PROCESSING
COOKBOOK
by Angelos Floros

EVENTFULL PARAMETERES

ANIMATION, INTERACTIVE

creative
scripting
m e a l s

03

exercises

sketches

FALLING BALLS

FALLING WORDS

ANIMATED POINTS

BOUNCING BALLS

BOUNCING BALLS PHYSICS

CIRCLE COLLISION

BOUNCING BUBBLES

FOLLOW MOUSE

CONSTRAIN

SCROLLBAR

MOVING ON CURVES

VECTOR MATH

FOLLOW1

FOLLOW2

FOLLOW3

REACH2

REACH3

REFLECTION1

TICKLE

FONT PHYSICS

Falling Balls

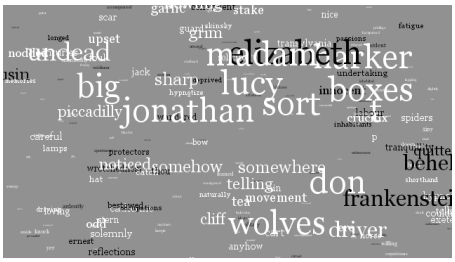
CODE

```
ArrayList<Ball> balls;
int ballWidth = 48;

void setup() {
  size(640, 360);
  noStroke();
  balls = new ArrayList<Ball>();  // Create an empty ArrayList stores
Ball objects)
  balls.add(new Ball(width/2, 0, ballWidth)); // Start by adding one
element
}

void draw() {
  background(255);
  for (int i = balls.size()-1; i >= 0; i--) {
    // An ArrayList doesn't know what it is storing so we have to cast
the object coming out
    Ball ball = balls.get(i);
    ball.move();
    ball.display();
    if (ball.finished()) { // Items can be deleted with remove()
      balls.remove(i);
    }
  }
}
```

```
/**
 * ArrayList of objects
 * by Daniel Shiffman.
 *
 * This example demonstrates how to
use a Java ArrayList to store
 * a variable number of objects.  Items
can be added and removed
 * from the ArrayList.
 *
 * Click the mouse to add bouncing
balls.
 */

  // With an array, we say balls.length,
with an ArrayList, we say balls.size()
  // The length of an ArrayList is
dynamic
  // Notice how we are looping through
the ArrayList backwards
  // This is because we are deleting
elements from the list
```

```
// Adds a new ball to the ArrayList
(by default to the end)

void mousePressed() {    balls.
add(new Ball(mouseX, mouseY,
ballWidth));
}

// BALL OBJECT
//Simple bouncing ball class

class Ball {
  float x;
  float y;
  float speed;
  float gravity;
  float w;
  float life = 255;

  Ball(float tempX, float tempY,
float tempW) {
    x = tempX;
    y = tempY;
    w = tempW;
    speed = 0;
    gravity = 0.1;
  }

  void move() {
  // Add gravity to speed
  speed = speed + gravity;
  // Add speed to y location
  y = y + speed;
  // If square reaches the bottom
  // Reverse speed
  if (y > height) {
    speed = speed * -0.8;
    y = height;
  }
 }
  boolean finished() {
  life--;  // Balls fade out
  if (life < 0) {
    return true;
  } else {
    return false;
  }
 }

 void display() {
  fill(0,life);    // Display the circle
  //stroke(0,life);
  ellipse(x,y,w,w);
 }
}
```

CODE




```
/**
 * HashMap example
 * by Daniel Shiffman.
 *
 * This example demonstrates how to use a HashMap to store
 * a collection of objects referenced by a key. This is much like an array,
 * only instead of accessing elements with a numeric index, we use a String.
 * If you are familiar with associative arrays from other languages,
 * this is the same idea.
 *
 * A simpler example is CountingStrings which uses IntHash instead of
 * HashMap.  The Processing classes IntHash, FloatHash, and StringHash
 * offer a simpler way of pairing Strings with numbers or other Strings.
 * Here we use a HashMap because we want to pair a String with a custom
 * object, in this case a "Word" object that stores two numbers.
 *
 * In this example, words that appear in one book (Dracula) only are colored white
 * while words the other (Frankenstein) are colored black.
 */

// The next line is needed if running in JavaScript Mode with Processing.js
/* @pjs font="Georgia.ttf"; */
```

Falling Words

```
HashMap<String, Word> words;  // HashMap object

void setup() {
  size(640, 360);

  // Create the HashMap
  words = new HashMap<String, Word>();

  // Load two files
  loadFile("dracula.txt");
  loadFile("frankenstein.txt");

  // Create the font
  textFont(createFont("Georgia", 24));
}

void draw() {
  background(126);

  // Show words
  for (Word w : words.values()) {
    if (w.qualify()) {
      w.display();
      w.move();
    }
  }
}

// Load a file
void loadFile(String filename) {
  String[] lines = loadStrings(filename);
  String allText = join(lines, " ").toLowerCase();
  String[] tokens = splitTokens(allText, " ,.?!:;[]-\"'");

  for (String s : tokens) {
    // Is the word in the HashMap
    if (words.containsKey(s)) {
      // Get the word object and increase the count
      // We access objects from a HashMap via its key, the String
      Word w = words.get(s);
      // Which book am I loading?
      if (filename.contains("dracula")) {
        w.incrementDracula();
      }
      else if (filename.contains("frankenstein")) {
        w.incrementFranken();
      }
    }
    else {
      // Otherwise make a new word
      Word w = new Word(s);
      // And add to the HashMap put() takes two arguments, "key" and "value"
      // The key for us is the String and the value is the Word object
      words.put(s, w);
      if (filename.contains("dracula")) {
        w.incrementDracula();
      } else if (filename.contains("frankenstein")) {
        w.incrementFranken();
      }
    }
  }
}

class Word {
```

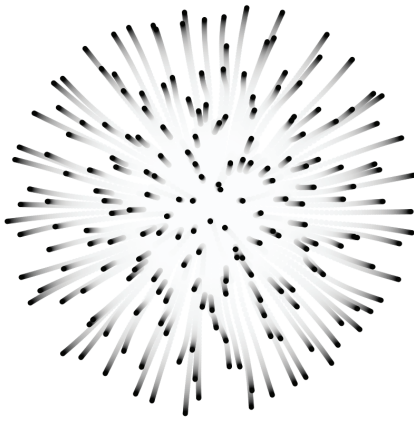
Animated Points

http://www.generative-gestaltung.de

DESCRIPTION

Distribute nodes on the display by letting them repel each other.

CODE



```
Node[] nodes = new Node[200]; // an array for the nodes

void setup() {
  size(800, 800);
  background(255);
  smooth();
  noStroke();

  for (int i = 0 ; i < nodes.length; i++) {  // init nodes
    nodes[i] = new Node(width/2+random(-1, 1), height/2+random(-1, 1));
    nodes[i].setBoundary(5, 5, width-5, height-5);
  }
}

void draw() {
  fill(255, 20);
  rect(0, 0, width, height);
  // let all nodes repel each other
  for (int i = 0 ; i < nodes.length; i++) {
    nodes[i].attract(nodes);
  }
  // apply velocity vector and update position
  for (int i = 0 ; i < nodes.length; i++) {
    nodes[i].update();
  }
  // draw node
  fill(0);
  for (int i = 0 ; i < nodes.length; i++) {
    ellipse(nodes[i].x, nodes[i].y, 10, 10);
  }
}

void keyPressed(){
  if(key=='r' || key=='R') {
    background(255);
    for (int i = 0 ; i < nodes.length; i++) {
      nodes[i].set(width/2+random(-5, 5), height/2+random(-5, 5), 0);
    }
  }
}

class Node extends PVector {

  // ------   properties ------
  // if needed, an ID for the node
  String id = "";
  float diameter = 0;

  float minX = -Float.MAX_VALUE;
  float maxX = Float.MAX_VALUE;
  float minY = -Float.MAX_VALUE;
  float maxY = Float.MAX_VALUE;
  float minZ = -Float.MAX_VALUE;
  float maxZ = Float.MAX_VALUE;
```

```
// animatedPoints.pde
// Node.pde
//
// Generative Gestaltung, ISBN: 978-3-87439-759-9
// First Edition, Hermann Schmidt, Mainz, 2009
// Hartmut Bohnacker, Benedikt Gross, Julia Laub, Claudius Lazzeroni
// Copyright 2009 Hartmut Bohnacker, Benedikt Gross, Julia Laub, Claudius Lazzeroni
//
// http://www.generative-gestaltung.de
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

/**
 * distribute nodes on the display by letting them repel each other
 *
 * KEYS
 * r          : reset positions
 * s          : save png
 */
```

```
  PVector velocity = new PVector();
  PVector pVelocity = new PVector();
  float maxVelocity = 10;

  float damping = 0.5f;
  // radius of impact
  float radius = 200;
  // strength: positive for attraction, negative for repulsion (default for Nodes)
  float strength = -1;
  // parameter that influences the form of the function
  float ramp = 1.0f;


  // ------ constructors ------
  Node() {
  }

  Node(float theX, float theY) {
    x = theX;
    y = theY;
  }

  Node(float theX, float theY, float theZ) {
    x = theX;
    y = theY;
    z = theZ;
  }

  Node(PVector theVector) {
    x = theVector.x;
    y = theVector.y;
    z = theVector.z;
  }

  // ------ rotate position around origin ------
  void rotateX(float theAngle) {
    float newy = y * cos(theAngle) - z * sin(theAngle);
    float newz = y * sin(theAngle) + z * cos(theAngle);
    y = newy;
    z = newz;
  }

  void rotateY(float theAngle) {
    float newx = x * cos(-theAngle) - z * sin(-theAngle);
    float newz = x * sin(-theAngle) + z * cos(-theAngle);
    x = newx;
    z = newz;
  }

  void rotateZ(float theAngle) {
    float newx = x * cos(theAngle) - y * sin(theAngle);
    float newy = x * sin(theAngle) + y * cos(theAngle);
    x = newx;
    y = newy;
  }

  // ------ calculate attraction ------
```

```
  void attract(Node[] theNodes) {
    // attraction or repulsion part
    for (int i = 0; i < theNodes.length; i++) {
      Node otherNode = theNodes[i];
      // stop when empty
      if (otherNode == null) break;
      // not with itself
      if (otherNode == this) continue;

      this.attract(otherNode);
    }
  }

  void attract(Node theNode) {
    float d = PVector.dist(this, theNode);

    if (d > 0 && d < radius) {
      float s = pow(d / radius, 1 / ramp);
      float f = s * 9 * strength * (1 / (s + 1) + ((s - 3) / 4)) / d;
      PVector df = PVector.sub(this, theNode);
      df.mult(f);

      theNode.velocity.x += df.x;
      theNode.velocity.y += df.y;
      theNode.velocity.z += df.z;
    }
  }

  // ------ update positions ------
  void update() {
    update(false, false, false);
  }

  void update(boolean theLockX, boolean theLockY, boolean theLockZ)
{

    velocity.limit(maxVelocity);
    pVelocity = velocity.get();

    if (!theLockX) x += velocity.x;
    if (!theLockY) y += velocity.y;
    if (!theLockZ) z += velocity.z;

    if (x < minX) {
      x = minX - (x - minX);
      velocity.x = -velocity.x;
    }
    if (x > maxX) {
      x = maxX - (x - maxX);
      velocity.x = -velocity.x;
    }

    if (y < minY) {
      y = minY - (y - minY);
      velocity.y = -velocity.y;
    }
```

```
  if (y > maxY) {
    y = maxY - (y - maxY);
    velocity.y = -velocity.y;
  }

  if (z < minZ) {
    z = minZ - (z - minZ);
    velocity.z = -velocity.z;
  }
  if (z > maxZ) {
    z = maxZ - (z - maxZ);
    velocity.z = -velocity.z;
  }

  velocity.mult(1 - damping);
}

// ------ getters and setters ------
String getID() {
  return id;
}

void setID(String theID) {
  this.id = theID;
}

float getDiameter() {
  return diameter;
}

void setDiameter(float theDiameter) {
  this.diameter = theDiameter;
}

void setBoundary(float theMinX, float theMinY, float theMinZ,
float theMaxX, float theMaxY, float theMaxZ) {
  this.minX = theMinX;
  this.maxX = theMaxX;
  this.minY = theMinY;
  this.maxY = theMaxY;
  this.minZ = theMinZ;
  this.maxZ = theMaxZ;
}

void setBoundary(float theMinX, float theMinY, float theMaxX,
float theMaxY) {
  this.minX = theMinX;
  this.maxX = theMaxX;
  this.minY = theMinY;
  this.maxY = theMaxY;
}

float getMinX() {
  return minX;
}
```

```
void setMinX(float theMinX) {
  this.minX = theMinX;
}

float getMaxX() {
  return maxX;
}

void setMaxX(float theMaxX) {
  this.maxX = theMaxX;
}

float getMinY() {
  return minY;
}

void setMinY(float theMinY) {
  this.minY = theMinY;
}

float getMaxY() {
  return maxY;
}

void setMaxY(float theMaxY) {
  this.maxY = theMaxY;
}

float getMinZ() {
  return minZ;
}

void setMinZ(float theMinZ) {
  this.minZ = theMinZ;
}

float getMaxZ() {
  return maxZ;
}

void setMaxZ(float theMaxZ) {
  this.maxZ = theMaxZ;
}

PVector getVelocity() {
  return velocity;
}

void setVelocity(PVector theVelocity) {
  this.velocity = theVelocity;
}

float getMaxVelocity() {
  return maxVelocity;
}
```

```
  void setMaxVelocity(float theMaxVelocity) {
    this.maxVelocity = theMaxVelocity;
  }

  float getDamping() {
    return damping;
  }

  void setDamping(float theDamping) {
    this.damping = theDamping;
  }

  float getRadius() {
    return radius;
  }

  void setRadius(float theRadius) {
    this.radius = theRadius;
  }

  float getStrength() {
    return strength;
  }

  void setStrength(float theStrength) {
    this.strength = theStrength;
  }

  float getRamp() {
    return ramp;
  }

  void setRamp(float theRamp) {
    this.ramp = theRamp;
  }

}
```

Bouncing Ball

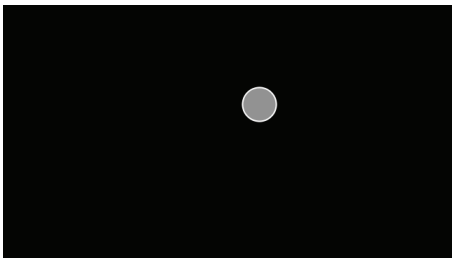
CODE

```
/**
 * Bounce.
 *
 * When the shape hits the edge of the window, it reverses its direction.
 */
```

```
int rad = 60;        // Width of the shape
float xpos, ypos;    // Starting position of shape

float xspeed = 2.8;  // Speed of the shape
float yspeed = 2.2;  // Speed of the shape

int xdirection = 1;  // Left or Right
int ydirection = 1;  // Top to Bottom


void setup()
{
  size(640, 360);
  noStroke();
  frameRate(30);
  ellipseMode(RADIUS);
  // Set the starting position of the shape
  xpos = width/2;
  ypos = height/2;
}

void draw()
{
  background(102);

  // Update the position of the shape
  xpos = xpos + ( xspeed * xdirection );
  ypos = ypos + ( yspeed * ydirection );

  // Test to see if the shape exceeds the boundaries of the screen
  // If it does, reverse its direction by multiplying by -1
  if (xpos > width-rad || xpos < rad) {
    xdirection *= -1;
  }
  if (ypos > height-rad || ypos < rad) {
    ydirection *= -1;
  }

  // Draw the shape
  ellipse(xpos, ypos, rad, rad);
}
```

Bouncing Ball Physics

CODE

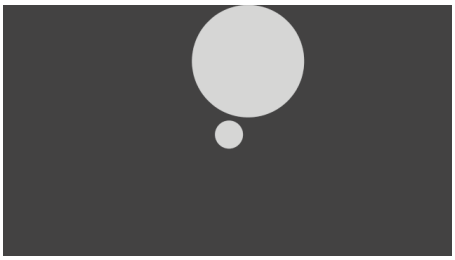


```
PVector location;  // Location of shape
PVector velocity;  // Velocity of shape
PVector gravity;   // Gravity acts at the shape's acceleration

void setup() {
  size(640,360);
  smooth();
  location = new PVector(100,100);
  velocity = new PVector(1.5,2.1);
  gravity = new PVector(0,0.2);

}

void draw() {
  background(0);

  // Add velocity to the location.
  location.add(velocity);
  // Add gravity to velocity
  velocity.add(gravity);

  // Bounce off edges
  if ((location.x > width) || (location.x < 0)) {
    velocity.x = velocity.x * -1;
  }
  if (location.y > height) {
    // We're reducing velocity ever so slightly
    // when it hits the bottom of the window
    velocity.y = velocity.y * -0.95;
    location.y = height;
  }

  // Display circle at location vector
  stroke(255);
  strokeWeight(2);
  fill(127);
  ellipse(location.x,location.y,48,48);
}
```

```
/**
 * Bouncing Ball with Vectors
 * by Daniel Shiffman.
 *
 * Demonstration of using vectors to
control motion of body
 * This example is not object-oriented
 * See AccelerationWithVectors for an
example of how to simulate motion
using vectors in an object
 */
```

Circle Collision

CODE



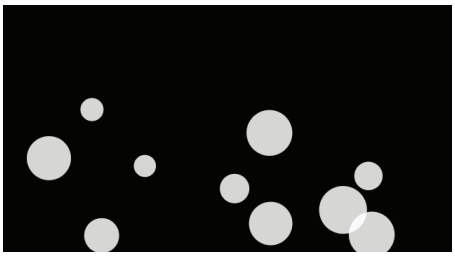
```
Ball[] balls =  {
  new Ball(100, 400, 20),
  new Ball(700, 400, 80)
};

void setup() {
  size(640, 360);
}

void draw() {
  background(51);
  for (Ball b : balls) {
    b.update();
    b.display();
    b.checkBoundaryCollision();
  }
  balls[0].checkCollision(balls[1]);
}

class Ball {
  PVector position;
  PVector velocity;

  float r, m;

  Ball(float x, float y, float r_) {
    position = new PVector(x, y);
    velocity = PVector.random2D();
    velocity.mult(3);
    r = r_;
    m = r*.1;
  }

  void update() {
    position.add(velocity);
  }

  void checkBoundaryCollision() {
    if (position.x > width-r) {
      position.x = width-r;
      velocity.x *= -1;
    }
    else if (position.x < r) {
      position.x = r;
      velocity.x *= -1;
    }
    else if (position.y > height-r) {
      position.y = height-r;
      velocity.y *= -1;
    }
    else if (position.y < r) {
      position.y = r;
      velocity.y *= -1;
    }
  }

  void checkCollision(Ball other) {
    // get distances between the balls components
    PVector bVect = PVector.sub(other.position, position);
    // calculate magnitude of the vector separating the balls
    float bVectMag = bVect.mag();
    if (bVectMag < r + other.r) {
      // get angle of bVect
      float theta  = bVect.heading();
```

```
/**
 * Circle Collision with Swapping
Velocities
 * by Ira Greenberg.
 *
 * Based on Keith Peter's Solution in
 * Foundation Actionscript Animation:
Making Things Move!
 */

    /* bTemp will hold rotated ball
positions. You
     just need to worry about bTemp[1]
position*/

      /* this ball's position is relative to
the other
       so you can use the vector between
them (bVect) as the
       reference point in the rotation
expressions.
       bTemp[0].position.x and bTemp[0].
position.y will initialize
       automatically to 0.0, which is what
you want
       since b[1] will rotate around b[0] */
```

```
    // precalculate trig values
    float sine = sin(theta);
    float cosine = cos(theta);

    PVector[] bTemp = { new PVector(), new PVector()};

      bTemp[1].x  = cosine * bVect.x + sine * bVect.y;
    bTemp[1].y  = cosine * bVect.y - sine * bVect.x;

    // rotate Temporary velocities
    PVector[] vTemp = { new PVector(), new PVector()};

    vTemp[0].x  = cosine * velocity.x + sine * velocity.y;
    vTemp[0].y  = cosine * velocity.y - sine * velocity.x;
    vTemp[1].x  = cosine * other.velocity.x + sine * other.velocity.y;
    vTemp[1].y  = cosine * other.velocity.y - sine * other.velocity.x;

    /* Now that velocities are rotated, you can use 1D conservation of
momentum equations to calculate the final velocity along the x-axis.
*/
    PVector[] vFinal = {
      new PVector(), new PVector()
      };

    // final rotated velocity for b[0]
    vFinal[0].x = ((m - other.m) * vTemp[0].x + 2 * other.m *
vTemp[1].x) / (m + other.m);
    vFinal[0].y = vTemp[0].y;

    // final rotated velocity for b[0]
    vFinal[1].x = ((other.m - m) * vTemp[1].x + 2 * m * vTemp[0].x)
/ (m + other.m);
    vFinal[1].y = vTemp[1].y;

    // hack to avoid clumping
    bTemp[0].x += vFinal[0].x;
    bTemp[1].x += vFinal[1].x;

    /* Rotate ball positions and velocities back
     Reverse signs in trig expressions to rotate
     in the opposite direction */
    // rotate balls
    PVector[] bFinal = {
      new PVector(), new PVector()
      };

    bFinal[0].x = cosine * bTemp[0].x - sine * bTemp[0].y;
    bFinal[0].y = cosine * bTemp[0].y + sine * bTemp[0].x;
    bFinal[1].x = cosine * bTemp[1].x - sine * bTemp[1].y;
    bFinal[1].y = cosine * bTemp[1].y + sine * bTemp[1].x;

    // update balls to screen position
    other.position.x = position.x + bFinal[1].x;
    other.position.y = position.y + bFinal[1].y;

    position.add(bFinal[0]);

    // update velocities
    velocity.x = cosine * vFinal[0].x - sine * vFinal[0].y;
    velocity.y = cosine * vFinal[0].y + sine * vFinal[0].x;
    other.velocity.x = cosine * vFinal[1].x - sine * vFinal[1].y;
    other.velocity.y = cosine * vFinal[1].y + sine * vFinal[1].x;
   }
  }

  void display() {
    noStroke();
    fill(204);
    ellipse(position.x, position.y, r*2, r*2);
  }
}
```

Bouncing Bubbles

CODE



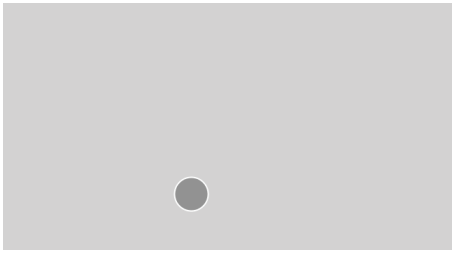
```
/**
 * Bouncy Bubbles
 * based on code from Keith Peters.
 *
 * Multiple-object collision.
 */
```

```
int numBalls = 10;
float spring = 0.05;
float gravity = 0.03;
float friction = -0.9;
Ball[] balls = new Ball[numBalls];

void setup() {
  size(640, 360);
  for (int i = 0; i < numBalls; i++) {
    balls[i] = new Ball(random(width), random(height), random(30,
70), i, balls);
  }
  noStroke();
  fill(255, 204);
}

void draw() {
  background(0);
  for (int i = 0; i < numBalls; i++) {
    balls[i].collide();
    balls[i].move();
    balls[i].display();
  }
}

class Ball {

  float x, y;
  float diameter;
  float vx = 0;
  float vy = 0;
  int id;
  Ball[] others;

  Ball(float xin, float yin, float din, int idin, Ball[] oin) {
    x = xin;
    y = yin;
    diameter = din;
    id = idin;
    others = oin;
  }

  void collide() {
    for (int i = id + 1; i < numBalls; i++) {
      float dx = others[i].x - x;
      float dy = others[i].y - y;
      float distance = sqrt(dx*dx + dy*dy);
      float minDist = others[i].diameter/2 + diameter/2;
      if (distance < minDist) {
        float angle = atan2(dy, dx);
        float targetX = x + cos(angle) * minDist;
        float targetY = y + sin(angle) * minDist;
        float ax = (targetX - others[i].x) * spring;
        float ay = (targetY - others[i].y) * spring;
        vx -= ax;
        vy -= ay;
```

```
      others[i].vx += ax;
      others[i].vy += ay;
    }
  }
 }

 void move() {
  vy += gravity;
  x += vx;
  y += vy;
  if (x + diameter/2 > width) {
   x = width - diameter/2;
   vx *= friction;
  }
  else if (x - diameter/2 < 0) {
   x = diameter/2;
   vx *= friction;
  }
  if (y + diameter/2 > height) {
   y = height - diameter/2;
   vy *= friction;
  }
  else if (y - diameter/2 < 0) {
   y = diameter/2;
   vy *= friction;
  }
 }

 void display() {
  ellipse(x, y, diameter, diameter);
 }
}
```

Follow Mouse

CODE



```
// A Mover object
Mover mover;

void setup() {
  size(640,360);
  mover = new Mover();
}

void draw() {
  background(200);

  // Update the location
  mover.update();
  // Display the Mover
  mover.display();
}

class Mover {

  // The Mover tracks location, velocity, and acceleration
  PVector location;
  PVector velocity;
  PVector acceleration;
  // The Mover's maximum speed
  float topspeed;

  Mover() {
    // Start in the center
    location = new PVector(width/2,height/2);
    velocity = new PVector(0,0);
    topspeed = 5;
  }

  void update() {

    // Compute a vector that points from location to mouse
    PVector mouse = new PVector(mouseX,mouseY);
    PVector acceleration = PVector.sub(mouse,location);
    // Set magnitude of acceleration
    acceleration.setMag(0.2);

    // Velocity changes according to acceleration
    velocity.add(acceleration);
    // Limit the velocity by topspeed
    velocity.limit(topspeed);
    // Location changes by velocity
    location.add(velocity);
  }

  void display() {
    stroke(255);
    strokeWeight(2);
    fill(127);
    ellipse(location.x,location.y,48,48);
  }
}
```

```
/**
 * Acceleration with Vectors
 * by Daniel Shiffman.
 *
 * Demonstration of the basics of motion with vector.
 * A "Mover" object stores location, velocity, and acceleration as vectors
 * The motion is controlled by affecting the acceleration (in this case towards the mouse)
 *
 * For more examples of simulating motion and physics with vectors, see
 * Simulate/ForcesWithVectors, Simulate/GravitationalAttraction3D
 */
```


```
/**
 * Acceleration with Vectors
 * by Daniel Shiffman.
 *
 * Demonstration of the basics of motion with vector.
 * A "Mover" object stores location, velocity, and acceleration as vectors
 * The motion is controlled by affecting the acceleration (in this case towards the mouse)
 */
```

Constrain

CODE


```
float mx;
float my;
float easing = 0.05;
int radius = 24;
int edge = 100;
int inner = edge + radius;

void setup() {
  size(640, 360);
  noStroke();
  ellipseMode(RADIUS);
  rectMode(CORNERS);
}

void draw() {
  background(200);

  if (abs(mouseX - mx) > 0.1) {
    mx = mx + (mouseX - mx) * easing;
  }
  if (abs(mouseY - my) > 0.1) {
    my = my + (mouseY- my) * easing;
  }

  mx = constrain(mx, inner, width - inner);
  my = constrain(my, inner, height - inner);
  fill(255,255,0);
  rect(edge, edge, width-edge, height-edge);
  fill(255);
  ellipse(mx, my, radius, radius);
}
```

```
/**
 * Constrain.
 *
 * Move the mouse across the screen to
move the circle.
 * The program constrains the circle to
its box.
 */
```

Scrollbar

CODE

```
// The next line is needed if running in JavaScript Mode with
Processing.js
/* @pjs preload="seedTop.jpg,seedBottom.jpg"; */


HScrollbar hs1, hs2;  // Two scrollbars
PImage img1, img2;  // Two images to load

void setup() {
  size(640, 360);
  noStroke();

  hs1 = new HScrollbar(0, height/2-8, width, 16, 16);
  hs2 = new HScrollbar(0, height/2+8, width, 16, 16);

  // Load images
  img1 = loadImage("seedTop.jpg");
  img2 = loadImage("seedBottom.jpg");
}

void draw() {
  background(255);

  // Get the position of the img1 scrollbar
  // and convert to a value to display the img1 image
  float img1Pos = hs1.getPos()-width/2;
  fill(255);
  image(img1, width/2-img1.width/2 + img1Pos*1.5, 0);

  // Get the position of the img2 scrollbar
  // and convert to a value to display the img2 image
  float img2Pos = hs2.getPos()-width/2;
  fill(255);
  image(img2, width/2-img2.width/2 + img2Pos*1.5, height/2);

  hs1.update();
  hs2.update();
  hs1.display();
  hs2.display();

  stroke(0);
  line(0, height/2, width, height/2);
}


class HScrollbar {
  int swidth, sheight;    // width and height of bar
  float xpos, ypos;       // x and y position of bar
  float spos, newspos;    // x position of slider
  float sposMin, sposMax; // max and min values of slider
  int loose;              // how loose/heavy
  boolean over;           // is the mouse over the slider?
  boolean locked;
```

```
/**
 * Scrollbar.
 *
 * Move the scrollbars left and right to
change the positions of the images.
 */
```

```
  float ratio;

  HScrollbar (float xp, float yp, int sw, int sh, int l) {
    swidth = sw;
    sheight = sh;
    int widthtoheight = sw - sh;
    ratio = (float)sw / (float)widthtoheight;
    xpos = xp;
    ypos = yp-sheight/2;
    spos = xpos + swidth/2 - sheight/2;
    newspos = spos;
    sposMin = xpos;
    sposMax = xpos + swidth - sheight;
    loose = l;
  }

  void update() {
    if (overEvent()) {
      over = true;
    } else {
      over = false;
    }
    if (mousePressed && over) {
      locked = true;
    }
    if (!mousePressed) {
      locked = false;
    }
    if (locked) {
      newspos = constrain(mouseX-sheight/2, sposMin, sposMax);
    }
    if (abs(newspos - spos) > 1) {
      spos = spos + (newspos-spos)/loose;
    }
  }

  float constrain(float val, float minv, float maxv) {
    return min(max(val, minv), maxv);
  }

  boolean overEvent() {
    if (mouseX > xpos && mouseX < xpos+swidth &&
       mouseY > ypos && mouseY < ypos+sheight) {
      return true;
    } else {
      return false;
    }
  }

  void display() {
    noStroke();
    fill(204);
    rect(xpos, ypos, swidth, sheight);
    if (over || locked) {
      fill(0, 0, 0);
    } else {
      fill(102, 102, 102);
    }
    rect(spos, ypos, sheight, sheight);
  }

  float getPos() {
    // Convert spos to be values between
    // 0 and the total width of the scrollbar
    return spos * ratio;
  }
}
```

Moving on Curves

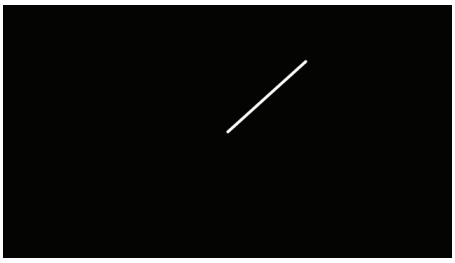
CODE

```
float beginX = 20.0;  // Initial x-coordinate
float beginY = 10.0;  // Initial y-coordinate
float endX = 570.0;   // Final x-coordinate
float endY = 320.0;   // Final y-coordinate
float distX;          // X-axis distance to move
float distY;          // Y-axis distance to move
float exponent = 4;   // Determines the curve
float x = 0.0;        // Current x-coordinate
float y = 0.0;        // Current y-coordinate
float step = 0.01;    // Size of each step along the path
float pct = 0.0;      // Percentage traveled (0.0 to 1.0)

void setup() {
  size(640, 360);
  noStroke();
  distX = endX - beginX;
  distY = endY - beginY;
}

void draw() {


  fill(0, 2);
  rect(0, 0, width, height);
  pct += step;
  if (pct < 1.0) {
    x = beginX + (pct * distX);
    y = beginY + (pow(pct, exponent) * distY);
  }


    endX = mouseX;
  endY = mouseY;
  distX = endX - beginX;
  distY = endY - beginY;

  fill(255);
  ellipse(x, y, 20, 20);
}

void mouseMoved() {
  pct = 0.0;
  beginX = x;
  beginY = y;
  endX = mouseX;
  endY = mouseY;
  distX = endX - beginX;
  distY = endY - beginY;
}
```

```
/**
 * Moving On Curves.
 *
 * In this example, the circles moves
along the curve y = x^4.
 * Click the mouse to have it move to a
new position.
 */
```

RotateLine

CODE

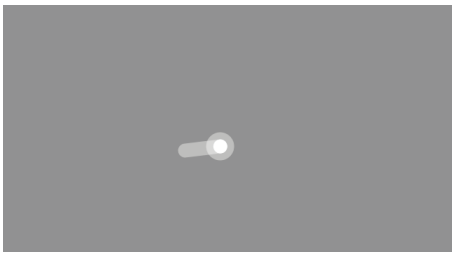


```
void setup() {
  size(640,360);
  smooth();
}

void draw() {
  background(0);

  // A vector that points to the mouse location
  PVector mouse = new PVector(mouseX,mouseY);
  // A vector that points to the center of the window
  PVector center = new PVector(width/2,height/2);
  // Subtract center from mouse which results in a vector that points
from center to mouse
  mouse.sub(center);

  // Normalize the vector
  mouse.normalize();

  // Multiply its length by 150 (Scaling its length)
  mouse.mult(150);

  translate(width/2,height/2);
  // Draw the resulting vector
  stroke(255);
  strokeWeight(4);
  line(0,0,mouse.x,mouse.y);

}
```

```
/**
 * Vector
 * by Daniel Shiffman.
 *
 * Demonstration some basic vector
math: subtraction, normalization,
scaling
 * Normalizing a vector sets its length
to 1.
 */
```

Follow1

CODE



```
float x = 100;
float y = 100;
float angle1 = 0.0;
float segLength = 50;

void setup() {
  size(640, 360);
  strokeWeight(20.0);
  stroke(255, 100);
}

void draw() {
  background(126);

  float dx = mouseX - x;
  float dy = mouseY - y;
  angle1 = atan2(dy, dx);
  x = mouseX - (cos(angle1) * segLength);
  y = mouseY - (sin(angle1) * segLength);

  segment(x, y, angle1);
  ellipse(x, y, 20, 20);
}

void segment(float x, float y, float a) {
  pushMatrix();
  translate(x, y);
  rotate(a);
  line(0, 0, segLength, 0);
  popMatrix();
}
```

```
/**
 * Follow 1
 * based on code from Keith Peters.
 *
 * A line segment is pushed and pulled
by the cursor.
 */
```

Follow2

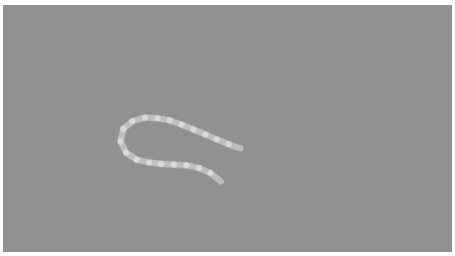
CODE

```
float[] x = new float[2];
float[] y = new float[2];
float segLength = 50;

void setup() {
  size(640, 360);
  strokeWeight(20.0);
  stroke(255, 100);
}

void draw() {
  background(0);
  dragSegment(0, mouseX, mouseY);
  dragSegment(1, x[0], y[0]);
}

void dragSegment(int i, float xin, float yin) {
  float dx = xin - x[i];
  float dy = yin - y[i];
  float angle = atan2(dy, dx);
  x[i] = xin - cos(angle) * segLength;
  y[i] = yin - sin(angle) * segLength;
  segment(x[i], y[i], angle);
}

void segment(float x, float y, float a) {
  pushMatrix();
  translate(x, y);
  rotate(a);
  line(0, 0, segLength, 0);
  popMatrix();
}
```

```
/**
 * Follow 2
 * based on code from Keith Peters.
 *
 * A two-segmented arm follows the cursor position. The relative
 * angle between the segments is calculated with atan2() and the
 * position calculated with sin() and cos().
 */
```

Follow3

CODE



```
float[] x = new float[20];
float[] y = new float[20];
float segLength = 18;

void setup() {
  size(640, 360);
  strokeWeight(9);
  stroke(255, 100);
}

void draw() {
  background(126);
  dragSegment(0, mouseX, mouseY);
  for(int i=0; i<x.length-1; i++) {
    dragSegment(i+1, x[i], y[i]);
  }
}

void dragSegment(int i, float xin, float yin) {
  float dx = xin - x[i];
  float dy = yin - y[i];
  float angle = atan2(dy, dx);
  x[i] = xin - cos(angle) * segLength;
  y[i] = yin - sin(angle) * segLength;
  segment(x[i], y[i], angle);
}

void segment(float x, float y, float a) {
  pushMatrix();
  translate(x, y);
  rotate(a);
  line(0, 0, segLength, 0);
  popMatrix();
}
```

```
/**
 * Follow 3
 * based on code from Keith Peters.
 *
 * A segmented line follows the mouse.
The relative angle from
 * each segment to the next is
calculated with atan2() and the
 * position of the next is calculated with
sin() and cos().
 */
```

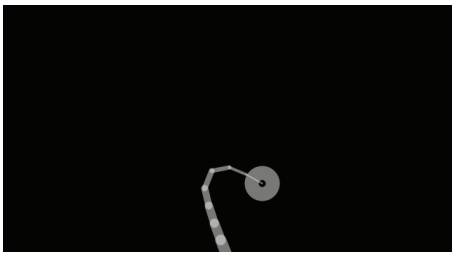
Reach2

CODE

```
int numSegments = 10;
float[] x = new float[numSegments];
float[] y = new float[numSegments];
float[] angle = new float[numSegments];
float segLength = 26;
float targetX, targetY;

void setup() {
  size(640, 360);
  strokeWeight(20.0);
  stroke(255, 100);
  x[x.length-1] = width/2;     // Set base x-coordinate
  y[x.length-1] = height;  // Set base y-coordinate
}

void draw() {
  background(126);

  reachSegment(0, mouseX, mouseY);
  for(int i=1; i<numSegments; i++) {
    reachSegment(i, targetX, targetY);
  }
  for(int i=x.length-1; i>=1; i--) {
    positionSegment(i, i-1);
  }
  for(int i=0; i<x.length; i++) {
    segment(x[i], y[i], angle[i], (i+1)*2);
  }
}

void positionSegment(int a, int b) {
  x[b] = x[a] + cos(angle[a]) * segLength;
  y[b] = y[a] + sin(angle[a]) * segLength;
}

void reachSegment(int i, float xin, float yin) {
  float dx = xin - x[i];
  float dy = yin - y[i];
  angle[i] = atan2(dy, dx);
  targetX = xin - cos(angle[i]) * segLength;
  targetY = yin - sin(angle[i]) * segLength;
}

void segment(float x, float y, float a, float sw) {
  strokeWeight(sw);
  pushMatrix();
  translate(x, y);
  rotate(a);
  line(0, 0, segLength, 0);
  popMatrix();
}
```

```
/**
 * Reach 2
 * based on code from Keith Peters.
 *
 * The arm follows the position of the
mouse by
 * calculating the angles with atan2().
 */
```

CODE



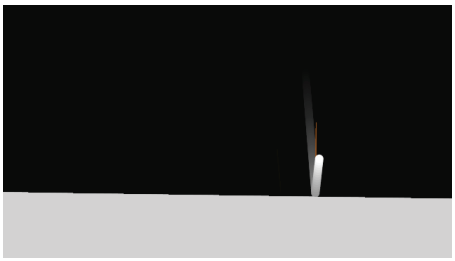
Reach3

```
int numSegments = 8;
float[] x = new float[numSegments];
float[] y = new float[numSegments];
float[] angle = new float[numSegments];
float segLength = 26;
float targetX, targetY;
float ballX = 50;
float ballY = 50;
int ballXDirection = 1;
int ballYDirection = -1;

void setup() {
  size(640, 360);
  strokeWeight(20.0);
  stroke(255, 100);
  noFill();
  x[x.length-1] = width/2;     // Set base x-coordinate
  y[x.length-1] = height;  // Set base y-coordinate
}

void draw() {
  background(0);
  strokeWeight(20);
  ballX = ballX + 1.0 * ballXDirection;
  ballY = ballY + 0.8 * ballYDirection;
  if(ballX > width-25 || ballX < 25) {
    ballXDirection *= -1;
  }
  if(ballY > height-25 || ballY < 25) {
    ballYDirection *= -1;
  }
  ellipse(ballX, ballY, 30, 30);
  reachSegment(0, ballX, ballY);
  for(int i=1; i<numSegments; i++) {
    reachSegment(i, targetX, targetY);
  }
  for(int i=x.length-1; i>=1; i--) {
    positionSegment(i, i-1);
  }
  for(int i=0; i<x.length; i++) {
    segment(x[i], y[i], angle[i], (i+1)*2);
  }
}

void positionSegment(int a, int b) {
  x[b] = x[a] + cos(angle[a]) * segLength;
  y[b] = y[a] + sin(angle[a]) * segLength;
}

void reachSegment(int i, float xin, float yin) {
  float dx = xin - x[i];
  float dy = yin - y[i];
  angle[i] = atan2(dy, dx);
  targetX = xin - cos(angle[i]) * segLength;
  targetY = yin - sin(angle[i]) * segLength;
}

void segment(float x, float y, float a, float sw) {
  strokeWeight(sw);
  pushMatrix();
  translate(x, y);
  rotate(a);
  line(0, 0, segLength, 0);
  popMatrix();
}
```

```
/**
 * Reach 3
 * based on code from Keith Peters.
 *
 * The arm follows the position of the ball by
 * calculating the angles with atan2().
 */
```

Reflection1

CODE



```
// Position of left hand side of floor
PVector base1;
// Position of right hand side of floor
PVector base2;
// Length of floor
float baseLength;

// An array of subpoints along the floor path
PVector[] coords;

// Variables related to moving ball
PVector position;
PVector velocity;
float r = 6;
float speed = 3.5;

void setup() {
  size(640, 360);

  fill(128);
  base1 = new PVector(0, height-150);
  base2 = new PVector(width, height);
  createGround();

  // start ellipse at middle top of screen
  position = new PVector(width/2, 0);

  // calculate initial random velocity
  velocity = PVector.random2D();
  velocity.mult(speed);
}

void draw() {
  // draw background
  fill(0, 12);
  noStroke();
  rect(0, 0, width, height);

  // draw base
  fill(200);
  quad(base1.x, base1.y, base2.x, base2.y, base2.x, height, 0, height);

  // calculate base top normal
  PVector baseDelta = PVector.sub(base2, base1);
  baseDelta.normalize();
  PVector normal = new PVector(-baseDelta.y, baseDelta.x);

  // draw ellipse
  noStroke();
  fill(255);
  ellipse(position.x, position.y, r*2, r*2);

  // move elipse
  position.add(velocity);

  // normalized incidence vector
  PVector incidence = PVector.mult(velocity, -1);
```

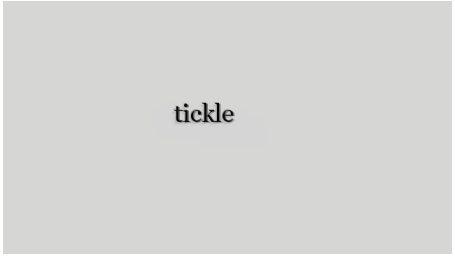
```
/**
 * Non-orthogonal Reflection
 * by Ira Greenberg.
 *
 * Based on the equation (R =
2N(N*L)-L) where R is the
 * reflection vector, N is the normal, and
L is the incident
 * vector.
 */
```

CODE

```
  incidence.normalize();

  // detect and handle collision
  for (int i=0; i<coords.length; i++) {
    // check distance between ellipse and base top coordinates
    if (PVector.dist(position, coords[i]) < r) {

      // calculate dot product of incident vector and base top normal
      float dot = incidence.dot(normal);

      // calculate reflection vector
      // assign reflection vector to direction vector
      velocity.set(2*normal.x*dot - incidence.x, 2*normal.y*dot -
incidence.y, 0);
      velocity.mult(speed);

      // draw base top normal at collision point
      stroke(255, 128, 0);
      line(position.x, position.y, position.x-normal.x*100, position.y-
normal.y*100);
    }
  }

  // detect boundary collision
  // right
  if (position.x > width-r) {
    position.x = width-r;
    velocity.x *= -1;
  }
  // left
  if (position.x < r) {
    position.x = r;
    velocity.x *= -1;
  }
  // top
  if (position.y < r) {
    position.y = r;
    velocity.y *= -1;
    // randomize base top
    base1.y = random(height-100, height);
    base2.y = random(height-100, height);
    createGround();
  }
}


// Calculate variables for the ground
void createGround() {
  // calculate length of base top
  baseLength = PVector.dist(base1, base2);

  // fill base top coordinate array
  coords = new PVector[ceil(baseLength)];
  for (int i=0; i<coords.length; i++) {
    coords[i] = new PVector();
    coords[i].x = base1.x + ((base2.x-base1.x)/baseLength)*i;
    coords[i].y = base1.y + ((base2.y-base1.y)/baseLength)*i;
  }
}
```

Tickle

CODE

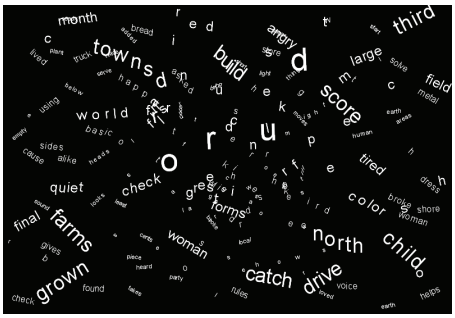



```
String message = "tickle";
float x, y; // X and Y coordinates of text
float hr, vr;  // horizontal and vertical radius of the text

void setup() {
  size(640, 360);

  // Create the font
  textFont(createFont("Georgia", 36));
  textAlign(CENTER, CENTER);

  hr = textWidth(message) / 2;
  vr = (textAscent() + textDescent()) / 2;
  noStroke();
  x = width / 2;
  y = height / 2;
}

void draw() {
  // Instead of clearing the background, fade it by drawing
  // a semi-transparent rectangle on top
  fill(204, 120);
  rect(0, 0, width, height);

  // If the cursor is over the text, change the position
  if (abs(mouseX - x) < hr &&
      abs(mouseY - y) < vr) {
    x += random(-5, 5);
    y += random(-5, 5);
  }
  fill(0);
  text("tickle", x, y);
}
```

```
/**
 * Tickle.
 *
 * The word «tickle» jitters when the cursor hovers over.
 * Sometimes, it can be tickled off the screen.
 */

// The next line is needed if running in JavaScript Mode with Processing.js
/* @pjs font=»Georgia.ttf»; */
```

Font Physics

CODE




```
import java.awt.geom.Line2D;

static final float wordRatio = 2.4; // approx ratio of string lenth to font size
static final int fontdepth = 5; // recursion depth for word creation

static final float G = -1.0; //change gravity
static final float cursor_mass = 10000000.; //mass of cursor
float damping = 2.5;

float max_length;

Letter[] letters; //array of the letters
Line2D[] lines; // text lines for intersections

static final char[] charset = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z' };

class Letter
{
  PVector p;
  PVector o;
  PVector dp;
  char letter;
  float fontsize;
  float theta;
  float mass;
  float k;
  Letter(PVector pos,PVector dpos,char letterval,float fontsizeval,float thetaval)
  {
    p = new PVector(pos.x,pos.y);
    o = new PVector(pos.x,pos.y);
    dp = new PVector(dpos.x,dpos.y);
    letter = letterval;
    theta = thetaval;
    fontsize = fontsizeval;
    //mass = fontsize/20.;
    mass = 1.;
    k = 10.;
  }
}

void setup()
{
  //size(600,600,P2D);
  //size(screen.width,screen.height,JAVA2D);
  size(800,550,JAVA2D);
  hint(ENABLE_NATIVE_FONTS);
  reset();
}

void keyPressed()
{
  if( key == ' ' ) {
    reset();
  } else if( key == '[' ) {
    damping *= 0.8;
  } else if( key == ']' ) {
    damping *= 1.2;
  }
}
```

CODE

```
void reset()
{
  max_length = min(width,height)/4.0;
  letters = new Letter[0];
  lines = new Line2D[0];
  textFont(createFont("Arial", max_length/wordRatio, true, charset));
  //textMode(SCREEN);
  textSize(24.0);
  text( "loading", (displayWidth - textWidth("loading"))/2.0,
(displayHeight - 24.0)/2.0);
  for(int i = 0; i < fontdepth; ++i) {
    float fontsize = (max_length*pow(0.66,i)/wordRatio);
    for(int j = 0; j < 10*i + 1; ++j) {
      boolean success = false;
      for(int t = 0;t < 100; ++t) {
        boolean thisTry = makeWord(fontsize);
        if( thisTry ) {
          success = true;
          break;
        }
      }
      if( !success ) {
        println("failed to make letter");
      }
    }
  }
  background(0);
}

boolean makeWord(float fontsize)
{
  textSize(fontsize);
  PVector pos = new PVector(random(width),random(height));
  String word = words[(int)random(words.length)];
  float theta = random(PI/2.0) - PI/4.0;
  float wlength = textWidth(word);
  PVector end = new PVector(pos.x + wlength*cos(theta), pos.y +
wlength*sin(theta));
  if( end.x < 0 || width < end.x || end.y < 0 || height < end.y ) {
    return false;
  }
  Letter[] newword = layoutWord(word, pos, fontsize, theta);
  Line2D[] newlines = new Line2D[4];
  float xoff = fontsize*cos(theta-HALF_PI);
  float yoff = fontsize*sin(theta-HALF_PI);
  newlines[0] = (Line2D) new Line2D.Float( pos.x, pos.y, end.x, end.y
);
  newlines[1] = (Line2D) new Line2D.Float( end.x, end.y, end.x +xoff,
end.y +yoff );
  newlines[2] = (Line2D) new Line2D.Float( end.x +xoff, end.y +yoff,
pos.x +xoff, pos.y +yoff);
  newlines[3] = (Line2D) new Line2D.Float( pos.x +xoff, pos.y +yoff,
pos.x, pos.y );

  for(int i = 0; i < lines.length; ++i) {
    for(int j = 0; j < 4; ++j) {
      if( newlines[j].intersectsLine(lines[i]) ) {
        return false;
      }
    }
  }
  letters = (Letter[])concat(letters, newword );
  lines = (Line2D[])concat(lines,newlines);
  return true;
}

float time = 0.0;
```

CODE

```
void draw()
{
  float dtime = millis()/1000.0 - time;
  time += dtime;
  calcAccel(dtime);
  calcMove(dtime);
  resetMatrix();
  background(0);
  fill(255);
  stroke(255);
  for(int i = 0; i < letters.length; ++i) {
    pushMatrix();
    translate(letters[i].p.x,letters[i].p.y);
    rotate(letters[i].theta);
    textSize(letters[i].fontsize);
    text(letters[i].letter,0,0);
    popMatrix();
  }
}

void calcAccel(float time)
{
  if( mousePressed ) {
    PVector mouse_p = new PVector((float)mouseX,(float)mouseY);
    float Gprime = G;
    if( mouseButton == RIGHT ) {
      Gprime *= -1;
    }
    for(int i = 0; i < letters.length; ++i) {
      PVector r = new PVector(letters[i].p.x, letters[i].p.y);
      r.sub(mouse_p);
      PVector d = new PVector(letters[i].p.x, letters[i].p.y);
      d.sub(letters[i].o);
      PVector accel = new PVector(r.x,r.y);
      accel.mult( Gprime*cursor_mass/max( pow(r.mag(),3.0), 250.0 )
);
      PVector tmp_1 = new PVector(d.x,d.y);
      tmp_1.mult( -letters[i].k/letters[i].mass );
      accel.add( tmp_1 );
      PVector tmp_2 = new PVector(letters[i].dp.x,letters[i].dp.y);
      tmp_2.mult( -damping/letters[i].mass );
      accel.add( tmp_2 );
      accel.mult( time );
      //println( str(accel.x) + str(accel.y) );
      letters[i].dp.add(accel);
    }
  } else {
    for(int i = 0; i < letters.length; ++i) {
      PVector d = new PVector(letters[i].p.x, letters[i].p.y);
      d.sub(letters[i].o);
      PVector accel = new PVector(d.x,d.y);
      accel.mult( -letters[i].k/letters[i].mass );
      PVector tmp_2 = new PVector(letters[i].dp.x,letters[i].dp.y);
      tmp_2.mult( -damping/letters[i].mass );
      accel.add( tmp_2 );
      accel.mult( time );
      letters[i].dp.add(accel);
    }
  }
}

void calcMove(float time)
{
  for(int i = 0; i < letters.length; ++i) {
    PVector tmp_3 = new PVector(letters[i].dp.x,letters[i].dp.y);
    tmp_3.mult(time);
    letters[i].p.add(tmp_3);
  }
}
```

CODE

```
Letter[] layoutWord(String word, PVector pos, float fontsize, float
theta)
{
  Letter[] newletters = new Letter[5];
  PVector cur_pos = new PVector(pos.x,pos.y);
  PVector zero = new PVector(0.0,0.0);
  for(int i = 0; i < 5; ++i) {
    Letter l = new Letter( cur_pos, zero, word.charAt(i), fontsize, theta
);
    newletters[i] = l;
    textSize(fontsize);
    float w = textWidth(word.charAt(i));
    cur_pos.x += w*cos(theta);
    cur_pos.y += w*sin(theta);
  }
  return newletters;
}

String[] words = {
  "which", "there", "their", "about", "would", "these", "other",
"words", "could", "write", "first", "water", "after", "where", "right",
"think", "three", "years", "place", "sound", "great", "again", "still",
"every", "small", "found", "those", "never", "under", "might",
"while", "house", "world", "below", "asked", "going", "large",
"until", "along", "shall", "being", "often", "earth", "began", "since",
"study", "night", "light", "above", "paper", "parts", "young",
"story", "point", "times", "heard", "whole", "white", "given",
"means", "music", "miles", "thing", "today", "later", "using",
"money", "lines", "order", "group", "among", "learn", "known",
"space", "table", "early", "trees", "short", "hands", "state", "black",
"shown", "stood", "front", "voice", "kinds", "makes", "comes",
"close", "power", "lived", "vowel", "taken", "built", "heart", "ready",
"quite", "class", "bring", "round", "horse", "shows", "piece",
"green", "stand", "birds", "start", "river", "tried", "least", "field",
"whose", "girls", "leave", "added", "color", "third", "hours",
"moved", "plant", "doing", "names", "forms", "heavy", "ideas",
"cried", "check", "floor", "begin", "woman", "alone", "plane",
"spell", "watch", "carry", "wrote", "clear", "named", "books",
"child", "glass", "human", "takes", "party", "build", "seems",
"blood", "sides", "seven", "mouth", "solve", "north", "value",
"death", "maybe", "happy", "tells", "gives", "looks", "shape",
"lives", "steps", "areas", "sense", "speak", "force", "ocean",
"speed", "women", "metal", "south", "grass", "scale", "cells",
"lower", "sleep", "wrong", "pages", "ships", "needs", "rocks",
"eight", "major", "level", "total", "ahead", "reach", "stars", "store",
"sight", "terms", "catch", "works", "board", "cover", "songs",
"equal", "stone", "waves", "guess", "dance", "spoke", "break",
"cause", "radio", "weeks", "lands", "basic", "liked", "trade",
  "fresh", "final", "fight", "meant", "drive", "spent", "local", "waxes",
"knows", "train", "bread", "homes", "teeth", "coast", "thick",
"brown", "clean", "quiet", "sugar", "facts", "steel", "forth", "rules",
"notes", "units", "peace", "month", "verbs", "seeds", "helps",
"sharp", "visit", "woods", "chief", "walls", "cross", "wings", "grown",
"cases", "foods", "crops", "fruit", "stick", "wants", "stage", "sheep",
"nouns", "plain", "drink", "bones", "apart", "turns", "moves",
"touch", "angle", "based", "range", "marks", "tired", "older",
"farms", "spend", "shoes", "goods", "chair", "twice", "cents",
"empty", "alike", "style", "broke", "pairs", "count", "enjoy", "score",
"shore", "roots", "paint", "heads", "shook", "serve", "angry",
"crowd", "wheel", "quick", "dress", "share", "alive", "noise", "solid",
"cloth", "signs", "hills", "types", "drawn", "worth", "truck", "piano",
"upper", "loved", "usual", "faces", "drove", "cabin", "boats",
"towns", "proud"
};
```