Peer-Review 1: UML

Andrea Sanguineti, Emanuele Santoro, Andrea Piras
Gruppo GC-45

4 aprile 2022

Valutazione del diagramma UML delle classi del gruppo GC-55.

1 Lati positivi

C'è una buona distinzione tra le varie classi, soprattutto per le carte speciali, divise a seconda dell'input. E' anche molto comodo tenere in una sola classe *ScoreRuleSet* tutti i possibili effetti delle carte speciali poi usati nell'ActionPhase, specie nel caso in cui si devono aggiungere nuove carte personaggio.
L'idea di creare la classe Team per gestire i player nel caso con 4 giocatori e mantenerla anche quando ci sono solo 2 o 3 giocatori è una buon modo per astrarre il tutto anche nel caso si decidesse in futuro di aggiungere altre modalità con più giocatori.
L'idea di far ereditare i vari componenti del gioco ad una interfaccia che contiene gli studenti può rilevarsi utile però non è sfruttata appieno.

2 Lati negativi

La classe *TurnRuleSet* non interagisce con nessun altro elemento al di fuori di Match e può essere vista solo come una struct. Non serve mai istanziarla più di una volta e non ha molto senso separarla dal Match se non per snellire la suddetta classe.
Le interfacce *ICanCointainPawn, IPhase e INameDescription* vengono utilizzate solo per "forzare" le classi che le implementano a dichiarare i loro metodi ma mai per astrarre effettivamente l'oggetto, di fatto non vi è alcun attributo con tipo statico di una di queste interfacce.

Relativamente a *INameDescription* inoltre riteniamo che nomi e descrizioni degli oggetti che la implementano andrebbero invece memorizzati lato client in quanto non necessari alla logica del gioco
Un nostro consiglio sarebbe di fondere insieme le classi *IslandTile* e *Archipelago* in quanto ridondanti e con la funzione merge, generare una nuova isola contenente tutti gli studenti e le torri di quelle rimosse.
Inoltre ci sembra un po' inefficiente la gestione dello *ScoreRuleSet* che pur permettendo di avere un unico metodo per andare a impostare i singoli parametri, allo stesso tempo non ne verrà mai modificato più di uno solo, perché nel singolo turno è possibile giocare una sola carta. *PreparationPhase* invece potrebbe venir gestita diversamente salvando solo il valore della carta giocata dai singoli player e calcolando solo prima dell'*ActionPhase* l'ordine dei giocatori.

3 Confronto tra le architetture

La nostra architettura si differenzia per il fatto che la logica applicativa è gestita da un'unica classe chiamata *Game* (in cui viene usato un pattern decorator per poi aggiungere le funzionalità aggiuntive dell'expert game).
Le classi *Player*, *Bag* e l'interfaccia per un oggetto che può contenere studenti sono simili a come sono stati implementati nel nostro modello, con la differenza che quest'ultima viene implementata anche da altri componenti che hanno studenti, quali *Bag*, *LunchHall* e *EntranceHall*.
Anche noi avevamo pensato di implementare le classi *Deck* e *AssistantCard* ma poi abbiamo deciso di inglobarle all'interno della classe *Player* all'interno di un array dopo aver notato che i valori sono incrementali e quindi l'unica cosa di interesse al gioco era sapere se quella carta era già stata giocata, cosa sicuramente più limitante per future modifiche alla struttura del gioco.
L'idea di utilizzare la classe *Team*, come descritta sopra, si è rivelata vantaggiosa anche nel nostro modello e abbiamo deciso di prendere spunto da essa. Confrontando i due UML ci siamo resi conto che fa utilizzo di molte più classi rispetto al nostro e potremmo quindi suddividere meglio alcune logiche di gioco, rendendolo più modulare come il vostro.