Peer-Review 1: UML

Gabriele Bergamaschi, Niccolò Anesa, Alberto Biffi, N. Berk Bolugiray

Gruppo AM20

Valutazione del diagramma UML delle classi del gruppo AM29.

# Lati positivi

Personal Goal Card:

-L’utilizzo di un array di interi per la gestione dei punteggio rende il calcolo di punteggi, alla fine delle partita, molto veloce.

Common Goal Card:

-Avendo costruito dei metodi che raccolgono alcuni aspetti in comune di diversi metodi aggiunge una flessibilità sulla gestione delle carte “common”.

ObjectCardDeck:

- E’ un oggetto necessario che facilita la gestione delle classi “Board” e “Bookshelf”.

Board:

-Essendo un ArrayList di oggetti, e non una matrice bi-dimensionale, è più adatto come struttura per il “board” di questo gioco.

Player:

-Creare delle variabili per la gestione di punteggi delle carte “common” è molto funzionale.

-La classe ha correttamente il metodo setFirstToComplete per sapere a chi assegnare il punto extra.

# Lati negativi

La nomenclatura dei vari oggetti non rispetta quella del rulebook del gioco, risultando meno rapida la comprensione delle funzionalità nel diagramma (esempio ObjectCard/Tile).

Classe Game:

- Non è ben specificato come il metodo addPlayer utilizza i parametri passati.

Personal Goal:

- La classe PersonalGoalEnum è dichiarata come una Enumeration, mentre pare funzionare come struct/Record, configurabile, per esempio, caricando i vari valori da un file JSON.

- Non è chiaro quale sia l’utilizzo dei vari metodi Goal1, Goal2, ecc., ma parrebbero

Costruttori, in tal caso ne basterebbe uno solo con un parametro GoalID in ingresso.

Board:

- Non è immediata la comprensione dell’utilizzo di un ArrayList per rappresentare la board, dato anche il metodo takeObject che utilizza come parametri presumibilmente le due coordinate di ciascuna delle 3 tile per trovare la loro posizione.

Object Card:

L’enumerazione per objectType non è strettamente necessaria, per “alleggerire” il codice basterebbe un attributo di tipo stringa.

Bookshelf

- L’attributo hasEndCard sarebbe più logicamente corretto inserirlo nella classe Player, sarebbe anche più rapida l’ottenimento di tale attributo da parte della classe Game nel determinare il vincitore.

- Non è chiara l’utilità di inserire un attributo isEmpty

Player

- Secondo il paradigma di object orentation, è sconsigliabile rendere direttamente accessibili gli attributi di una classe, playerState sarebbe più corretto renderlo attributo privato e fornire i relativi metodi getter e setter.

- Salvando in Player le tile prese, si rischia, in caso di mancato controllo anche solo lato client, che la partita si blocchi se un giocatore prende più tile di quelle che può posizionare nella propria Bookshelf.

# Confronto tra le architetture

Il diagramma UML del model implementa la gran parte delle funzionalità di gioco e le rappresenta in modo chiaro e ordinato, nel nostro UML dovremmo sistemare la parte ‘grafica’ rendendo il diagramma più leggibile ed interpretabile.

Ci è sembrato utile l’utilizzo di un tipo EMPTY per la tile, così da gestire meglio sia la livingRoom sia la bookShelf e sarà un aspetto che molto probabilmente implementeremo anche noi nel nostro modello.

Buona l’idea di utilizzare un array di interi per gestire le personalGoalCard in modo da semplificare il metodo per il calcolo dei punti a fine partita.

Nel nostro modello utilizziamo uno switch case, il quale potrebbe essere tranquillamente sostituito con l’utilizzo di questa implementazione.

I punti di forza di questo diagramma sono la chiarezza dell’UML e la buona gestione di tutti gli aspetti del modello.