Peer-Review 1: UML

Federico Perico, Pier Luigi Porri, Davide Edoardo Pellegrino

Gruppo 56

Valutazione del diagramma UML delle classi del gruppo 01

# Lati positivi

Abbiamo trovato la scelta di dividere il Controller e i relativi metodi dalla classe Game interna al Model appropriata per lo schema scelto: può risultare efficace per dividere i metodi chiamabili direttamente dall’utente da quelli interni al Model.

Dal nostro punto di vista, la gestione della partita con regole Esperto come classe figlia di Game è sulla carta una buona idea per dare ordine e linearità al modello, come quella dell’utilizzo di una classe ad hoc per i team: tuttavia, pensiamo che implementare in codice queste soluzioni potrebbe risultare piuttosto complesso.

Inoltre, è degno di nota il fatto che il modello è predisposto alla gestione della partita a 4 giocatori, in quanto la classe Game è molto completa e fornita per la maggior parte dei casi.

# Lati negativi

Come nella sezione precedente, indicare quali sono secondo voi i lati negativi.

Ci siamo accorti di un errore nella gestione dei turni: nel modello attuale viene salvato l’ordine dei giocatori dopo ogni round in un unico array che viene riordinato ogni volta. Questa soluzione non tiene in considerazione il fatto che tra le due fasi di Pianificazione e Azione l’ordine dei giocatori può essere diverso.

Non è chiaro come vengono gestiti i round e le fasi di gioco (Azione, Pianificazione): non è stata considerata la scelta di creare una classe dedicata a Round, utile a gestire la logica di base del gioco.

Nonostante la divisione tra controller e Game sia efficace, l’utilizzo dei loro metodi potrebbe non essere corretto: i metodi a volte richiedono oggetti come parametri, il che rende complicato lo scambio di informazioni via Internet, e dimostra un’incoerenza nell’uso dei parametri poiché alcuni dei metodi richiedono interi e stringhe, altri direttamente i corrispondenti oggetti.

Infine, nell’implementazione del Pattern Strategy per la gestione delle carte personaggio viene utilizzata un’interfaccia: tuttavia, questa contiene due attributi interi che verranno inevitabilmente modificati durante l’esecuzione, andando contro le caratteristiche di un’interfaccia in Java, che può contenere solo attributi finali.

# Confronto tra le architetture

Dalla nostra analisi del loro modello UML, sembra che i nostri colleghi abbiano dato molta importanza ad un eventuale bisogno di scalabilità futura, modifica che potremmo considerare anche nel nostro modello.

Inoltre, la divisione già citata tra Controller e Game è un’ottima idea, che abbiamo deciso di implementare anche noi.

Alla luce di quanto scritto, dopo un’analisi di confronto tra i modelli, pensiamo che il nostro sia stato meglio costruito con in mente l’implementazione in codice.