Peer-Review 1: UML

Daniele Gazzola, Edoardo Fullin, Giacomo Groppi

Gruppo 39

Valutazione del diagramma UML delle classi del gruppo 38.

Lati positivi

- la enum PlayerState sembra essere molto comoda per la gestione temporale del turno.
- la funzione applyEffect nella CharacterCard è un'idea che se implementata a dovere può risultare molto utile per aumentare la flessibilità dell' architettura
- implementare il game come un singleton può semplificare l' architettura nel caso in cui non ci siano partite in parallelo.
- classi non monolitiche e struttura flessibile

Lati negativi

Game:

- playersNumber è ridondante (players.length())

GameBoard:

- GameBoard non dovrebbe essere singleton (è già attributo di un singleton)
- NumIsland ridondante (Islands.length()).
- joinIsland e chooseCharacter non prendono parametri per decidere che oggetti modificare.
- in professorController il parametro numberOfStudents può essere calcolato nel metodo.
- calculateNewPositionMotherNature dovrebbe ricevere un int come parametro (numero mosse)
- non sembra esserci nessun modo per unire le isole.

CharacterCard:

- increasePrice potrebbe restituire void
- initializeCard deve essere implementata da tutte le classi figlie
- applyEffect può agire solo sul player sebbene necessiterebbe di agire anche sul resto del game

Island:

- Towers viene dichiarato come un array di lunghezza 1
- getPlayerTower dovrebbe restituire un TowerColor rappresentativo del player che controlla l'isola

Student:

- getColor dovrebbe ritornare un StudentColor

Bag:

- BAG potrebbe non essere singleton (è già attributo singolo di un attributo singolo di un singleton) quindi è già garantito unico
- getStudents e moveStudent assolvono alla stessa funzione logica: rimuovere uno studente dalla bag implica per forza spostarlo da qualche parte.

Player:

- un player dovrebbe avere una plancia (la sua?) ma non una lista di plance (quelle degli altri?)
- moveMotherNature non serve nel player in quanto non dipende da che player sta giocando
- la funzione svolta da conquerIsland non dovrebbe logicamente essere svolta dal player, forse sarebbe meglio implementarla nella gameBoard.
- ChooseAssistant dovrebbe avere come parametro l' assistente scelto.

AssistantCard:

- manca un attributo used nell'oggetto (o similare)
- la funziona isUsed non ha bisogno di parametri

AssistantCardsDeck:

- chooseNumberOfMoves non dovrebbe essere qua e non ha riferimenti a nulla come anche play card.

Entrance:

- le funzioni MoveStudentToStudentRow e moeStudentsToIsland dovrebbero ricevere anche uno studente ed essere in un punto che abbia accesso alla origin e alla destination

StudentRow:

- giveCoin dovrebbe avere un riferimento al player

- rowColor è ridondate, è già contenuto nel color degli studenti nell' array students

Tower:

- MoveTower dovrebbe essere in un punto che abbia accesso sia alla origin che alla destination (in modo da rimuovere l'oggetto dal punto di origine e inserirlo nella destinazione)

Confronto tra le architetture

In linea di massima i due diagrammi si assomigliano, tuttavia potrebbe essere una buona scelta implementare l' enum PlayerState per tenere traccia dello stato del player. Nella plank/canteen abbiamo usato due implementazioni diverse per gestire gli studenti, nel nostro caso le "studentRow" possono essere calcolate a Runtime direttamente nella view tenendo conto del colore degli studenti. Infine, per quanto riguarda la gestione dei Characters la abbiamo gestita in un modo simile (con funzione applyEffect ereditata) però la funzione ha bisogno di un riferimento a tutto il game in quanto potrebbe modificare lo stato di oggetti a cui il player non ha riferimenti; per gli effetti non istantanei un'idea potrebbe essere di tenersi una lista nella board (oppure nel game) che tiene le carte con gli effetti attualmente attivi. Durante la gestione di quella parte di parte di gioco si verifica quali effetti sono attivi ed (eventualmente) si applicano.