Yifei Yu 945753

COMP90015 Assignment 2 Report

Introduction

The goal of this assignment is to create a distributed shared whiteboard that allows more than one user to edit one whiteboard at the same time. The whiteboard is to be controlled by one manager, its role is to start or stop a whiteboard and be able to kick existing users. Everyone in the system will have access to the same whiteboard functionality, this includes drawing lines, shapes and write text. The system must be able to handle concurrency issues as well as dealing with networked communication.

What I Have Implemented

Basic features implemented:

- Multiple users can draw on a shared whiteboard
- A single whiteboard is shared between all users
- The whiteboard can draw shapes such as line, circle, oval and rectangle
- User can append text onto the whiteboard
- User can choose different colours to draw different shapes
- User can clear the whiteboard
- When the manager closes the whiteboard, the existing users will get a message notification and their whiteboard will be closed
- When the manager leaves the application, the existing users will get a message notification and their application will be closed

Advanced features implemented:

- A chat window that allows users to communicate among themselves
- Manager can save the whiteboard and load the saved whiteboard
- Manager can kick out a certain user
- User can draw freely using a ‘pen’
- New users need manager approval before they can join the whiteboard

System Design

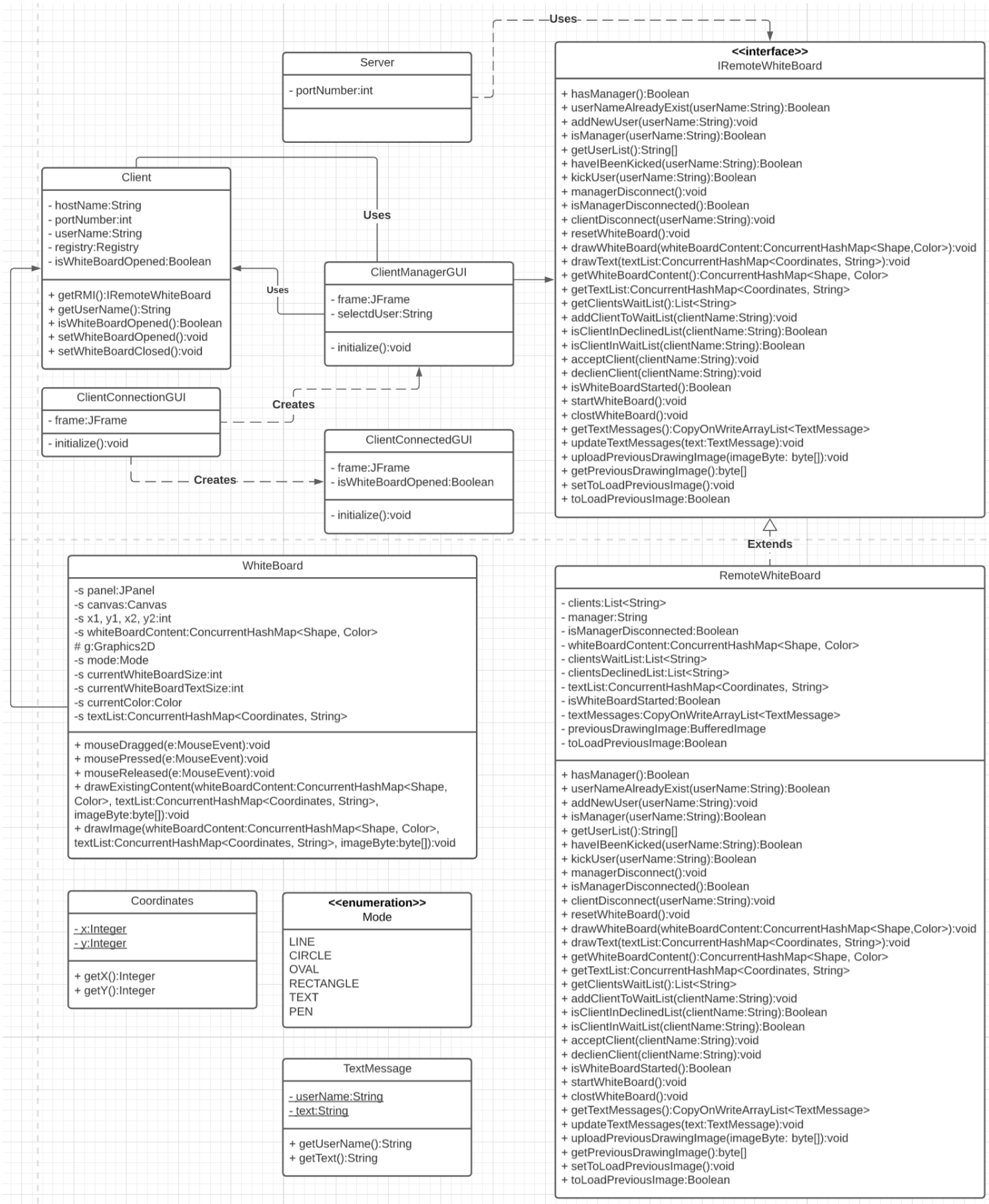
The system can be divided into two parts, which are the client and the server-side.

For the Server side, it acts as a whiteboard server and handles clients' requests. It is mainly consisting of two classes, they are the 'Server' class and the 'RemoteWhiteBoard' class that extends the 'IRemoteWhiteBoard' interface. The 'Server' class is used to set up a port for the RMI registry and binds an instance of the 'RemoteWhiteBoard' class to that RMI registry. The 'RemoteWhiteBoard' class is the remote object implementation. It stores information about the current session, such as how many users are connected as well as the whiteboard content. Clients will be able to call all the methods in this class remotely.

For the client-side, it is used to create and monitor the client GUI and communicate with the server based on the user's input. I have implemented four different GUIs, which are 'ClientConnectionGUI', 'ClientManagerGUI', 'ClientConnectedGUI' and 'WhiteBoard'. The 'ClientConnectionGUI' class creates a GUI that asks the user to enter the hostname and the port number of the server, as well as the username. The 'ClientManagerGUI' class creates a GUI that is to be used by the whiteboard manager, it includes a current user list, a kick button, a button to create a whiteboard and a chat window. The 'ClientConnectedGUI' class creates a GUI that is to be used by the normal user, it is mostly the same as the 'ClientManagerGUI', the only two differences are it does not have the kick user button and create a whiteboard button has changed to join the whiteboard. Lastly, the 'WhiteBoard' class creates a whiteboard, both the manager and the user use the same 'WhiteBoard' class, which allows them to draw shapes and add text on that whiteboard.

Yifei Yu 945753

UML Diagram



Communication Protocols and Message formats

The client-side communicate with the server-side using RMI. The advantage of using the RMI is that the network communication logic is handled by the RMI automatically, and multiple clients can be connected to the server at the same time.

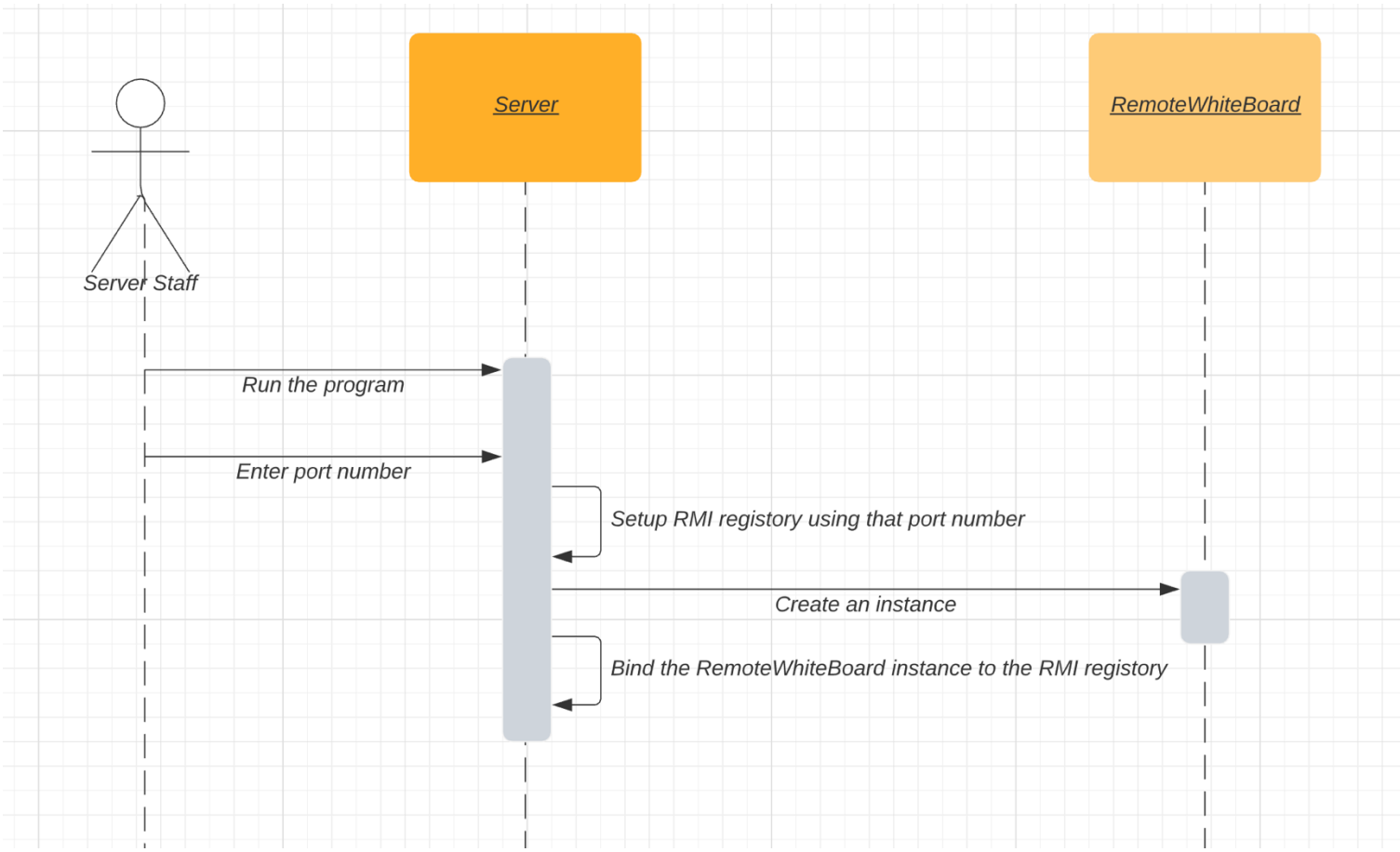
I have created two data format classes that help client-server communication, they are the 'Coordinates' class and the 'TextMessage' class. The 'Coordinates' class stores the coordinate of the text messages on the whiteboard. The 'TextMessage' class stores the text message and the username of that user who sends that text message.

The server sends the whiteboard contents to the client using two ConcurrentHashMap. The first ConcurrentHashMap uses Shape as the key and Color as the value. The client will be able to use this to recreate the drawing part of the whiteboard. The second ConcurrentHashMap uses Coordinates as the key and String as the value, this will recreate the text part of the whiteboard.

The server sends the chat window contents to the client using a CopyOnWriteArrayList<TextMessage>. The text messages are store in the CopyOnWriteArrayList according to the time the text messages are sent.

Interaction Diagram

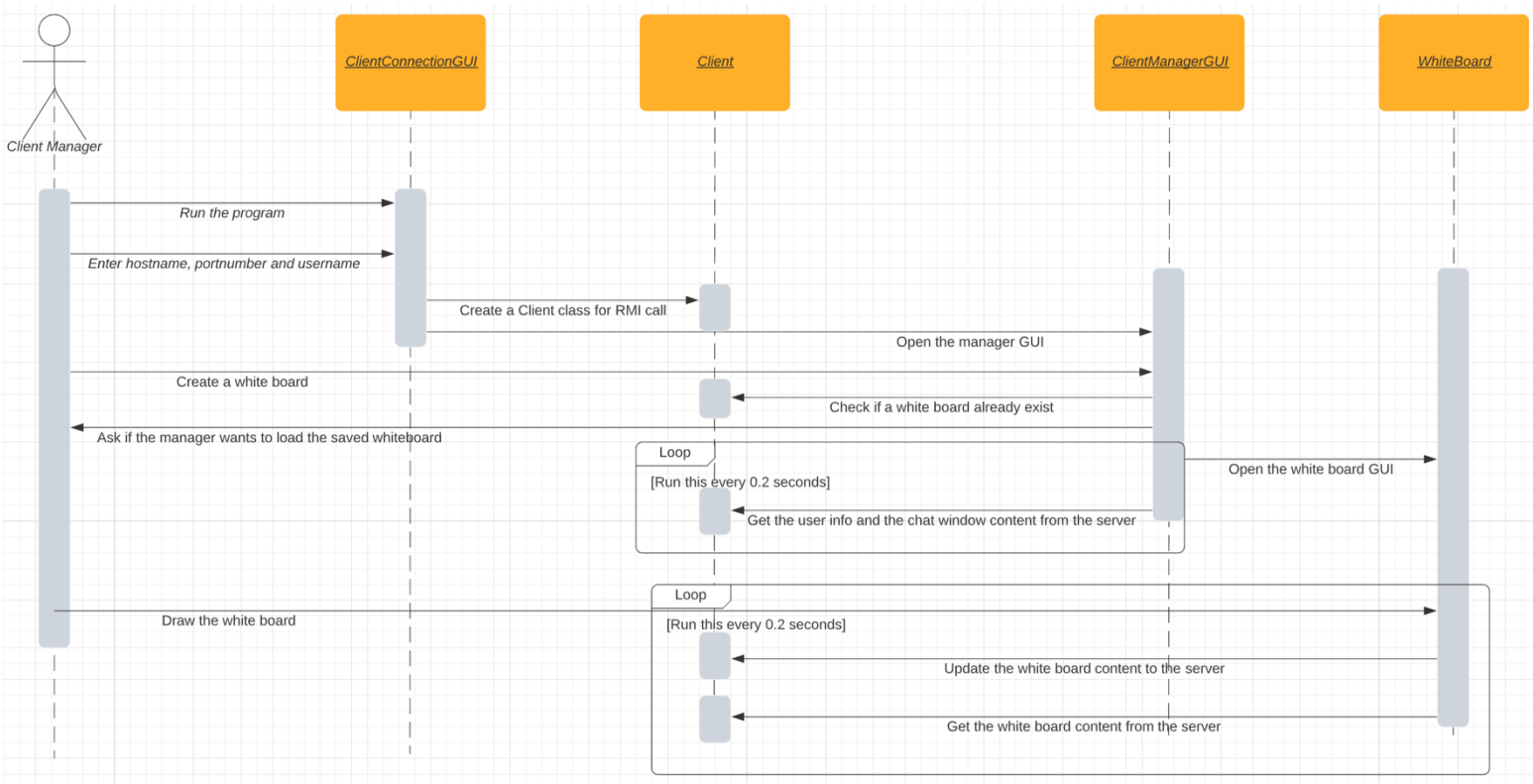
Server



When running the 'Server' class, it will ask the user to choose a port for the server to run on. An error message will be displayed if the port is already in use, or the entered port number is invalid. After that, it will bind an instance of the 'RemoteWhiteBoard' class to that port.

Yifei Yu 945753

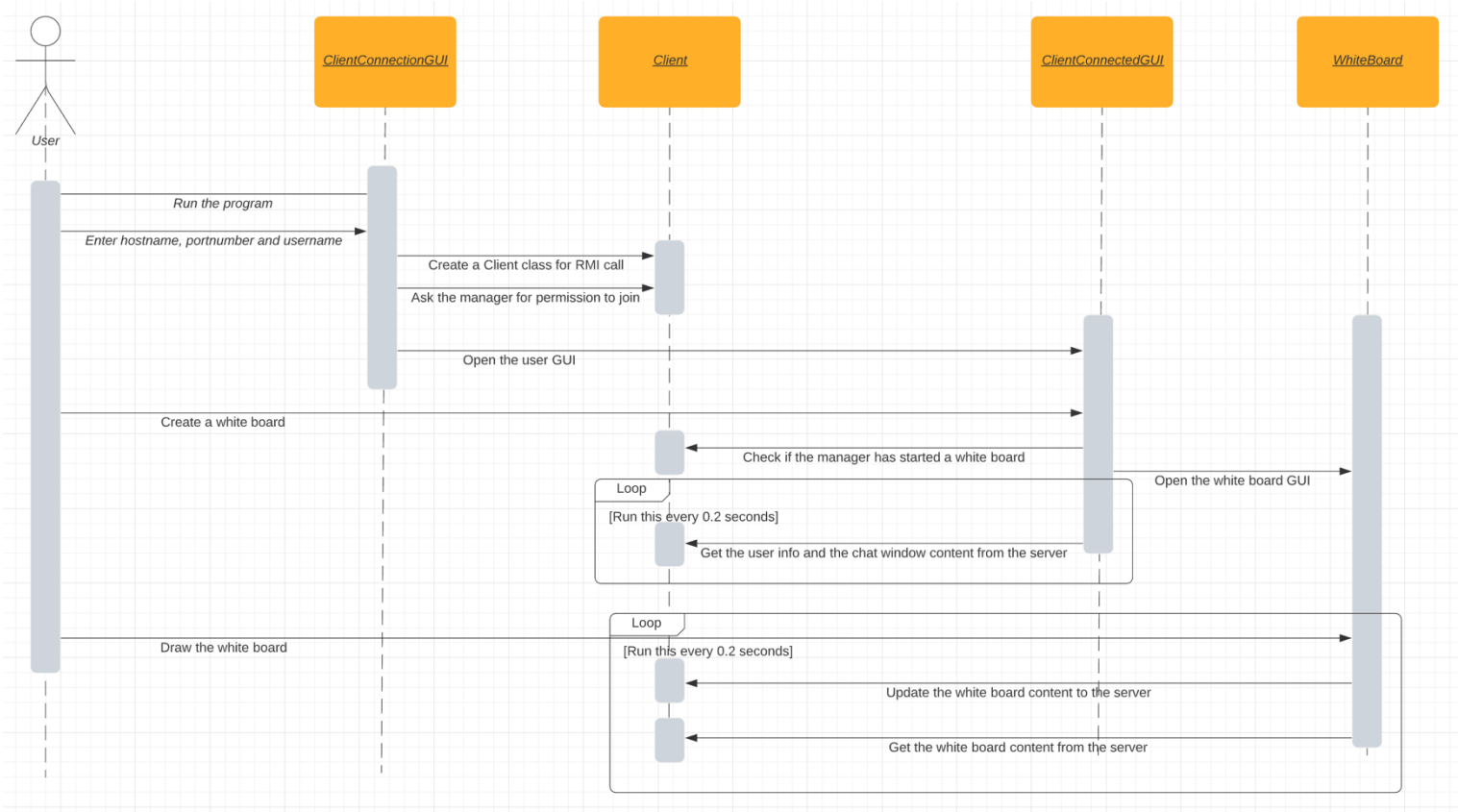
Client Manager



To run the client manager program, we need to run the 'ClientConnectionGUI' first. It opens up a GUI that asks the user to enter the hostname and the port number of the server and the username. An error message will be displayed if the server does not exist, or the username is already taken. If all of the fields are entered correctly, it will assign the first user to connect to the server to be the whiteboard manager and opens up the 'ClientManagerGUI' for the manager. The manager has the privilege to kick out any user at any given time and be able to create or delete the whiteboard. Only one whiteboard can be created at any given time. When the manager creates a whiteboard, it will first check if there is a saved whiteboard image in the local directory. If a save whiteboard image exists, it will ask the manager if he wants to load the saved image on to the new whiteboard. When the manager has selected an option, the 'WhiteBoard' GUI will be displayed, and the saved whiteboard content will show up in the whiteboard if the manager chose to load the save image. The 'WhiteBoard' will update the user's drawing to the server every 0.2 seconds, and it will also pull the whiteboard content from the server every 0.2 seconds.

Yifei Yu 945753

Normal User



Running the normal user program is the same as running the client manager program. There are four differences between these two programs. The first difference is that, when the user wants to connect to the server, it must wait for the client manager to approve its connection. After it has been approved, a user GUI called 'ClientConnectedGUI' will be created. The second difference is that the user does not have the privilege to kick any user. The third difference is that the user cannot save the whiteboard. The last difference is that the user can only join the existing whiteboard that is created by the client manager.

Critical Analysis

Since I am using the RMI, and there is no way for the server to message the client. This means that the client will need to constantly pull information from the server even if there are no new drawings on the whiteboard or new information to be passed to the client. This creates a lot of networks traffics between the client and server, and it is not ideal if there are a lot of clients connected to the same server at the same time as the server will be overloaded with all the incoming network traffics. A workaround is to implement an RMI on the client-side as well so that the server can pass information to the client whenever needed.

Since the client program that I have implemented push and pull information to and from the server every 0.2 seconds. This means that it takes 0.2 seconds for the information coming from the client to be updated to the server, and it takes another 0.2 seconds for the information coming from that client to be distributed among every single client. This creates a small lag between one person drawing and other people seeing that drawing. But I think the small lag is reasonable, and it does not affect the user experience.