Peer-Review 1: UML

<Azzi Sabrina>, <Berto Paolo>, <Caleca Gianvito>
Gruppo <AM44>

4 aprile 2022

Valutazione del diagramma UML delle classi del gruppo <AM17>.

1 Lati positivi

A seguito dell'analisi del class diagram del gruppo AM17 sono stati individuati diversi lati positivi.
In primo luogo, si evidenzia il diffuso utilizzo di design pattern presentati a lezione e non solo, infatti sono state usate diverse Strategy ed il pattern Facade. L'applicazione di queste soluzioni progettuali conferiscono al modello robustezza ed efficienza.
In particolare, l'utilizzo dell'interfaccia "Place" risulta essere un buon modo per accomunare diverse classi, quali Island, Cloud, Bag, DiningRoom ed Entrance, che presentano comportamenti simili tra loro. Infatti queste classi presentano una collezione di Student ed i metodi giveStudent(..) e receiveStudent(..) per trasferirli da un Place all'altro.
Un' altra soluzione interessante è quella di MovingCharacter, specializzazione della classe Character, che accomuna tutte le carte personaggio. MovingCharacter utilizza l'attributo booleano "swap" per segnalare la necessità di scambiare oggetti Student con quelli posseduti dalla carta stessa. Inoltre, destination e source esplicitano i Place tra cui effettuare lo scambio.
In secondo luogo, si osserva l'utilizzo di nomi efficaci per identificare le carte personaggio, infatti nell'enumeration CardName questi vengono chiamati in base all'effetto posseduto.
Infine, la presenza della classe ProfessorRoom rappresenta una buona astrazione per la gestione dei professori. Questi vengono istanziati nella Professor-

Room di Gameboard all'inizio del gioco, e successivamente vengono spostati tra le ProfessorRoom dei Player durante la partita.

2 Lati negativi

Class Diagram:

- Nella classe principale Game, che si interfaccia con il controller, sono presenti solo 3 metodi, di cui uno è il costruttore, mentre sono omessi i metodi necessari per interfacciarsi con le altre classi del modello.
- Spesso sono presenti riferimenti doppi tra le classi, anche nell'ambito di relazioni di composizione, questo può causare un eccessivo accoppiamento delle classi e una conseguente maggiore complessità di gestione del modello. In particolare si fa riferimento ad Assistant e Player, Game e GameBoard, Game e Round.
- In alcune classi sono presenti riferimenti ad oggetti di altre classi non segnalati nel modo consono, ad esempio PlayerActionPhase con Assistant.

Gestione dei Character:

- Il pattern strategy viene utilizzato per gestire le modifiche al modello apportate dall'effetto di alcuni Character, per alcuni di questi metodi potrebbe non essere necessario. Ad esempio, MNBonus e ProfessorWithDraw alterano il comportamento dei metodi standard in maniera minima. In particolare, ProfessorWithDraw aggiunge un nuovo metodo solo per la condizione di uguaglianza durante la fase di assegnazione dei Professori, non presente nel metodo principale.
- Per quanto riguarda le carte che necessitano di informazioni dall'utente, l'enumeration CardName lascia intendere che venga usato un valore fissato, anzichè richiederlo al giocatore. Tra queste carte sono presenti Postino Magico e Giullare.

3 Confronto tra le architetture

Nel confronto tra l'architettura del gruppo revisionato e la nostra, sono emersi diversi aspetti in comune. Innanzitutto, è presente una classe Bag, analoga alla nostra StudentBucket, che astrae la funzione del sacchetto del gioco. La classe Bag istanzia tutti i 130 Student all'inizio della partita e successivamente questi vengono spostati all'interno del modello grazie ai metodi offerti dall'interfaccia Place, mentre StudentBucket genera gli studenti nel corso della partita. Le due soluzioni trovate, risultano essere equivalenti.
In entrambe le architetture è presente l'intenzione di astrarre il comportamento di alcune classi del modello che si concretizza con l'interfaccia Place e la classe Character nel loro diagramma e con la classe astratta StudentContainer e l'interfaccia Character nel nostro. L'utilizzo delle interfacce consente di raggiungere un livello di astrazione superiore rispetto alle classi astratte, e di conseguenza di restare il più generici possibile, ma le classi astratte danno modo di fornire attributi comuni e di implementare quei metodi che non necessitano di essere implementati nuovamente dalle sottoclassi.
In entrambi i gruppi è stata presa la decisione di gestire alcuni elementi del gioco, quali torri e monete tramite contatori piuttosto che classi.

Alcuni aspetti, invece risultano concettualmente diversi. In primo luogo, nel loro diagramma la gestione delle fasi dei turni è affidata direttamente al Model, invece che al Controller. In questo modo viene messo in evidenza quali metodi appartengono alla PlanningPhase o alla ActionPhase, definendo in maniera completa da cosa è composto un Round. Questa soluzione rappresenta un punto di forza in quanto alleggerisce il carico di lavoro del Controller, che nel nostro caso gestisce interamente le fasi del gioco.
La presenza della classe ProfessorRoom, modella un aspetto del gioco che noi abbiamo incorporato all'interno della nostra classe Player. Alle istanze di questa classe hanno accesso le diverse classi che secondo le regole del gioco prevedono di entrare in possesso dei professori (anche GameBoard nel caso in cui questi non fossero ancora stati assegnati). Questo garantisce una maggiore fedeltà alla struttura fisica del gioco, e una maggiore astrazione.
Infine l'utilizzo del pattern Strategy per gestire le diverse implementazioni del metodo computeInfluence(), anzichè la segnalazione tramite apposite variabili, risulta essere una soluzione che garantisce estendibilità al codice. Possibilità alla quale avevamo pensato anche noi inizialmente.