Peer-Review 1: UML

Barabino Francesco, Bertolini Gianluca, Borrelli Elisa

Gruppo AM53/ Group AM53

Valutazione del diagramma UML delle classi del gruppo AM26.
Evaluation of the UML diagram of the classes of the group AM26.

Lati positivi/Positive sides:

We really liked how you merge islands, by only calling acquireIsland on the Island class.
The GameLobby itself is interesting, it makes clear the idea behind it, for the arrival of the players in the match.
The implementation of Independent/Team MatchManager handles well the two types of separate matches (very different from our interpretation but appreciable).

Lati negativi/Negative sides:

The management of many things is way too complicated, sometimes not necessary.
There are methods with too many parameters. (Even during classes it was told to the whole class that a method shouldn't have more than 4/5 parameters (e.g. in Character Class: StudentHostCard, GenericModifier...)).

→ The UML Class Diagram is kind of hard to read itself.
Some methods are unclear (like "setup" methods, in CharacterCard, StudentHostCard, GenericModifier, ActivateStopCard...).
Other examples about using the same method in different classes: useCard might be better if it was located just in one (CharacterCard).

For example, some critical issues are noted:

Character Side:
CharacterCardExtractor:
→ Why use a whole class just to extract 3 random cards, when the game could do the same?
CharacterCard:
→ You can use a determined Character card only once in a turn, it's not useful to track the timesUsedInTurn.
→ It is not so good to create 12 instances of CharacterCard just to extract 3 of them...

StudentHostCard:
→ Can't see where you host the Students. (It's unclear what getHostedStudents method should do).

Class: <<Enum>> StudentMovementSource/Destination:
→ Nothing wrong in the syntax… but this kind of movement management seems too complicated
(CharacterCardParamSet: no visible association with other classes?)

(Class: <<Enum>> StopCardMovementMode:
→ Nothing wrong in the syntax… but how could this be useful for the No Entry tiles, since the only thing they do is to NOT calculate the influence of where the Mother Nature Pawn lands?)

Map Side:
Class: <<Enum>> AssistantCard
→It can be instanciated as a single assistant, with proper priorityNumber and motherNatureSteps, but as an enum it contains every assistant type, how should it be used?
Also the differences between assistants consist just in 2 values, why making an assistants enum?
→AvailableCardsDeck, Table manager, Cloud have no attributes.
→Why using CardDeckBuilder to build AvailableCardsDeck?

Cloud and island both inherit from StudentHost, which provides mergeWith method which accept a StudentHost as parameter, but a cloud can't be merged with an island, and a cloud can't be merged with a cloud.

→StudentCollection has methods that work with Student enum but it stores just a list of integers.

Game Side:
Player:
→First, the Player seems ill-defined, due to the lack of other useful attributes besides the only two defined.
Perhaps incomplete of what it needs?
Also, looking at the methods, it seems to handle much more game than necessary (e.g. the SchoolBoard management would be to be merged elsewhere, as one imagines it logically/conceptually).
Then the 3 tower methods could be a repetition of code, maybe unnecessary (gainTower and pickAndRemoveTower).
This class results very massive, too much?

MatchManager:
→Since the idea of division between Planning Phase and Action Phase is visible, the two parts could have been divided into separate classes, to lighten this class from the high amount of methods (which follow step by step the development of the game).

Confronto tra le architetture/Architecture comparison:

At first impact, the two architectures are conceptually very distant, both for the immoderate use of classes and enums (compared to our vision), and for the level of partitioning of activities in the various methods.

Some points of inspiration, specific, have resulted in the management: of mother nature by "steps", in the division for a greater granularity of the methods for the two phases (Planning and Action), of the enum also through methods (we had not considered them) and eventually we will consider to better define the method to declare the victory.
Also, the playersSortedByCurrentTurnOrder, in the MatchManager class, has served as a cue, to improve our implementation.