Peer-Review 1: UML

<Agostino Contemi >, <Federico Consorte>, < Filippo Dodi>, <Jacopo Corsi>

Gruppo <04 >

Valutazione del diagramma UML delle classi del gruppo < 13>.

Lati positivi

Dopo aver esaminato l'UML del gruppo assegnatoci abbiamo riscontrato una serie di lati positivi . In particolare abbiamo apprezzato la differenziazione di *gameStatus* in modo tale che alcuni metodi risultino chiamabili solo in determinate fasi. La disposizione degli elementi del modello in mappe e liste è stata approvata, in quanto risultano più gestibili di implementazioni che sfruttano array di interi. La struttura del campo da gioco è comoda visto anche l'utilizzo di *PointsPattern* per i punteggi dati dalle carte obiettivo. Simulando la partita non sembrano esserci importanti problemi di utilizzo ad eccezione dell'incompletezza di alcune classi e le loro relative interfacce. La nomenclatura di metodi e attributi risulta abbastanza intuitiva tanto che a volte non necessita di ulteriori informazioni al fine di comprendere la funzione da essi svolta.

Lati negativi

Analizzando il modello inviatoci abbiamo notato che si possono fare alcuni accorgimenti. Risulta mancante l'implementazione delle classi *CardResource, CardStarter e CardGold* perciò il modello è incompleto, inoltre le interfacce delle rispettive classi per le carte devono essere implementate e non viceversa. Nella classe *Match* dove è presente *Player* va sostituito con *PlayerIF*. Visto l'utilizzo di varie interfacce riteniamo che potrebbe essere utile creare una *CardSidePlayableIF*, dare dei metodi a *CardSide* e sovrascriverli alle sottoclassi.
Abbiamo notato che nella enum dei colori il black non è necessario in quanto *FirstPlayer* è già il black. La classe *token* è superflua, invece potrebbe esserci direttamente un riferimento al colore all'interno del *Player.*

Confronto tra le architetture

Un elemento di distinzione tra le nostre architetture e dal quale abbiamo preso spunto è l'utilizzo delle coordinate, infatti ci siamo resi conto che nel nostro manoscritto non avevamo la possibilità di distinguere una carta piazzata sotto e una piazzata sopra, problema che abbiamo risolto aggiungendo l'attributo *PlacementTurn* in *CornerCardaFace*.
Per la creazione di tipi diversi di carte o mazzi da gioco essi hanno utilizzato *CardFactory,* design pattern più adeguato tanto che noi abbiamo optato per il metodo CreateCards nell'interfaccia *Deck* che poi viene implementato in modo diverso a seconda della classe che implementa tale interfaccia(override).
Le architetture in se risultano differenti, partendo proprio dalla scelta di inserire la logica di gioco interamente nel *GameModel* lasciando al Controller le funzioni di inoltro dati.