TriUlti

iFlowEdit

< (16/11/2011 - 21/11/2011) >

<21/11/2011>

KARAOĞUZ, Mehmet Ozan

DRAWING BASIC SHAPES WITH HTML5 CANVAS ELEMENT

One of the technologies we will use is HTML5 and its canvas element. Because of this, I have worked on canvas element and how to draw basic shapes like triangle, square and circle on it. I found that without a JavaScript code, drawing is not possible and to draw something on the canvas element, I should get its two dimensional context use some attributes and functions of it that I will explain now.

strokeSytle: It sets style of stroke. I used this attribute to set black as stroke color.
fillStyle: It sets style of fill. I used this attribute to set blue as fill color.
arc(): It creates an arc. I used this function to create circle.
moveTo(): It creates a new sub path which starts at the given point.
lineTo(): It creates a line. I used this function to create triangle.
rect(): It creates a rectangle. I used this function to create square and rectangle.
bezierCurveTo(): It creates a bezier curve. I used this function to create oval.
stroke(): It strokes the created shapes.
fill(): It fills the created shapes.

Here is a screenshot of the little web page (Figure 1) I create to demonstrate my work. To open that web page, just unzip the attached file named “HTML5 Canvas.zip” and open the “index.html” file with any browser you want. To see the source code; open the”index.html” file with any text editor you want.

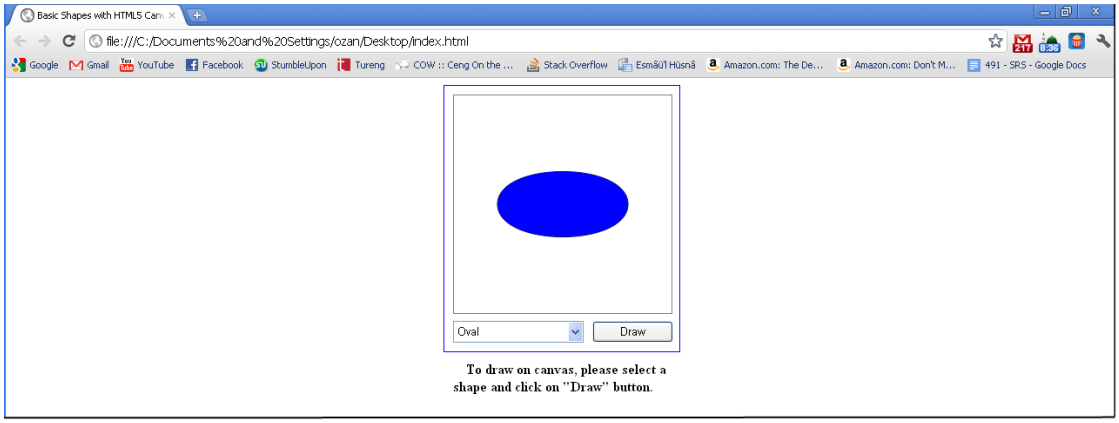

Figure 1 - Screenshot

KAYRAK, Alaattin

PLAYING WITH EXT-JS

I created ‘very’ rough design of the front-side of the project by using Sencha Ext-JS. As you can see, it consists of four major region; namely east, west, north, and center. Actually all of them are inside only one element, root element, viewport. There cannot be more than one viewport in one page. Regions’ three property is most important for showing them to users. They are explained (with usages) by Sencha in this way;

collapsible: True to make the panel collapsible and have an expand/collapse toggle Tool added into the header tool button area. False to keep the panel sized either statically, or by an owning layout manager, with no toggle Tool.

floatable: True to allow clicking a collapsed Panel's placeholder to display the Panel floated above the layout, false to force the user to fully expand a collapsed region by clicking the expand button to see it again.

split: True to create a SplitRegion and display a 5px wide Ext.SplitBar between this region and its neighbor, allowing the user to resize the regions dynamically. Defaults to false creating a Region.

In north region we have toolbox consists of possible some menu and its elements. Creating the toolbox helped to understand xtypes and possible different usage of them. In this region we have just split:false to get desired view.

In west region, there is treepanel element. This element shows some diagrams and its components. These components do not come via embedding them in treepanel store; they come via calling global variable. In this manner understanding of variable and sourcing concept progressed for Ext-Js. In this region we have just split:true to get desired view.

In east region, we have labels and textboxes to show property of selected element in canvas (in sample they are hard coded). Places of these elements are decided by type of the region. Therefore possible types are examined for the purpose. In this region we have just split, floatable, and collapsible all true to get desired view.

We have nothing in center region for now; however there will be a canvas element in future.

Here we have screenshot of the sample.

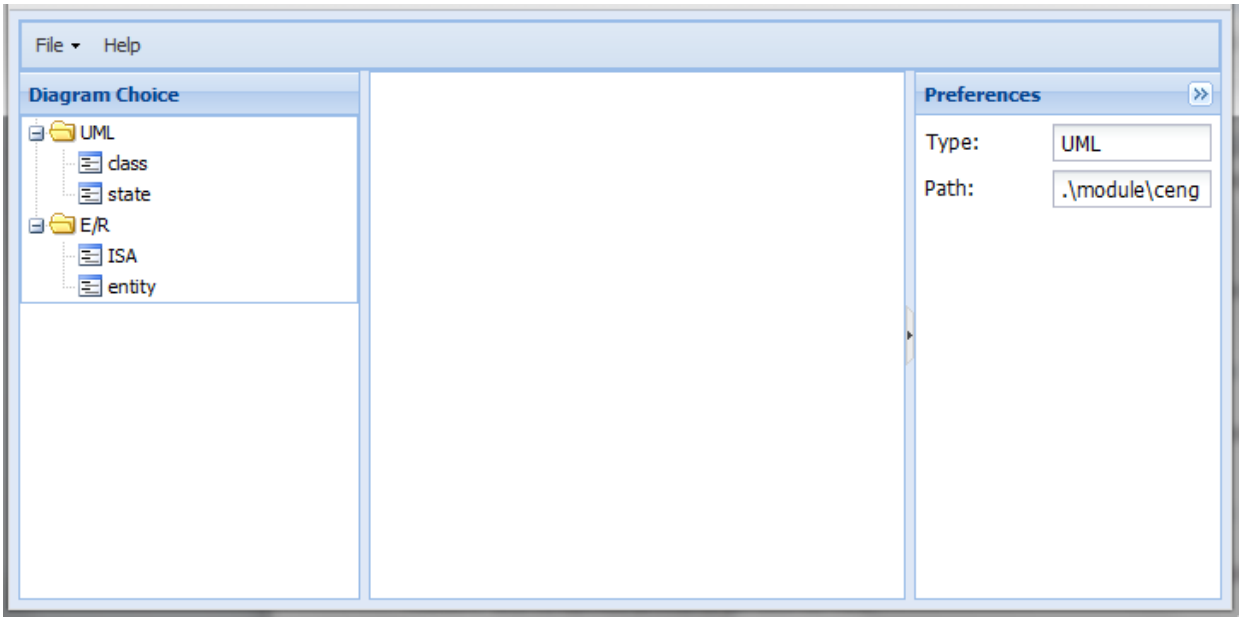

Figure 2

KORKMAZ, Ozan

WORKING ON PROCESSING AND PROCESSING.JS

To understand the basics of the usage of Processing language, I searched on Internet about it and analyzed some sample codes that I found. The website of Processing contains an API which shows the functions that can be used. Therefore, I glanced at the API, especially, 2D shaping parts. To be usable, creators of Processing had written a basic IDE that runs Processing code. In tests, I used this IDE to run my codes. Basic shapes like rectangle, line, circle, triangle, and square can be easily drawn with functions defined in Processing. For instance, after giving parameters for rec(x1, y1, width, height) function; in below example, rect(100, 150, 200, 100) with fill(colour_parameters) gives blue rectangle with background(colour_parameters). To run this code on web, we are going to use Processing.js library which was written to make Processing runnable on web. To see sample code of the below graph, there is a directory named “Processing Sample” which contains four files. “sample.html” file can be run on any modern browser thanks to HTML5 to see the graph.

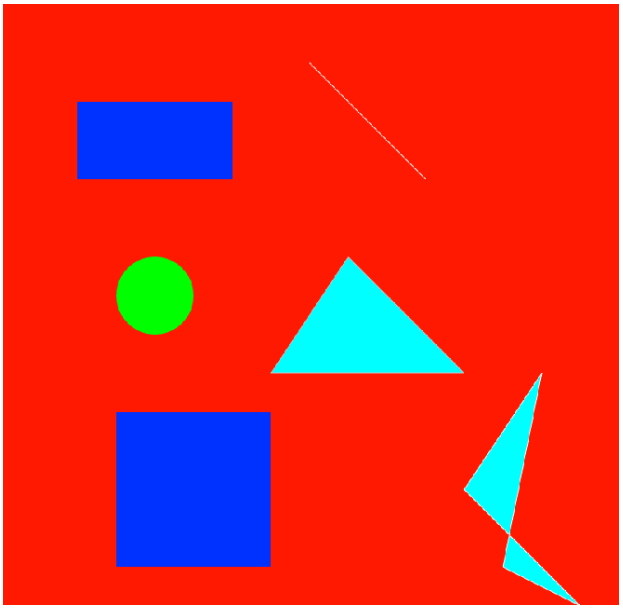

Figure 3

ORAL, Hakan

SHORTEST PATH ALGORITHMS

1. Dijkstra's Algorithm

It is a greedy algorithm that finds the shortest paths in a graph. For a given source vertex in the graph, this algorithm finds the shortest path with lowest cost between that vertex and every other vertex. Dijkstra's algorithm can also be used to find the shortest route between one item and all other items on the editor. Dijkstra's original algorithm does not use a min-priority queue and runs in O(|V|2) and its worst case performance is O(| E | + | V | log | V |) . Dijkstra's algorithm is a successive approximation scheme that solves the dynamic programming functional equation for the shortest path problem by the Reaching method.

Algorithm

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.

Mark all nodes except the initial node as unvisited. Set the initial node as current. Create a set of the unvisited nodes called the unvisited set consisting of all the nodes except the initial node.

For the current node, consider all of its unvisited neighbors and calculate their tentative distances.

For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be 6+2=8. If this distance is less than the previously recorded distance, then overwrite that distance. Even though a neighbor has been examined, it is not marked as visited at this time, and it remains in the unvisited set. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again; its distance recorded now is final and minimal. If the unvisited set is empty, then stop. The algorithm has finished. Set the unvisited node marked with the smallest tentative distance as the next "current node" and go back to step 3.

2. Floyd's Algorithm

This is a graph analysis algorithm to find the shortest paths with negative or positive edge weights. This is an example of dynamic programming and its worst case performance is O(|V|3), best case performance is Ω(|V|3) and worst case space complexity is Θ(|V|2).

Pseudocode

Conveniently, when calculating the kth case, one can overwrite the information saved from the computation of k − 1. This means the algorithm uses quadratic memory. Be careful to note the initialization conditions:

```
1 /* Assume a function edgeCost(i,j) which returns the cost of the edge from i to j
2    (infinity if there is none).
3    Also assume that n is the number of vertices and edgeCost(i,i) = 0
4 */
5
6 int path[][];
7 /* A 2-dimensional matrix. At each step in the algorithm, path[i][j] is the shortest path
8    from i to j using intermediate vertices (1..k-1).  Each path[i][j] is initialized to
9    edgeCost(i,j).
10 */
11
12 procedure FloydWarshall ()
13    for k := 1 to n
14       for i := 1 to n
15          for j := 1 to n
16             path[i][j] = min ( path[i][j], path[i][k]+path[k][j] );
```

Figure 4

In our project, we can use Dijkstra's algorithm because it has lower cost when finding the shortest path between one item and all other items.

References:

http://en.wikipedia.org/wiki/Dijkstra's_algorithm
http://www.mcs.anl.gov/~itf/dbpp/text/node35.html
http://www.cs.cornell.edu/courses/cs312/2002sp/lectures/lec20/lec20.htm