Name: Yuhan Wu

ID: 21064228

Project: RAIInet

Introduction

This is a document explaining my design of the game RAIInet.

• Overview (describe the overall structure of your project)

My project is designed around three core classes: Link, Player, and Board, which make up of the main parts of the game, including the basic game rules. As for the ownership aspect of these three core classes, a Player class owns an array of Link pointers (8), those Links are initialized by player class as well. The memory of the Link objects is managed by using smart pointers in the player class. While Board doesn't own Player and is not responsible for managing Player's memory, it uses a vector to store all the player pointers. Player class manage its own memory through its destructor.

An observer pattern to board is implemented such that Board is a concrete subject and can notify the observers to update the text and graphical presentation of the game.

For the ability perspective of the game. Class hierarchy patter is used here. An abstract base class Ability is designed, and it has eight subclasses each for a special ability the player can use.

There are some other components like Cell and Position are designed for the track of the grid on the Board.

• Design (describe the specific techniques you used to solve the various design challenges in the project)

By following the rule of "Low coupling and High Cohesion", I designed my program with clear structures.

As for the core classes, Link, Player, and board, they each have clear responsibility as the name suggests. Player owns Links and have to take care of its memory; Board doesn't own players but keeps track of different players in the game. The Board is responsible for tracking the game states. Those classes' responsibilities are designed by my own logic when player a board game. These three classes have high cohesion and together make up of the backbone of the game with the basic functionalities.

In order to achieve the text and graphical presentations of the game I decided to use the observer pattern. This achieves loose coupling between observers and the subject, and also provides automatic notification of changes, which really comes handy.

For the extra feature like abilities, I used class hierarchy to implement the common fields, features and methods of different abilities in the Ability base class and implement subclasses accordingly for individual features. This way I avoided repeated codes and made it easy to add or change any specific ability objects without affecting other parts.

• Resilience to Change (describe how your design supports the possibility of various changes to the program specification)

- For the change of game display

I am using the observer patterns to present the display. I have implemented two print methods in the player class, display_player_full_info () and display_player_info () which prints player information in the player's own view and in the opponent's view. In my text observer, I have two Player * fields to specify the player and opponent, and in the notify () method, it calls display_player_full_info () on the player, prints out the board, and calls display_player_info () on the opponent. In this way each player his own information but can only see partial information of his opponent.

- For the change of abilities

The class hierarchy for Abilities and its subclasses makes it easy to add new abilities and to change existing abilities. When a specific ability class need to be changed, modification is only needed for that class, not anywhere else, because each subclass is overriding the base classes activate () method.

- For the change of number of players

Since my Board class keeps track of all the players on the board, we can just append new players to the end of the vector field. The game logic will stay the same, we need to add new printing methods for players that are on left and right side of the board, and make

changes to functions like ‘move’, and maybe add extra parameters to some methods to track the other two players. The board’s dimension can also be easily updated through initialization with new dimension. However, the overall game logic won’t be changed, and we don’t need to start from scratch to implement those new players since they will just be another two instances of the player class.

• Answers to Questions (the ones in your project specification)

Question1: 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?

Solution:

I am using the observer pattern display the board.

For my graphical and text observers, there will be a field indicating which player’s view is being presented. There are also two prints methods specified in my player class, that is each player can either prints out the full information or just partial information revealed to the opponent. In the notify method, the full information printing method will be called on the current player and the partial information printing method will be called on the opponent, so presentations will change depending on the specific player.

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.

Solution:

I plan to use the base and subclass class pattern to achieve this. I will design the abstract base class as class Ability, with virtual methods like activate () which applies the effect of this ability. I will design each of the specific abilities to be a subclass of the class Ability, and they will override the public methods to correctly apply their own effect to the game.

In this way, if I want to add more abilities, I only need to declare it as a super class of the abstract base class Ability and override those performance methods.

New abilities added
6. Brutal:

This ability can be used on user's own links to make sure the link wins any battle.
7. Protected:

This ability can be use on user's own links to protect it from being downloaded directly by the opponent calling download ability on it. Note this doesn't change the link's behavior in a battle.
8. Trade:

Player can use this ability by calling "ability 8 [link name]", it can only be called on player's own links and this will trade players link with the opponent's corresponding link (a-A, B-b, etc.). A link that is already downloaded cannot be traded! (applies both to the player and the opponent, that is, if the corresponding link belonging to the opponent is downloaded, this ability fails as well). After the trading, the link name doesn't change, but the value and the type will change to the opponent's link, same applies to the opponent's link.

• Extra Credit Features (what you did, why they were challenging, how you solved them—if necessary)

- I have implemented the feature of allowing the user to enable ability for the game or not at the very beginning of the game. This is achieved by adding an extra Boolean variable indicating enable or not, and if no, then the initialization for the abilities will be skipped and each player will have 0 abilities to use. Otherwise, go through the ability initialization process.
- Use smart pointers to manage memory. This is achieved by using vector, map methods and smart pointers throughout my code.

• Final Questions (the last two questions in this document).

1. What lessons did this project teach you about developing software in teams? If you worked alone, what lessons did you learn about writing large programs?

I did this project alone, and I learned to consistently work on things that I didn't understand too well and brainstorming and troubleshooting along the way. It's important to stay calm and careful when working alone since there're no teammates' support. Also, my initial UML diagram and design weren't as good as the ones I have now, it's important to make little progress consistently as no one can success in just one step.

2. What would you have done differently if you had the chance to start over?

I would brainstorm a more efficient design, maybe shifting more responsibilities away from the board and implement a separate class called Game to take care of the game logic and let Board just simply acts as a Board with information on the board being updates as the game precede. This might achieve lower coupling and higher cohesion.

• Conclusion

This is a fun game to implement and through designing, planning, and implementing the game, I learned a lot and improved my coding skills so such!