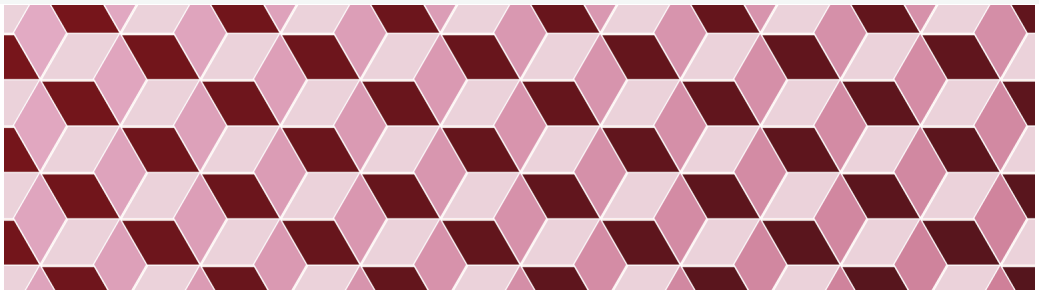


15-104 Introduction to Computing for Creative Practice

Fall 2021

20 Particles (Physics of Particle Motion)

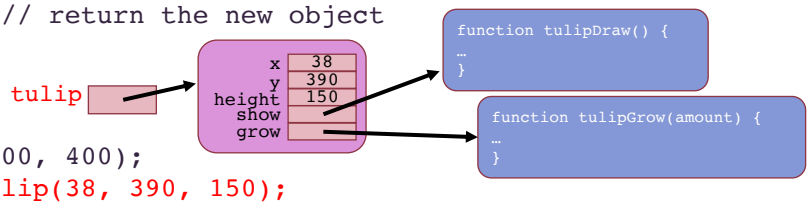
Instructor: Tom Cortina, tcortina@cs.cmu.edu, GHC 4117, 412-268-3514

1

Recall objects…

```
var tulip;
function tulipDraw() { ... }
function tulipGrow(amount) { ... }
function makeTulip(tx, ty, th) {
    var tulip = {x: tx, y: ty, height: th,
                 show: tulipDraw, grow: tulipGrow};  // new object
    return tulip; // return the new object
};

function setup() {
    createCanvas(400, 400);
    tulip = makeTulip(38, 390, 150);
    frameRate(5);
}
```



2

Classes vs. Objects

- A class in an object-oriented programming language is a definition that represents a class of objects, defining its properties (fields) and behaviors (methods).
- An object is an instance (member) of a class.
  - Each object instance has the same properties and behaviors, but the values for its properties will depend on the specific object.
- Javascript does not have a formal way to define classes (like Java does), but when we create an object, we are defining a member of a class. If each object is defined using the same properties and behaviors, then all of these objects belong to the same class.

3

Modeling a Particle System

- Each particle is represented as an object with the following fields and behaviors:
  - x – its current horizontal location
  - y – its current vertical location
  - dx – its current horizontal velocity (its change in x per frame, "delta x")
  - dy – its current vertical velocity (its change in y per frame, "delta y")
  - age - the number of frames (time steps) that this particle exists
  - stepFunction – behavior to update the particle for the next time step
  - drawFunction – behavior to draw the particle in its updated location

These fields will hold references to functions!

4

Particle methods

```
function particleStep() {
    this.age++;
    this.x += this.dx;
    this.y += this.dy;
}
function particleDraw() {
    point(this.x, this.y);
}
function makeParticle(px, py, pdx, pdy) {        // constructor
    p = {x: px, y: py, dx: pdx, dy: pdy, age: 0,
         stepFunction: particleStep, drawFunction: particleDraw }
    return p;
}
```

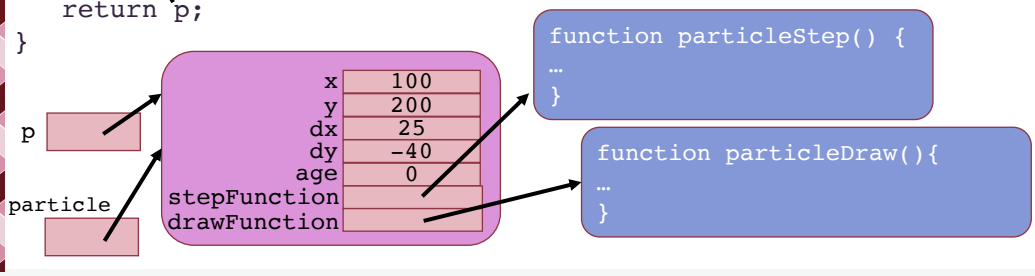
Whenever a method is called from an object, a reference to the object itself is implicitly passed to the method, represented by `this`. As an example, when a specific particle object calls its `particleDraw` method, it draws a point not at just any (x,y) but at the location of this object's x and y: `(this.x, this.y)`.

5

The Particle Object

```
particle = makeParticle(100, 200, 25, -40);

function makeParticle(px, py, dpx, dpy) {
   p =    {x: px, y: py, dx: pdx, dy: pdy, age: 0,
           stepFunction: particleStep, drawFunction: particleDraw }
   return p;
}
```



6

Drawing 100 particles

```
var np = 100;      // how many particles

function particleStep() {  // as before }
function particleDraw() {  // as before }
function makeParticle(px, py, pdx, pdy) {  // as before }

var particles = [];

function setup() {
    createCanvas(400, 400);
    for (var i = 0; i < np; i++) {
        var p = makeParticle(200,200,random(-50,50),random(-50,50));
        particles.push(p);
    }
    frameRate(10);
}
```

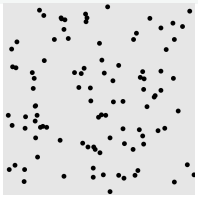
7

Drawing 100 particles (cont'd)

```
function draw() {
    background(230);
    stroke(0);
    strokeWeight(10);
    for (var i = 0; i < np; i++) {
        var p = particles[i];  // p is the ith particle
        // for each particle p, update its position (step)
        // and then draw it
        p.stepFunction();     // particles[i].stepFunction();
        p.drawFunction();
    }
}
```

8

Adding Bounce



```
function particleStep() {
    this.age++;
    this.x += this.dx;
    this.y += this.dy;
    if (this.x > width) {    // bounce off right wall
        this.x = width - (this.x - width); this.dx = -this.dx;
    } else if (this.x < 0) { // bounce off left wall
        this.x = -this.x; this.dx = -this.dx;
    }
    if (this.y > height) {   // bounce off bottom
        this.y = height - (this.y - height); this.dy = -this.dy;
    } else if (this.y < 0) { // bounce off top
        this.y = -this.y; this.dy = -this.dy;
    }
}
```

9

Spring and Gravity

- When a particle bounces off a wall, some of its energy is absorbed due to its spring.
  - Example: A spring coefficient of 0.9 means that a particle retains 90% of its velocity when it bounces off the wall.
- Gravity will pull the object downward over time by increasing its vertical velocity by some some amount (adding to velocity is acceleration).
- Add global variables:

```
var gravity = 0.1;
var spring = 0.9;
```

10

Adding Spring and Gravity

```
function particleStep() {
    this.age++;
    this.x += this.dx;
    this.y += this.dy;
    if (this.x > width) {    // bounce off right wall
        this.x = width - (this.x - width);
        this.dx = -this.dx * spring;
    } else if (this.x < 0) { // bounce off left wall
        this.x = -this.x;
        this.dx = -this.dx * spring;
    }
    ...
    this.dy = this.dy + gravity;
}
```

11

Drag

- The faster a particle moves, the more drag it will have on its surface (e.g. due to air friction).
- Drag is proportional to the velocity squared which is the sum of the squares of dx and dy
- The ratio of the old velocity to the new velocity goes up with velocity squared but can never be so high that the velocity reverses, so we limit this ratio to 1

12

Adding Drag

```
function particleStep() {
    ...
    this.dy = this.dy + gravity;
    var vs = Math.pow(this.dx, 2) + Math.pow(this.dy, 2);
    var d = vs * drag;
    d = min(d, 1);
    // scale dx and dy to include drag effect
    this.dx *= (1 - d);
    this.dy *= (1 - d);
}
```

13

Particle Decay

- We can simulate the decay of a particle by removing it from the array of particles after each reaches a set "age".
- Algorithm:
  As we draw each particle, check its age. If it's still ok, append it to a new array. Then once all particles are examined, use the new array as your array of particles.

14

Particle Decay

```
function draw() {
    background(230);
    stroke(0);
    strokeWeight(10);
    if (mouseIsPressed) {
        var newp = makeParticle(mouseX, mouseY,
                                random(-10, 10),
                                random(-10, 0));
        particles.push(newp);
    }
    // cont'd on next slide
```

Bonus feature:
Add more particles when you press and hold the mouse!

15

Particle Decay (cont'd)

```
    // draw cont'd
    newParticles = [];
    for (var i = 0; i < particles.length; i++) {
        var p = particles[i];
        p.stepFunction();
        p.drawFunction();
        if (p.age < 200) {
            newParticles.push(p);
        }
    }
    particles = newParticles;
}
```

16