Damon Doucet, Harini Kannan and Kulpreet Chilana
6.005 Software Construction
December 11, 2013

Final Design Documentation

Whiteboard Project

Summary of Changes

This design document has been updated to reflect the final design of our whiteboard project. It has been rewritten to be more concise and give a better overview of the architecture and design of the whiteboard server and client.

We made a few changes to our design, namely making it more detailed. Our Client architecture now has three components: A ClientSocket, various RequestHandlers (DrawingRequestHandler, LoginRequestHandler, MenuRequestHandler), and various Delegates (DrawingDelegate, LoginDelegate, MenuDelegate). The ClientSocket class handles both getting messages from the server and sending messages to the server.

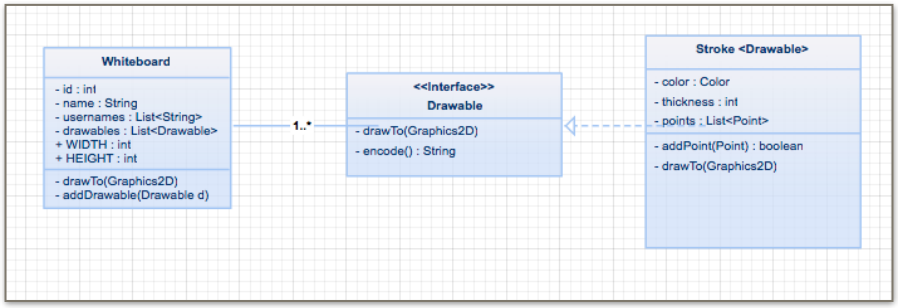
In the grammar, we have changed Color to be an integer, and added more colors than just black and white. We've also improved our GUI tests to be more robust and account for more edge cases.

Datatype Design

- **Drawable Interface:**
  - public void drawTo(Graphics2D graphics);
    - Uses graphics object to draw the stroke in the whiteboard window.
  - public String encode();
    - Returns String representation of Drawable for passing between server and client.
- **Whiteboard:** Whiteboard is a datatype representing a drawing board. It has a unique ID, name, a list of users connected to the board, and a list of Drawable components on the board.
  - Properties
    - private final int id;
    - private final String name;

- private final List<String> usernames;
- private final List<Drawable> components;

- Constants
  - public static final int WIDTH = 800;
  - public static final int HEIGHT = 600;
- Methods
  - public void drawTo(Graphics2D graphics)
    - calls drawTo on the list of Drawables with the same Graphics2D object, and draws itself and each of its components in turn.
  - public void signInUser(String username)
    - Add username to the list of users being maintained by the Whiteboard
  - public void signOffUser(String username)
    - Removes username from the list of users being maintained by the Whiteboard
  - public int getId()
    - Returns ID of whiteboard.
  - public String getName()
    - Returns name of whiteboard.
  - public List<String> getUsers()
    - Returns list of users currently on whiteboard.
  - public List<Drawable> getDrawables()
    - Returns list of Drawables currently on whiteboard.
  - public void addDrawable(Drawable d)
    - Adds a Drawable to the list of Drawables stored by the whiteboard.



- **Stroke** IMPLEMENTS **Drawable**

- Properties
  - private Color strokeColor;
  - private Integer strokeThickness;
  - private final ArrayList<Point> points;
- Methods
  - public boolean equals(Object rhs)
    - Returns true iff this equals rhs
  - public boolean equals(Stroke rhs)
    - Helper method for previous method. Returns true iff this equals rhs
  - public int hashCode()
    - Returns hashcode for instance of this
  - public void drawTo(Graphics2D graphics)
    - Draws the set of lines naturally defined by the Stroke's set of points with the specified color and thickness.
  - public void addPoint(int x, int y)
    - Adds a point to the list of points comprising the stroke.
  - public void addPoint(Point point)
    - Adds a point to the list of points comprising the stroke.
  - public String encode()
    - Returns String representation of the Stroke.
  - private String encodeColor()
    - Returns String representation of the Color.
  - private String encodePoint(Point point)
    - Returns String representation of the Point.

DrawableParser: DrawableParser is a helper class whose purpose is to parse Server messages into Stroke objects.

Properties: none.

Methods

- public Drawable parse(String[] request)

    - Requires that request should be in all lowercase and have at least two elements.
    - Returns Drawable object corresponding to server message.
- private Stroke parseStroke(String[] request)
    - Requires that request should be in all lowercase and have at least two elements.
    - Returns Stroke object corresponding to server message.
- private Color parseColor(String string)
    - Returns Color object associated with the String name of the color.
- private Point parsePoint(String string)
    - Returns Point object associated with the String representation of (x,y) coordinates.
- private Integer tryParse(String string)
    - Returns Integer object parsed from a String representing an x or y coordinate.

Networking Protocol

Actions

- **Login**
  - Client must input a unique nickname
- **Menu**
  - Client should know when a new whiteboard is created
    - Client should know the name of all the connected users
    - Whiteboards are never deleted
  - Client can join a whiteboard
  - Client can create a whiteboard with a given name
    - Names are not necessarily unique
- **Drawing**
  - Client can draw on a board
  - Client can return to the menu
  - Client should know when people join/leave a board

Grammar

```
General
      NICKNAME := [a-zA-Z0-9]{2,5}
      INT := [1-9][0-9]*
      NEWLINE := "\r"?  "\n"
      BOARD := BOARD_ID "-" BOARD_NAME
            BOARD_ID := INT
            BOARD_NAME := [a-zA-Z0-9][a-zA-Z0-9 ]{2,10}
      drawing_action := "DRAW" stroke NEWLINE
            stroke := "STROKE" COLOR THICKNESS POINT{2,}
            THICKNESS := INT
            COLOR := INT
                  the RGB of the color
            POINT := INT "," INT

Client to Server
      Login
            nickname := "NICK" NICKNAME NEWLINE
      Menu
            menu_action := [ make | join ]
            make := "MAKE" BOARD_NAME NEWLINE
            join := "JOIN" BOARD_ID NEWLINE
      Drawing
            on_draw := drawing_action
            leave := "LEAVE" NEWLINE
      Exit
            bye := "BYE" NEWLINE
```

```
Server to Client
      Login
            specify_nick := “SPECIFYNICK” NEWLINE
            nick_in_use := “NICKINUSE” NEWLINE
            nick_ok := “NICKOK” menu_list
      Menu
            menu_list := “MENU” BOARD* NEWLINE
            board_created := “NEW” BOARD NEWLINE
            id_not_found := “BADID” NEWLINE
      Drawing
            on_join
                  sends  a list of on_friend_draw and on_friend_join messages
            on_leave := menu_list
            on_friend_join := “LEAVE” NICKNAME NEWLINE
            on_friend_leave := “JOIN” NICKNAME NEWLINE
            on_friend_draw := drawing_action
```

Concurrency Strategy

Client

- GUI and Socket thread won't interact directly. The socket thread handles reading and writing to the socket and has a callback using the delegate pattern to the GUI thread for drawing and other messages from the server.
    - The socket has a threadsafe “send message” function for the client to call
        - Adds the message to a threadsafe queue that gets polled on the socket thread
    - The callback on the GUI adds the drawn stroke to the whiteboard and asks swing to repaint the GUI on its own thread using repaint()

Guarantee about concurrent edits: If two users draw at the same spot at the same time, the stroke that was begun earlier will be displayed below the stroke that was begun later once both strokes are finished. This ordering will be the same on both users whiteboards. The "beginning" of a stroke is defined to be when the user first clicks on the whiteboard, and the "end" of a stroke is defined to be when the user releases the mouse.

Server

- Separate thread for each client, a thread to accept clients, and a single thread for the server state
- The MessageBus and queueing mechanism allow all of these to communicate thread-safely
    - The MessageBus pushes messages onto thread-safe queues which the respective threads read
    - See comments in classes for details of server concurrency strategy
    - See SERVER_ARCHITECTURE.txt for complete explanation of server classes

Testing Strategy

Domain

- Models
    - We will use unit tests to test every single testable public method of our datatypes.
    - In the Stroke class, we will test drawTo, addPoint, encode, encodeColor, and encodePoint.
        - **Note that drawTo will be tested manually with the GUI
- More details about Encoding/Decoding test case partitions
    - Decoder
        - Valid inputs
            - Two points
            - Lots of points
            - Black and white

- Varying thicknesses
- Invalid inputs
  - Missing parameters
  - Not enough points in stroke
  - Bad colors
  - Floating point thickness
  - Point missing a parameter
- Encoder
  - Decode(Encode(x)) == x
  - Several general cases

Client

- The Client code will be decoupled into separate GUI objects for each type of window and the networking client, allowing us to test each component separately.

- GUI
  - The GUI will be tested manually for the following test case partitions:
    - LoginGUI Testing Strategy
      1. Test that lack of a server or a failed server connection should pop an error dialog
      2. Test that hitting the cancel button exits the program
      3. Test that typing in a nickname that already exists on the server prompts for a different nickname
      4. Test that typing in a unique nickname shows the MenuGUI
    - MenuGUI Testing Strategy
      5. Test that the menu displays a list of boards consistant with what is on the server
      6. Test that clicking NEW BOARD and then CANCEL returns the menu to focus
      7. Test that clicking NEW BOARD and then entering an empty string prompts the user to name the board
      8. Test that clicking NEW BOARD and then entering a string shows the WhiteboardGUI
      9. Test that clicking JOIN BOARD when there are no server boards prompts the user to create a new board
      10. Test that clicking JOIN BOARD when a board is selected shows the WhiteboardGUI
      11. Test that adding a new board in one client updates the other client with this change
    - WhiteboardGUI Testing Strategy
      12. Test that closing the WhiteboardGUI shows the MenuGUI
      13. Test that the WhiteboardGUI's title has the nickname of the current user
      14. Test that the users currently editing the board is consistent with the list of names in the user list
      15. Test that having another cent join the board updates the user list
      16. Test that having another client leave the board updates the user list
      17. For all colors, test that picking that color and then drawing, draws in that color
      18. Test that moving the thickness slider, affects how thick of a line is drawn
      19. Ensure multiple clients connected to the same board have consistent drawings
      20. Test that joining a board shows all previously drawn strokes
      21. Test that leaving and returning to a board leaves the user where they left off
      22. Test that leaving and returning to another board doesn't draw to the previous boards server

23. Test that starting to draw on one board and getting an update before finishing keeps boards consistent
24. Test that drawing out of bounds updates on both boards

- Networking
  - The main functionality to ensure is that the client can correctly parse the incoming messages from the server using the grammar.
  - We have written unit tests for the following test case partitions in the grammar: login, menu, drawing, and exit (see Grammar section for more details)
  - We will ensure the messages are parsed as they should be according to the grammar

Server

- MessageBus
  - Test publish/subscribe
  - Test hasKey for whiteboards + clients
- ClientHandler State Machine
  - Test state transitions by feeding input and watching state of the given ClientHandler
  - Test invalid input
- Server State
  - The methods here are too trivial to warrant unit tests. Any issues will be detected with manual inspection and end-to-end tests

End-To-End Tests

- Run the server
- Ensure multiple clients can connect
- Ensure the same nickname is not supported
- Ensure that creating a board on one shows for the others
- Ensure that joining/leaving a board updates connected users for both
- Ensure that drawing on the board updates for the other
- Ensure that joining the board after drawing has happened will show correct picture
- Ensure that creating another board is independent in both connected users and drawing of the other
- Ensure that everyone disconnecting from the server and then rejoining still has the same whiteboard

Incidentally, several of these end-to-end tests can (and should) also be written as unit tests.