//Emily Opel, Independent Study Project - Processing

1//       //An array of 8 planet objects

2//       Planet[] planets = new Planet[9];

3//       PImage img;

4//

5//       void setup() {

6//         size(950,950);

7//         img = loadImage("space.jpg");

8//         smooth();

9//

10//      // The planet objects are initialized using the counter

variable

11//      //Mercury, 0.2409

12//      planets[0] = new Planet(92, 20, 248, 150, 0, 0);

13//      //Venus, 0.616

14//      planets[1] = new Planet(126, 40, 165, 200, 150, 0);

15//      //Earth, 1.0

16//      planets[2] = new Planet(166, 46, 84, 0, 150, 190);

17//      //Mars, 1.9

18//      planets[3] = new Planet(211, 30, 29.5, 200, 0, 0);

19//      //Jupiter, 12.0

20//      planets[4] = new Planet(287, 120, 12.0, 230, 100, 0);

21//      //Saturn, 29.5

22//      planets[5] = new Planet(375, 90, 1.9, 200, 200, 170);

23//      //Uranus, 84

24//      planets[6] = new Planet(468, 76, 1.0, 150, 150, 255);

25//      //Neptune, 165

26//      planets[7] = new Planet(559, 70, 0.616, 50, 50, 255);

27//      //Pluto, 248

28//      planets[8] = new Planet(595, 10, 0.241, 150, 150, 150);

29//    }

30//

31//    void draw() {

32//      image(img, 0, 0, 1750, 1000);

33//

34//      // Drawing the Sun

35//      pushMatrix();

36//      stroke(0);

37//      fill(250, 250, 0);

38//      ellipse(0,0,200,200);

39//

40//      // Drawing all Planets

41//      for (int i = 0; i < planets.length; i++ ) {

42//        planets[i].update();

43//        planets[i].display();

44//      }

45//      popMatrix();

46//    }

47//

48//    class Planet {

49//      // Each planet object keeps track of its own angle of

rotation.

50//      float theta;   // Rotation around sun

51//      float diameter;   // Size of planet

52//      float distance;   // Distance from sun

53//      float orbitspeed; // Orbit speed

54//      int colorR;    // R color value

55//      int colorG;    // G color value

56//      int colorB;    // B color value

57//

58//      Planet(float distance\_, float diameter\_, float

   orbitspeed\_, int R, int G, int B) {

59//        distance = distance\_;

60//        diameter = diameter\_;

61//        theta = 0;

67//        orbitspeed = orbitspeed\_;

68//        colorR = R;

69//        colorG = G;

70//        colorB = B;

71//      }

72//

73//      void update() {

74//        // Increment the angle to rotate

75//        theta += orbitspeed\*0.00015;

76//      }

77//

78//      void display() {

79//        // Before rotation and translation, the state of the

  matrix is saved with pushMatrix().

80//        pushMatrix();

81//        // Rotate orbit

82//        rotate(theta);

83//        // translate out distance

84//        translate(distance\*1.5,0);

85//        stroke(0);

86//        fill(colorR, colorG, colorB);

87//        ellipse(0,0,diameter,diameter);

88//        // Once the planet is drawn, the matrix is restored

   with popMatrix() so that the next planet is not

  affected.

89//        popMatrix();

90//      }

91//    }

Tutorial:

           For the tutorial of my program, I’m going to explain how most of the methods used in my program work and what line(s) they are on so you can follow along with the written program above.

           On line 3, this line of code declares an image variable called, ‘img’ which is then instantiated in line 7 to be the JPEG image ‘space’ that I put into the data folder of the program which allowed me to use the image. This image is what you see as the background to the program, and it is placed and sized on line 32. The first two numbers in the method called on line 32 place the top left corner of the image at the coordinates (0, 0), while the other two numbers set the size of the image with first the length on the x-axis and then the length on the y-axis.

           Line 5 is the basic setup of the program where the size of the program window at run time is set. Also, this is where each of the planet objects are declared with their own distances from the sun, diameters, orbit speeds, and three RGB values to make the color of the planets.

           The draw method that starts on line 31 basically does as it says. In this method the sun is drawn in the upper left-hand corner of the program window and the for loop used in lines 41 through 44 draws the planets using the update and display methods from the Planet class. You may also see on lines 35 and 45 the methods, pushMatrix and popMatrix, these will be explained later when they are used in the Planet class method, display.

Line 48 starts the Planet class and on lines 49 to 56, each property of a planet is declared, and then instantiated in the constructor method starting at line 58.

The update method on line 73 is what helps to make the planets move by increasing their theta property by 0.015% of their orbital speed. The reason that theta is increased by 0.015% of the orbital speed rather than the full orbital speed is because when it I tried that, the planets were moving so fast that you could only see a glimpse of them every so often, so I had to slow it down.

Finally, the display method on line 78 is what really shows the planets and moves them around the sun in the program. It does this by first saving the previous matrix with the pushMatrix method. The matrix is basically the individual grid that each of the planets lie on. Then at the end of the method it uses the popMatrix method to restore the previous matrix that was saved by the pushMatrix method. If you need more help understanding this, I suggest looking at the Transform section of processing.org’s Reference page, and it will be able to explain these methods more thoroughly. Then, once the matrix has been saved and is able to be changed, the rotate method is used to rotate the grid by the planet object’s theta value which allows it to turn around the sun. Then the translate method is used to keep the planet object at the same distance its supposed to be from the sun by using its distance value multiplied by 1.5 as the x-value. This translate method uses the amounts given as x and y values to move the whole grid, which is just like moving the coordinates of an object but easier and better to use for animation purposes. Lines 85 to 87 are simply used to draw the planet object where it should be once the grid is moved. The stroke method used in line 85 makes the outline of the planets set to 0(black) in RGB values, and the fill method on line 86 uses the three RGB values given when the planet objects were declared as the filling color of the planet being drawn. The ellipse method on line 87 draws the planets at the top left corners of their individual grids and gives them their sizes using their diameter properties. Once all of this is done, the pushMatrix method is used and the previously saved matrix is restored so that each planet’s individual grid does not affect one another.