Peer-Review 1: UML

Lorenzo Benedetti, Vincenzo De Masi, Nicola Bomben, Pietro Massari

Gruppo AM38

Valutazione del diagramma UML delle classi del gruppo AM01.

Lati positivi

Model

1) L'utilizzo dei file Json ci è sembrata un'ottima idea per inizializzare e gestire le varie componenti come la **Board** e i vari Obiettivi. In questo modo, la distribuzione delle carte e l'inizializzazione della **Board** diventa facilmente modificabile, in quanto si evita di "hardcodare" quali posizioni sono bloccate nella **Board** e quali sono le possibili carte obiettivo tra cui scegliere.
2) L'utilizzo della classe **Cell** ci sembra una buona metodo per tenere traccia di quali posizioni della **Board** sono disponibili e quali sono bloccate.
3) Buona l'idea di introdurre un raking dei giocatori, che sebbene non siano strettamente necessario, troviamo che comunque possa arricchire il gioco.

Lati negativi

Model

1) La classe **Tile** e l'enumeration **Color** ci sembrano ridondanti, in quanto **Tile** ha come unico attributo *color* e unico metodo *getTileColor*, quindi potrebbe essere utilizzata solamente l'enum per indicare le tessere Oggetto.
2) Suggeriamo l'utilizzo di classi manager per alleggerire il carico su **GameModel**. Per esempio noi abbiamo utilizzato una classe per gestire le **CommonGoalCard** con i loro token e una per gestire la **Board** insieme alla **Bag**.

Controller

1) Manca la gestione dei turni, che potrebbe essere gestita direttamente da **GameController** o da una classe ad hoc di cui il **GameController** possiede un riferimento, pensavamo utilizzando il pattern **State**.
2) Sia la classe **GameModel** che **GameController** implementano la stessa interfaccia **CMD**, non pensiamo che sia concettualmente giusto in quanto il Controller ha il ruolo di raccogliere gli input dei Giocatori e successivamente stimolare il Modello attraverso i suoi metodi pubblici.

Confronto tra le architetture

Model

1) Nel nostro progetto abbiamo utilizzato direttamente un array bidimensionale di un enum **Item** per creare la **Board** con un attributo speciale per indicare che una posizione è bloccata. La vostra soluzione è più elegante e valutavamo di modificare anche la nostra implementazione.

2) Come già osservato nel punto 1 dei lati positivi, voi utilizzate i file Json per l'inizializzione delle varie classi, mentre noi per ora non avevamo ancora pensato ad una soluzione precisa.

3) Per quanto riguarda la **CommonGoalCard**, noi abbiamo utilizzato il pattern **Strategy** per separare l'aspetto del **check** del pattern e la gestione dello stack di tokens della carta, mentre voi avete adottato la soluzione con l'ereditarietà. Non abbiamo utilizzato una factory ad hoc per costruire le **CommonGoalCard**, ma abbiamo una classe manager che ne gestisce la creazione.

Per il resto le architetture sono simili.