A single particle

Before we can create an entire `ParticleSystem`, we have to create an object that will describe a single particle. The good news: we've done this already. Our `Mover` object from the Forces section serves as the perfect template. For us, a particle is an independent body that moves about the screen. It has `location`, `velocity`, and `acceleration`, a constructor to initialize those variables, and functions to `display()` itself and `update()` its location.

```
// A simple Particle object
var Particle = function(position) {
  this.acceleration = new PVector();
  this.velocity = new PVector();
  this.position = position.get();
};

Particle.prototype.update = function(){
  this.velocity.add(this.acceleration);
  this.position.add(this.velocity);
};

Particle.prototype.display = function() {
  stroke(0, 0, 0);
  fill(175, 175, 175);
  ellipse(this.position.x, this.position.y, 8, 8);
};
```

This is about as simple as a particle can get. From here, we could take our particle in several directions. We could add an `applyForce()` method to affect the particle’s behaviour (we’ll do precisely this in a future example). We could add variables to describe colour and shape, or use `image()` to draw the particle. For now, however, let’s focus on adding just one additional detail: ***lifespan***.

Typical particle systems involve something called an ***emitter***. The emitter is the source of the particles and controls the initial settings for the particles, location, velocity, etc. An emitter might emit a single burst of particles, or a continuous stream of particles, or both. The point is that for a typical implementation such as this, a particle is born at the emitter but does not live forever. If it were to live forever, our program would eventually grind to a halt as the number of particles increased to an unwieldy number over time. As new particles are born, we need old particles to die. This creates the illusion of an infinite stream of particles, and the performance of our program does not suffer.

There are many different ways to decide when a particle dies. For example, it could come into contact with another object, or it could simply leave the screen. For our first `Particle` object, however, we’re simply going to add a `timeToLive` property. It will act as a timer, counting down from 255 to 0, at which point we'll consider the particle to be "dead." And so we expand the `Particle` object as follows:

```
// A simple Particle object
var Particle = function(position) {
  this.acceleration = new PVector();
  this.velocity = new PVector();
  this.position = position.get();
  this.timeToLive = 255;
};


Particle.prototype.update = function(){
  this.velocity.add(this.acceleration);
  this.position.add(this.velocity);
  this.timeToLive -= 2;
};

Particle.prototype.display = function() {
  stroke(255, 255, 255, this.timeToLive);
  fill(127, 127, 127, this.timeToLive);
  ellipse(this.position.x, this.position.y, 8, 8);
};
```

The reason we chose to start the `timeToLive` at 255 and count down to 0 is for convenience. With those values, we can use `timeToLive` as the alpha transparency for the ellipse as well. When the particle is “dead” it will also have faded away onscreen.

With the addition of the `timeToLive` property, we’ll also need one additional method—a function that can be queried (for a true or false answer) as to whether the particle is alive or dead. This will come in handy when we are writing the `ParticleSystem` object, whose task will be to manage the list of particles themselves. Writing this function is pretty easy; we just need to check and see if the value of `timeToLive` is less than 0. If it is we return true, if not we return false.

```
Particle.prototype.isDead = function() {
  if (this.timeToLive < 0) {
      return true;
  } else {
    return false;
  }
};
```

Before we get to the next step of making many particles, it’s worth taking a moment to make sure our particle works correctly and create a sketch with one single `Particle` object. Here is the full code below, with two small additions. We add a convenience method called `run()` that simply calls both `update()` and `display()` for us. In addition, we give the particle a random initial velocity as well as a downward acceleration (to simulate gravity).

```
// A single Particle object
var Particle = function(position) {
  this.acceleration = new PVector(0, 0.05);
  this.velocity = new PVector(random(-1, 1), random(-1, 0));
  this.position = position.get();
  this.timeToLive = 255.0;
};

Particle.prototype.run = function() {
  this.update();
  this.display();
};

Particle.prototype.update = function(){
  this.velocity.add(this.acceleration);
  this.position.add(this.velocity);
  this.timeToLive -= 2;
};

Particle.prototype.display = function() {
  stroke(255, 255, 255, this.timeToLive);
  strokeWeight(2);
  fill(127, 127, 127, this.timeToLive);
  ellipse(this.position.x, this.position.y, 12, 12);
};


// Is the particle still useful?
Particle.prototype.isDead = function() {
  if (this.timeToLive < 0) {
      return true;
  } else {
      return false;
  }
};


var particle = new Particle(new PVector(width/2, 20));


var draw = function() {
  background(0, 116, 194);


  particle.run();
  if (particle.isDead()) {
    particle = new Particle(new PVector(width/2, 20));
  }
};
```

Now that we have an object to describe a single particle, we're ready for the next big step. How do we keep track of many particles, when we can't ensure exactly how many particles we might have at any given time?