# Peer-Review 1: UML

Giacomo Saracino, Zafferani Diego, Zullo Francesco, Alessandro Trimarchi

Gruppo 28

Valutazione del diagramma UML delle classi del gruppo 18

## Lati positivi

- Già presa in considerazione nell'UML la scelta di funzionalità avanzate data la presenza della classe Chat e anche della possibilità di partite multiple con l'attributo di Game gameID.
- Già presente una bozza del controller per gestire l'andamento della partita a livello server
- Definizione completa dei deck di carte giocabili specificate con il numero che le compongono
- Indicazioni di interfacce per alcune classi sono già presenti
- Nella documentazione è presente una spiegazione approfondita su tutti gli attributi e metodi presenti in ogni classe con indicazioni su inizializzazioni e funzionamento
- Presente l'indicazione di Design Pattern per le classi GoldCardStrategy e ObjectiveCardStrategy (vedi nota in confronto)
- Buona la scelta di tenere un deck board per accedere ai deck di gioco collegati a game (vedi nota sul confronto)
- Buona la creazione di una gerarchia per le carte, distinguendo PlayingCard e ObjectiveCard
- Buona la scelta in di usare una mappa in PersonalBoard per la gestione dei contatori delle risorse

## Lati negativi

- Sembrano mancare nella classe Game degli attributi e metodi per la gestione dei turni di gioco.
  Infatti può essere utile/necessario tenere traccia del tipo di azione (giocare una carta, pescare una carta, etc..) che ci si aspetta dal giocatore corrente, ad esempio tramite un enum (anche al fine della gestione degli errori).
  Può essere comodo definire un metodo che viene chiamato ogni qualvolta un giocatore esegue un'azione, che ha lo scopo di aggiornare il tipo di azione e il giocatore corrente per il prossimo turno.
  Abbiamo visto che della logica per la gestione dei turni è dentro alla bozza del Controller (ad esempio il metodo switchTurn), tuttavia pensiamo che debba essere il model ad avere questo tipo di logica. Il controller, per quanto abbiamo capito, è responsabile di chiamare azioni che a loro volta agiscono sul model. Ad esempio un client tramite il controller può chiamare il metodo per pescare una carta, a questo punto è il model che si occupa di verificare se è un'azione valida, dal giocatore di cui è il turno, e dopo aver eseguito l'azione, aggiorna il giocatore corrente e il tipo di azione per il turno successivo.
- Collegandosi al punto precedente, può essere utile definire una classe, o dei metodi, per la gestione degli errori. Noi abbiamo optato per una classe che ha come attributo la lista dei giocatori. Quando un giocatore fa una mossa, controlliamo se è il tipo di mossa attesa e se la mossa è legale, in caso negativo questa classe setta un attributo "error" dentro il player che ha fatto la mossa per indicare che c'è stato un errore e il tipo di quest'ultimo.

- Per quanto riguarda la classe Card, non ci sembra necessario l'attributo owner, infatti il possessore della carta è sempre determinato: se la carta non è dentro una mano del giocatore la carta non è di nessuno, se la carta è nella mano di un giocatore, la carta è di quel giocatore, se la carta è nella PersonalBoard di un giocatore, la carta è di quel giocatore. L'attributo "side" invece pensiamo sia meglio spostarlo dentro Box, infatti il fatto che una carta sia stata giocata sul fronte o sul retro non è una proprietà di una carta, ma piuttosto una proprietà di una carta piazzata su un tavolo, ovvero di Box.
- Come per il punto precedente, gli attributi x e y della PlayingCard sarebbero più sensati dentro a Box, infatti la posizione di piazzamento non è una proprietà dalla PlayingCard in sé.
  Una cosa ancora migliore, per quanto ci riguarda, sarebbe trasformare la Board di tipo Box[][] nel tipo Map<Coordinate, Box> dove Coordinate è una semplice classe immutabile con attributi x e y. In questo modo le informazioni sulle posizioni di piazzamento sarebbero nel posto più consono e facilmente ottenibili dalle chiavi dell'attributo board. Abbiamo notato che questo approccio è molto comodo. Elimina anche la duplicazione sull'informazione della posizione delle carte, che può esserci con l'approccio attuale (ovvero dentro PlayingCard e dentro gli indici di board).
- Può essere utile, dentro a Box, inserire un attributo per salvarsi l'ordine di gioco delle carte. Infatti, per ricostruire il tavolo da gioco, bisogna sapere se una carta è stata giocata prima o dopo di un altra.
- Mancanza di un metodo per gestire nel momento finale del gioco le risoluzioni di eventuali parità.
- Dentro la classe Player può essere necessario distinguere i punti effettuati da carte obiettivo e i punti effettuati in tutti gli altri casi (per determinare il vincitore in caso di pareggio).
- Per quanto riguarda la classe ObjectiveCard, non è chiaro perché faccia override di addPoints dato che è l'attributo strat di tipo ObjectiveCardStrategy, quindi è quest'ultimo a fare override. Inoltre il metodo addPoints di ObjectiveCardStrategy dovrà avere dei parametri, uno che rappresenta la board e l'altro mapResources, di PersonalBoard, per riuscire a calcolare correttamente i punti.
- Sembrano mancare attributi e/o metodi per verificare quando un giocatore arriva a 20 punti o i mazzi finiscono, calcolare i turni rimanenti e innescare la fine della partita.
- Per quanto riguarda la classe Chat, non ci è chiarissimo come venga gestita la chat globale. Potrebbe essere meglio spostare la chat globale in un attributo che è lo stesso per tutti i giocatori. Come è adesso, pare che ogni giocatore abbia una copia della chat globale.
- La classe GoldCard può essere definita come sotto classe di ResourceCard
- Sarebbe meglio spostare l'attributo strat e il metodo getStrat da PlayingCard a GoldCard

## Confronto tra le architetture

- Per quanto riguarda il DeckBoard, noi abbiamo creato degli attributi per il deck di carte iniziali e carte obiettivo. Anche se verrebbero utilizzati solo all'inizio per ottenere le carte necessarie e poi non più utilizzati, può essere utile averli in modo da mescolare i mazzi a inizio partita e randomizzare le carte in maniera equivalente a come accade per gli altri deck. Riteniamo comunque valido l'approccio usato.
- Per quanto riguarda la GoldCardStrategy noi abbiamo fatto una gerarchia con i soli due casi in cui una carta gold abbia anche il calcolo dei punti in base alle carte coperte o risorse sul campo che soddisfano la condizione, che rappresentano qualcosa di più specifico della

semplice carta Gold, quindi magari si può rivedere la gerarchia tuttavia come avete fatto rende il design pattern identificato più lineare.

- Idem per la ObjectiveCardStrategy dove magari si può rendere la gerarchia un po più complessa, distinguendo le carte per risorsa o posizione e successivamente creare una nuova gerarchia.
- Noi abbiamo creato una classe risorsa che si collega all'enum delle risorse, che avete pure voi, in modo che al momento della creazione delle hasmap per le carte possiamo usare la classe risorsa invece dell'enum, la vostra appare una semplificazione che però, se non vi porta a limitazioni, può risultare efficace.