Peer-Review 1: UML

<Gabriele Spagnolo>, <Mario Vallone>, <Lorenzo Zoccatelli>

Gruppo **<GC_11>**

Valutazione del diagramma UML delle classi del gruppo <GC_21>.

Lati positivi

1. L'utilizzo degli Optional nelle CharacterCardInput (classe comune per gestire tutti i parametri delle diverse carte, così da mantenere la stessa interfaccia) permette di evitare errori dovuti al valore "null" e diminuire la verbosità del codice.
2. Abbiamo apprezzato la divisione delle carte tra Stateful e Stateless Effect: anche secondo noi questa è la principale differenza tra le carte personaggio per l'implementazione.
3. Apprezziamo come le possibili mosse che un giocatore può fare sono state implementate come entità derivanti da PlayerAction (classe astratta) permettendo di eseguire le mosse con un generico metodo executeAction.

Lati negativi

Abbiamo notato una gestione un po' troppo frammentata delle azioni di gioco, all'interno di diverse classi sono presenti dei flag per cambiare il comportamento del gioco a seconda delle carte, in questo modo non viene sfruttato a pieno l'object orientation e nel caso una carta personaggio dovesse essere modificata sarebbe necessario cambiare tutte le classi riguardanti quella carta.

1. La gestione delle isole ci sembra confusionaria poiché la classe IslandGroup duplica in parte il concetto di Island, infatti a inizio partita sono presenti 12 Island e 12 IslandGroup, si potrebbe ridurre le Island semplicemente ad IslandGroup. Inoltre abbiamo notato che la classe IslandField contiene allo stesso momento i link a IslandGroup ed alle Island singole, potrebbero essere ridondanti.
2. Passare il riferimento dell'oggetto GameBoard ad altri oggetti potrebbe essere pericoloso. Se si vuole modificare un flag o qualche parametro del model basta creare un'interfaccia che verrà implementata da GameBoard e di conseguenza cambiare la signature nel costruttore della classe del character mettendo l'interfaccia al posto di gameboard: in questo modo l'accesso a ciò che è pubblico è notevolmente limitato. Come GC_21 spiega, questo serve per permettere alle singole entità di modificare flag (che supponiamo essere contenuti in Gameboard), ma oltre a rendere il codice più pesante, avere troppe entità che possono accedere a Gameboard aumenta anche la probabilità di errori (alto grado di accoppiamento).

3. Non è molto chiaro se il pattern Strategy menzionato nella descrizione all'interno del controller si riferisca alle azioni o al controllo della correttezza di queste ultime, nel caso in cui si riferisca all'esecuzione delle azioni, in quanto queste ereditano da playerAction (classe astratta) e implementano diversamente "executeAction", riteniamo che non possa essere definito come Strategy pattern in quanto quest'ultimo dovrebbe eseguire diversamente uno stesso algoritmo in maniera trasparente con la scelta dell'implementazione dell'algoritmo real time mentre l'esecuzione di diverse azioni produce diversi effetti sul model e quindi non rappresentano lo stesso algoritmo.

Confronto tra le architetture

1. Mentre noi abbiamo implementato Studente come un'entità e le Torri come semplici interi, il gruppo GC21 ha fatto esattamente il contrario, implementando gli Studenti come contatori e le Torri come entità. Inizialmente tutti i nostri studenti avevano un ID unico che li identificava tuttavia, dopo aver visto questo UML, abbiamo realizzato che possiamo farne a meno.
2. Abbiamo una gestione concettualmente equivalente dell'accesso alle classi nel gioco: nel nostro UML, è presente un'entità centrale "Game" nel loro "Gameboard" dalla quale è possibile accedere alle altre classi del modello.