Peer-Review 1: UML

Laura Colazzo, Filippo Giovanni Del Nero, Dennis De Maria

Gruppo AM19

Valutazione del diagramma UML delle classi del gruppo AM49

Lati positivi

- **Gestione torri**:
  troviamo che l'idea di introdurre un'entità ad-hoc per la gestione delle torri semplifichi notevolmente il modo di gestire le partite nella modalità a squadre.

- **Gestione assegnamento monete**:
  riteniamo interessante l'idea di far gestire all'entità "table" l'assegnazione delle monete solo se si sta aggiungendo uno studente nella casella che da diritto a ricevere una moneta.

- **Divisione DiningHall e Hall**:
  pensiamo che l'utilizzo di due classi separate per gestire questi due ambienti della plancia agevoli lo spostamento degli studenti da un'ambiente all'altro.

- **Gestione delle Carte Personaggio**:
  l'idea raggruppare le carte a fattor comune in base al loro effetto ha evitato di instanziare 12 classi diverse per modellare i 12 diversi effetti e per questo la riteniamo una valida soluzione.

- **Metodi del controller**:
  abbiamo valutato positivamente, già in questa fase iniziale del progetto, la presenza di metodi nel controller utili allo scambio di informazioni con la View.

Lati negativi

Design generale:

- Si evidenzia la sostanziale mancanza di interfacce. La loro introduzione potrebbe semplificare le relazioni fra le classi e migliorare l'estendibilità del modello.
  Ci sentiamo, ad esempio, di consigliare l'inserimento di una interfaccia per il movimento degli studenti da far implementare a tutte le classi che prevedono questa possibilità.

- Si riscontra una non completa conformità al pattern MVC: il Controller deve essere un mediatore di informazioni fra Model e View. Le informazioni relative alla logica di gioco devono risiedere nel Model, mentre il Controller deve solo avere dei metodi per modificare ed eventualmente consultare lo stato del Model.
  Ad esempio, i riferimenti alle Carte Personaggio potrebbero essere spostati nella GameBoard, la quale a questo punto esporrebbe i metodi necessari al loro utilizzo da parte del Controller.

- Non riteniamo necessario la presenza dell'interfaccia Controller perché i tre controller attualmente disponibili non presentano metodi comuni. PlayerController, inoltre, non ha alcun metodo pubblico o riferimenti ad altre classi del Model quindi, allo stato attuale, non

risulta utilizzabile dal GameController e non può reperire informazioni dal Model.

- Provando a simulare una partita sul diagramma UML, abbiamo notato l'impossibilità di mettere in comunicazione alcune classi, sia per la mancanza di metodi, sia per una strutturale mancanza di riferimenti, tali da non permettere il corretto svolgimento di determinate fasi della partita.
  A titolo d'esempio, riportiamo i seguenti due casi:
  - non risulta possibile, allo stato attuale, estrarre studenti da Bag e posizionarli su una Cloud, poiché quest'ultima non espone un metodo per aggiungere studenti e, allo stesso tempo, non mantiene traccia degli studenti presenti su di essa;
  - le classi PlayerBoard, Hall e DiningHall sono isolate rispetto alle restanti classi che compongono il tabellone di gioco.

  Consigliamo, dunque, di effettuare una simulazione completa di una partita reale percorrendo sul diagramma le varie fasi del gioco, in modo da poter evidenziare le eventuali mancanze di metodi ed elementi comunicativi fra le classi.

Calcolo influenza:

Dal Model fornito non risulta chiaro come possa avvenire il calcolo dell'influenza. Notiamo che ogni Player ha un campo InfluencePoint, con relativi getter e setter, ma non si evince come questa classe possa calcolare l'influenza di ciascun Player su una determinata Isola. Suggeriamo di delegare a GameBoard il compito di computare l'influenza di ciascun Player, poiché al momento è l'unica entità ad avere accesso a tutti i parametri necessari, a meno dei Professori presenti soltanto in Game.

Merge delle isole:

Simulando il processo di unificazione di due isole, ci siamo resi conto del seguente problema: dalla documentazione fornita, risulta che la merge prenda due isole, la prima quella di arrivo di MotherNature, e la seconda, quella adiacente da unire.
Stando all'attuale versione, la seconda isola verrebbe rimossa e il suo contenuto (studenti e torri) verrebbe aggiunto alla prima isola. A questo punto provando ad eseguire la merge tra la 12-esima isola (la prima da unire) e la numero 1 (la seconda da unire), l'operazione risulterebbe nella sopravvivenza della 12 esima isola, comprensiva del contenuto dell'isola numero 1, la quale, invece verrebbe eliminata. Terminato il processo di unificazione madre natura troverebbe come propria currentIsland la numero 12, con un array di isole da undici soli slot.
Consigliamo, pertanto, di rivedere il meccanismo di unificazione isole, ed inoltre ci sentiamo di consigliare l'abbandono degli indici per la gestione delle posizioni sulle isole, per far posto, invece, ad una versione che faccia uso di riferimenti alle singole Isole, così da poter sfruttare un iteratore per gestire la navigazione su di esse.

HashMap:

Per una maggior coerenza all'interno delle classi che ospitano studenti, consigliamo di uniformare la struttura delle mappe che tengono traccia della loro presenza sui diversi oggetti.

Confronto tra le architetture

Fra i punti di forza emersi nel primo paragrafo di questa review, evidenziamo i seguenti aspetti che hanno messo in discussione alcune soluzioni presenti nel nostro modello:

- **Gestione Torri**:
  Nell'approccio da noi utilizzato le torri vengono gestite internamente alla plancia di ciascun giocatore, non avendo previsto, per il momento, l'implementazione delle partite a quattro giocatori. L'idea di delegare ad una entità separata la gestione delle torri e i riferimenti ai giocatori che le posseggono potrebbe tornarci utile in futuro per modificare il nostro modello nel caso in cui decidessimo di implementare le partite a squadre.

- **Divisione DiningHall e Hall**:
  Non avendo inizialmente pensato di poter dividere in due classi separate entrata e sala da pranzo, nel nostro diagramma abbiamo creato un'unica classe plancia, la quale possiede dei metodi per spostare internamente gli studenti da un'ambiente all'altro.
  Tuttavia troviamo che la soluzione di tenere le due classi separate renda più lineare sia la gestione "ordinaria" dello spostamento delle pedine fra entrata e sala, sia l'attuazione dell'effetto di certe carte Personaggio.

- **Gestione assegnamento monete**:
  Allo stato attuale, l'entità che nel nostro UML modella la plancia di gioco è priva di un metodo per l'assegnazione delle monete al giocatore che sta per inserire lo studente nella posizione che da il diritto ad una nuova moneta.
  Prendendo spunto dal modello revisionato, abbiamo deciso di inserire un metodo che, nelle situazioni previste, assegni queste nuove monete ai giocatori.

- **Gestione delle Carte Personaggio:**
  Nella nostra attuale versione, le carte Personaggio sono associate a 12 classi diverse dotate di metodi che modificano il modello nella porzione prevista dal loro effetto.
  L'idea di accorpare le carte in dei sottogruppi, con metodi parametrici comuni, potrebbe tornarci utile per snellire il numero delle nostre classi. Terremo conto di questa valida alternativa durante la fase di implementazione.