Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного знамени

Федеральное государственное бюджетное учреждение

высшего образования

«Московский технический университет связи и информатики»

Кафедра МКиТ

Лабораторная работа № 4, 5, 6

«Рисование фракталов», «Выбор и сохранение фракталов», «Многопоточный генератор фракталов»

Выполнил студент

Группы БСТ1903

Самоваров А.П.

Вариант № 12

Москва 2021

**Оглавление**

[1. Цель лабораторной работы 3](#_Toc72367615)

[2. Задание на лабораторную работу 3](#_Toc72367616)

[3. Код программы 5](#_Toc72367617)

[4. Результат работы программы 14](#_Toc72367618)

[Вывод 17](#_Toc72367619)

1. Цель лабораторной работы

Реализовать простой веб-сканнер, который будет автоматически загружать веб-страницы из Интернета, искать на этих страницах новые ссылки и повторять их. Реализовать многопоточность.

2. Задание на лабораторную работу

Создать класс FractalExplorer, который позволит исследовать различные области фрактала, путем его создания, отображения через графический интерфейс Swing и обработки событий, вызванных взаимодействием приложения с пользователем.

.• Класс FractalExplorer должен отслеживать несколько важных полей для состояния программы:

1) Целое число «размер экрана», которое является шириной и высотой отображения в пикселях. (Отображение фрактала будет квадратным.)

2) Ссылка JImageDisplay, для обновления отображения в разных методах в процессе вычисления фрактала.

3) Объект FractalGenerator. Будет использоваться ссылка на базовый класс для отображения других видов фракталов в будущем.

4) Объект Rectangle2D.Double, указывающий диапазона комплексной плоскости, которая выводится на экран. Все вышеприведенные поля будут иметь тип доступа private.

• У класса должен быть конструктор, который принимает значение размера отображения в качестве аргумента, затем сохраняет это значение в соответствующем поле, а также инициализирует объекты диапазона и фрактального генератора. Данный конструктор не должен устанавливать какие-либо компоненты Swing; они будут установлены в следующем методе.

• Создать метод createAndShowGUI (), который инициализирует графический интерфейс Swing: JFrame, содержащий объект JimageDisplay, и кнопку для сброса отображения. Необходимо использовать java.awt.BorderLayout для содержимого окна; добавить объект отображения изображения в позицию BorderLayout.CENTER и кнопку в позицию BorderLayout.SOUTH.

• Реализовать вспомогательный метод с типом доступа private для вывода на экран фрактала, можете дать ему имя drawFractal ().

• Создать внутренний класс для обработки событий java.awt.event.ActionListener от кнопки сброса. Обработчик должен сбросить диапазон к начальному, определенному генератором, а затем перерисовать фрактал. После того, как создан этот класс, необходимо обновить метод createAndShowGUI ().

• Создать другой внутренний класс для обработки событий java.awt.event.MouseListener с дисплея.

• В заключении, необходимо создать статический метод main() для FractalExplorer так, чтобы можно было его запустить.

1) Также необходимо добавить поддержку нескольких фракталов и реализовать возможность выбирать нужный фрактал из выпадающего списка (элемент comboBox).

2) Добавить поддержку сохранения текущего изображения фрактала в файл.

3) Реализовать многопоточность.

3. Код программы

Код для класса FractalExplorer (в данных работах изменяется только данный класс):

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.io.IOException;

public class FractalExplorer extends JFrame {

private int size;

private JImageDisplay imageDisplay;

private FractalGenerator fractal;

private Rectangle2D.Double range;

private int rowRemaining;

JButton button;

JButton save;

JComboBox comboBox;

public static void main(String[] args) {

FractalExplorer fractalExplorer = new FractalExplorer(600);

fractalExplorer.creatAndShowGUI();

fractalExplorer.drawFractal();

}

FractalExplorer(int size){

this.size = size;

range = new Rectangle2D.Double();

fractal = new Mandelbrot();

fractal.getInitialRange(range);

imageDisplay = new JImageDisplay(size,size);

}

private void creatAndShowGUI(){

JFrame frame = new JFrame("Fractal");

Mandelbrot mandelbrot = new Mandelbrot();

Tricorn tricorn = new Tricorn();

BurningShip burningShip = new BurningShip();

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

JPanel northpanel = new JPanel();

JPanel southpanel = new JPanel();

//JComboBox comboBox = new JComboBox(fractals);

button = new JButton("Reset");

save = new JButton("Save");

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

button.setActionCommand("reset");

save.setActionCommand("save");

comboBox = new JComboBox();

comboBox.addItem(mandelbrot);

comboBox.addItem(tricorn);

comboBox.addItem(burningShip);

EventHandler eventHandler = new EventHandler();

MouseHandler mouseHandler = new MouseHandler();

imageDisplay.addMouseListener(mouseHandler);

button.addActionListener(eventHandler);

save.addActionListener(eventHandler);

comboBox.addActionListener(eventHandler);

imageDisplay.setLayout(new BorderLayout());

northpanel.add(label);

northpanel.add(comboBox);

southpanel.add(save);

southpanel.add(button);

frame.add(imageDisplay,BorderLayout.CENTER);

frame.add(northpanel,BorderLayout.NORTH);

frame.add(southpanel,BorderLayout.SOUTH);

frame.setSize(size,size + 100);

frame.pack();

frame.setVisible(true);

frame.setResizable(false);

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

}

private void drawFractal(){

rowRemaining = size;

enableUI(false);

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

FractalWorker fractalWorker = new FractalWorker(i);

fractalWorker.execute();

}

}

private void enableUI(boolean val){

button.setEnabled(val);

save.setEnabled(val);

comboBox.setEnabled(val);

}

private class EventHandler implements ActionListener {

public void actionPerformed(ActionEvent e) {

Object object = e.getSource();

if(object instanceof JComboBox){

fractal = (FractalGenerator) ((JComboBox) object).getSelectedItem();

fractal.getInitialRange(range);

drawFractal();

}

else if(object instanceof JButton) {

JButton button = (JButton) object;

if(button.getActionCommand().equals("reset")){

fractal.getInitialRange(range);

drawFractal();

}

else if (button.getActionCommand().equals("save")){

JFileChooser fileChooser = new JFileChooser();

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

fileChooser.setFileFilter(filter);

fileChooser.setAcceptAllFileFilterUsed(false);

if(fileChooser.showSaveDialog(button.getParent())

!= JFileChooser.APPROVE\_OPTION)return;

try {

ImageIO.write(imageDisplay.getBufferedImage(),"png",

fileChooser.getSelectedFile());

} catch (IOException ex) {

JOptionPane.showMessageDialog(button.getParent(), ex.getMessage(),

"Cannot Save Image", JOptionPane.ERROR\_MESSAGE);

}

}

}

}

}

private class MouseHandler extends MouseAdapter{

@Override

public void mouseClicked(MouseEvent e)

{

if(rowRemaining != 0)return;

int x = e.getX();

int y = e.getY();

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

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

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

drawFractal();

}

}

private class FractalWorker extends SwingWorker<Object, Object>{

private int y;

private int[] pixelRow;

FractalWorker(int yCoord){

y = yCoord;

}

@Override

protected Object doInBackground() throws Exception {

pixelRow = new int[size];

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

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

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

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

if (iter == -1)pixelRow[i] = 0;

else {

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

pixelRow[i] = Color.HSBtoRGB(hue, 1f, 1f);

}

}

return null;

}

@Override

protected void done() {

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

imageDisplay.drawPixel(i,y,pixelRow[i]);

}

imageDisplay.repaint(0,0,y,size,1);

rowRemaining--;

if(rowRemaining == 0) enableUI(true);

}

}

}

Код для класса JImageDisplay:

import java.awt.\*;

import java.awt.image.BufferedImage;

public class JImageDisplay extends javax.swing.JComponent {

private BufferedImage bufferedImage;

JImageDisplay(int width, int height){

bufferedImage = new BufferedImage(width,height,bufferedImage.TYPE\_INT\_RGB);

this.setPreferredSize(new Dimension(width,height));

}

@Override

public void paintComponent(Graphics g)

{

super.paintComponent(g);

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

}

public void clearImage(){

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

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

bufferedImage.setRGB(i,j,0);

}

}

}

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

bufferedImage.setRGB(x,y,rgbColor);

}

public BufferedImage getBufferedImage() {

return bufferedImage;

}

}

Код для класса FractalGenerator:

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.

\*

\*/

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);

}

Код для класса Mandelbrot (первый фрактал, реализованный в 4 работе):

import java.awt.geom.Rectangle2D;

public class Mandelbrot extends FractalGenerator {

public static final int MAX\_ITERATIONS = 2000;

public void getInitialRange(Rectangle2D.Double range) {

range.x = -2;

range.y = -1.5;

range.height = 3;

range.width = 3;

}

public int numIterations(double x, double y) {

int iteraion = 0;

double x1 = 0;

double y1 = 0;

while (x1\*x1 + y1\*y1 < 4 && iteraion < MAX\_ITERATIONS) {

double temp1 = x1 \* x1 - y1 \* y1 + x;

double temp2 = 2 \* x1 \* y1 + y;

x1 = temp1;

y1 = temp2;

iteraion++;

}

if (iteraion >= MAX\_ITERATIONS)return -1;

return iteraion;

}

public String toString() {

return "Mandelbrot";

}

}

Код для класса Tricorn (второй фрактал, реализованный в 5 работе):

import java.awt.geom.Rectangle2D;

public class Tricorn extends FractalGenerator {

public static final int MAX\_ITERATIONS = 2000;

public String toString() {

return "Tricorn";

}

public void getInitialRange(Rectangle2D.Double range) {

range.x = -2;

range.y = -2;

range.height = 4;

range.width = 4;

}

public int numIterations(double x, double y) {

int iteraion = 0;

double x1 = 0;

double y1 = 0;

while (x1\*x1 + y1\*y1 < 4 && iteraion < MAX\_ITERATIONS) {

double temp1 = x1 \* x1 - y1 \* y1 + x;

double temp2 = -2 \* x1 \* y1 + y;

x1 = temp1;

y1 = temp2;

iteraion++;

}

if (iteraion >= MAX\_ITERATIONS)return -1;

return iteraion;

}

}

Код для класса BurningShip (третий фрактал, реализованный в лабораторной работе № 5):

import java.awt.geom.Rectangle2D;

public class BurningShip extends FractalGenerator {

public static final int MAX\_ITERATIONS = 2000;

public String toString() {

return "BurningShip";

}

public void getInitialRange(Rectangle2D.Double range) {

range.x = -2;

range.y = -2.5;

range.height = 4;

range.width = 4;

}

public int numIterations(double x, double y) {

int iteraion = 0;

double x1 = 0;

double y1 = 0;

while (x1\*x1 + y1\*y1 < 4 && iteraion < MAX\_ITERATIONS) {

double temp1 = x1 \* x1 - y1 \* y1 + x;

double temp2 = Math.abs(2 \* x1 \* y1) + y;

x1 = temp1;

y1 = temp2;

iteraion++;

}

if (iteraion >= MAX\_ITERATIONS)return -1;

return iteraion;

}

}

4. Результат работы программы

На рисунке 1 представлен запуск программы через консоль.

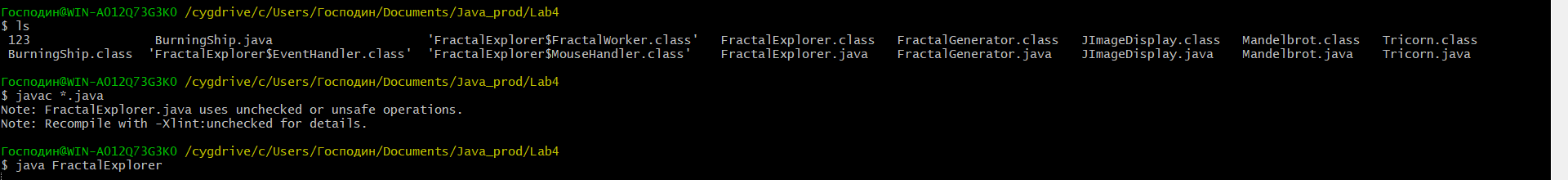


Рисунок 1 – Запуск программы через консоль

На рисунке 2 представлено окно, которое появляется после запуска программы.

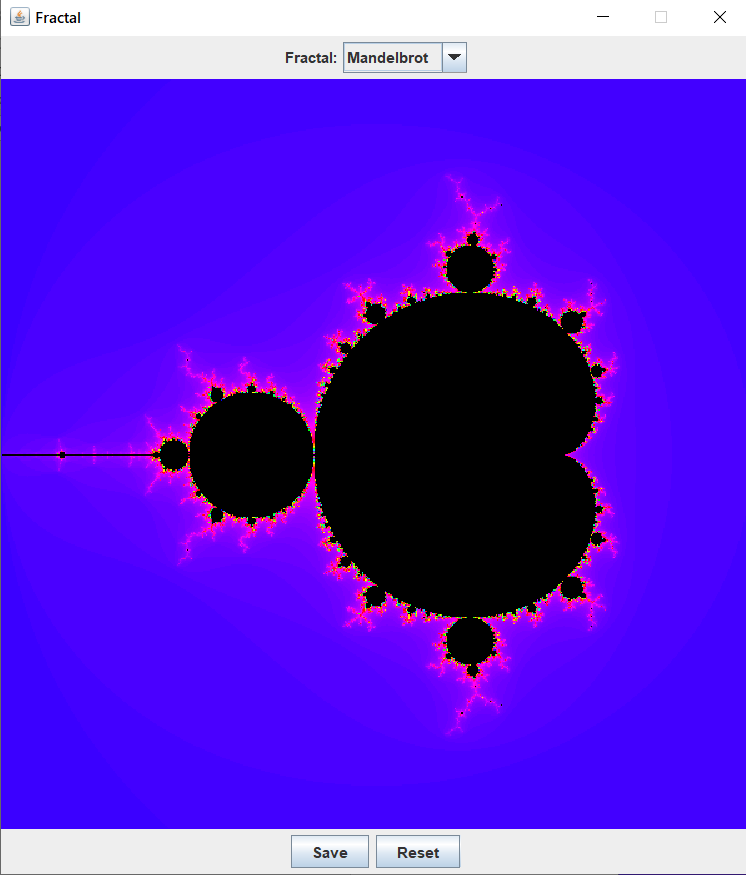


Рисунок 2 – Фрактал Мандельброта

На данном рисунке видно выпадающий список, а также кнопки сохранения и перезапуска фрактала.

На рисунке 2 представлен третий фрактал «Tricorn», а также показана работа выпадающего списка.

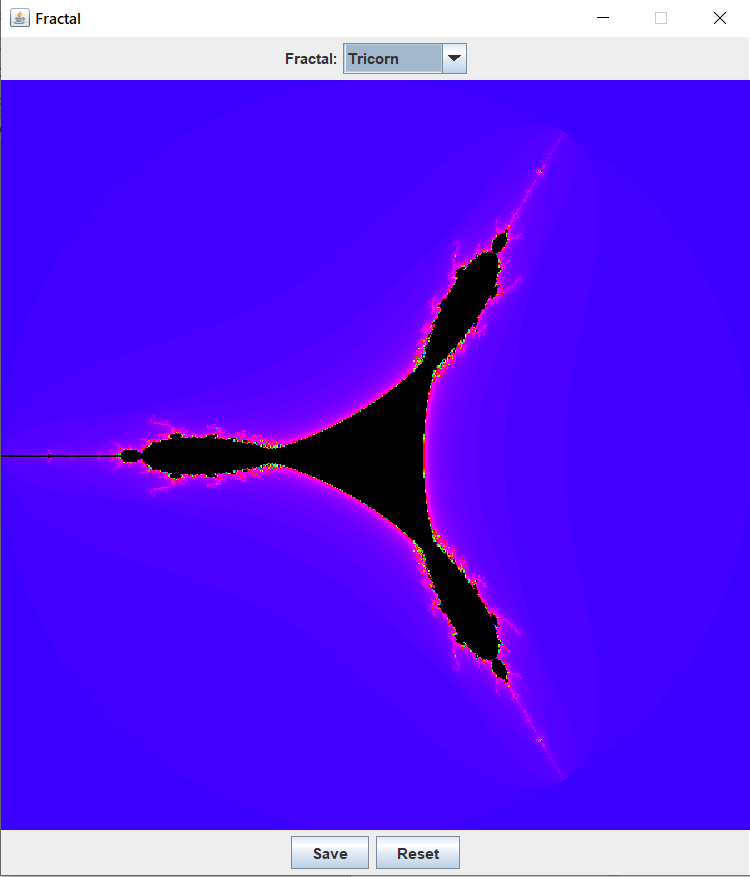


Рисунок 3 – Второй фрактал и выпадающий список

Далее на рисунке 4 представлена реакция программы на нажатие кнопки «Save».

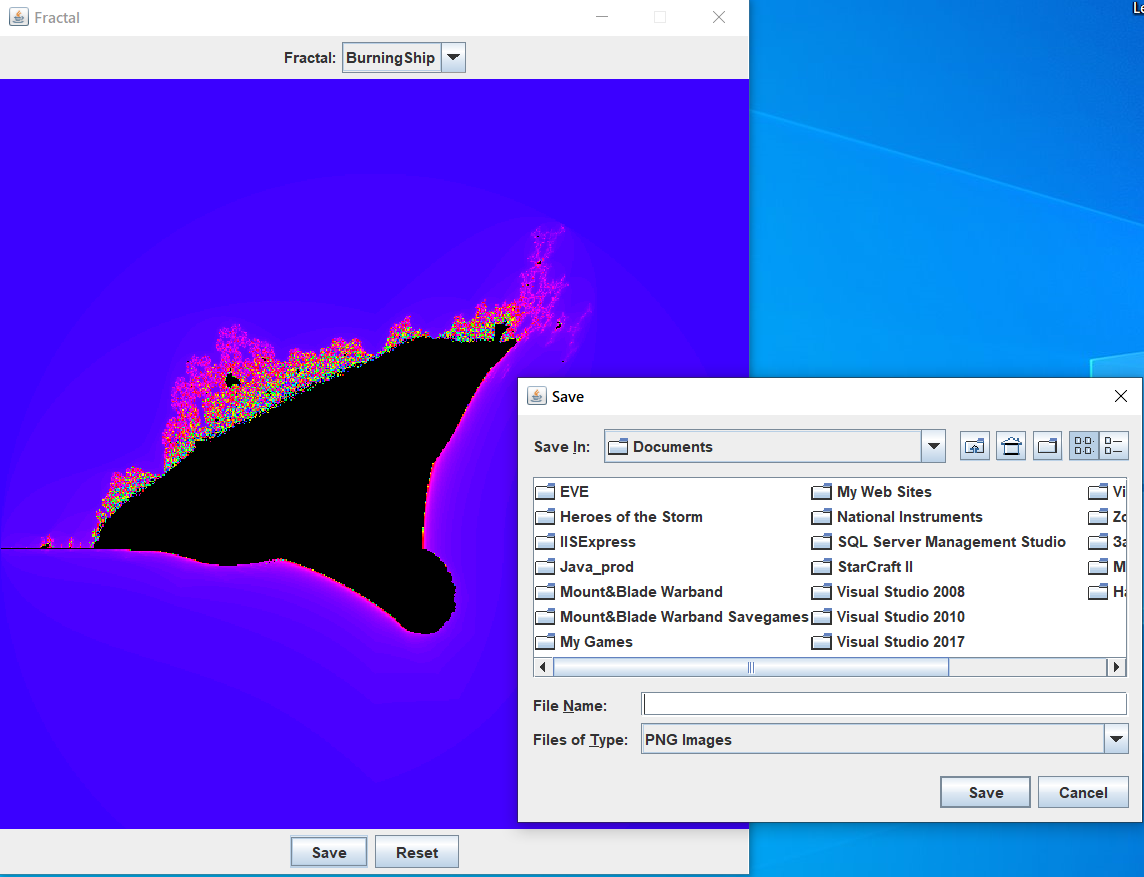


Рисунок 4 – Результат нажатия на кнопку «Save»

На рисунках 5-6 представлен результат нажатия на кнопку «Reset» и ЛКМ на рисунок.

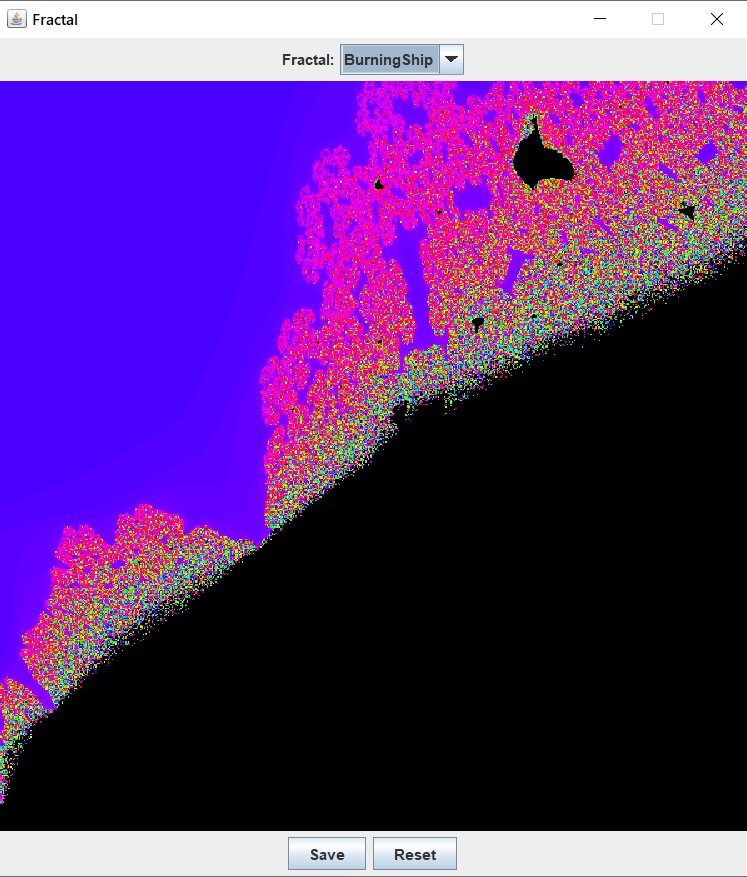


Рисунок 5 – Нажатие ЛКМ на рисунок

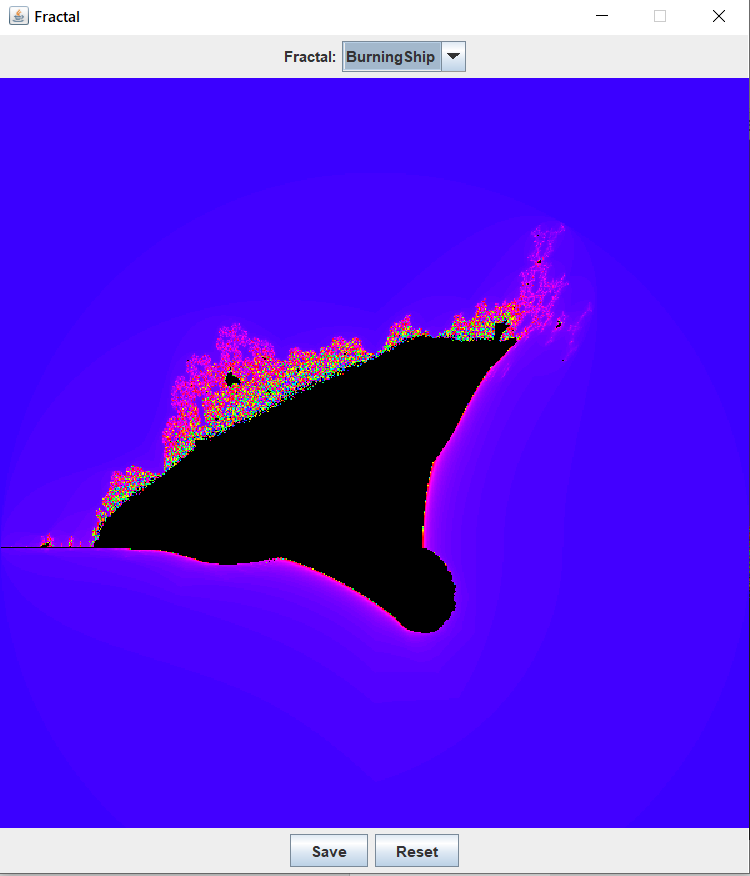


Рисунок 5 – Результат нажатия кнопки «Reset»

В 6 лабораторной работе многопоточность реализуется следующим образом:

1. Создается класс private class FractalWorker extends SwingWorker<Object, Object>.

2. При нажатии на кнопку Reset программа при помощи метода enableUI блокирует пользователю доступ к интерфейсу программы на время отрисовки фрактала, которая происходит в отдельном потоке, реализованном во вложенном классе FractalWorker, который после выполнения вызывает метод enableUI, который вновь открывает доступ пользователя к пользовательскому интерфейсу программы.

3. Также приложение реагирует на щелчки мышью, только в том случае, если больше нет строк, которые должны быть нарисованы

Вывод

В ходе данных лабораторных работ было реализовано рисование фракталов (Mandelbrot, Tricorn, BurningShip), так же добавлены некоторые элементы, которые позволяют пользователю взаимодействовать с программой, реализована многопоточность, которая ускоряет работу программы по рисованию фракталов.