Peer-Review 1: UML

<Rosano Francesco>, <Rutigliano Davide>, <Sangalli Matteo>, <Touati Amin>

Gruppo 14

Valutazione del diagramma UML delle classi del gruppo 04.

# Lati positivi

* L’UML del gruppo 04 risulta molto completo e ben organizzato dal punto di vista della suddivisione del model in ulteriori package, che forniscono in maniera modulare funzionalità specifiche relative all’ambiente di gioco.
* Il package Exception risulta ricco, con molte eccezioni ad hoc a verificare che i metodi del model siano chiamati al momento giusto e non avvengano desincronizzazioni tra le varie classi che lo costituiscono.
* La realizzazione dell’UML risulta molto avanzata anche dal punto di vista di come il model si interfaccia con gli altri componenti; seppure non completi, sono già presenti view, controller e l’interfaccia Listeners, che contribuiscono alla definizione e concezione del prodotto finale nella sua interezza.
* La classe DefaultValue è un’ottima idea per quanto riguarda la futura modifica del codice, nel caso si vogliano implementare ulteriori modalità di gioco o variazioni nelle regole.
* Inoltre si sfrutta la Strategy Pattern e la Factory Method in merito alle classi per gli Obiettivi Comuni, il che dimostra dimestichezza nel loro utilizzo e permette maggiore estendibilità e manutenibilità.
* L’mplementazione di un’interfaccia Card permette una visione più astratta del’insieme.

# Lati negativi

* Per quanto riguarda la leggibilità del diagramma in generale, essa risulta difficile data l’estrema suddivisione delle funzionalità in componenti minori; se da una parte è utile avere classi singole ad occuparsi di funzionalità particolari, in alcuni casi abbiamo trovato che si sarebbe potuto ricorrere a uno sviluppo più compatto delle funzionalità; ad esempio, per quanto riguarda la classe points, non si è condivisa la scelta di gestirla come classe a sé stante, dato che non ci sembra necessario tenere conto dei punti per l’intera partita, e nemmeno salvare quale punto sia associato a quale carta. Sarebbe invece stato possibile usare dei valori booleani associati ai giocatori per tenere conto dello stato di svolgimento degli obiettivi comuni durante la partita, e un altro array in ogni obiettivo comune per tenere l’ordine di completamento.
* Seppure sia derivata anche dalla scelta del punto precedente, non si è condivisa la scelta di rendere le classi obiettivo comune e obiettivo personale figlie di un’unica sottoclasse; essendo meccaniche di gioco da gestire in momenti diversi del gioco (le comuni turno per turno necessariamente, le personali non c’è motivo di non verificarle solo in end-game), risulta forzato l’uso dell’interfaccia astratta *Card* per accorpare le funzioni che le coinvolgono.
* Un altro aspetto non risultato corretto riguarda l’utilizzo dell’interfaccia GameListeners:

sebbene dia più interezza nell’insieme, non può essere implementata dall’interfaccia View o da sue istanze concrete.

Essa andrebbe implementata dalle classi del package Controller, il quale può davvero avere “controllo” effettivo sul Model, a differenza della View, come specificato dal design pattern architetturale di tipo MVC.

# Confronto tra le architetture

Ogni funzionalità del gruppo 04 risulta scomposta nell’interazione di tanti piccoli componenti specifici; nonostante in qualche caso, come discusso di sopra, questo approccio non sia stato apprezzato, il nostro progetto pecca a tratti nel lato opposto: nella nostra classe Player, per esempio, si è accorpato anche un attributo matrice library che rappresenta la classe che il gruppo 04 ha chiamato “Shelf”, quando in effetti ha senso che essa diventi un tipo di dato astratto, avente dei metodi che non interessano direttamente il player, secondo l’astrazione tipica della programmazione a oggetti.

Oltre a ciò, come discusso nel primo paragrafo, un grande punto di forza del progetto del gruppo è nell’uso delle eccezioni; nel nostro progetto, infatti, esse non sono risultano sfruttate a pieno per gestire casi limite o errori di tipo out-of-bound (es. dimensione della libreria o della plancia, tipi di tessera disponibile), mentre invece contribuirebbero a una maggiore chiarezza generale del codice e a una più robusta soluzione implementativa.

Inoltre, l’uso della classe DefaultValue è come già riportato ottimo, dato che nella nostra soluzione, qualora presenti dei valori di default (es: libreria 6\*5), essi sono implementati come attributi FINAL nelle diverse classi che li utilizzano, invece che essere raccolti come attributi di un’unica classe pubblica. Quest’ultima soluzione garantisce invece di modificare le dinamiche di gioco molto più semplicemente, tramite la modifica di una sola classe, e risulta quindi ingegneristicamente un approccio più corretto e flessibile.

Infine, nonostante l’utilizzo dell’interfaccia GameListeners risulti non appropriato per il design pattern architetturale scelto, se implementato dalle classi del package Controller risulterebbe più che corretto, o al massimo ridondante.