Peer-Review 1: UML

Pietro Mello Rella, Tommaso Montanari, Riccardo Negri

Gruppo 1

Valutazione del diagramma UML delle classi del gruppo 69

Lati positivi

Numero di monete nel gioco

Visto che nel manuale è specificato, è una buona idea tenere conto del numero totale di monete. Se non ci sono più monete disponibili i giocatori non possono più ottenerle riempiendo la dining room. In questo modo si considera il numero di monete non solo un limite fisico del gioco da tavolo, ma un elemento del gameplay. Un esempio sono le case del Monopoly: sono limitate anche nel videogioco e si può sfruttare questa regola per limitare le possibilità di gioco dell'avversario.

Fase del turno

Troviamo comodo tenere traccia della fase di turno in cui ci si trova, per gestire meglio il flusso del gioco: decidere quali scelte fornire all'utente, cosa mostrare e come agire sullo stato a seconda della fase. Possibili fasi: - Initialization phase - Planning phase - Action phase

Eliminare carte dal mazzo

È utile rimuovere completamente le carte che sono state giocate dal mazzo invece che utilizzare una variabile per tenere conto di quali carte sono state giocate e quali no. In questo modo quando si cicla la lista di carte nel mazzo non c'è bisogno di preoccuparsi del flag booleano. La rimozione non è un problema perché i dati di quella carta non sono più necessari in alcun modo una volta che il turno in cui è stata giocata è passato. Si può conservare la carta corrente (giocata in questo turno) in una variabile apposita per poterla mostrare nella UI.

Lati negativi

Implementazione personaggi

Consigliamo di creare classi differenti per ogni singolo Character che implementano una stessa classe astratta, invece di una sola classe con 12 metodi.

Metodi mancanti

Nella classe Bucket non è presente alcun metodo per estrarre gli studenti. Manca un metodo in Island per calcolare l'influenza. Manca un metodo in PlayerClient per giocare una carta.

Varie

Non si tiene traccia della posizione dei professori, consigliamo quindi di aggiungere una classe professori oppure di aggiungere un attributo professori nella classe Dashboard. Non si tiene traccia delle torri presenti su ogni Dashboard. È un'informazione che comunque si riesce a ricavare sapendo quante torri ha piazzato il giocatore e sottraendo le torri presenti sulle varie isole. Per semplicità consigliamo di aggiungere un attributo che rappresenta il numero di torri nella classe Dashboard. Per la rappresentazione delle nuvole si consiglia di usare una classe visto che la soluzione presente al momento (lista di studenti) può funzionare, ma non è comoda e sicura.

Design UML

Si consiglia di ridisegnare l'UML ponendo maggiore attenzione alla disposizione nello spazio delle classi per favorire la comprensione dello stesso. Nella dettaglio potrebbe essere utile porre la classe ActiveGame in alto seguita dalla classe Game seguita a sua volta dalla classe GamePhase. Potrebbe anche agevolare la leggibilità mettere le diverse classi contenute in gamePhase sullo stesso livello orizzontale e spostare gli enumeration nella parte inferiore dell'UML. Manca la relazione di composizione tra Bucket e Game.

Convenzioni UML

Si sono individuati alcuni errori nelle convenzioni usate nel diagramma UML: le frecce che connetto le Enumeration alle classi che le sfruttano dovrebbero essere nel verso opposto, alcune relazioni di aggregazione sono nel verso sbagliato e la rappresentazione della relazione tra gamePhase e le tre interfacce è sbagliata. Riteniamo inoltre che le relazioni di aggregazione in realtà siano di composizione e che quindi vada usato il rombo pieno invece di quello vuoto. Mancano diverse cardinalità nelle varie relazioni. Si suggerisce di prestare attenzione alla visibilità di metodi e soprattutto attributi delle varie classi, molti attributi pubblici dovrebbero invece essere privati.

Confronto tra le architetture

Gestione delle isole

La prima differenza che abbiamo trovato tra i due diagrammi è la gestione delle isole e in particolare la gestione della loro unione quando nel gioco due isole vengono unite. Abbiamo deciso di raggruppare le singole isole in group (arcipelaghi): all'inizio del gioco c'è una corrispondenza uno ad uno tra le isole e i gruppi (12 gruppi contengono ognuno una singola isola). Andando avanti nel gioco questo approccio permette di limitare la copiatura di dati, quando due isole adiacenti si uniscono, uno dei due gruppi viene eliminato e l'isola contenuta in esso viene spostata nel gruppo a cui si sta unendo. Tramite questo metodo teniamo traccia del colore delle torri a livello di gruppo impedendo quindi che

due isole dello stesso arcipelago abbiamo torri di colore diverso e riusciamo a mantenere il numero di studenti contenuti nelle singole isole, cosa che semplifica la visualizzazione della mappa e rende il gioco più simile alla versione fisica.

Gestione di madre natura

Un'altra differenza che abbiamo notato è nella gestione di madre natura. Noi abbiamo scelto di considerare madre natura come una classe con un attributo che punta all'arcipelago in cui si trova e che viene modificato quando si muove. Tuttavia considerare madre natura come attributo boolenano della classe isola può essere utile per velocizzare il processo di spostamento e potrebbe rendere più semplice l'eliminazione dell'arcipelago su cui è presente madre natura in caso di unione di due gruppi (arcipelaghi).

Gestione delle fasi di gioco

Un ultimo punto di confronto è la gestione delle fasi di gioco: noi non abbiamo separato le classi Game e GamePhase inserendo entrambe dentro a una unica classe Game, tuttavia può avere senso effettuare tale separazione in quanto permette di dividere il model in sé dalla gestione dei singoli turni.