Peer-Review 1: UML

<**Stefania Raibaldi**>, <**Veronica Viceconti**>, <**Yana Siao**>, <**Ishmeet Singh**>

Gruppo <**GC09**>

Valutazione del diagramma UML delle classi del gruppo <**GC56**>.

# Lati positivi

Indicare in questa sezione quali sono secondo voi i lati positivi dell’UML dell’altro gruppo. Se avete qualche difficoltà, provate a simulare il gioco a mano, immaginando quali sono le invocazioni di metodo che avvengono in certe situazioni che vi sembrano importanti (ad esempio, la fusione delle isole oppure il calcolo dell’influenza).

* Gestione dei pattern per le carte obiettivo: il fatto di aver suddiviso le carte obiettivo in pattern va a semplificare a livello di codice e di logica i controlli da fare quando si vanno a calcolare i vari obiettivi.
* Estensione dei diversi tipi di carte e isolamento delle StartCard.

# Lati negativi

Come nella sezione precedente, indicare quali sono secondo voi i lati negativi.

* Manca un riferimento/attributo che va a specificare le risorsa/e permanente/i sia per le StartCard che per le Resource e Gold card.
* Gestione delle backcard tramite un booleano invece di creare la classe backcard.
* GameArea dovrebbe essere all’interno della classe giocatore, essendo che i vari attributi presenti nella classe GameArea sono specifici per ogni giocatore, in quanto ciascuno ha la propria area di gioco.
* Toglierei la classe card perchè alla fine ha soltanto un attributo e metodi di getter, quindi si può direttamente collegare le sottoclassi alla classe deck.
* In StartCard c’è una classe StartObject che però non esiste nell’UML, uguale per StartObjects.
* LastPlay() in Game dovrebbe ritornare qualcosa diverso da void, per indicare qual’è l’ultimo giocatore, ma secondo noi basta anche solo il metodo whichPlayerHasTurn() può essere utilizzato quando si raggiungono i 20 punti e si vuole sapere chi è l’ultimo che ha giocato, perchè i punti si possono calcolare prima di cambiare giocatore.
* Hand[] dovrebbe avere dimensione 3, in quanto la carta obiettivo non fa parte delle carte giocabili, quindi le carte giocabili sono 3, inoltre la carta obiettivo non è in mano al giocatore, ma è da parte.
* Point è una sottoparte di totalPoint, non serve avere due attributi differenti perc.
* Tutte le classi che implementano ConditionPattern hanno colour come attributo, quindi forse al posto dell’interfaccia farei una classe astratta che ha come attributo colors.
* Enumerazione di kingdom
* Attributo che dimostra che la carta piazzata e’ stata contata solo una volta per un obiettivo dato
* Aggiungere i metodi per togliere le carte dal displayed cards

# Confronto tra le architetture

Individuate i punti di forza dell’architettura dell’altro gruppo rispetto alla vostra, e quali sono le modifiche che potete fare alla vostra architettura per migliorarla.

* Creare dei pattern per contenere e facilitare i controlli durante il calcolo dei punteggi per le carte obiettivo