William Haack
Shawn Jain
Blake Elias
6.005 Whiteboard design.

We follow a client-server pattern. The server is multithreaded.

DataType:

We define a state as a image buffer. When a client connects to a server, they receive the whole state.

Our application will also have a Queue<List<Command>> called updates.
Each list in the Queue is a list of commands for a given timestamp. At each timestamp interval (~50-100ms) we will send all of the commands in the next list in the queue to the server, and the server will update the main state with this list of commands and then return back the new state and the timestamp that this state was constructed at.

Protocol:
The server and the client interact with a few methods. Clients send draw(x0, y0, x1, y1, color, boardID) to the server to update it on - this represents a command to draw a segment between two points (x0, y0) and (x1, y1) with color "color" on the board indicated by boardID.

The grammar for the command is:
MSG : timestamp x0 y0 x1 y1 color boardID
timestamp: NUMBER+
x0: NUMBER+
y0: NUMBER+
x1: NUMBER+
y1: NUMBER+
boardID: NUMBER+
color: "RED" | "WHITE" | "BLUE" | "GREEN" | "ORANGE" | "PURPLE"
NUMBER: 0-9*

Concurrency Strategy:

When a user draws on his canvas, the client does the following:
1. instantly perform command on the user's local image buffer
2. save the command in a list ("commands") to be sent to the server.

Every 100 milliseconds (or some other constant time step), the client will:
1. send the current list "commands" to the server, in an asynchronous request. The server's reply will be an updated image where those commands have been drawn.

As the server receives lists of draw commands from multiple clients, it will apply the draw commands in the order it receives them.

2. append the list “commands” to a queue of sent commands
3. replace “commands” with an empty list

The server responds with an updated image, and a timestamp indicating which time step this is a response for (i.e. which draw commands it has incorporated into the master copy, as the responses will be delayed from the original time the requests were sent). Upon receiving a response, the client does the following:

1. Make a new image buffer with a copy of the server’s response
2. Remove from the queue the commands at the time step corresponding to this response
3. Take any commands still in the queue, and apply them to this new image buffer. This allows the user to see their local changes merged with the server response, even if the server has not yet received their most recent local changes.

Here is a diagram of our Queue<List<Commands>>

An element in the list looks like: timestamp,   x0, x1, y0, y1, color, boardID

Queue<List<Commands>>:

```
__
[0 1 2 3 4 blue 123,
0 5 6 7 8 red 123]
__
[1  2 4 6 9 green 123,
1  2 3 4 5 blue 123]
__
[2 1 3 5 7 orange 123]
```

Processing server responses:
r = “0: <image including all updates up to t=0>”

Take r + local queue updates 1 and 2 -> draw to screen

Response:
r = “1: <image>”

Show r + local update 2 from queue -> screen

Testing Strategy:

We will unit tests all components of our code.

In order to test our code for concurrency issues we will make a series of fake commands at given timestamps and test to ensure that we have the right state by checking our image buffer.

We will also manually test our application to ensure that there are no visual bugs that are hard to detect from written tests.