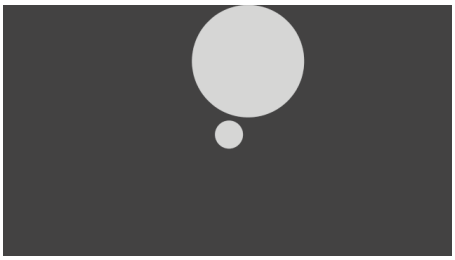
Circle Collision

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);
  }
}
```