Peer-Review 1: UML

Kodai Takigawa, Antonio Lanconelli, Emanuele Paesano

Gruppo 54

Valutazione del diagramma UML delle classi del gruppo 53.

Lati positivi

- Comprehensive and detailed model
- Clear distinction of each class' role inside the game
  - Every role in the game is shown as a class
- Use of strategy design pattern
- Use of abbreviation
  - TOWER
  - RACE

Lati negativi

- Overall
  - Too many classes.
  - The directions of composition arrows are opposite.
- How does this work? Why do you need it?
  - AssistantDeck
    - assistantDiscardPile: Stack<Assistant>
  - Archipelago
    - +getIslandsByIndex(index: Integer): ArrayList<Island>
    - +indexMap: Map<integer, integer>
  - Bag
    - reinsert()
    - By the rules, the only way to reinsert students in the bag is through the effect of a character card, and is based on color, but your method takes a list of students as a parameter.
- Unnecessary class/method/attribute
  - School
    - Subtowers
    - +AddStudentsToHall and +moveStudentToTable, what is the difference?
  - Student
    - +getRace(): RACE
    - +getColour(): String
    - Race and Color are always one-to-one relationships.
    - Same holds for the RACE enum
  - AssistantDeck can be integrated into the Player.
- Merging 3 islands in a single turn is not managed well.
  - Class: Archipelago
  - Method: mergeIslands(index1: int, index2: int): void
  - Using a list of Island is better? (mergeIslands(List<Island>))
- We don't think towers are managed properly:

- Can you make an array of enums? We thought an enum has only a specified number
of instances
  - Would be best to only store a int in the School and TOWER as attribute in the
Player
- Suggestion
  - Bag can be in GameState class

Confronto tra le architetture

- Add effect cards
- Use of Map data structure
  - We should use Map for Student. It's better than List.
    - Map<Color, NumberOfStudents>
- Use of design patterns besides MVC
  - strategy design pattern
- Use of abbreviation
  - TOWER
  - RACE