Plan of Attack

Schedule:

November 25th: Planning/Scheduling + Detailed UML design (4h)

November 26th: Finish Detailed UML design + Answer the Question (4h)

November 27th: TextDisplay + All the Square subclasses + Very Basic Player and Board (4h)

November 28th: Loading/Saving Games + Roll + Next + Rent Calculation (8h)

November 29th: Buying + Bankruptcy + Mortgage + SLC/RollUpRim (8h)

November 30th: Improvement + Trading + End Game + Auction (8h)

December 1st: Testing Mode +Tims Line + Documentation(4h)

December 2nd: Bonus/Debugging/Extra Time for Unfinished Stuff (4h)

December 3rd: Bonus/Debugging/Extra Time for Unfinished Stuff (4h)

December 4th: Bonus/Debugging/Extra Time for Unfinished Stuff (4h)

Question Response:

Question: After reading this subsection, would the Observer Pattern be a good pattern to use when implementing a gameboard? Why or why not?

Response: Observer Pattern would be a good idea for implementing a gameboard, since the entire game uses a MVC framework. Thus, the Controller is naturally an observer for the Model (or the Board class in our design)

Question. Suppose that we wanted to model SLC and Needles Hall more closely to Chance and Community Chest cards. Is there a suitable design pattern you could use? How would you use it?

Response: Decorator Pattern would be a suitable choice. On top of the basic features of SLC and Needles Hall, we can add a wide variety of separate features that resemble the Chance and Community Chest cards.

Question. What could you do to ensure there are never more than 4 Roll Up the Rim cups?

Response: We can have a variable in the Board class keeping track of the number of available Roll Up the Rim cups, and ask the function that is giving out Roll Up the Rim cups to check the number of available cups before giving one out.

Question. Is the Decorator Pattern a good pattern to use when implementing Improvements? Why or why not?

Response: It is not necessary to use the Decorator Pattern for implementing Improvements. There is not an obvious function that takes purchase cost and improvement cost as parameters, and returns the tuitions to charge for each improvement, so the decorator would have to be different for each building. Instead, we can use variable and constants attributes for the squares to provide us with this information. Since every improvement cost the same for a certain building, we can simply use a variable to keep track of the number of improvement that is built and an array of constants to tell us the tuition to charge with a certain number of improvements built.