List, Loops Interactive Animation

Preparatory Remarks

- Make a new directory in your class projects directory for this homework assignment.
- Check your starting code template for H07. The layout, directory structure, libraries, and starting code will all be familiar to you.

In this assignment, the starting template is similar to what you would have ended up with upon completing the previous homework. In this assignment, the big idea is to make *many* animated bouncing balls instead of just one. Our code will be very similar, except that we will loop over arrays of objects to initialize and then to animate them.

Learning Outcomes

Arrays, while loops, math for graphical animation (distance, mapping between ranges)

Assignment Instructions

1. Instead of one disk, create 50 in an array.
    - Hint: create a variable to store the number of disks you will have.
    - Hint create a variable that you will use to index the individual disks in the array.
2. Everything you did for one disk, you will now do for each in your array.
    - Use while loops that increment your array index each time through the loop.
3. Initialize the disk rates so that the disks move randomly in all directions.
4. Initialize the disk colors to a random hue by constructing and hsl color string (note the Raphael hsl function takes each of the 3 args in [0,1]). Save the color string for each disk as attributes on the disk objects themselves (just as you do for the position and rate information).

[At this point, you should have a riot of differently colored bouncing balls on the screen.] The rest of the assignment will develop some more “advanced” skills!

5. Next, we want to change a feature of the graphical objects depending on if and where the mouse is pushed. Disks that are within a certain distance of a pressed mouse will turn white, otherwise they show their own color. To do that,
    - Use Raphael to create a transparent rectangle on top of everything on the paper in order to catch mouse activity. Remember, graphical objects must be filled to catch mouse activity! Use the ‘fill-opacity’ attribute.
    - Create a “state” variable (object) named mouseState with three properties (‘pushed’, ’x’, and ‘y’) to keep track of the position and of whether or not the mouse is currently pushed.
    - Add the event listeners to your filled transparent rectangle that you can use to track the mouse state. Remember that you need to get the ‘node’ attribute of Raphael graphical objects in order to use the addEventListener method.
    - Test your mouse state code before proceeding by printing info to the console!

Assignment Instructions Continued

1. Write a distance function that takes 4 variables (x1, y1, x2, y2) and returns the distance between two points (x1, y1), (x2, y2). The Math object will come in handy. If necessary, search the web for the simple equation for finding the distance between two points.
2. In your draw routine, change the color of the disks to white if they are within, say, 100 pixels in distance from the mouse position if it is pushed. Change any white disks back to their original color if they are greater than 100 pixels from the mouse or if the mouse is not pushed.

   Hints: console.log is your friend - print out messages you need. During testing you might want to

   - Set the number of dots at 1 or 2 for testing
   - Set the speed (xrates and yrates) of the dot(s) to zero
   - Use your console messages and breakpoint debugging skills

-