Peer-Review 1: UML

<Aron Contini>, <Alessandro De Florio>, <Entiol Liko>

Gruppo <AM47>

Valutazione del diagramma UML delle classi del gruppo <AM20>.

# Lati positivi

* Uso dello “strategy pattern” per calcolare l’influenza durante il turno (noi, tuttavia, abbiamo deciso di mantenere il nostro modello che usa un’unica classe per calcolare l’influenza)
* Uso di una classe Island Manager per gestire l’unione delle isole, il movimento delle pedine et simila…
* Uso di una classe Professors che permette di modificare e vedere molto velocemente a quale giocatore appartiene un determinato professore; contiamo di implementare una strategia simile nel nostro modello nel prossimo futuro
* Uso di una classe astratta per descrivere le carte personaggio e uso opportuno dell’ereditarietà (noi abbiamo favorito l’uso di un’interfaccia)
* Uso intelligente delle enumerazioni per rendere alcuni attributi espandibili in modo molto semplice

# Lati negativi

* La presenza delle torri sia sulle singole isole che nella classe “PlayerBoard” potrebbe portare a una gestione del movimento delle torri (in caso di sostituzione) relativamente complicata:
  + Una serie di “callback” per incrementare il numero di torri al giocatore che perde l’influenza con conseguente riduzione per il vincitore
  + Operazioni fatte direttamente sull’attributo “owner” della singola isola, la quale tuttavia incrementa l’interconnessione tra le classi (leggi: minore modularità / astrazione)
* Sebbene non necessariamente una nota negativa, scarso information hiding: sono presenti molti metodi get e set per accedere a oggetti che compongono le classi - e.g. il metodo getPlayerBoard() in Player- questo tuttavia sembra giustificata dall’approccio più data-oriented (almeno così ci è sembrato)

# Confronto tra le architetture

|  |  |
| --- | --- |
| Il nostro modello | Modello gruppo 20 |
| Uso di una struttura dati comune (TokenCollection) per gestire le pedine colorate in varie strutture, trasparente tramite opportuno Information Hiding | Ogni classe che contiene delle pedine colorate (Entrance, Island, DiningRoom et simila) ha una propria struttura dati |
| Le carte assistente sono presenti in una classe cardHandler che si occupa solo e soltanto della corretta gestione delle carte in fase di pianificazione | Ogni oggetto player ha la propria lista di carte, il flusso di chiamate sarà gestito da una serie di metodi tra varie classi |
| Uso intensivo di mappe (o meglio TokenCollection) per gestire le pedine colorate, i player e vari riferimenti | Uso intensivo di liste per gli stessi propositi |
| Uso di una macchina a stati per determinare quali comandi possono essere invocati in una determinata fase del turno | Probabilmente il team ha deciso di delegare questa parte ad un livello più alto, eventualmente il futuro “controllore” |
| I dati sono raramente accessibili in modo diretto (e.g. i riferimenti ai giocatori sono delle stringhe ma la classe DashBoard rimane sempre astratta) | I dati sono più liberi di girare ma comunque intelligentemente modificabili mediante metodi opportuni |
| Approccio aritmetico per calcolare l’influenza (esistono dei flag che attivano o disattivano l’influenza di certi aspetti) | Uso di una classe strategy per il calcolo dell’influenza |
| Uso del Command pattern per gestire le scelte dei giocatori | Apposita classe PlayerChoice per il medesimo scopo |