Mutual attraction

Hopefully, you found it helpful that we started with a simple scenario—*one object attracts another object*—and moved on to *one object attracts many objects*. However, it's likely that you are going to find yourself in a slightly more complex situation: *many objects attract each other*. In other words, every object in a given system attracts every other object in that system (except for itself).

We've really done almost all of the work for this already. Let's consider a program with an array of `Mover` objects:

```
var movers = [];


for (var i = 0; i < movers.length; i++) {
    movers[i] = new Mover(random(0.1, 2), random(width), random(height));
}


draw = function() {
    background(255, 255, 255);
    for (var i = 0; i < movers.length; i++) {
        movers[i].update();
        movers[i].display();
    }
};
```

The `draw()` function is where we need to work some magic. Currently, we're saying: "for every mover i, update and display yourself." Now what we need to say is: "for every mover i, be attracted to every other mover j, and update and display yourself."

```
for (var i = 0; i < movers.length; i++) {
```

```
    // For every Mover, check every Mover!
    for (var j = 0; j < movers.length; j++) {
        var force = movers[j].calculateAttraction(movers[i]);
        movers[i].applyForce(force);
    }

    movers[i].update();
    movers[i].display();
}
```

In the previous example, we had an `Attractor` object with a method named `calculateAttraction()`. Now, since we have movers attracting movers, all we need to do is copy that method into the `Mover` object.

```
Mover.prototype.calculateAttraction = function(m) {
  var force = PVector.sub(this.position, m.position);
  var distance = force.mag();
  distance = constrain(distance, 5.0, 25.0);
  force.normalize();

  var strength = (G * this.mass * m.mass) / (distance * distance);
  force.mult(strength);
  return force;
};
```

Of course, there's one small problem. When we are looking at every mover i and every mover j, are we OK with the times that i equals j? For example, should mover #3 attract mover #3? The answer, of course, is no. If there are five objects, we only want mover #3 to attract 0, 1, 2, and 4, skipping itself. We do, however, want to calculate and apply both the force from mover #3 on mover #1, and mover #1 on mover #3. The calculated forces will be the same for the pair, but the resulting

acceleration will be different, depending on the mass of each mover. Our attraction table should look like:

0 ⇢ 1, 2, 3, 4
1 ⇢ 0, 2, 3, 4
2 ⇢ 0, 1, 3, 4
3 ⇢ 0, 1, 2, 4

And so, we finish this example by modifying our for loop so that the inner loop avoids movers attracting themselves:

```
for (var i = 0; i < movers.length; i++) {
    for (var j = 0; j < movers.length; j++) {
       if (i !== j) {
         var force = movers[j].calculateAttraction(movers[i]);
         movers[i].applyForce(force);
       }
    }

    movers[i].update();
    movers[i].display();
}
```

Let's see it all together now:

```
var G = 1;


var Mover = function(m, x, y) {
   this.mass = m;
   this.position = new PVector(x, y);
   this.velocity = new PVector(0, 0);
   this.acceleration = new PVector(0, 0);
```

```
};

Mover.prototype.applyForce = function(force) {
  var f = PVector.div(force, this.mass);
  this.acceleration.add(f);
};

Mover.prototype.update = function() {
  this.velocity.add(this.acceleration);
  this.position.add(this.velocity);
  this.acceleration.mult(0);
};

Mover.prototype.display = function() {
  stroke(0);
  strokeWeight(2);
  fill(255, 255, 255, 127);
  ellipse(this.position.x, this.position.y, this.mass*16, this.mass*16);
};

Mover.prototype.calculateAttraction = function(m, i) {
  // Calculate direction of force
  var force = PVector.sub(this.position, m.position);
  // Distance between objects
  var distance = force.mag();
  // Limiting the distance to eliminate "extreme" results for very close or very far objects
  distance = constrain(distance, 5.0, 25.0);
```

```
    // Normalize vector (distance doesn't matter here, we just want this vector for direction
    force.normalize();
    // Calculate gravitional force magnitude
    var strength = (G * this.mass * m.mass) / (distance * distance);
    // Get force vector --> magnitude * direction
    force.mult(strength);
    return force;
};
var movers = [];
for (var i = 0; i < 5; i++) {
    movers[i] = new Mover(random(0.1, 5), random(width), random(height));
}

var draw = function() {
    background(50, 50, 50);

    for (var i = 0; i < movers.length; i++) {
        for (var j = 0; j < movers.length; j++) {
            if (i !== j) {
                var force = movers[j].calculateAttraction(movers[i]);
                movers[i].applyForce(force);
            }
        }
        movers[i].update();
        movers[i].display();
    }
};
```