Peer-Review 1: UML Model

GRUPPO REVISORE
Gruppo: GC 34
Polizzi Chiara, Raimondi Francesco, Rubagotti Andrea, Santagata Maria Concetta (referente)

VALUTAZIONE DEL DIAGRAMMA UML DELLE CLASSI DEL GRUPPO GC 24

Lati positivi

- L'idea di programmare usando il design pattern Strategy è ottima per quanto concerne la leggibilità, ma si potrebbero raggruppare al **massimo** in 2 Strategy (*vd. sotto nei lati negativi*).
- Ok la gestione delle carte mediante la **classe astratta Card** e con le relative sottoclassi. Tuttavia questo causa alcuni problemi legati all'ereditarietà (*vd. sotto nei lati negativi*)
- Buona la presenza dello stato "**ERROR**" in Game, in quanto permette di gestire errori sul Model, eccezioni di disconnessione, ecc.

Lati negativi

PROBLEMI

- Vari problemi sulla gestione dell'ereditarietà:
    - per la classe Card: non sarà possibile chiamare metodi delle sottoclassi, ad esempio, da una carta piazzata sulla table (matrice di Cards). Per farlo, bisognerebbe ricorrere ad un cast, che tuttavia è spesso indice di cattiva progettazione (oppure utilizzare Override!).
    - per la classe Deck: sono liste di Card e non vengono differenziate in base alla loro tipologia (di classi figlie). Se pesco una Card essa sarà **staticamente** una Card e non potrà chiamare i metodi della classe figlia.
- Mancanza di alcune informazioni inerenti alle carte. Ad esempio, non è chiaro come venga verificato se una carta Gold richieda di coprire un numero *"n"* di angoli per attribuire i punti.
- È presente una classe denominata "Object", tuttavia questo è un **errore** in Java, in quanto la classe Object esiste già ed è la classe dalla quale tutte le classi ereditano (https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html).
- La classe Deck avrebbe più senso sotto forma di **Stack** piuttosto che di List. Infatti è sempre necessario pescare la carta in cima ad esso (metodo **pop()**), ed inoltre è possibile mescolarlo con più facilità. Ad esempio, si può utilizzare il metodo "**shuffle()**", richiedente uno Stack.
- La funzione "**drawCard()**", secondo noi, non dovrebbe essere di tipo void, ma dovrebbe essere di tipo Card. Inoltre potrebbe essere la classe Player ad invocare drawCard() richiamando un metodo che restituisca la carta in cima al Deck: ora come ora non viene passato come parametro il Player e risulta più complicato aggiungere la carta pescata direttamente alle carte che il giocatore tiene in mano.

- La matrice di Card disposte sul tavolo rischia di causare uno spreco di spazio. Per ovviare a questo problema, sarebbe possibile rendere dinamica l'allocazione della matrice, la quale dovrebbe avere dimensioni differenti in base al numero di giocatori presenti nella partita (poiché sarebbe diverso il numero massimo di carte giocabili da ogni singolo giocatore).
- Sarebbe possibile evitare di creare classi Resource e Oggetto (potrebbero bastare le enumerazioni).
- Per come è stata pensata la classe Corner, gli angoli *potrebbero* contenere contemporaneamente, ad esempio, sia una Resource che un Oggetto.
- ObjectiveDeck non viene **mai utilizzato**, nonostante sia presente all'interno dell'UML.
- Non è sufficientemente chiara la gestione dei Corner nelle carte iniziali, in quanto ognuna di esse può possedere risorse negli angoli sia su di un lato che sull'altro. Tuttavia dall'UML traspare il fatto che una Starter Card è composta solamente da una lista di 4 angoli. Questo è corretto per le altre tipologie di Card, poiché nelle carte oro e risorsa il retro possiede solamente angoli vuoti.

APPUNTI

- Per quanto riguarda il design pattern Strategy, si potrebbe ridurre drasticamente il numero di strategie, tenendone solamente una per trattare il pattern a "L" e una per trattare il pattern a diagonale. Questo potrebbe essere svolto mediante **l'aggiunta di alcuni attributi** alla classe ObjectiveCard.
- Sempre nella classe Card, l'attributo CardColorEnum ci risulta ridondante. Infatti il colore della carta dipende esclusivamente dalla risorsa presente sul retro della carta (**backResource**) di tipo Gold o Resource (le carte base non hanno un colore, in quanto possono possedere più risorse sul retro). Inoltre, in questa enum, è presente il colore "GOLD", non necessario, in quanto le carte Gold hanno la propria classe goldCard.
- Sicuramente un vantaggio di Java è quello di poter scomporre la programmazione con la suddivisione in varie classi diverse. Tuttavia per alcune cose l'UML ci è apparso **eccessivamente scomposto** (ad esempio: la classe GameInfo potrebbe essere inserita integralmente all'interno di Game, oppure è possibile evitare di creare una List che contenga i vari mazzi presenti sul tavolo di gioco, poiché sarebbe più leggibile gestirli con semplici attributi).
- In Player, non è molto adeguato mettere le carte in mano ad ogni giocatore come List. Piuttosto sarebbe più opportuno allocare un array di 3 posizioni. Infatti, seguendo le regole di gioco, prima avviene l'azione di piazzare una carta, e solo in seguito se ne pesca un'altra. Come conseguenza, nessuno avrà mai contemporaneamente più di 3 carte in mano.
- Gli obiettivi comuni sono 2, non uno solo. Di conseguenza la funzione "getObjectives()" risulta imprecisa, in quanto al posto di ritornare un Objective, dovrebbe ritornarne 2.
- Secondo noi sarebbe utile aggiungere un attributo che gestisce lo stato attuale del giocatore: "**IS_WAITING**", "**IS_PLAYING**".
- Nell'enumerazione che indica lo stato del gioco, è assente uno stato "**ENDED**", il quale sarebbe utile a bloccare le azioni di tutti i giocatori nel momento in cui la partita termina.
- Negli obiettivi è presente una lista di colori per ogni Strategy. Tuttavia in alcuni casi mettere una lista è inutile, in quanto le tre carte potrebbero essere dello stesso colore (per esempio, avviene sempre così nei pattern a diagonale).

- Nei Corner non è presente la possibilità che uno di essi sia assente. Probabilmente questa funzione sarebbe stata implementata con un riferimento a "*NULL*", nel caso in cui l'angolo sulla carta non sia segnato. Tuttavia per noi così è poco leggibile, in quanto "*NULL*" è uno stato eccessivamente vago, mentre l'angolo "ASSENTE" è un caso ben definito.

Confronto tra le architetture e punti di forza

- Nel vostro UML le enumerazioni sono state create sulla base di una divisione che guardava alla tipologia di simbolo (risorse/oggetti), nel nostro invece la divisione guardava alla **posizione del simbolo** (al centro della carta o negli angoli). Nel nostro UML è presente anche il tipo ABSENT per un angolo che non è neanche raffigurato sulla Card.
- Nel vostro UML è presente una divisione tra i mazzi di carte e la classe Game che gestisce la partita. Il risultato di questa divisione porta ad aver bisogno di più passaggi per accedere ai vari componenti del gioco.
- Giustamente il Game presenta la possibilità di avere uno stato di "**ERROR**", cosa che nel nostro UML non era stata pensata. Risulta sicuramente necessaria l'aggiunta di uno stato similare, in quanto permette di identificare il momento in cui un Game è in stato di errore magari dovuto ad un'azione non corretta sul Model .
- Nel vostro UML, la gestione degli angoli viene fatta nella classe Corner, mentre nel nostro UML la gestione degli stessi è inclusa nella classe Card tramite attributi. Con la vostra scelta crediamo che ci possa essere uno spreco di tempo nel momento in cui bisogna cercare un Corner con determinate caratteristiche (si scorrerebbe tutta la lista di Corner per raggiungere quello da controllare).
- Anche noi avevamo inizialmente pensato all'utilizzo del design pattern Strategy: successivamente siamo riusciti ad utilizzare **un singolo algoritmo** per gli obiettivi riguardanti il conteggio di risorse/oggetti ed uno per i pattern (impostabili in base ad alcuni attributi che abbiamo inserito nella classe ObjectiveCard).