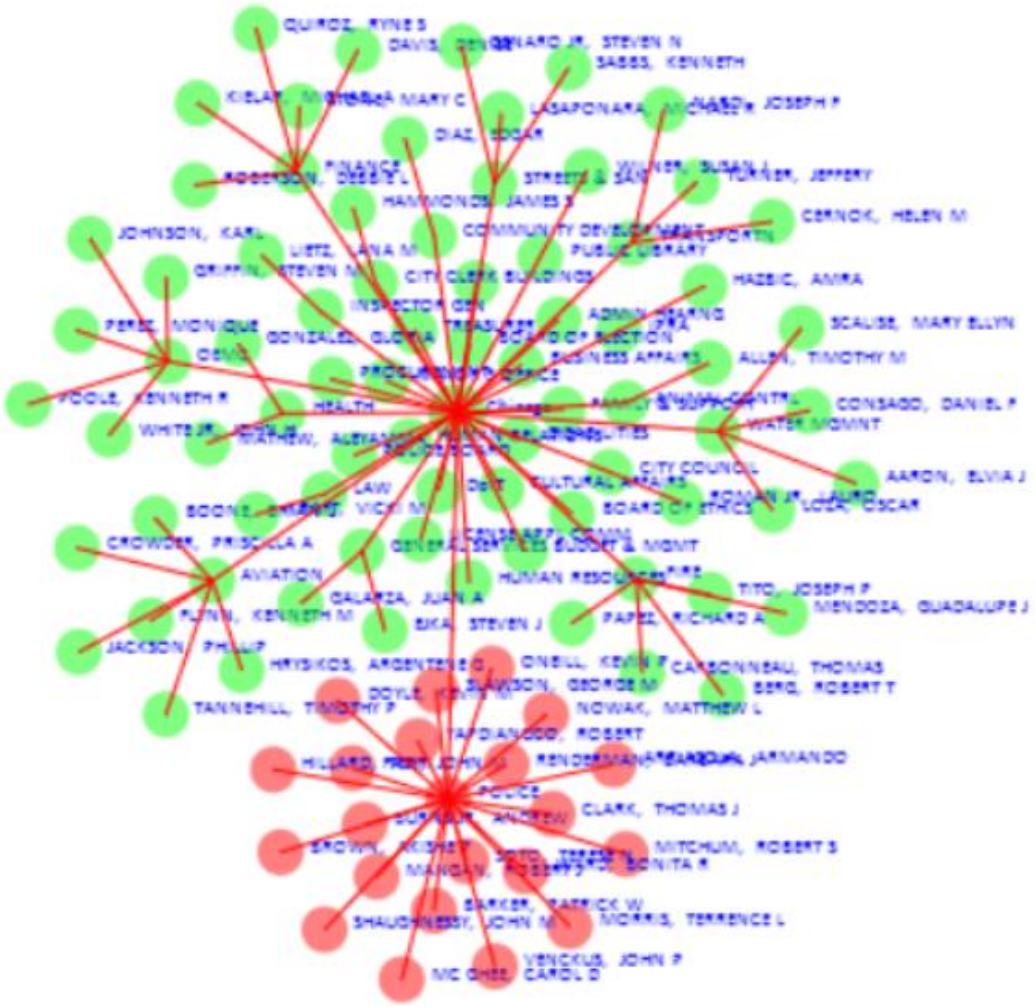
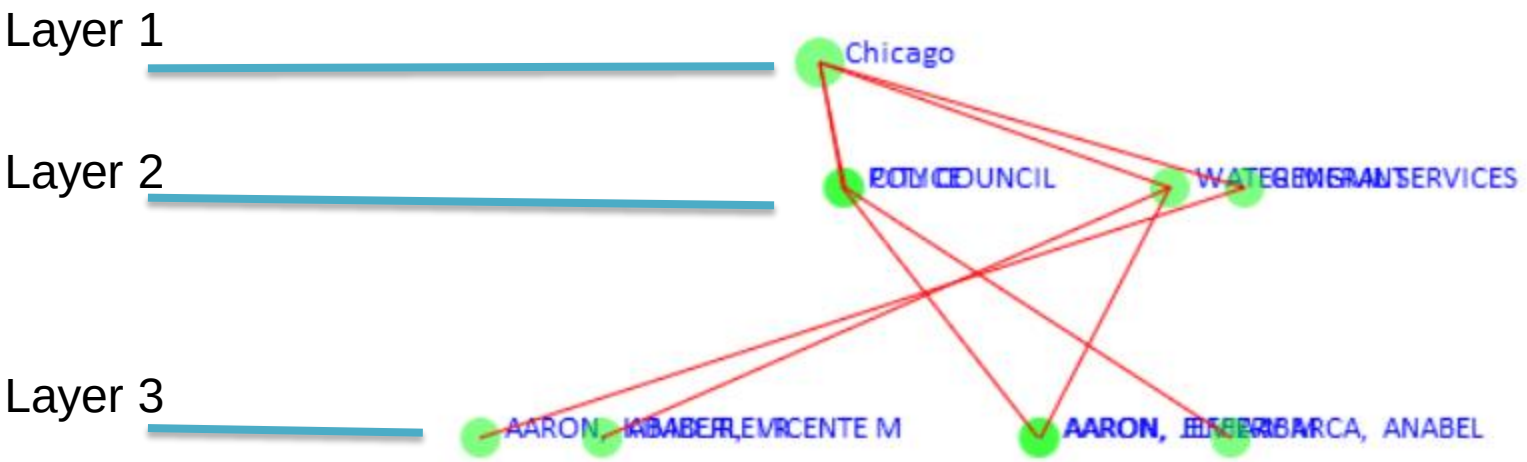
Lecture 29 – Graph Physics (Graphs part 2)

John R. Williams and Abel Sanchez, MIT

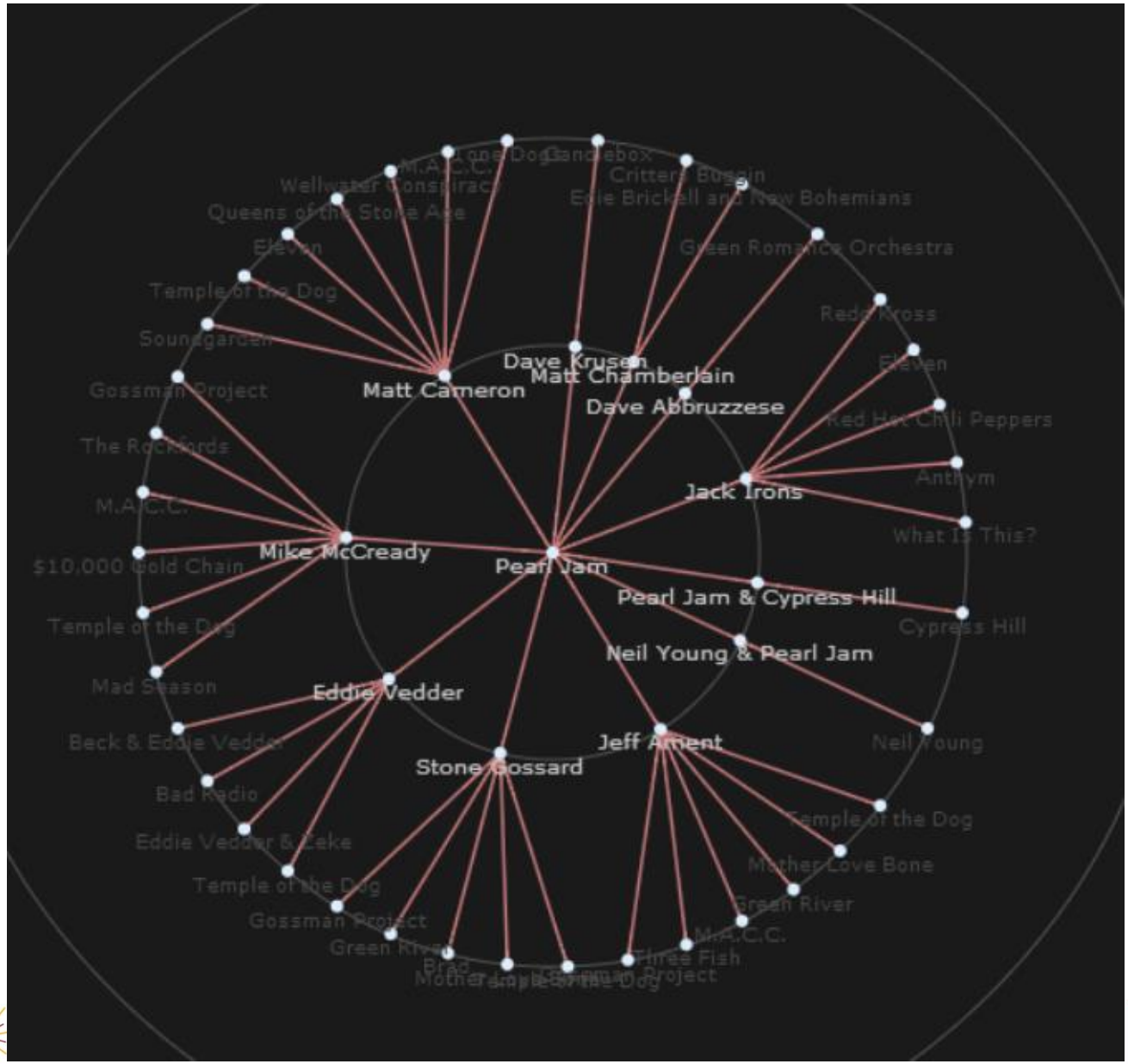


Review of Network Layout Issues

At present we are laying out our nodes in a rather random pattern. Typical strategies might be to lay out in layers as we have done (but with better structure) or on concentric circles.



Concentric Circle Layout

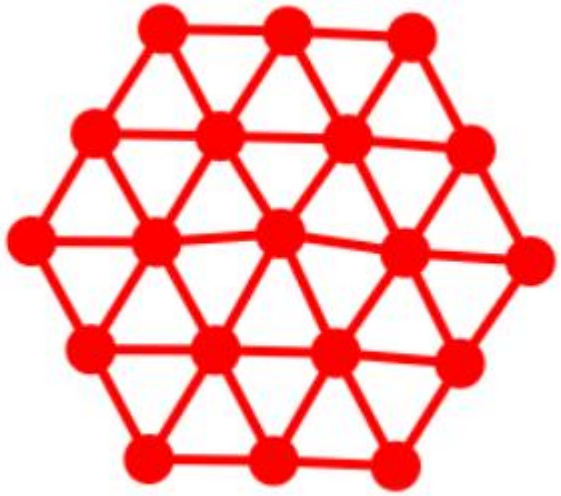


MITei

Massachusetts Institute of Technology

Remember Particle Mechanics

Remember in particle mechanics when we automatically put springs between particles and the result was the particles “self organized” themselves into structures such as below.
The effect of the springs was to keep particles apart and it was controlled by the ‘originalLength’ of the spring.
In our graphs we want to plot the ‘links’ so we might make each link a spring, which we plot.
However, we also want to put in ‘hidden-springs’ that keep particles separate just like the real springs. Lets call these a “repulsive-force”.



In Class Exercise 1

I've provided some code from our Particle Mechanics days so that you can use pieces to put physics into the starter code.

InClassGraphPhysicsEx1 – the starter code (same as we ended up with in last lesson.

Physics Bits - You need to use these to put physics into the starter code. They are taken from our Particle Mechanics days. Here are a few of the bits

```
function RunPhysics(){
  setInterval(UpdateAll, 10);
}
function UpdateAll(){
  // zero all the forces on the masses
  var n = sim.nodes.length;
  for(var i=0;i<n;i++){
    sim.nodes[i].force = Vector(0,0);
  }

  // update forces due to springs and apply to particles
  n = sim.links.length;
  for(i=0;i<n;i++){
    sim.links[i].updateSpringForces();
  }
   // detect neighbors and put springs
  detectNeighbors();

// put yield of links in here if needed later

  // update displacements
  n = sim.nodes.length;
  for(i=1;i<n;i++){
    sim.nodes[i].updateDisplacements();
  }
  plotAll();
}
function detectNeighbors(){
  var n = sim.nodes.length;
  for(var i=0;i<n;i++){
    for(var j=i+1;j<n;j++){
      if(sim.nodes[i].center.distance(sim.nodes[j].center) < sim.interactionDista
        if(noSpring(sim.nodes[i],sim.nodes[j])){ // check if spring exists alrea
          var s = Link(sim.nodes[i],sim.nodes[j]);
          sim.links.push(s);
```

```
function Node(center,radius){
  var velocity = Vector(0,0);
  var force = Vector(0,0);
  var mass = 1;
  var children = [];
  var color = 'Green';
  var visible = true;
  var id = null;
  var updateDisplacements = function updateDisplacements(){
    var p = this;
    p.velocity.x = sim.R2/sim.R1*p.velocity.x + (sim.deltaT/sim.R1)* p.force.x/(p.mass);
    p.velocity.y = sim.R2/sim.R1*p.velocity.y + (sim.deltaT/sim.R1)*(p.force.y - sim.gravity*mass)/(
      p.mass);
    //console.log('fx,fy ='+m.force.x+', '+m.force.y +' v ='+m.velocity.x+', '+m.velocity.y);

    // check if either mass will hit the wall
    checkWallCollision(p);

    // update the position of the masses and draw
    p.center.x += p.velocity.x *sim.deltaT;
    p.center.y += p.velocity.y *sim.deltaT;
    DrawCircle(p.center,3);
  };

  return {center:center,
    radius:radius,
    mass:mass,
    color:color,
    id:id,
    children:children,
    visible:visible,
    velocity:velocity,
    force:force,
    updateDisplacements: updateDisplacements
  };
```

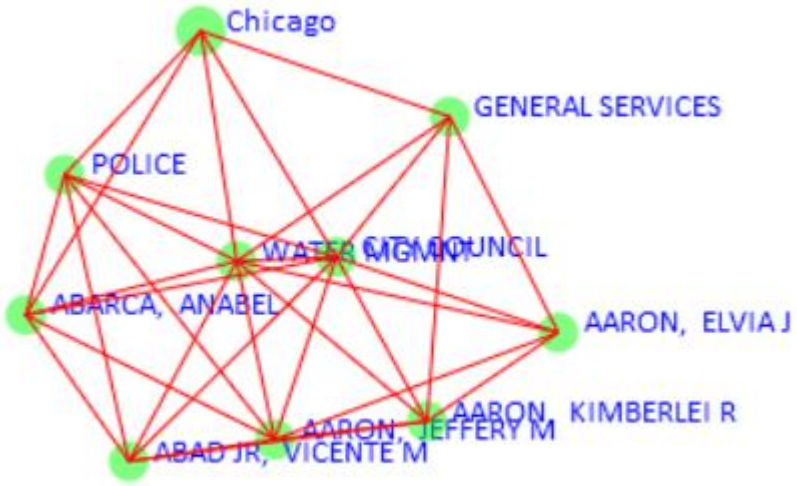
```
function Link(p0,p1){
  var nodes = [p0,p1]; // these are the particles
  var originalLength = 20;
  var color = 'Red';
  var visible = true;

  var updateSpringForces = function updateSpringForces(){
    var c0 = nodes[0].center;
    var c1 = nodes[1].center;
    var len1 = c1.minus(c0).abs();
    var forcemag = (len1 - originalLength)*sim.stiffness;
    color = 'Red';
    if(forcemag>0)color='Blue';

    var unitVec = c1.minus(c0).unit();
    if (forcemag > sim.yieldForce ) {  // if yieldForce > 0 then
      forcemag = 0;  // this throws away a little energy and damps the system
      sim.deleteThisSpring.push(this);
    }
    var f = unitVec.scale(forcemag);

    nodes[0].force = nodes[0].force.plus(f);
    nodes[1].force = nodes[1].force.minus(f);


  };

  return {nodes:nodes, color:color, visible:visible,
    originalLength:originalLength,
    updateSpringForces:updateSpringForces
    };
  }
```

List of things you need.

```
125 function detectNeighbors(){
126   var n = sim.nodes.length;
127   for(var i=0;i<n;i++){
128     for(var j=i+1;j<n;j++){
129       if(sim.nodes[i].center.distance(sim.nodes[j].center) < sim.interactionDistance){
130         if(noSpring(sim.nodes[i],sim.nodes[j])){ // check if spring exists already
131           var s = Link(sim.nodes[i],sim.nodes[j]);
132           sim.links.push(s);
133           s.color = 'Blue';
134           //s.visible = false;
135         }
136       }
137     }
138   }
139 }
140 function noSpring(p0,p1){
141   var n = sim.links.length;
142   for(var i=0;i<n;i++){ // check spring p0 to p1 or p1 to p0
143     if(sim.links[i].nodes[0]===p0 && sim.links[i].nodes[1]===p1) return false;
144     if(sim.links[i].nodes[0]===p1 && sim.links[i].nodes[1]===p0) return false;
145   }
146   return true;
147 }
```

Running physics

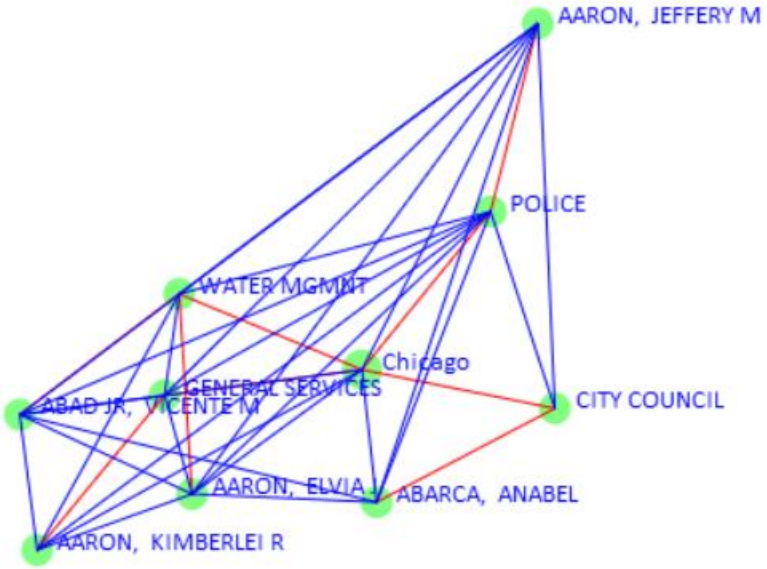
If you plot all the ‘new’ springs and the ‘Links’ you get a get something like below when you run physics.

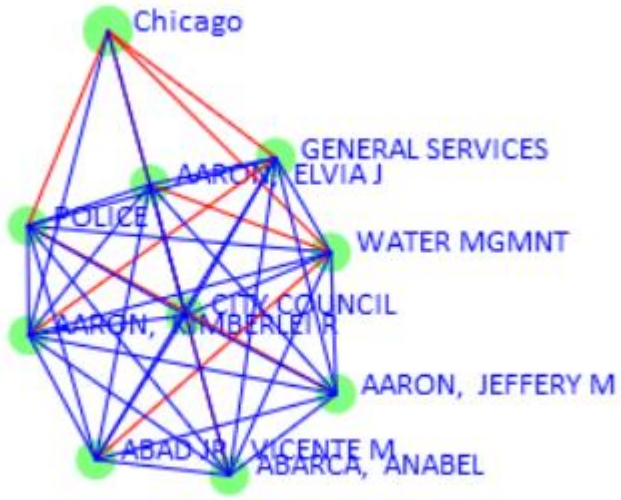


Its not bad but its not great either.

In Class Exercise 2a

Change the color of the ‘new’ springs added by physics to ‘Blue’, leaving the ‘old’ graph links in red. You should get something like this. This now makes two categories of springs red ones and blue ones.

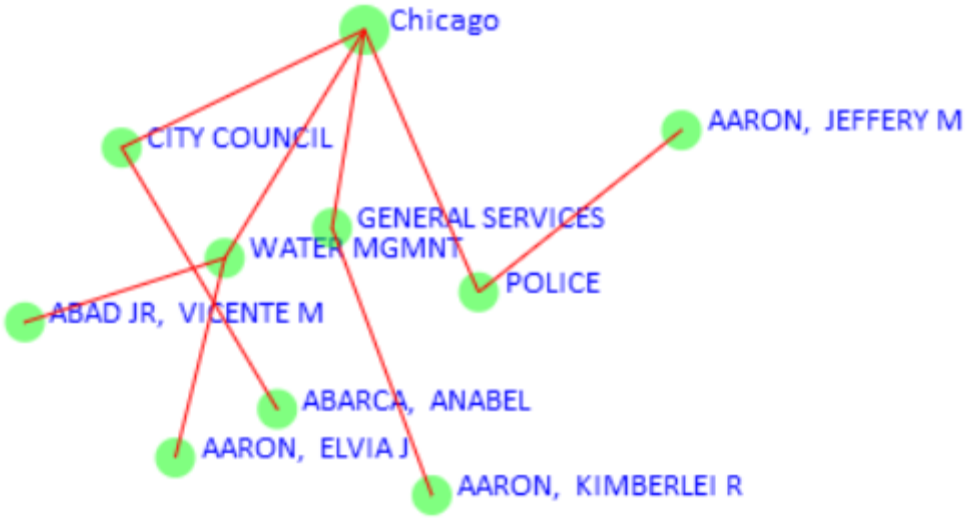




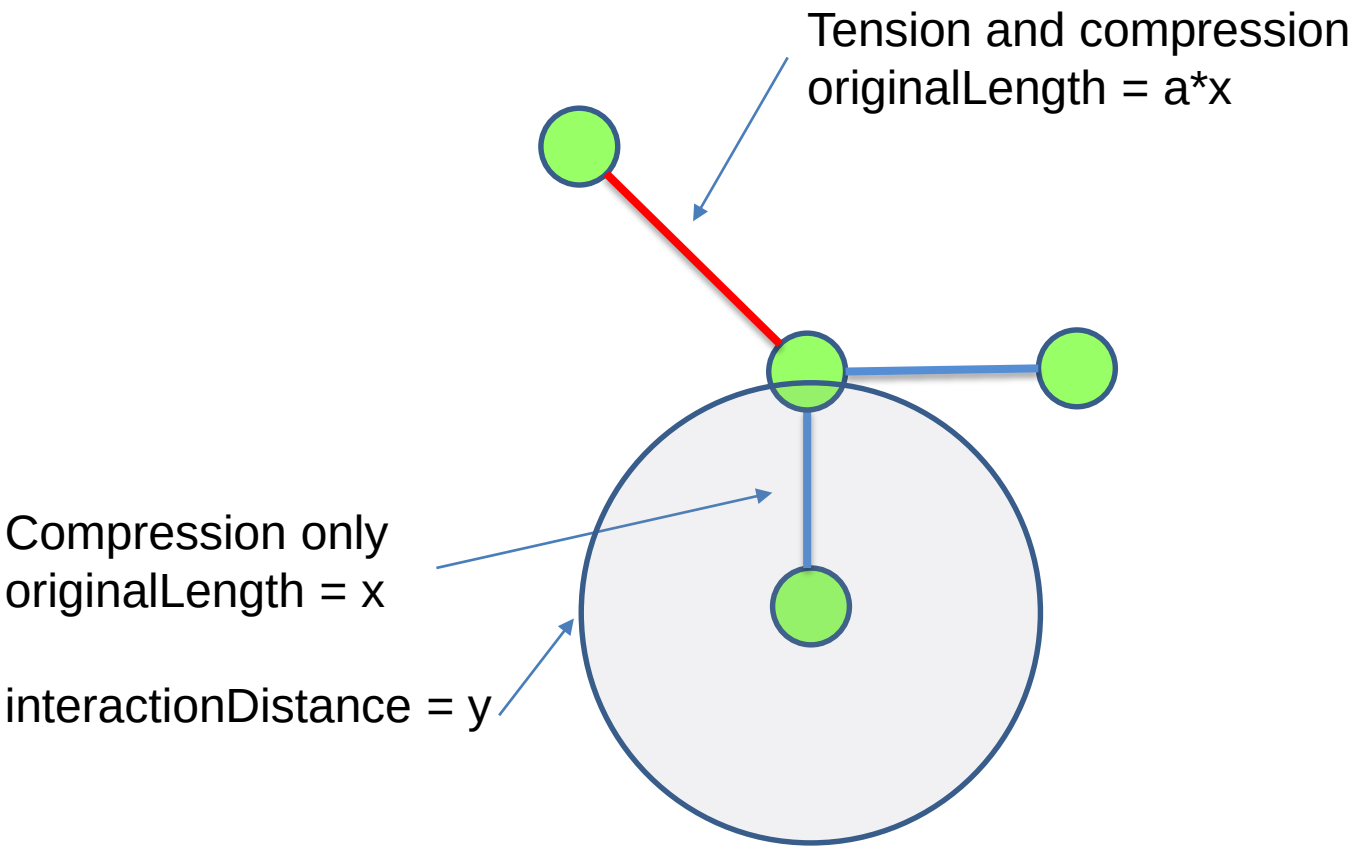
Its still not great.

In Class Exercise 2b

Instead of blue just make the ‘new’ springs invisible by turning visible = false; for the new springs.



Red and Blue Spring Behavior



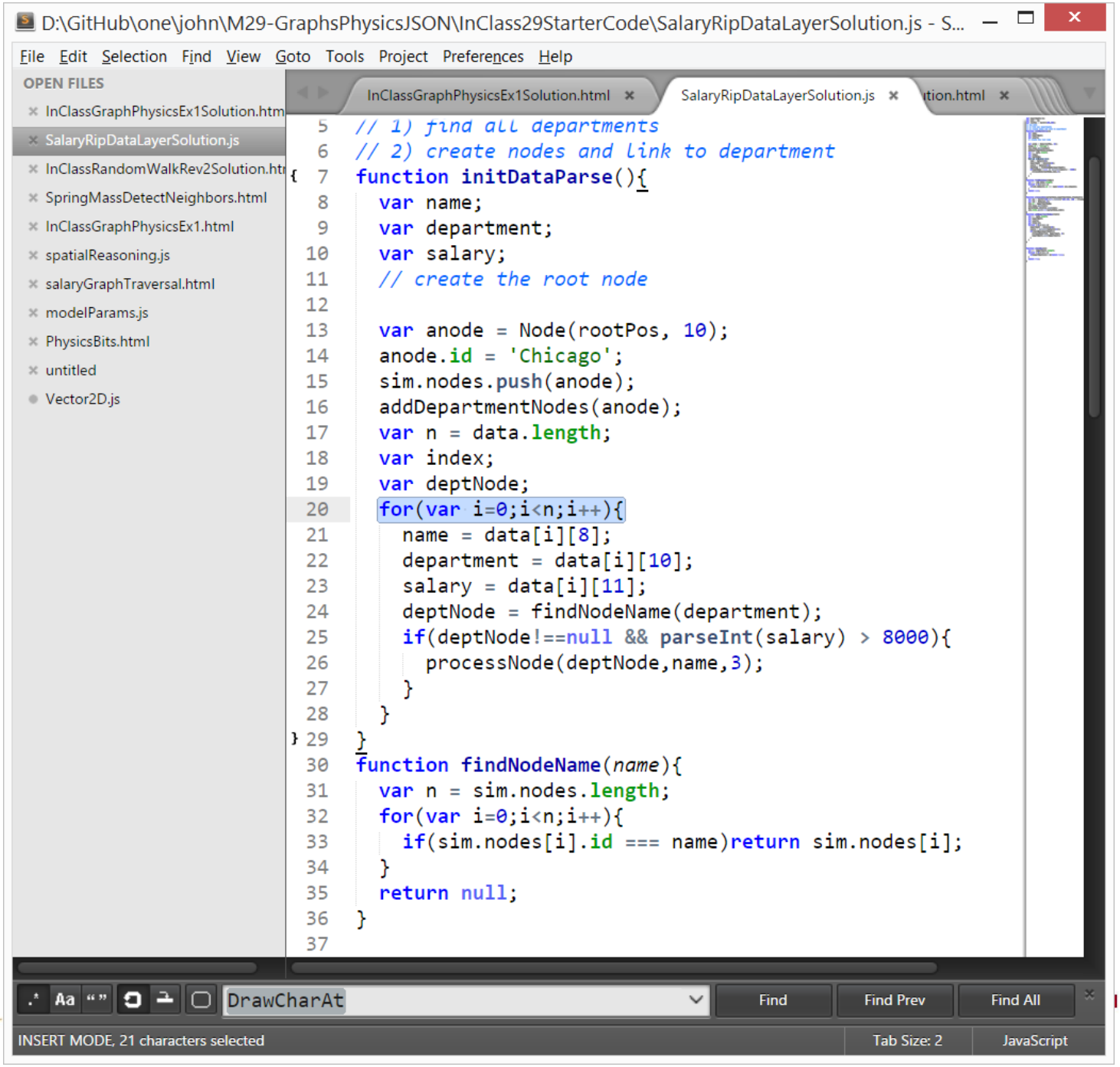
In Class Exercise 3

The springs work well to keep the particles apart but they also tend to pull the particles together when they go into tension. What might work better is springs that only work when pushing particles apart but then don't go into tension.

Modify the code so that the behavior of 'Red' springs is unchanged but 'Blue' springs exert zero force once they go into tension.

Once the code is working on the small data set try the larger data set. Change the code in SalaryRipDataLayer.js so that only 1 in 500 employees is read from the data set. (See next slide) Also turn off writing out the names on the nodes so you can see more clearly the overall layout.

Hand in your final code and a screen shots for both data sets.



Result

You should get something like this.