******

**TM240: Introduction to Computer Graphics**

**Declaration of No Plagiarism by Student (to be signed and submitted by student with TMA work):**

I hereby declare that this submitted TMA work is a result of my own efforts and I have not plagiarized any other person's work. I have provided all references of information that I have used and quoted in my TMA work.

Name of Student: Omar Hany Ahmed

Signature: Omar Hany

Date:20-11-2024

**Name: Omar Hany Ahmed**

**ID: 21512049**

**Section: 5**

|  |  |
| --- | --- |
| **Question one** |  |

Code:

package floatingboatanimation;

import javax.swing.\*;

import java.awt.\*;

import java.awt.geom.AffineTransform;

import java.awt.geom.GeneralPath;

public class FloatingBoatAnimation extends JPanel implements Runnable {

private double waveOffset = 0;

private double boatAngle = 0;

private double horizontalOffset = 0;

private static final int WIDTH = 800;

private static final int HEIGHT = 600;

private boolean movingRight = true;

public FloatingBoatAnimation() {

setPreferredSize(new Dimension(WIDTH, HEIGHT));

Thread thread = new Thread(this);

thread.start();

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

// Background

g2d.setColor(Color.WHITE);

g2d.fillRect(0, 0, WIDTH, HEIGHT);

// Draw waves

g2d.setColor(Color.BLACK);

drawWaves(g2d);

// Draw boat

drawBoat(g2d);

}

private void drawWaves(Graphics2D g2d) {

int waveHeight = 10;

int waveLength = 150;

int yOffset = HEIGHT / 2;

GeneralPath wavePath = new GeneralPath();

wavePath.moveTo(0, yOffset);

for (int x = 0; x < WIDTH; x++) {

double y = waveHeight \* Math.sin((x + waveOffset) \* 2 \* Math.PI / waveLength);

wavePath.lineTo(x, y + yOffset);

}

g2d.draw(wavePath);

}

private void drawBoat(Graphics2D g2d) {

int boatWidth = 150;

int boatHeight = 50;

int boatX = WIDTH / 2 - boatWidth / 2 + (int) horizontalOffset;

int boatY = HEIGHT / 2 - 60;

double bobbing = 10 \* Math.sin((waveOffset + WIDTH / 2) \* 2 \* Math.PI / 100);

boatY += bobbing;

boatAngle = Math.sin((waveOffset + WIDTH / 2) \* 2 \* Math.PI / 200) \* 30;

g2d.translate(boatX + boatWidth / 2, boatY + boatHeight / 2);

g2d.rotate(Math.toRadians(boatAngle));

g2d.translate(-(boatX + boatWidth / 2), -(boatY + boatHeight / 2));

g2d.setColor(Color.BLACK);

g2d.fillArc(boatX, boatY, boatWidth, boatHeight, 0, -180);

// Draw sails

g2d.setColor(Color.BLACK);

int[] sailX = {boatX + boatWidth / 2, boatX + boatWidth / 8 - -60, boatX + boatWidth / 3 + 60};

int[] sailY = {boatY, boatY - 60, boatY};

g2d.fillPolygon(sailX, sailY, 3);

g2d.setColor(Color.BLACK);

int[] SecondSailX = {boatX + boatWidth / 3, boatX + boatWidth / 8 + 40, boatX + boatWidth / 8 + -20};

int[] Second2SailY = {boatY, boatY - 90, boatY};

g2d.fillPolygon(SecondSailX, Second2SailY, 3);

g2d.setTransform(new AffineTransform());

}

@Override

public void run() {

while (true) {

waveOffset += 2;

// left-right movement

if (movingRight) {

horizontalOffset += 3;

if (horizontalOffset >= 450) { // Change direction at the right edge

movingRight = false;

}

} else {

horizontalOffset -= 3; // Move left

if (horizontalOffset <= -330) { // Change direction at the left edge

movingRight = true;

}

}

repaint();

try {

Thread.sleep(30); // Control animation speed

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public static void main(String[] args) {

JFrame frame = new JFrame("Floating Boat Animation");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.add(new FloatingBoatAnimation());

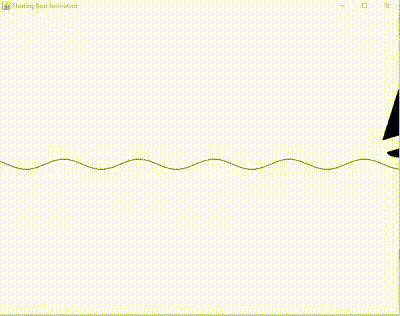
frame.pack();

frame.setVisible(true);

}

}

Output:



|  |  |  |
| --- | --- | --- |
| **Question Two** |  |  |

|  |  |
| --- | --- |
| **Question Three** |  |

|  |  |
| --- | --- |
| **Question Four** |  |

package javaapplication1;

import javax.swing.\*;

import java.awt.\*;

public class Butterfly extends JPanel {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2 = (Graphics2D) g;

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

// Define grid layout dimensions (2x3)

int rows = 2;

int columns = 3;

int cellWidth = getWidth() / columns;

int cellHeight = getHeight() / rows;

// Variants to draw

String[] variants = {"normal", "noisy", "bold", "zigzag", "circular", "dashed"};

for (int row = 0; row < rows; row++) {

for (int col = 0; col < columns; col++) {

int index = row \* columns + col;

if (index < variants.length) {

drawButterflyVariant(g2, variants[index], col \* cellWidth, row \* cellHeight, cellWidth, cellHeight);

}

}

}

}

private void drawButterflyVariant(Graphics2D g2, String variant, int offsetX, int offsetY, int width, int height) {

double scale = 0.65;

double prevX = 0, prevY = 0;

boolean isFirstPoint = true;

// Draw the butterfly curve

for (double t = 0; t <= 12 \* Math.PI; t += 0.01) {

// Parametric equations for the butterfly

double x = Math.sin(t) \* (Math.exp(Math.cos(t)) - 2 \* Math.cos(4 \* t) - Math.pow(Math.sin(t / 12), 5));

double y = Math.cos(t) \* (Math.exp(Math.cos(t)) - 2 \* Math.cos(4 \* t) - Math.pow(Math.sin(t / 12), 5));

double rotatedX = -x;

double rotatedY = -y;

rotatedX \*= scale;

rotatedY \*= scale;

// Translate to fit within the cell

int scaledX = offsetX + (int) (width / 2 + (width / 4) \* rotatedX);

int scaledY = offsetY + (int) (height / 2 - (height / 4) \* rotatedY);

// Skip drawing on the first iteration

if (isFirstPoint) {

prevX = scaledX;

prevY = scaledY;

isFirstPoint = false;

continue;

}

switch (variant) {

case "noisy":

scaledX += (int) ((Math.random() - 0.5) \* 2);

scaledY += (int) ((Math.random() - 0.5) \* 6);

g2.fillOval(scaledX, scaledY, 2, 6);

break;

case "bold":

g2.fillOval(scaledX, scaledY, 5, 5);

break;

case "zigzag":

double zigzagFactor = 0.05 \* Math.sin(400 \* t);

scaledX += (int) (50 \* zigzagFactor);

scaledY += (int) (50 \* zigzagFactor);

g2.drawLine((int) prevX, (int) prevY, scaledX, scaledY);

break;

case "circular":

int circleRadius = 3;

g2.drawOval(scaledX - circleRadius, scaledY - circleRadius, circleRadius \* 2, circleRadius \* 2);

break;

case "dashed":

if ((int) (t \* 200) % 20 < 10) {

g2.drawLine((int) prevX, (int) prevY, scaledX, scaledY);

}

break;

default: // Normal butterfly

g2.drawLine((int) prevX, (int) prevY, scaledX, scaledY);

break;

}

prevX = scaledX;

prevY = scaledY;

}

}

public static void main(String[] args) {

JFrame frame = new JFrame("Butterfly");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(800, 600);

frame.add(new Butterfly());

frame.setVisible(true);

}

}

Output:

