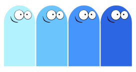
Four Shades of Blue 

This tutorial explains how to use D3.js to visualize genetic algorithms. In particular, we are interested in a dynamic visualization of the search space, where each individual in a given generation is represented solely by its X and Y coordinates, as well as its fitness, in a 3-element JS array. A generation, by its turn, must be an array of individuals. At any given time, four generations are presented in the browser screen. Individuals are represented as circles, where the radius is proportional to its fitness. The color of each individual represents which of the four presented generations it belongs to: deep blue represents the newest generation and light blue the oldest one.

Plotting a generation

First define a plot function that takes a generation as argument. The generation must be an array with coordinates for each individual, as well as their fitness. These numbers are assumed to be normalized between 0 and 1. the code in this section must be inserted in a function with the following signature:

```
function draw(data) { .. }
```

Define the dimensions of the plotting space. We'll be using a default 960x480 resolution:

```
var width = 960;
var height = 480;

var xScale = d3.scale.linear()
  .domain([0, 1])
  .range([0, width]);

var yScale = d3.scale.linear()
  .domain([0, 1])
  .range([height, 0]);
```

As explained, we'll be using four different shades of blue in our visualization. So, go ahead and define a color scale:

```
var colors = ['#ADD8E6','#1E90FF','#0000FF','#191970'];
var colorScale = d3.scale.quantile()
  .domain([0, 1, 2, 3])
  .range(colors);
```

Now, define a canvas so we can plot our blue circles on:

```
var margin = {top: 20, right: 40, bottom: 50, left: 50};
var svg = d3.select("svg")
  .attr("width", width + margin.left + margin.right)
  .attr("height", height + margin.top + margin.bottom)
  .style("border", "2px solid black")
  .append("g")
  .attr("transform", "translate(" + margin.left + "," + margin.top + ")");
```

We will now create one circle for each individual in our current generation:

```
svg.selectAll("circle")
  .data(data)
  .enter()
  .append("circle")
```

Se the position of each individual as well as its radius, which represents the fitness:

```
      .attr("cx", function(d) {return xScale(d[0]);})
      .attr("cy", function(d) {return yScale(d[1]);})
      .attr("r", function(d) { return 15*d[2]; })
```

By last, let's fill it with the highest color in our blue scale. Later on we explain how do make the color change happen:

```
      .style("fill", function(d) {return colors[3]})
```

Return our canavas, because we'll need it later:

```
    return svg;
```

Building transitions

We explained how to draw a single generation. Now we show how to code the transitions that will allow us to visualize the entire search space. Initially, define two global variables to hold the generation counter and our canvas:

```
    var svg = null;
    var i = 0;
```

Now, define a function that will contain our transition code:

```
    function run() {
```

We assume that the data for each generation is stored as a JSON file in the current directory in the server. The file for generation I should be named "gen_I.json". Tell d3 to read the file as follows:

```
      var name = 'gen_' + i + '.json';
      d3.json(name, function(error, data) {
```

Before drawing the next generation, remove the oldest generation from the screen, by selecting all the circles with the lightest shade of blue:

```
        if(svg) {
          d3.selectAll('circle').filter(function(d, i) {return d[3] == 0;}).remove();
```

Now, as we are going to advance one generation, we need to decrease the color of all individuals by one point in the scale, as follows:

```
          d3.selectAll('circle').filter(function(d, i) {
            return d[3]--;
          }).style("fill", function(d) {
              return colors[d[3]]
          ;})
        }
```

Finally, draw the next generation:

```
        svg = draw(data)
```

To control the transition speed, set a timeOut with, at least 50 milliseconds:

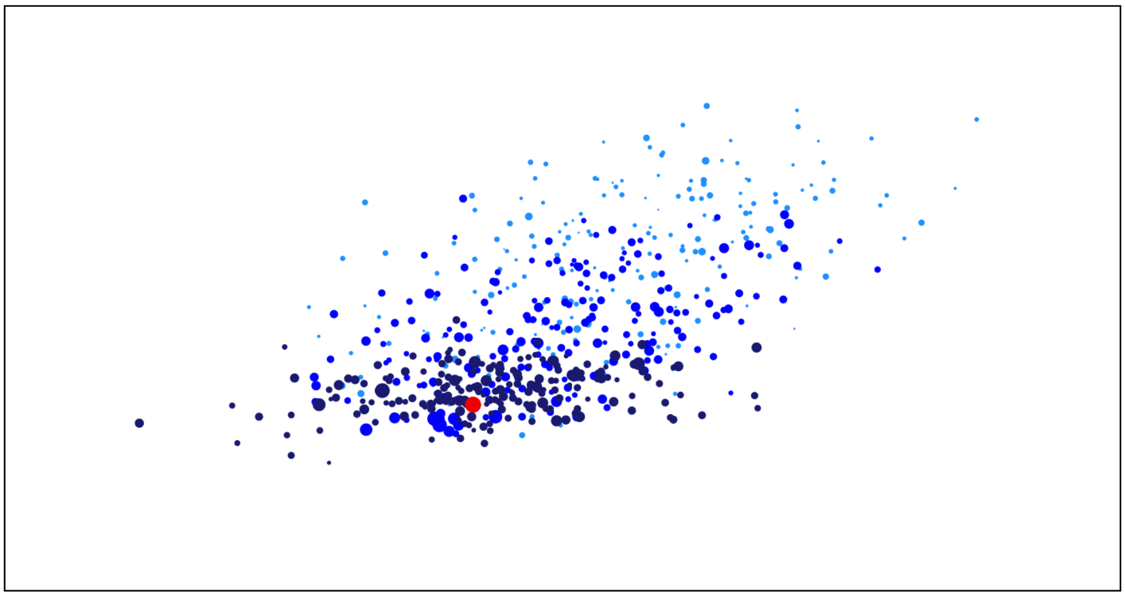
```
        setTimeout(next, 80);
        });
```

Finally, increment the generation counter and trigger the transition:

```
    i ++;
  }
  run();
```

Outcome

This resulting screen should look something like the following image;



DONE!!!