**Федеральное агентство связи**

**Ордена трудового Красного Знамени федеральное государственное бюджетное**

**образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра Математическая кибернетика и информационные технологии

Отчет по лабораторной работе №5

по дисциплине «Объектно-ориентированное программирование»

Выполнил: студент группы БВТ2002

ФИО: Мосева Алеся Сергеевна

Проверил:Полянцева Ксения Андреевна

Москва, 2020

**Цель работы**:расширить приложение из лабораторной работы №4 двумя новыми функциями. Во-первых, добавить поддержку нескольких фракталов и реализовать возможность выбирать нужный фрактал из выпадающего списка. Во-вторых, добавить поддержку сохранения текущего изображения в файл.

**Код программы:**

import java.awt.geom.Rectangle2D;

public class BurningShip extends FractalGenerator{

int MAX\_ITERATIONS = 2000;

@Override

public void getInitialRange(Rectangle2D.Double range) {

range.x = -2;

range.y = -2.5;

range.height = 4;

range.width = 4;

}

@Override

public int numIterations(double x, double y) {

int iteration = 0;

double z\_real = 0;

double z\_imaginary = 0;

while (iteration < MAX\_ITERATIONS && z\_real \* z\_real + z\_imaginary \* z\_imaginary < 4)

{

double z\_realUpdated = z\_real \* z\_real - z\_imaginary \* z\_imaginary + x;

double z\_imaginaryUpdated = 2 \* Math.abs(z\_real) \* Math.abs(z\_imaginary) + y;

z\_real = z\_realUpdated;

z\_imaginary = z\_imaginaryUpdated;

iteration += 1;

}

if (iteration == MAX\_ITERATIONS)

{

return -1;

}

return iteration;

}

public String toString(){

return "BurningShip";

}

}

import javax.imageio.ImageIO;

import javax.swing.\*;

import javax.swing.filechooser.FileFilter;

import javax.swing.filechooser.FileNameExtensionFilter;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.geom.Rectangle2D;

import java.awt.image.BufferedImage;

import java.io.File;

public class FractalExplorer {

private int size;

private JImageDisplay jDisplay;

private FractalGenerator fractal;

private Rectangle2D.Double range;

public FractalExplorer (int display\_size) {

size = display\_size;

range = new Rectangle2D.Double();

fractal = new Mandelbrot();

fractal.getInitialRange(range);

jDisplay = new JImageDisplay(display\_size, display\_size);

}

public void createAndShowGUI () {

jDisplay.setLayout(new BorderLayout());

JFrame frame = new JFrame();

frame.add(jDisplay, BorderLayout.CENTER);

JButton resetButton = new JButton("Reset");

ResetButtonHandler clearAction = new ResetButtonHandler();

resetButton.addActionListener(clearAction);

MyMouseListener mouse = new MyMouseListener();

jDisplay.addMouseListener(mouse);

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

String[] items = {"Mandelbrot", "Tricorn", "BurningShip"};

JComboBox comboBox = new JComboBox(items);

JLabel label = new JLabel("Fractal:");

JPanel panelBox = new JPanel();

panelBox.add(label);

panelBox.add(comboBox);

frame.add(panelBox, BorderLayout.NORTH);

ChooseButtonHandler chooseAction = new ChooseButtonHandler();

comboBox.addActionListener(chooseAction);

JButton saveButton = new JButton("Save Image");

SaveImageButton saveAction = new SaveImageButton();

saveButton.addActionListener(saveAction);

JPanel panelButtons = new JPanel();

panelButtons.add(resetButton);

panelButtons.add(saveButton);

frame.add(panelButtons, BorderLayout.SOUTH);

frame.pack();

frame.setVisible(true);

frame.setResizable(false);

}

private void drawFractal () {

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

for (int y = 0; y < size; y ++) {

double xCoord = fractal.getCoord(range.x,range.x + range.width, size, x);

double yCoord = fractal.getCoord(range.y, range.y + range.height, size, y);

int iterations = fractal.numIterations(xCoord,yCoord);

if (iterations == -1) jDisplay.drawPixel(x,y,0);

else {

float hue = 0.7f + (float) iterations / 200f;

int rgbColor = Color.HSBtoRGB(hue, 1f, 1f);

jDisplay.drawPixel(x, y, rgbColor);

}

}

}

jDisplay.repaint();

}

public class ResetButtonHandler implements ActionListener {

@Override

public void actionPerformed (ActionEvent e) {

fractal.getInitialRange(range);

drawFractal();

}

}

public class ChooseButtonHandler implements ActionListener{

@Override

public void actionPerformed(ActionEvent e) {

JComboBox combo = (JComboBox)e.getSource();

String name = (String) combo.getSelectedItem();

if (name == "Mandelbrot"){

fractal = new Mandelbrot();

}

if (name == "Tricorn") {

fractal = new Tricorn();

}

if (name == "BurningShip") {

fractal = new BurningShip();

}

fractal.getInitialRange(range);

drawFractal();

}

}

public class SaveImageButton implements ActionListener{

@Override

public void actionPerformed(ActionEvent e) {

JFileChooser chooser = new JFileChooser();

FileFilter filter = new FileNameExtensionFilter("PNG Images", "png");

chooser.setFileFilter(filter);

chooser.setAcceptAllFileFilterUsed(false);

int result = chooser.showSaveDialog(jDisplay);

if (result == JFileChooser.APPROVE\_OPTION) {

File dir = chooser.getSelectedFile();

String dir\_string = dir.toString();

try{

BufferedImage image = jDisplay.getImage();

ImageIO.write(image, "png", dir);

}

catch(Exception exception){

JOptionPane.showMessageDialog(chooser, exception.getMessage(),"Can not save image", JOptionPane.ERROR\_MESSAGE);

}

}

}

}

private class MyMouseListener extends MouseAdapter {

@Override

public void mouseClicked(MouseEvent e) {

int x = e.getX();

double xCoord = fractal.getCoord(range.x, range.x+range.width, size,x);

int y = e.getY();

double yCoord = fractal.getCoord(range.y, range.y+range.height, size,y);

fractal.recenterAndZoomRange(range, xCoord, yCoord, 0.5);

drawFractal();

}

}

public static void main (String[] args) {

FractalExplorer display = new FractalExplorer(800);

display.createAndShowGUI();

display.drawFractal();

}

}

import java.awt.geom.Rectangle2D;

/\*\*

\* This class provides the common interface and operations for fractal

\* generators that can be viewed in the Fractal Explorer.

\*/

public abstract class FractalGenerator {

/\*\*

\* This static helper function takes an integer coordinate and converts it

\* into a double-precision value corresponding to a specific range. It is

\* used to convert pixel coordinates into double-precision values for

\* computing fractals, etc.

\*

\* @param rangeMin the minimum value of the floating-point range

\* @param rangeMax the maximum value of the floating-point range

\*

\* @param size the size of the dimension that the pixel coordinate is from.

\* For example, this might be the image width, or the image height.

\*

\* @param coord the coordinate to compute the double-precision value for.

\* The coordinate should fall in the range [0, size].

\*/

public static double getCoord(double rangeMin, double rangeMax,

int size, int coord) {

assert size > 0;

assert coord >= 0 && coord < size;

double range = rangeMax - rangeMin;

return rangeMin + (range \* (double) coord / (double) size);

}

/\*\*

\* Sets the specified rectangle to contain the initial range suitable for

\* the fractal being generated.

\*/

public abstract void getInitialRange(Rectangle2D.Double range);

/\*\*

\* Updates the current range to be centered at the specified coordinates,

\* and to be zoomed in or out by the specified scaling factor.

\*/

public void recenterAndZoomRange(Rectangle2D.Double range,

double centerX, double centerY, double scale) {

double newWidth = range.width \* scale;

double newHeight = range.height \* scale;

range.x = centerX - newWidth / 2;

range.y = centerY - newHeight / 2;

range.width = newWidth;

range.height = newHeight;

}

/\*\*

\* Given a coordinate <em>x</em> + <em>iy</em> in the complex plane,

\* computes and returns the number of iterations before the fractal

\* function escapes the bounding area for that point. A point that

\* doesn't escape before the iteration limit is reached is indicated

\* with a result of -1.

\*/

public abstract int numIterations(double x, double y);

}

import javax.swing.\*;

import java.awt.image.\*;

import java.awt.\*;

public class JImageDisplay extends JComponent{

private BufferedImage image;

public BufferedImage getImage(){

return image;

}

public JImageDisplay (int h, int w) {

image = new BufferedImage(h,w, BufferedImage.TYPE\_INT\_RGB);

Dimension size = new Dimension(h,w);

super.setPreferredSize(size);

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

g.drawImage(image,0,0,image.getWidth(),image.getHeight(),null);

}

public void clearImage() {

for (int i=0; i < image.getHeight(); i++ ) {

for (int j = 0; j < image.getWidth(); j ++) {

drawPixel(i,j,0);

}

}

}

public void drawPixel (int x, int y, int rgbColor) {

image.setRGB(x,y,rgbColor);

}

}

import java.awt.geom.Rectangle2D;

public class Mandelbrot extends FractalGenerator {

public static final int MAX\_ITERATIONS = 2000;

@Override

public void getInitialRange(Rectangle2D.Double range) {

range.x = -2;

range.y = -1.5;

range.width = 3;

range.height = 3;

}

@Override

public int numIterations(double x, double y) {

int iteration = 0;

double z\_real = 0;

double z\_imaginary = 0;

while (iteration < MAX\_ITERATIONS && z\_real \* z\_real + z\_imaginary \* z\_imaginary < 4)

{

double z\_realUpdated = z\_real \* z\_real - z\_imaginary \* z\_imaginary + x;

double z\_imaginaryUpdated = 2 \* z\_real \* z\_imaginary + y;

z\_real = z\_realUpdated;

z\_imaginary = z\_imaginaryUpdated;

iteration += 1;

}

if (iteration == MAX\_ITERATIONS)

{

return -1;

}

return iteration;

}

public String toString(){

return "Mandelbrot";

}

}

import java.awt.geom.Rectangle2D;

public class Tricorn extends FractalGenerator{

public static final int MAX\_ITERATIONS = 2000;

@Override

public void getInitialRange(Rectangle2D.Double range) {

range.x = -2;

range.y = -2;

range.width = 4;

range.height = 4;

}

@Override

public int numIterations(double x, double y) {

int iteration = 0;

double z\_real = 0;

double z\_imaginary = 0;

while (iteration < MAX\_ITERATIONS && z\_real \* z\_real + z\_imaginary \* z\_imaginary < 4)

{

double z\_realUpdated = z\_real \* z\_real - z\_imaginary \* z\_imaginary + x;

double z\_imaginaryUpdated = -2\* z\_real \* z\_imaginary + y;

z\_real = z\_realUpdated;

z\_imaginary = z\_imaginaryUpdated;

iteration += 1;

}

if (iteration == MAX\_ITERATIONS)

{

return -1;

}

return iteration;

}

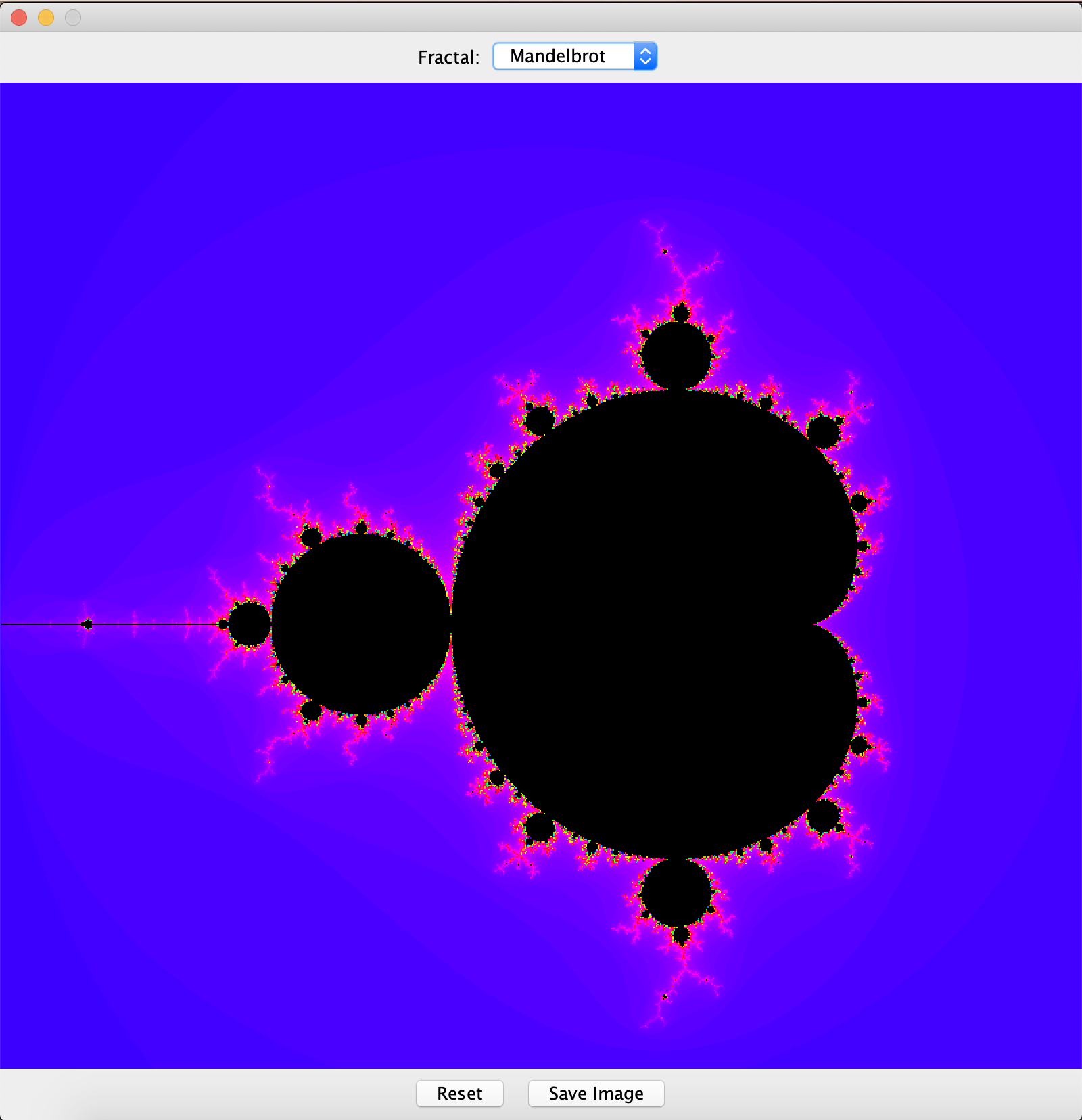
public String toString(){

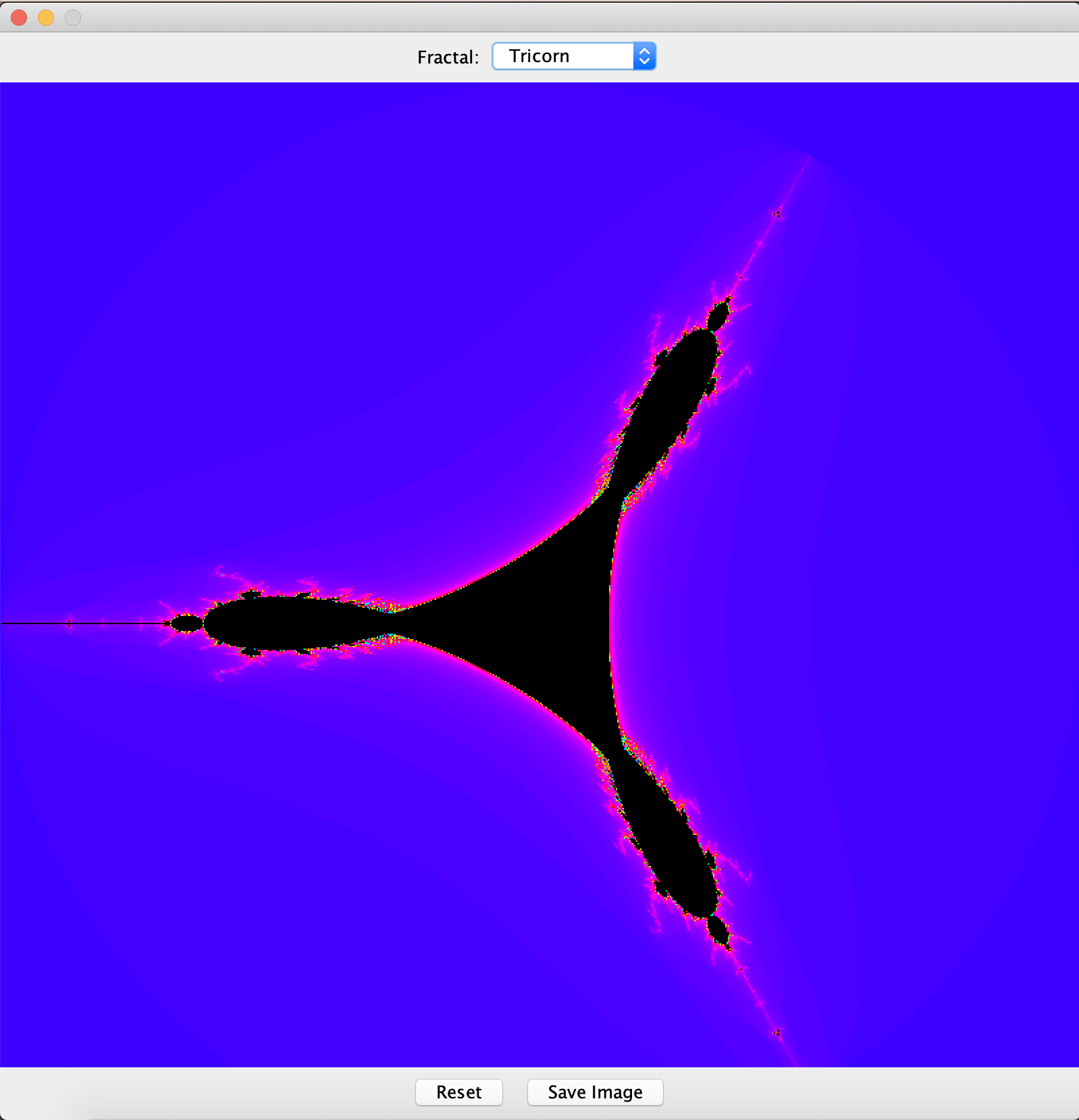
return "Tricorn";

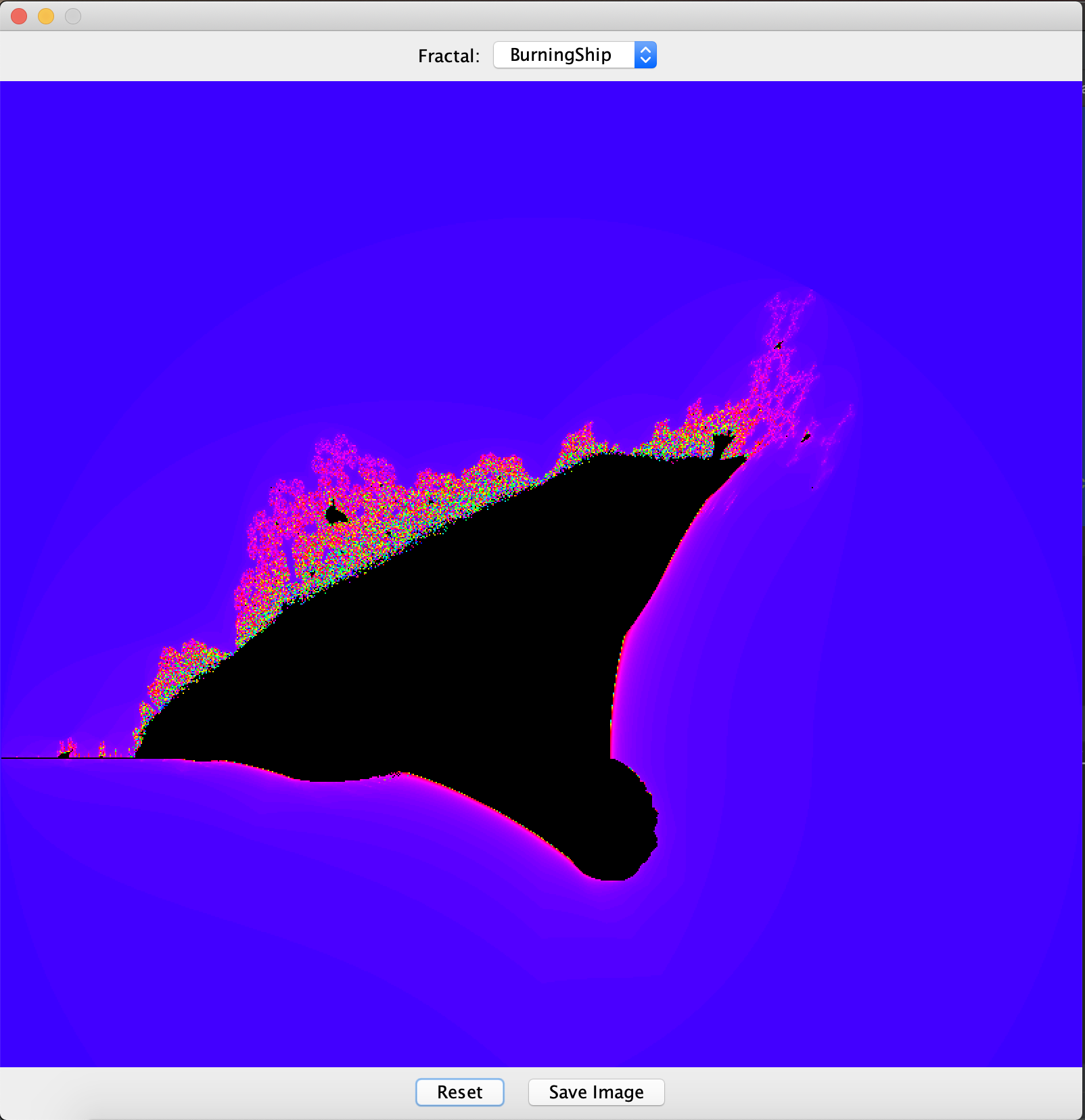
}

}

**Скриншоты выполнения:**

****

****

****

**Заключение:**

Таким образом, мы расширили функционал приложения, реализованного в прошлой лабораторной работе.