Design Document

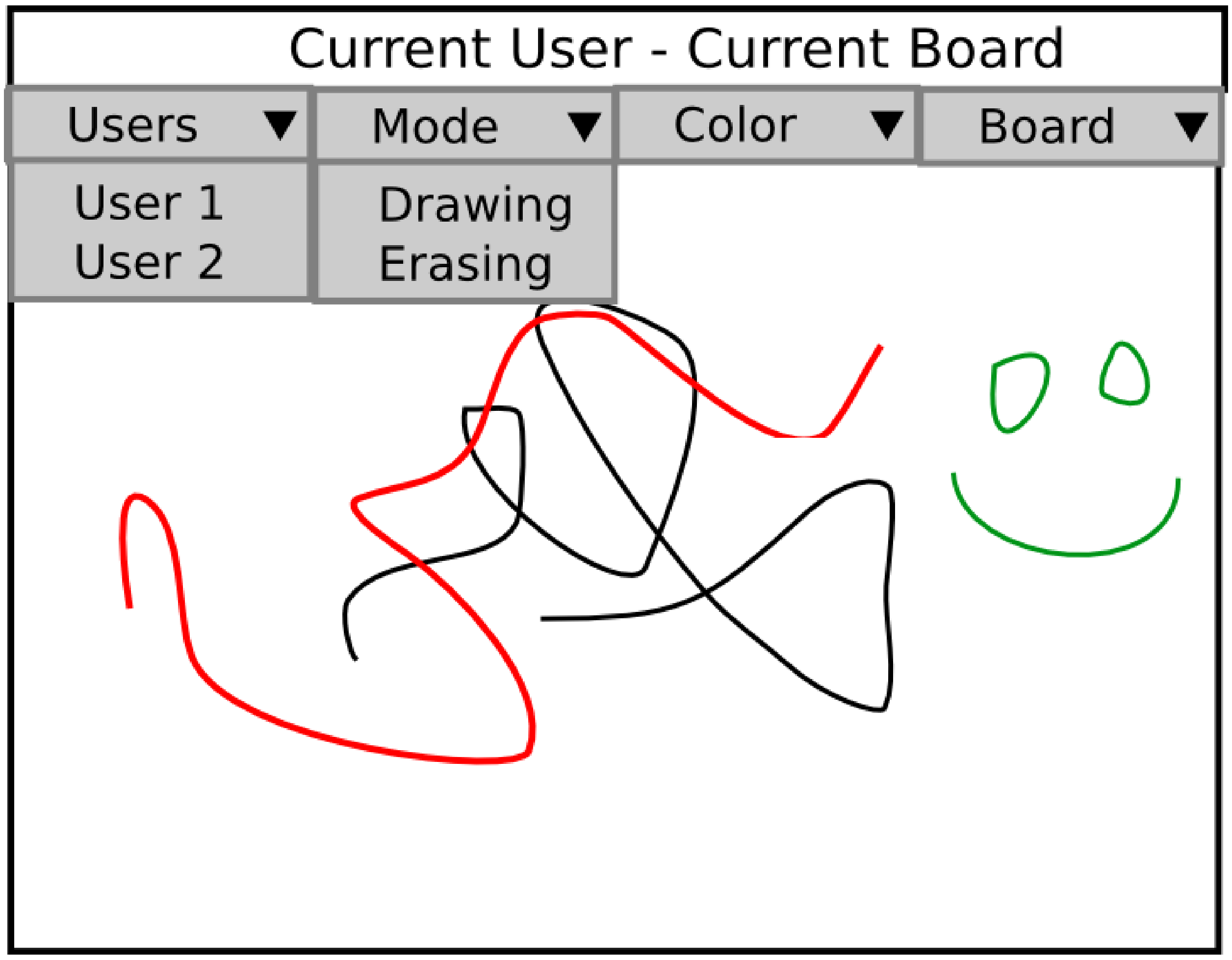
Project 2 - Interactive Whiteboard

Jessica Morgan Andersen

Juan Diego Castrillon

Josh Daniel Haimson

DataType Design:



Whiteboard Design:

The whiteboard will consist of a bitmap image stored on both the server and the client. Drawing commands will be sent from the client to the server and then propagated down to the canvases of all clients. For our first, simple iteration, the user is able to select the mode, color, and board they wish to work on. In addition, the user is able to see the other users who are also currently viewing the whiteboard.

- Mode: (Drawing, Erasing). Drawing will place a free hand draw of stroke 10 of whichever color is selected. Erasing is similar only that it will draw in white instead.
- Color: User is able to select from a color palette
- Board: User is able to select from the current open boards or select a new board.

Two users drawing simultaneously on the whiteboard will never create a problem because of the way the protocol is established between client and server. The client is only sending changes to the server while the server is pushing those changes back to each client. Thus, when two clients are modifying the whiteboard, they will each push their changes and the server will push both of those changes back to the clients.

If multiple users draw on the same pixel, the user who drew on the pixel last will show on the whiteboard. In this way, the latest user is drawing on top of the other drawings.

Java Classes:

- Client
    - Data
        - String username
        - String currentBoardName
        - Canvas canvas
        - JFrame frame
        - Color currentColor
        - float currentWidth
        - Socket socket
        - String boards[]
    - Methods
        - public void switchBoard(String newBoardName)
            - Switches currentBoardName to newBoardName (a Board that has already been created)
            - Gets image of this canvas from the server
            - Calls update users
        - public void newBoard(String newBoardName)
            - adds a new Board on the server
            - sets newBoardName to currentBoardName
            - wipes the canvas clean
            - sets users to only the current user
        - public void updateCanvasImage(BufferedImage newImage)
            - sets the image of the canvas to the newImage
        - public void updateCanvasCommand(String boardName, Command command)
            - checks that the currentBoardName and boardName are the same
            - performs the command on the canvas
        - public void updateUsers()
            - Gets the users for the current board from the server and sets them
        - public void setCanvasUsers(String[] users)
            - Updates the list of users for the canvas
        - public void updateBoards()
            - Updates the list of boards available
        - public void changeColor(String newColor)
            - changes the stroke's color, currentColor, to newColor
        - public void changeWidth(float newWidth)
            - changes the stroke's width, currentWidth, to newWidth
        - Getters/Setters as needed

- Canvas
    - Data
        - BufferedImage drawingBuffer
        - String name
        - String[] users
        - EventListener currentListener
        - Client client
    - Methods
        - public Thread drawLineSegment(int x1, int y1, int x2, int y2, Color color, float width)
            - Draws line on local buffer and then spins off thread which transmits line data to server
        - public void repaint()
        - Getters/Setters as needed
- ClientSendProtocol implements Runnable
    - Data
        - Socket socket
        - String message
    - Methods
        - public void run()
            - sends message to server over a PrintWriter
- ClientRecieveProtocol implements Runnable
    - Data
        - Socket socket
        - Client client
    - Methods
        - public void run()
            - wait for message from server and calls appropriate request handler
        - public void updateUsers(String usersMessage)
            - parses the message into a list of users
            - calls client.setCanvasUsers
        - public void updateBoards(String boardsMessage) {
            - parses the message into a list of boards
            - calls client.setBoards
        - public void updateCanvasImage(String imageMessage)
            - parses the imageMessage into a 2D array then into a BufferedImage
            - Calls client.updateCanvasImage
        - public void updateCanvasCommand(String commandMessage)
            - Parses the string into a command object then calls client.updateCanvasCommand
- Server

- Data
    - Map<String, SimpleCanvas> boards
    - Socket[] clients
- Methods
    - public synchronized void updateClients(String command, Socket socket)
        - Iterates through all clients except for the one given in the parameters and passes the new command back to all of them
    - public void updateBoard(String boardName, Command command)
        - performs the command on the SimpleCanvas for the boardName
    - public boolean newBoard(String boardName)
        - Creates a new board with the specified boardName
        - returns whether or not the name was unique and therefore successful
    - public void switchBoard(String username, Socket socket, String oldBoardName, String newBoardName)
        - Removes user from oldBoard and then adds user to newBoard
        - Updates the user's canvas to have the BufferedImage of the newBoard
    - public void exit(String username)
        - Remove user from all boards
    - public void enter(String username, String boardName)
        - Adds the user to the board after they've picked one
    - public String[] getBoards() {
        - returns all board names
    - public boolean checkUsers(String username, String boardName)
        - Checks to see if the username is already taken and if not calls enter

- ServerProtocol implements Runnable
    - Data
        - Socket socket
        - Server server
    - Methods
        - public void run()
            - Waits on client to send data and then calls appropriate request handler
        - public void updateBoard()
            - Sends the command in string format to all clients with updateClients
            - Parses the command into a Command and calls server.updateBoard
        - public void newBoard()
            - calls server.newBoard
            - returns the result (a boolean) to the client

        - public void switchBoard()
            - calls server.switchBoard
        - public void exit()
            - Close connections and calls server.exit
        - public void getBoards()
            - calls server.getBoards and returns the result to the client
        - public void checkUsers()
            - calls server.checkUsers and returns the result to the client
- SimpleCanvas
    - Data
        - BufferedImage drawingBuffer
        - String name
        - String[] users
        - int width
        - int height
    - Methods
        - Same as canvas except with all JFrame and EventListener references removed
- Command
    - Data
        - String command
        - String[] arguments
    - Methods
        - public void invokeCommand(Object canvas)
            - Determines whether the object is a Canvas or SimpleCanvas, then finds the method with a name matching the command name, then invokes the method with the command's arguments
        - public void parseCommand(String commandString)
            - Parses a string received from the client that has already been determined to be a draw command

Protocol:

Client to Server interactions:

- New Board = “new *boardName*”
- Switch Board = “switch *username newBoardName*”
- Exit = “exit *username*”
- Draw = “draw *boardName command param1 param2 param3*”
    - Example: “draw *boardName* drawLineSegment *x1 y1 x2 y2 color width*”
- Get Users = “users *boardName”*
- Get boards = “boards”
- Check Users = “check *username boardName*”

Server to Client interactions:

- Update Users = “users *boardName user1 user2 user3...*”
- Update Available Boards = “boards *board1 board2 board3*”
- Draw = “draw *boardName command param1 param2 param3*”
    - Example: “draw *boardName* drawLineSegment *x1 y1 x2 y2 color width*”
- Update Board = “board *boardName sample1 sample2 sample3*
- Check Users = “check *username boolean*”
- New Board = “new *boardName boolean*”

Concurrency Strategy:

Client Side Concurrency:

- GUI will be made thread-safe by using Java Swing's asynchronous invokeLater method
- The local Canvas is only used as a temporary canvas to ensure that drawing is smooth and not delayed by requests from the server. Thus the master Canvas is stored on the server and any updates on the server will overwrite all updates to the local Canvas. Since all updates to the local canvas are sent to the server anyway, they will eventually propagate back down to the local Canvas. Thus, there are no concurrency issues with the Canvas or Board objects on the Client Side as they will always be overwritten by what's on the server.

Server Side Concurrency:

- Server.updateClients will be synchronized - this will ensure that no changes can be made to any of the boards while new data is being pushed out to all clients
- All methods that access the boards' rep invariants will also be synchronized - this ensures that two clients can't simultaneously modify a board

Deadlocks

- Since there are no methods which require nested locks, there is no chance of deadlock with this design

Race Conditions

- The precautions taken to make the server and board classes thread-safe will prevent race conditions from occurring

Testing Strategy:

Basic level testing:

- Testing the GUI
  - We will visually test that drawing on a canvas work
  - Drawing with different colors
  - Drawing with different Stroke Widths
    - Test user input data (must be positive integer no greater than 15 for example)
- Testing new Boards
  - Creating a new Board
  - Testing switching between Boards
  - Test the stroke width, color are maintained in new Boards.
  - Creating more than 10, 15, and 20 new Boards
  - Testing switching between multiple Boards

Testing Multiple Users

- Testing One Board, Two Users:
  - Testing drawing
    - one user then the other
    - users drawing simultaneously
    - testing fidelity in stroke weight and color
  - Testing Active Users on Board
- Testing Multiple Boards, Two Users:
  - Same Process as above
- Testing Multiple Boards, Multiple Users:
  - Same Process as above

This will be mostly tested visually because it is hard to test the GUI using JUnit tests. However some components will be able to be tested using JUnit Tests such as the protocol. Most of the bugs will come from handling multiple users and multiple boards and concurrency; thus, extra caution must be taken into account when dealing with multiple users on a single board. Therefore we plan to systematically test concurrent threads.

To generate our tests, we will be partitioning our input space, and will use a full Cartesian product strategy to ensure maximum coverage. We will test for bad inputs, for critical input values, and will brainstorm ways that someone might try and break our code and then test for that. We will be systematic with our tests, will test early and often, and will automate the running of our tests.

•