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.2

of Computer Graphics

Checked by:

Olga Grosu, *university assistant*

DISA, FCIM, UTM

Chișinău – 2023

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

Task 1:

- To make a sketch in Processing with the next conditions.

Task 2:

- Make a sketch with a 2d primitives function and move it.

2. **Condition:**

Task 1:

- the window dimension for the sketch is 400x400, use background for the window – choose the color of it. - use function setup() and draw()
- Draw a rectangle and the line inside - modify the position of x and y coordinates (first two coordinates of the rectangle) in the center of the sketch for drawing it. Next to coordinates rx and ry (half of the width and a half of the height).
- Declarate x,y float variable and rx, ry int variable.
- Use the key word width and height to determinate the center of the sketch. The thickness of the line is 2 for rectangle and the line. Put all this code (Picture1) inside setup() function. Change the color to draw lines and borders around rectangle.
- rx=160 ry=190 (is a half of the width and a half of the height)
- Put next code in a function draw() - The thickness of the line is 3. Change the color for each shape.

a) Draw an arc inside the rectangle in quadrant II

b) Draw a chord inside the rectangle. - angle to start the chord is in the middle of quadrant IV - angle to end the chord is the start of quadrant III

c) Draw a Pie inside the rectangle. - angle to start the pie is the start of quadrant I - angle to end the pie is the middle of quadrant III Change the RX for 10 points in pie function (rx-10; ry-10)

Task 2:

- you can use random function or increase the coordonate of your sketch combination)

- use VARIABLES, CONDITIONALS, LOOPS function and create your own function and call it in Draw function

3. **Program code:**

Task 1 code:

```
/*
MADE BY ADRIAN COPTA | FAF-223
*/

float x, y; // Position of the rectangle
int rx, ry; // Half of the width and half of the height of the rectangle

void setup() {
  size(400, 400);
  background(255, 255, 0); // Yellow background

   // Calculate the center of the sketch using width and height
  x = width / 2;
  y = height / 2;

   // Set half of the width and half of the height of the rectangle
  rx = 160; // Half of 320
  ry = 190; // Half of 380

  stroke(0); // Set stroke color to black
  strokeWeight(2); // Set line thickness to 2
  // Draw the rectangle with borders
  rectMode(CENTER);
  noFill(); // No fill for the rectangle
  rect(x, y, rx * 2, ry * 2);

  // Draw diagonals
  line(x - rx, y - ry, x + rx, y + ry);
  line(x - rx, y + ry, x + rx, y - ry);

  // Draw middle lines
  line(x - rx, y, x + rx, y);
  line(x, y - ry, x, y + ry);
}

void draw() {
  // Set the stroke weight and color for the new shapes
  strokeWeight(3);

  // Draw an arc in quadrant II
  stroke(255, 0, 0); // Red color
  arc(x, y, rx * 2, ry * 2, PI, PI + HALF_PI);

  // Draw a chord
  stroke(0, 255, 0); // Green color
  float startAngle = PI; // Middle of quadrant IV
   float endAngle = PI*2+HALF_PI/2; // Start of quadrant
III
```

```
    arc(x,  y,  (rx-5)  *  2,  (ry-5)  *  2,  startAngle,
endAngle, CHORD);

  // Draw a pie
  stroke(0, 0, 255); // Blue color
   arc(x,  y,  (rx  -  10)  *  2,  (ry  -  10)  *  2,HALF_PI  +
HALF_PI / 2, PI*2, PIE);

  noLoop(); // Stop draw loop
}
```

Task 2 code:

```
/*
MADE BY ADRIAN COPTA | FAF-223
*/

float rectX;          // x-coordinate of the rectangle
float rectY;          // y-coordinate of the rectangle
float rectWidth = 80;  // Width of the rectangle
float rectHeight = 50;  // Height of the rectangle
float rectSpeedX = 2;  // Speed in the x-direction
float rectSpeedY = 2;  // Speed in the y-direction
color rectColor;      // Color of the rectangle

void setup() {
  size(400, 400);
  rectX = random(width - rectWidth);
  rectY = random(height - rectHeight);
```

```
      rectColor   =   color(random(255),   random(255),
random(255));
}

void draw() {
  background(0);

   // Call the custom function to draw the bouncing
rectangle
      drawBouncingRectangle(rectX,   rectY,   rectWidth,
rectHeight, rectColor);

  // Update the rectangle's position based on speed
  rectX += rectSpeedX;
  rectY += rectSpeedY;

  // Check for collisions with the canvas boundaries
  if (rectX <= 0 || rectX >= width - rectWidth) {
       rectSpeedX  *=  -1;   //  Reverse  the  horizontal
direction
         rectColor  =  color(random(255),  random(255),
random(255)); // Change color on collision
  }
  if (rectY <= 0 || rectY >= height - rectHeight) {
    rectSpeedY *= -1;  // Reverse the vertical direction
         rectColor  =  color(random(255),  random(255),
random(255)); // Change color on collision
  }
}
```

```
// Custom function to draw a bouncing rectangle
void drawBouncingRectangle(float x, float y, float
width, float height, color fillColor) {
  fill(fillColor);
  rect(x, y, width, height);
}
```

4. **Program execution:**

Task 1:

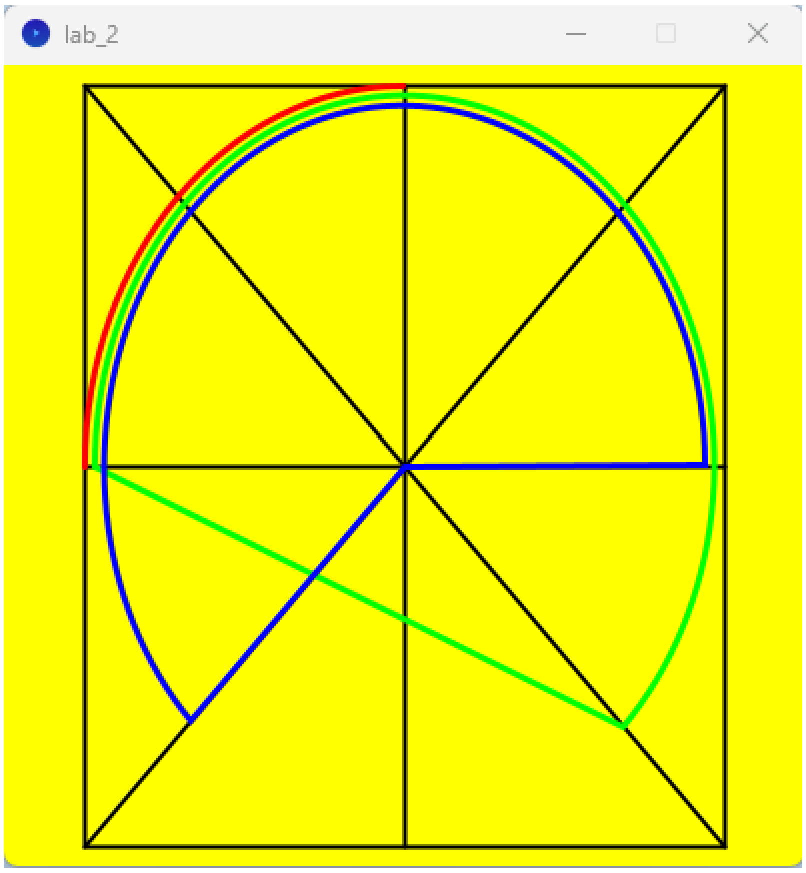

fig.(1)

Task 2:

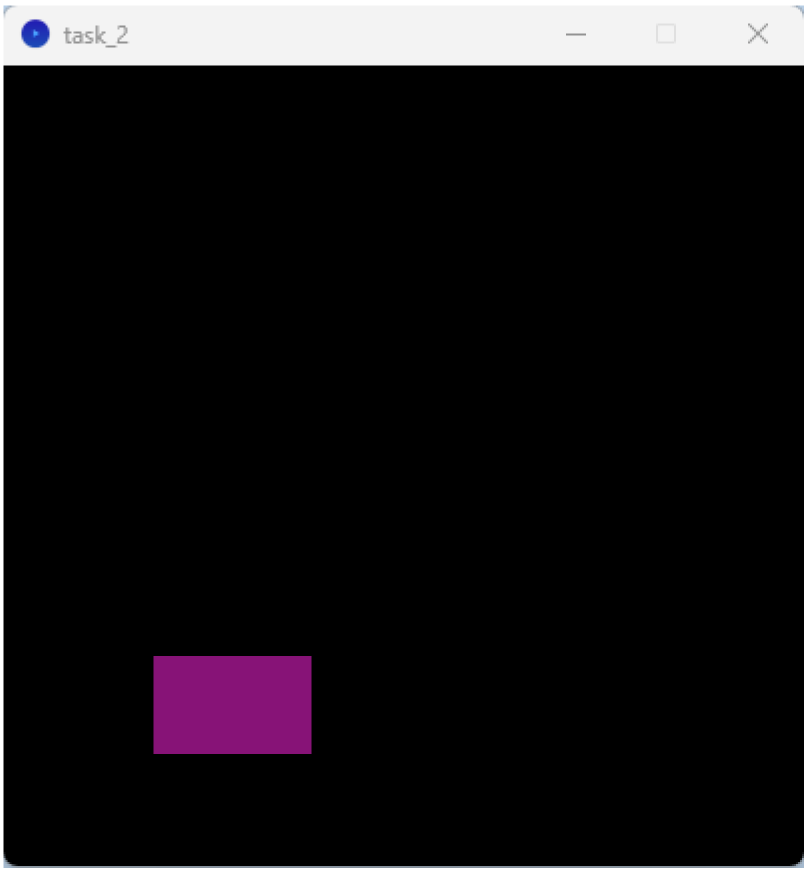

fig.(2)

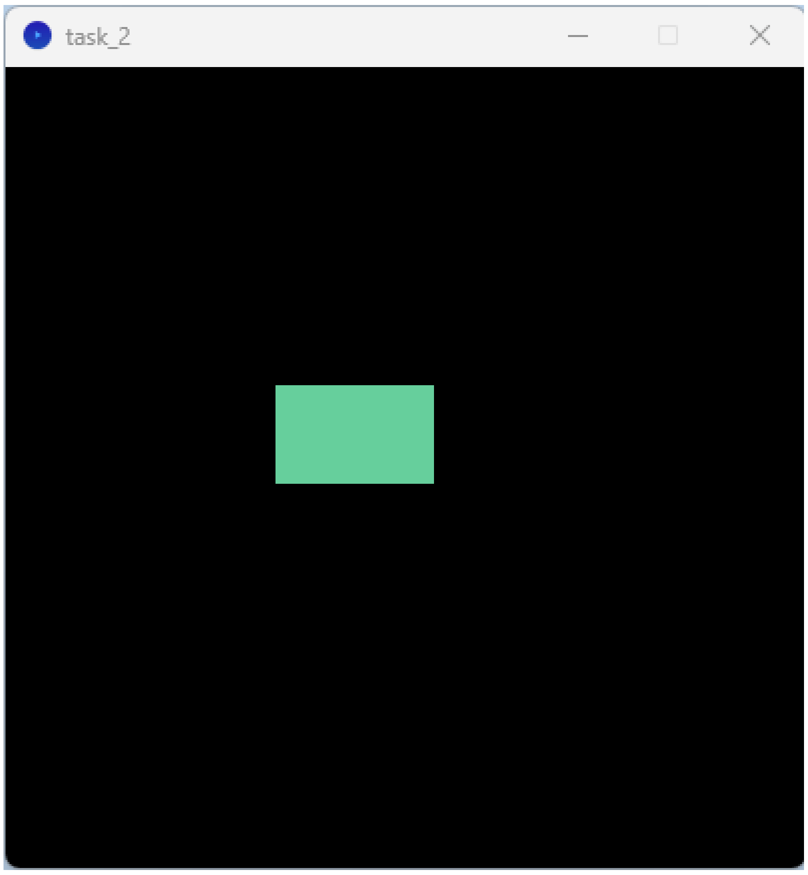

fig.(3)

5. **Conclusion:**

In the first task, we have successfully executed a series of tasks. Each task was meticulously implemented to create a visually engaging composition. We initiated the sketch by defining the position and dimensions of a rectangle. The use of rectMode(CENTER) ensured that subsequent shapes would be drawn symmetrically around their centers. To further enhance our geometric exploration, we drew diagonals and middle lines within the rectangle. These lines not only divide the rectangle into distinct sections but also provide visual references for subsequent shapes. The code expertly drew an arc in Quadrant II of the canvas. By utilizing a striking red color, we highlighted the distinct properties of this arc, extending from π to π + π/2 radians. A chord, positioned in Quadrant III was accurately drawn. The angles and positioning were meticulously calculated, resulting in a precise representation that accentuated our grasp of trigonometric principles. Finally, a pie-shaped region in Quadrant IV was meticulously created. The use of a well-calculated angle range, resulted in an aesthetically pleasing and well-defined shape. Throughout the sketch, we maintained control over stroke weights and colors to ensure visual coherence and clarity. Furthermore, we effectively utilized the noLoop() function to halt the draw loop, preserving the static state of our composition. In conclusion, this Processing sketch exemplifies a successful execution of assigned tasks, showcasing our ability to manipulate geometry and trigonometry within a creative context.

In the second task, the provided Processing sketch demonstrates a simple animation effect using a rectangle. The sketch initializes essential variables, such as the rectangle's position (rectX and rectY), dimensions (rectWidth and rectHeight), speed (rectSpeedX and rectSpeedY), and color (rectColor). The canvas size is set to 400x400 pixels. In the setup() function, the initial values for the rectangle's position, dimensions, and color are randomly generated within the canvas boundaries. This ensures that each run of the sketch begins with a different starting configuration. The core animation takes place in the draw() function, which is continuously executed. The following steps occur within this loop: The background is cleared (set to black), providing the appearance of a clean frame for the animation. The drawBouncingRectangle() custom function is called to draw the rectangle at its current position with its current color. The rectangle's position is updated based on its speed in the x and y directions, causing it to move across the canvas. Collision detection checks whether the rectangle has hit the canvas boundaries. If a collision occurs, the rectangle's direction is reversed (by changing the sign of the speed values), and its color is randomized, providing a visual indication of the bounce. The drawBouncingRectangle() custom function is used to encapsulate the code responsible for drawing the rectangle. It simplifies the main draw() function and enhances code readability. This sketch serves as a basic example of how to create a visually engaging animation using Processing. It demonstrates the use of variables, conditionals, loops, and custom functions to create a dynamic and interactive display. By modifying parameters such as the canvas size, initial conditions, and speed, users can customize the behavior of the bouncing rectangle. Overall, this report illustrates the fundamental principles of animation and interactivity in creative coding using the Processing environment.