Peer-Review 1: UML

Paolo Pertino, Leonardo Pesce, Alberto Paddeu
Gruppo GC3

4 aprile 2022

Valutazione del diagramma UML delle classi del gruppo GC13.

1 Lati positivi

Di seguito elenchiamo gli aspetti apprezzati della struttura del modello di gioco.

Influence Calculator Riteniamo vincente l'utilizzo di una tale tipologia di oggetto per gestire il calcolo dell'influenza. Come osservabile dal modello, esso rende trasparente la logica di calcolo e sue possibili modifiche, prevedendo anche estensioni del suo utilizzo in futuro.

Game class Fornire un'interfaccia unificata per l'insieme di quelle dei sotto componenti del modello semplifica ed allo stesso tempo protegge l'accesso di enti terzi al sottosistema model.

Round class Un aspetto interessante di questa classe è la separazione di parte della logica di controllo. Racchiudere quest'ultima all'interno di un oggetto facilita il lavoro al controller in termini di gestione di flussi di operazioni del turno corrente, anche in termini di interazione tra diversi giocatori.

2 Lati negativi

Analogamente elenchiamo qualche passaggio che riteniamo migliorabile.

Gestione Modalità per Esperti Non siamo particolarmente convinti di come venga caratterizzata una partita con regole per esperti, rispetto ad una normale. In particolare l'attributo boolean ***expertMode*** suggerisce la possibile presenza di molteplici if...then...else... cases all'interno dei metodi della classe ***Game***, eventualmente evitabili delegando ad una classe figlia le possibili aggiunte e/o variazioni che la modalità per esperti implica.

Carte personaggio Gli attributi ed i metodi presenti nelle diverse classi delle carte non ci convincono pensando ad una loro possibile attivazione ed applicazione di effetto durante il Round. I metodi presentati dalle classi delle carte non ci sembrano sufficienti per circoscrivere l'implementazione dell'effetto di un personaggio sul gioco unicamente all'interno della sua classe. Questo, a nostro avviso, appesantisce tutti i metodi che regolano le diverse attività di gioco, comporta la necessità di effettuare controlli nelle diverse parti del codice interessate dall'effetto della carta stessa.

Inoltre, ci sentiamo di sconsigliare l'assegnamento del metodo *getMotherNatureAdditionalMoves* alla classe astratta *CharacterCard* in quanto esso verrebbe ereditato da tutte le sottoclassi, tuttavia utilizzato da una soltanto.

3 Confronto tra le architetture

Gestione dei Round La nostra gestione dei round è implementata completamente nel Game controller, senza subordinare tale logica ad una classe specifica, come Round del modello dei nostri colleghi. Analizzeremo la possibilità di aggiungere tale logica di controllo al nostro modello.

Utilizzo di calcolatori Anche la nostra struttura prevede l'utilizzo di specifici calcolatori per, ad esempio, il calcolo dell'influenza o delle condizioni di scambio dei professori. Queste idee ci sembrano vincenti in quanto l'eventuale alterazione del loro comportamento (e.g. da parte di una carta personaggio) è trasparente rispetto al normale flusso di attività del gioco.

Contenitore di Studenti I nostri colleghi gestiscono tutti gli oggetti fisici che possono contenere studenti con una classe apposita *AStudentsContainer* che raggruppa tutti i metodi per gestire il flow di aggiunta/rimozione dal contenitore stesso. Noi, invece, a seconda dell'oggetto contente studenti gestiamo il loro contenimento e i loro spostamenti a seconda della situazione. Questo, per come è stato ideato, permetto di avere metodi riusabili per quel determinato scenario.

Gestione dell'ownership dei professori Il nostro modulo sparge il concetto di ownership dei professori nelle plance dei giocatori, mentre i nostri colleghi gestiscono tale pratica all'interno della classe Game principale. Troviamo utile tale approccio in termini di semplicità nella manipolazione dei professori stessi, in quanto per effettuarne lo scambio e per verificare eventuali condizioni basterà fare accesso ad una struttura dati centralizzata nella classe principale.