Design Description for Milestone 3

Author: Maximus Curkovic, Tony Massaad, Kareem El-Hajjar, Cory Helm

Due to the view handling too much logic that the model should perform, there was a lot of integration between the view and models. Therefore, the view now handles only user input and displays, while the model handles all the functionality respectively, and what to be displayed to the view.

The purpose of this new design is to make the project as cohesive as possible. Hence, a lot of operations and methods were broken down into more methods and interfaces. For interfaces, they would show high similarities between classes, in this case Property, RailRoad and Utility as they are all buyable locations.

For implementing AI, we separated the Player class to be a super abstract class for AI and User. The reason behind this is to allow for open code. Additionally, despite AI and User performing the same functionalities all together, we kept them separate as AI functionality landing on a Location is different than the User. Hence, we check the instance of who the Player is, and play the game accordingly.

For the buying of houses, it was implemented so that the user can only purchase houses that they can afford and for properties that can still get houses. This made it very easy to avoid heavy condition checking as it prevents the Player to break the game.

For the selling of houses, the user can only sell houses according to the number of houses they own, and the cost of each house they profit from is the same cost they bought them for.

For Player information, the players have a choice to view or not view each player information. This was done through a separate class which handles a lot of string manipulation using constants. However, it made it quite nice to display information. Clicking the button when on “+” displays the information, while “-” removes the information. It allows the user to choose what they want to seem which is nice for a game like Monopoly.

The game side panel and the game board are operated in different classes than the BoardView. The BoardView calls methods from those classes to operate

accordingly. But for clean and cohesion code, they are separate as they handle a lot of functionality to display the necessary components. This ensured that the BoardView doesn't do too much, but separates it's view into different classes that handles what needs to be displayed.

Lastly, even though Junit testing was not a requirement, it was done anyways for necessity to ensure methods were working as required. Like Milestone 2, simple getters/setters were not tested as they were straight forward. However, components that handled practical functionality were tested (not including location functionality as that handles user input and such).