REVIEW GRUPPO ING SOFT

Game component sarebbe meglio implementato se fosse usato come interfaccia, al di là di questo comunque va contro le regole del gioco il fatto che si possano inserire studenti nella Bag o rimuovere dalle classi Island e LaunchRoom. Potrebbe essere uno spunto per future modifiche alle regole del gioco, ma vanno gestite bene in case non si possano usare.

La classe Game implementata attraverso un Decorator Patter può essere un buon approccio alla gestione dell’expert mode; tuttavia, implementare sia una classe Game che una classe Controller può essere ridondante, perdendo uno dei vantaggi della programmazione Object Oriented. Infatti i metodi sono divisi tra le due classi anche se svolgono funzioni molto collegate 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 Game.

Possibile consiglio per le sottoclassi ereditate dall’astratta GameComponent potrebbe essere ridistribuire i metodi inseriti per ora nella classe Game nelle rispettive classi (per esempio calculateInfluence direttamente su Island e che verrà invocata dalla classe Game per le varie fasi del gioco).

L’utilizzo dello Stategy Pattern per le carte personaggio (classe Character) è una buona soluzione per gestire le varie funzionalità, tuttavia bisogna studiare bene il collegamento con i metodi di ExpertGame.

Errore grave dell’implementazione dei metodi all’interno della classe Game sono tutti i metodi che restituiscono direttamente l’oggetto istanziato, permettendo così a classi esterne di restituire degli attributi privati all’esterno, perdendo una delle caratteristiche fondamentali della programmazione ad oggetti (siete in buona compagnia, avevamo commesso lo stesso errore, è riparabile facilmente cambiando semplicemente l’approccio e rivedendo i metodi che possono funzionare in modo simile restituendo copie per vedere lo stato, e richiamando poi un metodo della classe specifica che ne modifica lo stato interno modificando gli attributi privati).

Un esempio è la lista di Player che restituisce direttamente la LaunchHall, sarebbe meglio creare un metodo del tipo showLaunchHallState() 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.

La gestione delle torri, delle carte e degli studenti è abbastanza confusa, sono entrambi gestiti attraverso contatori, 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.

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).

La struttura dati delle carte potrebbe esser stata implementata attraverso byte (con un funzionamento simile a dei flag) 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.