Peer-Review 1: **UML**

Lorenzo Trombini, Niccolò Benetti, Matteo Scarlino, Salim Salici
Gruppo AM49

Valutazione del diagramma UML delle classi del gruppo **AM06**.

Lati positivi

Un aspetto positivo riscontrato è la scelta di rappresentare l'area giocabile di un giocatore tramite un grafo. Ciò consente un utilizzo di memoria ridotto rispetto ad un approccio a matrice. Bisogna però prestare attenzione all'implementazione dei metodi di ricerca delle carte all'interno del grafo.

La decisione di gestire gli obiettivi come entità separate dalle carte riduce di molto la complessità della classe `Card`, permettendo anche una maggiore chiarezza dell'UML.

Buona l'idea di rappresentare i pattern tramite offset di coordinate x e y.

Ci è piaciuta l'idea dello state pattern perché se implementata bene modellizza accuratamente il funzionamento di un gioco da tavolo.

L'UML è ben suddiviso nelle varie aree rappresentanti i macro aspetti del gioco.

Lati negativi

Sospettiamo che il pattern "State" non sia implementato correttamente poiché nella classe principale `GameState` ci sono alcuni metodi che non vengono utilizzati da tutte le sottoclassi, come per esempio `onDrawFromTop`, che presumibilmente può essere chiamato solo dallo stato `DrawPhase`.

Il metodo `removePlayer` della classe `Round` è rischioso in quanto potrebbe far perdere le informazioni relative al player rimosso. Questa osservazione potrebbe essere valida solo nel caso in cui si voglia implementare la funzionalità aggiuntiva **"Resilienza alle disconnessioni".**

Nella enum `Token` è erroneamente presente il colore `black`, che dovrebbe essere invece sostituito con il colore `green`. Il token nero nel gioco fisico serve solo a identificare quel'è il giocatore iniziale.

Non è chiaro il perchè dell'attributo `tokenColors` di tipo `Stack<Token>` nella classe `Game`, soprattutto se ai giocatori viene data la possibilità di scegliere il proprio token, come suggerito dal metodo `onChooseTokenColor` della classe `GameState`. Da una struttura di tipo stack è infatti possibile estrarre solo il token in cima allo stack, e non un token arbitrario.

Nella enum Collectable l'attributo `plant` si ripete due volte e manca l'attributo `wolf`.

L'attributo `cornersFlipped` è presente sia nella classe `Card` che nella sottoclasse `StarterCard`.

Sospettiamo che l'attributo requests della classe PatternObjective debba essere di tipo `ArrayList<RelativePatternComponent>` e non `ArrayList<Card>`.

Confronto tra le architetture

Una modifica che potremmo apportare è quella di adottare una rappresentazione dei pattern tramite offset di coordinate piuttosto che con il nostro approccio attuale. Questo permetterebbe di poter rappresentare ogni pattern possibile. Inoltre valuteremo se alleggerire le responsabilità della classe `Game`, che al momento gestisce gran parte della logica di gioco.

Valuteremo anche l'utilizzo del pattern "State", che potrebbe essere utile nel caso in cui in futuro si volessero aggiungere ulteriori stati.

Riteniamo valido il nostro approccio di gestire il calcolo dei punti dei diversi tipi di obiettivi e delle carte oro tramite lo "Strategy" pattern, e consigliamo al gruppo revisionato di considerare tale approccio.