Short notes on attractors

Start by creating an N number of attractors at fixed random positions. You can later on create each attractor by using a right mouse click for example.
You should have a list of attractors.

```
var attractors = [];
```

Each attractor is an object with a position and an attraction strength. You should draw these attractors in order to observe their visual result.

```
var Attractor = function(pos, s){
      this.pos = pos.copy();
      this.strength = s;

      this.draw = function(){
            ellipse(this.pos.x, this.pos.y,
                  this.strength, this.strength);
      }
}
```

As for particles, the attractors can be drawn in a loop inside the p5.js draw function:

```
attractors.forEach(function(attractor){
      attractor.draw();
});
```

Now, each particle, should update its motion regarding the all the attractors in the system. If there are three attractors, then the resulting force of these attractors will have to be computed. As known the magnitude of such forces varies with the inverse of the squared distance: when a particle is far it is barely attracted, and when it is close the force is higher. The parameterization of such forces in these simulations is usually done empirically, so you will have to discover your own constants to divide and multiply in order to attain a natural effect.

Each particle has a position (pos), a velocity (vel) and an acceleration (acc). They are updated in the update function of each particle. The acceleration will result from the attraction of all the attractors, weighted by their distances to particles. For this, you will have to compute vectors that point from one position to another, more specifically, that point from the particle to each of the attractors - which is the attraction force. A vector `ATTRACT` that points from particle `P` to attractor `A` is given by subtracting the vectors in this order: `A – P`. If using a function, it would be `.sub(A, P)` -> this is a vector that points from `P` to `A`.

Inside the particle's update function, create the appropriate acceleration:

```
// goes through each of the attractors A
attractors.forEach(function(A){
      //compute the vector that points from the particle to the
            attractor
      //this function creates a new vector att. If we did
            directly for example, A.pos.sub(this.pos), we would
            be *altering* A's position vector!!
      var att = p5.Vector.sub(A.pos, this.pos);
      //this vector should be made smaller according with the
            squared distance of the particle to the attractor.
            the distance if given by the length of this vector.
            caution: we don't want to divide the vector by
            numbers smaller than 1.0, as this would scale the
            vector up! (e.g. dividing by 0.0001 is the same as to
            multiply by 10,000). therefore a conditional clause
            should be inserted.
```

```
      var distanceSq = att.magSq(); //this is the distance
            squared already, we don't need the function
            vector.mag() that gives the distances uses more
            computational power as it uses a square root inside
            it! when the squared distance is one, so is the
            distance 1*1 = 1.
      if(distanceSq > 1){
            att.div(distanceSq);
            //now, multiply or divide the vector as needed, from
                  your experimental observations. if the
                  attractor is stronger, so is the force.
            att.mult(MAGIC_NUMBER_TO_DISCOVER*A.strength);
            //this vector should be added to the acceleration
            this.acc.add(att);
      }
});
```

Now that the resulting acceleration is set regarding all the attractors, we can proceed with the the customary update on the particle:

```
this.vel.add(this.acc);
this.pos.add(this.vel);
```

Something extremely important, do not forget to reset your acceleration after you used it! The acceleration is not cumulative through time, it is its own at any given point in time.

```
this.acc.mult(0);
```

Also, you might need to limit your velocity, as particles tend to speed beyond proper simulation steps when they get too close to attractors or when there are too many attractors.

```
this.vel.limit(ANOTHER_MAGIC_NUMBER);
```