Design Document

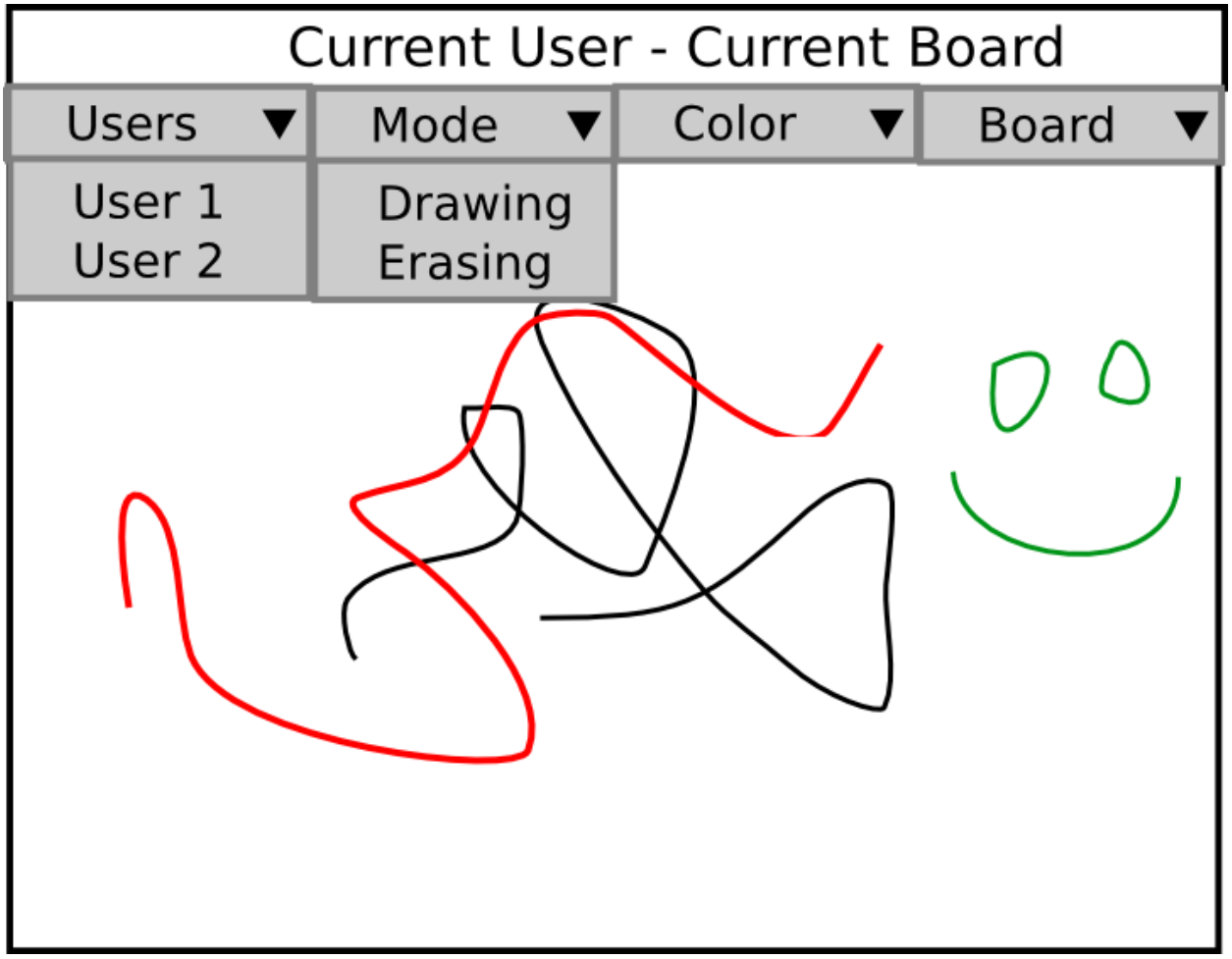
Project 2 - Interactive Whiteboard

Jessica Morgan Andersen

Juan Diego Castrillon

Josh Daniel Haimson

DataType Design:



Whiteboard Design:

The whiteboard will consist of a bitmap image that is passed from the server to the client. Drawing commands will be sent from the client to the server. 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 whiteboard to the server while the server is pushing the updated whiteboard back to each client. Thus, when two clients are modifying the whiteboard, they will each push their changes and the server will push back the updated completed image of the whiteboard with both of their changes.

If multiple users draw on the same pixel, the user whom 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:

- InteractiveWhiteboard
  - Data
    - String username
    - String[] boards
    - Board currentBoard
    - JFrame frame
    - String currentColor
    - float currentSize
    - Socket socket
  - Methods
    - public void switchBoard(String newBoard)
      - Switches currentBoard to newBoard (a Board that has already been created)
    - public void newBoard()
      - makes a new Board and sets it as currentBoard
    - public void updateBoard(String boardID, Image newCanvasImage, String[] newUsers, String[] newBoards)
      - Asserts that we are updating the right board
      - Updates boards array
      - Calls CurrentBoard.update with new image and users
    - public void changeColor(String newColor)
      - changes the stroke's color, currentColor, to newColor
    - public void changeSize(float newSize)
      - changes the stroke's width, currentSize, to newSize
    - public void repaint()
      - calls currentBoard.repaint()
      - updates user and id display
    - Getters/Setters as needed
- Board
  - Data
    - String[] users
    - String ID
    - Canvas canvas
  - Methods
    - public synchronized void update(String[] users, Image newImage)
      - Changes board's canvas image to newImage
      - Updates users array
    - public synchronized void removeUser(String userID)
      - Removes userID from users array

        - public synchronized void addUser(String userID)
            - Adds userID to users array
        - public void repaint()
            - calls canvas.repaint()
        - Getters/Setters as needed
- Canvas
    - Data
        - Image drawingBuffer
    - Methods
        - public void updateImage(Image newImage)
            - Updates canvasImage to new image
        - 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
        - InteractiveWhiteboard whiteboard
    - Methods
        - public void run()
            - wait for message from server and calls appropriate request handler
        - public void updateBoardData()
            - calls whiteboard.updateBoard(String boardID, Image newCanvasImage, String[] newUsers, String[] newBoards)
        - public void updateCurrentBoardID(String newBoardID)
            - calls whiteboard.switchBoard(newBoardID)
- Server
    - Data
        - Map<String, Board> boards
        - Socket[] clients
    - Methods
        - public synchronized void updateClients()

                - Iterates through all clients and updates data
            - public void updateBoard(String boardID, boardData)
                - Updates a specific board with new data
            - public String newBoard()
                - Creates a new board and returns unique board id
            - public void switchBoard(user, oldBoardID, newBoardID)
                - Removes user from oldBoard and then adds user to newBoard
            - public void exit()
                - Remove user from all boards
            - public void drawLineSegment(boardID, x1, y1, x2, y2, color, width)
                - Calls draw lineSegment on board with boardID
- ServerRecieveProtocol implements Runnable
    - Data
        - Socket socket
        - Server server
        - String currentBoardID
    - Methods
        - public void run()
            - Waits on client to send data and then calls appropriate request handler
            - After handler is called, calls server.updateClients()
        - public void newBoard()
            - calls server.newBoard()
            - Writes a new Board server response to inform client of new boards id
        - public void switchBoard()
            - calls server.switchBoard(user, oldBoardID, newBoardID)
        - public void exit()
            - Close connections and calls server.exit()
        - public void drawLineSegment(boardID, x1, y1, x2, y2, color, width)
            - Calls server.drawLineSegment

Protocol:

Client to Server interactions:

- New Board = “New Board”
- Switch Board = “Switch Board “ + (ID Number of board)
- Exit = “Exit”
- Draw = “drawLineSegment boardID x1 y1 x2 y2 color width”

Server to Client interactions:

- Update = “Update {boardID} {user1 user2 user3} {board1 board2 board3} {binaryImage}”
- New Board = “New Board BoardID”

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.