**Collaborative Drawing**

*Specification Document*

by Casey Hofford and Soeren Walls

In this simultaneous collaborative drawing application, the “canvas” is simply a table of cells, and each cell has a size of 3x3 pixels, with a background color property. The client detects mouse clicks/drags to appropriately assign the current Pen Color to each affected cell, and possibly to the cells around each clicked cell depending on the current Brush Size. The client sends an array of **only** changed cells (as opposed to all cells in the table) to the server when cells’ colors are changed, and also continuously “polls” the server for changes made by other clients. Unlike the client, the server sends the **entire** canvas (4,000 cells) back to each client during each “poll”.

There are essentially an infinite number of events sequences that can take place in this program. Here are some of the primary ones this program is built to handle:

1. **Ideal Scenario**
   1. (client) Single user clicks on a cell.
   2. (client) Client sends changed cells to server.
   3. (server) Server receives updated cells from client, updates server canvas array.
   4. (client) Client polls server for updated cells.
   5. (server) Server receives poll request and sends entire canvas to client.
   6. (client) Client receives canvas, but makes no changes, since all data matches.
2. **Dragging Scenario 1**
   1. (client) Single user clicks and drags the mouse around the canvas.
   2. (client) Client flags all affected cells as “clicked”.
   3. (client) Client looks for changed cells, finds none.
   4. (client) **Repeat steps 2.2 and 2.3 any number of times.**
   5. (client) User ends dragging motion and client detects “mouseup” event.
   6. (client) Client flags all affected cells as “changed”.
   7. (client) Client looks for changed cells, sends changed cells to server.
   8. (server) Server receives updated cells from client, updates server canvas array.
   9. (client) Client polls server for updated cells.
   10. (server) Server receives poll request and sends entire canvas to client.
   11. (client) Client receives canvas, but makes no changes, since all data matches.
3. **Dragging Scenario 2**
   1. (client) Client polls server for updated cells.
   2. (client) Single user clicks and drags the mouse around the canvas.
   3. (client) Client flags all affected cells as “clicked”.
   4. (client) Client looks for changed cells, finds none.
   5. (client) **Repeat steps 2.2 and 2.3 any number of times.**
   6. (server) Server receives poll request and sends entire canvas to client.
   7. (client) Client receives canvas and attempts to make changes. For any cells updated on the client more recently than the poll request was made (including cells affected by the user’s current clicking motion), the client temporarily ignores any changes made by the server. All other cells are changed accordingly.
   8. (client) User ends dragging motion and client detects “mouseup” event.
   9. (client) Client flags all affected cells as “changed”.
   10. (client) Client looks for changed cells, sends changed cells to server.
   11. (server) Server receives updated cells from client, updates server canvas array.
   12. (client) Client polls server for updated cells.
   13. (server) Server receives poll request and sends entire canvas to client.
   14. (client) Client receives canvas, but makes no changes, since all data matches.
4. **Two Users Scenario**
   1. (client) Bob clicks and drags the mouse around the canvas.
   2. (client) Bob’s client flags all affected cells as “clicked”.
   3. (client) Bob’s client looks for changed cells, finds none.
   4. (client) **Repeat steps 2.2 and 2.3 any number of times.**
   5. (client) Bob ends dragging motion and Bob’s client detects “mouseup” event.
   6. (client) Bob’s client flags all affected cells as “changed”.
   7. (client) Bob’s client looks for changed cells, sends changed cells to server.
   8. (client) Stewie clicks and drags the mouse around the canvas.
   9. (client) Stewie’s client flags all affected cells as “clicked”.
   10. (client) Stewie’s client looks for changed cells, finds none.
   11. (client) **Repeat steps 2.2 and 2.3 any number of times.**
   12. (client) Stewie ends dragging motion and Stewie’s client detects “mouseup” event.
   13. (client) Stewie’s client flags all affected cells as “changed”.
   14. (client) Stewie’s client looks for changed cells, sends changed cells to server.
   15. (server) Server receives updated cells from Bob’s client, updates server canvas array.
   16. (server) Server receives updated cells from Stewie’s client, **some of which are the same cells Bob changed**, then updates server canvas array.
   17. (client) Bob’s client polls server for updated cells.
   18. (server) Server receives poll request and sends entire canvas to Bob’s client.
   19. (client) Bob’s client receives canvas, and makes **all** changes Stewie made that don’t match Bob’s canvas.
   20. (client) Stewie’s client polls server for updated cells.
   21. (server) Server receives poll request and sends entire canvas to Stewie’s client.
   22. (client) Stewie’s client receives canvas, and makes **all** changes Bob made that don’t match Stewie’s canvas.

**Running the Program**

The **server** is initialized and sets serverFn to listen to url requests from the client. The first request will open index.html which runs the **client-side** load function. This function creates all of the html elements, including the table containing all the colored cells, and the selectors for brush color and brush size. Then, event listeners are added to the *CanvasOverlay*, which is a div overlaying the table that detects all clicks and drags of the mouse. The location of the mouse on the *CanvasOverlay* during a mouse click or drag is used to calculate which cell(s) in the table to change. During a mouse click or drag, all affected cells are pushed onto the *clickedCells* array. When the user terminates the event with a “*mouseup*,” all of these affected cells are transferred in bulk from the *clickedCells* array into the *changedCells* array.

The *sendColors* function is called every 100ms; it checks for elements in the *changedCells* array, and if found, this function pushes them to the server. On the **server side**, the function *getCellsFromURL* updates the server-side *canvas* array. This is simply a multidimensional array that stores the color of each cell on the “global” canvas. The **client** calls the *pollColors* function every 500ms in order to resolve any conflicts between its local canvas and the server’s canvas.

The server calls the *sendChangedCells* function upon receiving an XMLHttpRequest from a client that has just called *pollColors*. The server is consequently responsible for sending the global *canvas* array to ALL clients, who can parse the server’s array using the *colorListener* function.

Similarly, the *clearCanvas* function sends an XMLHttpRequest to the server, which subsequently resets the *canvas* array. The next time any client polls the server, the entire canvas on the client will be erased.