Peer-Review 1: UML

Giuseppe Anzillotti, Daniele Asciutti, Samuele Borgonovo

Gruppo AM30

Valutazione del diagramma UML delle classi del gruppo AM03

*Diamo per scontato che abbiate volutamente omesso dal diagramma UML alcuni getter e setter essenziali per semplicità di rappresentazione*

# Lati positivi

* Buona gestione dell’inizializzazione delle partite tramite le classi GameCreator e GameHandler, nonché un’ottima separazione tra fasi di gioco e turni dei giocatori.
* Ottima la scelta di creare una classe per Madre Natura che ha un riferimento all’isola su cui è posizionata.
* Troviamo intelligente gestire fin da subito e in modo chiaro la funzionalità avanzate delle partite a 4 giocatori attraverso una classe dedicata, nonché della classe Team.
* Condividiamo l’idea di implementare le carte Assistente come classe Enum, poiché gli Assistenti contengono solo informazioni di tipo numerico.

# Lati negativi

* Mancanza di una completa gestione delle monete per l’uso dei Personaggi (non abbiamo trovato metodi che facciano aumentare il costo dell’invocazione di un personaggio dopo il primo utilizzo).
* Non abbiamo capito lo scopo delle classi Game2P e Game3P in quanto vuote. Pensiamo anzi che la gestione delle partite a 2 o a 3 giocatori possa essere unita facilmente in una sola classe, in quanto le regole non sono molto diverse.
* Pensiamo che la gestione dei Personaggi e dei loro effetti possa essere semplificata con l’introduzione di una classe astratta contenente un metodo di cui poi le classi degli effetti veri e propri fanno l’Override, senza la necessità dell’introduzione della classe EffectHandler.
* Suggeriamo l’introduzione di un’interfaccia nel Model per comunicare con il Controller che contenga solo i metodi di cui il controller ha effettivamente bisogno, nascondendo gli altri.

# Confronto tra le architetture

Rispetto al nostro design, il vostro gestisce molto meglio la logica di inizializzazione di una partita (essendo Game una sottoclasse e non la classe principale come nel nostro caso). Preferiamo però non utilizzare attributi protected, poiché ci sembra più corretto creare dei metodi specifici o delocalizzare gli attributi nelle classi dove realmente servono e sono usati.

In generale, notiamo che entrambi i modelli hanno un approccio centralizzato, tuttavia ci sembra che il nostro modello risulti più rigido alle modifiche (e forse più robusto), poiché abbiamo creato più classi che contengono gli elementi di gioco alleggerendo la classe Game.