Design Document

Project 2 – Collaborative Whiteboard

Genghis Chau, Kevin Peng, Parker Zhao – December 11th, 2013

1 TABLE OF CONTENTS

2 SUMMARY OF CHANGES

The largest changes to the design document since the revised milestone design were to the datatypes we defined. While the Whiteboard class went mostly unchanged aside from a few small performance adjustments, the User datatype was almost completely revamped. Many of the objects included in the User datatype were removed, mostly because they were more relevant to individual whiteboards than to individual users. Many methods were also deleted as they were unnecessary and extraneous. In addition to the two previously defined datatypes, an extra datatype was added, LineSegment.

The protocol was also heavily expanded on, as we found that we needed more types of messages between the server and client to properly signal each other. Lastly, our implementation ended up using more threads than we had originally expected, so more concurrency strategies were employed to keep race conditions and deadlock from occurring.

3 DATATYPE DESIGN

The first datatype implemented is the LineSegment datatype, which represents a short, colored line segment. The class is used a building block for the whiteboard drawings, as it holds information for the segment's color, thickness, and endpoints. Since drawings can be broken up into a series of many short line segments, this class allows the drawing to be converted into simpler, easier-to-reproduce components. Each LineSegment instance is immutable, and only has appropriate observer methods for its instance variables.

LineSegment (Used in both client and server):

Point start
: Represents the start point of the line segment.

Point end
: Represents the ending point of the line segment.

Color color
: Stores the color (in RGB) that the line segment is drawn in.

int strokeSize
: Represents the width of the line segment.

public Point getStartPoint()
: Returns the start point.

public Point getEndPoint()
: Returns the ending point.

public Color getColor()
: Returns the color of the line segment.

public int strokeSize()
: Returns the width of the line segment.

public String toString()
: Returns a String containing a protocol-friendly representation of the segment.

The next datatype implemented is the Whiteboard datatype, which represents the entire image as a whole. The Whiteboard stores its information as an array of LineSegments, and since LineSegment objects are easy to use and to send as data, Whiteboard instances can be easily reproduced. There are two versions of the Whiteboard datatype – one that the server uses and one that the client uses. Since the client must have a GUI to view the image, the client uses a version of Whiteboard with GUI elements, including a WhiteboardGUI object and a Canvas object, and is able to display the actual line segments on the screen using these classes. The client also stores the array of LineSegment objects that compose the drawing. The server on the other hand, does not require the GUI and thus does not use either of the two GUI elements. The server version only relies on the array of LineSegments, which is sufficient to keep track of what strokes have been made on the whiteboard. Both versions of the Whiteboard datatype include a name and set of users. The name is used to uniquely identify whiteboards, and the set of users is used to keep track of the names of users currently editing the image.

Whiteboard (Used in both server and client):

String name

The name of the whiteboard.

HashSet<String> users

A set of usernames of the users who are viewing/editing the whiteboard.

ArrayList<LineSegment> lineSegments

An array of LineSegments representing the lines drawn on the whiteboard.

Canvas canvas *(is non-null only on the client)*

GUI element that allows users to draw and edit the whiteboard.

WhiteboardGUI gui *(is non-null only on the client)*

GUI element that combines the canvas and the list of users editing the same whiteboard.

public String getName()

Returns the name of the whiteboard.

public HashSet<String> getUsers()

Returns a set of the users viewing the same whiteboard.

public void setUsers(HashSet<String>)

Adds a set of users to the list of currently viewing users.

public boolean addUser(String)

Adds a single user to the list of currently viewing users and returns if the user was successfully added.

public boolean removeUser(String)

Removes a single user from the list of currently viewing users and returns if the user was successfully removed.

public void addLineSegment(LineSegment)

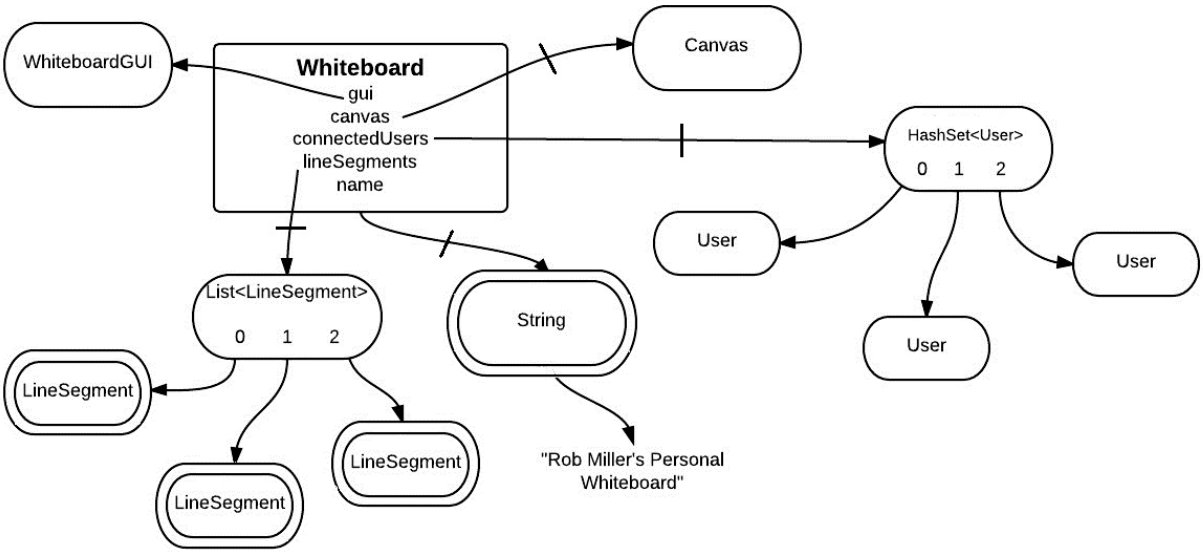
Adds a line segment to the whiteboard and draws it on the GUI if being used by the client.

public ArrayList<LineSegment> getLineSegments()

Returns the array of line segments in the drawing.

public void display()

Initializes the whiteboard GUI. Only called from the client.



The third and final datatype is the User datatype. This datatype represents a user using the whiteboard application and holds much of the information that the user needs. The user will have access to his own version of the shared whiteboards, which will be stored in a hash map. The user will also have a username, which will be used by the server to identify different clients.

User (Used in client):

HashMap<String, Whiteboard> whiteboards

A hash map holding a copy of each of the whiteboards that the user currently has open. The String used to hash is the name of the whiteboard.

String username

The username chosen by the user.

public String getUsername()

Returns the current username of the user.

public void addWhiteboard(Whiteboard)

Adds a whiteboard to the hash map. This implies that the user has opened the whiteboard up for editing.

public void removeWhiteboard(String)

Removes a whiteboard from the hash map. This implies that the user has closed the whiteboard.

4 PROTOCOL

Client-server interaction will use a simple HTTP protocol to transfer data. Since whiteboards are represented as an array of LineSegment objects, most of the interaction is done by exchanging data about line segments. Other requests, such as requesting usernames, joining a whiteboard, editing a whiteboard, updating the list of current users, and disconnecting, are also handled by sending HTTP requests.

List of Protocol Requests

Client to Server Messages

add username [USERNAME]

Sends a requested username to the server. The server will check if the username is available; if it is taken by another user, then the server will reply accordingly. Otherwise the server reports success to the client.

disconnect username [USERNAME]

Sends a disconnect signal to the server. The server will then remove the user from the list of online users so other connecting users can take the username.

create whiteboard [WHITEBOARD]

Sends a request to the server to create a new, blank whiteboard. The server will check if the server name is available and either report success or failure accordingly to the client.

join whiteboard [WHITEBOARD]

Sends a request to the server to join a specific whiteboard. If the whiteboard exists, the server will send a success signal back to the client, a list of the users looking at the whiteboard, and the line segments that have been drawn prior to joining.

exit whiteboard [WHITEBOARD]

Sends a request to the server to exit a specific whiteboard. The server will remove the user from the whiteboard's list of users and report success assuming the client has joined the whiteboard. Otherwise, it will report an error.

draw whiteboard [WHITEBOARD] [INT] [INT] [INT] [INT] [INT] [INT] [INT] [INT]

Sends a draw line request to the server. The server will add the corresponding line segment to the server copy of the whiteboard and then tell each of the connected users to make the line segment visible on their own copies. The first four integers are the x and y-coordinates of the start point and end point, respectively. The next three integers are the RGB values of the color of the line segment while the last integer is the width of the stroke.

Server to Client Messages

retry username

Sent when the user has chosen a username that is already taken on the server or is invalid (contains whitespace). Will force the user to enter in another username.

success username

Sent when the user has chosen an available and appropriate username.

success whiteboard join [WHITEBOARD]

Sent when the user has successfully joined the given whiteboard.

list whiteboard [WHITEBOARD] [WHITEBOARD] [WHITEBOARD] …

Sent when the user first connects to the server or the list of whiteboard changes. Each word after the word "whiteboard" is the name of a whiteboard. This serves to tell the client what whiteboards currently exist on the server.

success whiteboard exit [WHITEBOARD]

Sent when the user has successfully exited a whiteboard he/she was using.

error whiteboard

Sent when an error occurs in joining or exiting a whiteboard. This normally indicates that the user tried to join or exit a non-existent whiteboard, or tried to join a whiteboard he/she was already in or exit a whiteboard he/she wasn't already in. These messages should never realistically be sent from the server to the client because of the way the client GUI works, but exist as a failsafe.

add whiteboard [WHITEBOARD]

Sent to all connected users when a new blank whiteboard is created.

retry whiteboard naming
Sent when the user tries to create a whiteboard with the same name as another one currently on the server.

list whiteboard-user [USER] [USER] [USER] …
Sent when the user first joins a whiteboard. Each word after "whiteboard-user" is the username of a user currently viewing the whiteboard. This serves to notify the newly joining user of the other users already editing the whiteboard.

add whiteboard-user [USER]
Sent to all users viewing a whiteboard when a new user joins the whiteboard.

remove whiteboard-user [USER]
Sent to all users viewing a whiteboard when a user leaves the whiteboard.

5 OVERALL DESIGN

This project represents a collaborative whiteboard, which has a central server and a client that can connect to the server. The whiteboard gives the user the ability to draw concurrently with other users in a variety of colors, as well as erase. To log into the server, users must choose a unique identifier, which cannot contain whitespace. For simplicity, the server registers all usernames are purely lowercase, preventing users from using different capitalization to confuse other users. Users are also able to create whiteboards and give them unique names, which much also not contain whitespace. These whiteboard names are also automatically converted to lower case so that they can be more easily hashed on the server. Since the whiteboard allows for concurrent drawing, the server will receive both of the edits and based on which edit is logged onto the queue first (if the edits are at the same time, this will be arbitrary), the last edit on the shared region will be the one shown on the screen for clients.

Server – *package* server

The program is split into two major components – the server and the client. The server consists of four classes: Packet, WhiteboardMainServer, WhiteboardServerThread, and WhiteboardDataServer. The Packet class is used to more compactly send different types of information between the various threads on the server. There are three types of packets sent on the server, and the type of packet dictates what data it stores. The first type is a Queue packet, which is used when a client requests a username from the server. Prior to the user picking a username, the server thread that is handling that client's messages has no direct way for the data server to send back data. When the username tries to pick one, the server thread also sends along a BlockingQueue to the central data server so that it can receive data back. The second type is a Server packet, which is used when the central data server sends back messages to the individual server threads so that it can be forwarded back to the proper clients. Lastly the third type is a Normal packet, which is used when individual server threads (which are each connected to a different client) send non-username related messages to the central data server.

The WhiteboardMainServer is where the server is started, and this creates an object at the given port (default 4444, but can be changed in terminal by starting it with the –port [PORT] flag) that will listen for incoming client connections. Before beginning to block on the server socket, it creates a WhiteboardDataServer which will handle all data processing of the whiteboards.

The WhiteboardServerThread is the individual server thread that handles a client connection. Each WhiteboardServerThread is connected to a different client, and acts as the intermediary between the client and the data server. All of the server threads share a blocking queue with the data server, which will be used solely for sending messages from the client to the data server, as well as a personal blocking queue that will be used solely to receive packets from the data server. Since multiple blocks are required, the server thread also creates a helper thread that listens to the socket port and appropriately forwards information to the data server, while the original thread listens for messages from the data server on the personalized blocking queue and updates state accordingly. The original thread is also responsible for writing into the socket, giving messages back to the client.

Lastly, the WhiteboardDataServer is where all of the computation is done on the server. It contains the hash map of whiteboards and a hash map of users and the BlockingQueue needed to send a user a message. The data server receives messages off the shared blocking queue that all of the individual server threads share, and updates the state of the server version. For example, when a user joins or exits a whiteboard, the data server sends a messages to all other existing users notifying them of the change, and also sends the joining/exiting user any needed information, such as line segments already on the board or a list of users already editing the board. Since the data server is run on a single thread and communicates with the other server threads solely by blocking queues, the server can process data sequentially and minimizes the amount of shared data.

Client – *package* client

The client requires eight classes, many of which are used for the various GUI components that are necessary for user interaction. The class that the user uses the open the client is WhiteboardClientMain, which will create a LoginGUI so that the user can connect to a server. The class also creates a WhiteboardClient when the socket is created.

The WhiteboardClient is the class that handles the connection between the server and client. As the only class with direct access to the socket after logging in, all messages to and from the server must travel through the class. The WhiteboardClient is created after a successful connection is made, and immediately creates a helper thread that handles messages from the server. This helper thread also changes the user version of the whiteboard and performs appropriate tasks depending on the server message received. The class also provides all necessary methods to print to the socket to the server, which are called by the GUI components on the AWT Event Dispatch Thread. Lastly, the WhiteboardClient class contains a reference to the GUI displaying the list of whiteboards so that the client can properly tell the GUI to update its list of whiteboards. WhiteboardClient does not expect that window to ever close, so closing functionality on the whiteboard list GUI is disabled unless the user logs out.

The User class is described in Section 2, and simply represents the user and contains information about the whiteboards the user currently has open. The other five classes are used for GUI purposes – the Canvas class allows the user to see the whiteboard on the screen, and also handles the actual drawing of line segments on the screen. The Canvas will call the WhiteboardClient to send messages when lines are drawn. LoginGUI is the initial screen that users see and allow them to connect to the server. The Socket is made when the OK button is clicked – if the Socket fails to be made, the LoginGUI will not disappear and will prompt an error message. Usernames, due to the limitations of the protocol, cannot contain spaces. They also should not contain non-ASCII characters, as they will render oddly.

SimplePromptGUI is used in three situations – when a username is rejected by the server, when the user wants to create a whiteboard, and when the whiteboard name entered is rejected by the server. It consists of a simple text field, where the user will enter the username or whiteboard name. Clicking OK will then call the appropriate method in WhiteboardClient to send the data to the server. Like usernames, whiteboard names should not contain a space. WhiteboardGUI combines the Canvas (the whiteboard GUI) with a list of users that are currently editing the same whiteboard. Finally, WhiteboardListGUI is the main GUI that displays a list of whiteboards on the server and allows users to create and select whiteboards to view, as well as log out. Since this GUI is expected to be running for the entirety of the session, users are not able to close the window unless the Logout button is clicked, which will send a disconnect message to the server before closing. If the list GUI is forced to close abnormally (i.e. ⌘-Q on MacOS), the server will likely be unable to register that the client has disconnected. As such, the username that the user was using prior to the abnormal termination will be unavailable for use until the server is restarted.

6 CONCURRENCY

The design detailed in this document is focused on reducing the amount of shared data as much as possible between threads. As such, the primary strategy used is thread confinement. The server and clients have their own, separate versions of whiteboards, and the system is set up so that any changes on the clients need to be processed by the server before the clients see the change. Since all versions of the same whiteboard are separate, the server and users make their own changes to the whiteboard when signaled. As such, almost everything is confined to a thread.

On the server, there many threads, mostly to handle the client connections. Each client has two dedicated threads which share the same socket. One thread is used to listen to the socket and perform status changes, while the other thread listens to the blocking queue for messages from the server and only writes to the socket. The only thing they use together is the socket, and since the two threads use different parts of the socket (one exclusively uses the read side, while one exclusively uses the write side), there are no concurrency problems here. Each of the pair server threads are connected to the central data server by blocking queues, and every thread shares a BlockingQueue with the central data server. This utilizes message passing to send information from thread to thread, as each server thread can add messages onto the queue for the central data server to process sequentially. Since BlockingQueue is thread safe, we have no concurrency issues here as well. Since the central data server runs on a single thread, using a BlockingQueue will require the server to process information in a set order so that, for example, two users can never really draw on the same pixel at the same time – one will definitely occur before the other.

The client only has two threads running at the same time – one that first establishes the connection to the server and then continuously listens to the socket for server messages, and the AWT Event Dispatch Thread, where the many GUI functions are all handled. The main information that is shared across these two threads is the WhiteboardClient object, which provides the EDT thread with the ability to write messages to the socket. The GUI components only use the WhiteboardClient to write to the socket, while the other thread never writes to the socket, so since these operations are independent, there will be no concurrency problems. (Note, too, that when the GUI calls the socket writing methods, no changes are made to any other part of the WhiteboardClient object except for the write side of the socket.) Some methods in the GUI classes are called from the socket thread, but only serve to update the underlying models of the GUI and do

not actually modify the GUI. All other data shared between the threads are immutable (Strings) and do not have any beneficial mutations, so once again the client is thread-safe.

7 TESTING STRATEGY

Our automated tests were the following: we have a test file called UserTest.java for testing operations in the User class, a test file called WhiteboardTest.java for testing operations in the Whiteboard class, a test file called LineSegmentTest.java for testing operations in the LineSegment class, a test file called PacketTest.java for testing operations in the the Packet class, and a test file called MessagePassingTest.java for testing messages, defined by our protocol, between the WhiteboardDataServer and WhiteboardClient. The WhiteboardMainServer class is also tested by this test file (and TestUtil.java) since it runs the server and creates a socket.

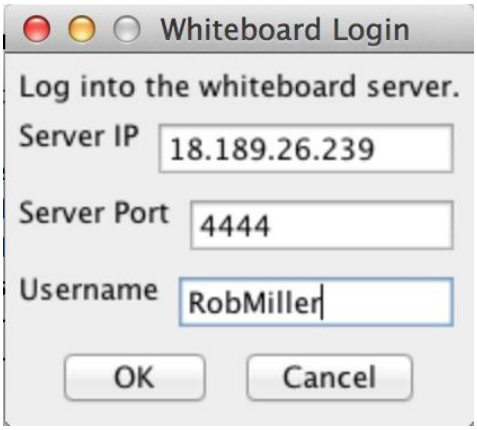
We also tested various sections manually. We tested the login feature by entering incorrect IPs and correct IPs. The incorrect IPs lead to a popup that tells the user that the connection is invalid. For correct IPs, we tried usernames that were already taken and new usernames. For a taken username, a popup will appear that tells the user to try another username. On that window, entering a taken username would lead back to the same window. New usernames in either of the windows will lead to a list of whiteboards on the server. We also tested the cancel buttons, which exit the program.

On the list of whiteboards, we tested adding a whiteboard and trying to add new whiteboards with the same name as another whiteboard. We tested hitting the select whiteboard button while having selected a row and hitting the select whiteboard button while not having selected a row. We tested the logout button. Adding a whiteboard with a name that already exists produces a popup that reprompts the user to pick a new whiteboard name. Hitting the select whiteboard button while not having selected a row does nothing, as expected. Hitting the select whiteboard button while having selected a row opens that whiteboard. The logout button closes the socket and disconnects the user. We also tested opening multiple whiteboards. If the user has already opened a whiteboard and tries to open a whiteboard that's already open, the user will get a popup message saying that the user has the whiteboard open already. Otherwise, the user will be able to open the whiteboard. We also tested that our scrollpane works by connecting a lot of users until the table was longer than the height of the frame.

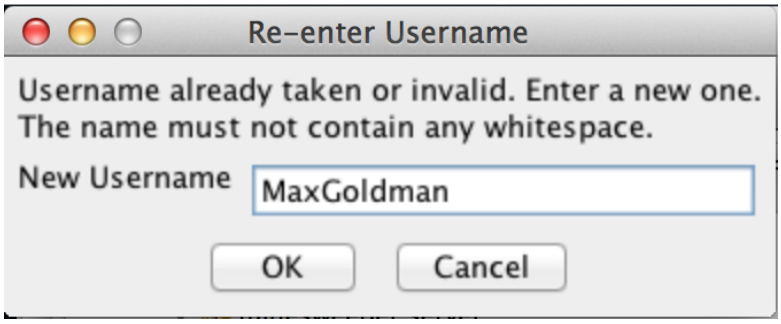
We manually tested the following components of our canvas: color picking, erasing, changing stroke size, and drawing. We started off with default settings and tested drawing strokes on the board. Then we clicked the color picker tool, selected a color, and drew more lines on the board. We repeated this five times for five different colors: white, red, green, blue, and black, selecting from the different tabs provided by Java's JColorChooser library. We then toggled the eraser button and tested erasing. We changed the stroke size to test different eraser sizes. Then we toggled off the eraser button to test different pen sizes.

For multiple users, we tested multiple users opening multiple whiteboards. We had three users each open three whiteboards. We were able to see when a user entered and when a user left, as well as all the users in the whiteboard at a given time. Each user picked a different pen color in each of the whiteboards. Then, we tested users drawing on the same whiteboard at the same time, over the same spot at the same time and over different spots at the same time. Then we tested users drawing on different whiteboards at the same time. We got expected behavior: each user's drawing appeared on his or her whiteboard, but not on whiteboards with different names. We also tested exiting and reentering a whiteboard. A user was able to see all of the contents redrawn on the whiteboard before the user could make any changes. In these tests we did not find any concurrency issues; all users who were looking at the same whiteboard saw the same images at the same time.

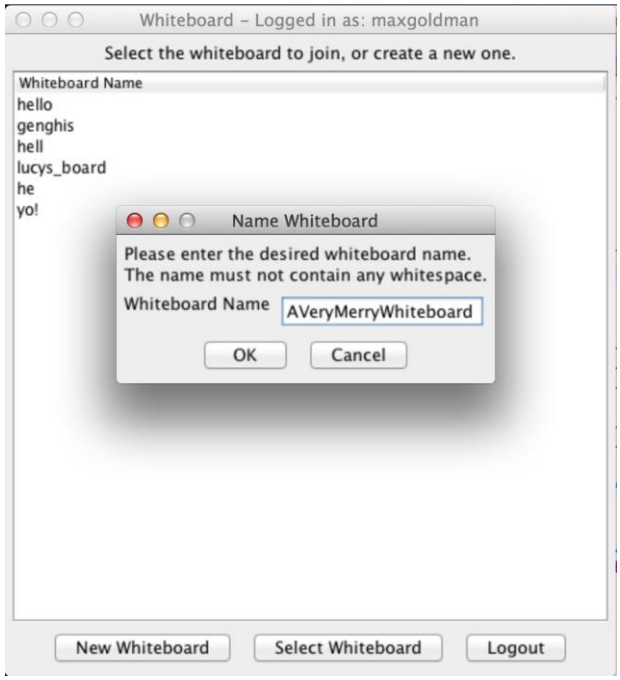
8 SCREENSHOTS



The login screen used for entering in details of the server.
The default port is 4444.



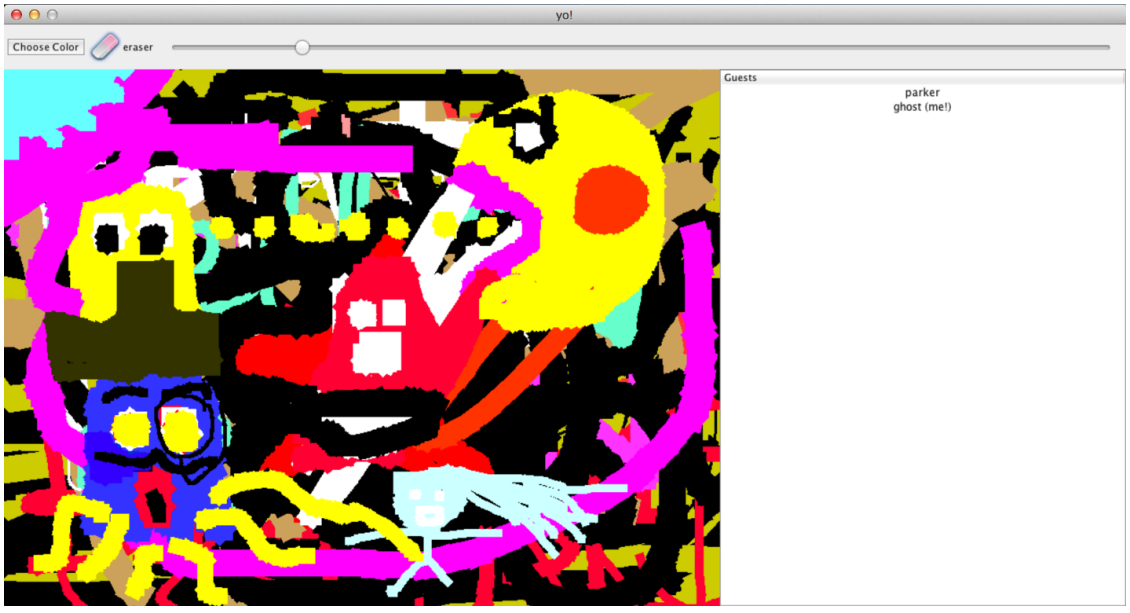
If a username is already taken on the server, another prompt will appear and ask the user to select a new username. Note the capitalization of the username.



After logging in, users are given the ability to create a new whiteboard.
Also visible is the main GUI, where users see a list of whiteboards and select the desired board. Note that the title displays the username, which is all lowercase.

After creating a whiteboard, it now appears on the list of whiteboards.
Note that it has been converted to all lowercases.

A view of the whiteboard in action.



Another example of the whiteboard in action.