COMP90015 Distributed Systems Assignment2:Distributed Shared Whiteboard

Xunkai HU

1145188

xunkaih@student.unimelb.edu.au

I. Introduction

In this project, a shared whiteboard system has been designed and implemented allowing to be edited simultaneously by multiple users. Once the manager creates a new board as a server whiteboard, it keeps waiting for users to join in. After permitted by the manager, users can share the exact board. The system supports a variety of functions, including freehand drawing, drawing various shapes such as lines, circles, rectangles and ovals with specific colors and thicknesses. Moreover, advanced features like chat room, file operation and kicking out users have been added.

II. Structure

By creating the whiteboard, manage himself has established a server for users to join in. In the matter of program structure, both manager and user share a similar logic as shown in Fig 2.1 and Fig 2.2. Overall, a whiteboard is initialized through the JavaFx controller CanvasController.java, and ClientThread.java is required by both manager and user to achieve message exchange. To handle concurrency, synchronized methods are functioned inside the CreateWhiteBoard.java. By doing so, the information updating and broadcasting are operated.

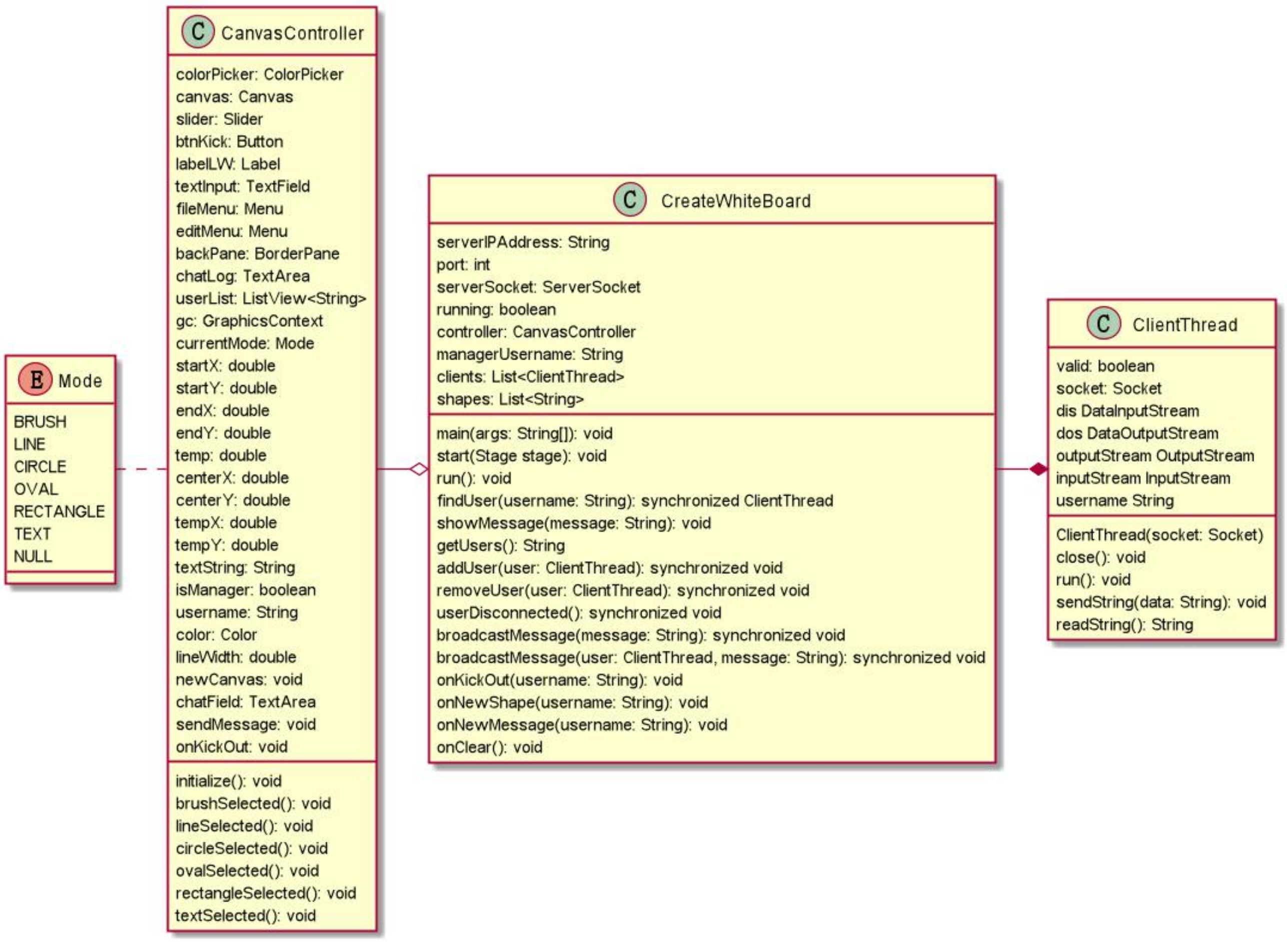

Fig 2.1, Running CreateWhiteBoard.java creates a whiteboard and establish a server

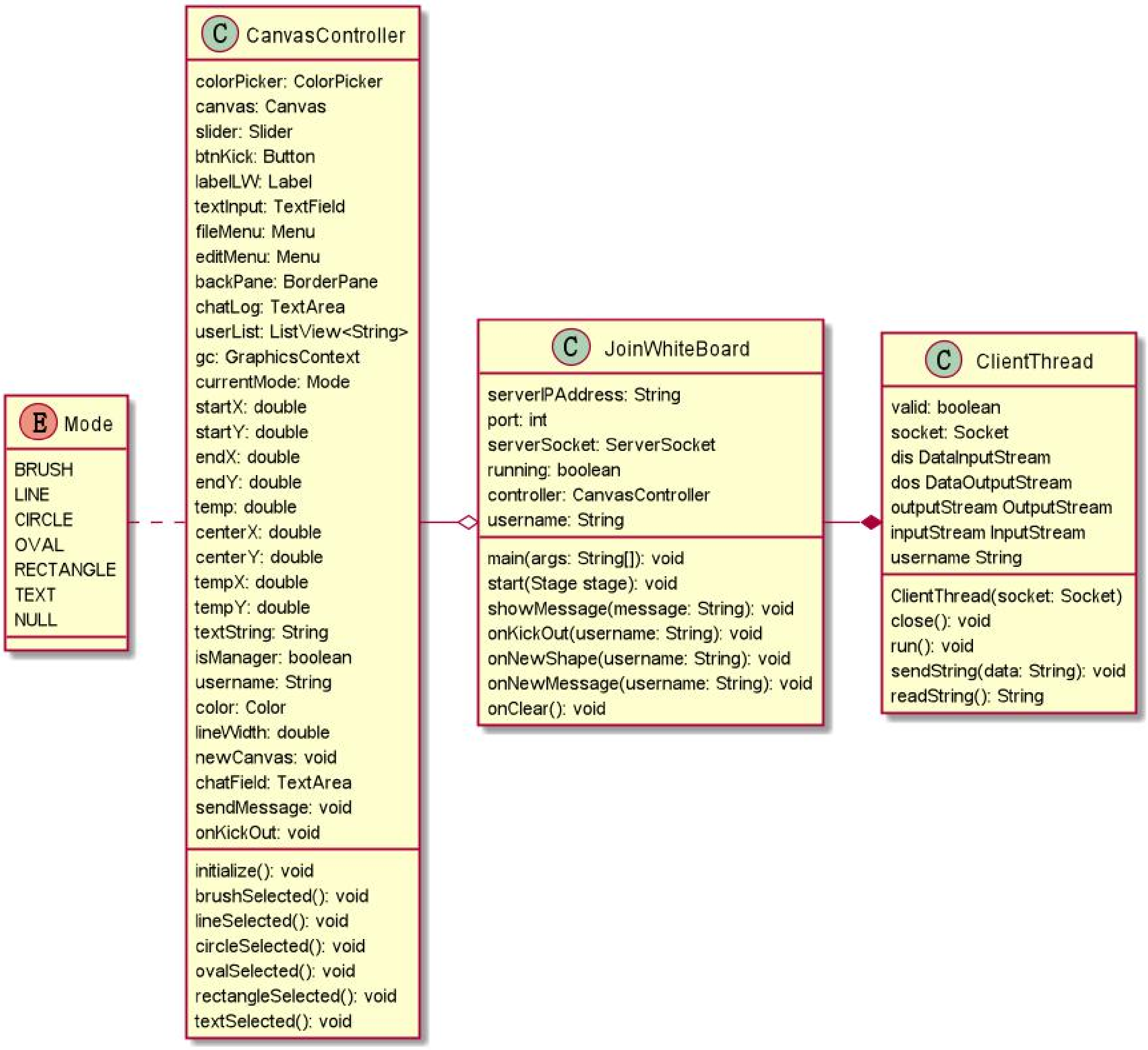

Fig 2.2, JoinWhiteBoard.java has similar contents as CreateWhiteBoard.java with less privileges.

III. Protocols and Architecture

For using TCP protocol, all the clients share the the same messaging format with the server. Messages are sent with a header “shape”, “message”, or “users”. Before sending, the bytes needed for storing are also initialized.

For example, the “shape” stands for drawing sharing, every clients send a message to the server(host) as a format shown following: “Stroke Mode, Coordinates, Line Width, RGB Values, Text String (null excepting TEXT mode)”, then the host records each drawing operation as a log by using StringBUilder.append() . Therefore, the saving and loading can also be implemented by write and read a *.txt file.

For specific options on message exchanging, the sequence diagrams show the actual process as shown in Fig 3.1~3.3.

IV. Whiteboard Design

A JavaFx related scene builder is used to design the main whiteboard GUI. Through javafx.scene.canvas all painting functions including drawing a straight line, oval, circle or rectangle are implemented by handling proper mouse events.As shown in Fig 4.1,the JavaFx also provides a ColorPicker for users to choose their favourite colors easily.

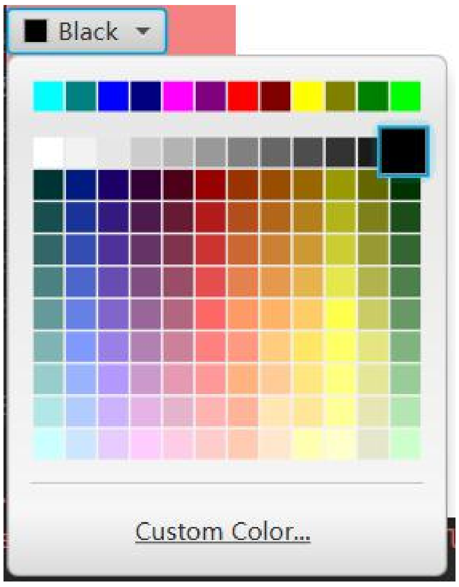

Fig 4.1, ColorPicker.

On the top, the menu tab is only available while logging as a manager, so that only a host can operate the file option.

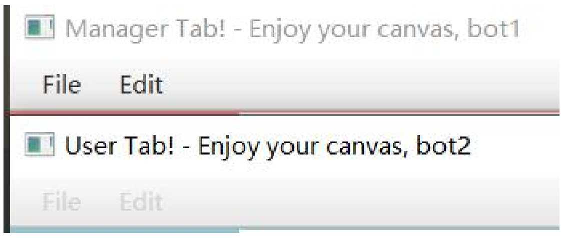
Manager Tab! - Enjoy your canvas, bot1
File Edit
User Tab! - Enjoy your canvas, bot2
File Edit

Fig 4.2, Top menu bar.

The user list and the chat room are set on the right hand side, where users can see other participated peers and chatting with them.

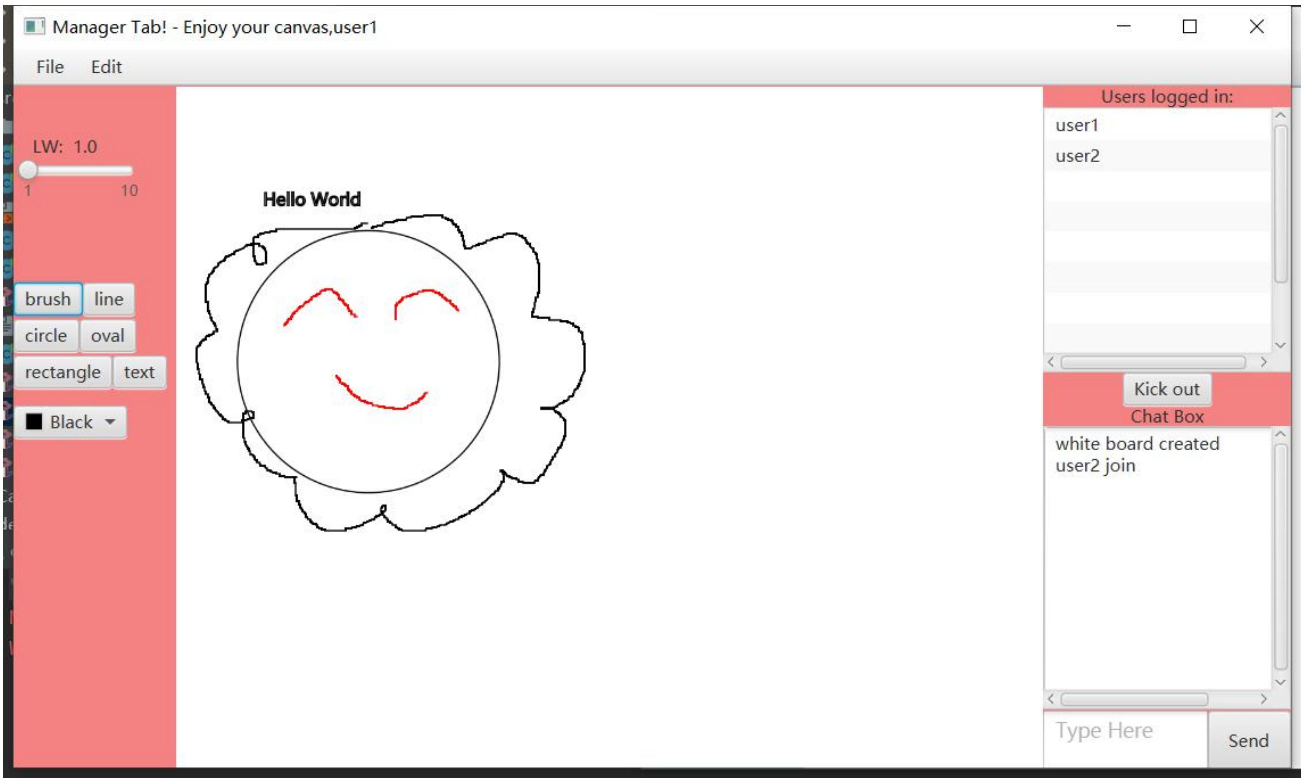

Fig 4.3, Whiteboard design.

V. Critical Analysis

Overall, the system shows a stable connection among users with a flat-designed GUI. The stability is acquired by synchronized modification on the whiteboard. As for easing the pressure to the server brought by massive drawing messages, the stroke is synchronized only when a user's mouse is released from the canvas. However, since the thread-per-connection strategy is applied, the increasing clients may lead to undesirable delays on their networks. Further more, compared to the RMI, TCP sockets system requires a lengthy protocol design for the transmitted message.

For future updates, the whiteboard may replace drawing tool buttons with related icons to make it more user-friendly. In addition, the drawing can be improved to show mouse-dragged results allowing users to preview their drawing outcomes. For dealing with delay issues by extending clients, strategies like thread-pool might be an option. Additionally, a central sever can be added for the system to created a common lobby for multiple painting rooms.

Another useful improvement could be adding Undo & Redo, Rotate, Zoom in and Zoom out features allowing users to modify their painting easily.

VI. Conclusion

The system implements a distributed whiteboard among multiple users via using TCP sockets. Once allowed to join in the system by a manager, users can paint and chat collaboratively. The manager can also save an exist painting or open a new one, and manage other peer users. TCP sockets are applied with thread-per-connection strategy to facilitate message transmissions among peer users.