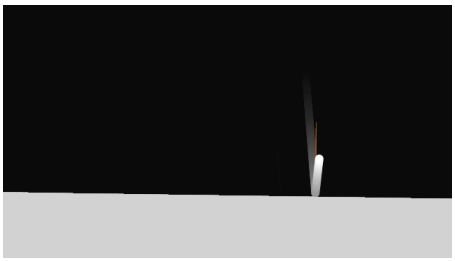
Reflection1

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;
  }
}
```