Design Document
bschang, kellyliu, krwang

Datatype Design: We will be using a client server model

Client Side: We will be using a Model View Controller pattern. The Model will be the Canvas/ The View is the whiteboardGUI, and the controller is the whiteboardClient

WhiteboardClient: (controller)

- Spec: this class will receive updates from the Canvas and pass them on to the Server. It will then receive changes from the Server that will be reflected in the Whiteboard GUI
    - @param Whiteboard GUI: keeps a copy of the Whiteboard GUI so it has access to the Canvas and the current state of the tools panel
    - @param socket ID: stores the ID of the particular socket associated with this client
    - Note: each client can only refer to one whiteboard at a time
- Communication to the Server (send() message): WhiteboardClient processes drawing events from the GUI and sends them to the Server
    - The Canvas listens for actions from the user, and passes the actions to back to the Client for processing
    - The Client then updates its local image in the Canvas (via a draw() method), and sends this local update (using our brush action syntax) to the Server
    - The rest of the processing is described in the Server documentation
- Communication from the Server (receive() message): WhiteboardClient processes drawing events sent from the Server and updates local image
    - The WhiteboardClient receives an action from the Server with the parameters of the action (brush type, brush size, brush color, x1, x2, y1, y2)
    - The WhiteboardClient processes this action using the WhiteboardGUI's draw() method, which updates the Canvas image using the provided arguments
- Clarification: The WhiteboardClient that sends an action automatically updates its own snapshot locally, and then all other WhiteboardClients connected to the same Canvas receive an update message from the Server
    - This communication and processing will be threaded (anonymous threads)
    - The protocol used follows a loose publish-subscribe pattern between all clients connected to the same Canvas, where one WhiteboardClient publishes a message and all other connected WhiteboardClients are subscribers that receive the message after it is processed and sent out by the server

WhiteboardGUI:(view)

- public WhiteboardGUI(): constructs a WhiteboardGUI
- Note that User methods and instance variables will be altered later on to accommodate for multiplle users on same whiteboard--should also have a user instances
- this will also contain a switch button: if the user clicks it, they will be able to choose a new

whiteboard to open and then their current whiteboard will close

- Spec: this class combines the canvas and tools panel in one jFrame to present a unified appearance for the GUI
    - it will also keep track of the current users and display their usernames. the users will be stored in a Set, so there can't be repeat usernames
    - each client can only have one whiteboardGUI open at a time
    - each whiteboardGUI can only contain one unique canvas
    - @param Canvas
    - @param Toolspanel
- Communication between the GUI and the Canvas (View and Model): this will be done with Listeners of whatever mouse event is happening, and this action event is passed on to the Canvas

Canvas: does not deal with the gui (model)

- Purpose: this class represents the actual surface upon which the user can draw with their mouse. It also stores the image on the screen, which will be saved so that when the user reaccesses that specific instance of the canvas, the image will be saved with it
    - this image is a 2d array of pixels
- Spec: The Canvas class is the representation of the client's drawing
    - The Canvas class will contain a name for that canvas, and each name can only be used once (as in, a user cannot create multiple canvases with the same name because the name is used as the id)
    - @param Image: this is the image of the entire drawing on the canvas with changes from multiple users
    - @param username list: this is a list of the users currently using the whiteboard, which will be updated by passing the username to the server, which will then update all whiteboards
    - @param canvas name
- Clarification: The Canvas contains both the image and the listeners, but the two never interact with each other directly, only through the WhiteboardClient as an intermediary
    - In this sense, the Canvas acts as a sort of model because it stores the Image data, but it also acts as a view in its processing of user actions through listeners
- public Canvas(int width, int height): creates a canvas of a designated width and height
- public void paintComponent(Graphics g): the method in which the drawing buffer is actually drawn on the screen

Concurrency: The user's own changes to the canvas will be reflected on the user's own whiteboardGUI first (locally), and then other users' changes will be reflected a little bit later through updates from the Server

Server Side: We will be using a producer consumer model to create a Queue that stores all of the actions performed by all clients on various whiteboards. In this case, the client is producer, and the server is consumer

Spec: creates a serversocket to a specific port and then listens for the clients that are connecting to the server

- @param Hashmap that maps the name of each Whiteboard to its respective canvas that contains all of the drawn lines from all of the users working on the same whiteboard
    - Canvases must have unique names, so as to prevent any potential overwriting or duplicate entries in the hash map
- the inputs received are the actions performed on the client's whiteboard, which will be passed to the server's master copy of that particular canvas, which will be updated with the appropriate changes
- @param ArrayBlockingQueue: messages from the clients will be stored in this queue
- @param Hashmap mapping the name of the whiteboard to the corresponding socket

contains the following methods:

- serve(): runs the server and listens for the connection of clients to the server and handles their inputs by referring to the handleConnection method
    - we will be creating unique threads for each socket so that each client has one unique socket and won't overlap in data input and will thus be threadsafe
- handle(): reads client's input and puts it in a queue(the input will consists of the protocol, the socket name)
- handleRequest(): will handle run the methods that should occur for different protocol commands sent to it (open, bye, draw, erase)
- output(): sends the output of the handleRequest() back to the client

or should we show we show the user change first and then update it
just send the changes made to be faster
only send whole change on load and change--need full image
Note: *If it's a whiteboard that is being opened, we will send over the entire image saved to the master canvas*

If the whiteboard is already open, the we will just send the changes that have been made ot the file master canvas

ToolsPanel:

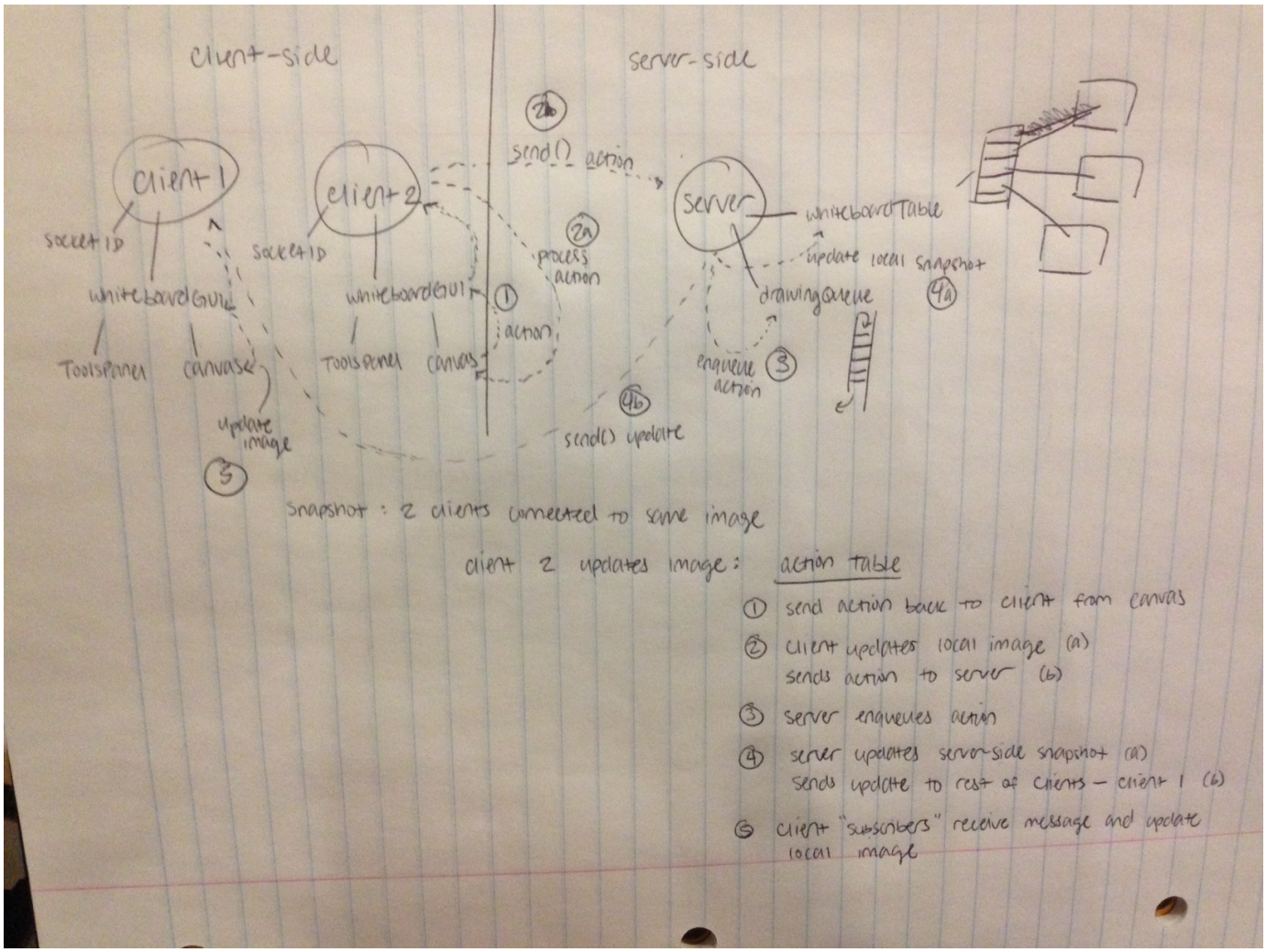
- Purpose: Allows users to customize their current drawing settings (brush type, brush size, brush color)
- Part of the overall GUI
- public ToolsPanel(): constructs a ToolsPanel
- All functions contained in action listeners, as the panel responds to user actions to customize their brush settings

EntryGUI:

- Purpose: GUI for users to specify whether they want to create a new Whiteboard on the server or load a previously created Whiteboard from the server
- When creating a new Whiteboard, the GUI prompts the user to enter a name for the canvas that the server will store the canvas under. The name for the canvas must be

unique to all other canvas names in the server
- Contains a definition and constructor for a NameGUI instance, which is the prompt for the user to input a name
- The user will also have to input a unique username that will be sent to the server as an indication of the number of people accessing each canvas
- public EntryGUI(): constructs a EntryGUI

Snapshot Diagram:



Protocol: designing how client and server interact

- the client will be passing information to the server by using a text based protocol, and we will be using telnet to test the protocol
- the client to server protocol
    - "open": sends request to server for the canvas with the specified name
        - open must be written as "open canvasName": the second word after this will be the whiteboard's name (which the client knows) so that the

whiteboard can be mapped to its name in the hashmap
        - if the whiteboard already exists, it opens the whiteboard
        - if it does not, then a new whiteboard is added to the server's hashmap under the specified name
    - "draw color size x1 x2 y1 y2 whiteboardname username" :this can only be written if the whiteboard has already been created.
        - draws from (x1,y1) to (x2, y2) with the specified color and size on the whiteboard with the same name as whiteboardname that is stored in the server's hashmap of whiteboards.
        - collects the username in a list attached to the whiteboard that will later be sent to each client
        - after receiving all of these commands, the server will then access its copy of that whiteboard in its hashmap and apply the changes to that copy of the whiteboard
    - "erase color size x1 x2 y1 y2 whiteboardname": this can only be written if the whiteboard has already been created.
        - erases from (x1,y1) to (x2, y2) with the specified color and size on the whiteboard with the same name as whiteboardname that is stored in the server's hashmap of whiteboards.
        - after receiving all of these commands, the server will then access its copy of that whiteboard in its hashmap and apply the changes to that copy of the whiteboard
    - "bye": terminates the connection with the server and removes the name of the user on that whiteboard so that the whiteboard gui on the remaining clients can later be updated
- Server to Client Protocol:
    - "draw color size x1 x2 y1 y2 whiteboard name"
    - "erase color size x1 x2 y1 y2 whiteboard name"
    - "users whiteboard name" will send a message for the list of users currently using the same whiteboard to then be output into the whiteboard GUI
    - "The client requests a pre-existing or new canvas instance from the server, and the WhiteboardGUI automatically sends updated canvas instances to the server after specific time intervals (autosave) and after all users exit a whiteboard "room"

Concurrency Strategy:

- We would like to have all users be able to draw on whatever whiteboard they choose concurrently
- There cannot be any deadlocks because users are not dependent on other users, there is no dependency between classes
- Another concurrency problem would occur if two users are trying to draw on the same whiteboard at the same time because the users could have different settings with which they're drawing
    - we will solve this by determining the order of the actions by which one reaches the server action queue first.
- We will employ a client-server design implementation along with a confinement strategy to

prevent multi-user access to anything but the shared Canvas object, which will in turn ensure thread-safe behavior for all datatypes aside from the Canvas class (the thread safety argument for the Canvas class is described above)

- essentially, each whiteboard client will first update its on whiteboard GUI with the changes that its client made, and then the additional changes from other clients will be made a little later in order in which they were received in the server queue
- We will be updating the whiteboards instantaneously to prevent merge conflicts between the user's initially different versions of the same whiteboard

Testing Strategy

- **Helper Classes:** will test each method with JUnit tests
- **we will be testing the model, networking, and gui**
- **model:** to test the model, we will be testing the Canvas class:
    - test that the strokes drawn by the user are accurate on the Canvas's image by comparing the location of the stroke with the location on the Canvas
    - test that the model is named correctly according to the user's input for the whiteboard name
    - test that the username list contains all users currently using that canvas
    - test that the correct image is drawn when the listeners are invoked for action events on the canvas
- networking: to test the networking, we will test several parts:
    - testing the protocol: we will test that all the commands result in the right text commands for the protocol
        - we will also test that all text commands result in the right action according to the protocol
    - testing that the server's canvas image is identical to that of the different clients that are all working on the same canvas whenever different clients draw on the same whiteboard
    - testing the sending of a draw message
    - testing the sending of an erase message
    - test the sending of a bye message
    - test the sending of an open message--if the whiteboard is already created, or hasn't been created yet
    - **Server:** Using JUnit tests we will test all messages that can be passed to the server and check that the server returns an appropriate response.
        - test opening/closing/loading a Whiteboard
        - test opening/closing/loading multiple Whiteboards (make sure the server keeps track of which edits are performed on which Whiteboards)
        - test one user editing a Whiteboard
        - test multiple users editing one Whiteboard (checking for race conditions?)

        - testing having up to 3 users editing one Whiteboard
        - test accessing a whiteboard that has already been created in the past
    - test when opening and closing same whiteboard that the image is still saved there

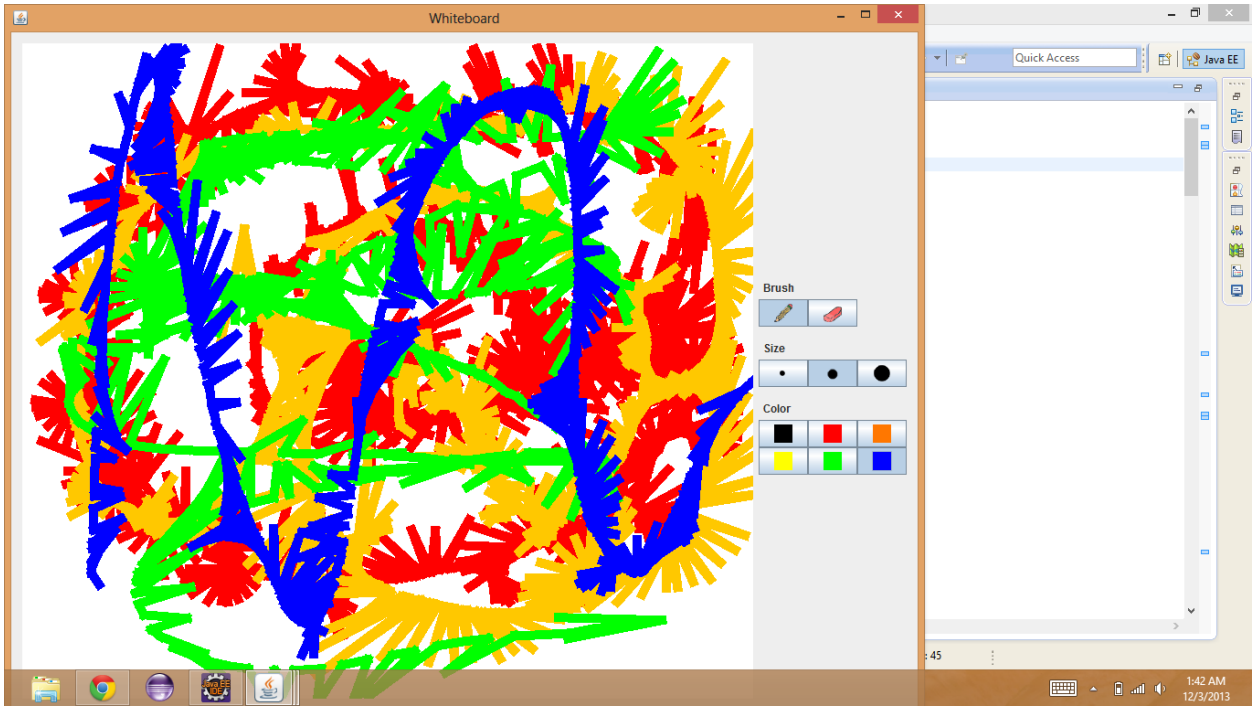
Test whole project: document the possible interactions with the GUI and use println to display what the GUI reads as the action and the response. We will run the GUI and tests each possibility.

- test one user one whiteboard--test every type of brush, every color, every size brush
- test one user multiple whiteboards: drawing on multiple whiteboards to make sure drawing are accurate
- test multiple user same board: drawing at the same time, drawing at different times, drawing at the same location at the same time, and using different brushes/colors/sizes all of these times
- test multiple users multiple whiteboards
- test someone drawing on a Whiteboard and someone erasing from the same spot on the same Whiteboard at the same time (indeterminate result???)

We are essentially trying to test every single aspect of our project, so this amount of effort should be enough. We are also testing all potential problems with each aspect of our project.

Testing GUI:

test location of drawing
test length of drawing
test erase function
test color change
test brush stroke size change
test drawing a line segment


Whiteboard
Brush
Size
Color
Quick Access
Java EE
1:42 AM
12/3/2013