Peer-Review 1: UML

<Marco Conti>, <Davide Corradina>, <Flavio De Lellis>, <Nicola De March>

Gruppo <AM26>

Valutazione del diagramma UML delle classi del gruppo <AM35>.

Lati positivi

La struttura del modello UML proposto è semplice ed efficace, la suddivisione in classi è ben fatta e le enumerazioni sono correttamente utilizzate. Il controller è essenziale e la trasformazione delle stringhe provenienti dalla rete in comandi del controller è funzionale.

La scelta di ridurre il numero delle sottoclassi delle *Common_Goal* (da 12 a 8) è sicuramente un aspetto positivo: il fatto che il codice presenti meno classi, in vista della conclusione del progetto in cui sarà molto ricco, è un aspetto da non sottovalutare così come vedere classi tutte uguali sparse per lo schema dell'UML non è funzionale.

Interessante e utile è l'idea di utilizzare *Bag*, *Common_Deck* e *Personal_Deck*, classi che fungono da "contenitori" per gli *Item* e le *Card*. Aumentano la chiarezza del codice, facilitano il processo di gestione delle estrazioni (rendendolo simile a ciò che accede nel gioco fisico) e avvicinano il modello alla struttura grafica del gioco, che presenta sacchetti e mazzi al suo interno.

Il sistema di calcolo del punteggio è, infine, un aspetto molto apprezzato. Sfrutta i *Point_Token* nonché l'invocazione di una specifica funzione presente sulle *Personal_Goal* suddividendo il carico computazionale su classi differenti.

Lati negativi

Uno dei principali aspetti negativi dell'UML è la sua limitata chiarezza in alcune parti del grafico. Molte classi, quali *GameController*, mancano di riferimenti alle classi utilizzate o di metodi in grado di ritornare i riferimenti necessari. La classe *Game*, inoltre, presenta funzionalità poco chiare e metodi di difficile comprensione: *check_yes()* per esempio non si capisce che funzione abbia. Ovviamente voi, autori del codice, avete chiaro il significato di queste tuttavia si cerca sempre, per quanto possibile, di rendere il codice facile da comprendere anche da persone esterne.
Rimanendo sulla classe *Game*, appare graficamente come unica interfaccia fra model e controller; tuttavia le funzioni e i riferimenti in essa contenuti sembrano insufficienti per questo ruolo. Per esempio, non è chiaro se le invocazioni che interessano la *Board*, passino da *Game*, da *Player* o se agiscano direttamente sulla *Board*.

La classe *Selected_Item* è ridondante e poco utile ai fini del funzionamento del codice. Si potrebbe creare un metodo più semplice per gestire la selezione degli item: per esempio inserire direttamente degli *Item_Tile* o delle coordinate nella lista *Selected_Item*.

Riteniamo inoltre che la struttura dati utilizzata per gestire la *Board* non sia adatta. L'idea di utilizzare una matrice di esattamente 45 elementi per ridurre la complessità spaziale del codice si allontana troppo dalla struttura effettiva della *Board* generando i seguenti problemi:

- codice molto complesso per *fill()* e il *check_SelectedItem()* che riteniamo essere rispettivamente il metodo che "popola" il tavolo di tessere e il metodo che controlla se una tessera può o no essere presa dal giocatore;
- sono necessari algoritmi di conversione input-output per passare dalle coordinate reali a quelle della struttura dati che, inoltre, devono adattarsi in base al numero di giocatori;
- si ha una maggiore complessità temporale, più rilevante del guadagno in termini spaziali.

A nostro parere sarebbe stato conveniente cercare di rispecchiare la "struttura grafica della board", per esempio con una struttura dati astratta (in stile matrici sparse) o con una matrice più grande in cui alcune celle, evidentemente, non vengono utilizzate.

Le 12 sottoclassi utilizzate per rappresentare le diverse *Personal_Goal* possono essere evitate. Queste carte sono facilmente accorpabili: hanno una struttura simile e la procedura di calcolo del punteggio segue metodologie affini (a differenza delle *Common_Goal*). Per questo motivo sarebbe consigliato cercare soluzioni alternative, ovvero sfruttare file esterni o hashmap.

Pensiamo infine, che un lato negativo dell'UML sia la creazione di costruttori che richiedono il passaggio di parametri differenti per le varie *Common_Goal*. Questi, infatti, rendendo impossibile utilizzare la stessa linea di codice per creare diverse carte (in un ipotetico "ciclo for"): per ogni specifica carta è necessario specificare lo specifico costruttore.

Confronto tra le architetture

Salvo specifiche scelte soggettive, la struttura del nostro UML e quella del gruppo AM35 sono molto simili, perciò, non vi sono grandi modifiche in termini architetturali per cui un gruppo potrebbe prendere ispirazione dall'altro. Vi sono invece delle differenti scelte di design, quali:

- nel nostro UML la stringa-messaggio proveniente dal Network viene passata ad un Handler, che, in base alla stringa, invoca una specifica funzione del controller. Nel modello del gruppo AM35 la stringa viene prima convertita in messaggio quindi tradotta in azione.
  Entrambe le soluzioni hanno pregi e difetti: la nostra riduce gli "switch" necessari per le traduzioni (da stringa a messaggio e da messaggio ad azione), quella del gruppo AM35 rispecchia maggiormente l'idea dell'object-orientation;
- nel nostro modello UML c'è una classe dedicata alla conversione della stringa in azione, nel modello del gruppo AM35, invece, se ne occupa una funzione della classe *GameController*: questo comporta una classe di dimensioni maggiori (la classe *GameController* gestisce praticamente l'interezza delle funzioni della sezione controller) ma una classe in meno;
- nel nostro modello facciamo uso di "Optional" al posto di un colore EMPTY per la gestione delle caselle nella libreria non interessate dalle tessere. La nostra scelta comporta l'onere di gestire gli "Optional", tuttavia permette di evitare alcune azioni, come per esempio la necessità di riempire ad EMPTY la libreria.

Un elemento mancante nel nostro UML e che invece è presente nel modello del gruppo AM35 è la funzione dedicata alla scelta dell'ordine di inserimento delle tessere nella libreria nonché un identificativo della partita, utile in ottica multiplayer.