Plan of Attack Watopoly

<u>Class Descriptions</u>

Controller

<u>Controller</u>

For MVC architecture, this is the controller. As such it is the class that contains the main function and handles the input from the user. The input is then relayed to the Board class to the be interpreted.

View

<u>BoardDisplay</u>

The BoardDisplay is the class that outputs the state of the Board to the user. To begin this class will output text to the console, and later may be implemented to use graphics.

Model

<u>Board</u>

This is the central class of the model, containing the list of players currently in the game, as well as each of the individual cells that compose the Watopoly board.

<u>Cell</u>

Each possible space on the board is represented by a Cell, with its different subclasses have different actions to be executed by the polymorphic action() method.

<u>Property</u>

Subclass of Cell that represents Cell that can be bought by a Player

<u>ImprovableProperty</u>

Subclass of Property representing properties that can be improved with houses and hotels

<u>Gym, Residences, SLC, Needles, TimsLine, GoToTims, GooseNesting, CoopFee, Tuition</u>

All are subclasses of Cell that have the actions as defined in the Watopoly specifications

Design Patterns

For our implementation of Watopoly, we are planning on using the observer design pattern in a similar way to how it was used in A4Q3. Since any buyable property's rent can change depending on who owns the other properties in its set, we will use the observer pattern to notify any of the other properties in a set when a property has changed owners, so each property knows how to calculate its rent properly. This means that the property class will be acting as both an observer and a subject in our implementation. The observer is also used for the Board class to observe the state of the players that it holds.

Schedule of Progress:

Completed by August 5th:

- Work on the UML Chart (Alex and James)
- Work on the Initial Plan of Attack (Alex and Jay)
- Work on answer the dedicated questions (Alex, James, and Jay)
- Create a git server that is accessible by all members of the group. All code will be stored on the git server. All members will pull and push to the server to ensure that the code is up to date (James)

Expected by August 7th:

- Will initially divide up the three majors classes of the project (Player, Cell, and Board), all of which can be tested and implemented independently. Members will have documented code that it is followable and coherent such that other members are able to understand, regardless of authorship. **Note:** Integrating the classes and working on specific will be done later (Alex, James, and Jay)
- Player Class (Alex)
- Cell Class and its subclasses (James)
- Board Class and Board Display Class (Jay)

Expected by August 8th:

- Members will have thoroughly reviewed each other's code for possible logic errors. This will give a chance for all members to be familiar with all of the code written so far (Alex, James, and Jay)
- Will have integrated the Player Class, Cell Class and it subclasses, and Board Class and Board Display Class, ensuring it is functional (Alex, James, and Jay)

Expected by August 9th:

- Will have implemented the controller class, which will first focus on player movements. Player position will be updated accordingly to the dice roll and will be shown on the display (Jay)
- Will have incorporated a saving and loading mechanism such that Player positions will be stored and loaded (Jay)

Expected by August 10th:

- Will have implemented the act of Players buying properties and paying rent to other Players (James)
- Will have updated the saving and loading mechanism to include saving player's money and properties owned (James)

Expected by August 11th:

- Will have implemented rules for bankruptcy such that a Player can now declare bankruptcy (Alex)
- WIll have updated the saving and loading mechanism to include data about which player is still in the game (Alex)

Expected by August 12th:

- Will have implemented the SLC and Needles Hall Cards such that it interacts with the Players (Jay)
- Will have implemented the act of Players buying property improvements, ensuring that a Player must own a full group beforehand (James)
- WIll have updated the saving and loading mechanism to include data about Players's property improvements (James)

Expected by August 13th:

- Will have tested the game thoroughly such that all functions implemented work properly (Alex)
- Will have ensured that the saving and loading mechanism works for all aspects of the game that need to be saved. There should be no difference in the game between when the game is saved and loaded (Alex)

Expected by August 14th:

- Will have implemented the act of trading properties and cash between players (Jay)
- WIll have implemented the act of winning (Jay)

Expected by August 15th:

- Will have thoroughly tested the project completely and will have solved most bugs and logic errors (Alex, James, and Jay)
- Will have thoroughly documented the code (Alex, James, and Jay)
- Will have submitted the executable, the final design document, and the updated UML diagram (Alex, James, and Jay)
- Will have submitted the demo files for the project (Alex, James, and Jay)
- Will have submitted the final design document (Alex, James, and Jay)
- Will have answered any questions that remain to be answered (Alex, James, and Jay)

The above steps are an approximation of the work needed to be done by such date. However, nothing is set in stone and we may differ in our approach. The same can be said about who is doing the work.

The following steps are tasks that are only to be completed if we have extra time.

1. Work on adding the House Rules to the game. Can be done by adding a decorator design pattern
2. Work on implementing graphics instead of a console display. Since we adhere to MVC principles this should only require us to re-implement the BoardDisplay class.
3. Work on implementing Computer Players to play as opponents.

Questions:

Q1: 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?

A1: We think that if we were to model SLC and Needles Hall more closely to Chance and Community Chest cards, we would use the Template Method design pattern. This is because any two cards that could be drawn would be quite similar in what they required to be operational, most likely just a text field as well as a different action for each. Hence each individual card would just override the action function in it’s implementation

Q2: Is the Decorator Pattern a good pattern to use when implementing Improvements? Why or why not?

A2: While it is possible to use the decorator pattern to implement the improvements, it is not the most efficient way to do so, and it thus not a good pattern for this task. Since the cost at each improvement level scales non-uniformly, this makes each respective level dependent on the order, which is not suitable to the decorator pattern. While the pattern might be able to be fitted to this problem, it is much simpler to count the number of improvements, and calculate the rent from there.

Q3: After reading this subsection, would the Observer Pattern be a good pattern to use when implementing a gameboard? Why or why not?

A3: We think that the Observer Pattern would be a good pattern to use when implementing a gameboard because it allows for the board to be easily updated whenever something about the display needs to change. For example, if the number of improvements on a property increases or decreases, this is something that the display would need to know about right after it happens to keep it up to date with what’s happening with the game.