Ayaan Nadamal (anadamal)
Shihan Sharar (ssharar)
Sheehan Shams (s2shams)

CS246 - F24 - Project - **RAIInet**

Date	**Plan**	**Assigned to**
Nov 15	- Initial Meetup - Decide which project - Initial Draft of UML and plan	ALL
Nov 16-17	- Discuss what OOP principles to follow and implement - Determine the design patterns needed	ALL
Nov 18	- Updated UML and plan drafts - Created document to determine test scenarios and scope of tests	ALL
Nov 19	- Implement main.cc - Created .h and .cc base files. - Complete board, cell, link, and ability classes	Ayaan: ability & base files Sheehan: board Shihan: main, cell & link
Nov 20	- Finalized UML and plan documents - Update Game.cc to execute abilities - Complete textObserver	Ayaan: text observer & game Sheehan: game Shihan: game
Nov 21	- Complete Game and Player - Start GraphicsObserver implementation	Ayaan: graphics observer Sheehan: Game Shihan: Player
Nov 22	- Submit DD1 documents with final touches - Check in with code progress - Break, no coding	ALL
Nov 23-24	- Complete graphical observer - Ensure MVC best practices	ALL
Nov 25	- First round of tests - Add additional enhancements such as sound effects, win/lose graphics, two player displays	Ayaan: win/lose graphics Sheehan: multiplayer displays Shihan: sound effects
Nov 26	- Round 2 of Testing - Final Design Document review - Debug code - Practice demo	ALL
Nov 27-28	- Code correction - Review functionality - Update GitHub for final	ALL
Nov 29	- Submit files to Marmoset	ALL

Ayaan Nadamal (anadamal)
Shihan Sharar (ssharar)
Sheehan Shams (s2shams)

Question 1: In this project, we ask you to implement a single display that maintains the same point of view as turns switch between player 1 and player 2. How would you change your code to instead have two displays, one of which is from player 1's point of view, and the other of which is player 2's point of view?

Basic Idea: In order to implement two displays, we would have each observer be dedicated to a single player. Hence, for a two player game, we would have two text observers and two graphical observers (if graphics is enabled) defined in main, one for each player. We would then attach all these observers to the concrete subject game.

We would follow these steps as follows:

- Update the concrete observers Text and Graphics to have a member field that tracks which player they are observing
- Within each observer, update the code to flip the board (if necessary) and consider changes such as the links that are visible according to the player being observed (with a simple if condition checking which player is being observed by that observer).
- Create dedicated observers for each player in main and attach them to the game subject as such :
  TextObserver t1{game, 1} for player 1 and TextObserver t2{game, 2) for player 2
- Since the game has these observers attached, every time the state changes, each observer's notify method will be called.

Question 2: How can you design a general framework that makes adding abilities easy? To demonstrate your thought process for this question, you are required to add in three new abilities to your final game. One of these may be similar to one of the already-present abilities, and the other two should introduce some novel, challenging mechanic. You are required to actually implement these.

To design a general framework that makes adding abilities easy in our game, we centralized the ability logic within the Model component of our Model-View-Controller (MVC) architecture (with few checks within the Controller to understand the number and type of input we should take) . We implemented an overridden function in the Game class that handles all ability-related commands. When an ability is invoked, this generic function calls the corresponding ability handler function implemented within the Game class. Each specific ability function within the Game class encapsulates the logic required to apply its effects on the game state, such as updating the board, etc. By centralizing state changes within the Game class, we ensure that all modifications to the game state are managed in one place.

To add a new ability, we only need to add a new condition to the generic useAbility function to handle the new ability command and implement the specific ability logic in the Game class (along with just including another ability type within the AbilityType enum). Hence the changes to code are extremely minimal. In essence, we only need to add another function within Game that handles that specific ability, since adding an ability type and another if case within the generic function is trivial.

Ayaan Nadamal (anadamal)
Shihan Sharar (ssharar)
Sheehan Shams (s2shams)

Question 3: One could conceivably extend the game of RAIInet to be a four-player game by making the board a "plus" (+) shape (formed by the union of two 10x8 rectangles) and allowing links to escape off the edge belonging to the opponent directly adjacent to them. Upon being eliminated by downloading four viruses, all links and firewall controlled by that player would be removed, and their server ports would become normal squares. What changes could you make in your code to handle this change to the game? Would it be possible to have this mode in addition to the two-player mode with minimal additional work?

To make the code simple in terms of minimal editing when changing from 2-player to 4-player, we will be editing the cell class to have a boolean field locked specifically to check if we are in bounds, since it is not as trivial as a square. We are planning to use a 10 by 10 square, however each of the corner cells would have the boolean field locked as true, thus conforming to the new plus shape of the board.

Moreover, we would also have a method to validate if any of the players are moving towards an edge. In principle, we would extend our functionality from checking only the row position to download a specific link in the case of 2 players, to also checking the column position; the rest would follow the basic game semantics, and thus would not need much change.

Finally, moving onto the case when the player loses; in this case, our Player class has a shared pointer to each of the locations for each of the Link's positions, and we would change each of the symbol's to a dot in our Cell class. Similarly, each of the Ability objects that the eliminated player owns also has the position as a field, and we do the exact same replacement with the dot. Finally, our Cell class also has the field to check if the cell is a server port, so we would make these false once the player is eliminated.

In summary, we would go about in these steps to generalize our 2-player case to a 4-player case:

- Instead of a fixed size of players, we have a vector of players so that each player can be initialized from the controller with ease using a for loop
- Include a locked feature in Cell class to check if we are at corner of 10 by 10
- Utilize a method that does the same checks for rows, but also for the columns, since there will be an addition of player 3 and player 4
- Generalize text/graphical observer to print in the point of view for player 3 and player by 4, by retrieving row and column cells as the transpose of the board
- Retrieve location of eliminated Player's Link objects and Ability objects and replace the char of the corresponding Cell class with a dot
- Change the boolean field for the server port eliminated in Player's Cell class to false