Peer-Review 1: UML

Lucrezia Tagliatti, Andrea Scaffidi, Federico Zanon

Gruppo GC47

Valutazione del diagramma UML delle classi del gruppo GC57.

Lati positivi

- Per gestire le Carte Personaggio, il gruppo revisionato ha creato una classe CharacterCardDeck, contenente le otto carte personaggio scelte, che si occupa di estrarre le tre Carte da usare nella partita. Le Carte Personaggio sono state divise in quattro gruppi che si differenziano tra loro a seconda dell'effetto che hanno sul gioco.
  Le carte che fanno parte dello stesso gruppo, che è una sottoclasse di CharacterCard, hanno degli effetti simili. Quindi invece di creare otto sottoclassi distinte, una per ogni personaggio, ne sono state create quattro, una per ogni gruppo.
  Questo garantisce maggiore elasticità al modello, infatti, nel caso in cui si vogliano aggiungere ulteriori Character Cards al gioco con caratteristiche comuni a quelle già implementate, non è necessario creare una nuova classe, ma basta incorporarle in uno dei gruppi già individuati.

- Per gestire la problematica dell'unione delle isole, il gruppo ha pensato di definire un'Island come elemento di una lista Table (presente come attributo di Match, la classe che gestisce tutti i componenti del gioco). Il gruppo revisionato ha spiegato che in questo modo, all'occorrenza, il metodo MergeIsland() si occupa di unificare un gruppo di Island sommandone studenti e torri, eliminando poi uno o più elementi dalla lista, permettendo di considerare un gruppo di isole come un singolo elemento della lista.
  In questo modo, dato che una volta unificate le Island non si possono più dividere e contano come un'unica Island, non serve definire ulteriori classi e/o metodi per trattare le Island in gruppo. Inoltre, è immediato controllare il numero di Island o gruppi di Island in gioco, utile ad esempio quando bisogna capire se il gioco è terminato (quindi quando il numero di gruppi di isole è <= 3).

- Il modello, nella sua compattezza, permette di distinguere le relazioni in modo chiaro.

Lati negativi

- Un potenziale lato negativo potrebbe essere la non differenziazione esplicita tra una partita in modalità normale e una partita in modalità esperto all'interno del modello. Infatti, anche nel caso in cui si scegliesse di giocare ad un gioco normale, le classi conservano metodi e/o attributi utilizzati solo in modalità esperto. Questo potrebbe rendere l'implementazione del codice più difficoltosa, dovendo differenziare due casi all'interno della stessa classe/ metodo, compromettendo anche la leggibilità del modello.

- Un altro lato negativo potrebbe essere la quantità ridotta di classi, in cui figurano però molti attributi. Risulta quindi difficile individuare gli elementi atomici che compongono il

gioco, risultando in un modello poco esplicativo e comprensibile, soprattutto per quanto riguarda la relazione tra i componenti.

- Un altro aspetto problematico, che si riconduce a quanto detto sopra, potrebbe essere individuato nella forse eccessiva compattezza della classe School (probabilmente l'equivalente della School Board), in quanto tutti i componenti di essa (Entrance, Dining Room, Professor Table, etc…) non sono chiaramente individuabili ma "distribuite", sottoforma di attributi, tra la classe Player e School.

- Un piccolo appunto sulla molteplicità delle relazioni: la classe Match è in relazione con 3..12 istanze della classe Island e non 1..*.

Confronto tra le architetture

Uno degli aspetti in cui si differenziano le architetture è appunto il fatto che il nostro gruppo, rispetto a quello revisionato, per trattare le Character Cards, ha deciso di creare 12 classi (nel nostro caso abbiamo deciso di implementarne 12 invece che 8), invece che creare dei gruppi di Character Cards con caratteristiche comuni.
Adottare la strategia del gruppo revisionato potrebbe risultare in una semplificazione notevole del modello, per cui ne valuteremo l'implementazione.