**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII AL REPUBLICII MOLDOVA Universitatea Tehnică a Moldovei Facultatea Calculatoare, Informatică şi Microelectronică Departamentul Inginerie Software și Automatică**

Copta Adrian | FAF-223

# Report

*Laboratory work n.4 part 1*

***of Computer Graphics***

Checked by:

**Olga Grosu,** *university assistant*

DISA, FCIM, UTM

**Chișinău – 2023**

1. **Purpose of the task work:**

Analyze the Example from Introduction and the Example from Chapter 1. Vectors and make the Exercise 1.1-1.8.

2. **Condition:**

Exercise 1.1: Find something you've previously made in Processing using separate x and y variables and use PVectors instead.

Exercise 1.2: Take one of the walker examples from the introduction and convert it to use PVectors.

Exercise 1.3: Extend the bouncing ball with vectors example into 3D. Can you get a sphere to bounce

Exercise 1.4: Write the limit() function for the PVector class.

Exercise 1.5: Create a simulation of a car (or runner) that accelerates when you press the up key and brakes when you press the down key.

Exercise 1.6 Referring back to the Introduction, implement acceleration according to Perlin noise.

Exercise 1.7: Translate the following pseudocode to code using static or non-static functions where appropriate. The PVector v equals (1,5). The PVector u equals v multiplied by 2. The PVector w equals v minus u. Divide the PVector w by 3.

Exercise 1.8: Try implementing the above example with a variable magnitude of acceleration, stronger when it is either closer or farther away.

3. **Program code:**

**Exercise 1.1:**

```
/*
Adrian Copta | FAF-223
Varianta 1
Exercise 1.1
*/

int numRaindrops = 100;  // Number of raindrops
PVector[] positions;     // Positions of the raindrops
PVector[] velocities;    // Velocities of the raindrops
float gravity = 0.2;        // Gravity acting on the
raindrops

void setup() {
  size(400, 400);

  // Initialize arrays for positions and velocities
  positions = new PVector[numRaindrops];
  velocities = new PVector[numRaindrops];

  // Initialize raindrops
  for (int i = 0; i < numRaindrops; i++) {
          positions[i]  =  new  PVector(random(width),
random(height));  // Starting position at random (x, y)
     velocities[i] = new PVector(0, random(1, 5));   //
Initial random velocity
  }
```

```
}

void draw() {
  background(220);

  // Update and draw each raindrop
  for (int i = 0; i < numRaindrops; i++) {
    velocities[i].y += gravity;  // Apply gravity
    positions[i].add(velocities[i]);

    // Check if the raindrop is out of the canvas, reset it to the top
    if (positions[i].y > height) {
      positions[i].y = 0;
      positions[i].x = random(width);
      velocities[i] = new PVector(0, random(1, 5));
    }

    // Draw the raindrop
    stroke(0, 0, 255);  // Blue raindrop
    line(positions[i].x, positions[i].y, positions[i].x, positions[i].y + 10);  // Line representing the raindrop
  }
}
```

**Exercise 1.2:**

```
/*
Adrian Copta | FAF-223
Varianta 1
```

```
Exercise 1.2
*/

void setup() {
  size(400, 400);
  walker = new Walker();
}

void draw() {
  background(220); // Background color to light gray
  walker.step();
  walker.display();
}

class Walker {
  PVector position;
  PVector velocity;

  Walker() {
    position = new PVector(width / 2, height / 2);
        velocity  =  PVector.random2D().mult(2);  //  Set
initial random velocity with a maximum magnitude of 2
  }

  void step() {
    PVector acceleration = PVector.random2D();
    velocity.add(acceleration);
    velocity.limit(2); // Limit the speed
    position.add(velocity);
```

```
    // Wrap around the screen
    if (position.x < 0) position.x = width;
    if (position.x > width) position.x = 0;
    if (position.y < 0) position.y = height;
    if (position.y > height) position.y = 0;
  }

  void display() {
    fill(0); // Set ball color to black
    ellipse(position.x, position.y, 20, 20); // Increase
the ball size to 20x20
  }
}

Walker walker;
```

**Exercise 1.3:**

```
/*
Adrian Copta | FAF-223
Varianta 1
Exercise 1.3
*/

PVector location;
PVector velocity;
float sphereSize = 40; // Increased sphere size
float boxSize = 200;   // Size of the bounding box
color sphereColor;     // Color of the sphere
```

```
color boxColor;         // Color of the bounding box

void setup() {
  size(640, 360, P3D);
   location = new PVector(random(-boxSize/2, boxSize/2),
random(-boxSize/2,     boxSize/2),     random(-boxSize/2,
boxSize/2));
   velocity = new PVector(random(1, 3), random(1, 3),
random(1, 3));
  sphereColor = color(255, 0, 0); // Set sphere color to
red
   boxColor = color(0, 0, 255, 50); // Set box color to
semi-transparent blue
}

void draw() {
  background(240); // Background color to light gray

  location.add(velocity);

   // Check and correct the position if it goes outside
the box
   if (location.x + sphereSize / 2 > boxSize / 2 ||
location.x - sphereSize / 2 < -boxSize / 2) {
    velocity.x = -velocity.x;
     location.x = constrain(location.x, -boxSize / 2 +
sphereSize / 2, boxSize / 2 - sphereSize / 2);
  }
```

```
   if (location.y + sphereSize / 2 > boxSize / 2 ||
location.y - sphereSize / 2 < -boxSize / 2) {
    velocity.y = -velocity.y;
     location.y = constrain(location.y, -boxSize / 2 +
sphereSize / 2, boxSize / 2 - sphereSize / 2);
  }
   if (location.z + sphereSize / 2 > boxSize / 2 ||
location.z - sphereSize / 2 < -boxSize / 2) {
    velocity.z = -velocity.z;
     location.z = constrain(location.z, -boxSize / 2 +
sphereSize / 2, boxSize / 2 - sphereSize / 2);
  }

  stroke(0);
  fill(sphereColor); // Set sphere color to red
  pushMatrix();
  translate(width / 2, height / 2, 0);
  translate(location.x, location.y, location.z);
  sphere(sphereSize); // Draw a fully colored sphere
  popMatrix();

  // Draw the bounding box centered in the sketch
  noFill();
  stroke(0);
  translate(width / 2, height / 2, 0);
  box(boxSize);
}
```

**Exercise 1.4:**

```
/*
Adrian Copta | FAF-223
Varianta 1
Exercise 1.4
*/

void limit(float max) {
  if (mag() > max) {
    normalize();
    mult(max);
  }
}
```

**Exercise 1.5:**

```
/*
Adrian Copta | FAF-223
Varianta 1
Exercise 1.5
*/



PVector position;
PVector velocity;
PVector acceleration;
float carSize = 30;
float maxSpeed = 4;
float accelerationRate = 0.2;
float brakeRate = 0.4;
boolean accelerate = false;
boolean brake = false;
```

```
void setup() {
  size(400, 400);
   position = new PVector(width / 2, height, carSize *
2);
  velocity = new PVector(0, 0);
  acceleration = new PVector(0, 0);
}

void draw() {
  background(220);

  // Check user input
  if (accelerate) {
     acceleration = new PVector(0, -accelerationRate); //
Accelerate upwards
  } else if (brake) {
      acceleration = new PVector(0, brakeRate); // Brake
downwards
  } else {
    acceleration.mult(0); // No input, no acceleration
  }

  // Update velocity and limit speed
  velocity.add(acceleration);
  velocity.limit(maxSpeed);

  // Update position
  position.add(velocity);
```

```
  // Check and correct position if it goes out of bounds
  if (position.y < -carSize) {
    position.y = height;
  }

  // Draw the car
  fill(255, 0, 0);
  rectMode(CENTER);
  pushMatrix();
  translate(position.x, position.y);
  rotate(PI); // Rotate the car 180 degrees to move from
bottom to top
  rect(0, 0, carSize, carSize * 2);
  popMatrix();

 // Draw the car lights
  fill(255, 50, 0);
    ellipse(position.x - carSize / 2, position.y +
carSize, carSize / 4, carSize / 4);
    ellipse(position.x + carSize / 2, position.y +
carSize, carSize / 4, carSize / 4);

  fill(255, 255, 0);
    ellipse(position.x - carSize / 4, position.y -
carSize, carSize / 8, carSize / 8);
    ellipse(position.x + carSize / 4, position.y -
carSize, carSize / 8, carSize / 8);
}
```

```
void keyPressed() {
  if (key == CODED) {
    if (keyCode == UP) {
      accelerate = true;
    } else if (keyCode == DOWN) {
      brake = true;
    }
  }
}

void keyReleased() {
  if (key == CODED) {
    if (keyCode == UP) {
      accelerate = false;
    } else if (keyCode == DOWN) {
      brake = false;
    }
  }
}
```

**Exercise 1.6:**

```
/*
Adrian Copta | FAF-223
Varianta 1
Exercise 1.6
*/
```

```
float tx = 0, ty = 10000; // Initialize noise variables
for x and y
float accelerationX = 0.01; // Initialize initial
acceleration values for x and y
float accelerationY = 0.01;

void setup() {
  size(400, 400); // Set canvas size
}

void draw() {
  background(0); // Background color to black

  // Calculate x and y positions based on Perlin noise
  float x = map(noise(tx), 0, 1, 0, width);
  float y = map(noise(ty), 0, 1, 0, height);

  // Draw an ellipse at the calculated position
  fill(255, 0, 0); // Ellipse color to red
  ellipse(x, y, 16, 16);

  tx += accelerationX;
  ty += accelerationY;

  // Increase acceleration over time
  accelerationX += 0.0001;
  accelerationY += 0.0001;
}
```

**Exercise 1.7:**

```
/*
Adrian Copta | FAF-223
Varianta 1
Exercise 1.7
*/



PVector v = new PVector(1, 5);
PVector u = PVector.mult(v, 2);
PVector w = PVector.sub(v, u);
w.div(3);
```

**Exercise 1.8:**

```
/*
Adrian Copta | FAF-223
Varianta 1
Exercise 1.8
*/

int numCircles = 15; // Number of circles in the array
Circle[] circles = new Circle[numCircles];

void setup() {
  size(400, 400); // Set canvas size to 400x400
  for (int i = 0; i < numCircles; i++) {
    float x = random(width);
    float y = random(height);
    circles[i] = new Circle(x, y);
  }
}

void draw() {
  background(0); // Bbackground color to a black

  for (int i = 0; i < numCircles; i++) {
    circles[i].followMouse();
    circles[i].update();
    circles[i].display();
  }
}
```

```
class Circle {
  PVector location;
  PVector velocity;
  PVector acceleration;
  float topspeed = 4;

  Circle(float x, float y) {
    location = new PVector(x, y);
    velocity = new PVector(0, 0);
    acceleration = new PVector(0, 0);
  }

  void followMouse() {
    PVector mouse = new PVector(mouseX, mouseY);
    PVector dir = PVector.sub(mouse, location);
    dir.normalize();
    dir.mult(0.5);
    acceleration = dir;
  }

  void update() {
    velocity.add(acceleration);
    velocity.limit(topspeed);

    // Update the position without wrapping
    location.add(velocity);
  }

  void display() {
     fill(255, 0, 0); // Circle color to red (RGB 255, 0,
0)
    ellipse(location.x, location.y, 20, 20);
  }
}
```

4. **Program execution:**

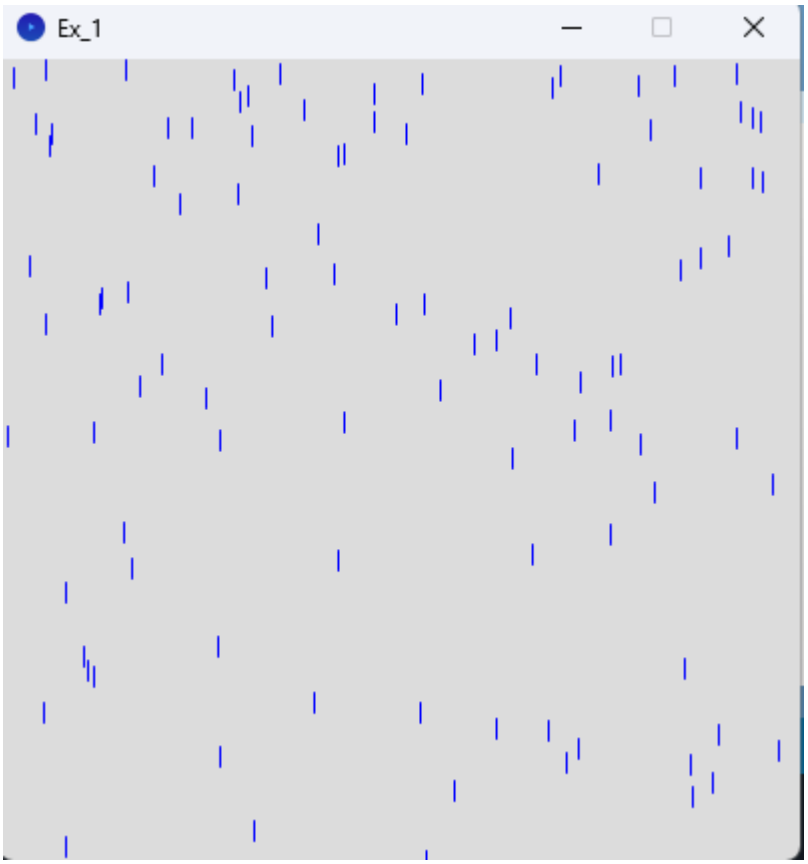

fig.(1) Exercise 1.1 output

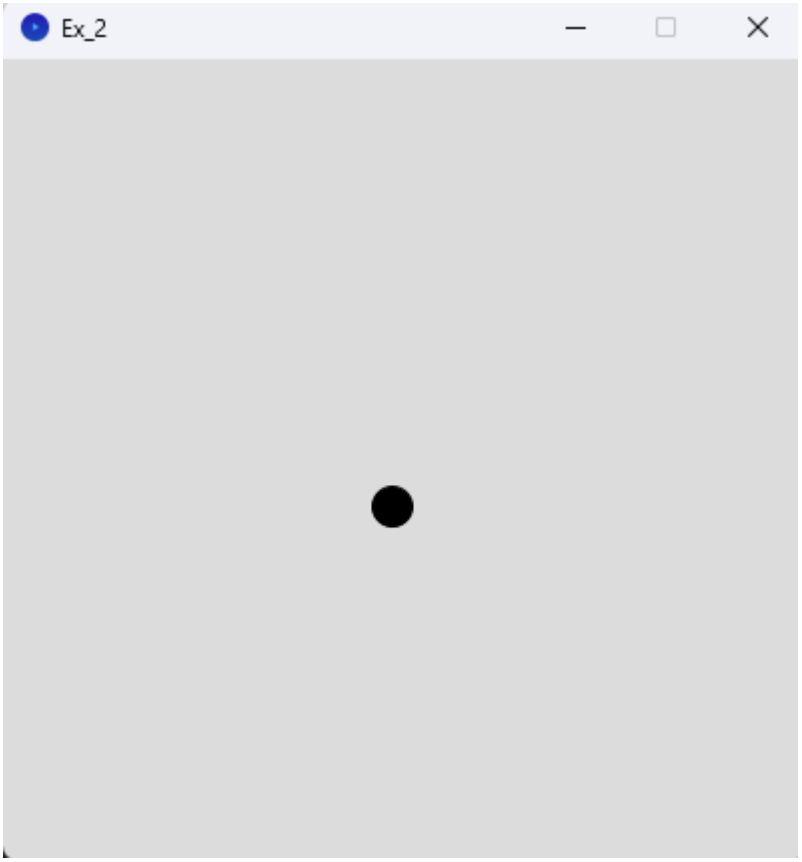

fig.(2) Exercise 1.2 output

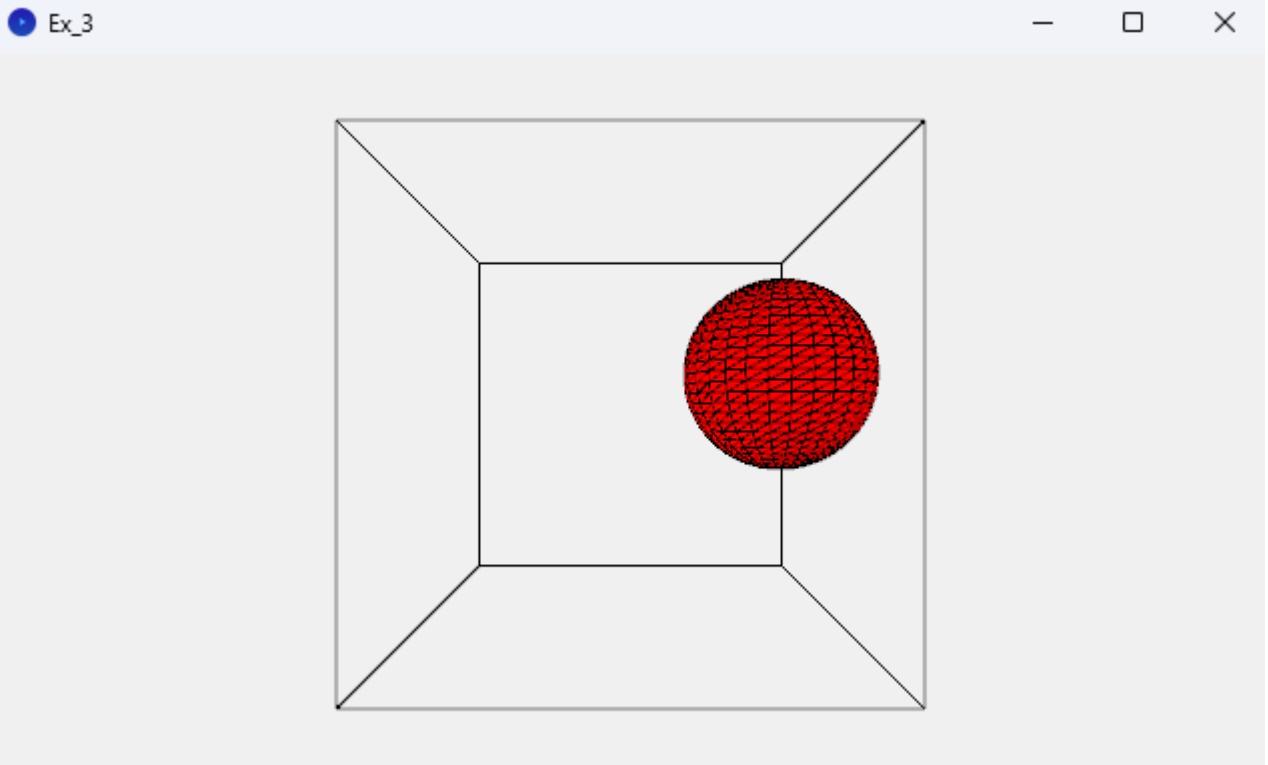


fig.(3) Exercise 1.3 output

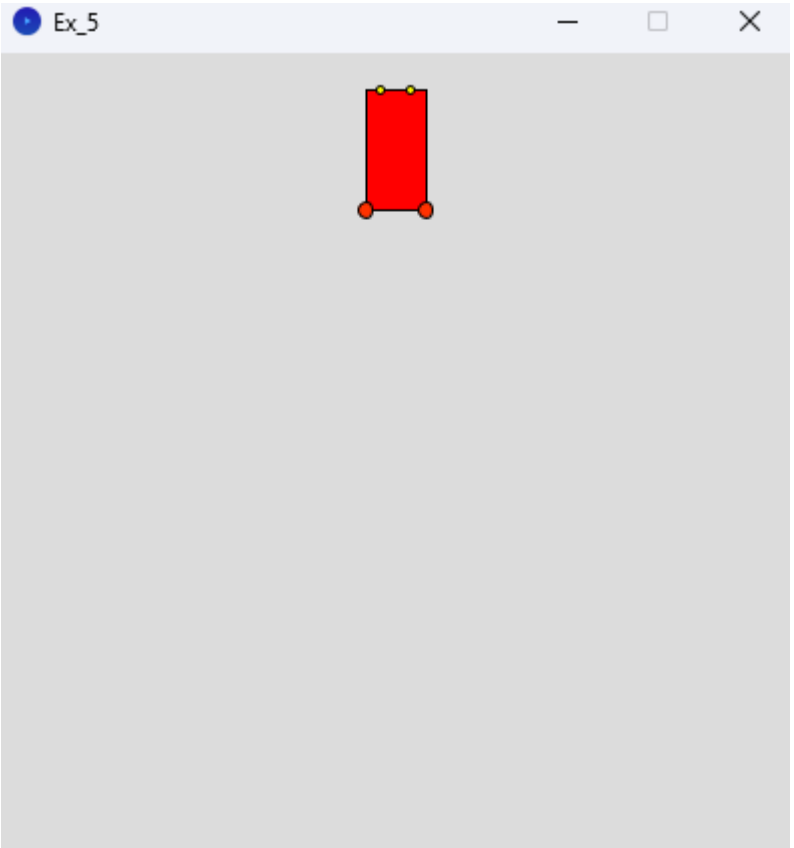


fig.(4) Exercise 1.5 output

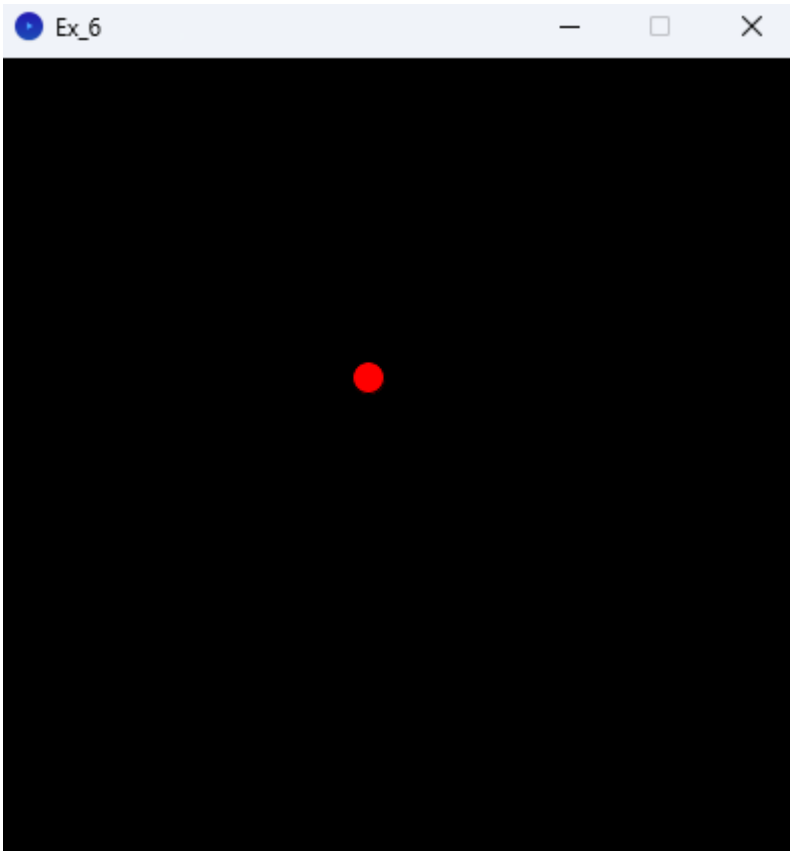


fig.(5) Exercise 1.6 output

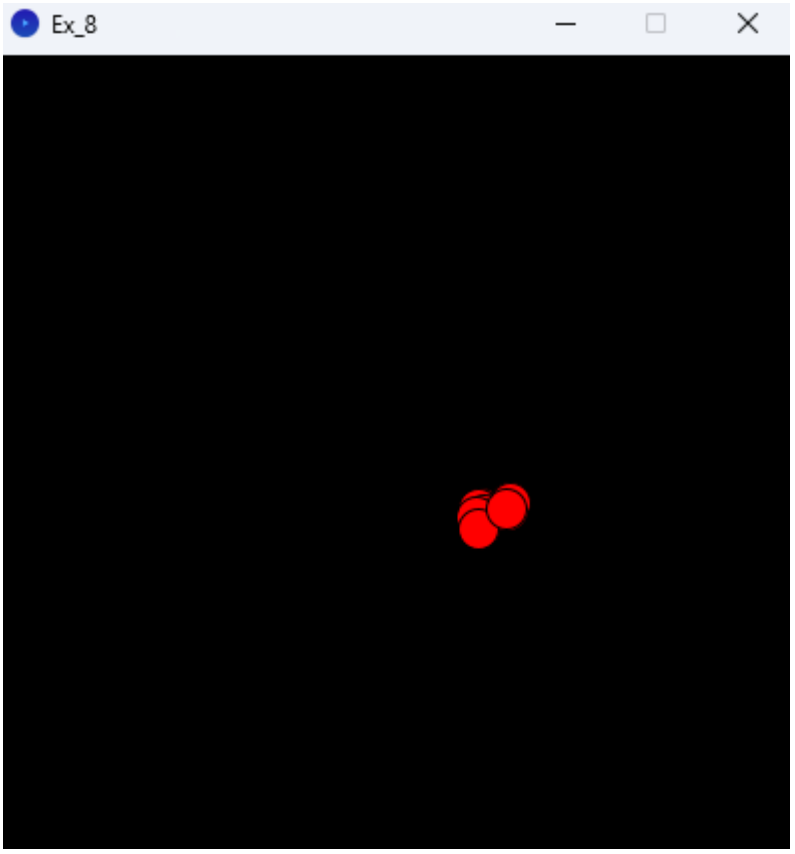


fig.(6) Exercise 1.8 output

5. **Conclusion:**

Conclusion: In this laboratory work, I delved into various exercises aimed at exploring the capabilities of Processing and its PVector class. The exercises provided a valuable opportunity to enhance my understanding of vectors and their applications in creative coding. Here is a summary of what I implemented and learned throughout this laboratory work: Exercise 1.1: I replaced separate x and y variables with PVectors, allowing for more efficient and concise handling of 2D coordinates. This exercise demonstrated the advantages of using PVectors for representing points and simplifying calculations. Exercise 1.2: I transformed a walker example into a PVector-based implementation. By using PVectors to represent the walker's position and to calculate its movement, I gained insight into how vectors can simplify the code and improve readability. Exercise 1.3: I extended the bouncing ball example into the 3D space, enabling a sphere to bounce. This exercise illustrated the versatility of PVectors in handling 3D graphics and dynamics. Exercise 1.4: I implemented a custom limit() function for the PVector class, which allowed me to restrict the magnitude of PVectors. This exercise emphasized the flexibility of Processing and the PVector class for customizing functionality. Exercise 1.5: I created a simulation of a car or runner that could accelerate when the up key was pressed and brake when the down key was pressed. This exercise demonstrated the practical application of PVectors in simulating dynamic systems and user interactions. Exercise 1.6: I implemented acceleration using Perlin noise, as discussed in the introduction. This exercise introduced me to the concept of using noise functions for creating organic and natural-looking motion. Exercise 1.7: I translated pseudocode into code using PVectors, performing vector arithmetic operations. This exercise reinforced my understanding of vector mathematics and their ease of use in Processing. Exercise 1.8: I attempted to implement variable magnitude acceleration based on distance. This exercise introduced the concept of controlling acceleration based on proximity, emphasizing how vectors can be used to create more complex and dynamic simulations. In conclusion, this laboratory work was a valuable journey into the world

of creative coding with Processing and PVectors. I gained a deeper understanding of how vectors can simplify code, improve computational efficiency, and open the door to a wide range of creative possibilities. By completing these exercises, I honed my skills in vector mathematics and learned how to apply them to various scenarios, from simple 2D graphics to complex 3D simulations. This experience has equipped me with essential tools for future creative coding projects and a stronger grasp of vector-based programming.