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

Project 4: My Draw – Using a Display List

due 11:59pm Thursday, October 30, 2014.

Goal

This assignment completes the progress of the previous 2 assignments by adding in the ability to draw shapes using drag resizing and manipulate them using drag and drop interactions. The goal of this assignment is to get some experience with more complex user interactions and play with using a display list paradigm to manipulate interactive graphics. This assignment will also have you create reusable javascript objects.

For this assignment we will be using a DisplayList class that I have written myself and provided with this spec. This DisplayList does some of the basic management of graphical objects, however you will have to write the graphical objects that go into the list. You will also have to write some additional management functions on top of the DisplayList to achieve the drag and drop interactions.

Your task is 2 fold:

First, you must translate the drawing functions (square, circle, and star; you are not required to implement your custom brush) you used in the previous assignment into graphical object classes. Each graphical object should retain all of the properties it needs to draw its shape as well as a constructor to initialize those properties. In addition to an object constructor each object should implement the following 4 function:

1. `draw(ctx)` – This function should take as a parameter a CanvasRenderingContext, i.e. the result of calling `canvas.getContext(“2d”)` and use it to draw the shape according to its retained properties. For example calling `draw(ctx)` on a Square object should result in a square appearing on the main canvas.
2. `contains(x,y)` – This function should take as parameters the x and y-coordinates of a point and return true if that point would be inside of the drawn shape, and otherwise return false. For complex shapes, like the star, it is ok to assume that they would be contained in an invisible square and return true if (x,y) is contained in the square, rather than doing complex geometry calculations.
3. `moveTo(x,y)` – This function should move the graphical object to a new location specified by the provided x and y-coordinates. You can assume that this (x,y) positions is meant to be the new center of the shape, as if the user had clicked there in the painting program.
4. setSize(size) – This function should take 1 parameter specifying its new size. Each shape can be described by a single size value, as it was in the previous assignment. For example, the size of a square is its side length, the size of a circle is its radius, and the size of the star is its edge length. What size specifically changes is up to you as long as it makes sense.

Secondly, you will need to write a set of management functions which leverage the DisplayList to accomplish drawing interactions. The expected behavior follows this finite state machine:

1. Idle state – this is the default state of the program.
    a. If the user presses their mouse down on the canvas over an empty part of the canvas then a new graphical object is created at that point, using the properties currently selected in the toolbar and made the focus. Then the program proceeds to the Draw state.
    b. If the user presses their mouse down on the canvas over a graphical object then that graphical object becomes the focus and the program proceeds to the Drag state.
2. Draw state – this is when a user is dragging their mouse after starting a new shape.
    a. As the user drags their mouse the focused shape (the newly drawn one) changes size according to the distance between the original mouse down and the current mouse position.
    b. If the user releases the mouse then the focus is cleared (set to null) and the program transitions into the Idle state.
3. Drag state – this is when the user is dragging their mouse after clicking on an existing shape.
    a. As the user drags their mouse the focused shape moves along with the mouse to a new position. The shape's position relative to the mouse should not change as it moves, i.e. the distance between the mouse and the center of the shape should remain (roughly) constant.
    b. If the user releases the mouse then the focus is cleared (set to null) and the program transitions into the Idle state.
4. Delete state – this is a special state for deleting objects reached by setting the shape type in the toolbar to delete.
    a. If the user presses their mouse down on the canvas over a graphical object then that graphical object gets deleted from the canvas, i.e. removed from the DisplayList.
    b. If the user sets the toolbar's shape type to something other than delete then the program transitions into the Idle state.

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

As with the previous assignment you should start this project by building off of your existing toolbar code. You should make some edits to the toolbar to make it more usable for drawing. Firstly, change the Eraser option to Delete. Secondly, remove the brush size field. Since users will drag shapes out to whatever size they want the brush size field is no longer necessary.

The other provided file with this assignment is displaylist.js. This file contains the specification for the DisplayList class. The DisplayList makes itself available as a global variable named `dl`, and has 5 functions:

1. `setup(canvas)` – this function sets up the DisplayList's reference to the canvas and MUST be called before any other function will work. The canvas parameter provided to this function should be a reference to your main canvas. If `setup()` is not called first all other functions will throw errors.
2. `addGraphicalObject(go)` – this function is used to add a graphical object to the list. This function assumes that the go object implements the 4 required functions and will log a warning to the console if it does not.
3. `removeGraphicalObject(go)` – this function removes a graphical object from the list and returns the object if it was successful, otherwise it will return null.
4. `redraw()` – this function clears the canvas and redraws all of the objects in the list in the order in which they were added. If an object does not have a `draw()` function this will throw an error.
5. `getObjectContaining(x,y)` – this function searches through the list for an object that would contain the point (x,y). If an object is found it will be returned otherwise the function will return null. If an object in the list does not have a `contains()` function defined this will throw an error.

Hints

- You may find it useful to make multiple javascript files to organize yourself. My organization looks like this:
    - toolbar.js – all of the functions from the previous assignments related to managing the toolbar, like tracking the mouse coordinates and properly updating color.
    - shapes.js – all of my shape class definitions are here set as global functions so I can use them elsewhere.
    - main.js – the finite state machine that responds to mouse input is here.
- The DisplayList has a number of checks built into it that will display warnings and errors to the console when it is being used incorrectly. If you have the Netbeans plugin in chrome the console appears in the output tab of the Netbeans interface. If you are not using Netbeans the console log can be viewed if you right click on the page and select 'Inspect Element' then go to the console tab.
- The formula to find the distance between 2 points (x1,y1) and (x2,y2) is: $\sqrt{(x1-x2)^2+(y1-y2)^2}$, you can do square root with `Math.sqrt(val)` and square can be accomplished by `Math.pow(val,exp)` so `Math.pow(val,2)`
- You will again need to change how window resizing is handled. Every time the canvas changes size it will need to redraw to the new size.
- Do not name any of your shape classes `Rect`, various browsers (Chrome in particular) behave weirdly when you do this because they rely on a hidden class named `Rect`. It's safer to use a name like `Rectangle` or `Square`.

Tentative Rubric

A Square graphical object is defined:	
The Square's draw function is correct:	5 points
The Square's contains function is correct:	5 points
The Square's moveTo function is correct:	5 points
The Square's setSize function is correct:	5 points
A Circle graphical object is defined:	
The Circle's draw function is correct:	5 points
The Circle's contains function is correct:	5 points
The Circle's moveTo function is correct:	5 points
The Circle's setSize function is correct:	5 points
A Star graphical object is defined:	
The Star's draw function is correct:	5 points
The Star's contains function is correct:	5 points
The Star's moveTo function is correct:	5 points
The Star's setSize function is correct:	5 points
Mouse down on empty canvas creates a new graphical object with correct shape and color:	5 points
Dragging the mouse resizes the shape:	5 points
Mouse down on an existing object makes it a focus:	5 points
Dragging the mouse moves the focused object:	5 points
The shape's center stays a constant distance from the mouse:	5 points
Clicking a graphical object in delete mode removes it:	5 points
Changes done to one graphical object do not affect the others:	5 points
All objects redraw correctly when the window is resized	5 points
Custom shape implemented (based on complexity)	Up to 10 bonus points
Interface allows for users to define fill and stroke properties	Up to 10 bonus points
Interface visually indicates which object is currently in focus	Up to 5 bonus points
Total	100 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. Your program now has the backbone of a drawing editor like Adobe Illustrator. How might you implement more advanced features like layering?
2. Currently the display list erases the canvas and redraws every graphical object when any one of them changes, potentially leading to lots of unnecessary redrawing. How might you change the display list so it only draws what it needs to?

Turning it in

Project 4 is due by 11:59pm October 30th, 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.