Peer-Review 1: UML

<Kevin Wang>, <Pietro Sacchi>, <Maria Francesca Sfondrini>, <Valerio Cipolloni>

Gruppo <AM47>

Valutazione del diagramma UML delle classi del gruppo <AM04>.

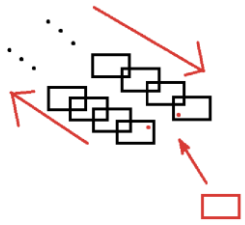
Si segnala la documentazione fornitoci è ampiamente insufficiente (es: cosa vogliono dire i valori speciali definite dentro le enumeration?), non ci si aspettava la sequenza delle chiamate delle funzioni, ma una spiegazione dell'implementazioni di certe idee/logiche in certi punti critici; quindi, alcune supposizioni fatte per scrivere la documentazione sottostante può essere frutto di una semplice incomprensione.

Lati positivi

- Messo la logica dei listener e definito quali possibili eventi/messaggi il client potrebbe scatenare/servire.
- Buon utilizzo delle enumerazioni.

Lati negativi

- Modo di scrivere UML non standard, nella prima parte di una classe dovevano esserci le variabili e nella seconda i metodi (ad es: in GameModelInstance che è diviso in 4 parti, non si capisce cosa si riferisca la quarta parte), inoltre nelle classi riguardanti le carte sono presenti costanti ripetizioni di attributi.
- Per tenere traccia della mappa di ogni player viene utilizzato un grafo, questo renderà difficile poi la verifica della copertura di angoli di carte adiacenti, in quanto ogni volta che si piazza una carta é necessario controllare l'intero grafo per verificare quali sono gli angoli adiacenti a quella carta, cosa estremamente ridondante e complessa.



- Eccessivo disordine nella gestione delle carte, decisamente troppe classi che fanno riferimento allo stesso concetto di carte -> genera troppa confusione, è raccomandabile avere meno classi che sono però essenziali (si consiglia di studiare i pattern in comune che presentano le carte di vario tipo ed estrarre le informazioni in comune).
- Assenza della classe o metodo che implementi l'algoritmo associato ad ogni "achievementCard".
- Nella implementazione della chat, risulta mancante il concetto di "destinatario" del messaggio (ma è presente solo il sender) -> non c'è una logica per distinguere i messaggi pubblici da quelli privati (es: se voglio vedere solo i messaggi sia pubblici che privati di player2, come faccio?).
- Non utilizzare le "abstract class" per classi che non istanzierete (es: Deck), in quanto per quei casi, in base all'UML, istanzierete solo le sottoclassi e non la sopraclasse.
- Eccessivo utilizzo della struttura dati "Map" (perchè mappare gli int per i simboli quando si può utilizzare String e sfruttare i valori già presenti nella enumeration?).

- Eccessivo spreco computazionale per il calcolo delle risorse presenti in un dato momento in una mappa del player, dovrei chiamare ogni volta “getSymbolCount()” in manuscript ogni volta che voglio piazzare una corta obbiettivo (con restrizioni) e per le missioni (con calcolo di risorsa).

Confronto tra le architetture

A valle delle nostre considerazioni fatte, abbiamo notato che l'UML che ci è stato consegnato non presenta vantaggi significativi rispetto al nostro, apprezziamo però l’implementazione del listener e della gestione degli eventi, da cui prenderemo spunto.