AY 2021/2022

Politecnico di Milano

# Peer-Review 1: UML

Gruppo 41:
Gabriele Giannotto  Maria Pia Marini  Christian Lisi

**Version 1.2**

# Indice

# 1 Introduzione

Valutazione del diagramma UML delle classi del gruppo 40.

# 2 Lati positivi

(I) Riconoscimento delle classi cardine neccessarie al funzionamento del gioco («Game», «Player», «SchoolBoard» ecc.), anche se la «SchoolBoard» potrebbe essere ulteriormente scomposta in «GameBoard» e «PlayerBoard».

Questa ulteriore suddivisione favorirebbe la scomposizione dei compiti e permetterebbe di definire quale sezione di mappa è visualizzabile da tutti i player (la «GameBoard»: dove sono presenti isole, nuvole, sacchetto studenti ecc.) e quale invece è "proprietaria" del giocatore (la «Playerboard» che modellizza la plancia di gioco del giocatore con «Entrance», «DininRoom» ecc.).

(II) Avere un riferimento ad un oggetto player nella classe «Arcipelago».

La nostra soluzione prevedeva solo l'utilizzo del colore della torre per settare l'influenza di un player su un isola. Questa scelta adottata dal gruppo 40, potrebbe essere una soluzione migliore e sarà per noi oggetto di discussione.

# 3 Lati negativi

(I) Nel diagramma UML è presente la classe vuota «SchoolBoard» (senza attributi né metodi) a cui sono collegate «Entrance», «DinigRoom» e «TowerColorGroup».

(II) Sono presenti classi "isolate" (non hanno nessun collegamento ad altre classi).

Ad esempio la classe «Bag» dovrebbe essere collegata alla classe «SchoolBoard», e quest'ultima dovrebbe avere un attributo studentBag che diventerà riferimento all'oggetto Bag.

(III) Potrebbe divetare poco leggibile utilizzare una List<int> per modellizzare i gruppi di studenti.

Sarebbe più consono utilizzare una Map<K,V> con: come KEY il tipo di studente (utilizzando l'enumeratore «Creature»), e come VALUE il numero di studenti di tale tipo.

(IV) Con i metodi riportati nel UML dovrebbe essere possibile simulare una partita. Questo non è ancora possibile.

Nella classe «Player» ad esempio mancano ancora dei metodi che modellizzino alcune azione che un giocatore può effettuare (eg. "moveStudentsToIsland" o "moveStudentToDiningRoom") che saranno poi richiamati dal controller.

(V) Si potrebbe generalizzare ulteriormente la modellizzazione di un arcipelago. Essendo un arcipelago un agglomerato di isole, si potrebbe creare una classe «Isola» con alcuni degli attributi e metodi ora presenti nella classe «Archipelago» e modificare quest'ultima in modo che un arcipelago sia effettivamente una collection di «Isole».

Comprendiamo però la scelta implementativa effettuata dal gruppo 40. Dato che un arcipelago una volta creato non potrà mai essere separato nuovamente, la loro implementazione permettere in caso di aggregazioni di isole, la semplice somma degli studenti di equal tipo.

# 4 Confronto tra le architetture

(I) Come anticipato nella sezione Lati positivi, l'aggiunta di un riferimento all'oggetto Player nella classe «Archipelago» per indicare e aggiornare l'influenza sull'isola ci sembra una buona soluzione. La nostra implementazione prevede solo l' utilizzo del colore della torre, essendo la scelta di quest'ultimo univoca per ogni player.

L' implementazione scelta dal gruppo 40 è stata per noi oggetto di discussione, dato che abbiamo qualche dubbio sul passare l'**intero** oggetto player alla classe «Arcipelago» solo per riferirsi a quale giocatore ha influenza su tale isola, senza mai effettivamente utilizzare quell'oggetto (potrebbe essere considerata come una bad practice). Durante la discussione si è proposto come soluzione alternativa quella di passare l'attributo playerID di «Player».

(II) Abbiamo notato inoltre che si è tentato di modelizzare la funzionalità avanzata delle carte personaggio, ma durante la scelta dell' implementazione più corretta sono sorti dei dubbi al grupo 40. La nostra scelta implementativa è stata l'applicazione del pattern Factory method.

Più nello specifico creeremo un file XML per avere tutte le informazioni sulle carte e i loro effetti. Ogni carta avrà un tag <cardEffect> contentente una stringa univoca che identifica il tipo di effetto della carta. Il factory method in base alla stringa contenuta in <cardEffect> ottenuta facendo parsing del XML creerà la carta corretta.

Si dovranno scrivere quindi 12 classi differenti, una per ogni carta con relativo effetto, dato che questi sono tutti unici ed hanno poche similitudini tra di loro.