Peer-Review 1: UML

Aryan Sood, Federico Villa, Matteo Vitali, Marco Zanzottera
Gruppo GC19

Valutazione del diagramma UML delle classi del gruppo *GC09*.

1 Lati positivi

La scelta di dividere tra `Resources`, `Object` e lo stato dei `Corner` è ottima. Condensare tutto in una sola `enum`, infatti, vi avrebbe costretto ad introdurre catene di `if-else` e avrebbe impattato la leggibilità.

Nella classe `Player` l'utilizzo di strutture dati dinamiche per memorizzare le carte giocate e presenti sul tavolo è molto utile: riduce lo spazio complessivamente occupato dal gioco e rende più leggera e facile un'eventuale implementazione del salvataggio degli oggetto per svolgere la *funzionalità avanzata* della *Persistenza*. Risulta quindi molto comodo avere `HashMap<Card, Integer[]> map` e `ArrayList<Card> table`.

Nella classe `Game`:

- All'avvio del gioco, il metodo `startMatch()` si occupa di preparare il modello di gioco alla partita;
- Sono presenti nella classe `Game` i metodi per pescare una carta a faccia in su dal campo, oppure a faccia in giù dai mazzi: `drawFaceUpCard()` e `drawFaceDownCard()`;
- È presente un metodo per aggiornare il punteggio con i punti ottenuti da carte obiettivo, `updateFinalPoints()`;

2 Lati negativi

Alcuni aspetti su cui vi invitiamo a ragionare:

1. *Forte dipendenza funzionale tra gli oggetti* `Card` *e lo schema di gioco di un giocatore*

   Memorizzare in `Corner` la carta che lo copre può causare problemi in fase di serializzazione e riduce notevolmente la leggibilità del codice. Una soluzione potrebbe quella di gestire lo schema di gioco del giocatre in una classe separata. Potete eliminare da `Card` anche l'attributo `used`: alla fine l'informazione sulle carte utilizzate è già memorizzata nella `HashMap<Card, Integer[]> map`.

2. *Carte iniziali*

   La `InitialCard`, a nostro parere, dovrebbe estendere `Card` in modo da non dover dichiarare metodi diversi per gestire le carte iniziali e quelle risorsa o oro.

3. *Gestione dei simboli visibili*

   Avete costruito due enumerazioni diverse per gli oggetti e per i simboli; si poteva fare in modo che entrambe estendessero una qualche interfaccia (ad esempio `Symbol`) in modo da avere `HashMap<Symbol, Integer>` (e non `HashMap<Strng, Integer>` per i simboli visibili.

4. *Strategie associate alle carte*

   Avete rappresentato le strategie associate agli `Objective` e e alle `GoldCard` mediante delle stringhe. Ciò potrebbe condurvi ad avere nel controller infiniti *switch case* per capire la condizione associata ad una carta. Potreste, quindi, applicare un pattern *strategy*.

5. *Aggiunta di un attributo per lo stato di un turno di gioco*

   Nella classe `Game` non è presente un attributo che possa tenere traccia dello stato in cui il turno si trova: questo dovrebbe essere rappresentato nel model, e permetterebbe di capire se un'azione compiuta da un giocatore in un dato istate è valida. Esempio: *il giocatore prova a pescare una carta*

mentre è l'inizio del suo turno: come impedisco che lo faccia prima di aver piazzato una carta? Con la classe attuale è solo possibile impedire che un giocatore compia azioni al di fuori del suo turno, confrontando chi compie l'azione con il giocatore attuale ottenuto da `getPlayerCurrentTurn()`;

6. *Aggiunta di un attributo per determinare se la partita è terminata*

   Al termine della partita, viene chiamato nella classe `Game` il metodo `updateFinalPoints()`, che aggiunge i punti delle carte obiettivo al punteggio dei giocatori. A questo si potrebbe aggiungere un attributo allo stato di gioco che permetta di determinare se la partita è terminata o meno, per distinguere l'azione di aumento dei punti da carte risorsa e oro da quello finale delle carte obiettivo, permettendo infine di decretare i vincitori.

7. *Due mani separate per le Carte Iniziali e le carte Oro/Risorsa*

   Vi invitiamo a rivalutare la scelta di avere nella classe `Player` due array per salvare la mano dei giocatori: `hand[]` e `hand1[]`. Inanzitutto è senz'altro necessario, come avete d'altro canto già pensato, gestire la carta iniziale come una carta a tutti gli effetti da avere nella mano, così da poter gestire il piazzamento come una normale carta oro o risorsa. Non capiamo, però, il senso di utilizzare un array di carte iniziali quando la carta iniziale è una sola per ogni singolo giocatore.

   Può avere senso trasformare l'array `hand1[]` in un singolo riferimento ad una carta iniziale, `InitialCard hand1`, però potrebbe risultarvi più difficile gestire i piazzamenti e le varie view, dato che bisogna considerare due tipologie di mani separate.

   Dato che la mano con l'initial card è presente solo all'inizio del gioco, si potrebbe gestire separatamente le due mani attraverso opportune view e una logica del controller più complessa che considera le due mani.

   Avendo noi implementato le carte piazzabili sul tavolo (`Oro`, `Risorse` e `Iniziali`) come derivanti da una classe `PlayableCard`, che viene utilizzata in tutti i metodi per il pescaggio e/o piazzamento delle carte, vi consigliamo di seguire un approccio simile.

3 Confronto tra le architetture

Gestendo le carte presenti sul tavolo con strutture dati dinamiche è possibile avere maggiore efficienza nei client, che devono, quindi, utilizzare una quantità di memoria proporzionale al numero delle carte fin'ora giocate.
Il nostro approccio alla gestione delle carte sul tavolo si basa su strutture dati statiche e, quindi, risulta in maggiore memoria occupata all'inizio del gioco.