# Peer-Review 1: UML

Samuele Scherini, Matteo Spreafico, Ludovica Tassini

Gruppo GC58

Valutazione del diagramma UML delle classi del gruppo GC03.

## Lati positivi

Il diagramma UML è completo e dettagliato, così come il materiale fornito dal gruppo per agevolarne la comprensione. La suddivisione dei metodi fra le classi Game e GameController ci è sembrata ottimale, così come la scelta di delegare le funzionalità avanzate ad una classe che eredita da Game (in particolare, data l'idea iniziale – poi scartata – di rendere Game astratta e di creare tre classi concrete distinte, ciascuna per gestire le variazioni del flusso di gioco in base al numero di giocatori). Riteniamo che il design pattern utilizzato per la gestione delle carte personaggio sia valido, come approfondiremo nella sezione del confronto fra le architetture.
Un altro aspetto positivo è la gestione delle carte assistente, il cui tipo è specificato nella classe Assistant dall'enumerazione AssistantType in cui sono descritte le caratteristiche delle carte stesse.
Infine, la struttura dati scelta per gestire le isole è perfettamente calzante in relazione alle regole del gioco, e facilita notevolmente le operazioni di *merge* delle isole.

## Lati negativi

Il progetto è indubbiamente ottimo. Tuttavia, una scelta effettuata non ci convince: il gruppo ha creato delle classi per alcuni oggetti fisici del gioco che, a nostro giudizio, potevano essere gestiti mediante semplici attributi, al fine di rendere l'UML e l'intero progetto un po' più snello. Dalla documentazione si legge che questa scelta è stata fatta in virtù della futura implementazione della GUI, dato che tutte le classi che rappresentano oggetti fisici implementeranno l'interfaccia GameObject. In realtà, anche noi ci eravamo interrogati circa questa necessità, e dopo averne parlato con i professori siamo stati rassicurati del fatto che questo accorgimento non fosse necessario. In particolare, riteniamo che si possano gestire diversamente Madre Natura, le torri e i professori.

## Confronto tra le architetture

Studiando l'UML del gruppo, abbiamo messo in discussione alcune delle scelte precedentemente fatte. Le differenze principali fra le architetture sono le seguenti:

- Come già accennato, abbiamo apprezzato la gestione del controller. Nel nostro UML, non avevamo rappresentato il controller poiché non era ancora stato ben delineato. Quello proposto dal gruppo è una valida opzione che terremo bene a mente.
- Riguardo alla gestione delle carte personaggio, avevamo adottato un approccio differente. Mentre il gruppo ha optato per un decoratore, noi avevamo deciso di differenziare le carte in base a come influenzano il codice: per quelle carte che triggerano un algoritmo altrimenti non previsto dalle regole base, avevamo pensato ad uno strategy pattern, mentre per quelle carte che modificano degli algoritmi parte del gioco, pensavamo, in GameExpertMode, di fare overriding dei metodi della classe Game se tali carte sono flaggate come attive. Tuttavia, procedendo con l'implementazione del codice, ci siamo resi conto che questa soluzione ci esponeva a diverse problematiche. Abbiamo tratto ispirazione dall'UML del gruppo per ideare un nuovo pattern per il nostro codice.

- Come precedentemente detto, in nome della semplicità, abbiamo tradotto molti degli oggetti “passivi” (cioè quelli che vengono solo spostati) in attributi.
- Abbiamo notato che il gruppo ha raccolto, coerentemente, molte dinamiche di gioco dentro classi, che fungono quindi da collezioni di metodi. Anche se inizialmente avevamo pensato ad un approccio simile unicamente per i metodi specifici della fase di pianificazione e della fase azione, allo stato attuale dello sviluppo, abbiamo ritenuto che non fosse necessario.
- Come evidenziato negli aspetti positivi, riteniamo che l’utilizzo delle *Circular Doubly Linked List* sia la scelta migliore per gestire le isole, e dunque ne trarremo ispirazione. Noi avevamo pensato ad una struttura dati simile, senza però trovarne una che ci convincesse fino in fondo.
- Infine, abbiamo optato per una classe che contenga costanti utili all’intero codice, mentre il gruppo ha preferito dividerle per le varie classi.