**PEER REVIEW UML GRUPPO GC-45 PROGETTO INGEGNERIA DEL SOFTWARE**

Per revisionare l’UML del gruppo a noi assegnato abbiamo cercato di essere il più esterni possibili ma allo stesso tempo critici facendo dei paragoni anche con il nostro modello, per esempio stiamo valutando l’adozione di alcuni pattern che che sembrano una buona soluzione nel modello del gruppo GC-45. Un pattern di cui stiamo valutando l’uso è il decorator per la classe che svolge il compito di controller.

La prima classe su cui ci sono sorti dei dubbi è Game component, sarebbe infatti meglio se fosse usato come interfaccia per levarie sottoclassi, le quali implementeranno poi i loro metodi nella maniera più corretta. Al di là di questo, bisogna comunque notare che va contro le regole del gioco il poter inserire studenti nella Bag, o rimuoverli dalle classi Island e LaunchRoom. Potrebbe essere uno spunto per future modifiche alle regole del gioco, ma vanno gestite in caso non si possano usare (come, appunto, per le nostre regole).

La classe Game, implementata attraverso un Decorator Pattern, può essere un buon approccio alla gestione della expert mode; tuttavia, implementare sia una classe Game che una classe Controller può essere ridondante, perdendo uno dei vantaggi della programmazione Object Oriented. Di fatto i metodi sono divisi tra le due classi anche se svolgono funzioni molto simili e che potrebbero essere gestite da un’unica classe Game con le annesse sottoclassi per NormalGame e ExpertGame. Il server in questo caso si collegherebbe direttamente a quest’unica nuova classe, generata dall’unione delle due.

Possibile consiglio per le sottoclassi ereditate dal padre astratto GameComponent potrebbe essere ridistribuire i vari metodi, presenti per ora nella classe Game, all’interno delle sopracitate (per esempio calculateInfluence() direttamente in Island, la quale verrà invocata dalla classe Game per le varie fasi del gioco, tra cui appunto il calcolo dell’influenza).

L’utilizzo dello Stategy Pattern per le carte personaggio (classe Character) è una buona soluzione per gestire le varie funzionalità (adottata anche da noi in un modo molto simile), tuttavia bisogna studiare bene il collegamento con i metodi di ExpertGame.

Errore grave nell’implementazione dei metodi all’interno della classe Game, sono tutti quelli che restituiscono direttamente l’oggetto istanziato, permettendo così a classi esterne di vedere il rep (struttura dati interna privata, non modificabile direttamente da una classe esterna), perdendo quindi una delle caratteristiche fondamentali della programmazione ad oggetti. (#informale: siete in buona compagnia, avevamo commesso lo stesso errore… è riparabile facilmente cambiando l’approccio e rivedendo i metodi, i quali possono funzionare in modo simile restituendo una copia del rep per avere informazioni sullo stato. Per modificare poi concretamente il valore degli attributi privati interni alla classe basta creare in quest’ultima dei metodi che, in funzione dei parametri passati vanno a cambiare effettivamente lo stato).

Un esempio è la lista di Player che restituisce direttamente l’oggetto di tipo LaunchHall: sarebbe meglio creare un metodo del tipo showLaunchHallStudents() in LaunchHall che restituisce una copia dello stato interno, e un altro metodo che verrà richiamato dalla classe Game sullo specifico Player che gestisce un’azione, come per esempio il movimento degli studenti da Cloud a Entrance. Tutto questo può successivamente essere usato anche per la view, ottenendo facilmente una copia dello stato interno da leggere e mostrare (serializzandolo per la comunicazione tra server e client).

La gestione delle torri, delle carte e degli studenti è abbastanza confusa, sono tutti mantenuti attraverso contatori e byte, rendendo possibile in caso di errori o funzionamenti errati l’esistenza di studenti in punti diversi senza essere spostati correttamente (cosa gestibile meglio attraversi degli oggetti di tipo Student e Card (rimozione da un lato prima e inserimento dall’altro dopo, como ad esempio lo spostamento degli studenti da Bag a Cloud ecc).

Inoltre, le carte non riportano il valore del movimento di madre natura, e una volta giocate la gestione dei movimenti e delle azioni successive è poco comprensibile (Game come agisce dopo che è stata giocata una carta?). Il mazzo di carte è gestito solo da Player, unificando parti del gioco come la gestione delle carte e la gestione della plancia di gioco che sarebbe meglio a nostro parere tenere separate (anche con classi diverse).

L’idea di base dietro all’uso di byte come strutture dati potrebbe esser stata implementata per facilitare lo scambio di dati con il server; tuttavia, la stessa cosa può essere fatta in modo più pulito attraverso un protocollo di comunicazione ad hoc che si occupa di scambiare lo stato del gioco grazie a delle stringhe, nelle quali vengono codificati dei particolari dati che servono per la gestione del gioco (processo di serializzazione).

#informale - citazione :

