visibilità protected nella dichiarazione dei metodi, ottenendo un risultato analogo. Anche la gestione dei character tramite interfacce è interessante, e da valutare durante la scrittura delle classi ad essi relative.