05433-B/05633-B User Interface Lab (Section B - GUI)

Project 3: My Paint – Painting on the Canvas

due 11:59pm Tuesday, October 14, 2014.

Goal

This assignment continues the progression from the previous assignment by adding the actual canvas painting to our My Paint application. The goal of the assignment is to get experience with the `<canvas>` tag's drawing functions and to do some more sophisticated user input handling.

Your task is to add the functionality for 5 different kinds of brushes to the toolbar from the last assignment. These 5 brushes include: Square, Circle, Star, Eraser and a custom brush of your own design. The expected behavior of each brush is as follows:

- **Square:** The brush should draw a square **centered** at the cursor with side lengths equal to the brush size value. It is ok if the square is slightly off center for even brush sizes.
- **Circle:** the brush should draw a circle centered at the cursor with a **diameter** equal to the brush size value.
- **Star:** The brush should draw a regular 5-pointed star centered at the cursor where each line draw in the star has a length equal to the brush size.
- **Eraser:** The eraser brush should erase a square of the canvas centered at the cursor with side lengths equal to the brush size. Additionally, when the eraser brush is selected users should not be able to change the brush color.
- **Custom:** The behavior of the custom brush is up to you with the following requirements. First, the custom brush must involve at least 2 drawing commands (i.e. `lineTo()`, `arc()`, `rect()`, etc. but not including `save()`, `restore()`, `beginPath()`, `fill()` or `stroke()`). Second, the custom brush cannot be the smiley face function that we did in class. Include a comment in your javascript describing what the Custom brush's expected behavior is.

In addition to implementing the 5 brushes you will need to implement a few other cosmetic/usability features. Firstly, paint programs generally use a number of different cursors depending on what the mouse is over. Using the CSS `cursor` property, ensure that your interface uses a logical set of cursors. You might consider asking some other people what their expectations for a cursor would be in different contexts.

Secondly, the functionality that resizes the canvas according to the resizing of the window will also clear the canvas, which is likely to confuse or frustrate users. To counteract this you will need to augment the canvas resizing function (either the one you wrote or the one from the previous assignment's answer key) to repaint the canvas in the event of a window resize. There are 2 levels to this feature. The first, and the only required level, is to have your program redraw what it can fit in the canvas after a resize and discard the rest. The

second level, which is available for bonus points, stores a copy of the canvas that remembers what can't fit on the screen and will redraw the full image if the window is resized back to a larger size. Note that this is more than simply making the canvas larger than what can fit on the screen. Both of these options are demonstrated in the project video.

As with the previous assignment you should feel free to make any individual changes that you think would make the program more useful or usable.

Provided Files

This assignment has no official provided files other than this specification. You should start out by expanding your submission to the previous toolbar assignment. I have made 2 answer keys available from the previous assignment, the one from the video and a new revised one. You should feel free to reference these answer key files and incorporate any features from them that you might want to use. No points will be taken off for any errors related to features that were required by the previous assignment so feel free to take what is necessary to get your toolbar working correctly.

Hints

- A few parts of this assignment require the use of angles and trigonometry. The standard trigonometric functions are available in javascript through `Math.sin()`, `Math.cos()`, and `Math.tan()` etc. Additionally, all angles in javascript are in radians. If you're more used to working with degrees rather than radians you might find it useful to define a constant `var deg2rad = Math.PI/180`, which you can multiply any degree amount by to get the equivalent amount in radians, e.g. `45 * deg2rad` would be the radian equivalent of 45 degrees.
- Redrawing a whole canvas will involve an imageData object (http://www.w3schools.com/tags/ref_canvas.asp, under Pixel Manipulation). The challenge in creating the second form of redrawing comes from the imageData object being one giant 1D array instead of the 2D data it actually represents.

Tentative Rubric

Turned in & compiles and your name is in a comment:	5 points
The Square brush paints a square according to the specification:	5 points
The Circle brush paints a circle according to the specification:	5 points
The Star brush paints a star according to the specification:	10 points
The Custom brush paints something according to the specification:	10 points
The Eraser brush erases a square according to the specification:	5 points
The interface employs logical contextual cursors:	5 points
Resizing the window does not clear the entire canvas:	5 points
The canvas redraws content that was previously off screen	Up to 5 bonus points
The star brush supports a user defined number of points	Up to 5 bonus points
Total	50 points

Critical Thinking

You are not required to turn in answers to any of the questions in this section, but we recommend that you explore and think about some of the questions.

1. Functions like `document.getElementById("someId")` and `canvas.getContext("2d")` are deceptively slow functions. In normal circumstances you won't notice the time it takes them to run but when they are called frequently, e.g. inside a mousemove event listener, the interface will begin to appear laggy. What strategies could you use to mitigate the performance problems that come from expensive functions like this?
2. The current interface will draw shapes according to absolute pixel sizes. On smaller devices, e.g. phones, it could become hard for users to see what they are doing behind their fingers. What strategies could you use to make painting easier across device form factors?

Turning it in

Project 3 is due by 11:59pm October 14th, 2014 as a zipped file. Email your file to Erik at eharpste@cs.cmu.edu. Late entries will be penalized -5% for every late day.