# Peer-Review 1: UML

Favento, Laini, Macaluso
Gruppo 32

4 aprile 2022

Valutazione del diagramma UML delle classi del gruppo 31.

# 1 Lati positivi

## 1.1 Overview

In generale il Model ci sembra completo e in grado di eseguire tutte le funzionalità principali del gioco, infatti è possibile realizzare una simulazione della partita attraverso l'invocazione dei metodi. Abbiamo inoltre apprezzato l'ordine e la pulizia dell'UML, che presenta le cardinalità corrette nelle relazioni tra le varie classi.

## 1.2 Assistant

Abbiamo apprezzato il modo in cui viene gestita la classe `Assistant` tramite Enum, che consente un codice più ordinato, visto che è sufficiente inizializzare le 10 carte all'inizio della partita e poi non c'è bisogno di modificare i loro attributi. Prenderemo spunto da questa implementazione per migliorare la nostra gestione degli assistenti.

# 2 Lati negativi

## 2.1 Professori

La gestione dei professori tramite ArrayList di `PawnType` in ogni `SchoolBoard` non è vantaggiosa e può potenzialmente portare alla presenza di più di 5 pro-

fessori contemporaneamente. Crediamo che sia più appropriata la creazione di 5 oggetti di tipo Professore, che poi vengano assegnati nel corso della partita alla `SchoolBoard` del singolo `Player`.

### 2.2 Altri dettagli

All'interno di `StudentsBag`, il metodo `fillWith` prevede il passaggio come parametro di un altro oggetto di tipo `StudentList`, mentre sarebbe possibile riempire la StudentsBag semplicemente richiamando i metodi della sua `StudentList`. Il problema dell'approccio attuale è la richiesta del parametro in ingresso, che necessita la creazione di un'ulteriore `StudentList`.

Alcuni metodi in `Player` ci sembrano ripetizioni non necessarie di metodi presenti in `SchoolBoard`, ad esempio il metodo `addStudentToDiningRoom` o `getNumStudentOf`, nonostante la funzionalità e la visibilità di entrambi i metodi sia pubblica. Queste ripetizioni rendono la classe `Player` troppo grande e con un numero molto elevato di metodi.

## 3 Confronto tra le architetture

### 3.1 Gestione studenti

Il metodo di gestione degli studenti, confrontato al nostro UML, è diverso: noi abbiamo optato per rappresentare gli studenti appartenenti a una classe attraverso liste di oggetti `Student`, mentre qui viene istanziata la classe `StudentList`, composta da 5 attributi di tipo intero che rappresentano il numero di studenti di ogni colore, in ogni classe che deve contenere studenti.

### 3.2 Calcolo influenza

Il calcolo dell'influenza viene gestito in modo differente rispetto alla nostra implementazione. Qui troviamo il metodo `conquerIsland` nel `GameModel` che permette a un giocatore di conquistare un'isola. La nostra gestione invece prevede un metodo che calcola l'influenza del singolo giocatore direttamente nella classe isola.