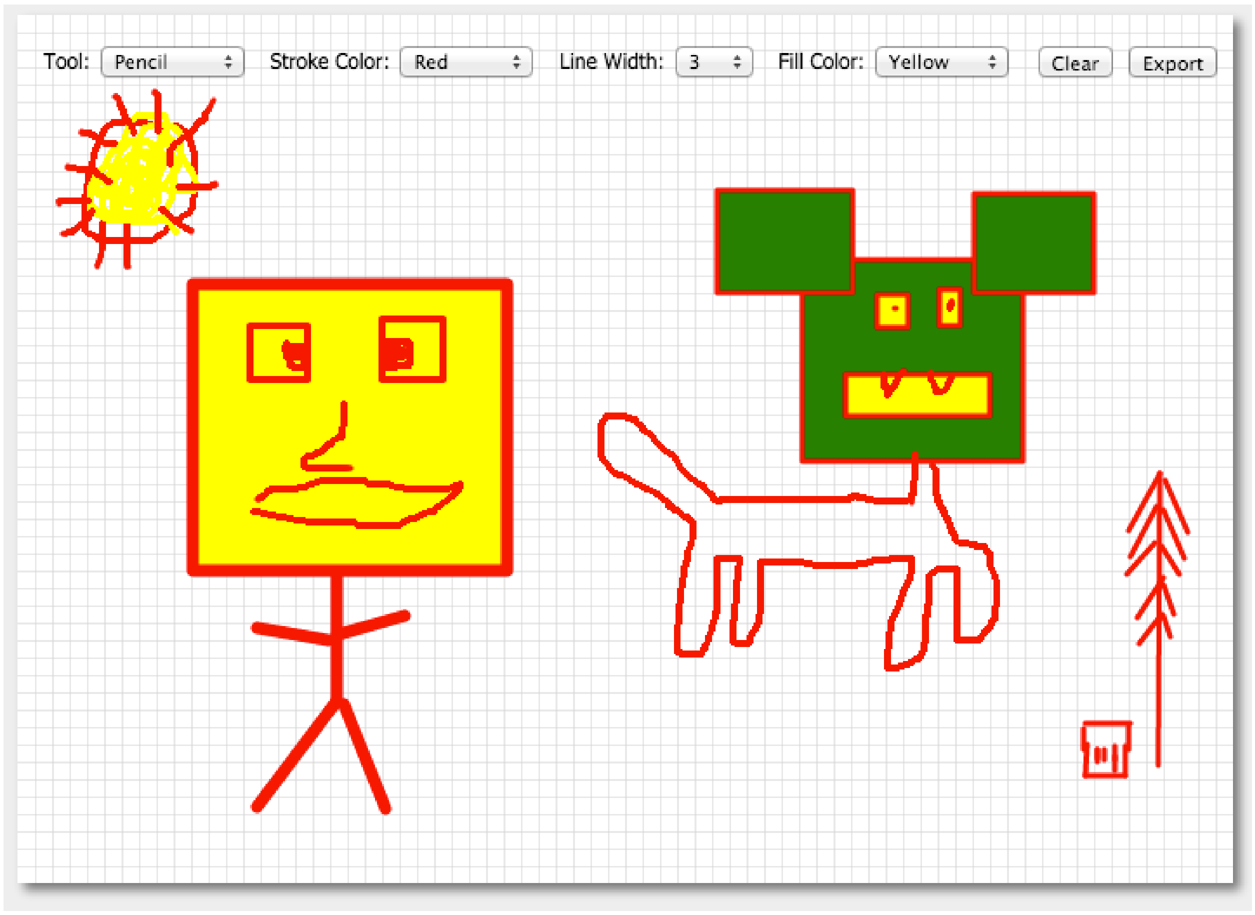
Canvas Painting App Part II

In this part of the exercise we are going to continue working on our canvas painting app. What we'll add today:

- Controls for **Rectangle** and **Line** tools

- Controls for fill color

- Recall that the **Rectangle** and **Line** tools will allow the user to preview their drawing before it is committed to the canvas, and the operation will be cancelled if the user drags the pointer out of the canvas window.


Tool: Pencil Stroke Color: Red Line Width: 3 Fill Color: Yellow

Part A - Laying the foundation

Before we continue, we first need to add some HTML to our document.

1) First add two more tools to the `#toolChooser <select>`:

```
<option value="toolRectangle">Rectangle</option>
<option value="toolLine">Line</option>
```

2) Create a `#fillStyleChooser <select>` – which will be very similar to your `#strokeStyleChooser` you did last time. If you copy/paste, don't forget to change the `id` !

3) Create another `<canvas>` element with an `id` of `#topCanvas`. It's identical to `#mainCanvas` except for the id - the HTML looks like this:

```
<canvas id="topCanvas" width="700" height="500">
     Get a real browser!
</canvas>
```

What's up with a second canvas? We're going to find we need it as a temporary drawing layer when we start drawing rectangles and lines.

If you reload the page, the `#topCanvas` isn't really visible yet (only a pixel or so) because the `#mainCanvas` is sitting on top of it. The app should still work fine though.

For right now `#topCanvas` isn't doing anything, so let's move on.

Part B - Start Coding

** Reminder! Don't forget to periodically reload the page and test the app to catch those inevitable syntax errors early. **

1) Add 4 new global constants in the CONSTANTS section

```
var DEFAULT_FILL_STYLE = "blue";
var TOOL_PENCIL = "toolPencil";
var TOOL_RECTANGLE = "toolRectangle";
var TOOL_LINE = "toolLine";
```

2) Add 3 new global variables to the GLOBALS section

```
currentTool
fillStyle // used by rectangle tool
origin // used by rectangle and line tools
```

3) In the *// initialize some* `globals` section of `init(),` add the following code:

```
fillStyle = DEFAULT_FILL_STYLE;
currentTool = TOOL_PENCIL;
origin = {}; // empty object
```

4) Let's enable our tool choosing `<select>`. In the `// hook up event listeners` section of `init()`, add the following code:

```
document.querySelector('#toolChooser').onchange = function(e){
      currentTool = e.target.value;
      console.log("currentTool=" + currentTool);
};
```

Notice we're using an **anonymous function** as the value of the `onchange`, rather than a named function like `doToolChange`. Neither approach is incorrect, it's mostly a matter of preference and coding style. But doing it the way we did here groups the event handling code with the actual code that will execute, which makes it easier for the developer to quickly know what's going on because you don't have to scroll down the page looking for the *doToolChange* function.

Test the tool select, and check the console to be sure the value of `currentTool` is being changed.

5) Similar to the above, get the `#fillStyleChooser <select>` working so that the `fillStyle` global gets changed. Be sure to log it out and test it as we just did above.

(Alternatively, pop in a breakpoint and watch the `fillStyle` variable in the debugger instead of using `console.log())`

Part C - Coding the Rectangle tool (first try)

1) Replace your implementation of `doMousedown` with this one:

```
function doMousedown(e){
      dragging = true;
      var mouse = getMouse(e);

      switch(currentTool) {
            case TOOL_PENCIL:
                  ctx.beginPath();
                  ctx.moveTo(mouse.x, mouse.y);
                  break;

            case TOOL_RECTANGLE:
            case TOOL_LINE:
                  origin.x = mouse.x;
                  origin.y = mouse.y;
                  break;
      }
}
```

Notice we've added a switch that will run different code based on the value of `currentTool`.

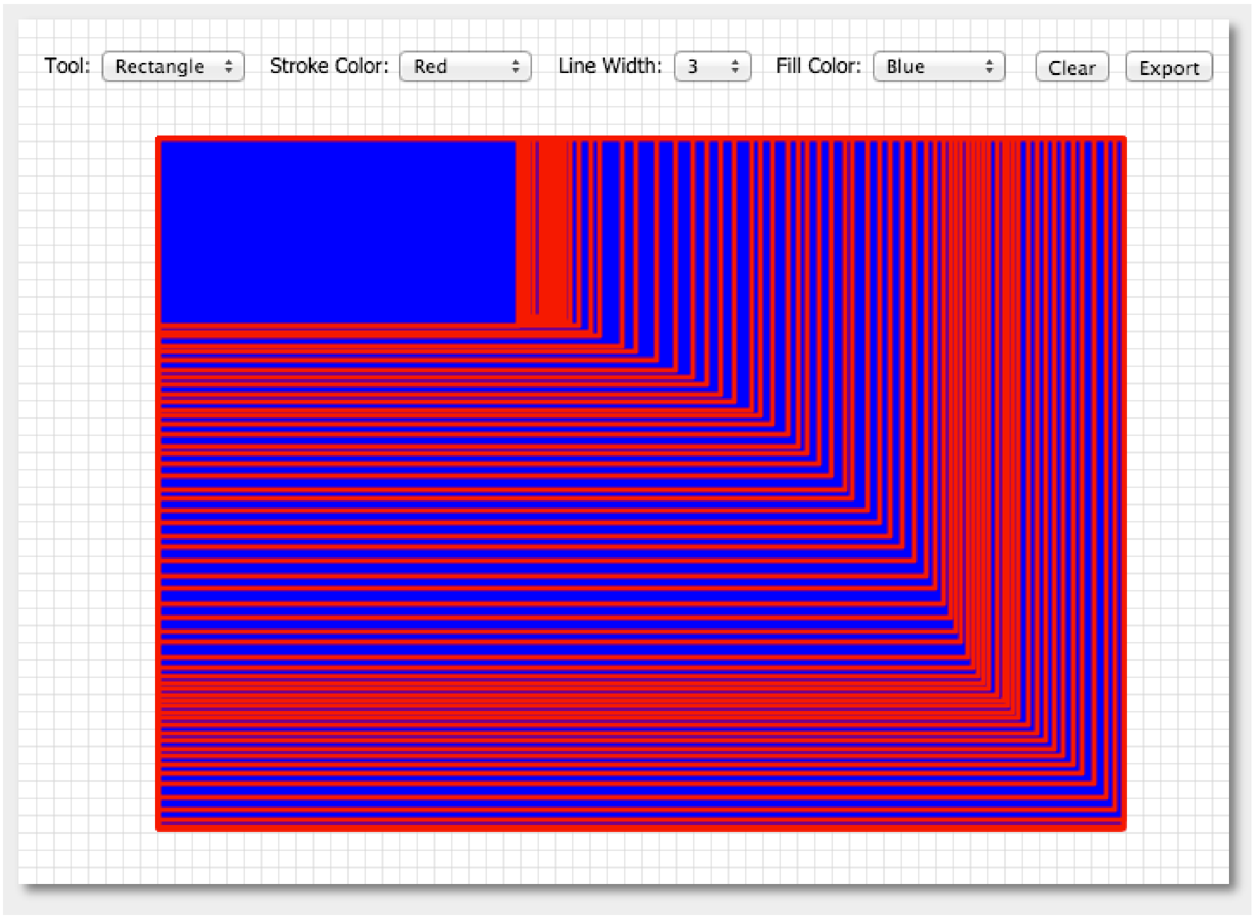
Fun Fact: most computer languages only allow you to have a `case` that is an integer value, but JS allows you to specify a string or even a conditional as a `case` value.

Run the code and test all three tools - the pencil should work as before, the line and rectangle will still draw, but oddly.

2) Now make `doMouseMove` appear as follows:

```
function doMousemove(e) {
      if(!dragging) return; // bail out if the mouse button is not down
      var mouse = getMouse(e); // get location of mouse in canvas coordinates

      switch(currentTool) {
            case TOOL_PENCIL:
                  // set ctx.strokeStyle and ctx.lineWidth to correct values
                  ctx.strokeStyle = strokeStyle;
                  ctx.lineWidth = lineWidth;

                  // draw a line to x,y of mouse
                  ctx.lineTo(mouse.x, mouse.y);

                  ctx.stroke(); // stroke the line
                  break;

            case TOOL_RECTANGLE:
                  /*
                  These adjustments keep the rectangle tool working even if
                  the user drags from right to left. Recall that canvas will
                  draw a rectangle left to right with the ctx.fillRect() and
                  ctx.strokeRect() methods
                  */
                  var x = Math.min(mouse.x,origin.x);
                  var y = Math.min(mouse.y,origin.y);
                  var w = Math.abs(mouse.x - origin.x);
                  var h = Math.abs(mouse.y - origin.y);

                  // fill and stroke the rectangle
                  ctx.strokeStyle = strokeStyle;
                  ctx.fillStyle = fillStyle;
                  ctx.lineWidth = lineWidth;
                  ctx.fillRect(x,y,w,h);
                  ctx.strokeRect(x,y,w,h);

                  break;

            case TOOL_LINE:
                  break; // line tool is broken for now

      }
}
```

Now test the code. The **Pencil** tool should work fine, the **Line** tool won't work at all, and the **Rectangle** tool - **Whiskey Tango Foxtrot**!


Tool: Rectangle
Stroke Color: Red
Line Width: 3
Fill Color: Blue

You'll be OK as long as you're dragging left-to-right. But the moment you drag right-to-left your old rectangle will still be visible, which results in something like the above.

Part D - Coding the Rectangle tool (a better way)

There are a few ways to fix this problem. The approach we're going to use here:

i) Draw just the rectangle to a temporary canvas, and then clear out that temporary canvas and redraw the current state of the rectangle every time the user moves the mouse. Periodically clearing the canvas before re-drawing the rectangle will get rid of the undesirable "ghosting" effect.

ii) When the user releases the mouse, we will then copy the contents of the temporary canvas to our main canvas.

1) First let's position our `#topCanvas` canvas (the "temporary" canvas mentioned above) to sit on top of `#mainCanvas`. Add the following CSS to the `<style>` element.

```
#topCanvas{
    background: rgba(0,0,0,0);
    position:absolute;
    left: 10px;
    top: 10px;
    z-index: 1;
    cursor: pointer;
}
```

Notice we have 3 major elements on the page: `#mainCanvas, #topCanvas`, and `#controls,` and they are now stacked on top of one another using `position:absolute, z-index`, and the `top` and `left` properties.

Be sure to look over the CSS and be sure you understand what's going on. If not, ask!

2) Try the **Pencil** and **Rectangle** tool again - you'll notice they no longer work - why?

Because the `#mainCanvas` currently has all of the mouse event handlers attached to it, and it's being obscured by `#topCanvas`

There's an easy way to fix this in modern versions of the browsers Chrome, Firefox, and Safari. Add the following to the `#topCanvas` CSS rule:

```
pointer-events:none;
```

This tells `#topCanvas` to ignore mouse events and pass them on to any elements below it.

Here's the docs on pointer-events:
https://developer.mozilla.org/en-US/docs/Web/CSS/pointer-events

Here's the current adoption chart for `pointer-events`. Notice that IE 11 is required for IE users: http://caniuse.com/#feat=pointer-events . We'll ignore IE < 11 users for now. Test the code, the **Pencil** and **Rectangle** should work as before.

3) Add 3 new global variables to the GLOBALS section

```
topCanvas
topCtx
```

4) In the `// initialize some globals` section of `init(),` add the following code:

```
topCanvas = document.querySelector('#topCanvas');
topCtx = topCanvas.getContext('2d');
```

5) In the `// set init properties of the graphics context` section of `init(),` modify the code to set the values for both contexts using multiple assignment:

```
// set initial properties of both graphics contexts
topCtx.lineWidth = ctx.lineWidth = lineWidth;
topCtx.strokeStyle = ctx.strokeStyle = strokeStyle;
topCtx.fillStyle = ctx.fillStyle = fillStyle;
topCtx.lineCap = ctx.lineCap = "round";
topCtx.lineJoin = ctx.lineJoin = "round";
```

6) Because we don't want to alienate IE < 11 users, we'll change the mouse event handlers to point at `#topCanvas`

```
topCanvas.onmousedown = doMousedown;
topCanvas.onmousemove = doMousemove;
topCanvas.onmouseup = doMouseup;
topCanvas.onmouseout = doMouseout;
```

You'll also have to delete the `pointer-events:none;` line in the `#topCanvas` CSS

7) Test the App - it should function as it did at the end of Part C.

8) Soon we're going to need a helper function to clear out the `#topCanvas` - go ahead and add it:

```
function clearTopCanvas(){
      topCtx.clearRect(0,0,topCtx.canvas.width,topCtx.canvas.height);
}
```

9) Since we've got this `topCtx`, let's draw to it instead of `ctx` whenever the user is using the **Rectangle** tool. Change the `TOOL_RECTANGLE` case in `doMousemove` to this:

```
case TOOL_RECTANGLE:
var x = Math.min(mouse.x,origin.x);
var y = Math.min(mouse.y,origin.y);
var w = Math.abs(mouse.x - origin.x);
var h = Math.abs(mouse.y - origin.y);

topCtx.strokeStyle = strokeStyle;
topCtx.fillStyle = fillStyle;
topCtx.lineWidth = lineWidth;

// erase old rectangle
clearTopCanvas()

// draw new rectangle
topCtx.fillRect(x,y,w,h);
topCtx.strokeRect(x,y,w,h);

break;
```

Test the app. You should now be able to draw one rectangle on the top canvas without the "ghosting" effect. But when you start to draw a second rectangle - the first one disappears. Why? The short answer: because we told it to - see the `clearTopCanvas()` function above.

What's the solution? Easy - once the user is done drawing the rectangle, we just need to copy the contents of `#topCanvas` down to the bottom `#ctx`.

10) Copying the rectangle to the main canvas:

- **When** do we do the copying? In the `doMouseup()` callback function when the user is done drawing the rectangle.

- **How** do we do the copying? Using `ctx.drawImage()` – which copies one of 3 things to the canvas: an image, a canvas, or the contents of a `<video>` tag.

Here's an online example that demos the various forms of draw image:
http://www.w3schools.com/tags/canvas_drawimage.asp

We'll be using the simplest form of `ctx.drawImage()` which is:

```
ctx.drawImage(image,x,y) // just draw the image starting at x,y
```

Go ahead and modify `doMouseup()` to appear as follows:

```
function doMouseup(e) {
      switch(currentTool) {
            case TOOL_PENCIL:
                  ctx.closePath();
                  break;
            case TOOL_RECTANGLE:
            case TOOL_LINE:
                  if(dragging){
                        ctx.drawImage(topCanvas,0,0);
                        clearTopCanvas();
                  }
                  break;
      }
      dragging = false;
}
```

Test it. You should be able to draw multiple rectangles now.

11) To get the rectangle drawing to “cancel” if the mouse leaves the canvas, make `doMouseout()` look like this:

```
// if the user drags out of the canvas
function doMouseout(e) {
      switch(currentTool) {
            case TOOL_PENCIL:
            ctx.closePath();
                  break;
            case TOOL_RECTANGLE:
            case TOOL_LINE:
                  // cancel the drawing
                  clearTopCanvas()
                  break;
      }

      dragging = false;
}
```

12) At this point you should be able to draw multiple rectangles, and change both their `strokeStyle` and `fillStyle` using the drop downs. Well done!

Part E - Coding the Line tool

1) To finish up, you need to get the **Line** tool working. Here are some hints:

- note that in `doMousedown()` for both the TOOL_RECTANGLE and the TOOL_LINE cases we're grabbing the position of the mouse and storing it in origin. You don't need to modify any code here - it's all set.

- in `doMousemove()` you need to complete the case for TOOL_LINE and do your line drawing there. You'll be drawing to the `topCtx` line the Rectangle above. The start point of the line should be the origin, the end point of the line will be the position of the mouse over the canvas. You'll need to describe a path and then stroke it. Don't forget to erase the previous line first.

Hint: You won't need the `Math.min()` and `Math.abs()` functions that we used in the rectangle code because canvas is perfectly fine with drawing lines right-to-left and any other direction up or down.

- note that `doMouseup()` - which does the `topCtx` to `ctx` drawing, and `doMouseout()` which cancels the drawing, are the same for both the **Rectangle** and **Line** tools. You won't need to modify these methods.

2) Your **Line** tool should now function as the **Rectangle** tool does - good job!

Be sure to go back over the code and be sure you understand what everything is doing - if you don't, ask!

Part F - Submission

Rubric

DESCRIPTION	SCORE	VALUE %
Fill Tool – Fill Tool works correctly filling in shapes with the appropriate fill color.		**10**
Rectangle Tool – Rectangle tool works correctly drawing rectangles. There are no trailing artifacts and leaving the canvas cancels the operation. Letting off the mouse draws to the bottom canvas and there are not any weird scaling issues.		**40**
Line Tool – Line tool draws lines correctly. There are no trailing artifacts and leaving the canvas cancels the operation. Letting off the mouse draws to the bottom canvas.		**50**
(Bonus) Rectangle Preview – Rectangles drawn on the #topCanvas as the user is dragging are drawn at 30% opacity. Once the user releases the mouse, the Rectangle is then drawn at 100% opacity on the #mainCanvas.		**+10 (bonus)**
(Bonus) Circle Tool – Add a circle tool. This needs to work correctly to receive the bonus. Recall that the center of a circle is the x,y passed into arc() function.		**+20 (bonus)**
(Bonus) Linear Gradient Tool - Add a linear gradient as a fill option. This needs to work correctly to receive the bonus. A gradient is simply a transition between 2 or more colors. See this demo: http://www.w3schools.com/tags/canvas_createlineargradient.asp		**+20 (bonus)**
Previous Tool Errors – Pencil, clear and export should still work correctly. Errors with this will mean deductions in the grade.		**-50% (up to)**
Errors Thrown – Any errors thrown in the console.		**-20% (this time)**
Additional Penalties – These are point deductions for poorly written code or improper code. There are no set values for penalties. The more penalties you make the more points you will lose.		
TOTAL		**100%**

ZIP and post - see dropbox for due date. **Include a link in your submission comments!**