Peer-Review 1: UML

Matteo Matassoni, Rahman Khalaf, Vincenzo Lamonea, Alessandro Lugato Gruppo 69

4 aprile 2022

Valutazione del diagramma UML delle classi del gruppo 68.

# Lati positivi

## Design UML

La divisione tra Model e Controller rende di facile lettura il diagramma. Sugge- riamo di evidenziare ulteriormente la relazione tra Game e GameController per rendere chiara la relazione tra le due parti del diagramma.

## Factory Pattern

Troviamo gradevole l’utilizzo di un metodo factory per gestire la creazione di una partita. L’approccio oﬀre molta ﬂessibilità, tra cui la possibilità, in futuro, di recuperare una versione in cache di una partita dal server nel caso ne esista una.

## Uso delle liste

È apprezzabile l’uso delle liste che suggerisce l’intenzione di un uso eﬃciente della memoria. In particolare, l’uso delle liste per gli l’implementazione degli studenti è signiﬁcativa se si intende allocare la memoria a loro relativa solo alla creazione della partita per poi fare uso di riferimenti per la loro gestione.

# Lati negativi

## Implementazione dei personaggi

La classe Character non è relazionata a nessuna altra classe nel diagramma, per cui non risulta eﬀettivamente implementata. Inoltre è consigliabile adottare una classe intermedia per gestire le istruzioni all’interno di useEﬀect() per evitare di avere un riferimento diretto alla classe Game all’interno di Character.

## Attributi diﬃcili da interpretare

Vogliamo far notare che l’utilizzo di una variabile booleana per l’implementazio- ne del Sack all’interno di Gameboard potrebbe essere una svista. Inoltre, l’attri- buto "attribute8" ed il metodo "operation16()" non hanno un nome abbastanza signiﬁcativo da rendere chiara la loro necessità.

## Singleton Pattern

Non è chiara la scelta di usare una classe Singleton per implementare solo il Sack e non, ad esempio, la classe Gameboard.

## Classi "superﬂue"

Consigliamo di eliminare le seguenti classi con un unico attributo: Tower, Stu- dent, Professor, Card. Infatti queste classi possono essere sostituite da un attri- buto dal nome esplicativo nelle classi appropriate. Noi suggeriamo di sostituirle con: enum TowerType, enum ProfessorColor, enum StudentColor (queste ulti- me due possono anche essere ridotte ad una sola enum Color), int CardImage. Poi non capiamo l’utilità della classe Trashpile, infatti le carte scartate non vengono più utilizzate e quindi non ha senso salvarle.

## Convenzioni UML

Abbiamo individuato molti errori nelle convenzioni usate nel diagramma UML: la maggior parte delle relazioni risulta essere invertita. Ad esempio, secondo il diagramma, la classe Card (che dovrebbe essere speciﬁcata come classe astratta) implementa sia la classe Assistant che Character quando è vero il contrario. Altri esempi sono Game e Gameboard, Dashboard e Row, ecc.

# Confronto tra le architetture

## Uso delle liste circolari

Troviamo intuitivo l’uso delle liste circolari per rappresentare le isole dal mo- mento che la loro percorrenza è continua. Adotteremo anche noi questo tipo di liste per rappresentare le isole nel nostro progetto!

## Gestione carte assistente e stato del giocatore

Ci siamo resi conto che nel nostro diagramma non abbiamo tenuto traccia della diversa natura delle varie carte assistente, limitandoci ad assegnare un ID ad ognuna. Pensiamo di adottare anche noi una convenzione simile per migliorare la comprensione del diagramma e facilitare lo sviluppo del progetto. La classe State per identiﬁcare lo stato di un’utente è una feature che vogliamo implementare anche noi perché riteniamo sia un modo semplice ed eﬃcace per monitorare lo stato di un utente.