Assignment 1 – Canvas and GUI

UPR Río Piedras, Computer Science – Computer Graphics, Spring 2020

Abstract: Implement graphics drawing and GUI interaction in HTML5 and its canvas.

Guidelines for assignment hand in:
- Upload one ZIP file on Moodle, with extension .zip (please, no RAR or 7zip files)
- This ZIP file contains all code and data necessary to run your assignment. It is organized as follows
- Program should be using the 2D HTML5 canvas with Javascript.
- File organization (filenames, flat or hierarchical subfolders…) is up to you, but should be clear and meaningful.
- Make sure the example run by simply opening the HTML file, without having to modify the code. If any additional action is required to run the results, this must be explained in a clear way directly when opening the HTML page in the browser, with additional details in a README.txt if needed (for instance, if it needs to run through a webserver instead of directly from the file).

Useful references:
Canvas:
https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial
http://www.html5canvastutorials.com/

SVG:
https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial
http://tutorials.jenkov.com/svg/index.html

Debugging JS code:
https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_are_browser_developer_tools
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_went_wrong
https://developers.google.com/web/tools/chrome-devtools/

Part 0: Questions [15]

Answer these questions in a Questions.pdf or Questions.html document. You are welcome to add graphics to support your explanations.

Q1. [5] Explain the listener pattern (for instance used to react to user input in a GUI)

Q2. [5] Explain the difference between retained mode and immediate mode graphics API

Q3. [5] A screen is advertised to have 16M distinct colors. Is it a lot? Knowing that typical RGB images use 8 bits integer values, comment on that.

Implementation, Part 1 and 2:

Documentation and quality [10]
- Your code should be well organized (pay attention to indenting) and commented (Include comments that specify all relevant convention or intent).
- The HTML page should provide below the canvas a short but clear explanation for the user to know how to access all the functionalities you developed.

Do not forget to provide description of the implemented features directly on the HTML page so that the user can discover them.

Part 1: Simple Graphics – Vector and Raster [30]

Objective: Design graphics using both Canvas and SVG, and add interaction. Experience the difference between the two approaches.

Use template code `simplegraphics.html` that contains a canvas and a SVG element side by side. You need to submit only the final code, as the questions build on each other.

Q1: [20] Recreate the following image containing PacMan Ghost BLINKY using two techniques:
a) Canvas API, using paths and text
b) SVG, using path/poly tags and text tag
Hints:
- You may use Inkscape or any other SVG editing software to help in the design of the graphics, but the final code (both SVG and canvas API) should be simple and not contain any unnecessary code, as with hand written code.
- The eyes are ellipses (slightly more tall that wide). SVG has the ellipse element. The canvas API does have an ellipse API only on recent Firefox or Chrome versions, you may use it.
- Make sure the text is centered below the ghost

Q2: [10] We want to add interactivity to the graphics: when the user clicks on it, it will change color and change the name of the ghost, and cycle through all 4 of them: CLYDE (orange), BLINKY (red), PINKY (pink), INKY (cyan).
A simple way to store the information required to change the color and name is to create a model variable with the following information:

```
var ghosts = {
    current : 1,
    names : ['CLYDE', 'BLINKY', 'PINKY', 'INKY'],
    colors : ['orange', 'red', 'pink', 'cyan']
}
```

Implement the color change in both approaches:
a) In vector graphics, you just need to change the content of the SVG elements in the DOM.
Hints: The SVG tags belong to the DOM, so you can access them using `document.getElementById` *or* `document.getElementsByTagName`*.*
For the `path` *or* `poly` *tag that represents the red shape: you can change the attribute* `fill` *with* `path.setAttribute`*. For the text, this is not an attribute, but html code inside the tag.* `myText.innerHTML = "newText"` *will modify the* text *of tag* `myText`*.*

b) In canvas graphics, you have to redraw the whole scene from scratch.
Hint: Don't forget to clear the background with black color before drawing. You can use `fillRect` for that purpose.

Do not forget to provide description of the implemented features directly on the HTML page.

Part 2: Interactive Drawing [45]

Objective: We want to develop a drawing app that allows creating and modifying polygons in the 2D Canvas. Figure 1 shows a mockup of a typical final result. The final application should let the user:
- Draw a polygon and its bounding box /
- Modify the position of each polygon vertex
- Add or remove vertices

The directory `polygon` provides some startup template.

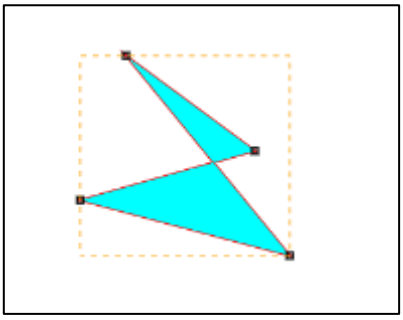

Figure 2: Mockup of the polygon editing app

In the following, we will try to follow the MVC pattern as much as possible. Define a global variable called `model` to store all state variables.

We define a *polygon* shape as an array of 2d points. For instance, a polygon passing through the vertices of coordinates (x1,y1), (x2,y2) and (x3,y3) is defined as
`pts = [{x:x1,y:y1}, {x:x2,y:y2}, x3,y3]`

Q1) Drawing shapes [10]
For this question, define a global model `model.pts`, initialize it to an arbitrary polygon at startup, and draw immediately to the canvas.

a) Write a function `drawPolygon(ctx, pts)` that draws an arbitrary polygon shape `pts` using context `ctx`. It should plot the lines and the vertices in different colors.
Hint: no need to go down to drawing pixels, you can use `moveTo` and `lineTo`.

b) Write a function `drawBoundingBox(ctx, pts)` that computes the bounding box of the `pts` polygon and draw it using context `ctx`. Use it to draw the bounding box of the polygon in question a.
Hint: For computing the bounding box represented as (left,top,right,bottom), left is defined as the minimum value of the vertices x coordinates. Similar reasoning for top, right, bottom.

Q2) Selecting and moving vertices [15]
a) Let the user select a vertex of `model.pts` by clicking close to it on the canvas. The selected vertex should appear in a different color. Make sure only one vertex is highlighted even when the user clicks several times. If the click is not close to any vertex, unselect the currently selected vertex.
Hint:define a `findVertex(x,y)` *function that computes for each vertex the Euclidean distance to the place where the user has clicked and returns the id of the one that close enough. Define a convention to handle the case where no vertex is selected (undefined or -1). Use it in the mousedown callback to update a variable* `model.selectedVertex`*. Write a* `redraw()` *function that draws both the polygon and the selected vertex.*

b) Let the user move the position of the selected vertex by +/-10 pixels horizontally and vertically by using the arrow keys from the keyboard.

Q3) Adding and removing vertices [10]
a) When the user presses the key '-', remove the current vertex. All remaining vertices should be reorganized in the pts array, and the id of the current vertex also, before redrawing.
Hint: You can use `pts.splice(start, 2)` *to remove 2 items* `pts[start]` *and* `pts[start+1]` *from the array and shift the rest of the array content accordingly. Note that the function modifies* `pts` *directly. See*
https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Array/splice

b) When the user presses the key '+', add a vertex in the midpoint between the current vertex and the next vertex, then redraw.
Hint:
- You can use `pts=pts.splice(start, 0, a,b)` *to insert* `a` *and* `b` *as* `pts[start]` *and* `pts[start+1]` *in the array and shift the rest of the array content accordingly. Note that the function modifies* `pts` *directly.*
- The midpoint between (x1,y1) and (x2,y2) is defined as ((x1+x2)/2, (y1+y2)/2)
- If the selected vertex `id` *is the last one, the next vertex is not vertex* `id+1`*, but vertex* `0`*. You may test the case with* `if (id==numVertices)`*, or use modular arithmetic:* `(id+1)%numVertices`

Q4) Dragging vertices [10]
Let the user move the position of the selected vertex by dragging the mouse (mousedown, move, mouseup). The display (of the polygon and its bounding box) should be refreshed after each mouseMove until the mouse button is released.
Hint: Use a dragMove callback called during a mouseMove event to handle the dragging by updating the vertex position and redrwaing. Define a Boolean state variable `model.isDragging` *that is set to true when a drag action is occurring, and to false when it is done.*

Bonus ideas (max [10]):
Implement non-trivial functionality to the existing code. Make sure you explain the functionality on the page so that it is visible and clear what was done.
Examples:
- Part1: Ghost eyes follow the mouse when it moves [5] / show several ghosts, each of which that can change individually from normal to afraid state when clicked [5]…
- Part 2: Add vertices by clicking on the polygon side [5], allow user to change vertex types to Bezier curves with control points [10]…