Week 10 Tutorial - Multi-user interaction

This tutorial will explore **multiple users** can have a shared web experience. We use a "chat room" model of communication, where anyone who navigates to our site will exchange messages with all others.

In the previous video lecture, we did a "talk through code along" to develop a basic text chat system where each client could post and receive text messages that were posted to a text area on the browser with the user name of the message sender identified.

In this tutorial, we will start from the text chat code you developed and extend it to include a shared drawing space (If for some reason you don't have it handy, ~~your doomed~~ you can use mine http://nm2207.org/creativeweb/tutorials/Chatn_Draw_0.zip). This tutorial will demonstrate how powerful the simple communications system can be for creating interactive applications.

Challenges

1. You'll recall that in our chat system, we created a message object to send info to the server to be blasted back out to all connected clients. We created a ‘data’ property (or attribute) to hold the actual message text. But what if we wanted to use the "data" field in our message object to carry different kinds of messages, such as instructions for drawing something on a canvas? In that case, we would need a **new** property to tell the message receivers how to interpret the data - is it text to be displayed in a chat window, or (for example) a "path" to be interpreted as graphics to draw?
   - Create a new property on the object literal you are now sending for this purpose - call it ‘datatype’. Decide on a value that you want to use for identifying the datatype as chat text (e.g. the string ‘chatText’). Add this property/value pair to the messages you are currently sending which come from the typing box.
   - On the receiving end, check to make sure the mtype is "chatText" (or whatever you called it) before printing it to the Chat Box, and just print out a console message if the mtype is something else
   - Check your work! (Run the server and two client browser windows)

The next thing that we'll do is allow the user to draw. Here is a "high level" description of the challenge (smaller steps are below if you need them). Using Raphael, track the mouse in the drawing area so that when it is pushed and dragged, a line is drawn on the screen.

Think through the problem. The basic idea is to create a variable to hold a pathString (with the svg commands such as M and L and coordinates). Then you'll listen for mousedown, mousemove, and mouseup commands, creating a Raphael path and initializing the pathSting with mousedown, extending the pathString with mousemoves, and then setting the raphael path "path" attr with pathString when the mouse goes up. As usual, you'll need to keep track of whether the mouse is down or up. Got it? See if you can do it before reading more detailed hints!

OK, in more detail if necessary:

2. Notice there is already a div with the id="svgcanvas" on the html html file. Grab it from the DOM and assign it to a variable (let's call it svgdiv)
3. Create a new Raphael paper:
   - var paper = new Raphael(svgcanvas);

4. create variables:
   var raphaelPath; // for holding the
   raphael path var pathString; // for
   holding the path string
   var mousePushed=false; // for remembering the state of the mouse.
5. Create 3 listeners on the svgDiv: one for mousedown, one for mouseup, and one for mousemove remembering that your callback functions take a mouse event argument.
6. In the mouse down, initialize the pathString with the svg "moveto" (M) symbol followed by the comma-separated coordinates (offsetX and offsetY) you get from the mouse event passed in to your listener. Then use the Raphael paper to create a path (passing in the pathString as an argument) and assign the results to the variable you already created called raphaelPath. Give the path a stroke and a stroke-width, too. Store the state of the mouse.
7. In the mousemove listener, if the mouse is pushed, extend the pathString with the SVG symbol for "lineto" (L) followed by the x and y coordinates of the mouse that you can get from the event passed in to your listener. Then update the "path" attr of you raphaelPath.
8. In the mouseup listener, extend the pathString again with the mouse position, update the "path" attr of raphaelPath with your pathString. Don't forget to update your mousePushed variable. Now you have the complete path drawn and stored in a variable.

Now turn your lone artist canvas into a shared canvas for all. See if you can think of the necessary steps before reading the following steps. The basic idea is that you will send the pathString just as you did the chat text using the ‘data’ attribute in a message object. You will use a new value for the ‘datatype’ attribute that will indicate that the data is a path for drawing rather than text for typing.

9. When a path is complete (where in your code would that be?), call the iosocket send method (just like we did to send our chat text) with an object that has its data attribute set to be your pathString. Make sure to change the datatype to something so that when you receive such a message, you know what to do with it (make up a value you will use for this purpose).
10. On the receiving end, check for your datatype, and if you are getting a path string for data, use it to create a Raphael path and draw it!

Bonus coolness: OK, so you got to this point and you aching for more? Overview of this section: Add controls for the path attributes (color, stroke-width, transparency, fill options, whatever). We'll start with color. Use these for drawing locally, but also send this data to others in your message.

11. Put a 'clear' button in your 'aside' pane that clears the paper of the drawing.
12. Put 3 sliders in the 'aside' pane on your html page. Give them a range of [0,1] and a step of .05. This is the range Raphael likes for hsl values.
13. Create variables for your sliders in your JavaScript code, and grab the sliders from the DOM. (No need to add event listeners to them - we'll just grab their values when we need them!)
14. In your mousedown listener, create a colorString variable and initialize it to the hsl string using the slider values: var colorString = "hsl(" + hSlider.value + "," + sSlider.value + "," + lSlider.value + ")";
    Use that colorString to set the 'stroke' attribute of the path.
15. In your mouseup listener, also create colorString as above. Send it with the object so that others will know how to color the strokes they receive. Remember, you can create the data structure you send however you like, as long as you know what to do with it on the receiving end!
16. Create another slider for stroke-width. Use if for your local path drawing, and send it to others, too.

What you send/receive between your participants is up to your imagination. **This kind of messaging to control graphics is the foundations of multi-player games!**