William Haack
Shawn Jain
Blake Elias
Design Doc

Brief Overview:

Our application is split into three parts: a server, a whiteboard picker client, and a whiteboard client. The whiteboard picker client is the portion of the application that lets users choose which whiteboard they want to work on and allows them to create new blank white boards for them and others to use. The white board client allows users to draw on a white board and broadcast their actions to the server for all the other white board clients to see.

Data Types

Whiteboard: the underlying data structure for a whiteboard is a 2D array of pixels, each of which also has a color. We will use Java's ImageBuffer to store this. There is a method called whiteboard.applyAction(Action a) - it takes an Action object (described below) and mutates the ImageBuffer stored in whiteboard to reflect this action.

Action: an action represents a segment that can be drawn between two points on the board. Specifically, it has int fields x0, y0, x1, y1 representing the coordinates of the endpoints of a segment, a Color field called color representing the color of this segment, and an int field called strokeWidth representing the width of this pen stroke on the board.

Server:

Data Structures:

The server maintains a list of all the whiteboards ever created:

- whiteboards (List<MasterWhiteboard>)

Where the MasterWhiteboard type contains a complete master list of all actions performed on that board:

- history (java.util.BlockingQueue<Action>)

Any time we refer to a "whiteboard" in the context of our server program, we are referring to an object of type MasterWhiteboard (to distinguish from the client-side Whiteboard type, which does not contain a full history of actions but simply stores an image as pixels).

The server also maintains a map from boards to a lists of users connected to those boards

- whiteboardToUsers (Map<MasterWhiteboard, List<User>>)

The server maintains a few threads:

Communication with white board picker client:
The server has a thread that listens for messages that inquire how many whiteboards there are. This gives the whiteboard picker client a list of all the whiteboard available whiteboards, since whiteboards are represented by the integer corresponding to the order they were created in.
This same thread can also take a message that adds a new whiteboard to the list of whiteboards. Lastly, this thread can take an “open whiteboard” message which will spawn two new threads that handle the connection between the server and a whiteboard (i.e. the white board client that is simultaneously spawned on the user’s machine).

Communication with white board client:
For each whiteboard client, there are two threads running on the server.

- One thread opens a socket connection to the whiteboard client running on the user’s machine. It constantly listens (and blocks execution while waiting) for a message containing an object of type List<Action>. Whenever it receives this action list it pushes all the Actions from that list into whiteboard.history (must acquire a lock on whiteboard.history first, so as not to interfere with the updates in the second thread described below). (Where whiteboard is the MasterWhiteboard object for this board which is shared amongst all the threads communicating with clients connected to this whiteboard). This thread is what makes whiteboard.history have a complete history of every action drawn on the board.
- The other thread runs a loop which constantly looks at the shared queue whiteboard.history,
    - sending all new[1] actions in this queue to the client (this is how one client hears about all other clients’ changes - because all threads of the first type, described above, are pushing the actions they receive from their clients onto the history queue)
    - updating a variable, lastQueueSize, set to the value of whiteboard.history.size(), private to this thread
    - The above two operations must be executed together as one atomic operation, and will only run when they acquire a lock on whiteboard.history. This is how we prove that no new actions are added from the first thread before the second thread updates its lastQueueSize, as such a condition would cause some actions to potentially not be sent to all clients.

User:

[1] The variable lastQueueSize is what allows the thread to send only new entries in whiteboard.history to the client. The only entries that need to be sent are indices in the range [lastQueueSize, whiteboard.history.size() - 1].

Whiteboard picker client:

When the user first opens our program, the program opens just a whiteboard picker client. When the whiteboard picker client starts it opens a connection to the server and displays all of the available whiteboards to edit. Upon double clicking a whiteboard a new window opens up with a white board client for the chosen board (a new thread, as described in the next paragraph). The user can open multiple whiteboards at a time by going back to the list of boards in the picker client - each will spawn its own whiteboard client. When the user clicks a button “create new whiteboard” the whiteboard picker client sends a message to the server in a new thread, and on response modifies the list of available whiteboards.

Whiteboard client:

The whiteboard client displays a given whiteboard and allows users to draw on it. The whiteboard client has a thread running that listens for user input and when the user makes these inputs it does both of the following:

- adds the actions to a BlockingQueue called actionsToSend.
- draws these actions on the Whiteboard on the screen (by calling whiteboard.apply(action) as well as gui.repaint() on the GUI object).

At every timestep, there is another thread that sends the actionsToSend queue to the server. There is a third thread that constantly listens for messages from the server (which are drawing actions) and applies them to the canvas, calling repaint() on the GUI after all received actions from that iteration are applied.

The above 3-thread system means a user’s actions will be drawn to his own screen immediately, but will also be sent to the server, and that the user’s board will eventually be updated (overwritten) with new, synchronized state from the server. This system has the property that all clients eventually will have the same picture on their screen if provided enough time for all network transactions to complete.

The repaint is only once at the end of an iteration, rather than after every action, to give a slight speedup and to prevent a “flickering” effect as multiple actions are drawn that possibly overlap each other. If another user’s change really should have come before the local user’s change, but the local user’s change was drawn first because it was done locally, the other segment will be smoothly drawn “under” the user’s change once it is received over the network without any flickering (a “mine” - “theirs” - “mine” switcheroo would look annoying).

Testing Strategy:

We will have a testing strategy for each of the main components of our application.

Testing Server:

The server is considered working if we get the responses/behavior we expect after giving the server some mock messages from the client. We will test we get the expected responses/behavior from

- Creating a query for the list of white boards
- Sending a create new whiteboard query
- Sending a List<Action>

Testing whiteboard picker client:

The whiteboard picker client is considered working if it displays all the current whiteboard, gives the ability to create a new whiteboard, and allows users to open a new whiteboard. We will test we get the expected responses/behavior from

- Creating a new whiteboard
- Opening a whiteboard from the list.

We will also manually test to make sure all the visual components of the application are working.

Testing whiteboard client:

The whiteboard client is considered working if it allows the users to properly draw on the board, broadcasts those drawings to the server, and applies actions that are broadcasted from the server. We will test we get the expected response/behavior from

- Clicking any of the paintbrush modifying buttons
- Having a user draw on the board
- Having the server send a list of actions to apply

Like before, we will manually est to make sure all the visual components of the application are working.

Protocol:

Server<->Whiteboard Picker Client

The whiteboard picker client can send the following two messages

1. Get all whiteboards
2. Create new whiteboards

The server responds to message 1 with a List<Integer> and to message 2 with an Integer (the whiteboard created.)

Server<->Whiteboard Client

The whiteboard client only sends the following type of message:

List<Action>

The server also only sends a list of actions to the client (the actions that should be applied.)